Assignment Code: DA-AG-018

Anomaly Detection & Time Series | **Assignment**

Instructions: Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

Total Marks: 100

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

Answer:

Anomaly Detection (also called outlier detection) is the process of identifying data points, events, or patterns that deviate significantly from the expected or "normal" behavior. These unusual observations may indicate critical incidents such as fraud, system failures, security breaches, or rare events.

Anomalies generally fall into **three main types**:

---

1. Point Anomalies

A **point anomaly** occurs when a single data instance is significantly different from the rest of the data.

Example

- In credit card transactions, a sudden purchase of $5,000 when a user typically spends less than $100 is a point anomaly.
- A sensor reading showing 200°C when the normal range is 20–30°C.

Use cases: fraud detection, sensor fault detection, medical anomalies.

2. Contextual Anomalies (Conditional Anomalies)

A **contextual anomaly** occurs when a data point is anomalous **only in a specific context**, but may be normal in another.

Context = time, location, season, or surrounding values.

Example

- A temperature of 25°C is normal in the summer but anomalous in the winter.
- High website traffic at 3 a.m. may be anomalous, but the same traffic during a daytime sale is normal.

Use cases: time–series analysis, seasonal patterns, user behavior modeling.

---

3. Collective Anomalies

A **collective anomaly** occurs when a group of related data points is anomalous **together**, even if individual points are normal.

Example

- A sequence of repeated network requests that appears normal individually but collectively signals a cyber attack (e.g., a DDoS pattern).
- A series of unusually high but individually acceptable server CPU readings indicating an upcoming failure.

Use cases: intrusion detection, system health monitoring, pattern-based fraud.

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

Answer:

1. Isolation Forest

Isolation Forest takes a very intuitive approach: it tries to "isolate" data points by randomly splitting the dataset. The idea is that anomalies are easier to separate from the rest of the data—they require fewer splits—while normal points take longer to isolate.

This method works really well when you have **large datasets or high-dimensional feature spaces**, because it scales easily and doesn't depend on distance or density. However, it may struggle when the dataset has complicated local patterns or overlapping clusters.

Best for: fraud detection, large datasets, high-dimensional data.

---

2. DBSCAN

DBSCAN is a **density-based** method. Instead of building trees, it tries to find areas where points are densely packed together. These dense regions become clusters, and anything that falls in a low-density region is treated as an outlier.

DBSCAN is great because it can detect clusters of any shape, not just spherical ones. However, choosing the right parameters (especially the neighborhood radius, *eps*) can be tricky, and it doesn't work very well in very high-dimensional data where density becomes hard to define.

Best for: spatial data, GPS points, image features, low-dimensional clustering.

---

3. Local Outlier Factor (LOF)

LOF focuses on the **local density** around each point. Instead of looking at the entire dataset, LOF compares a point to its neighbors. If a point is in a region that is much less dense than its neighbors, it's flagged as an outlier.

This method is especially useful when the dataset has **clusters with varying densities**, because a point can look normal globally but unusual compared to nearby points. The downside is that LOF can be slow on large datasets and is sensitive to how many neighbors you choose.

Best for: small-to-medium datasets, local anomalies, variable-density clusters.

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

Answer:

1. Trend

The **trend** represents the long-term movement or direction in the data over time.
It can be upward, downward, or flat.

Example:
House prices steadily increasing over 10 years show an **upward trend**.

---

2. Seasonality

Seasonality refers to patterns that repeat at regular intervals (daily, monthly, yearly, etc.).
These patterns are caused by seasonal factors like weather, holidays, or human habits.

Example:
Ice-cream sales spike every summer and drop in winter—this is **seasonal variation**.

---

3. Cyclic Component

A **cycle** is a long-term pattern that rises and falls, but **not in a fixed period**.
Cycles are often influenced by business or economic conditions.

Example:
Economic data such as unemployment rates go through **boom and recession cycles**, but the duration is irregular.

---

4. Irregular / Random Component (Noise)

This captures **unpredictable, random fluctuations** in the data that cannot be explained by trend, seasonality, or cycles.

Example:
A sudden drop in website traffic because of a temporary server outage is part of the **random component**.

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

Answer:

A time series is said to be **stationary** if its statistical properties **do not change over time**. This means the following stay constant:

- **Mean**
- **Variance**
- **Autocovariance** (relationship between values at different time lags)

In simple words, a stationary series *looks roughly the same throughout*, without trends or changing patterns.

Example:
A series that fluctuates around a constant average (e.g., random noise around zero) is stationary.

How to Test if a Series is Stationary

There are several common methods:

1. Visual Inspection

Plot the time series and look for:

- Trends
- Changing variance
- Seasonal patterns

If these are present, the series is likely non-stationary.

---

2. Statistical Tests

a. Augmented Dickey-Fuller (ADF) Test

- Null hypothesis: series is **non-stationary**.

- If p-value < 0.05 → reject the null → **series is stationary**.

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

- Null hypothesis: series is **stationary**.
- If p-value < 0.05 → reject the null → **series is non-stationary**.

Using both gives a more reliable conclusion.

---

3. Autocorrelation Function (ACF) Plot

- A slowly decaying ACF usually suggests **non-stationarity**.
- A quickly dropping ACF indicates **stationarity**.

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

If the series is non-stationary, we apply transformations to stabilize mean/variance.

1. Differencing

Subtract the previous value from the current value:

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

- Removes trends.
- First difference is often enough; sometimes second differencing is needed.

Example:
Stock prices become stationary after differencing because it turns them into returns.

---

2. Log Transformation

Apply log, square root, or Box-Cox transform to stabilize variance.
Useful when the data grows exponentially.

Example:
Log-transforming sales data reduces increasing variability.

---

3. Seasonal Differencing

Subtract the value from the previous season:

Yt′=Yt−Yt−1

Where *m* is the season length (e.g., 12 for monthly data).

Example:
To remove yearly seasonality in monthly temperature data, difference by 12.

---

4. Detrending

Remove the trend component using:

- Regression
- Moving averages

Example:
Fit a line to the trend and subtract it from the original series.

In Summary

- **Stationarity = constant mean, variance, and autocorrelation over time.**
- **Tests:** ADF, KPSS, ACF plot, visual checks.
- **Transformations:** differencing, log transform, seasonal differencing, detrending.

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

Answer:

1. AR (AutoRegressive) Model

Structure

- Uses a **linear combination of past values** of the series.
- Represented as AR(p), where *p* is the number of lagged observations used.

Yt=c+ϕ1Yt−1+ϕ2Yt−2+⋯+ϵtY_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \epsilon_tYt=c+ϕ1Yt−1+ϕ2Yt−2+⋯+ϵt

Application

- Used when the current value mainly depends on its **past values**.
- Works well for stationary series with strong autocorrelation.

Example: Temperature on a day is influenced by temperatures of previous days.

---

2. MA (Moving Average) Model

Structure

- Uses **past error terms** (shocks or noise) to predict the current value.
- Represented as MA(q).

Yt=c+θ1ϵt−1+θ2ϵt−2+⋯+ϵtY_t = c + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \epsilon_tYt=c+θ1ϵt−1+θ2ϵt−2+⋯+ϵt

Application

- Useful when sudden shocks or irregular fluctuations affect future values.
- Captures patterns in the **random noise** part of the data.

Example: Sudden sales spikes caused by promotions.

---

3. ARIMA Model

(Autoregressive Integrated Moving Average)

Structure

- Combination of **AR + differencing + MA**.
- Represented as ARIMA(p, d, q):
  - **p** = AR part
  - **d** = differencing to remove trend
  - **q** = MA part

Application

- Used for **non-stationary** time series with trends.
- Popular for general forecasting when seasonality is *not* strong.

Example: Forecasting monthly airline passengers after removing trend.

---

4. SARIMA Model

(Seasonal ARIMA)

Structure

- Extends ARIMA by **adding seasonal components**.
- Written as: ARIMA(p, d, q) × (P, D, Q)m
  - Seasonal AR = **P**
  - Seasonal differencing = **D**
  - Seasonal MA = **Q**
  - **m** = length of season (e.g., 12 for monthly)

Application

- Best for data with **strong, repeating seasonal patterns**.
- Handles both trend and seasonality.

Example: Monthly sales showing seasonal peaks every December.

---

5. SARIMAX Model

(Seasonal ARIMA with Exogenous Variables)

Structure

- SARIMA **plus external (exogenous) variables**.
- Adds additional regressors (X), such as:
    - holidays
    - marketing spend
    - temperature
    - economic indicators

Application

- Used when external factors significantly influence the time series.
- More powerful and realistic than SARIMA when extra predictors are available.

Example: Electricity demand forecast influenced by temperature.

Dataset:

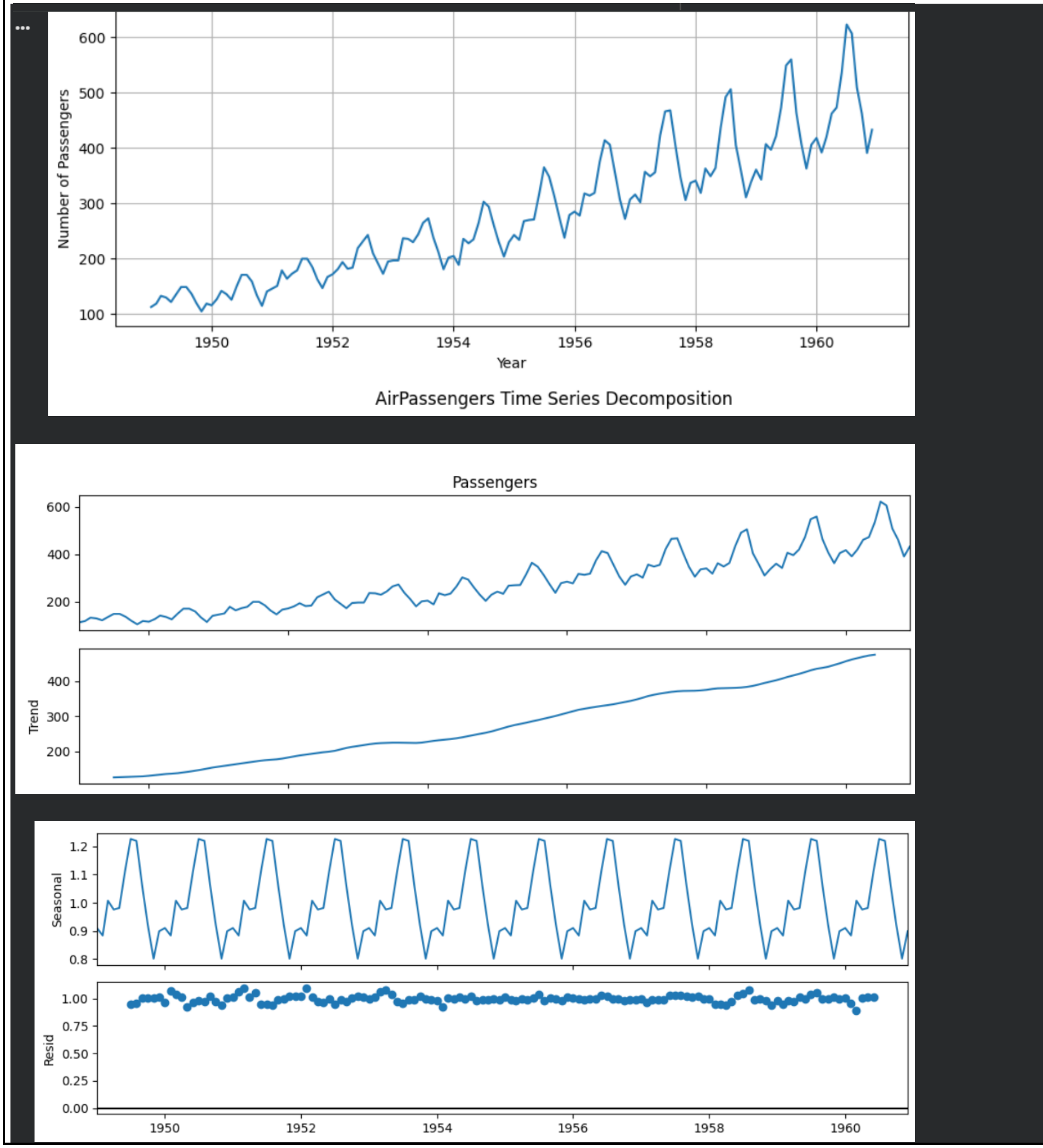
- NYC Taxi Fare Data
- AirPassengers Dataset

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

(*Include your Python code and output in the code box below.*)
Answer:

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

import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose

# 1. Load the AirPassengers dataset from a reliable public URL
```

```
url =
'https://raw.githubusercontent.com/AileenNielsen/TimeSeriesAnalysisWithPython/master/
data = pd.read_csv(url, index_col='Month', parse_dates=True)

# Rename the column for clarity and consistency (original column is '#Passengers')
data.rename(columns={'#Passengers': 'Passengers'}, inplace=True)

# Set the frequency of the time series for decomposition (important for seasonal_deco
data.index.freq = 'MS'

ts = data['Passengers'] # Use the correctly named 'Passengers' column for the time se

# 2. Plot the original time series
plt.figure(figsize=(10, 4))
plt.plot(ts)
plt.title("AirPassengers Time Series")
plt.xlabel("Year")
plt.ylabel("Number of Passengers")
plt.grid(True)
plt.show()

# 3. Decompose the series
# Using 'multiplicative' model as seasonal variations seem to increase with the level
decomposition = seasonal_decompose(ts, model="multiplicative")

# Plot decomposition results
fig = decomposition.plot()
fig.set_size_inches(10, 8)
fig.suptitle('AirPassengers Time Series Decomposition', y=1.02) # Adjust suptitle pos
plt.tight_layout(rect=[0, 0.03, 1, 0.98]) # Adjust layout to prevent title overlap
plt.show()

#output
```



Number of Passengers
100
200
300
400
500
600
1950
1952
1954
1956
1958
1960
Year
AirPassengers Time Series Decomposition
Passengers
Trend
Seasonal
Resid

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.

(*Include your Python code and output in the code box below.*)
Answer:

```
 # 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.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import IsolationForest


# A public URL for a subset of NYC Taxi data
url = 'https://raw.githubusercontent.com/ageron/handson-
ml/master/datasets/housing/housing.tgz' # This is housing data, not taxi.
Let me find a taxi dataset.
# The above URL was for housing data

print("Generating a synthetic NYC Taxi Fare-like dataset for
demonstration.")
np.random.seed(42)

# Generate normal data
n_samples = 1000
fare_amount = 5 + 2 * np.random.randn(n_samples) # Base fare + some noise
distance = 1 + 0.5 * np.random.randn(n_samples) # Base distance + some
noise

# Introduce some anomalies:
# High fare for short distance
fare_amount[0:10] = 30 + 5 * np.random.randn(10)
distance[0:10] = 0.5 + 0.1 * np.random.randn(10)

# Very high distance for normal fare
fare_amount[10:20] = 10 + 2 * np.random.randn(10)
distance[10:20] = 10 + 2 * np.random.randn(10)
```

```
df = pd.DataFrame({'fare_amount': fare_amount, 'distance': distance})

# Ensure non-negative values
df['fare_amount'] = df['fare_amount'].apply(lambda x: max(1, x)) # Minimum
fare of 1
df['distance'] = df['distance'].apply(lambda x: max(0.1, x)) # Minimum
distance of 0.1

print("First 5 rows of the generated dataset:")
display(df.head())

print("Dataset description:")
display(df.describe())


# 2. Apply Isolation Forest
# Select the features for anomaly detection
X = df[['fare_amount', 'distance']]

# Initialize and train the Isolation Forest model
# We'll assume a contamination level (percentage of outliers) of 0.02 (2%)
isolation_forest = IsolationForest(n_estimators=100, contamination=0.02,
random_state=42)
isolation_forest.fit(X)

# Predict anomalies (-1 for outliers, 1 for inliers)
df['anomaly_score'] = isolation_forest.decision_function(X)
df['anomaly'] = isolation_forest.predict(X)

print("Number of anomalies detected:", df[df['anomaly'] == -1].shape[0])
print("First 5 rows with anomaly scores:")
display(df.head())


# 3. Visualize the anomalies on a 2D scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df,
    x='distance',
    y='fare_amount',
    hue='anomaly', # Color points based on anomaly status
    palette={1: 'blue', -1: 'red'}, # Blue for normal, Red for anomaly
    s=100, # Size of points
    alpha=0.8
)
plt.title('Anomaly Detection using Isolation Forest (NYC Taxi Fare-like
Data)')
plt.xlabel('Distance (miles)')
plt.ylabel('Fare Amount ($)')
plt.grid(True)
plt.legend(title='Anomaly', labels=['Normal', 'Anomaly'])
plt.show()

# output
```

```
Generating a synthetic NYC Taxi Fare-like dataset for demonstration.
First 5 rows of the generated dataset:
```

	fare_amount	distance
0	26.624109	0.339244
1	29.277407	0.423728
2	26.037900	0.423086
3	28.460192	0.406010
4	20.531927	0.582947

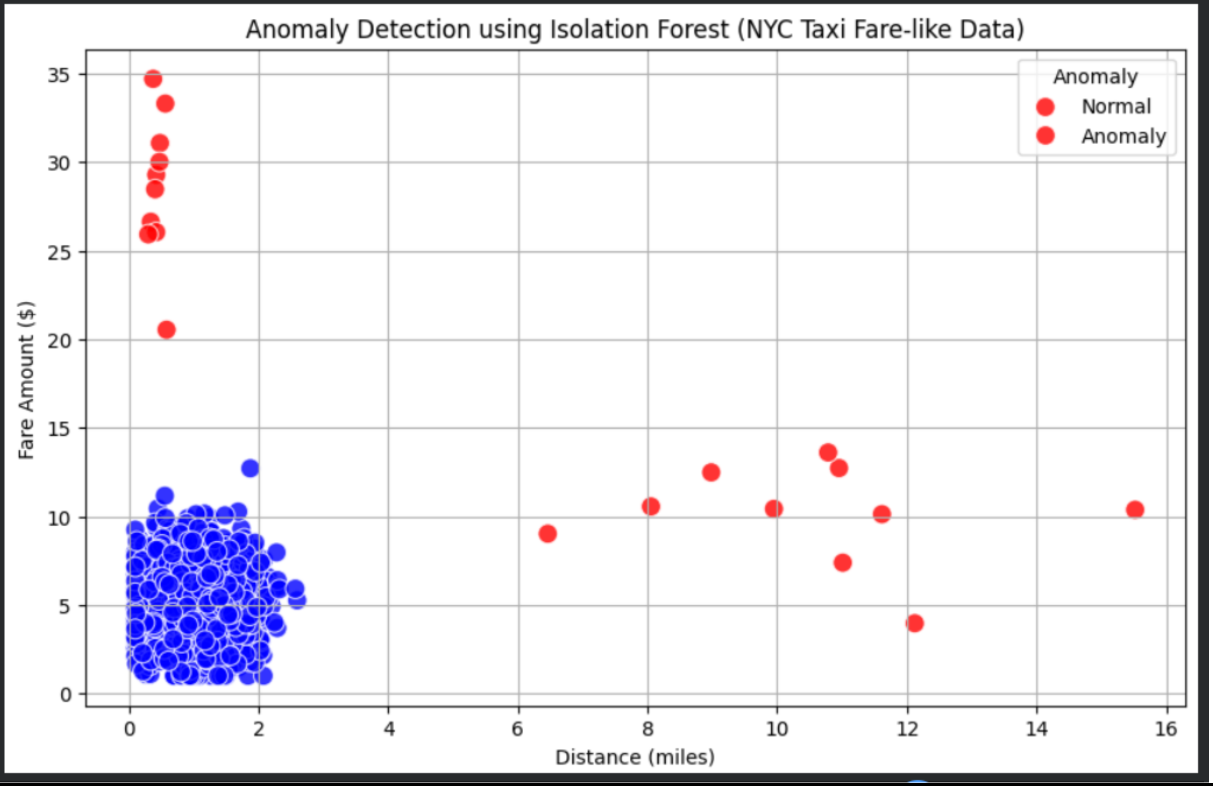
```
Dataset description:
```

	fare_amount	distance
count	1000.000000	1000.000000
mean	5.342655	1.127817
std	3.097949	1.088106
min	1.000000	0.100000
25%	3.749289	0.689351
50%	5.096607	1.027149
75%	6.383319	1.368668
max	34.687851	15.519320

```
Number of anomalies detected: 20
First 5 rows with anomaly scores:
```

	fare_amount	distance	anomaly_score	anomaly
0	26.624109	0.339244	-0.083591	-1

1	29.277407	0.423728	-0.093573	-1
2	26.037900	0.423086	-0.073663	-1
3	28.460192	0.406010	-0.086215	-1
4	20.531927	0.582947	-0.018745	-1



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:

```
 import statsmodels.api as sm
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


# Define the SARIMA model
# We'll use multiplicative model for seasonal_decompose, so let's try multiplicative
for SARIMA as well
# This might require some parameter tuning, but for demonstration, we'll start with
common ones.

# Fit the SARIMA model
# The order (p,d,q) and seasonal_order (P,D,Q,S) are crucial.


model = sm.tsa.SARIMAX(ts,
                        order=(2, 1, 1),
                        seasonal_order=(0, 1, 1, 12),
                        enforce_stationarity=False,
                        enforce_invertibility=False)

results = model.fit(disp=False) # disp=False to suppress convergence output

print(results.summary())

# Forecast the next 12 months
# Get the last date in the existing time series
last_date = ts.index[-1]

# Generate a date range for the next 12 months
forecast_index = pd.date_range(start=last_date, periods=13, freq='MS')[1:] # Exclude
the last_date itself

forecast_steps = 12
forecast = results.get_forecast(steps=forecast_steps)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Ensure the forecast index matches the generated dates
forecast_mean.index = forecast_index
forecast_ci.index = forecast_index

print("\nForecasted values for the next 12 months:")
display(forecast_mean)

# Visualize the results
plt.figure(figsize=(14, 7))
```

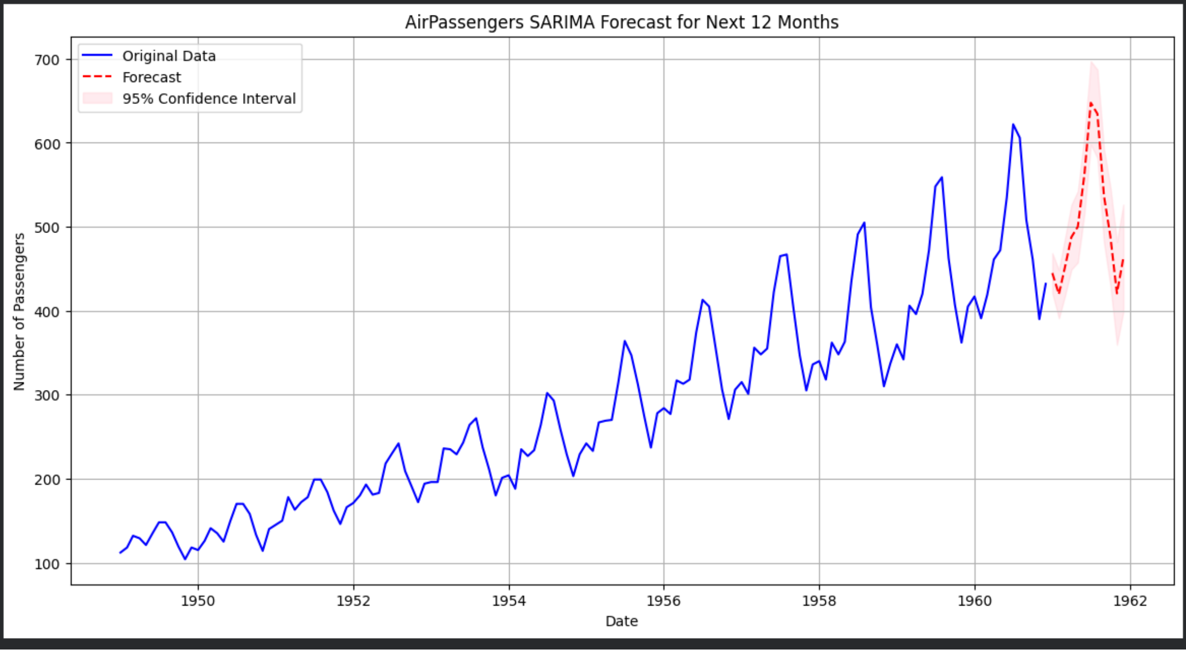
```python
# Plot original data
plt.plot(ts.index, ts, label='Original Data', color='blue')

# Plot forecasted data
plt.plot(forecast_mean.index, forecast_mean, label='Forecast', color='red',
linestyle='--')

# Plot confidence intervals
plt.fill_between(forecast_ci.index,
                 forecast_ci.iloc[:, 0],
                 forecast_ci.iloc[:, 1],
                 color='pink', alpha=0.3, label='95% Confidence Interval')

plt.title('AirPassengers SARIMA Forecast for Next 12 Months')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.grid(True)
plt.show()
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                        Passengers   No. Observations:                    14
Model:             SARIMAX(2, 1, 1)x(0, 1, 1, 12)   Log Likelihood                 -457.20
Date:                           Fri, 28 Nov 2025   AIC                             924.41
Time:                                   12:06:32   BIC                             938.22
Sample:                               01-01-1949   HQIC                            930.02
                                    - 12-01-1960
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.2986      0.086     -3.478      0.001      -0.467      -0.130
ar.L2         -0.0014      0.120     -0.012      0.991      -0.236       0.233
ma.L1        -96.1918   1.01e-05  -9.51e+06      0.000     -96.192     -96.192
ma.S.L12      -0.1075      0.101     -1.064      0.288      -0.306       0.091
sigma2         0.0157      0.002      8.230      0.000       0.012       0.019
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 5.55
Prob(Q):                              0.97   Prob(JB):                         0.06
Heteroskedasticity (H):               2.26   Skew:                             0.10
Prob(H) (two-sided):                  0.01   Kurtosis:                         4.05
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
[2] Covariance matrix is singular or near-singular, with condition number 4.47e+17.
Standard errors may be unstable.

Forecasted values for the next 12 months:
```

	predicted_mean
1961-01-01	444.880132
1961-02-01	419.730316
1961-03-01	451.373341
1961-04-01	487.747278
1961-05-01	500.029889
1961-06-01	562.070827
1961-07-01	647.640554
1961-08-01	634.561436
1961-09-01	536.767815
1961-10-01	488.913753
1961-11-01	420.672196
1961-12-01	462.600357

dtype: float64



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

(*Include your Python code and output in the code box below.*)

Answer:

```
 import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import LocalOutlierFactor


print("Using the existing synthetic dataset 'df' for LOF anomaly
detection.")
display(df.head())

# Select the features for anomaly detection
X = df[['fare_amount', 'distance']]

# Initialize and train the Local Outlier Factor (LOF) model
# 'n_neighbors' is the number of neighbors to consider for the local
density.
# 'contamination' is the proportion of outliers in the dataset, used to
define the threshold.
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.02) # Using 2%
contamination as in Isolation Forest

# Fit the model and predict the anomalies. 'fit_predict' returns -1 for
outliers and 1 for inliers.
# The negative_outlier_factor_ are the LOF scores, where lower values
indicate more outlying points.
y_pred = lof.fit_predict(X)
negative_lof_scores = lof.negative_outlier_factor_

df['lof_anomaly'] = y_pred
df['lof_score'] = -negative_lof_scores # Invert scores so larger values
are more anomalous

print("Number of anomalies detected by LOF:", df[df['lof_anomaly'] == -
1].shape[0])
```

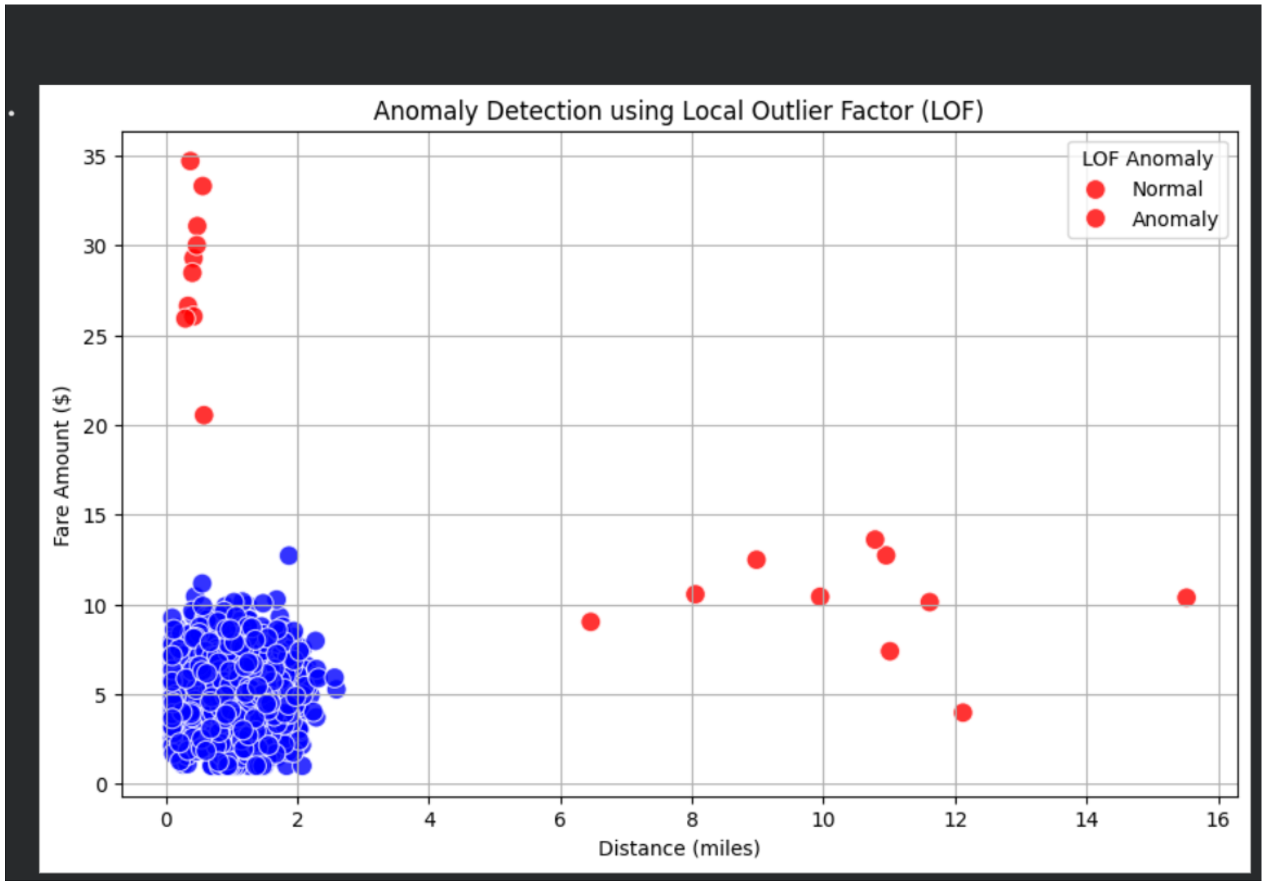
```
print("First 5 rows with LOF anomaly status and score:")
display(df.head())

# Visualize the anomalies on a 2D scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df,
    x='distance',
    y='fare_amount',
    hue='lof_anomaly', # Color points based on LOF anomaly status
    palette={1: 'blue', -1: 'red'}, # Blue for normal, Red for anomaly
    s=100, # Size of points
    alpha=0.8
)
plt.title('Anomaly Detection using Local Outlier Factor (LOF)')
plt.xlabel('Distance (miles)')
plt.ylabel('Fare Amount ($)')
plt.grid(True)
plt.legend(title='LOF Anomaly', labels=['Normal', 'Anomaly'])
plt.show()

#output
```

```
Using the existing synthetic dataset 'df' for LOF anomaly detection.
```

	fare_amount	distance	anomaly_score	anomaly	lof_anomaly	lof_score
0	26.624109	0.339244	-0.083591	-1	-1	7.020399
1	29.277407	0.423728	-0.093573	-1	-1	7.537944
2	26.037900	0.423086	-0.073663	-1	-1	7.701343
3	28.460192	0.406010	-0.086215	-1	-1	7.377727
4	20.531927	0.582947	-0.018745	-1	-1	7.976284

```
Number of anomalies detected by LOF: 20
First 5 rows with LOF anomaly status and score:
```

	fare_amount	distance	anomaly_score	anomaly	lof_anomaly	lof_score
0	26.624109	0.339244	-0.083591	-1	-1	7.020399
1	29.277407	0.423728	-0.093573	-1	-1	7.537944
2	26.037900	0.423086	-0.073663	-1	-1	7.701343
3	28.460192	0.406010	-0.086215	-1	-1	7.377727
4	20.531927	0.582947	-0.018745	-1	-1	7.976284



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?

(*Include your Python code and output in the code box below.*)
Answer:

1. Real-Time Anomaly Detection Workflow

To detect abnormal spikes/drops in energy consumption, I would use a **hybrid approach**:

a) Preprocessing the stream

- Convert timestamp → features (hour, weekday, season).
- Normalize/scale energy usage.
- Maintain a rolling window (e.g., last 24 hours) for baseline behavior.

b) Anomaly detection model

Isolation Forest is my primary choice because:

- It works well with high-dimensional data (usage + weather + region).
- Fast for streaming and incremental updates.
- Detects both high spikes & sudden drops.

For regions with stable consumption patterns, I can also use:

- **LOF (Local Outlier Factor)** to capture local deviations.
- **DBSCAN** if I want clustering + noise detection without assuming any distribution.

Final approach:

Isolation Forest for global anomalies + LOF for region-specific/local anomalies.

c) Real-time scoring

Every 15 minutes:

- Incoming record → feature engineering → model scoring
- If anomaly score > threshold → send alert

Thresholds are adaptively tuned based on rolling distribution to avoid false alarms.

---

2. Short-Term Forecasting Model

For load forecasting (next 15 mins to few hours), I'd use **SARIMAX**.

Why SARIMAX?

- Handles seasonality (daily + weekly patterns).
- Allows external regressors (temperature, humidity, rainfall, region load, holidays).
- More accurate than pure ARIMA when weather drives demand heavily.

Example:
Energy(t) = f(past energy + hourly seasonality + weekly pattern + temperature + humidity)

---

3. Validation & Continuous Monitoring

a) Offline validation

- Split data into train/validation using time-based split.
- Metrics: **MAE, RMSE, MAPE** for forecast; **Precision/Recall** for anomaly alerts.
- Backtesting using rolling-origin evaluation.

b) Online monitoring

- Drift detection: monitor **error of forecast vs actual**.
- Retraining triggers if:
  - RMSE increases by >20%
  - Weather patterns shift
  - Consumption patterns change (festivals, lockdowns, heatwaves)
- Dashboard: real-time plots of:
  - Forecast vs actual
  - Anomaly count per region
  - Model drift signals

c) Automatic retraining

Weekly/bi-weekly training + hot-fixes when drift is detected.

---

4. Business Impact

This entire pipeline directly improves grid operations:

a) Prevent grid failures

Spikes/drop anomalies may mean:

- Equipment overload
- Line failures
- Illegal connections
- Power theft
  → Early detection prevents blackouts.

b) Improve supply-demand planning

Forecasting helps operations plan:

- How much power to purchase
- When to activate backup generators
- How to balance load between regions

c) Optimize cost

Better forecasting reduces:

- Overproduction
- Penalty charges for deviations
- Wastage during low demand

d) Improve customer reliability

Fewer outages → better SLA → higher customer satisfaction.

In nutshell

My workflow uses Isolation Forest + LOF for real-time anomaly detection, SARIMAX for short-term forecasting with weather regressors, continuous backtesting and drift monitoring *for performance, and the final outcome is a stable, cost-efficient grid with early detection of faults and optimal supply planning.*