ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

кваліфікаційна робота

Програмні засоби виявлення мережних аномалій

Виконав:
студент гр. КІУКІ-21-4
Федько Д.А.

Керівник:
ст. викл. каф. ЕОМ,
Дяченко В.О.

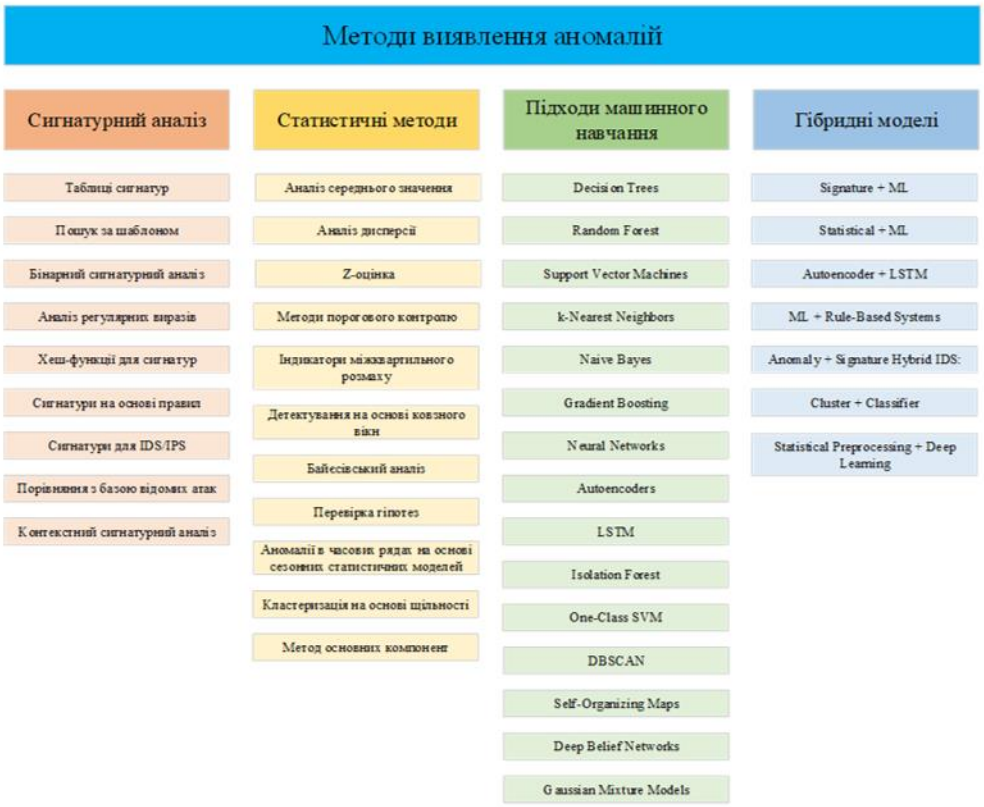
Мета роботи та завдання 2

Метою роботи є розробка, реалізація програмних засобів для виявлення мережних аномалій з використанням сучасних методів машинного навчання, що дозволяють виявляти як відомі, так і нові загрози в комп'ютерних мережах.

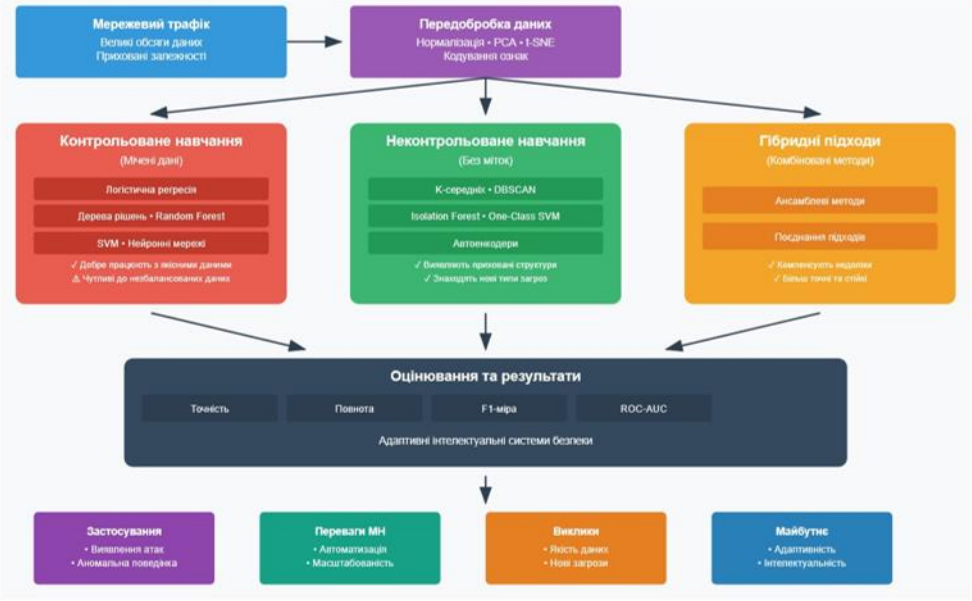
Задачі:

- провести аналіз існуючих типів мережевих атак та аномалій, які можуть загрожувати інформаційним системам;
- вивчити класифікацію та принципи роботи сучасних систем виявлення вторгнень;
- дослідити підходи до виявлення аномалій: сигнатурні, аномальні, гібридні; визначити їх переваги та обмеження;
- ознайомитися з методами машинного навчання, що застосовуються для аналізу мережевого трафіку;
- обґрунтувати вибір алгоритму машинного навчання для реалізації виявлення аномалій;
- реалізувати власну модель виявлення аномалій у середовищі Google Colab з використанням відкритого датасету;
- провести тестування розробленого рішення, оцінити його точність, ефективність і продуктивність.

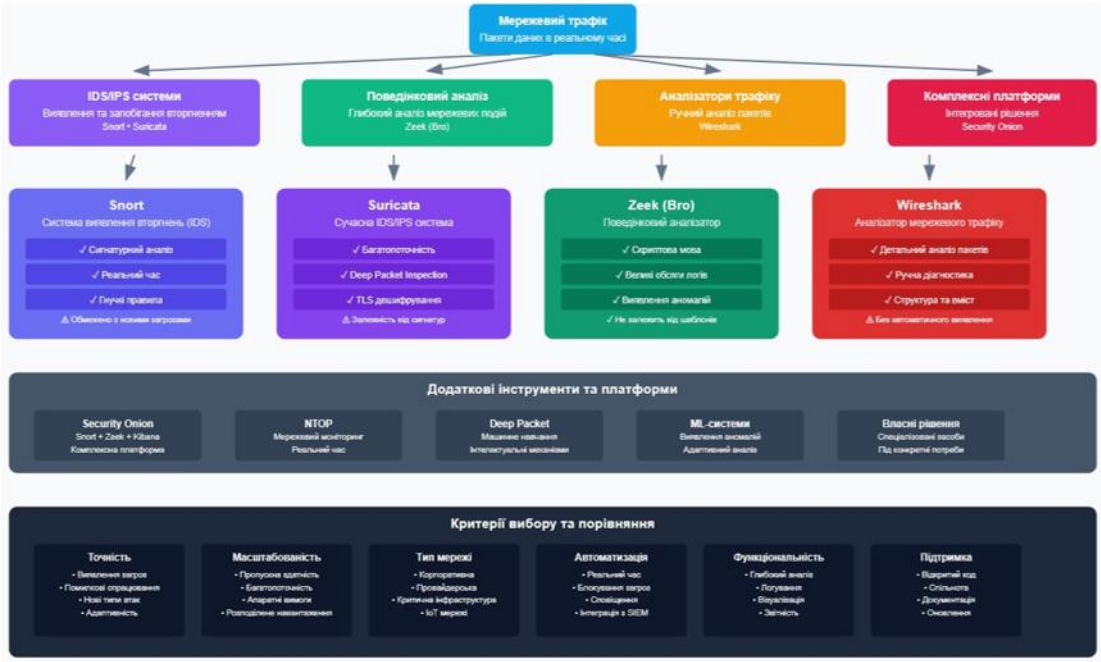
Методи виявлення аномалій 3



Методи машинного навчання для виявлення аномалій 4



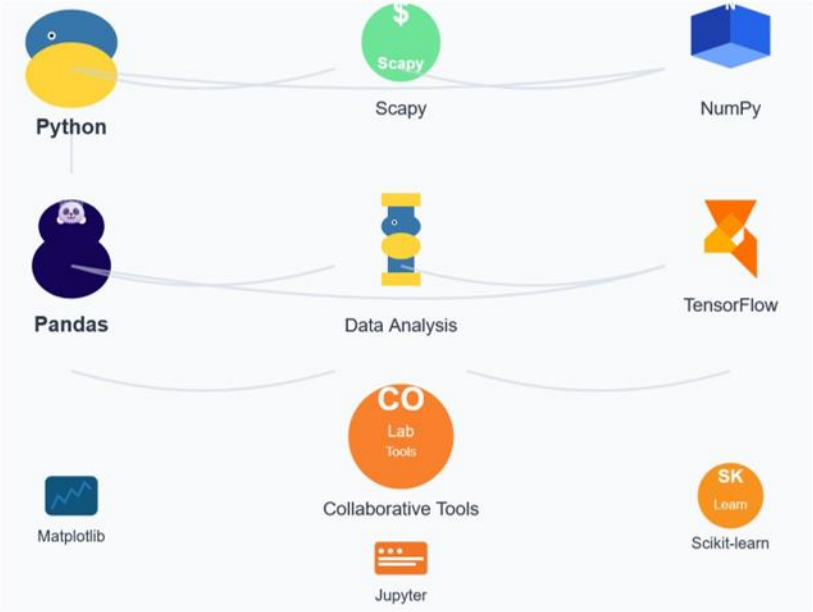
Існуючі програмні засоби виявлення аномалій в мережі 5



Етапи методу виявлення аномалій. Загальна структура 6

- **Збір вхідних даних**
- **Попередня обробка**,
- **Побудову профілю "нормальної" поведінки**
- **Реальне функціонування програмних засобів**
- **Модуль логування та моніторингу**
- **Динамічне оновлення поведінкового профілю**

Вибір програмних засобів та середовища розробки 7



Архітектура програмного забезпечення виявлення аномалій в мережі 8

Реалізація в Google Colab 9

```
# Додатковий графік: розподіл 'duration' у нормальному та аномальному трафіку
plt.figure(figsize=(8, 6))
sns.kdeplot(df[df['label'] == 0]['duration'], label='Нормальні', shade=True, color='green')
sns.kdeplot(df[df['label'] == 1]['duration'], label='Аномалії', shade=True, color='red')
plt.title("Розподіл тривалості з'єднань")
plt.xlabel("Тривалість")
plt.ylabel("Щільність")
plt.legend()
plt.grid(True)
plt.show()
```

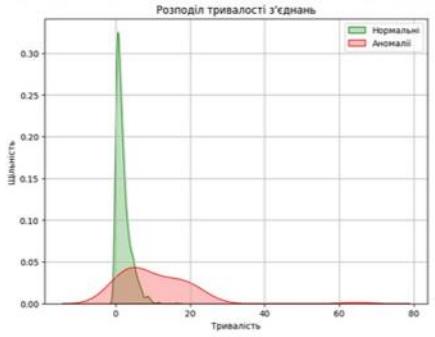
```
/tmp/ipython-input-10-3534474050.py:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[df['label'] == 0]['duration'], label='Нормальні', shade=True, color='green')
/tmp/ipython-input-10-3534474050.py:4: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[df['label'] == 1]['duration'], label='Аномалії', shade=True, color='red')
```



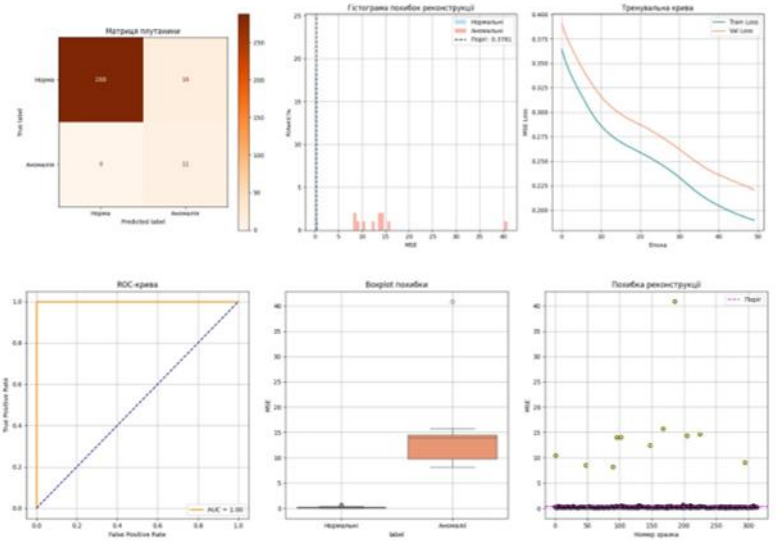

```
[11] # Додатковий графік: парна діаграма для вибраних ознак
sns.pairplot(df.sample(200), hue='label', vars=['src_bytes', 'dst_bytes', 'count', 'same_srv_rate'], palet
plt.suptitle("Парна діаграма вибраних параметрів", y=1.02)
plt.show()
```


```
output_layer = Dense(input_dim, activation='linear')(decoded)
autoencoder = Model(inputs=input_layer, outputs=output_layer)
autoencoder.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
history = autoencoder.fit(X_train_norm, X_train_norm, epochs=50, batch_size=
                          shuffle=True, validation_split=0.1, verbose=1)
```

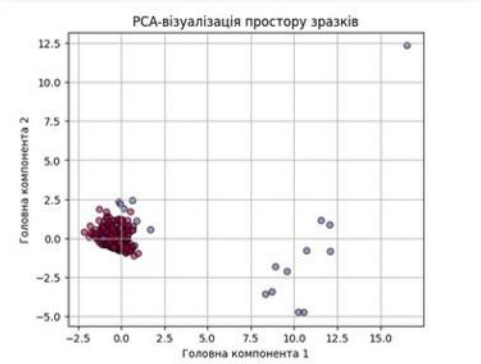
```
Epoch 1/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 42ms/step - loss: 0.3595 - val_loss: 0.3904
Epoch 2/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.3587 - val_loss: 0.3793
Epoch 3/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 0.3368 - val_loss: 0.3700
Epoch 4/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.3313 - val_loss: 0.3618
Epoch 5/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.3243 - val_loss: 0.3539
Epoch 6/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 0.3150 - val_loss: 0.3466
Epoch 7/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 0.3038 - val_loss: 0.3397
Epoch 8/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.3130 - val_loss: 0.3332
Epoch 9/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.3026 - val_loss: 0.3270
Epoch 10/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2982 - val_loss: 0.3215
Epoch 11/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.2892 - val_loss: 0.3163
Epoch 12/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.2808 - val_loss: 0.3117
Epoch 13/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.2759 - val_loss: 0.3079
Epoch 14/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.2797 - val_loss: 0.3042
Epoch 15/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2677 - val_loss: 0.3010
Epoch 16/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2725 - val_loss: 0.2982
Epoch 17/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2739 - val_loss: 0.2955
Epoch 18/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.2684 - val_loss: 0.2932
Epoch 19/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2661 - val_loss: 0.2915
Epoch 20/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2631 - val_loss: 0.2892
Epoch 21/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.2548 - val_loss: 0.2872
Epoch 22/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.2598 - val_loss: 0.2854
Epoch 23/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2486 - val_loss: 0.2833
Epoch 24/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2592 - val_loss: 0.2807
Epoch 25/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.2467 - val_loss: 0.2785
Epoch 26/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2501 - val_loss: 0.2758
Epoch 27/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2436 - val_loss: 0.2733
Epoch 28/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2381 - val_loss: 0.2707
Epoch 29/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.2436 - val_loss: 0.2680
```

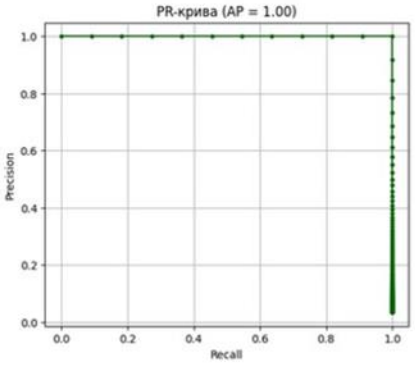
```
[5] X_test_pred = autoencoder.predict(X_test)
mse = np.mean(np.square(X_test - X_test_pred), axis=1)
threshold = np.percentile(mse[y_test == 0], 95)
predictions = (mse > threshold).astype(int)
fpr, tpr, _ = roc_curve(y_test, mse)
roc_auc = auc(fpr, tpr)
error_df = pd.DataFrame({'MSE': mse, 'label': y_test})
```

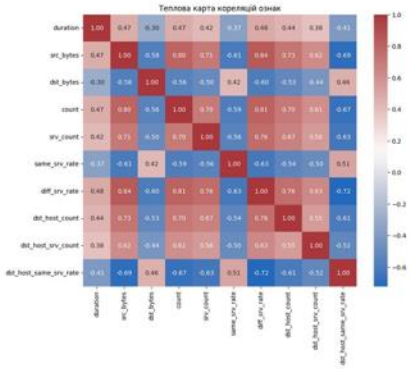
Аналіз результатів 10



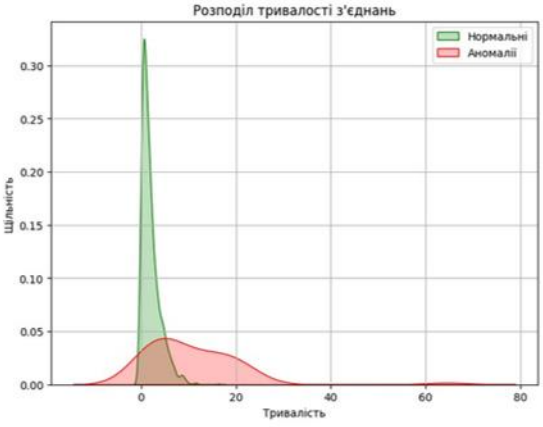
Результати роботи 11

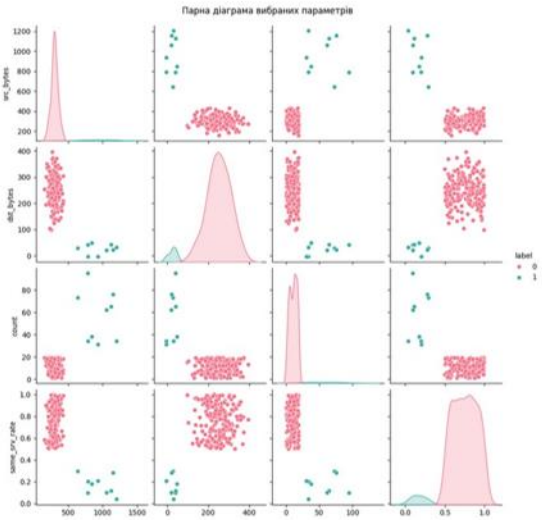






Результати роботи 12





13 Висновки

У результаті виконання кваліфікаційної роботи розроблено програмні засоби для виявлення мережевих аномалій у корпоративному середовищі з використанням методів машинного навчання. У межах роботи було здійснено глибокий теоретичний аналіз підходів до автоматизованої детекції аномалій, зокрема методів класифікації, кластеризації, гібридних алгоритмів та моделей глибокого навчання. На основі порівняльного аналізу обґрунтовано доцільність використання автоенкодерної архітектури з можливістю розширення рекурентними LSTM-шарами для врахування тимчасової динаміки трафіку.

Було реалізовано програмний прототип, що охоплює повний цикл обробки мережевих даних: від збору, очищення та нормалізації трафіку – до його подачі в нейронну модель, аналізу результатів реконструкції та автоматичного прийняття рішень про наявність аномалій. Розробку здійснено мовою Python із використанням бібліотек Scapy, Pandas, NumPy, Scikit-learn, TensorFlow, Loguru, що забезпечило масштабованість, гнучкість та можливість інтеграції з реальними корпоративними системами.

Експериментальна частина була реалізована в середовищі Google Colab, що дозволило швидко здійснювати тестування моделі на синтетично сформованому, але структурно наближеному до реального, наборі даних на основі NSL-KDD. У ході перевірки було підтверджено здатність моделі з високою точністю розрізняти нормальні та аномальні зразки, що підтверджується як числовими метриками, так і графічним аналізом.

Результати демонструють, що навіть за умов обмеженого навчального корпусу та високої варіативності трафіку, автоенкодерна модель здатна до узагальнення, стабільно ідентифікує відхилення, має високу чутливість до латентних змін та низький рівень хибних спрацювань. Таким чином, розроблені програмні засоби довели свою практичну придатність як гнучке, адаптивне та надійне рішення для автоматизованого моніторингу мережевої безпеки. Його можлива інтеграція у більш складні інфраструктури відкриває перспективи подальшого розширення функціональності – зокрема, за рахунок донавчання на нових даних, підтримки потокового аналізу та інтеграції з інструментами реагування на інциденти.

ДОДАТОК Б

Програмний код

Б.1 Лістинг коду

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, confusion_matrix,
ConfusionMatrixDisplay
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.metrics import precision_recall_curve,
average_precision_score
def generate_data():
    np.random.seed(42)
    n_normal = 1000
    n_anomalous = 50
    normal_data = {
        "duration": np.random.exponential(2.0, n_normal),
        "src_bytes": np.random.normal(300, 50, n_normal),
        "dst_bytes": np.random.normal(250, 60, n_normal),
        "count": np.random.randint(1, 20, n_normal),
        "srv_count": np.random.randint(1, 15, n_normal),
        "same_srv_rate": np.random.uniform(0.5, 1.0, n_normal),
        "diff_srv_rate": np.random.uniform(0.0, 0.2, n_normal),
        "dst_host_count": np.random.randint(10, 100, n_normal),
        "dst_host_srv_count": np.random.randint(5, 100,
n_normal),
        "dst_host_same_srv_rate": np.random.uniform(0.6, 1.0,
n_normal),
        "label": [0]*n_normal
    }
    anomalous_data = {
        "duration": np.random.exponential(10.0, n_anomalous),
        "src_bytes": np.random.normal(1000, 200, n_anomalous),
        "dst_bytes": np.random.normal(30, 15, n_anomalous),
        "count": np.random.randint(30, 100, n_anomalous),
        "srv_count": np.random.randint(20, 60, n_anomalous),
        "same_srv_rate": np.random.uniform(0.0, 0.3,
n_anomalous),
        "diff_srv_rate": np.random.uniform(0.7, 1.0,
```

```
n_anomalous),
        "dst_host_count": np.random.randint(200, 255,
n_anomalous),
        "dst_host_srv_count": np.random.randint(100, 255,
n_anomalous),
        "dst_host_same_srv_rate": np.random.uniform(0.0, 0.4,
n_anomalous),
        "label": [1]*n_anomalous
    }
    df = pd.concat([pd.DataFrame(normal_data),
pd.DataFrame(anomalous_data)], ignore_index=True)
    return df.sample(frac=1.0,
random_state=42).reset_index(drop=True)
df = generate_data()
X = df.drop(columns=["label"]).values
y = df["label"].values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_
input_dim = X_train.shape[1]
input_layer = Input(shape=(input_dim,))
encoded = Dense(8, activation='relu')(input_layer)
encoded = Dense(4, activation='relu')(encoded)
decoded = Dense(8, activation='relu')(encoded)
output_layer = Dense(input_dim, activation='linear')(decoded)
autoencoder = Model(inputs=input_layer, outputs=output_layer)
autoencoder.compile(optimizer=Adam(learning_rate=0.001),
loss='mse')
history = autoencoder.fit(X_train_norm, X_train_norm, epochs=50,
batch_size=32,
                          shuffle=True, validation_split=0.1,
verbose=1)
X_test_pred = autoencoder.predict(X_test)
mse = np.mean(np.square(X_test - X_test_pred), axis=1)
threshold = np.percentile(mse[y_test == 0], 95)
predictions = (mse > threshold).astype(int)
fpr, tpr, _ = roc_curve(y_test, mse)
roc_auc = auc(fpr, tpr)
error_df = pd.DataFrame({'MSE': mse, 'label': y_test})
plt.figure(figsize=(18, 12))
plt.subplot(2, 3, 1)
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC =
{roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend()
plt.grid(True)

plt.subplot(2, 3, 2)
sns.boxplot(x='label', y='MSE', data=error_df, palette="Set2")
plt.xticks([0, 1], ['Нормальні', 'Аномалії'])
```

```
plt.title("Boxplot похибки")
plt.grid(True)

plt.subplot(2, 3, 3)
plt.scatter(range(len(mse)), mse, c=y_test, cmap='viridis',
edgecolors='k')
plt.axhline(y=threshold, color='magenta', linestyle='--',
label='Поріг')
plt.title("Похибка реконструкції")
plt.xlabel("Номер зразка")
plt.ylabel("MSE")
plt.legend()
plt.grid(True)

plt.subplot(2, 3, 4)
cm = confusion_matrix(y_test, predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=["Норма", "Аномалія"])
disp.plot(ax=plt.gca(), cmap='Oranges', values_format='d')
plt.title("Матриця плутанини")
plt.grid(False)

plt.subplot(2, 3, 5)
plt.hist(mse[y_test == 0], bins=50, alpha=0.6,
label="Нормальні", color='skyblue')
plt.hist(mse[y_test == 1], bins=50, alpha=0.6,
label="Аномальні", color='salmon')
plt.axvline(threshold, color='black', linestyle='--',
label=f'Поріг: {threshold:.4f}')
plt.title("Гістограма похибок реконструкції")
plt.xlabel("MSE")
plt.ylabel("Кількість")
plt.legend()
plt.grid(True)

plt.subplot(2, 3, 6)
plt.plot(history.history['loss'], label='Train Loss',
color='teal')
plt.plot(history.history['val_loss'], label='Val Loss',
color='coral')
plt.title('Тренувальна крива')
plt.xlabel('Епоха')
plt.ylabel('MSE Loss')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_test)
plt.figure(figsize=(6, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=predictions,
cmap='Spectral', alpha=0.6, edgecolors='k')
plt.title("PCA-візуалізація простору зразків")
```

```
plt.xlabel("Головна компонента 1")
plt.ylabel("Головна компонента 2")
plt.grid(True)
plt.show()
precision, recall, _ = precision_recall_curve(y_test, mse)
avg_precision = average_precision_score(y_test, mse)
plt.figure(figsize=(6, 5))
plt.plot(recall, precision, marker='.', color='darkgreen')
plt.title(f'PR-крива (AP = {avg_precision:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid(True)
plt.show()
plt.figure(figsize=(10, 8))
sns.heatmap(pd.DataFrame(X, columns=df.columns[:-1]).corr(),
annot=True, fmt=".2f", cmap="vlag")
plt.title("Теплова карта кореляцій ознак")
plt.show()
# Графік: розподіл 'duration' у нормальному та аномальному
трафіку
plt.figure(figsize=(8, 6))
sns.kdeplot(df[df['label'] == 0]['duration'], label='Нормальні',
shade=True, color='green')
sns.kdeplot(df[df['label'] == 1]['duration'], label='Аномалії',
shade=True, color='red')
plt.title("Розподіл тривалості з'єднань")
plt.xlabel("Тривалість")
plt.ylabel("Щільність")
plt.legend()
plt.grid(True)
plt.show()
# Графік: парна діаграма для вибраних ознак
sns.pairplot(df.sample(200), hue='label', vars=['src_bytes',
'dst_bytes', 'count', 'same_srv_rate'], palette='husl')
plt.suptitle("Парна діаграма вибраних параметрів", y=1.02)
plt.show()
```