This document provides a detailed description of the technical steps, tools, methods, and scripts used to perform time series analysis for fraud detection using the IEEE-CIS Fraud Detection dataset.

**1. Objective**

To analyze temporal patterns in fraudulent transactions and develop statistical measures for anomaly detection, leveraging SQL and Python for:

* Identifying seasonal trends and patterns.
* Creating moving averages and visualizing temporal dynamics.
* Detecting anomalies in transaction behavior.

**2. Tools and Technologies**

**2.1 Python**

* **Libraries Used**:
  + **pandas**: For data manipulation and analysis.
  + **matplotlib/seaborn/plotly**: For data visualization.
  + **statsmodels**: For time series decomposition.
  + **scikit-learn**: For anomaly detection using machine learning models.
  + **scipy**: For statistical anomaly detection.

**2.2 SQL**

* SQL queries were used for moving averages, anomaly detection, and aggregations.
* Compatible with:
  + **SQLite**: For local testing.
  + **PostgreSQL/MySQL**: For production-grade analysis.

**3. Data Preparation**

**3.1 Load the Dataset**

The dataset is loaded into a pandas DataFrame:

import pandas as pd

df = pd.read\_csv('train\_transaction.csv')

**3.2 Extract Temporal Features**

Transform the TransactionDT column into readable date and time features:

df['TransactionDate'] = pd.to\_datetime('2017-12-01') + pd.to\_timedelta(df['TransactionDT'], unit='s')

df['Date'] = df['TransactionDate'].dt.date

df['Hour'] = df['TransactionDate'].dt.hour

df['DayOfWeek'] = df['TransactionDate'].dt.dayofweek

**4. Temporal Analysis**

**4.1 Aggregation**

Group data to analyze trends in fraudulent (isFraud=1) and non-fraudulent (isFraud=0) transactions:

fraud\_trends = df[df['isFraud'] == 1].groupby('Date')['TransactionAmt'].sum()

non\_fraud\_trends = df[df['isFraud'] == 0].groupby('Date')['TransactionAmt'].sum()

**4.2 Visualization**

Visualize daily trends:

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.plot(fraud\_trends.index, fraud\_trends.values, label='Fraudulent Transactions')

plt.plot(non\_fraud\_trends.index, non\_fraud\_trends.values, label='Non-Fraudulent Transactions')

plt.title('Transaction Trends Over Time')

plt.xlabel('Date')

plt.ylabel('Transaction Amount')

plt.legend()

plt.show()

**5. Moving Averages and Seasonal Trends**

**5.1 Moving Averages**

Smooth time series data using rolling averages:

df['7\_day\_avg'] = fraud\_trends.rolling(window=7).mean()

**5.2 Seasonal Decomposition**

Decompose time series data into trend, seasonal, and residual components:

from statsmodels.tsa.seasonal import seasonal\_decompose

decomposition = seasonal\_decompose(fraud\_trends, model='additive', period=30)

decomposition.plot()

plt.show()

**6. Anomaly Detection**

**6.1 Z-Score Based Detection**

Detect anomalies using z-scores:

from scipy.stats import zscore

fraud\_trends\_z = pd.DataFrame(fraud\_trends)

fraud\_trends\_z['z\_score'] = zscore(fraud\_trends)

anomalies = fraud\_trends\_z[fraud\_trends\_z['z\_score'] > 3] # Threshold for anomaly

**6.2 Isolation Forest**

Detect anomalies using machine learning:

from sklearn.ensemble import IsolationForest

model = IsolationForest(contamination=0.01, random\_state=42)

fraud\_trends\_z['anomaly'] = model.fit\_predict(fraud\_trends.values.reshape(-1, 1))

**7. SQL Analysis**

**7.1 Moving Average**

Calculate moving averages using SQL

SELECT

TransactionDate,

SUM(CASE WHEN isFraud = 1 THEN TransactionAmt ELSE 0 END) AS FraudAmount,

AVG(SUM(CASE WHEN isFraud = 1 THEN TransactionAmt ELSE 0 END))

OVER (ORDER BY TransactionDate ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS MovingAvg

FROM Transactions

GROUP BY TransactionDate;

**7.2 Anomaly Detection**

Identify anomalies using standard deviation:

SELECT

TransactionDate,

SUM(CASE WHEN isFraud = 1 THEN TransactionAmt ELSE 0 END) AS FraudAmount,

AVG(SUM(CASE WHEN isFraud = 1 THEN TransactionAmt ELSE 0 END))

OVER () AS AvgFraud,

STDDEV(SUM(CASE WHEN isFraud = 1 THEN TransactionAmt ELSE 0 END))

OVER () AS StdDevFraud,

CASE

WHEN SUM(CASE WHEN isFraud = 1 THEN TransactionAmt ELSE 0 END) >

AVG(SUM(CASE WHEN isFraud = 1 THEN TransactionAmt ELSE 0 END)) + 3 \*

STDDEV(SUM(CASE WHEN isFraud = 1 THEN TransactionAmt ELSE 0 END))

THEN 'Anomaly'

ELSE 'Normal'

END AS AnomalyFlag

FROM Transactions

GROUP BY TransactionDate;

**8. Visualization**

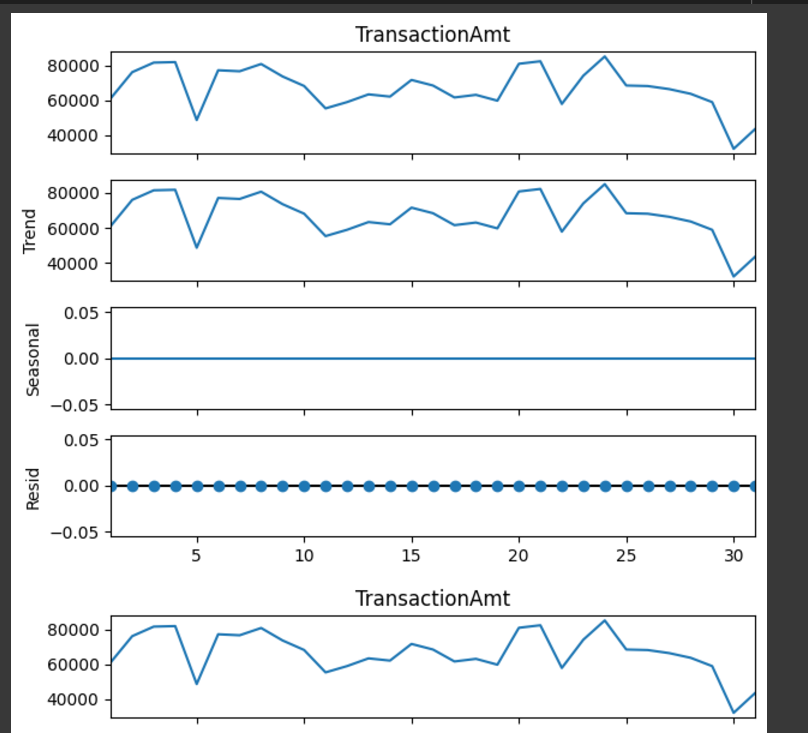
**8.1 Interactive Plots**

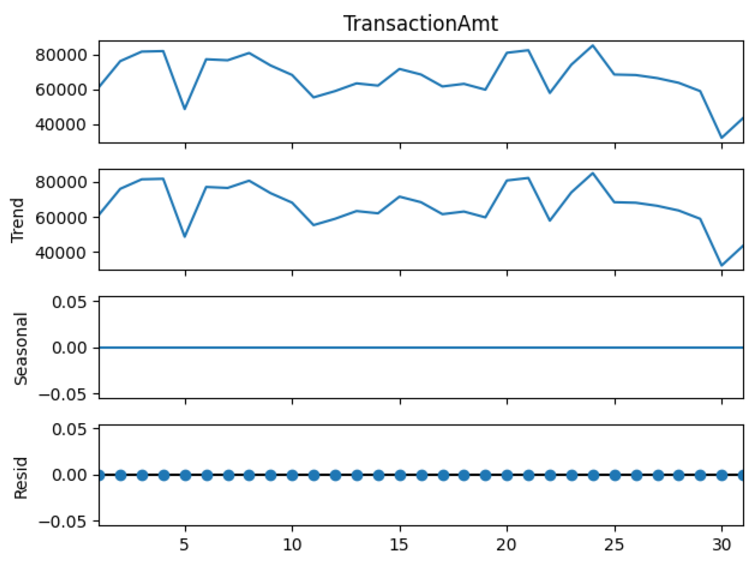
Use Plotly for interactive visualizations:

import plotly.express as px

fig = px.line(fraud\_trends, title='Fraudulent Transactions Over Time')

fig.show()





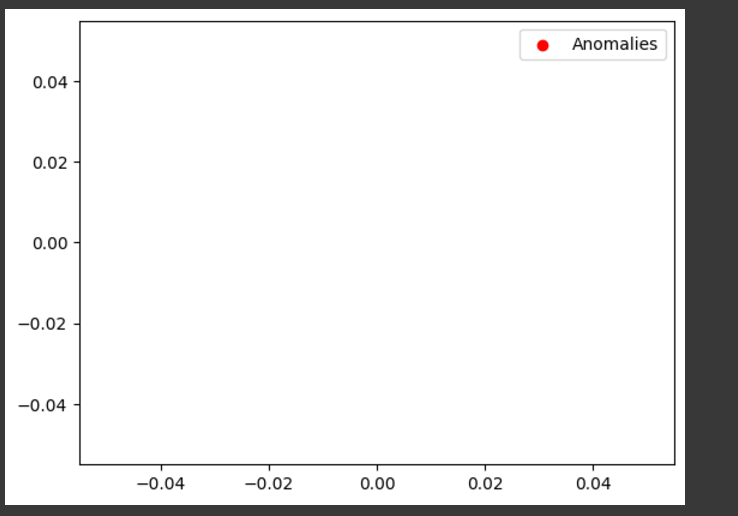
**8.2 Highlight Anomalies**

Overlay anomalies on trends:

plt.scatter(anomalies.index, anomalies['TransactionAmt'], color='red', label='Anomalies')

plt.legend()

plt.show()

****

**9. Report Creation**

**9.1 Dashboard**

* Use Tableau or Power BI to create an interactive dashboard that showcases:
  + Trends in fraud and non-fraud transactions.
  + Highlighted anomalies.
  + Seasonal decomposition results.