Anomaly Detection & Time Series Assignment

Question - 1: What is Anomaly Detection? Explain its types (point, contextual, and collective anomalies) with examples.

Answer:

Anomaly Detection (or Outlier Detection) is a method that detects data points, events, or observations that differ considerably from the majority of the data. These atypical cases tend to represent significant events like fraud, network intrusions, machine failures, or unusual diseases.

Types of Anomalies
Anomalies are primarily of three types:
1. Point Anomalies
2. Contextual Anomalies
3. Collective Anomalies

1. Point Anomalies
A point anomaly is when one data point is greatly different from the rest of data.

Example:
- Suppose in a data of daily credit card transactions, all transactions are less than ₹10,000 and suddenly one transaction is ₹1,00,000, then that transaction is a point anomaly.
- In a city's temperature readings, if the mean is 30°C and a reading is 55°C, it's a point anomaly.

Use cases:
- Credit card fraud detection
- Sensor data monitoring
- Intrusion detection

2. Contextual Anomalies (Conditional Anomalies)
A contextual anomaly is an anomalous data point that is anomalous only in a given context, but not in others.
Context, in this case, is time, location, or season — any condition that determines what's "normal" for that context.

Example
- An ordinary summer temperature is 30°C, but an unusual winter temperature.
- There can be an unexpected surge in website traffic at 3 AM on a regular day, but not during a large sale online.

Use cases:
- Time-series data analysis
- Analysis of weather patterns
- Monitoring stock market trends

3. Collective Anomalies
A collective anomaly is when a set of data points collectively exhibits abnormal behavior, even if single points might appear normal.

Example:
- In network traffic, repeated small but quick data movements may signal a cyberattack — it's the sequence, not an isolated instance, that's suspicious.
- An unexpected group of hospital patients with similar unusual symptoms may signal an epidemic.

Use cases:
- Network intrusion detection
- System fault detection
- Pattern detection in healthcare

Question 2: Compare Isolation Forest, DBSCAN, and Local Outlier Factor in terms of their approach and suitable use cases.

Answer :

All three are well-known Anomaly Detection algorithms, but they are different in approach, assumptions, and applications.

1. Isolation Forest

Approach:

- Based on the concept that anomalies are more isolatable than regular points.
- Constructs numerous random decision trees (isolation trees).
- Points which get separated quickly (in less splits) are anomalies.

Key Characteristics:

- Good on high-dimensional data.
- Efficient and scalable on large datasets.
- Does not need distance or density calculations.

Use Cases:

- Credit card fraud analysis
- Intrusion detection in networks
- Anomaly detection in high-dimensional data such as logs or transactions

2. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Approach:

- A density-based clustering algorithm.
- Clusters close points (dense areas).
- Points that are not part of any dense cluster are considered to be outliers.

Key Characteristics:

- Identifies clusters of any shape.
- Puts to best use low-dimensional spatial data.
- Sensitive to parameters eps (radius) and min_samples (minimum points per cluster).

Use Cases:-

- Geographical data analysis
- Spatial outlier detection (e.g., earthquakes, traffic accidents)
- Image segmentation

3. Local Outlier Factor (LOF)

Approach :

- A density-based approach that checks the local density of a point against the density of its neighbors.
- Points with significantly lower local density compared to neighbors are identified as outliers.

Key Characteristics:

- Identifies local anomalies well (helpful when data density is not uniform).
- Based on k-nearest neighbors (KNN) idea.
- Not suited for extremely high-dimensional data because of distance problems.

Use Cases:

- Fraud detection with local clusters
- Rare event detection in sensor networks
- Local anomaly detection in blended datasets

Question 3: What are the key components of a Time Series? Explain each with one example.

Answer :

A Time Series is a series of data points that were gathered or recorded at continuous time intervals (for example, hourly, daily, monthly).

The key components of a time series are:

1. Trend

- The general direction or movement of the data over the long term.

- May be rising, falling, or level.

Example:

The rising annual sales of a company over 5 years indicate an upward trend.

2. Seasonality

- Periodic and recurring patterns that repeat at regular intervals (e.g., every day, every month, every year).

- Due to seasonal reasons such as weather, holidays, or events.

Example:

Ice cream sales during summer months increase and decrease during winter months → seasonal pattern.

3. Cyclic Component

- Variations around the trend that are of varying duration.

- Sometimes connected to economic or business cycles (e.g., booms and recessions).

Example:

Stock market patterns that experience growth and decline periods for a number of years.

4. Irregular (Random) Component

- Random or unforeseen variation generated by sudden events.

- Cannot be explained by trend, seasonality, or cycles.

Example:

Unpredictable decrease in air travel caused by pandemic or natural disaster.

Question 4: Define Stationary in time series. How can you test and transform a non-stationary series into a stationary one?

Answer :

A stationary time series is one whose statistical properties — such as mean, variance, and autocorrelation — remain constant over time.
In other words, the series does not depend on time , its behavior is consistent throughout.

How to Test for Stationarity

There are two main ways to test it:

1. Visual Inspection

Plot the time series and observe:

- If the mean and variance appear stable → likely stationary.
- If there's a trend or seasonality → likely non-stationary.

2. Statistical Tests

a) Augmented Dickey-Fuller (ADF) Test

- Null Hypothesis (H_0): Series is *non-stationary*.
- Alternative Hypothesis (H_1): Series is *stationary*.
- If p-value < 0.05, reject H_0 → series is stationary.

b) KPSS (Kwiatkowski–Phillips–Schmidt–Shin) Test

- Null Hypothesis (H_0): Series is *stationary*.
- Alternative Hypothesis (H_1): Series is *non-stationary*.
- If p-value < 0.05, reject H_0 → series is non-stationary.

How to Transform a Non-Stationary Series into a Stationary One

If a series is non-stationary, you can make it stationary using **transformations**:

1. Differencing

- Subtract the previous observation from the current one.

$$Y_t' = Y_t - Y_{t-1}$$

- Removes **trend** and sometimes **seasonality**.

Example:
If monthly sales show a continuous increase, differencing will flatten the upward trend.

2. Log Transformation

- Reduces **variance** and **stabilizes fluctuations**.

$$Y_t' = \log(Y_t)$$

Example:
If data values grow exponentially, applying log helps make it more stable.

3. Seasonal Differencing

- Subtract the value from the same season in the previous cycle.
$$Y_t' = Y_t - Y_{t-s}$$
where s = seasonal period (e.g., 12 for monthly data).

Example:
Subtract last year's January sales from this year's January sales to remove yearly seasonality.

4. Detrending

- Remove the **trend component** using regression or decomposition.

Example:
Fit a linear trend line $Y_t = a + bt$and use the residuals as the stationary series.

Question 5: Differentiate between AR, MA, ARIMA, SARIMA, and SARIMAX models in terms of structure and application.

Answer:

Time series forecasting models assist in forecasting future values using historical data. The most popular among these models is AR, MA, ARIMA, SARIMA, and SARIMAX. All of them are of different structures and best suited for time series data of particular types based on whether it is stationary, seasonal, or impacted by exogenous factors.

1. Autoregressive Model (AR)

The Autoregressive (AR) model estimates future values based on the past data of the same series. It considers that the present value is linearly dependent upon its own past values. The model models the link between an observation and a fixed number of lagged observations.

Application:

The AR model is appropriate for time series data that exhibits obvious reliance on its past values but no seasonality. Stock prices in the stock market, temperature records, or economic measures over time are some such examples.

2. Moving Average Model (MA)

The Moving Average (MA) model forecasts the present value of the series using historical errors or shocks instead of historical observations. It assumes that present results are affected by random disturbances that happened at previous time steps.

Application :

The MA model is most suitable for stationary time series in which random short-run shocks impact the data. It is appropriate for short-run forecasting, e.g., forecasting each day's demand or short-run sales patterns.

3. ARIMA Model (Autoregressive Integrated Moving Average)

ARIMA model takes both the AR and MA ideas and incorporates the concept of Integration, that is, differencing the data to eliminate trends so that it becomes stationary. This makes ARIMA appropriate for non-stationary data where patterns shift with time. The model captures both the dependence on past values and the influence of errors at past times.

Application:

ARIMA is commonly applied for non-seasonal prediction, e.g., forecasting monthly sales, economic growth rate, or population growth after detrending the data.

4. SARIMA Model (Seasonal ARIMA)

SARIMA is a variation of ARIMA that accommodates data with a seasonal trend. It has extra parameters to fit the seasonal and non-seasonal components of the data. SARIMA is therefore able to pick up periodic, recurring patterns that take place within fixed periods, like monthly or quarterly trends.

Application :

SARIMA is ideal for time series that have clear seasonal effects, such as predicting electricity consumption, rainfall levels, or product sales that increase every holiday season.

5. SARIMAX Model (Seasonal ARIMA with Exogenous Variables)

The SARIMAX model also expands SARIMA with the addition of external or explanatory variables (a.k.a. exogenous variables) that can affect the target series. The extra inputs enable the model to make more precise predictions if external factors heavily influence the data.

Application:

SARIMAX is employed when time series data is influenced by other variables, such as predicting ice cream sales using both seasonal trends and temperature, or forecasting airline passenger volumes while accounting for holidays or promotions.

.Dataset:

- **NYC Taxi Fare Data**
- **AirPassengers Dataset**

Question 6: Load a time series dataset (e.g., AirPassengers), plot the original series, and decompose it into trend, seasonality, and residual components.

ANSWER :

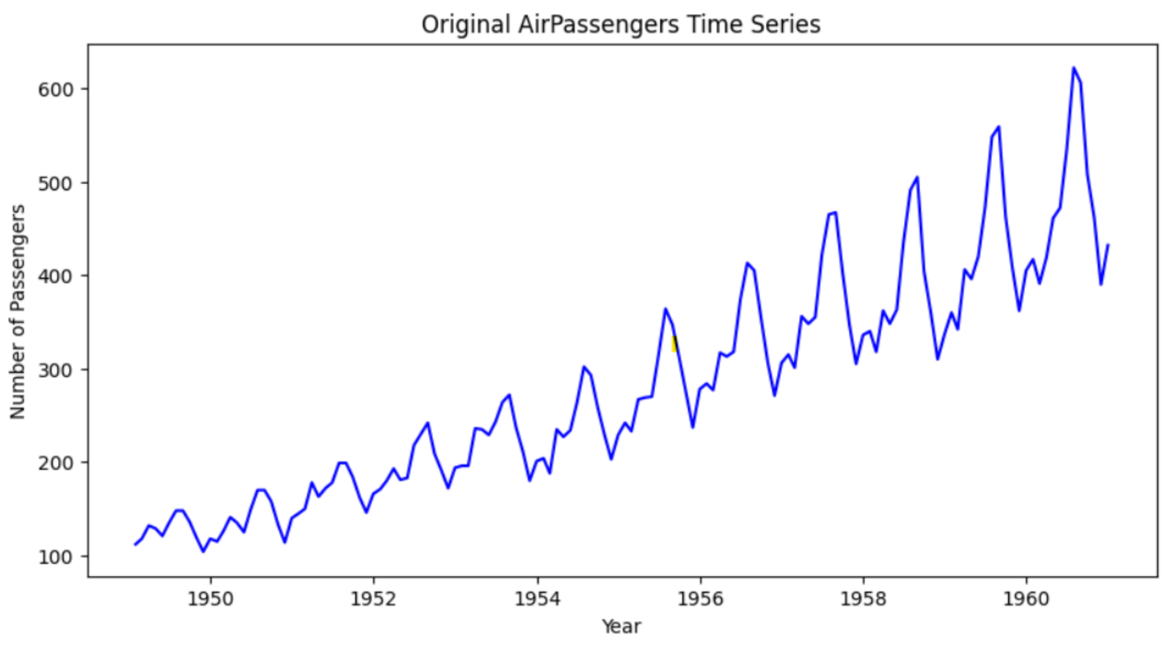
CODE :

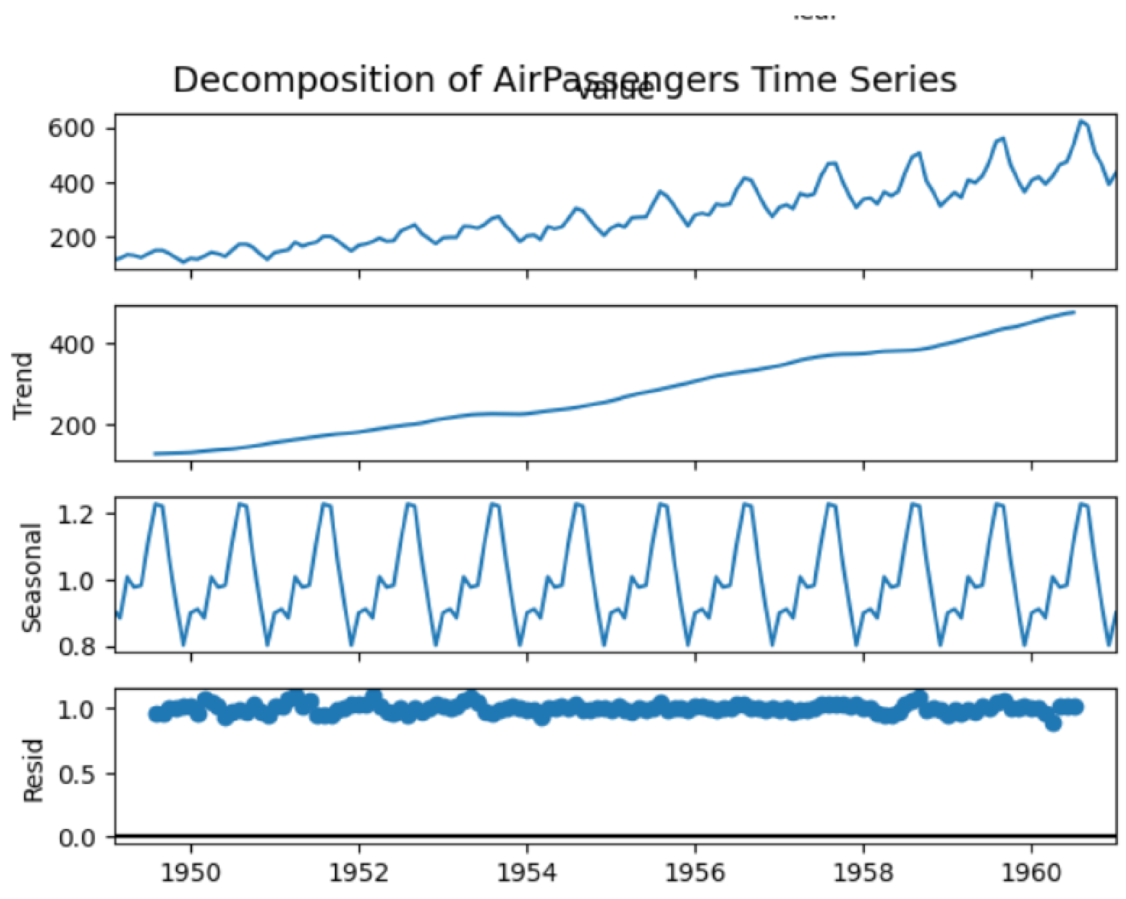
```
# Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.datasets import get_rdataset

# Load the AirPassengers dataset
data = get_rdataset("AirPassengers").data

# Display first few rows
print(data.head())

# The dataset is already a time series indexed by month
# Convert to datetime format if necessary
data['time'] = pd.date_range(start='1949-01', periods=len(data), freq='M')
data.set_index('time', inplace=True)

# Plot the original time series
plt.figure(figsize=(10, 5))
plt.plot(data['value'], color='blue')
plt.title("Original AirPassengers Time Series")
plt.xlabel("Year")
plt.ylabel("Number of Passengers")
plt.show()
```

```
# Decompose the time series
decomposition = seasonal_decompose(data['value'], model='multiplicative', period=12)

# Plot the decomposed components
decomposition.plot()
plt.suptitle("Decomposition of AirPassengers Time Series", fontsize=14)
plt.show()
```

OUTPUT :

```
         time  value
0  1949.000000    112
1  1949.083333    118
2  1949.166667    132
3  1949.250000    129
4  1949.333333    121
```



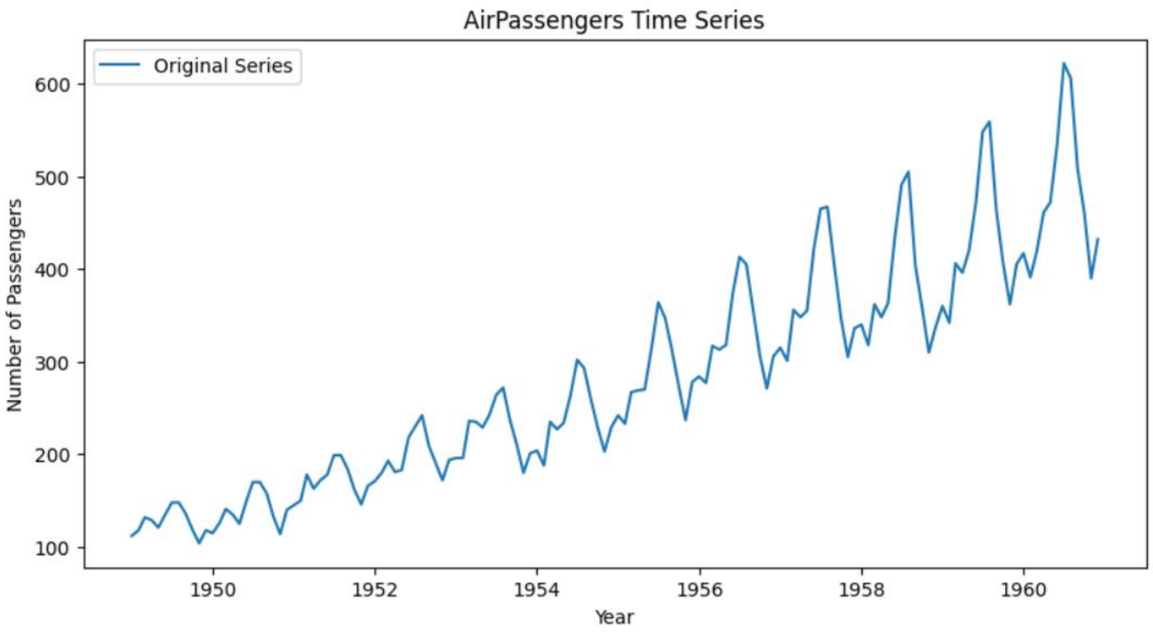


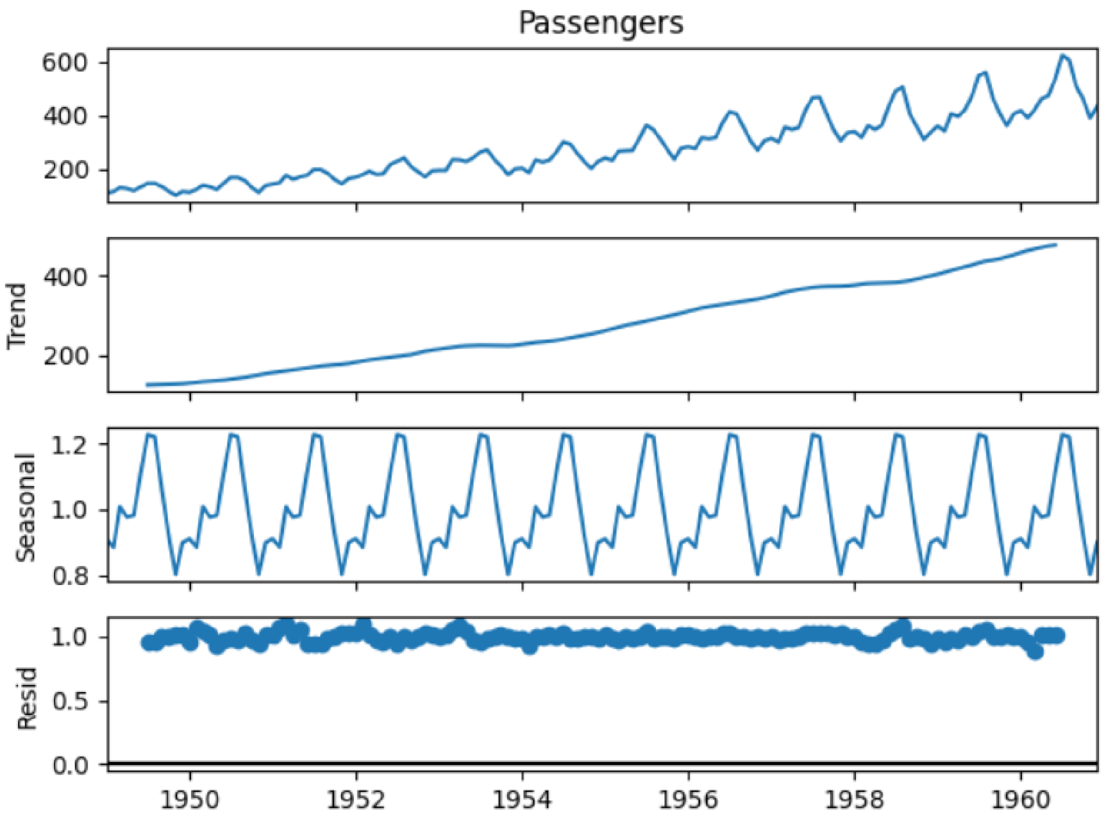
Question 7: Apply Isolation Forest on a numerical dataset (e.g., NYC Taxi Fare) to detect anomalies. Visualize the anomalies on a 2D scatter plot.

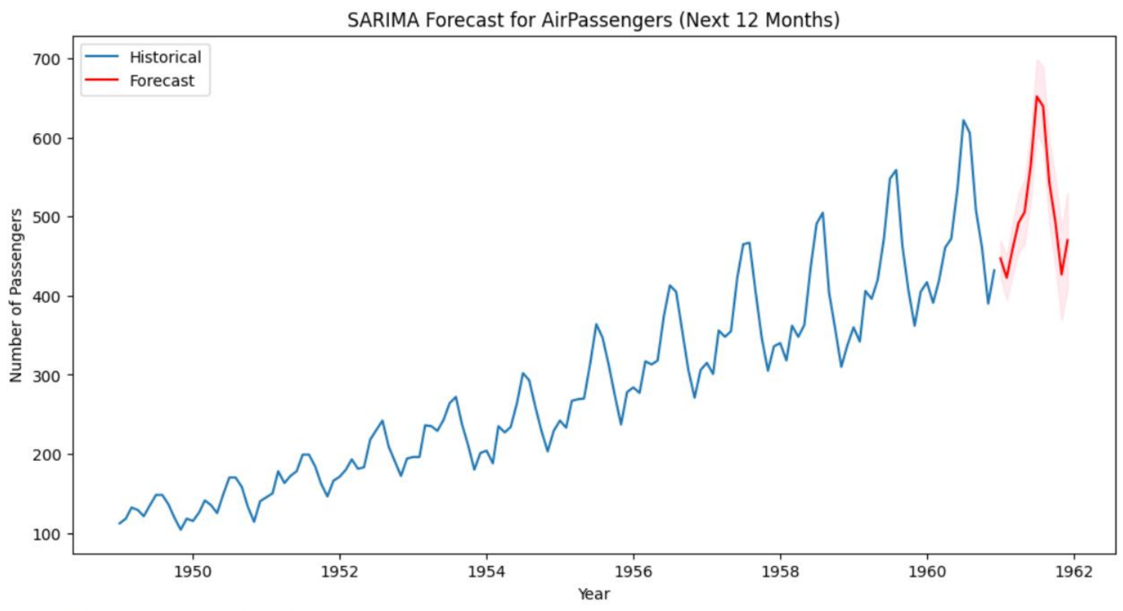
Answer :

Question 8: Train a SARIMA model on the monthly airline passengers dataset. Forecast the next 12 months and visualize the results. (Include your Python code and output in the code box below.)

Answer:


AirPassengers Time Series
Original Series
600
500
400
300
200
100
Number of Passengers
1950
1952
1954
1956
1958
1960
Year


Passengers
600
400
200
Trend
400
200
Seasonal
1.2
1.0
0.8
Resid
1.0
0.5
0.0
1950
1952
1954
1956
1958
1960



```
Forecasted Passengers for next 12 months:
1961-01-01    447.222620
1961-02-01    422.734098
1961-03-01    457.700895
1961-04-01    492.277505
1961-05-01    505.563366
1961-06-01    565.671782
1961-07-01    651.986404
1961-08-01    639.634242
1961-09-01    543.058845
1961-10-01    493.727026
1961-11-01    426.938001
1961-12-01    470.135609
Freq: MS, dtype: float64
```

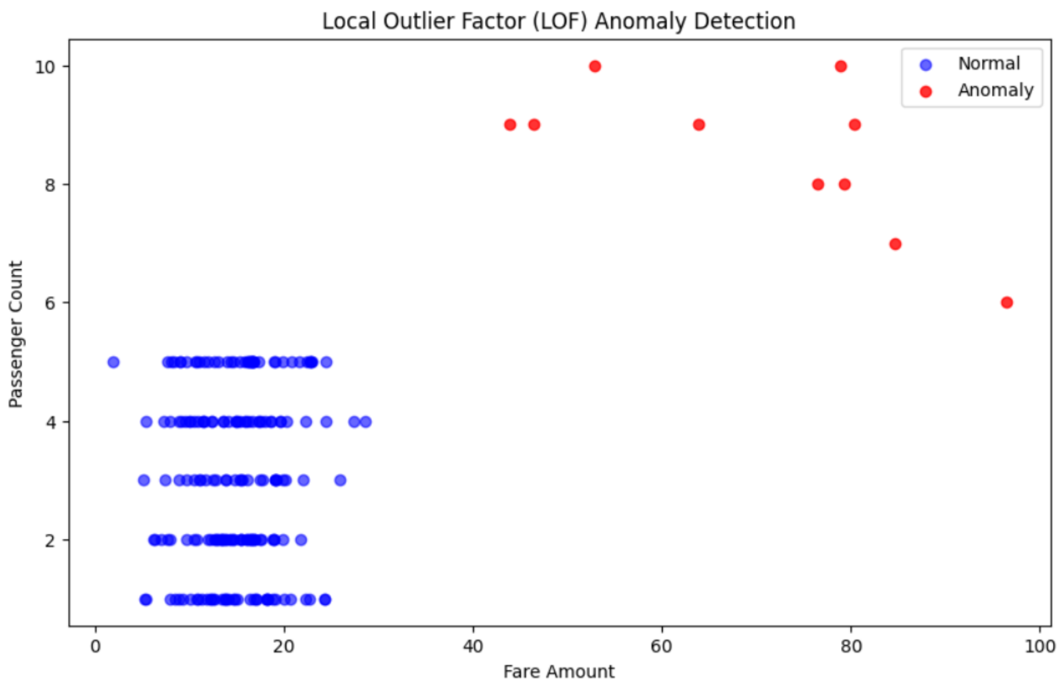
Question 9: Apply Local Outlier Factor (LOF) on any numerical dataset to detect anomalies and visualize them using matplotlib.

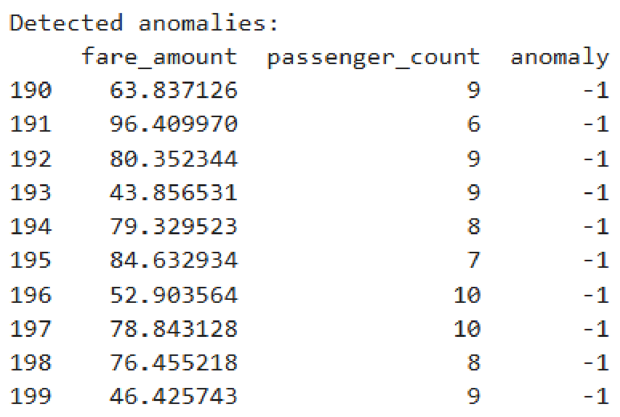
Answer :

```
# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.neighbors import LocalOutlierFactor
```

```
# Step 1: Create a synthetic numerical dataset
np.random.seed(42)

# Normal data
fare_amount = np.random.normal(15, 5, 200)   # average fare ~$15
passenger_count = np.random.randint(1, 6, 200)

# Inject some anomalies
fare_amount[-10:] = fare_amount[-10:] * 5
passenger_count[-10:] = passenger_count[-10:] + 5

data = pd.DataFrame({
    'fare_amount': fare_amount,
    'passenger_count': passenger_count
})

# Step 2: Apply Local Outlier Factor
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.05)
data['anomaly'] = lof.fit_predict(data)  # -1 for outliers, 1 for normal

# Separate normal points and anomalies
normal = data[data['anomaly'] == 1]
anomalies = data[data['anomaly'] == -1]

# Step 3: Visualize anomalies
plt.figure(figsize=(10,6))
plt.scatter(normal['fare_amount'], normal['passenger_count'], c='blue', label='Normal', alpha=0.6)
plt.scatter(anomalies['fare_amount'], anomalies['passenger_count'], c='red', label='Anomaly',
alpha=0.8)
plt.xlabel('Fare Amount')
plt.ylabel('Passenger Count')

plt.title('Local Outlier Factor (LOF) Anomaly Detection')

plt.legend()

plt.show()


# Step 4: Print detected anomalies

print("Detected anomalies:")

print(anomalies)
```

OUTPUT :





```
Detected anomalies:
     fare_amount  passenger_count  anomaly
190    63.837126                9       -1
191    96.409970                6       -1
192    80.352344                9       -1
193    43.856531                9       -1
194    79.329523                8       -1
195    84.632934                7       -1
196    52.903564               10       -1
197    78.843128               10       -1
198    76.455218                8       -1
199    46.425743                9       -1
```

Question 10: You are working as a data scientist for a power grid monitoring company. Your goal is to forecast energy demand and also detect abnormal spikes or drops in real-time consumption data collected every 15 minutes. The dataset includes features like timestamp, region, weather conditions, and energy usage.

Explain your real-time data science workflow:

● How would you detect anomalies in this streaming data (Isolation Forest / LOF / DBSCAN)?

● Which time series model would you use for short-term forecasting (ARIMA / SARIMA / SARIMAX)?
● How would you validate and monitor the performance over time?

● How would this solution help business decisions or operations?

ANSWER :

Detecting Anomalies in Streaming Data

For real-time anomaly detection in energy usage:

- **Challenge:** Energy consumption data is collected every 15 minutes. Anomalies can be sudden spikes (unexpected high usage) or drops (equipment failure or outages).

Approach:

- **Feature selection:** Use numerical features like energy_usage, and optionally include weather-related features (temperature, humidity) and region-based features for more context.

- **Sliding window:** For real-time detection, maintain a sliding window of the last n observations (e.g., last 24 hours) to detect anomalies dynamically.

- **Algorithms:**

  1. **Isolation Forest:** Effective for high-dimensional streaming data; isolates anomalies faster as it works by randomly partitioning the feature space.

- **Local Outlier Factor (LOF):** Can detect local anomalies in consumption relative to neighboring patterns; works well when energy usage varies by region or time of day.
- **DBSCAN:** Useful if you want to detect clusters of abnormal points; less suitable for continuous streaming but good for batch anomaly analysis.

Implementation:

- Preprocess each incoming data batch.
- Standardize numerical features.
- Apply Isolation Forest or LOF to label anomalies in near real-time.
- Trigger alerts for any detected anomaly for operational monitoring.

Time Series Model for Short-Term Forecasting

- **Objective:** Predict energy demand for the next few intervals (15 min, 30 min, 1 hour) to optimize grid load and resources.
- **Recommended Model:**
  - **SARIMAX** – Seasonal ARIMA with exogenous variables.

Reason:

- Energy consumption has **daily/weekly seasonality** (morning/evening peaks, weekdays vs weekends).
- Exogenous variables like **temperature, weather, and region** can significantly impact demand.
- SARIMAX handles both seasonal effects and external predictors, giving accurate short-term forecasts.

Validation and Performance Monitoring

Offline validation:

- Split historical data into training and test sets.
- Use metrics like **MAE, RMSE, MAPE** for forecasting accuracy.
- For anomaly detection, use **precision, recall, F1-score**, if labeled anomalies are available.

Online monitoring (streaming validation):

- Use a **rolling window evaluation**: compare predicted vs actual energy usage in real-time.
- Track **drift in data patterns** – if performance drops, retrain or update the model.
- Log detected anomalies and forecast errors for continual improvement.
- Visual dashboards: real-time plots of actual vs predicted energy usage and anomaly flags.

Business and Operational Benefits

- **Proactive grid management:**
  - Detect sudden drops in consumption → indicate outages or equipment failures.
  - Detect spikes → identify abnormal consumption or potential system overloads.
- **Resource optimization:**
  - Forecasting demand allows the company to **balance load**, **reduce energy waste**, and **plan maintenance**.
- **Decision support:**
  - Operations teams can receive **real-time alerts** and insights to act immediately.
  - Strategic planning: analyze seasonal or regional trends for infrastructure upgrades.
- **Safety and reliability:**
  - Reduce risk of blackouts by anticipating demand peaks.
  - Improve customer satisfaction by ensuring consistent energy supply.

This workflow ensures **real-time monitoring, short-term forecasting, and proactive operational decision-making**, which is critical for power grid reliability and efficiency.

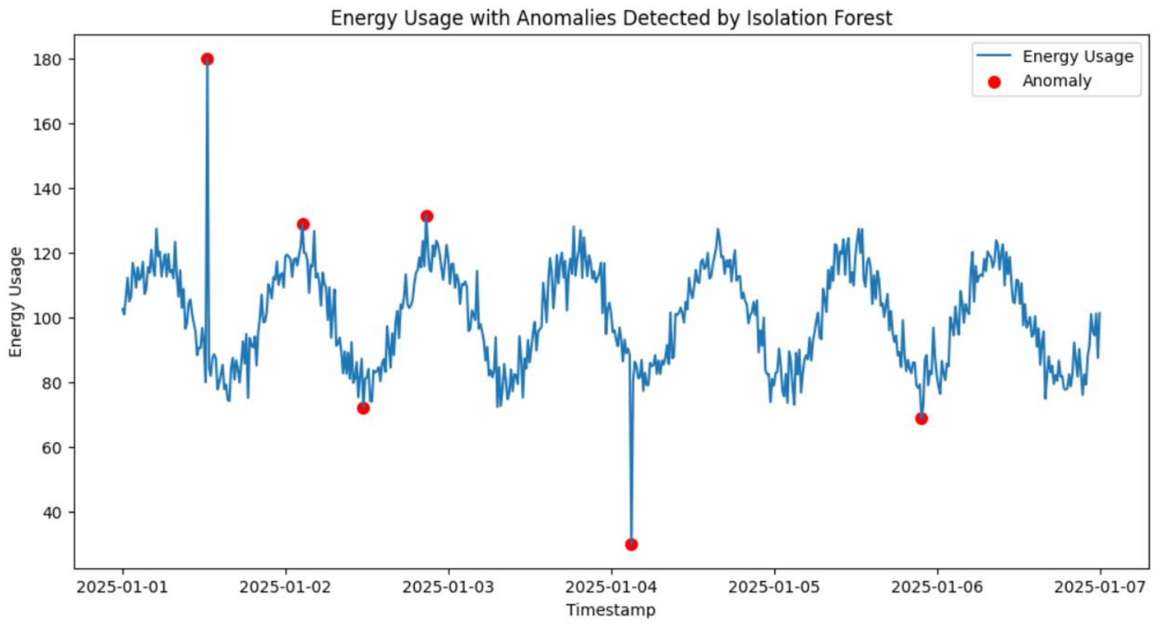
CODE :

```
# Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest

from statsmodels.tsa.statespace.sarimax import SARIMAX


# Step 1: Simulate sample energy consumption dataset

np.random.seed(42)

date_rng = pd.date_range(start='2025-01-01', end='2025-01-07', freq='15T')  # 1 week, 15-min intervals

n = len(date_rng)


# Simulated energy usage with daily seasonality + random noise

energy_usage = 100 + 20*np.sin(np.linspace(0, 2*np.pi*7, n)) + np.random.normal(0, 5, n)
```

```
# Inject anomalies
energy_usage[50] = 180  # spike
energy_usage[300] = 30  # drop

# Create DataFrame
data = pd.DataFrame({'timestamp': date_rng, 'energy_usage': energy_usage})
data.set_index('timestamp', inplace=True)

# Step 2: Anomaly Detection using Isolation Forest
iso_forest = IsolationForest(contamination=0.01, random_state=42)
data['anomaly'] = iso_forest.fit_predict(data[['energy_usage']])  # -1 = anomaly, 1 = normal

# Separate anomalies and normal points
normal = data[data['anomaly'] == 1]
anomalies = data[data['anomaly'] == -1]

# Plot anomalies
plt.figure(figsize=(12,6))
plt.plot(data.index, data['energy_usage'], label='Energy Usage')
plt.scatter(anomalies.index, anomalies['energy_usage'], color='red', label='Anomaly', s=50)
plt.xlabel('Timestamp')
plt.ylabel('Energy Usage')
plt.title('Energy Usage with Anomalies Detected by Isolation Forest')
plt.legend()
plt.show()

# Step 3: Short-Term Forecasting using SARIMAX
# Fit SARIMAX model (simple seasonal order for demonstration)
sarima_model = SARIMAX(data['energy_usage'],
                order=(1,1,1),
                seasonal_order=(1,1,1,96),  # 96 periods per day (15-min intervals)
```

```
                    enforce_stationarity=False,

                    enforce_invertibility=False)


sarima_results = sarima_model.fit(disp=False)


# Forecast next 12 periods (3 hours)

forecast_steps = 12

forecast = sarima_results.get_forecast(steps=forecast_steps)

forecast_index = pd.date_range(start=data.index[-1] + pd.Timedelta(minutes=15),
periods=forecast_steps, freq='15T')

forecast_series = pd.Series(forecast.predicted_mean.values, index=forecast_index)

conf_int = forecast.conf_int()


# Plot forecast

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

plt.plot(data.index, data['energy_usage'], label='Historical Energy Usage')

plt.plot(forecast_series.index, forecast_series, color='red', label='Forecast')

plt.fill_between(forecast_index, conf_int['lower energy_usage'], conf_int['upper energy_usage'],
color='pink', alpha=0.3)

plt.xlabel('Timestamp')

plt.ylabel('Energy Usage')

plt.title('SARIMAX Forecast of Energy Usage (Next 3 Hours)')

plt.legend()

plt.show()


# Print forecasted values

print("Forecasted Energy Usage for next 3 hours (15-min intervals):")

print(forecast_series)
```

OUTPUT :


Energy Usage with Anomalies Detected by Isolation Forest
Energy Usage
Anomaly
180
160
140
120
100
80
60
40
Energy Usage
2025-01-01
2025-01-02
2025-01-03
2025-01-04
2025-01-05
2025-01-06
2025-01-07
Timestamp