ДОДАТОК А

Програмний код алгоритмів прогнозування витрат та доходів із застосуванням методу машинного навчання (RandomForestRegressor)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score
# Завантаження даних
# Припускаємо, що CSV-файл "financial_data.csv" містить
колонки: "date", "expenses", "revenues"
df = pd.read_csv("financial_data.csv")
# Перетворення колонки з датами у формат datetime та
сортування даних
df['date'] = pd.to_datetime(df['date'])
df.sort_values('date', inplace=True)
df.set_index('date', inplace=True)
# Створення лагових ознак для витрат та доходів (lag-1,
lag-2, lag-3)
df['lag_1_expenses'] = df['expenses'].shift(1)
df['lag_2_expenses'] = df['expenses'].shift(2)
df['lag_3_expenses'] = df['expenses'].shift(3)
df['lag_1_revenues'] = df['revenues'].shift(1)
df['lag_2_revenues'] = df['revenues'].shift(2)
df['lag_3_revenues'] = df['revenues'].shift(3)
# Додавання часових ознак: день тижня та місяць
df['day_of_week'] = df.index.dayofweek
df['month'] = df.index.month
# Видалення рядків з пропущеними значеннями (перші кілька
рядків через лагові ознаки)
df.dropna(inplace=True)
# Формування цільових змінних: прогнозування витрат та
доходів наступного дня
```

```
    df['target_expenses'] = df['expenses'].shift(-1)
    df['target_revenues'] = df['revenues'].shift(-1)
    # Видалення останнього рядка, у якому з'явиться NaN після створення цільових змінних
    df.dropna(inplace=True)
    # Визначення набору ознак для моделювання
    features = [
        'expenses', 'revenues',
        'lag_1_expenses', 'lag_2_expenses', 'lag_3_expenses',
        'lag_1_revenues', 'lag_2_revenues', 'lag_3_revenues',
        'day_of_week', 'month'
    ]
    X = df[features]
    y_exp = df['target_expenses']
    y_rev = df['target_revenues']
    # Розбиття даних на тренувальну та тестову вибірки за часовим принципом (80% для тренування, 20% для тестування)
    train_size = int(len(df) * 0.8)
    X_train, X_test = X.iloc[:train_size], X.iloc[train_size:]
    y_exp_train, y_exp_test = y_exp.iloc[:train_size], y_exp.iloc[train_size:]
    y_rev_train, y_rev_test = y_rev.iloc[:train_size], y_rev.iloc[train_size:]
    # Створення і навчання моделей прогнозування витрат та доходів
    model_exp = RandomForestRegressor(n_estimators=100, random_state=42)
    model_rev = RandomForestRegressor(n_estimators=100, random_state=42)
    model_exp.fit(X_train, y_exp_train)
    model_rev.fit(X_train, y_rev_train)
    # Прогнозування на тестових даних
    y_exp_pred = model_exp.predict(X_test)
    y_rev_pred = model_rev.predict(X_test)
    # Оцінка ефективності моделей за допомогою метрик MSE, MAE та R²
```

```
    mse_exp = mean_squared_error(y_exp_test, y_exp_pred)
    mae_exp = mean_absolute_error(y_exp_test, y_exp_pred)
    r2_exp = r2_score(y_exp_test, y_exp_pred)
    mse_rev = mean_squared_error(y_rev_test, y_rev_pred)
    mae_rev = mean_absolute_error(y_rev_test, y_rev_pred)
    r2_rev = r2_score(y_rev_test, y_rev_pred)
    print("Прогнозування витрат:")
    print("MSE: {:.2f}, MAE: {:.2f}, R²:
{:.2f}".format(mse_exp, mae_exp, r2_exp))
    print("\nПрогнозування доходів:")
    print("MSE: {:.2f}, MAE: {:.2f}, R²:
{:.2f}".format(mse_rev, mae_rev, r2_rev))
    # Візуалізація результатів прогнозування витрат
    plt.figure(figsize=(12, 6))
    plt.plot(y_exp_test.index, y_exp_test, label='Фактичні
витрати')
    plt.plot(y_exp_test.index, y_exp_pred, label='Прогнозовані
витрати', linestyle='--')
    plt.xlabel('Дата')
    plt.ylabel('Витрати')
    plt.title('Прогнозування витрат')
    plt.legend()
    plt.show()
    # Візуалізація результатів прогнозування доходів
    plt.figure(figsize=(12, 6))
    plt.plot(y_rev_test.index, y_rev_test, label='Фактичні
доходи')
    plt.plot(y_rev_test.index, y_rev_pred, label='Прогнозовані
доходи', linestyle='--')
    plt.xlabel('Дата')
    plt.ylabel('Доходи')
    plt.title('Прогнозування доходів')
    plt.legend()
    plt.show()
```

ДОДАТОК Б

Програмний код реалізації алгоритмів класифікації та кластеризації фінансових транзакцій

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,
accuracy_score
from sklearn.cluster import KMeans
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import hstack, csr_matrix
# ================================
# Завантаження даних
# ================================
# Припускаємо, що CSV-файл "transactions.csv" містить такі
колонки:
# 'transaction_id', 'date', 'amount', 'description',
'category'
df = pd.read_csv("transactions.csv")
# Виведемо перші рядки даних для перевірки
print("Перші рядки даних:")
print(df.head())
# Видаляємо рядки з пропущеними значеннями у критичних
колонках
df.dropna(subset=['description', 'amount', 'category'],
inplace=True)
# ================================
# Частина 1: Класифікація транзакцій
# ================================
```

```
    # Мета: класифікувати транзакції за категоріями на основі текстового опису та суми операції
    # Розділяємо дані на ознаки (X) та цільову змінну (y)
    X = df[['description', 'amount']]
    y = df['category']
    # Розбиття даних на тренувальну та тестову вибірки (80/20)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )
    # Текстова обробка: перетворення опису транзакцій у TF-IDF вектори
    vectorizer = TfidfVectorizer(max_features=500)
    X_train_desc_tfidf = vectorizer.fit_transform(X_train['description'])
    X_test_desc_tfidf = vectorizer.transform(X_test['description'])
    # Обробка числової ознаки "amount": перетворення у матрицю (reshape для сумісності)
    X_train_amount = X_train['amount'].values.reshape(-1, 1)
    X_test_amount = X_test['amount'].values.reshape(-1, 1)
    # Перетворення числових даних у формат sparse для подальшого об’єднання
    X_train_amount_sparse = csr_matrix(X_train_amount)
    X_test_amount_sparse = csr_matrix(X_test_amount)
    # Об’єднання текстових (TF-IDF) та числових ознак
    X_train_combined = hstack([X_train_desc_tfidf, X_train_amount_sparse])
    X_test_combined = hstack([X_test_desc_tfidf, X_test_amount_sparse])
    # Створення та навчання моделі класифікації (Random Forest)
    clf = RandomForestClassifier(n_estimators=100, random_state=42)
    clf.fit(X_train_combined, y_train)
    # Прогнозування категорій на тестовій вибірці
    y_pred = clf.predict(X_test_combined)
```

```
    # Оцінка ефективності моделі
    print("\nКласифікаційна точність:
{:.2f}".format(accuracy_score(y_test, y_pred)))
    print("\nЗвіт по класифікації:\n",
classification_report(y_test, y_pred))
    # ================================
    # Частина 2: Кластеризація транзакцій
    # ================================
    # Мета: згрупувати транзакції за схожими
характеристиками (без використання міток категорій)
    # Використовуємо дані всіх транзакцій (без стовпця
'category')
    # Використовуємо текстовий опис транзакцій для TF-IDF
векторизації
    # (можна створити окремий vectorizer для кластеризації,
щоб не залежати від попереднього навчання)
    vectorizer_clust = TfidfVectorizer(max_features=500)
    tfidf_matrix =
vectorizer_clust.fit_transform(df['description'])
    # Обробка числової ознаки "amount"
    amounts = df['amount'].values.reshape(-1, 1)
    amounts_sparse = csr_matrix(amounts)
    # Об’єднання ознак для кластеризації
    features_clust = hstack([tfidf_matrix, amounts_sparse])
    # Застосування алгоритму KMeans для кластеризації
    # Кількість кластерів може бути підібрана емпірично; в
цьому прикладі використовуємо 4
    n_clusters = 4
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(features_clust)
    # Додаємо отримані кластери до DataFrame
    df['cluster'] = clusters
    # Для візуалізації зменшимо розмірність даних до 2-х
вимірів за допомогою TruncatedSVD
    svd = TruncatedSVD(n_components=2, random_state=42)
    features_2d = svd.fit_transform(features_clust)
```

```
    # Побудова графіка кластерів
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=features_2d[:, 0], y=features_2d[:, 1],
hue=clusters, palette='viridis')
    plt.title("Кластеризація фінансових транзакцій")
    plt.xlabel("Компонента 1")
    plt.ylabel("Компонента 2")
    plt.legend(title="Кластер")
    plt.show()
    # Виведення кількості транзакцій у кожному кластері
    print("\nКількість транзакцій у кожному кластері:")
    print(df['cluster'].value_counts())
```

ДОДАТОК В

Програмний код тестування та оцінку результатів роботи розроблених алгоритмів

```
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd
    from sklearn.metrics import (mean_squared_error,
mean_absolute_error, r2_score,
                                 classification_report,
accuracy_score, silhouette_score)
    from sklearn.cluster import KMeans
    from sklearn.datasets import make_blobs
    # ------------------------------------------------------
    # Функції для оцінки роботи алгоритмів
    # ------------------------------------------------------
    def evaluate_forecasting(y_true, y_pred, title="Оцінка
прогнозування"):
        """
        Оцінює роботу алгоритмів прогнозування шляхом
обчислення метрик:
        MSE, MAE та R², а також будує графік порівняння
фактичних та прогнозованих значень.
        Параметри:
            y_true: pandas Series з фактичними значеннями
(індекс повинен містити дату або час)
            y_pred: numpy array або pandas Series з
прогнозованими значеннями
            title: Заголовок графіку
        """
        mse = mean_squared_error(y_true, y_pred)
        mae = mean_absolute_error(y_true, y_pred)
        r2 = r2_score(y_true, y_pred)
        print(f"{title}:")
        print("MSE: {:.2f}".format(mse))
```

```
        print("MAE: {:.2f}".format(mae))
        print("R²: {:.2f}".format(r2))
        plt.figure(figsize=(12, 6))
        plt.plot(y_true.index, y_true, label="Фактичні
значення")
        plt.plot(y_true.index, y_pred, label="Прогнозовані
значення", linestyle="--")
        plt.xlabel("Дата/Час")
        plt.ylabel("Значення")
        plt.title(title)
        plt.legend()
        plt.show()
    def evaluate_classification(y_true, y_pred):
        """
        Оцінює роботу алгоритмів класифікації, обчислюючи
точність та виводячи
        докладний звіт (precision, recall, F1-score для
кожного класу).
        Параметри:
            y_true: масив або список із фактичними мітками
класів
            y_pred: масив або список з прогнозованими мітками
класів
        """
        acc = accuracy_score(y_true, y_pred)
        print("Точність класифікації: {:.2f}".format(acc))
        print("Звіт по класифікації:\n",
classification_report(y_true, y_pred))
    def evaluate_clustering(features, labels):
        """
        Оцінює якість кластеризації за допомогою Silhouette
Score та
        візуалізує результати, якщо простір ознак має 2
виміри.

        Параметри:
```

```
            features: 2D numpy array з ознаками (наприклад,
після зниження розмірності)
            labels: масив з призначеними номерами кластерів
        """
        sil_score = silhouette_score(features, labels)
        print("Silhouette Score: {:.2f}".format(sil_score))

        plt.figure(figsize=(10, 6))
        plt.scatter(features[:, 0], features[:, 1], c=labels,
cmap='viridis', marker='o')
        plt.xlabel("Компонента 1")
        plt.ylabel("Компонента 2")
        plt.title("Візуалізація кластерів")
        plt.colorbar(label="Номер кластера")
        plt.show()
    # ------------------------------------------------------------
    # Приклади використання функцій оцінки
    # (для демонстрації створюються штучні дані; у реальному
проєкті замініть на свої змінні,
    #  отримані в результаті роботи алгоритмів)
    # ------------------------------------------------------------
    if __name__ == "__main__":
        # 1. Оцінка алгоритмів прогнозування витрат/доходів
        # Створюємо штучні дані часових рядів для демонстрації
        dates = pd.date_range(start="2025-01-01", periods=50,
freq="D")
        # Фактичні значення (наприклад, витрат)
        y_true_forecast = pd.Series(np.random.rand(50) * 100,
index=dates)
        # Прогнозовані значення – фактичні значення з доданим
шумом
        y_pred_forecast = y_true_forecast +
np.random.normal(0, 5, size=50)
        evaluate_forecasting(y_true_forecast, y_pred_forecast,
title="Оцінка прогнозування витрат")
        # 2. Оцінка алгоритму класифікації транзакцій
```

```
        # Створюємо штучні дані для класифікації (наприклад,
категорії транзакцій)
        categories = ["Продукти", "Транспорт", "Комунальні
послуги", "Розваги"]
        # Фактичні мітки класів
        y_true_class = np.random.choice(categories, size=100)
        # Прогнозовані мітки класів (для демонстрації
використаємо випадковий прогноз)
        y_pred_class = np.random.choice(categories, size=100)
        evaluate_classification(y_true_class, y_pred_class)
        # 3. Оцінка алгоритму кластеризації транзакцій
        # Створюємо штучні дані (двовимірний простір) для
кластеризації
        features_dummy, _ = make_blobs(n_samples=200,
centers=3, n_features=2, random_state=42)
        # Використовуємо алгоритм KMeans для визначення
кластерів (демонстрація)
        kmeans = KMeans(n_clusters=3, random_state=42)
        cluster_labels = kmeans.fit_predict(features_dummy)
        evaluate_clustering(features_dummy, cluster_labels)
```

ДОДАТОК Г

Програмний код алгоритму прогнозування цінових рухів на фінансових ринках за допомогою рекурентної нейронної мережі LSTM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
# Завантаження даних. Припускаємо, що CSV-файл
"price_data.csv" містить колонки: "Date" та "Close"
data = pd.read_csv("price_data.csv", parse_dates=['Date'])
data.sort_values("Date", inplace=True)
data.set_index("Date", inplace=True)
# Використовуємо закриття цін для прогнозування
prices = data['Close'].values.reshape(-1, 1)
# Нормалізація даних у діапазоні [0, 1]
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_prices = scaler.fit_transform(prices)
# Визначення параметрів: кількість попередніх днів для
прогнозу (window size)
sequence_length = 60  # використовується інформація за
останні 60 днів
# Формування послідовностей даних
X, y = [], []
for i in range(sequence_length, len(scaled_prices)):
    X.append(scaled_prices[i-sequence_length:i, 0])
    y.append(scaled_prices[i, 0])
X, y = np.array(X), np.array(y)
# Зміна розмірності X для LSTM: [samples, timesteps,
features]
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
# Розподіл даних на тренувальну та тестову
вибірки (80%/20%)
```

```
    train_size = int(len(X) * 0.8)
    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]
    # Побудова моделі LSTM
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))  # прогноз наступного значення
ціни
    model.compile(optimizer='adam', loss='mean_squared_error')
    model.summary()
    # Навчання моделі
    history = model.fit(X_train, y_train, epochs=50,
batch_size=32, validation_split=0.1)
    # Прогнозування на тестових даних
    predicted_prices = model.predict(X_test)
    predicted_prices =
scaler.inverse_transform(predicted_prices)
    y_test_actual = scaler.inverse_transform(y_test.reshape(-
1, 1))
    # Візуалізація результатів прогнозування
    plt.figure(figsize=(12, 6))
    plt.plot(data.index[-len(y_test_actual):], y_test_actual,
color='red', label='Фактичні ціни')
    plt.plot(data.index[-len(predicted_prices):],
predicted_prices, color='blue', label='Прогнозовані ціни',
linestyle='--')
    plt.title('Прогнозування цінових рухів за допомогою LSTM')
    plt.xlabel('Дата')
    plt.ylabel('Ціна')
    plt.legend()
    plt.show()
```

ДОДАТОК Д

Програмний код створення моделі для автоматизованої торгівлі за допомогою підсиленого навчання

```
import gym
import gym_anytrading  # Потрібно встановити: pip install gym-anytrading
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from stable_baselines3 import PPO  # Потрібно встановити: pip install stable-baselines3
from stable_baselines3.common.vec_env import DummyVecEnv
# Завантаження історичних даних про ціни
# Припускаємо, що CSV-файл "stock_data.csv" містить колонки: "Date" та "Close"
data = pd.read_csv("stock_data.csv", parse_dates=["Date"])
data.sort_values("Date", inplace=True)
data.set_index("Date", inplace=True)
# Налаштування параметрів для середовища торгівлі:
# - window_size: кількість попередніх днів, які використовуються як контекст
# - frame_bound: інтервал, який використовується для формування послідовності (від якого до якого індексу даних)
env = gym.make("stocks-v0", df=data, window_size=10, frame_bound=(10, len(data)))
env = DummyVecEnv([lambda: env])  # Обгортання середовища для сумісності зі stable-baselines3
# Створення та навчання торгового агента за допомогою алгоритму PPO
model = PPO("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=10000)
# Тестування навченого агента на середовищі
obs = env.reset()
done = False
```

```
    while not done:
        # Агент прогнозує дію для поточного спостереження
        action, _states = model.predict(obs)
        obs, reward, done, info = env.step(action)
        # Візуалізація поточного стану (графік з торговими
сигналами та позиціями)
        env.render()
    # Після завершення тестування візуалізуємо результати у
вигляді інтерактивного графіка
    env.render_all()
    plt.show()
```

ДОДАТОК Е

Програмний код процесу валідації та оптимізації торгових алгоритмів

```
import backtrader as bt
import datetime
# Визначення торгової стратегії на основі перетину ковзних середніх
class MovingAverageCrossStrategy(bt.Strategy):
    params = (
        ('fast_period', 10),   # період для швидкої ковзної середньої
        ('slow_period', 30),   # період для повільної ковзної середньої
    )
    def __init__(self):
        # Обчислення ковзних середніх
        self.fast_ma = bt.indicators.SimpleMovingAverage(self.data.close,
                                                         period=self.params.fast_period)
        self.slow_ma = bt.indicators.SimpleMovingAverage(self.data.close,
                                                         period=self.params.slow_period)
        # Визначення сигналу перетину
        self.crossover = bt.indicators.CrossOver(self.fast_ma, self.slow_ma)
    def next(self):
        # Якщо немає відкритої позиції та швидка ковзна перетинає повільну знизу вгору – відкриваємо позицію (купівля)
        if not self.position:
            if self.crossover > 0:
                self.buy()
        # Якщо є позиція і швидка ковзна перетинає повільну зверху вниз – закриваємо позицію
```

```
            elif self.crossover < 0:
                self.close()
    # Створення об’єкта Cerebro для управління процесом
симуляції
    cerebro = bt.Cerebro(optreturn=False)
    # Завантаження історичних даних (в даному прикладі
використовується дані Apple з Yahoo Finance)
    data = bt.feeds.YahooFinanceData(
        dataname='AAPL',
        fromdate=datetime.datetime(2024, 1, 1),
        todate=datetime.datetime(2025, 1, 1)
    )
    cerebro.adddata(data)
    # Встановлення початкового капіталу
    cerebro.broker.setcash(100000)
    # Додавання аналітиків для оцінки ефективності стратегії
    cerebro.addanalyzer(bt.analyzers.SharpeRatio,
_name='sharpe', timeframe=bt.TimeFrame.Days)
    cerebro.addanalyzer(bt.analyzers.DrawDown,
_name='drawdown')
    # Налаштування оптимізації:
    # Ми будемо перебирати параметри fast_period (5, 10, 15)
та slow_period (20, 30, 40, 50)
    cerebro.optstrategy(
        MovingAverageCrossStrategy,
        fast_period=range(5, 16, 5),      # 5, 10, 15
        slow_period=range(20, 51, 10)     # 20, 30, 40, 50
    )
    # Запуск оптимізації (за потреби можна використовувати
maxcpus>1 для прискорення)
    optimized_runs = cerebro.run(maxcpus=1)
    # Пошук найкращої комбінації параметрів на основі Sharpe
Ratio
    best_sharpe = None
    best_params = None
    for run in optimized_runs:
```

```
        # Кожен run – це список екземплярів стратегії (в
нашому випадку лише один екземпляр)
        strategy = run[0]
        sharpe_ratio =
strategy.analyzers.sharpe.get_analysis().get('sharperatio',
None)
        drawdown =
strategy.analyzers.drawdown.get_analysis().get('max', 0)
        if sharpe_ratio is not None:
            print(f"Параметри:
fast_period={strategy.params.fast_period},
slow_period={strategy.params.slow_period}, "
                  f"Sharpe Ratio: {sharpe_ratio:.2f}, Max
DrawDown: {drawdown:.2f}")
            if best_sharpe is None or sharpe_ratio >
best_sharpe:
                best_sharpe = sharpe_ratio
                best_params = (strategy.params.fast_period,
strategy.params.slow_period)
    print("\nНайкращі параметри за Sharpe Ratio:")
    print(f"fast_period = {best_params[0]}, slow_period =
{best_params[1]}, Sharpe Ratio = {best_sharpe:.2f}")
```

ДОДАТОК Є

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	95 с.
	Інші документи	
2.	Презентаційні матеріали	10 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Черкасов М.М.			Дослідження методів машинного навчання для автоматизації ведення особистих фінансів та торгівлі на фінансових ринках
Перевірив		Удовенко С.Г.			
Н.контр.		Малєєва І.А.			
Затв.		Золотухін О.В.			

Шифр групи	Код напр./спец.
СШІмзд-23-1	122
ХНУРЕ кафедра ШІ	