Міністерство освіти і науки України
Луцький національний технічний університет

DATAMINING

Методичні вказівки до самостійної роботи для здобувачів
першого (бакалаврського) рівня вищої освіти
освітньої програми «Штучний інтелект та аналіз масивів даних»
спеціальності «Прикладна математика»
денної форми навчання

Луцьк 2025

УДК 004.8(075.8)

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ
Директор бібліотеки_____ Н. Поліщук

Рекомендовано до видання вченою радою факультету архітектури, будівництва та дизайну ЛНТУ,
протокол № ___ від «__» _____ 2025 року.

Голова вченої ради факультету ФАБД _____ О. Андрійчук

Розглянуто і схвалено на засіданні кафедри прикладної математики та механіки ЛНТУ,
протокол №___ від «___» _____ 2025 року.

Завідувачка кафедри прикладної математики та механіки _____ О. Мікуліч

Укладач: _____К. Вавринюк, асистент кафедри прикладної математики та механіки ЛНТУ

Рецензент: _____ кандидат технічних наук, доцент О. Приходько

Відповідальний за випуск: _____ О. Мікуліч, доктор технічних наук, професор, завідувачка кафедри прикладної математики та механіки ЛНТУ

DataMining методичні вказівки до самостійної роботи для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Штучний інтелект та аналіз масивів даних» спеціальності «Прикладна математика» денної форми навчання/ уклад. К. Вавринюк Луцьк: ЛНТУ, 2025. – 63 с.

Методичні вказівки до самостійної роботи для дисципліни «DataMining». Призначений для студентів освітньої програми «Штучний інтелект та аналіз масивів даних» спеціальності «Прикладна математика» денної форми навчання.

ЗМІСТ

ВСТУП

Самостійна робота з дисципліни «Основи DataMining» є невід'ємною складовою освітнього процесу і спрямована на:

- поглиблення теоретичних знань, отриманих на лекційних заняттях, через самостійне вивчення додаткових матеріалів, наукових публікацій та технічної документації;
- розвиток практичних навичок роботи з інструментами аналізу даних (Python, бібліотеки Pandas, Scikit-learn, Matplotlib, Seaborn) шляхом виконання додаткових завдань;
- формування дослідницьких компетентностей через самостійне формулювання гіпотез, проведення експериментів та аналіз результатів на реальних датасетах;
- розвиток навичок самоорганізації та планування при виконанні комплексних завдань з аналізу даних, що включають весь цикл OSEMN (Obtain, Scrub, Explore, Model, iNterpret);

У процесі виконання самостійної роботи здобувачі вищої освіти мають:

- закріпити теоретичний матеріал з усіх тем курсу через самостійне опрацювання лекційних конспектів та рекомендованої літератури;
- розширити практичні навички роботи з екосистемою python для data science, освоївши додаткові можливості бібліотек pandas, numpy, matplotlib, seaborn;
- навчитися самостійно шукати та опрацьовувати датасети з відкритих джерел;
- оволодіти методиками дослідницького аналізу даних, очищення та підготовки даних, інженерії ознак;
- освоїти алгоритми навчання з учителем та без учителя;

– навчитися критично оцінювати якість побудованих моделей та обирати оптимальні метрики залежно від бізнес-контексту;
– розвинути навички презентації результатів та формулювання бізнес-висновків на основі проведеного аналізу;
– усвідомити етичні аспекти роботи з даними та алгоритмами, навчитися виявляти та мінімізувати упередженість.

Загальний обсяг самостійної роботи становить 90 годин.

Самостійна робота безпосередньо сприяє формуванню наступних компетентностей:

Загальні компетентності

ЗК 3 – Здатність генерувати нові ідеї (креативність) – через розробку власних підходів до аналізу даних та створення нових ознак;

ЗК 5 – Здатність проведення дослідження на відповідному рівні – через самостійне формулювання гіпотез та їх перевірку;

ЗК 10 – Навички використання інформаційних комунікаційних технологій – через практичну роботу з Python та хмарними платформами (Google Colab, Kaggle).

Фахові компетентності

ФК 2 – Здатність виконувати завдання, сформульовані у математичній формі – через реалізацію алгоритмів Data Mining;

ФК 14 – Здатність сформулювати математичну постановку задачі та обрати метод її розв'язання – через аналіз реальних бізнес-кейсів;

ФК 17 – Здатність до ефективного використання методів штучного інтелекту – через практичне застосування алгоритмів машинного навчання;

ФК 18 – Здатність до вибору адекватних моделей машинного навчання – через порівняльний аналіз різних алгоритмів.

Програмні результати навчання

ПРН03 – Формалізація задач та вибір раціонального методу вирішення;

ПРН07 – Проведення практичних досліджень;

ПРН19 – Збір та інтерпретація даних з урахуванням етичних аспектів;

ПРН21 – Побудова математичних моделей штучного інтелекту;

ПРН22 – Застосування методів глибинного аналізу даних.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання самостійної роботи необхідно встановити наступне ПЗ:

1. Python (версія 3.8 або вища)

   Офіційний сайт: https://www.python.org/downloads/

   Рекомендується встановлення через Anaconda Distribution

2. Anaconda Distribution

   Завантаження: https://www.anaconda.com/products/distribution

   Містить Jupyter Notebook, Spyder та інші інструменти

3. Основні бібліотеки Python:

   Встановлення через pip:

```
pip install numpy pandas matplotlib seaborn scikit-learn
pip install jupyter notebook
```

   Або через conda:

```
conda install numpy pandas matplotlib seaborn scikit-learn
conda install jupyter notebook
```

4. Додаткові бібліотеки:

`pip install statsmodels`	для аналізу часових рядів
`pip install mlxtend`	для асоціативних правил
`pip install plotly`	для інтерактивних візуалізацій
`pip install xgboost lightgbm`	для gradient boosting

5. Середовища розробки: Jupyter Notebook, JupyterLab, VS Code з розширенням Python, PyCharm Community Edition, Google Colab

Інструкції з встановлення середовища

Варіант 1: Встановлення через Anaconda.

1. Завантажте Anaconda з офіційного сайту
2. Запустіть інсталятор та дотримуйтесь інструкцій
3. Після встановлення відкрийте Anaconda Navigator
4. Запустіть Jupyter Notebook

5. Перевірте встановлення, виконавши у новому ноутбуці:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
print("Усі бібліотеки встановлені успішно!")
```

Варіант 2: Встановлення через pip

1. Встановіть Python з офіційного сайту
2. Відкрийте командний рядок (Terminal/CMD)
3. Оновіть pip: `python -m pip install --upgrade pip`
4. Встановіть бібліотеки:

```
pip install numpy pandas matplotlib seaborn scikit-learn jupyter
```

5. Запустіть Jupyter: jupyter notebook

Варіант 3: Використання Google Colab

1. Перейдіть на https://colab.research.google.com/
2. Увійдіть через Google акаунт
3. Створіть новий ноутбук
4. Усі необхідні бібліотеки вже встановлені
5. Можна підключити Google Drive для збереження файлів

Корисні онлайн-ресурси

Офіційна документація:

Python: https://docs.python.org/3/

NumPy: https://numpy.org/doc/

Pandas: https://pandas.pydata.org/docs/

Matplotlib: https://matplotlib.org/stable/contents.html

Seaborn: https://seaborn.pydata.org/

Scikit-learn: https://scikit-learn.org/stable/

Джерела датасетів:

Kaggle Datasets: https://www.kaggle.com/datasets

UCI ML Repository: https://archive.ics.uci.edu/ml/

Google Dataset Search: https://datasetsearch.research.google.com/

Awesome Public Datasets: https://github.com/awesomedata/awesome-public-datasets

ЗМІСТОВИЙ МОДУЛЬ 1: ОСНОВИ DATA SCIENCE ТА ДОСЛІДНИЦЬКИЙ АНАЛІЗ ДАНИХ

Тема 1. Основи та інструментарій

Мета: поглибити знання про екосистему Python для Data Science, освоїти розширені можливості бібліотек Pandas, NumPy, Matplotlib та Seaborn для ефективної роботи з даними.

1. Вивчення документації основних бібліотек.

NumPy (https://numpy.org/doc/stable/): основи роботи з масивами, broadcasting та векторизовані операції, універсальні функції, лінійна алгебра.

Pandas (https://pandas.pydata.org/docs/): структури даних (Series, DataFrame), індексація та вибірка даних, методи агрегації та трансформації, робота з категоріальними даними.

Matplotlib (https://matplotlib.org/stable/): архітектура бібліотеки (Figure, Axes), налаштування стилів та тем, підграфіки (subplots).

Seaborn (https://seaborn.pydata.org/): статистичні візуалізації, параметр hue для груп, теми та палітри кольорів.

2. Розширені операції з Pandas.

Приклад коду для практики

```
import pandas as pd
import numpy as np
# Створіть датафрейм з продажами
sales_data = {
    'date': pd.date_range('2024-01-01', periods=100),
    'product': np.random.choice(['A', 'B', 'C'], 100),
    'region': np.random.choice(['North', 'South', 'East', 'West'],
100),
    'sales': np.random.randint(100, 1000, 100),
    'quantity': np.random.randint(1, 20, 100)
}
df = pd.DataFrame(sales_data)
```

Завдання

Складні операції з DataFrame

1. Створіть pivot table з продажами по продуктах та регіонах
2. Використайте melt для перетворення wide format на long format
3. Об'єднайте два датафрейми за допомогою merge (inner, outer, left, right)
4. Використайте concat для вертикального та горизонтального об'єднання

Завдання на групування та агрегація даних:

1. Згрупуйте дані за продуктом та регіоном, порахуйте різні статистики
2. Використайте agg() з кількома функціями
3. Застосуйте transform() для додавання агрегованих значень
4. Використайте apply() з власною функцією
5. Створіть MultiIndex та виконайте операції з ним

Робота з часовими рядами. Завдання:

1. Встановіть date як індекс та використайте DatetimeIndex
2. Виконайте resample для агрегації по тижнях/місяцях
3. Розрахуйте rolling mean та rolling std
4. Виявіть пропуски в часовому ряді та заповніть їх
5. Створіть lag features для прогнозування

Створення власного набору візуалізацій. Побудуйте комплексний набір візуалізацій для аналізу даних:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Гістограма з KDE для розподілу продажів
2. Boxplot для порівняння продажів по регіонах
3. Scatter plot з hue для залежності між ознаками
4. Heatmap кореляційної матриці
5. Violin plot для розподілу по групах

6. Pair plot для всіх числових змінних
7. Bar plot з помилками (error bars)
8. Line plot з тренд-лінією

Контрольні питання

1. Які переваги має Pandas DataFrame над списками та словниками?
2. Коли vectorized операції ефективніші за цикли?
3. Що таке broadcasting в NumPy і як його використовувати?
4. У чому різниця між merge() та join()?
5. Коли використовувати concat замість merge?
6. Що таке ключі злиття (keys) та як вони працюють?
7. Які види об'єднань існують (inner, outer, left, right)?
8. Який графік використовувати для показу розподілу?
9. Як візуалізувати зв'язок між двома числовими змінними?
10. Яку візуалізацію обрати для категоріальних даних?
11. Як показати зміну в часі?

Тема 2. Глибоке занурення в очищення даних

Мета: освоїти розширені стратегії роботи з пропущеними значеннями та викидами, навчитися застосовувати просунуті методи імпутації.

1. Дослідження типів пропусків

Опрацюйте матеріал про три типи механізмів пропусків.

Практична частина

```
import pandas as pd
import numpy as np
import missingno as msno  # pip install missingno
```

Завдання 1: Симуляція різних типів пропусків

```
def create_mcar_data(df, column, missing_rate=0.2):
    """Створити MCAR пропуски"""
    # Ваш код тут
    pass

def create_mar_data(df, column, dependent_column, threshold):
    """Створити MAR пропуски залежно від іншої змінної"""
    # Ваш код тут
    pass

def create_mnar_data(df, column, quantile_threshold):
    """Створити MNAR пропуски (наприклад, відсутні високі значення)"""
    # Ваш код тут
    pass
```

Завдання 2: Візуалізація патернів пропусків (використайте missingno для створення): matrix plot, bar chart, heatmap кореляцій пропусків, dendrogram.

2. Просунуті методи імпутації

KNN Imputation: `from sklearn.impute import KNNImputer`

1. Реалізуйте KNN імпутацію з різною кількістю сусідів (k=3, 5, 10).
2. Порівняйте результати з простим заповненням середнім.
3. Візуалізуйте розподіл до і після імпутації.
4. Оцініть вплив на кореляційну структуру даних.

Приклад коду:

```
imputer = KNNImputer(n_neighbors=5)
df_imputed = imputer.fit_transform(df_numeric)
```

Iterative Imputer (MICE)

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

1. Застосуйте IterativeImputer з estimators BayesianRidge та RandomForestRegressor
2. Порівняйте кількість ітерацій до конвергенції
3. Оцініть якість імпутації на синтетичних даних з відомими значеннями

Multiple Imputation

1. Створіть кілька імпутованих датасетів (m=5).
2. Побудуйте модель на кожному датасеті.
3. Об'єднайте результати.
4. Оцініть невизначеність, пов'язану з пропусками.

Контрольні питання

1. Чим відрізняються MCAR, MAR та MNAR? Наведіть приклади.
2. Чому простого видалення рядків з пропусками часто недостатньо?
3. Як KNN Imputation працює? Які його переваги та недоліки?
4. Що таке MICE і чому він ефективний?
5. Як оцінити якість імпутації?
6. Які методи можна використати для виявлення типу пропусків?

Тема 3. Інженерія ознак (Feature Engineering)

Мета: опанувати просунуті техніки створення нових ознак, навчитися трансформувати дані для покращення якості моделей.

1. Створення поліноміальних ознак

```
from sklearn.preprocessing import PolynomialFeatures
```

1. Створіть поліноміальні ознаки другого та третього ступеня.
2. Проаналізуйте збільшення кількості ознак.
3. Побудуйте модель та порівняйте з лінійною.
4. Виявіть проблему multicollinearity.

```
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)
```

2. Взаємодія між ознаками

1. Створіть ознаки взаємодії вручну (multiply, divide, add, subtract)
2. Використайте domain knowledge для створення змістовних комбінацій
3. Оцініть важливість створених ознак

Приклади:

```
df['price_per_sqm'] = df['price'] / df['area']
df['total_rooms'] = df['bedrooms'] + df['bathrooms']
df['luxury_score'] = df['price'] * df['quality_rating']
```

3. Target Encoding з регуляризацією

Завдання: реалізуйте Target Encoding з smoothing

```
def target_encode_smooth(train, test, column, target, alpha=5):
    """
    Target Encoding з регуляризацією для уникнення overfitting

    Parameters:
    -----------
    train : DataFrame
    test : DataFrame
    column : str - назва категоріальної колонки
    target : str - назва цільової змінної
    alpha : float - параметр згладжування

    Returns:
    --------
```

```
        train_encoded, test_encoded : DataFrame
        """
```

1. Розрахувати глобальне середнє: `global_mean = train[target].mean()`

2. Розрахувати статистику для кожної категорії: `category_stats = ...`

3. Застосувати smoothing: `smoothed_mean = (category_mean * n + global_mean * alpha) / (n + alpha)`

```
    return train_encoded, test_encoded
```

4. Бінінг та дискретизація: порівняйте різні стратегії та їх вплив на модель

```
    from sklearn.preprocessing import KBinsDiscretizer
     # 1. Equal-width binning
    age_bins = pd.cut(df['age'], bins=5)

    # 2. Equal-frequency binning (quantile-based)
    age_quantiles = pd.qcut(df['age'], q=4)

    # 3. Custom bins based on domain knowledge
    age_custom = pd.cut(df['age'],
                        bins=[0, 18, 35, 50, 65, 100],
                        labels=['youth',    'young_adult',    'middle_age',
'senior', 'elderly'])

    # 4. Automatic binning з KBinsDiscretizer
    discretizer     =     KBinsDiscretizer(n_bins=5,     encode='ordinal',
strategy='quantile')
```

5. Трансформації для нормалізації розподілу: застосуйте різні трансформації. Візуалізуйте розподіли до та після трансформації, створіть автоматизований pipeline

```
    import scipy.stats as stats

    # 1. Log transformation
    df['log_price'] = np.log1p(df['price'])
    # 2. Square root
    df['sqrt_area'] = np.sqrt(df['area'])
    # 3. Box-Cox transformation
    df['boxcox_income'], lambda_param = stats.boxcox(df['income'] + 1)
    # 4. Yeo-Johnson (works with negative values)
    from sklearn.preprocessing import PowerTransformer
    pt = PowerTransformer(method='yeo-johnson')
    df['yj_feature'] = pt.fit_transform(df[['feature']])

    from sklearn.pipeline import Pipeline
    from sklearn.compose import ColumnTransformer
    class FeatureEngineer:
        """
        Клас для автоматичної інженерії ознак
        """
```

```
            def __init__(self, numeric_features, categorical_features):
                self.numeric_features = numeric_features
                self.categorical_features = categorical_features
            def create_interactions(self, X):
                """Створення ознак взаємодії"""
                pass
            def create_aggregations(self, X, group_col):
                """Створення агрегованих ознак"""
                pass
            def create_polynomial(self, X, degree=2):
                """Створення поліноміальних ознак"""
                pass
            def fit_transform(self, X):
                """Застосування всіх трансформацій"""
                X_new = X.copy()
                # Застосувати всі методи
                return X_new
        # Використання:
        fe = FeatureEngineer(numeric_features=['age', 'income'],
                             categorical_features=['city', 'occupation'])
X_engineered = fe.fit_transform(X_train)
```

Контрольні питання

1. Що таке Feature Engineering і чому він важливий?
2. Коли використовувати поліноміальні ознаки? Які ризики?
3. Чим Target Encoding відрізняється від One-Hot Encoding?
4. Як уникнути data leakage при створенні ознак?
5. Що таке feature importance і як його оцінювати?
6. Які трансформації використовувати для skewed розподілів?

Тема 4. Проста лінійна регресія

Мета: поглибити розуміння простої лінійної регресії, навчитися інтерпретувати результати та діагностувати проблеми моделі.

Завдання для самостійного виконання

1. Розуміння математичної основи

Опрацюйте теоретичний матеріал:

- метод найменших квадратів (OLS)
- припущення лінійної регресії
- коефіцієнти та їх інтерпретація
- статистичні тести (t-test, F-test)

Завдання: побудуйте модель та проаналізуйте статистику

```
import statsmodels.api as sm
X = sm.add_constant(X_train)
model = sm.OLS(y_train, X).fit()
print(model.summary())
```

Проаналізуйте:

- R-squared та Adjusted R-squared
- P-values для коефіцієнтів
- F-statistic
- Residual analysis

2. Розширені техніки регресії

а) Ridge Regression (L2 регуляризація): побудуйте Ridge регресію з різними значеннями alpha. Побудуйте графік залежності score від alpha. Проаналізуйте, як регуляризація впливає на коефіцієнти.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
alphas = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
ridge_scores = []
for alpha in alphas:
```

```
        ridge = Ridge(alpha=alpha)
        scores = cross_val_score(ridge, X_train, y_train, cv=5,
scoring='r2')
        ridge_scores.append(scores.mean())
```

б) Lasso Regression (L1): побудуйте Lasso з різними alpha. Відстежте, які коефіцієнти стають нульовими. Використайте LassoCV для автоматичного підбору alpha. Порівняйте важливість ознак.

```
from sklearn.linear_model import Lasso, LassoCV
lasso = Lasso(alpha=1.0)
lasso.fit(X_train, y_train)
```

в) ElasticNet: застосуйте ElasticNet (комбінація L1 та L2). Підберіть l1_ratio (співвідношення між L1 та L2). Порівняйте з Ridge та Lasso.

```
from sklearn.linear_model import ElasticNet
```

2. Grid Search для гіперпараметрів*:* оптимізація гіперпараметрів. Візуалізуйте grid search результати. Знайдіть найкращі параметри. Оцініть на тестовій вибірці

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1.0, 10.0],
    'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
}

grid_search = GridSearchCV(
    ElasticNet(),
    param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
```

3. Аналіз коефіцієнтів та інтерпретація*:* інтерпретація моделей. Порівняйте коефіцієнти різних моделей. Візуалізуйте коефіцієнти. Які ознаки найважливіші? Як регуляризація вплинула на коефіцієнти? Які ознаки видалив Lasso?

```
coef_df = pd.DataFrame({
    'feature': feature_names,
    'linear': linear_reg.coef_,
    'ridge': ridge.coef_,
    'lasso': lasso.coef_,
    'elastic': elastic.coef_
})

plt.figure(figsize=(12, 6))
coef_df.set_index('feature').plot(kind='bar')
plt.title('Порівняння коефіцієнтів різних моделей')
plt.xticks(rotation=45)
plt.legend()
```

4. Діагностика регресії: перевірка припущень лінійної регресії

а) Аналіз залишків (residuals)

```
residuals = y_test - y_pred
```

1) Графік залишків vs прогнози

```
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
```

2) Q-Q plot для перевірки нормальності

```
import scipy.stats as stats
stats.probplot(residuals, dist="norm", plot=plt)
```

3) Тест на гомоскедастичність (постійність дисперсії)

```
from scipy.stats import levene
```

б) Виявлення мультиколінеарності

```
from statsmodels.stats.outliers_influence import
variance_inflation_factor

# Розрахунок VIF для кожної ознаки
vif_data = pd.DataFrame()
vif_data["feature"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i)
                   for i in range(len(X_train.columns))]
# Інтерпретація: VIF > 10 вказує на проблему
```

в) Виявлення впливових точок

```
from statsmodels.stats.outliers_influence import OLSInfluence
# Cook's distance
# Leverage
# Studentized residuals
```

Практичний кейс: прогнозування цін на нерухомість

Завдання: повний цикл аналізу

1. Завантажте датасет (House Prices або власний)
2. EDA та підготовка даних
3. Feature engineering
4. Побудуйте 4 моделі: Linear, Ridge, Lasso, ElasticNet
5. Порівняйте результати
6. Виконайте діагностику найкращої моделі
7. Інтерпретуйте результати та дайте рекомендації

Контрольні питання

1. У чому різниця між L1 та L2 регуляризацією?
2. Чому Lasso може зменшувати коефіцієнти до нуля?
3. Коли використовувати Ridge, а коли Lasso?
4. Що таке мультиколінеарність і чому вона проблема?
5. Як інтерпретувати VIF?
6. Що показує графік залишків?

Тема 5. Множинна лінійна регресія та оцінка моделі

Мета: практично освоїти повний цикл побудови та оцінки регресійних моделей, включаючи валідацію, порівняння метрик та інтерпретацію результатів.

Завдання для самостійного виконання

1. Розширення набору ознак

Завдання: створіть комплексну модель.

1. Числові ознаки з трансформаціями
2. Категоріальні ознаки (OHE або Target Encoding)
3. Ознаки взаємодії
4. Поліноміальні ознаки (обережно з розмірністю)

```
# Приклад pipeline:
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer',                    SimpleImputer(strategy='constant',
fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', Ridge())
])
```

2. Cross-Validation та Learning Curves: K-Fold Cross-Validation.Різні стратегії CV: KFold, StratifiedKFold (для регресії з binning), TimeSeriesSplit (якщо є часова залежність). Learning Curves. Чи є overfitting? Чи допоможе більше даних?

```
from sklearn.model_selection import cross_val_score, learning_curve
cv_scores = cross_val_score(model, X_train, y_train,
                            cv=5,
                            scoring='neg_mean_squared_error')
```

```
    print(f"CV      RMSE:      {np.sqrt(-cv_scores.mean()):.3f}      (+/-
{np.sqrt(cv_scores.std()):.3f})")
    train_sizes, train_scores, val_scores = learning_curve(
        model, X_train, y_train,
        train_sizes=np.linspace(0.1, 1.0, 10),
        cv=5,
        scoring='neg_mean_squared_error',
        n_jobs=-1
    )

    # Візуалізація
    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, -train_scores.mean(axis=1), label='Training')
    plt.plot(train_sizes, -val_scores.mean(axis=1), label='Validation')
    plt.xlabel('Training Set Size')
    plt.ylabel('MSE')
    plt.legend()
    plt.title('Learning Curves')
```

3. Аналіз помилок*:* детальний аналіз помилок. Розподіл помилок. Визначення найбільших помилок. Виявлення патернів у помилках. Чи є систематична упередженість? У яких випадках модель помиляється найбільше?

```
    errors = y_test - y_pred
    plt.hist(errors, bins=50)
    plt.xlabel('Error')
    plt.ylabel('Frequency')
    plt.title('Distribution of Prediction Errors')

    # 2. Найбільші помилки
    error_analysis = pd.DataFrame({
        'actual': y_test,
        'predicted': y_pred,
        'error': errors,
        'abs_error': np.abs(errors),
        'pct_error': np.abs(errors) / y_test * 100
    })

    # Топ-10 найбільших помилок
    worst_predictions = error_analysis.nlargest(10, 'abs_error')

    # 3. Аналіз помилок по діапазонах цін
    error_analysis['price_range'] = pd.cut(y_test, bins=5)
    error_by_range                                                    =
error_analysis.groupby('price_range')['abs_error'].mean()
```

4. Feature Importance та Selection*:* визначення важливості ознак

Метод 1: Коефіцієнти для лінійних моделей

```
    feature_importance = pd.DataFrame({
        'feature': feature_names,
        'importance': np.abs(model.coef_)
    }).sort_values('importance', ascending=False)
```

Метод 2: Permutation Importance

```
    from sklearn.inspection import permutation_importance
    perm_importance = permutation_importance(
        model, X_test, y_test,
        n_repeats=10,
        random_state=42
    )
    # Візуалізація
    plt.barh(feature_names, perm_importance.importances_mean)
    plt.xlabel('Permutation Importance')
    plt.title('Feature Importance')
```

Метод 3: Recursive Feature Elimination

```
    from sklearn.feature_selection import RFE
    rfe = RFE(estimator=Ridge(), n_features_to_select=10)
    rfe.fit(X_train, y_train)
selected_features = [f for f, s in zip(feature_names, rfe.support_) if s]
```

Практичне завдання: створіть повний ML pipeline

```
from sklearn.feature_selection import SelectKBest, f_regression
full_pipeline = Pipeline([
    ('preprocessing', preprocessor),
    ('feature_selection', SelectKBest(f_regression, k=20)),
    ('regressor', Ridge(alpha=1.0))
])
# Grid search по всьому pipeline
param_grid = {
    'feature_selection__k': [10, 15, 20, 25],
    'regressor__alpha': [0.1, 1.0, 10.0]
}
grid_search = GridSearchCV(
    full_pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    verbose=2
)
grid_search.fit(X_train, y_train)
```

Контрольні питання

1. Чому важливо використовувати cross-validation?
2. Як інтерпретувати learning curves?
3. Яка різниця між MAE та RMSE?
4. Що означає негативне значення R^2?
5. Як виявити overfitting за допомогою CV?
6. Що таке permutation importance?

Тема 6. Поняття класифікації

Мета: освоїти основи бінарної класифікації, зрозуміти роботу логістичної регресії та навчитися обирати правильний поріг класифікації.

Завдання для самостійного виконання

1. Реалізація логістичної регресії. Базова модель та аналіз коефіцієнтів

```
from sklearn.linear_model import LogisticRegression

# Завдання 1: Базова модель
logreg = LogisticRegression(random_state=42)
logreg.fit(X_train, y_train)

# Отримання ймовірностей
y_proba = logreg.predict_proba(X_test)[:, 1]

# Завдання 2: Аналіз коефіцієнтів
coef_df = pd.DataFrame({
    'feature': feature_names,
    'coefficient': logreg.coef_[0],
    'odds_ratio': np.exp(logreg.coef_[0])
})
```

Пояснення:

Odds Ratio > 1: збільшення ознаки підвищує ймовірність класу 1

Odds Ratio < 1: зменшує ймовірність

3. Вибір порогу класифікації*:* експериментування з порогом

```
thresholds = np.linspace(0, 1, 101)
metrics_by_threshold = []

for threshold in thresholds:
    y_pred_custom = (y_proba >= threshold).astype(int)

    precision = precision_score(y_test, y_pred_custom)
    recall = recall_score(y_test, y_pred_custom)
    f1 = f1_score(y_test, y_pred_custom)

    metrics_by_threshold.append({
        'threshold': threshold,
        'precision': precision,
        'recall': recall,
        'f1': f1
    })

metrics_df = pd.DataFrame(metrics_by_threshold)

# Візуалізація
plt.figure(figsize=(12, 6))
```

```
    plt.plot(metrics_df['threshold'],             metrics_df['precision'],
label='Precision')
    plt.plot(metrics_df['threshold'], metrics_df['recall'], label='Recall')
    plt.plot(metrics_df['threshold'], metrics_df['f1'], label='F1-Score')
    plt.xlabel('Threshold')
    plt.ylabel('Score')
    plt.legend()
    plt.title('Metrics vs Classification Threshold')
    plt.grid(True)
```

Питання для аналізу:

1. При якому порозі F1-score максимальний?
2. Як змінюється trade-off між Precision та Recall?
3. Який поріг обрати для вашої бізнес-задачі?

4. Калібрація ймовірностей

Перевірка калібрації моделі

```
    from       sklearn.calibration       import       calibration_curve,
CalibratedClassifierCV
    # 1. Побудова calibration curve
    prob_true, prob_pred = calibration_curve(y_test, y_proba, n_bins=10)

    plt.figure(figsize=(8, 6))
    plt.plot(prob_pred, prob_true, marker='o')
    plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
    plt.xlabel('Predicted Probability')
    plt.ylabel('True Probability')
    plt.title('Calibration Curve')
    plt.grid(True)
```

Пояснення: якщо крива близька до діагоналі - модель добре калібрована, якщо вище - модель надто оптимістична, якщо нижче - надто песимістична

Калібрація моделі

```
    calibrated_model      =      CalibratedClassifierCV(logreg,      cv=5,
method='sigmoid')
    calibrated_model.fit(X_train, y_train)
y_proba_calibrated = calibrated_model.predict_proba(X_test)[:, 1]
```

5. Порівняння з іншими класифікаторами: порівняйте базові класифікатори

```
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.naive_bayes import GaussianNB

    models = {
        'Logistic Regression': LogisticRegression(),
        'Decision Tree': DecisionTreeClassifier(max_depth=5),
        'KNN': KNeighborsClassifier(n_neighbors=5),
        'Naive Bayes': GaussianNB()
    }

    results = []

    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        results.append({
            'Model': name,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1-Score': f1
        })

    results_df = pd.DataFrame(results)
print(results_df)
```

Контрольні питання

1. Чому логістична регресія називається "регресією", хоча є класифікатором?
2. Що таке odds ratio і як його інтерпретувати?
3. Чому поріг 0.5 не завжди оптимальний?
4. Що таке калібрація моделі?
5. Коли логістична регресія працює добре, а коли ні?

Тема 7. Оцінка моделей класифікації

Мета: опанувати розширені метрики класифікації, навчитися працювати з незбалансованими даними та використовувати ROC-AUC для оцінки моделей.

Завдання для самостійного виконання

1. ROC-крива та AUC метрика. Побудуйте ROC-криву. Знайдіть оптимальний поріг за критерієм Youden's Index. Порівняйте ROC-криві кількох моделей

```
from sklearn.metrics import roc_curve, roc_auc_score, auc
import matplotlib.pyplot as plt

# Побудова ROC-кривої
y_proba = model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Візуалізація
plt.figure(figsize=(10, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',
         label='Random Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.grid(True, alpha=0.3)
plt.show()

# Знайдіть оптимальний поріг за критерієм Youden's Index
# Youden's Index = TPR - FPR
youdens_index = tpr - fpr
optimal_idx = np.argmax(youdens_index)
optimal_threshold = thresholds[optimal_idx]

print(f"Оптимальний поріг: {optimal_threshold:.3f}")
print(f"TPR: {tpr[optimal_idx]:.3f}, FPR: {fpr[optimal_idx]:.3f}")

# Порівняння ROC-кривих кількох моделей
models_roc = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(n_estimators=100),
    'Gradient Boosting': GradientBoostingClassifier()
}

plt.figure(figsize=(10, 8))

for name, model in models_roc.items():
```

```
    model.fit(X_train, y_train)
    y_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves Comparison')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

2. Precision-Recall крива: побудуйте PR-криву для незбалансованих даних

```
from sklearn.metrics import precision_recall_curve, average_precision_score

#PR-крива для незбалансованих даних

precision, recall, thresholds_pr = precision_recall_curve(y_test, y_proba)
avg_precision = average_precision_score(y_test, y_proba)

plt.figure(figsize=(10, 8))
plt.plot(recall, precision, color='blue', lw=2,
         label=f'PR curve (AP = {avg_precision:.2f})')
plt.axhline(y=y_test.mean(), color='red', linestyle='--',
            label=f'Baseline (Positive Rate = {y_test.mean():.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Пояснення: на відміну від ROC, PR-крива більш інформативна для незбалансованих класів,оскільки фокусується на позитивному класі

3. Робота з незбалансованими класами

а) Техніки Resampling: перевірка балансу класів. Oversampling з SMOTE. Undersampling. Комбінований підхід.

```
from imblearn.over_sampling import SMOTE, ADASYN, RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler, TomekLinks
from imblearn.combine import SMOTETomek
# Завдання 1: Перевірка балансу класів
print(f"Розподіл класів:\n{y_train.value_counts()}")
print(f"Співвідношення:            {y_train.value_counts()[1]            /
y_train.value_counts()[0]:.3f}")
# Завдання 2: Oversampling з SMOTE
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
print(f"\nПісля SMOTE:\n{pd.Series(y_train_smote).value_counts()}")
# Побудова моделі на збалансованих даних
```

```
model_smote = LogisticRegression()
model_smote.fit(X_train_smote, y_train_smote)
y_pred_smote = model_smote.predict(X_test)
# Порівняння результатів
print("\nОригінальні дані:")
print(classification_report(y_test, y_pred_original))
print("\nПісля SMOTE:")
print(classification_report(y_test, y_pred_smote))
# Завдання 3: Undersampling
undersampler = RandomUnderSampler(random_state=42)
X_train_under, y_train_under = undersampler.fit_resample(X_train, y_train)
# Завдання 4: Комбінований підхід
smote_tomek = SMOTETomek(random_state=42)
X_train_combined,   y_train_combined   =   smote_tomek.fit_resample(X_train,
y_train)
# Завдання 5: Порівняння всіх підходів
strategies = {
    'Original': (X_train, y_train),
    'SMOTE': (X_train_smote, y_train_smote),
    'Undersampling': (X_train_under, y_train_under),
    'SMOTE+Tomek': (X_train_combined, y_train_combined)
}
results_resampling = []
for name, (X, y) in strategies.items():
    model = LogisticRegression()
    model.fit(X, y)
    y_pred = model.predict(X_test)
       results_resampling.append({
        'Strategy': name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1': f1_score(y_test, y_pred),
        'AUC': roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
    })
results_df = pd.DataFrame(results_resampling)
print(results_df)
```

б) Class Weights: використання вбудованих механізмів балансування

```
# Автоматичне обчислення ваг
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight(
    'balanced',
    classes=np.unique(y_train),
    y=y_train
)

class_weight_dict = {i: w for i, w in enumerate(class_weights)}
print(f"Class weights: {class_weight_dict}")

# Модель з вагами
model_weighted = LogisticRegression(class_weight='balanced')
model_weighted.fit(X_train, y_train)

# Або manually:
model_manual = LogisticRegression(class_weight=class_weight_dict)
model_manual.fit(X_train, y_train)
```

```
# Порівняння
y_pred_weighted = model_weighted.predict(X_test)
print("\n3 class_weight='balanced':")
print(classification_report(y_test, y_pred_weighted))
```

в) Threshold Moving: корекція порогу для незбалансованих класів

```
# Знайдіть оптимальний поріг за F1-score
y_proba = model.predict_proba(X_test)[:, 1]
thresholds = np.linspace(0, 1, 100)

f1_scores = []
for threshold in thresholds:
    y_pred_threshold = (y_proba >= threshold).astype(int)
    f1 = f1_score(y_test, y_pred_threshold)
    f1_scores.append(f1)

optimal_threshold = thresholds[np.argmax(f1_scores)]
print(f"Оптимальний поріг для максимального F1: {optimal_threshold:.3f}")

# Застосування оптимального порогу
y_pred_optimal = (y_proba >= optimal_threshold).astype(int)
print("\n3 оптимальним порогом:")
print(classification_report(y_test, y_pred_optimal))
```

4. Додаткові метрики: розрахунок розширених метрик

```
from sklearn.metrics import (
    matthews_corrcoef,
    cohen_kappa_score,
    balanced_accuracy_score,
    log_loss,
    brier_score_loss
)

def comprehensive_evaluation(y_true, y_pred, y_proba):
    """
    Комплексна оцінка класифікатора
    """
    metrics = {
        'Accuracy': accuracy_score(y_true, y_pred),
        'Balanced Accuracy': balanced_accuracy_score(y_true, y_pred),
        'Precision': precision_score(y_true, y_pred),
        'Recall': recall_score(y_true, y_pred),
        'F1-Score': f1_score(y_true, y_pred),
        'MCC': matthews_corrcoef(y_true, y_pred),
        'Cohen\'s Kappa': cohen_kappa_score(y_true, y_pred),
        'AUC-ROC': roc_auc_score(y_true, y_proba),
        'Log Loss': log_loss(y_true, y_proba),
        'Brier Score': brier_score_loss(y_true, y_proba)
    }
    return metrics

# Застосування
metrics_result = comprehensive_evaluation(y_test, y_pred, y_proba)
for metric, value in metrics_result.items():
    print(f"{metric}: {value:.4f}")

# Пояснення метрик:
```

```
# - MCC (Matthews Correlation Coefficient): враховує всі 4 комірки confusion
matrix
# - Cohen's Kappa: враховує випадкове співпадіння
# - Balanced Accuracy: середнє від Recall по класах
# - Log Loss: оцінює впевненість прогнозів
# - Brier Score: MSE для ймовірностей
```

Практичний кейс: виявлення шахрайських транзакцій

Завдання: повний цикл роботи з незбалансованими даними

1. Завантажте датасет (Credit Card Fraud Detection з Kaggle)
2. Проведіть EDA, зверніть увагу на дисбаланс класів
3. Побудуйте baseline модель
4. Застосуйте різні техніки роботи з дисбалансом:
   - SMOTE
   - Class weights
   - Threshold tuning
5. Порівняйте результати за допомогою:
   - ROC-AUC
   - PR-AUC
   - F1-score
   - MCC
6. Оберіть найкращу стратегію
7. Проведіть аналіз помилок (які транзакції модель пропускає?)
8. Сформулюйте рекомендації для бізнесу

Контрольні питання

1. У чому різниця між ROC-AUC та PR-AUC?
2. Коли використовувати SMOTE, а коли class weights?
3. Що таке Matthews Correlation Coefficient?
4. Чому Accuracy може бути оманливою метрикою?
5. Як інтерпретувати ROC-криву?
6. Що краще для незбалансованих класів: precision чи recall?

ЗМІСТОВИЙ МОДУЛЬ 2: РОЗШИРЕНІ ТЕХНОЛОГІЇ

Тема 8. Дерева рішень

Мета: глибоко зрозуміти механізм роботи дерев рішень, навчитися контролювати їх складність та інтерпретувати результати.

Завдання для самостійного виконання

1. Розуміння механізмів розбиття

Завдання 1: реалізуйте розрахунок ентропії та Gini impurity

```
import numpy as np

def entropy(y):
    """
    Розрахунок ентропії
    H(S) = -Σ p_i * log2(p_i)
    """
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    return -np.sum([p * np.log2(p) for p in probabilities if p > 0])

def gini_impurity(y):
    """
    Розрахунок Gini impurity
    Gini(S) = 1 - Σ p_i²
    """
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    return 1 - np.sum([p**2 for p in probabilities])
```

Завдання 2: розрахунок Information Gain

```
def information_gain(parent, left_child, right_child, criterion='entropy'):
    """
    IG = H(parent) - (n_left/n_total * H(left) + n_right/n_total * H(right))
    """
    if criterion == 'entropy':
        impurity_func = entropy
    else:
        impurity_func = gini_impurity

    n_total = len(parent)
    n_left = len(left_child)
    n_right = len(right_child)

    parent_impurity = impurity_func(parent)
    weighted_child_impurity = (
        (n_left / n_total) * impurity_func(left_child) +
        (n_right / n_total) * impurity_func(right_child)
    )

    return parent_impurity - weighted_child_impurity
```

Приклад використання

```
# Батьківський вузол: 6 клас_0, 4 клас_1
parent = np.array([0]*6 + [1]*4)
# Ліва дитина після розбиття: 5 клас_0, 1 клас_1
left = np.array([0]*5 + [1]*1)
# Права дитина: 1 клас_0, 3 клас_1
right = np.array([0]*1 + [1]*3)

print(f"Entropy parent: {entropy(parent):.4f}")
print(f"Gini parent: {gini_impurity(parent):.4f}")
print(f"Information Gain: {information_gain(parent, left, right):.4f}")
```

2. Експерименти з гіперпараметрами

Завдання 1: вплив max_depth

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

depths = range(1, 21)
train_scores = []
test_scores = []

for depth in depths:
    tree = DecisionTreeClassifier(max_depth=depth, random_state=42)
    tree.fit(X_train, y_train)

    train_scores.append(tree.score(X_train, y_train))
    test_scores.append(tree.score(X_test, y_test))

# Візуалізація
plt.figure(figsize=(10, 6))
plt.plot(depths, train_scores, label='Train', marker='o')
plt.plot(depths, test_scores, label='Test', marker='s')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.title('Model Complexity vs Performance')
plt.legend()
plt.grid(True)
plt.show()
```

Завдання 2: комплексний Grid Search

```
param_grid = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5, 10],
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random']
}

grid_search = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid,
    cv=5,
```

```
    scoring='f1',
    verbose=1,
    n_jobs=-1
)

grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
print(f"Best CV score: {grid_search.best_score_:.4f}")
```

Завдання 3: аналіз результатів Grid Search

```
results_df = pd.DataFrame(grid_search.cv_results_)
results_df = results_df.sort_values('rank_test_score')

# Візуалізація топ-10 конфігурацій
top_configs = results_df.head(10)
plt.figure(figsize=(12, 6))
plt.barh(range(10), top_configs['mean_test_score'])
plt.yticks(range(10), [str(p) for p in top_configs['params']])
plt.xlabel('Mean F1 Score')
plt.title('Top 10 Hyperparameter Configurations')
plt.tight_layout()
plt.show()
```

3. Візуалізація та інтерпретація

Завдання 1: візуалізація дерева

```
from sklearn import tree
import graphviz

best_tree = grid_search.best_estimator_

# Метод 1: Matplotlib
plt.figure(figsize=(20, 10))
tree.plot_tree(
    best_tree,
    feature_names=feature_names,
    class_names=['Class_0', 'Class_1'],
    filled=True,
    rounded=True,
    fontsize=10
)
plt.title('Decision Tree Visualization')
plt.show()

# Метод 2: Graphviz (кращий вигляд)
dot_data = tree.export_graphviz(
    best_tree,
    out_file=None,
    feature_names=feature_names,
    class_names=['Class_0', 'Class_1'],
    filled=True,
    rounded=True,
    special_characters=True
)
```

```
graph = graphviz.Source(dot_data)
graph.render("decision_tree", format='png', cleanup=True)
# Збережеться файл decision_tree.png
```

Завдання 2: аналіз правил

```
def get_tree_rules(tree, feature_names):
    """
    Витягнути правила з дерева у текстовому вигляді
    """
    tree_ = tree.tree_
    feature_name = [
        feature_names[i] if i != tree.tree_.TREE_UNDEFINED else "undefined!"
        for i in tree_.feature
    ]

    def recurse(node, depth, rules_list, current_rule):
        indent = "  " * depth
        if tree_.feature[node] != tree.tree_.TREE_UNDEFINED:
            name = feature_name[node]
            threshold = tree_.threshold[node]

            # Ліва гілка (<=)
            left_rule = current_rule + f"{name} <= {threshold:.2f}"
            recurse(tree_.children_left[node],  depth  +  1,  rules_list,
left_rule + " AND ")

            # Права гілка (>)
            right_rule = current_rule + f"{name} > {threshold:.2f}"
            recurse(tree_.children_right[node],  depth  +  1,  rules_list,
right_rule + " AND ")
        else:
            # Листовий вузол
            class_name = np.argmax(tree_.value[node][0])
            rules_list.append({
                'rule': current_rule[:-5],  # видалити останнє " AND "
                'class': class_name,
                'samples': tree_.n_node_samples[node]
            })

    rules = []
    recurse(0, 1, rules, "IF ")
    return pd.DataFrame(rules)

# Отримання правил
rules_df = get_tree_rules(best_tree, feature_names)
print("Top 5 rules by sample count:")
print(rules_df.sort_values('samples', ascending=False).head())
```

Завдання 3: Feature Importance

```
importances = best_tree.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.bar(range(len(importances)), importances[indices])
plt.xticks(range(len(importances)),  [feature_names[i]  for  i  in  indices],
rotation=45)
plt.tight_layout()
```

```
plt.show()

# Таблиця важливості
importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': importances
}).sort_values('importance', ascending=False)

print(importance_df)
```

4. Обрізка дерев (Pruning)

Завдання: Cost Complexity Pruning

```
# 1. Побудова повного дерева
full_tree = DecisionTreeClassifier(random_state=42)
full_tree.fit(X_train, y_train)

# 2. Отримання шляху обрізки
path = full_tree.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
impurities = path.impurities

# 3. Навчання дерев з різними alpha
trees = []
for ccp_alpha in ccp_alphas:
    tree = DecisionTreeClassifier(random_state=42, ccp_alpha=ccp_alpha)
    tree.fit(X_train, y_train)
    trees.append(tree)

# 4. Оцінка на train та test
train_scores = [tree.score(X_train, y_train) for tree in trees]
test_scores = [tree.score(X_test, y_test) for tree in trees]

# Візуалізація
fig, ax = plt.subplots(1, 2, figsize=(15, 5))

# Графік 1: Accuracy vs alpha
ax[0].plot(ccp_alphas,     train_scores,     marker='o',     label='Train',
drawstyle="steps-post")
ax[0].plot(ccp_alphas,      test_scores,      marker='s',      label='Test',
drawstyle="steps-post")
ax[0].set_xlabel('alpha')
ax[0].set_ylabel('Accuracy')
ax[0].set_title('Accuracy vs alpha for training and testing sets')
ax[0].legend()
ax[0].grid()

# Графік 2: Кількість вузлів vs alpha
node_counts = [tree.tree_.node_count for tree in trees]
depth = [tree.tree_.max_depth for tree in trees]

ax[1].plot(ccp_alphas,   node_counts,   marker='o',   label='Node   Count',
drawstyle="steps-post")
ax[1].plot(ccp_alphas, depth, marker='s', label='Depth', drawstyle="steps-
post")
ax[1].set_xlabel('alpha')
ax[1].set_ylabel('Count/Depth')
ax[1].set_title('Tree Complexity vs alpha')
ax[1].legend()
```

```
ax[1].grid()

plt.tight_layout()
plt.show()

# Вибір оптимального alpha
optimal_idx = np.argmax(test_scores)
optimal_alpha = ccp_alphas[optimal_idx]
optimal_tree = trees[optimal_idx]

print(f"Optimal alpha: {optimal_alpha:.6f}")
print(f"Test accuracy: {test_scores[optimal_idx]:.4f}")
print(f"Tree depth: {depth[optimal_idx]}")
print(f"Node count: {node_counts[optimal_idx]}")
```

Контрольні питання

1. У чому різниця між Gini та Entropy?
2. Що таке Information Gain?
3. Які параметри контролюють складність дерева?
4. Що таке pre-pruning та post-pruning?
5. Чому дерева рішень схильні до overfitting?
6. Як інтерпретувати feature importance?

Тема 9. Ансамблеві методи. Випадковий ліс

Мета: освоїти принципи ансамблевого навчання, порівняти різні підходи та навчитися оптимізувати Random Forest і Gradient Boosting.

Завдання для самостійного виконання

1. Розуміння ансамблів. Демонстрація "мудрості натовпу"

```
# Симуляція: кожен класифікатор має точність 0.6
n_models = 100
n_samples = 1000
accuracy_single = 0.6

# Генеруємо прогнози від кожної моделі
np.random.seed(42)
predictions = np.random.rand(n_samples, n_models) < accuracy_single

# Реальні мітки
y_true = np.random.randint(0, 2, n_samples)

# Прогноз ансамблю (голосування більшості)
ensemble_pred = (predictions.sum(axis=1) > n_models / 2).astype(int)

# Точність окремих моделей vs ансамбль
individual_accuracies = [(predictions[:, i] == y_true).mean()
                         for i in range(n_models)]
ensemble_accuracy = (ensemble_pred == y_true).mean()

print(f"Середня точність окремої моделі:
{np.mean(individual_accuracies):.4f}")
print(f"Точність ансамблю: {ensemble_accuracy:.4f}")

# Візуалізація
plt.figure(figsize=(10, 6))
plt.hist(individual_accuracies, bins=30, alpha=0.7, label='Individual
Models')
plt.axvline(ensemble_accuracy, color='r', linestyle='--', linewidth=2,
           label=f'Ensemble ({ensemble_accuracy:.4f})')
plt.xlabel('Accuracy')
plt.ylabel('Count')
plt.title('Wisdom of the Crowd: Individual vs Ensemble Performance')
plt.legend()
plt.show()
```

2. Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
import lightgbm as lgb
```

Завдання 1: Sklearn Gradient Boosting

```
gb = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
```

```
    random_state=42
)

gb.fit(X_train, y_train)

# Learning curve для GB
train_scores_gb = []
test_scores_gb = []

for i, (train_pred, test_pred) in enumerate(zip(
    gb.staged_predict(X_train),
    gb.staged_predict(X_test)
)):
    train_scores_gb.append(accuracy_score(y_train, train_pred))
    test_scores_gb.append(accuracy_score(y_test, test_pred))

plt.figure(figsize=(10, 6))
plt.plot(train_scores_gb, label='Train')
plt.plot(test_scores_gb, label='Test')
plt.xlabel('Boosting Iteration')
plt.ylabel('Accuracy')
plt.title('Gradient Boosting: Learning Curve')
plt.legend()
plt.grid(True)
plt.show()
```

Завдання 2: XGBoost

```
xgb_model = xgb.XGBClassifier(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    random_state=42,
    eval_metric='logloss'
)

# Навчання з early stopping
xgb_model.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)],
    early_stopping_rounds=10,
    verbose=False
)

print(f"Best iteration: {xgb_model.best_iteration}")
print(f"Best score: {xgb_model.best_score:.4f}")
```

Завдання 3: LightGBM

```
lgb_train = lgb.Dataset(X_train, y_train)
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)

params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
```

```
    'bagging_freq': 5,
    'verbose': -1
}

lgb_model = lgb.train(
    params,
    lgb_train,
    num_boost_round=1000,
    valid_sets=[lgb_eval],
    callbacks=[lgb.early_stopping(stopping_rounds=10)]
)
```

Завдання 4: порівняння всіх ансамблів

```
models_comparison = {
    'Random Forest': best_rf,
    'Gradient Boosting': gb,
    'XGBoost': xgb_model,
    'LightGBM': lgb_model
}

comparison_results = []

for name, model in models_comparison.items():
    if name == 'LightGBM':
        y_pred = (model.predict(X_test) > 0.5).astype(int)
        y_proba = model.predict(X_test)
    else:
        y_pred = model.predict(X_test)
        y_proba = model.predict_proba(X_test)[:, 1]

    comparison_results.append({
        'Model': name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'F1': f1_score(y_test, y_pred),
        'AUC': roc_auc_score(y_test, y_proba)
    })

comparison_df = pd.DataFrame(comparison_results)
print(comparison_df)

# Візуалізація порівняння
comparison_df.set_index('Model').plot(kind='bar', figsize=(12, 6))
plt.title('Ensemble Methods Comparison')
plt.ylabel('Score')
plt.legend(loc='lower right')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

4. SHAP для інтерпретації: інтерпретація Random Forest за допомогою SHAP

```
import shap

# 1. Створення SHAP explainer
explainer = shap.TreeExplainer(best_rf)
shap_values = explainer.shap_values(X_test)
```

```
# Якщо бінарна класифікація, shap_values може бути списком
if isinstance(shap_values, list):
    shap_values = shap_values[1]  # Для класу 1

# 2. Summary plot (глобальна важливість)
plt.figure()
shap.summary_plot(shap_values,     X_test,     feature_names=feature_names,
show=False)
plt.tight_layout()
plt.show()

# 3. Force plot для окремого прогнозу
# Пояснення для першого зразка
plt.figure()
shap.force_plot(
    explainer.expected_value[1]   if   isinstance(explainer.expected_value,
list)
    else explainer.expected_value,
    shap_values[0,:],
    X_test.iloc[0,:],
    feature_names=feature_names,
    matplotlib=True
)
plt.show()

# 4. Dependence plot
# Показує залежність SHAP value від значення ознаки
shap.dependence_plot(
    "most_important_feature",  # Замініть на реальну назву
    shap_values,
    X_test,
    feature_names=feature_names
)
```

Практичний кейс: прогнозування відтоку клієнтів

Завдання: повний проєкт з ансамблями

1. Завантажте Telco Customer Churn датасет
2. Проведіть повний EDA
3. Feature Engineering
4. Побудуйте 5 моделей:
   - Logistic Regression (baseline)
   - Decision Tree
   - Random Forest
   - XGBoost
   - *LightGBM*
5. *Оптимізуйте гіперпараметри найкращих 2-3 моделей*

6. Створіть stacking ensemble

7. Проаналізуйте feature importance

8. SHAP аналіз

9. Бізнес-рекомендації: яких клієнтів треба утримувати?

Контрольні питання

1. У чому різниця між bagging та boosting?
2. Чому Random Forest менш схильний до overfitting?
3. Що таке out-of-bag error?
4. Як працює Gradient Boosting?
5. Коли використовувати XGBoost, а коли LightGBM?
6. Що таке SHAP values?

Тема 10. Кластеризація. Метод K-середніх

Мета: освоїти методи навчання без учителя, зокрема алгоритми кластеризації, навчитися визначати оптимальну кількість кластерів та інтерпретувати результати.

Завдання для самостійного виконання

1. Розуміння алгоритму K-Means

Завдання 1: Реалізація K-Means "з нуля"

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
class SimpleKMeans:
    """
    Проста реалізація K-Means для навчальних цілей
    """

    def __init__(self, n_clusters=3, max_iters=100):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
        self.centroids = None
        self.labels = None

    def fit(self, X):
        # 1. Випадкова ініціалізація центроїдів
        random_indices    =    np.random.choice(len(X),    self.n_clusters,
replace=False)
        self.centroids = X[random_indices]

        for iteration in range(self.max_iters):
            # 2. Assignment: призначити кожну точку до найближчого центроїда
            distances      =      np.sqrt(((X      -      self.centroids[:,
np.newaxis])**2).sum(axis=2))
            self.labels = np.argmin(distances, axis=0)

            # 3. Update: перерахувати центроїди
            new_centroids = np.array([X[self.labels == k].mean(axis=0)
                                      for k in range(self.n_clusters)])

            # 4. Перевірка конвергенції
            if np.allclose(self.centroids, new_centroids):
                break

            self.centroids = new_centroids

        return self

    def predict(self, X):
        distances       =       np.sqrt(((X       -       self.centroids[:,
np.newaxis])**2).sum(axis=2))
        return np.argmin(distances, axis=0)
```

Завдання 2: Візуалізація кроків алгоритму

```
def visualize_kmeans_steps(X, n_clusters=3, n_steps=5):
    """
    Анімація кроків K-Means
    """
    kmeans = KMeans(n_clusters=n_clusters, init='random', n_init=1,
max_iter=1)

    fig, axes = plt.subplots(1, n_steps, figsize=(20, 4))

    for step in range(n_steps):
        kmeans.fit(X)

        axes[step].scatter(X[:, 0], X[:, 1], c=kmeans.labels_,
cmap='viridis', alpha=0.6)
        axes[step].scatter(kmeans.cluster_centers_[:, 0],
                           kmeans.cluster_centers_[:, 1],
                           c='red', marker='X', s=200, edgecolors='black')
        axes[step].set_title(f'Iteration {step + 1}')
        axes[step].set_xlabel('Feature 1')
        axes[step].set_ylabel('Feature 2')

        # Продовжити ітерації
        kmeans.max_iter = step + 2

    plt.tight_layout()
    plt.show()

# Застосування на синтетичних даних
from sklearn.datasets import make_blobs

X, y_true = make_blobs(n_samples=300, centers=3, n_features=2,
                       cluster_std=0.6, random_state=42)
visualize_kmeans_steps(X, n_clusters=3, n_steps=5)
```

2. Визначення оптимальної кількості кластерів

Завдання 1: метод ліктя

```
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score

inertias = []
silhouette_scores = []
davies_bouldin_scores = []
calinski_scores = []

K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)

    inertias.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X, kmeans.labels_))
    davies_bouldin_scores.append(davies_bouldin_score(X, kmeans.labels_))
    calinski_scores.append(calinski_harabasz_score(X, kmeans.labels_))

# Візуалізація всіх метрик
```

```
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# Elbow Method
axes[0, 0].plot(K_range, inertias, 'o-')
axes[0, 0].set_xlabel('Number of Clusters (k)')
axes[0, 0].set_ylabel('Inertia (WCSS)')
axes[0, 0].set_title('Elbow Method')
axes[0, 0].grid(True)

# Silhouette Score (вище - краще)
axes[0, 1].plot(K_range, silhouette_scores, 'o-', color='green')
axes[0, 1].set_xlabel('Number of Clusters (k)')
axes[0, 1].set_ylabel('Silhouette Score')
axes[0, 1].set_title('Silhouette Analysis')
axes[0, 1].grid(True)

# Davies-Bouldin Index (нижче - краще)
axes[1, 0].plot(K_range, davies_bouldin_scores, 'o-', color='red')
axes[1, 0].set_xlabel('Number of Clusters (k)')
axes[1, 0].set_ylabel('Davies-Bouldin Index')
axes[1, 0].set_title('Davies-Bouldin Index (lower is better)')
axes[1, 0].grid(True)

# Calinski-Harabasz Index (вище - краще)
axes[1, 1].plot(K_range, calinski_scores, 'o-', color='purple')
axes[1, 1].set_xlabel('Number of Clusters (k)')
axes[1, 1].set_ylabel('Calinski-Harabasz Score')
axes[1, 1].set_title('Calinski-Harabasz Score (higher is better)')
axes[1, 1].grid(True)

plt.tight_layout()
plt.show()
```

Завдання 2: Silhouette Plot

```
from sklearn.metrics import silhouette_samples

def plot_silhouette(X, n_clusters):
    """
    Детальний Silhouette plot для кожного кластера
    """
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(X)

    silhouette_avg = silhouette_score(X, cluster_labels)
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    fig, ax = plt.subplots(figsize=(10, 7))
    y_lower = 10

    for i in range(n_clusters):
        # Силуети для i-го кластера
        ith_cluster_silhouette_values                                    =
sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = plt.cm.viridis(float(i) / n_clusters)
```

```
        ax.fill_betweenx(np.arange(y_lower, y_upper),
                         0, ith_cluster_silhouette_values,
                         facecolor=color, edgecolor=color, alpha=0.7)

        ax.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        y_lower = y_upper + 10

    ax.set_title(f'Silhouette Plot for {n_clusters} clusters')
    ax.set_xlabel('Silhouette Coefficient')
    ax.set_ylabel('Cluster Label')
    ax.axvline(x=silhouette_avg, color="red", linestyle="--",
               label=f'Average: {silhouette_avg:.3f}')
    ax.legend()
    ax.set_yticks([])
    plt.show()

# Візуалізація для різних k
for k in [2, 3, 4, 5]:
    plot_silhouette(X, k)
```

Завдання 3: Gap Statistic

```
def gap_statistic(X, k_max=10, n_refs=10):
    """
    Розрахунок Gap Statistic для визначення оптимального k
    """
    gaps = []
        for k in range(1, k_max + 1):
        # Кластеризація реальних даних
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(X)
        real_dispersion = np.log(kmeans.inertia_)

        # Кластеризація випадкових даних
        ref_dispersions = []
        for _ in range(n_refs):
            # Генеруємо випадкові дані в тих же межах
            random_data  =  np.random.uniform(X.min(axis=0),  X.max(axis=0),
X.shape)
            kmeans_ref = KMeans(n_clusters=k, random_state=42)
            kmeans_ref.fit(random_data)
            ref_dispersions.append(np.log(kmeans_ref.inertia_))

        gap = np.mean(ref_dispersions) - real_dispersion
        gaps.append(gap)

    # Візуалізація
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, k_max + 1), gaps, 'o-')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Gap Statistic')
    plt.title('Gap Statistic Method')
    plt.grid(True)
    plt.show()

    return gaps

gaps = gap_statistic(X, k_max=10, n_refs=10)
optimal_k_gap = np.argmax(gaps) + 1
print(f"Оптимальне k за Gap Statistic: {optimal_k_gap}")
```

3. Альтернативні методи кластеризації

Завдання 1: Hierarchical Clustering

```
from sklearn.cluster import AgglomerativeClustering, DBSCAN
from scipy.cluster.hierarchy import dendrogram, linkage

# Побудова дендрограми
linkage_matrix = linkage(X, method='ward')

plt.figure(figsize=(12, 8))
dendrogram(linkage_matrix, truncate_mode='level', p=5)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index or (Cluster Size)')
plt.ylabel('Distance')
plt.axhline(y=10, c='red', linestyle='--', label='Cut threshold')
plt.legend()
plt.show()

# Агломеративна кластеризація
hierarchical = AgglomerativeClustering(n_clusters=3, linkage='ward')
hierarchical_labels = hierarchical.fit_predict(X)

# Візуалізація результатів
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=hierarchical_labels, cmap='viridis')
plt.title('Hierarchical Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Cluster')
plt.show()
```

Завдання 2: DBSCAN (Density-Based)

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X)

# Кількість кластерів (виключаючи шум -1)
n_clusters_dbscan = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels else
0)
n_noise = list(dbscan_labels).count(-1)

print(f"DBSCAN знайшов {n_clusters_dbscan} кластерів")
print(f"Шумових точок: {n_noise}")

# Візуалізація
plt.figure(figsize=(10, 6))
unique_labels = set(dbscan_labels)
colors = plt.cm.viridis(np.linspace(0, 1, len(unique_labels)))

for k, col in zip(unique_labels, colors):
    if k == -1:
        # Шум - чорні точки
        col = 'black'

    class_member_mask = (dbscan_labels == k)
    xy = X[class_member_mask]
    plt.scatter(xy[:, 0], xy[:, 1], c=[col], label=f'Cluster {k}' if k != -
1 else 'Noise',
                alpha=0.6, edgecolors='k', s=50)
```

```
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Завдання 3: Порівняння методів

```
from sklearn.metrics import adjusted_rand_score,
normalized_mutual_info_score

methods = {
    'K-Means': KMeans(n_clusters=3, random_state=42).fit_predict(X),
    'Hierarchical': hierarchical_labels,
    'DBSCAN': dbscan_labels
}

# Порівняння з справжніми мітками (якщо є)
if 'y_true' in locals():
    comparison = []
    for name, labels in methods.items():
        comparison.append({
            'Method': name,
            'ARI': adjusted_rand_score(y_true, labels),
            'NMI': normalized_mutual_info_score(y_true, labels),
            'Silhouette': silhouette_score(X, labels) if -1 not in labels
else np.nan
        })

    comparison_df = pd.DataFrame(comparison)
    print(comparison_df)
```

4. Практична сегментація клієнтів

Завдання: застосування K-Means для реальної бізнес-задачі. Завантажте Mall Customers датасет або створіть синтетичні дані.

Структура: CustomerID, Age, Annual Income, Spending Score.

Приклад аналізу:

```
# 1. Стандартизація даних
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[['Annual Income', 'Spending Score']])

# 2. Визначення оптимального k
# (використайте методи з попереднього завдання)

# 3. Кластеризація
optimal_k = 5
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# 4. Профілювання кластерів
```

```
cluster_profiles = df.groupby('Cluster').agg({
    'Age': ['mean', 'std'],
    'Annual Income': ['mean', 'std'],
    'Spending Score': ['mean', 'std'],
    'CustomerID': 'count'
}).round(2)

cluster_profiles.columns = ['_'.join(col) for col in
cluster_profiles.columns]
cluster_profiles = cluster_profiles.rename(columns={'CustomerID_count':
'Size'})

print("Профілі кластерів:")
print(cluster_profiles)

# 5. Візуалізація та інтерпретація
plt.figure(figsize=(12, 8))
scatter = plt.scatter(df['Annual Income'], df['Spending Score'],
                      c=df['Cluster'], cmap='viridis', s=100, alpha=0.6,
edgecolors='k')
plt.scatter(kmeans.cluster_centers_[:, 0] * scaler.scale_[0] +
scaler.mean_[0],
            kmeans.cluster_centers_[:, 1] * scaler.scale_[1] +
scaler.mean_[1],
            c='red', marker='X', s=300, edgecolors='black', linewidths=2,
label='Centroids')
plt.xlabel('Annual Income (k)')
plt.ylabel('Spending Score (1-100)')
plt.title('Customer Segmentation')
plt.colorbar(scatter, label='Cluster')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# 6. Бізнес-інтерпретація кластерів
# Наприклад:
# Cluster 0: Висока зарплата, високі витрати -> "VIP клієнти"
# Cluster 1: Висока зарплата, низькі витрати -> "Потенційні клієнти"
# Cluster 2: Низька зарплата, високі витрати -> "Імпульсивні покупці"
# і т.д.

# 7. Рекомендації для маркетингу
def generate_recommendations(cluster_profiles):
    """
    Генерація маркетингових рекомендацій для кожного сегменту
    """
    recommendations = {}

    for cluster_id in cluster_profiles.index:
        profile = cluster_profiles.loc[cluster_id]

        # Логіка на основі характеристик кластера
        if profile['Annual Income_mean'] > 70 and profile['Spending
Score_mean'] > 70:
            recommendations[cluster_id] = "VIP Program, Premium Products"
        elif profile['Annual Income_mean'] > 70 and profile['Spending
Score_mean'] < 40:
            recommendations[cluster_id] = "Loyalty Programs, Exclusive
Offers"
        elif profile['Annual Income_mean'] < 40 and profile['Spending
Score_mean'] > 60:
```

```
            recommendations[cluster_id]    =    "Budget-Friendly    Options,
Promotions"
        else:
            recommendations[cluster_id] = "General Marketing"

    return recommendations

recommendations = generate_recommendations(cluster_profiles)
for cluster, rec in recommendations.items():
   print(f"Cluster {cluster}: {rec}")
```

Контрольні питання

1. Як працює алгоритм K-Means?
2. Які обмеження має K-Means?
3. Що таке Silhouette Score і як його інтерпретувати?
4. У чому переваги DBSCAN над K-Means?
5. Як обрати оптимальну кількість кластерів?
6. Що таке проблема "локальних мінімумів" в K-Means?

Тема 11. Зменшення розмірності. Метод головних компонент (PCA)

Навчальна мета:

Мета: освоїти методи зменшення розмірності даних, зрозуміти математичну основу PCA та навчитися застосовувати альтернативні методи.

Завдання для самостійного виконання

1. Математична основа PCA

Завдання 1: реалізація PCA "з нуля"

```
import numpy as np
from sklearn.decomposition import PCA

class SimplePCA:
    """
    Проста реалізація PCA для розуміння алгоритму
    """

    def __init__(self, n_components=2):
        self.n_components = n_components
        self.components = None
        self.mean = None
        self.explained_variance = None

    def fit(self, X):
        # 1. Центрування даних
        self.mean = np.mean(X, axis=0)
        X_centered = X - self.mean

        # 2. Розрахунок коваріаційної матриці
        cov_matrix = np.cov(X_centered.T)

        # 3. Знаходження власних значень та векторів
        eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

        # 4. Сортування за спаданням власних значень
        idx = eigenvalues.argsort()[::-1]
        eigenvalues = eigenvalues[idx]
        eigenvectors = eigenvectors[:, idx]

        # 5. Вибір перших n_components компонент
        self.components = eigenvectors[:, :self.n_components]
        self.explained_variance = eigenvalues[:self.n_components]

        return self

    def transform(self, X):
        # Проєкція на головні компоненти
        X_centered = X - self.mean
        return np.dot(X_centered, self.components)

    def fit_transform(self, X):
        self.fit(X)
        return self.transform(X)
```

```
    def inverse_transform(self, X_transformed):
        # Відновлення з простору головних компонент
        return np.dot(X_transformed, self.components.T) + self.mean
```

Завдання 2: Порівняння з sklearn

```
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

# Власна реалізація
simple_pca = SimplePCA(n_components=2)
X_pca_simple = simple_pca.fit_transform(X)

# Sklearn реалізація
sklearn_pca = PCA(n_components=2)
X_pca_sklearn = sklearn_pca.fit_transform(X)

# Перевірка (можуть відрізнятися знаком)
print("Власна реалізація схожа на sklearn:",
      np.allclose(np.abs(X_pca_simple), np.abs(X_pca_sklearn)))

# Візуалізація
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

axes[0].scatter(X_pca_simple[:, 0], X_pca_simple[:, 1], c=y, cmap='viridis')
axes[0].set_title('Custom PCA Implementation')
axes[0].set_xlabel('PC1')
axes[0].set_ylabel('PC2')

axes[1].scatter(X_pca_sklearn[:,    0],    X_pca_sklearn[:,    1],    c=y,
cmap='viridis')
axes[1].set_title('Sklearn PCA')
axes[1].set_xlabel('PC1')
axes[1].set_ylabel('PC2')

plt.tight_layout()
plt.show()
```

2. Аналіз Explained Variance

Завдання: визначення оптимальної кількості компонент

```
# 1. PCA з усіма компонентами
pca_full = PCA()
pca_full.fit(X)

# 2. Explained Variance Ratio
explained_variance_ratio = pca_full.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance_ratio)

# Візуалізація
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Scree plot
```

```
axes[0].bar(range(1,       len(explained_variance_ratio)       +       1),
explained_variance_ratio)
axes[0].set_xlabel('Principal Component')
axes[0].set_ylabel('Explained Variance Ratio')
axes[0].set_title('Scree Plot')
axes[0].grid(True, alpha=0.3)

# Cumulative explained variance
axes[1].plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
'o-')
axes[1].axhline(y=0.95, color='r', linestyle='--', label='95% threshold')
axes[1].axhline(y=0.90,    color='orange',    linestyle='--',    label='90%
threshold')
axes[1].set_xlabel('Number of Components')
axes[1].set_ylabel('Cumulative Explained Variance')
axes[1].set_title('Cumulative Explained Variance')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Знайти кількість компонент для 95% variance
n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1
print(f"Кількість компонент для 95% variance: {n_components_95}")

# 3. Бі-plot (для 2D)
def biplot(score, coeff, labels=None, y=None):
    """
    Візуалізація PCA з векторами ознак
    """
    xs = score[:, 0]
    ys = score[:, 1]
    n = coeff.shape[0]

    plt.figure(figsize=(10, 8))

    # Точки даних
    if y is not None:
        scatter = plt.scatter(xs, ys, c=y, cmap='viridis', alpha=0.6, s=50)
        plt.colorbar(scatter)
    else:
        plt.scatter(xs, ys, alpha=0.6, s=50)

    # Вектори ознак
    for i in range(n):
        plt.arrow(0, 0, coeff[i, 0]*3, coeff[i, 1]*3,
                  color='r', alpha=0.5, head_width=0.1)
        if labels is None:
            plt.text(coeff[i, 0]*3.5, coeff[i, 1]*3.5,
                     f"Var{i+1}", color='g', ha='center', va='center')
        else:
            plt.text(coeff[i, 0]*3.5, coeff[i, 1]*3.5,
                     labels[i], color='g', ha='center', va='center')

    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.title("PCA Biplot")
    plt.grid(True, alpha=0.3)
    plt.show()
```

```
# Застосування biplot
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X)
biplot(X_pca_2d, pca_2d.components_, labels=iris.feature_names, y=y)
```

Практичне застосування

Завдання: PCA для візуалізації Fashion MNIST

```
from sklearn.datasets import fetch_openml

# Завантаження даних
print("Завантаження Fashion MNIST...")
fashion_mnist = fetch_openml

('Fashion-MNIST', version=1, cache=True, parser='auto') X_fashion = fashion_mnist.data[:5000] # Беремо перші 5000 для швидкості y_fashion = fashion_mnist.target[:5000].astype(int)
Стандартизація
from sklearn.preprocessing import StandardScaler scaler = StandardScaler()
X_fashion_scaled = scaler.fit_transform(X_fashion)
PCA
print("Виконання PCA...") pca_fashion = PCA(n_components=50) X_fashion_pca = pca_fashion.fit_transform(X_fashion_scaled)
Аналіз explained variance
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(np.cumsum(pca_fashion.explained_variance_ratio_))
plt.xlabel('Number of Components') plt.ylabel('Cumulative Explained Variance') plt.title('Fashion MNIST: PCA Explained Variance') plt.grid(True)
plt.subplot(1, 2, 2)
Візуалізація перших 2 компонент
scatter = plt.scatter(X_fashion_pca[:, 0], X_fashion_pca[:, 1], c=y_fashion, cmap='tab10', s=5, alpha=0.5) plt.xlabel('PC1') plt.ylabel('PC2')
plt.title('Fashion MNIST: First 2 Principal Components')
plt.colorbar(scatter, label='Class')
plt.tight_layout() plt.show()
Реконструкція зображень
def plot_reconstruction(original, reconstructed, n_samples=5): """ Порівняння оригінальних та реконструйованих зображень """ fig, axes = plt.subplots(2, n_samples, figsize=(15, 6))
for i in range(n_samples):
    # Оригінал
    axes[0, i].imshow(original[i].reshape(28, 28), cmap='gray')
    axes[0, i].axis('off')
    if i == 0:
        axes[0, i].set_title('Original', fontsize=12)

    # Реконструкція
    axes[1, i].imshow(reconstructed[i].reshape(28, 28), cmap='gray')
    axes[1, i].axis('off')
    if i == 0:
        axes[1, i].set_title('Reconstructed', fontsize=12)

plt.tight_layout()
plt.show()
Реконструкція з різною кількістю компонент
for n_comp in [10, 50, 100]: pca_rec = PCA(n_components=n_comp) X_pca_rec = pca_rec.fit_transform(X_fashion_scaled[:5]) X_reconstructed =
```

```
pca_rec.inverse_transform(X_pca_rec)           X_reconstructed           =
scaler.inverse_transform(X_reconstructed)
print(f"\nРеконструкція з {n_comp} компонентами "
      f"({pca_rec.explained_variance_ratio_.sum()*100:.1f}% variance)")
plot_reconstruction(X_fashion[:5], X_reconstructed, n_samples=5)
```

Контрольні питання

1. Що таке "прокляття розмірності"?
2. Як PCA знаходить головні компоненти?
3. Що означає explained variance ratio?
4. У чому різниця між PCA та t-SNE?
5. Коли використовувати PCA, а коли UMAP?
6. Чи можна використовувати PCA для feature selection?

Тема 12. Пошук асоціативних правил

Мета: освоїти методи виявлення прихованих взаємозв'язків у транзакційних даних, зрозуміти математичне підґрунтя метрик якості правил та навчитися застосовувати алгоритм Apriori для аналізу ринкового кошика.

Завдання для самостійного виконання

1. Опрацювання метрик якості правил. Вивчіть визначення Support, Confidence та Lift. Розрахуйте вручну ці метрики для невеликого набору з 5 транзакцій, щоб зрозуміти, як значення Lift > 1 вказує на позитивну залежність між товарами.
2. Реалізація алгоритму Apriori. Підготуйте дані за допомогою TransactionEncoder з бібліотеки mlxtend (перетворення списків покупок у розріджену матрицю True/False). Проведіть експерименти з різними порогами min_support, щоб побачити, як змінюється кількість згенерованих частих наборів .
3. Фільтрація та інтерпретація правил. Згенеруйте асоціативні правила та відфільтруйте їх за високим значенням Lift та Confidence. Сформулюйте бізнес-рекомендації (наприклад, щодо розміщення товарів на полицях) на основі знайдених правил.
4.

Приклад практичної реалізації

```
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd

# 1. Підготовка даних
dataset = [['Milk', 'Bread', 'Eggs'], ['Bread', 'Butter'], ['Milk', 'Bread',
'Butter'], ['Milk', 'Eggs']]
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

```
# 2. Пошук частих наборів (min_support = 50%)
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)

# 3. Генерація правил (min_confidence = 70%)
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.7)
print(rules[['antecedents', 'consequents', 'support', 'confidence',
'lift']])
```

Контрольні питання

1. У чому полягає властивість «антимонотонності» підтримки в алгоритмі Apriori? [7]
2. Чому висока достовірність (Confidence) не завжди означає корисність правила (роль метрики Lift)?
3. Як асоціативні правила використовуються в рекомендаційних системах?

Тема 13. Основи аналізу часових рядів

Мета: ознайомитися зі специфікою даних, що залежать від часу, навчитися проводити декомпозицію ряду на компоненти та перевіряти його на стаціонарність.

Завдання для самостійного виконання

1. Структурний аналіз ряду. Вивчіть поняття тренду, сезонності та залишків. Використовуйте seasonal_decompose для візуалізації цих компонентів на прикладі даних про продажі або температуру.
2. Перевірка стаціонарності. Опрацюйте поняття стаціонарного ряду (сталість середнього та дисперсії). Реалізуйте тест Дікі-Фуллера. Спробуйте перетворити нестаціонарний ряд у стаціонарний за допомогою методу різниць (.diff()).

Приклад практичної реалізації

```
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt

# 1. Декомпозиція (модель 'additive')
analysis = seasonal_decompose(df['value'], model='additive', period=12)
analysis.plot()
plt.show()

# 2. Тест на стаціонарність
result = adfuller(df['value'])
print(f'ADF Statistic: {result[0]}, p-value: {result[1]}')
# Якщо p-value > 0.05, ряд нестаціонарний
```

Контрольні питання

1. Яка різниця між адитивною та мультиплікативною моделями декомпозиції?
2. Чому стаціонарність є важливою умовою для багатьох моделей прогнозування?
3. Чим зважене ковзне середнє краще за просте?

Тема 14. Етика в Data Mining та MLOps

Мета: усвідомити етичні ризики при роботі з алгоритмами, навчитися виявляти упередженість у моделях та розуміти процеси підтримки моделей після їх розгортання

Завдання для самостійного виконання

1. Аудит моделі на справедливість (Fairness Audit).Навчіть модель класифікації (наприклад, прогноз доходу або кредиту). Оцініть метрики (Accuracy, Recall) окремо для різних соціальних груп (за статтю або віком) та проаналізуйте розбіжності.
2. Вивчення концепцій дрейфу. Опрацюйте поняття Data Drift та Concept Drift. Складіть список причин, чому модель, яка працювала сьогодні, може перестати бути точною через місяць.
3. Розробка плану моніторингу. Опишіть цикл зворотного зв’язку (Feedback Loop): як часто перенавчати модель та які метрики відстежувати в реальному часі.

Приклад аналізу справедливості (Fairness Check):

Завдання: Порівняння Recall для двох груп

```
groups = ['Male', 'Female']
for g in groups:
    mask = X_test['Gender'] == g
    recall = recall_score(y_test[mask], model.predict(X_test[mask]))
    print(f"Recall for {g}: {recall:.4f}")
```

Контрольні питання

1. Як biased data впливає на етичність рішень моделі?
2. Що таке "проксі-змінні" і чому видалення ознаки "Раса" не завжди гарантує відсутність дискримінації?
3. У чому різниця між Data Drift та Concept Drift?

Тема 15. Підсумковий проєкт

Мета: закріпити практичні навички, пройшовши повний ітеративний цикл OSEMN на реальному наборі даних за вибором студента.

Етапи виконання проєкту

1. Вибір та завантаження даних: пошук датасету на Kaggle або UCI. Формулювання задачі (класифікація/регресія/кластеризація).
2. Підготовка та очищення: робота з пропусками, аномаліями та перетворення типів даних.
3. Дослідницький аналіз: візуалізація розподілів, кореляційна матриця, пошук інсайтів.
4. Моделювання та інженерія ознак: створення нових ознак, вибір алгоритму, навчання та тюнінг гіперпараметрів.
5. Інтерпретація та висновки: аналіз важливості ознак, оцінка бізнес-ефекту та підготовка фінального звіту.

Вимоги до оформлення результатів

1. Звіт у форматі Jupyter Notebook з детальними текстовими поясненнями до кожного кроку.
2. Візуалізації мають бути підписані та прокоментовані.
3. Фінальний висновок повинен відповідати на питання: "Яку проблему вирішує дана модель?".

СПИСОК ЛІТЕРАТУРИ

1. McKinney W. Python for Data Analysis. 3rd ed. Boston : O'Reilly Media, 2022. 582 p. URL: https://wesmckinney.com/book/
2. Вавринюк К. Аналіз впливу вибору ознак на точність моделей класифікації. «Проблеми та перспективи реалізації та впровадження міждисциплінарних наукових досягнень»: збірник наукових праць з матеріалами IX Міжнародної наукової конференції, м.Луцьк, 13 червня, 2025р. / Міжнародний центрнаукових досліджень. – Вінниця: ТОВ УКРЛОГОС Груп 2025. С. 251–252.
3. Burkov A. The Hundred-Page Machine Learning Book. 2019. 150 p. URL: http://themlbook.com/
4. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 3rd ed. Boston : O'Reilly Media, 2022. 856 p. URL: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/
5. Печончик Н.Р., Приходько О. С. Порівняльний аналіз моделей машинного навчання для прогнозування міцності бетону// Міжвузівський збірник «Наукові нотатки». – Луцьк, 2025. – № 84. – С. 273–278. – DOI: https://doi.org/10.36910/775.24153966.2025.84.26
6. James G., Witten D., Hastie T., Tibshirani R. An Introduction to Statistical Learning: with Applications in R. 2nd ed. Springer, 2021. 607 p. (Springer Texts in Statistics). URL: https://www.statlearning.com/
7. Knaflic C. N. Storytelling with Data: A Data Visualization Guide for Business Professionals. New Jersey : John Wiley & Sons, 2015. 288 p. URL: https://www.storytellingwithdata.com/books
8. Hyndman R. J., Athanasopoulos G. Forecasting: Principles and Practice. 3rd ed. Melbourne : OTexts, 2021. URL: https://otexts.com/fpp3/
9. O'Neil C. Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy. New York : Crown, 2016. 272 p. URL:https://www.sfu.ca/~palys/ONeil-2018-WeaponsOfMathDestruction.pdf

DataMining: методичні вказівки до самостійної роботи для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Штучний інтелект та аналіз масивів даних» спеціальності «Прикладна математика» денної форми навчання/ уклад. К. Вавринюк Луцьк: ЛНТУ, 2025. – 63 с.

Комп'ютерний набір К. Вавринюк
Редактор К. Вавринюк

Підп. до друку « ___ »_____ 2025 р. Формат 60x84/16. Папір офс.
Гарн. Таймс. Ум. друк. арк. 3,9.
Тираж 50 прим.

Інформаційно-видавничий відділ
Луцького національного технічного університету
43018, м. Луцьк, вул. Львівська, 75
Друк – ІВВ ЛНТУ