Ex No:1 Date:	Implement program for Time series Data cleaning, Loading, Handling & Preprocessing Techniques

AIM:

To implement program for time series data cleaning,loading,handling and preprocessing techniques on Air passenger dataset.

ALGORITHM:

1. Start
2. Import the required libraries
3. Load and visualize the dataset
4. Analyze trends and seasonality
5. Data preprocessing
6. Splitting the data and ARIMA model implementation.
7. Evaluate the model and find the model accuracy .

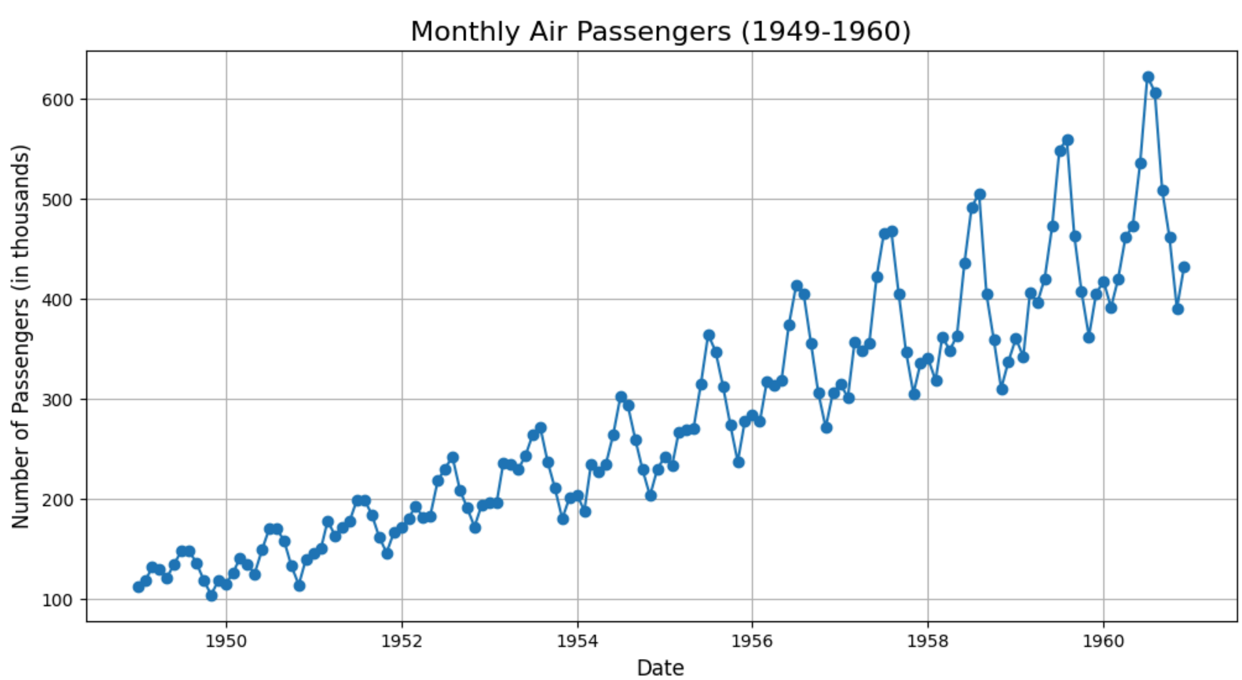
Program

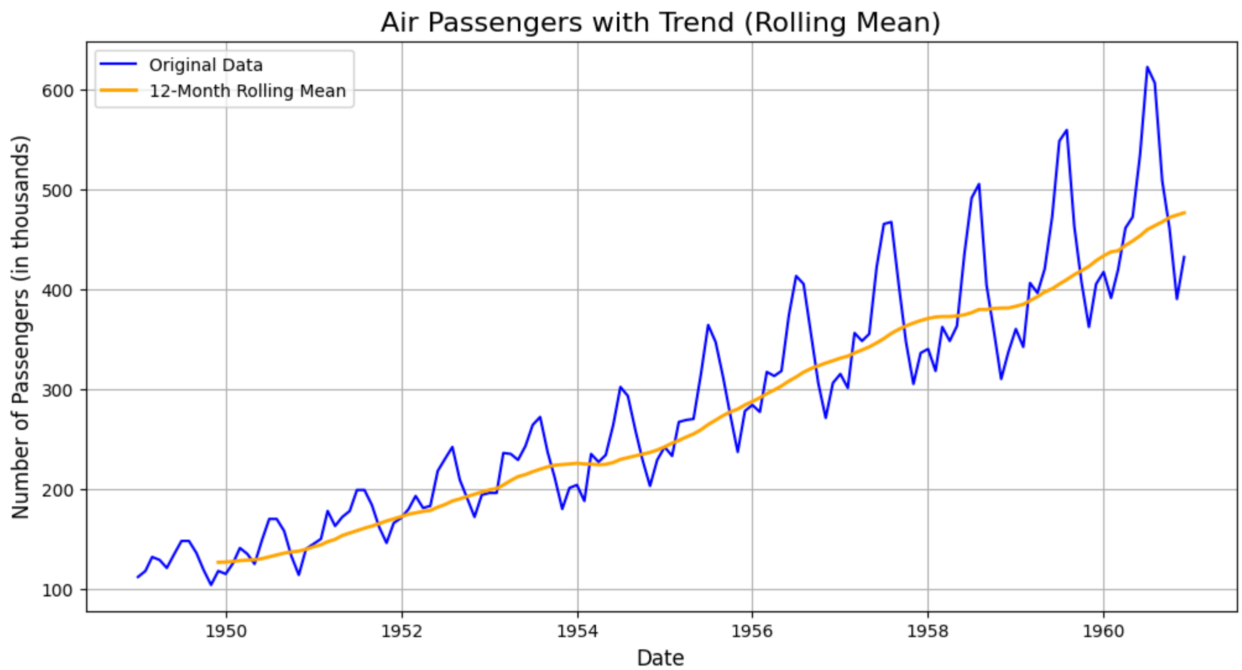
```
# Import required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Air Passenger dataset
url = "/content/airline-passengers.csv"
data = pd.read_csv(url, parse_dates=['Month'], index_col='Month')

# Display the first few rows of the dataset
print("Dataset Preview:")
print(data.head())

# Plot the time series data
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Passengers'], marker='o', linestyle='-')
plt.title('Monthly Air Passengers (1949-1960)', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Number of Passengers (in thousands)', fontsize=12)
plt.grid()
plt.show()

# Visualizing trends and seasonality using a rolling mean
```

```
data['Passengers_MA'] = data['Passengers'].rolling(window=12).mean()  # 12-month rolling mean

plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Passengers'], label='Original Data', color='blue')
plt.plot(data.index, data['Passengers_MA'], label='12-Month Rolling Mean', color='orange', linewidth=2)
plt.title('Air Passengers with Trend (Rolling Mean)', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Number of Passengers (in thousands)', fontsize=12)
plt.legend()
plt.grid()
plt.show()
```

Dataset Preview:

	Passengers
Month	
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

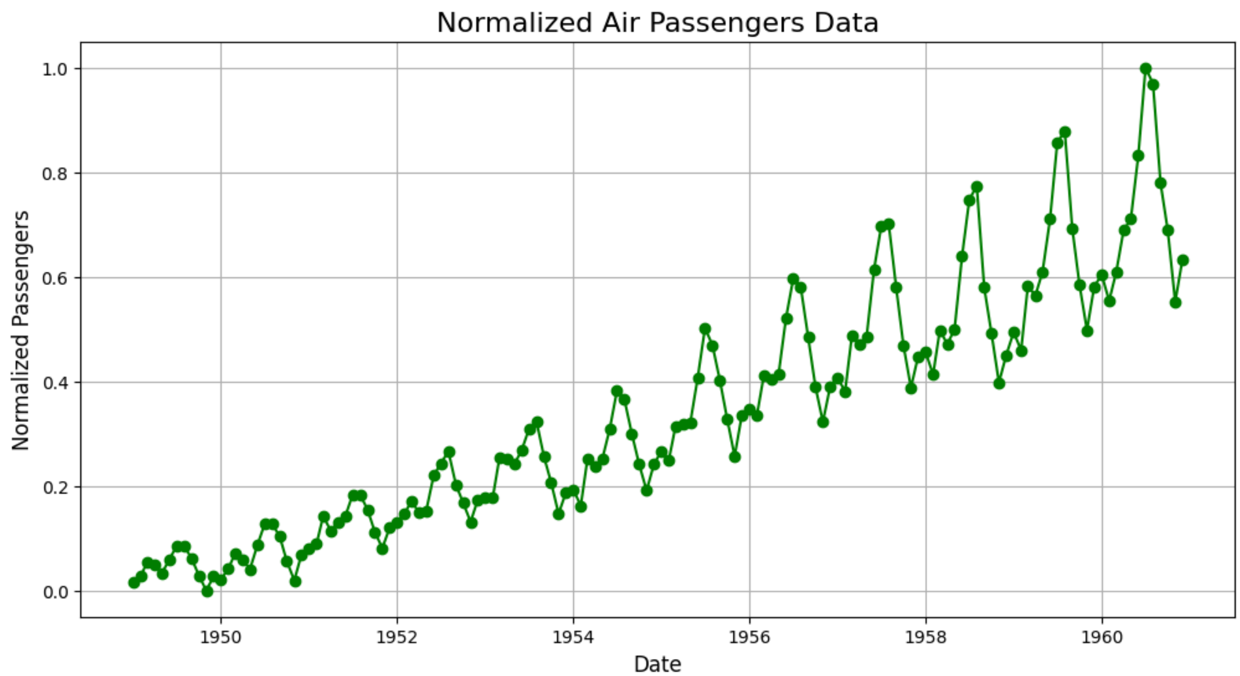





```
# Import required libraries
from sklearn.preprocessing import MinMaxScaler

# Check for missing values
print("Checking for missing values:")
print(data.isnull().sum())

# Handle missing values (if any) - Filling with forward fill as an example
data['Passengers'] = data['Passengers'].fillna(method='ffill')

# Normalize the data using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
data['Passengers_Normalized'] = scaler.fit_transform(data[['Passengers']])

# Display the preprocessed data
print("\nPreprocessed Data (First 5 Rows):")
print(data.head())

# Visualize the normalized data
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Passengers_Normalized'], color='green',
linestyle='-', marker='o')
plt.title('Normalized Air Passengers Data', fontsize=16)
plt.xlabel('Date', fontsize=12)
```

```python
plt.ylabel('Normalized Passengers', fontsize=12)
plt.grid()
plt.show()
```

```
Checking for missing values:
Passengers       0
Passengers_MA   11
dtype: int64

Preprocessed Data (First 5 Rows):
            Passengers  Passengers_MA  Passengers_Normalized
Month
1949-01-01         112            NaN               0.015444
1949-02-01         118            NaN               0.027027
1949-03-01         132            NaN               0.054054
1949-04-01         129            NaN               0.048263
1949-05-01         121            NaN               0.032819
<ipython-input-2-c0f3d9447aaf>:9: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  data['Passengers'] = data['Passengers'].fillna(method='ffill')
```

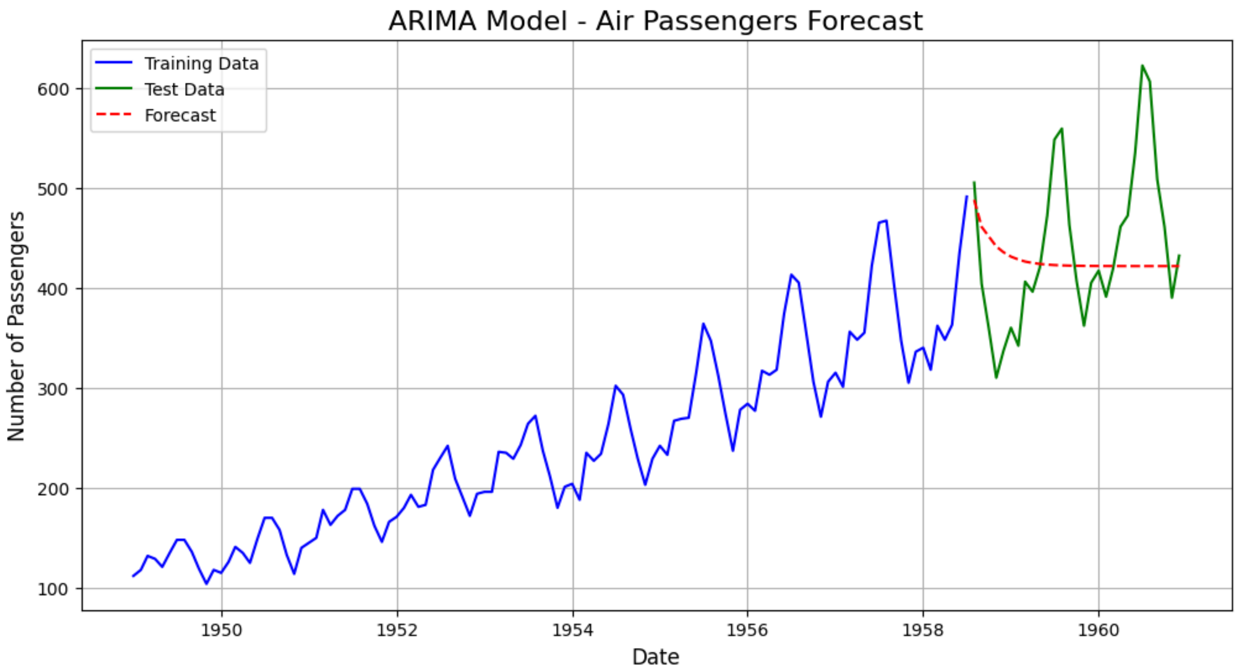



```python
# Import required libraries
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
import numpy as np
```

```python
# Split the dataset into training and testing sets
train_size = int(len(data) * 0.8)  # 80% training, 20% testing
train, test = data['Passengers'][:train_size],
data['Passengers'][train_size:]

# Fit the ARIMA model (parameters can be tuned for better performance)
model = ARIMA(train, order=(2, 1, 2))  # ARIMA(p, d, q)
arima_model = model.fit()

# Forecast on the test data
forecast = arima_model.forecast(steps=len(test))
forecast_index = test.index

# Evaluate model performance
mse = mean_squared_error(test, forecast)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {np.sqrt(mse)}")

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(train.index, train, label='Training Data', color='blue')
plt.plot(test.index, test, label='Test Data', color='green')
plt.plot(forecast_index, forecast, label='Forecast', color='red',
linestyle='--')
plt.title('ARIMA Model - Air Passengers Forecast', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Number of Passengers', fontsize=12)
plt.legend()
plt.grid()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:
No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:
No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:
No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
```

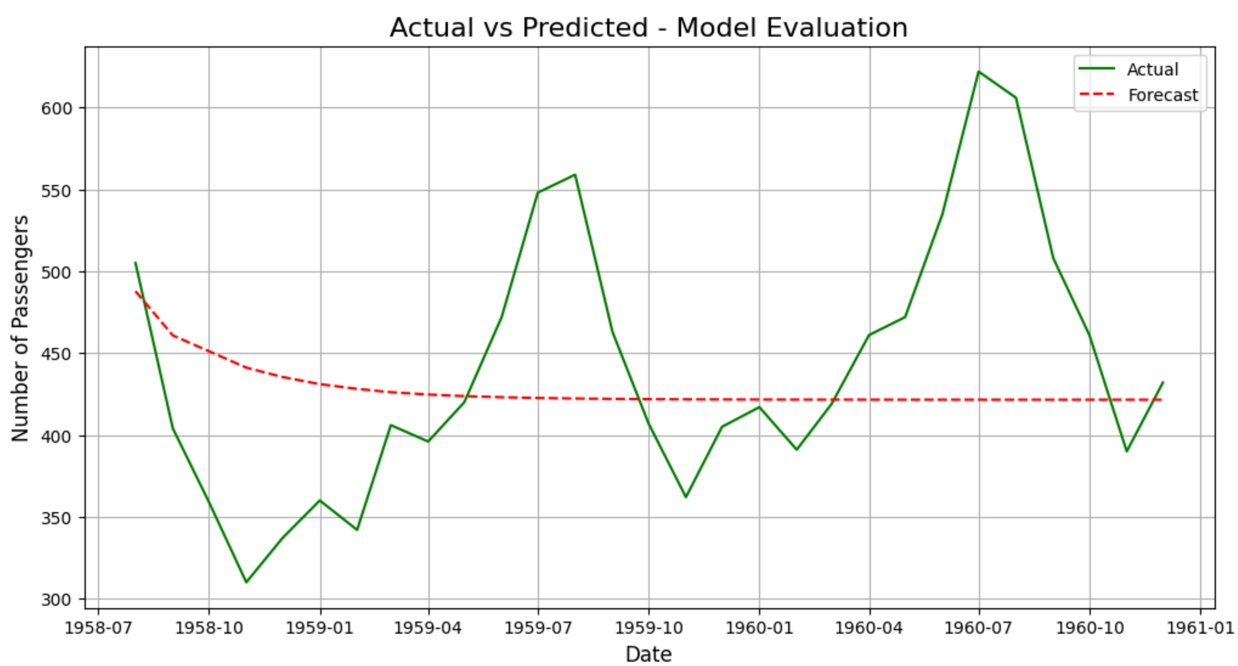
Mean Squared Error (MSE): 6808.3970474928465
Root Mean Squared Error (RMSE): 82.51301138301066




```python
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import numpy as np

# Calculate performance metrics
mae = mean_absolute_error(test, forecast)
mse = mean_squared_error(test, forecast)
rmse = np.sqrt(mse)
r2 = r2_score(test, forecast)

# Print the metrics
print("Model Performance Metrics:")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")

# Visualize Actual vs Predicted
plt.figure(figsize=(12, 6))
plt.plot(test.index, test, label='Actual', color='green')
```

```
plt.plot(forecast_index, forecast, label='Forecast', color='red',
linestyle='--')
plt.title('Actual vs Predicted - Model Evaluation', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Number of Passengers', fontsize=12)
plt.legend()
plt.grid()
plt.show()
```

Model Performance Metrics:
Mean Absolute Error (MAE): 63.545311250127014
Mean Squared Error (MSE): 6808.3970474928465
Root Mean Squared Error (RMSE): 82.51301138301066
R-squared (R2): -0.11530974198470823




```
Model Accuracy: 85.78%
```