```
print("""
Import all required libraries for data manipulation, visualization,
statistical testing, and ARIMA modeling.""")
```

```
Import all required libraries for data manipulation, visualization,
statistical testing, and ARIMA modeling.
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_absolute_error, mean_squared_error

import warnings
warnings.filterwarnings("ignore")

print("""Load the airline passenger dataset and convert the Month
column into datetime format.""")
```

```
Load the airline passenger dataset and convert the Month column into
datetime format.
```

```
data = pd.read_csv(r"D:\Downloads\archive\AirPassengers.csv")

data['Month'] = pd.to_datetime(data['Month'])
data.set_index('Month', inplace=True)

data.head()
```

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

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 144 entries, 1949-01-01 to 1960-12-01
Data columns (total 1 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   #Passengers  144 non-null    int64
dtypes: int64(1)
memory usage: 2.2 KB
```

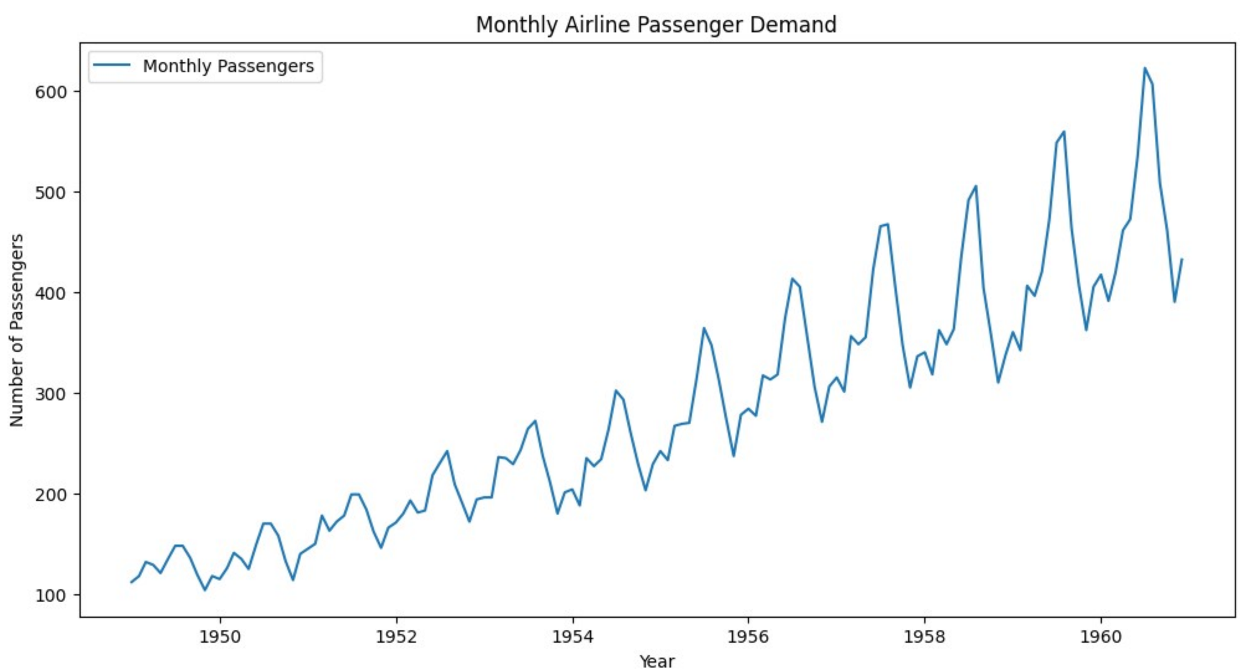
```
data.describe()
```

	#Passengers
count	144.000000
mean	280.298611
std	119.966317
min	104.000000
25%	180.000000
50%	265.500000
75%	360.500000
max	622.000000

```
print("""Plot the original time series to identify trend and
seasonality.""")
```

```
Plot the original time series to identify trend and seasonality.
```

```
plt.figure(figsize=(12,6))
plt.plot(data['#Passengers'], label='Monthly Passengers')
plt.title("Monthly Airline Passenger Demand")
plt.xlabel("Year")
plt.ylabel("Number of Passengers")
plt.legend()
plt.show()
```




```
print(""" Check whether the time series is stationary.""")
```

```
 Check whether the time series is stationary.
```

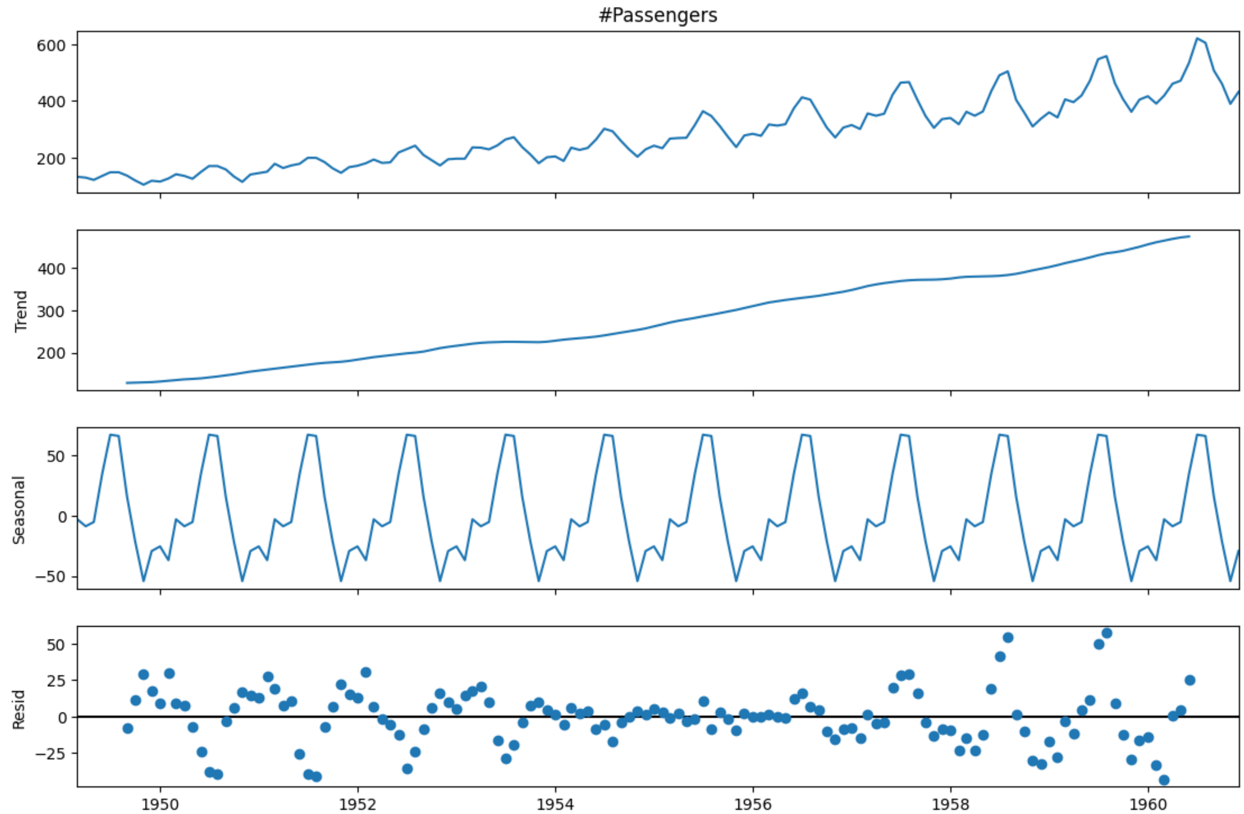
```python
def adf_test(series):
    result = adfuller(series)
    print("ADF Statistic:", result[0])
    print("p-value:", result[1])
    for key, value in result[4].items():
        print(f"Critical Value ({key}): {value}")

adf_test(data['#Passengers'])
```

```
ADF Statistic: 0.8153688792060463
p-value: 0.991880243437641
Critical Value (1%): -3.4816817173418295
Critical Value (5%): -2.8840418343195267
Critical Value (10%): -2.578770059171598
```

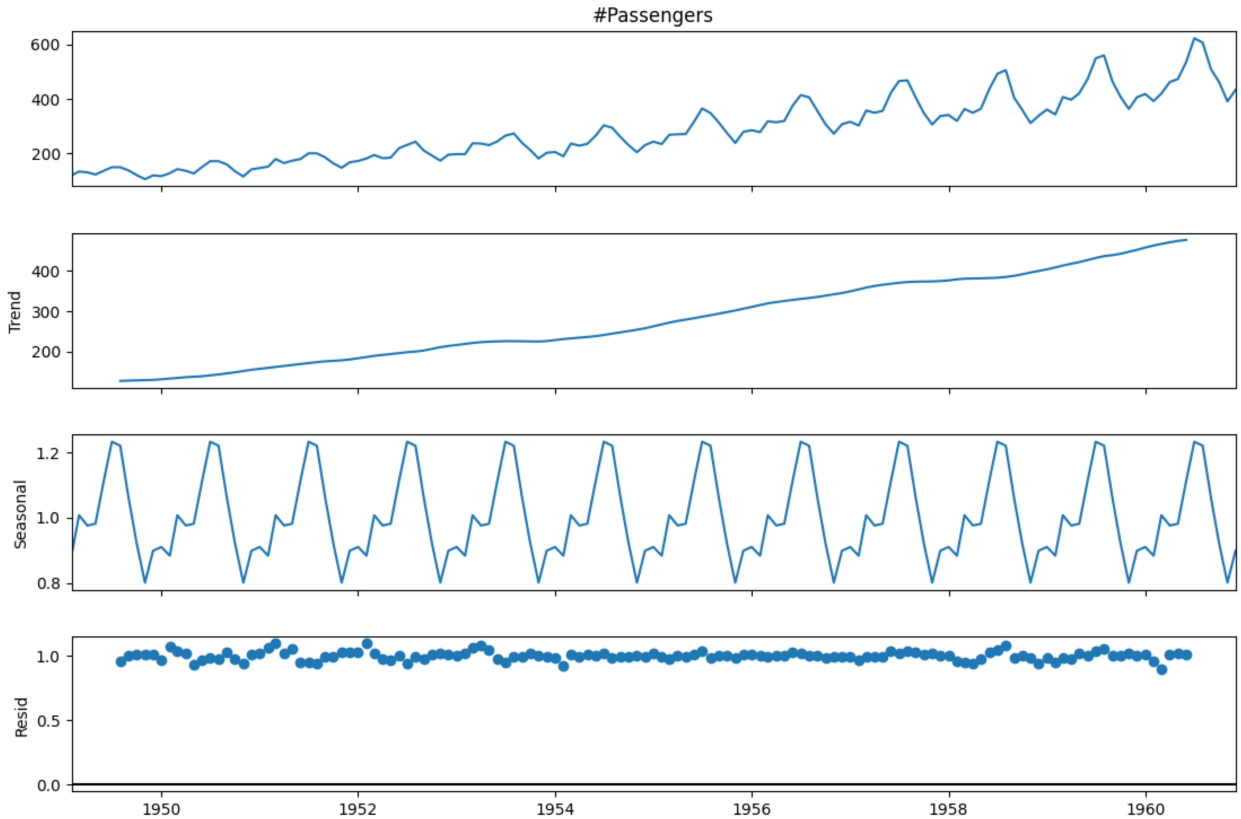
```python
print("""Apply differencing to remove trend and make the series
stationary.""")
```

```
Apply differencing to remove trend and make the series stationary.
```

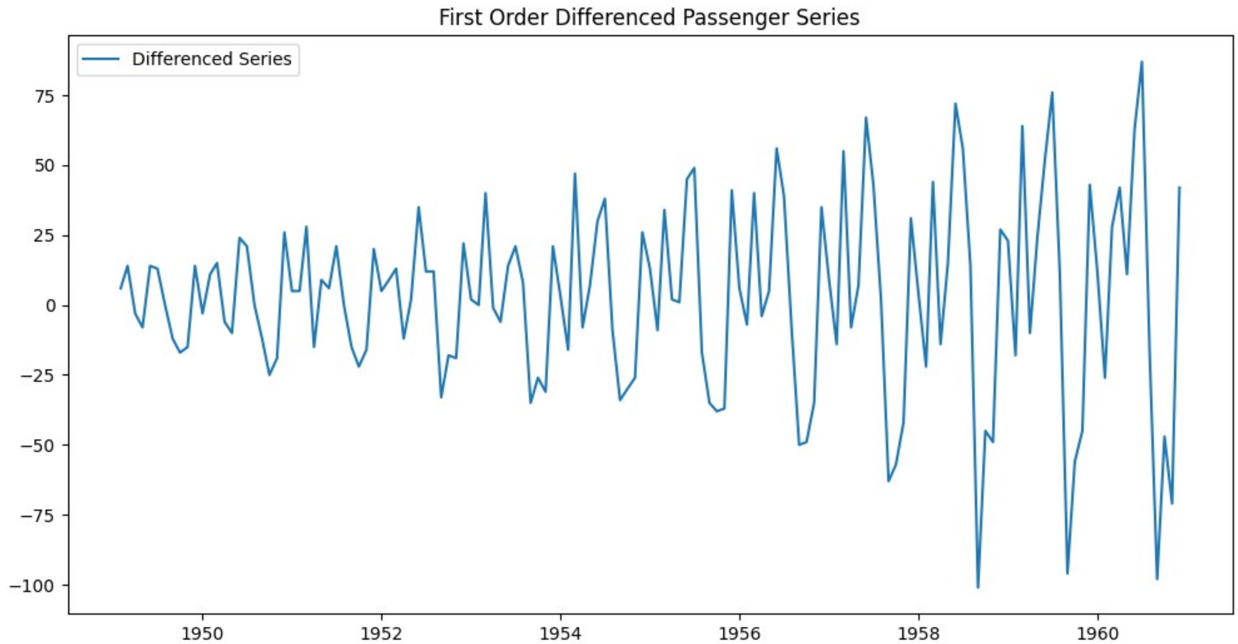
```python
from statsmodels.tsa.seasonal import seasonal_decompose

additive_decomposition = seasonal_decompose(
    data['#Passengers'],
    model='additive',
    period=12
)

fig = additive_decomposition.plot()
fig.set_size_inches(12, 8)
plt.show()
```




```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition = seasonal_decompose(
    data['#Passengers'],
    model='multiplicative',
    period=12
)

fig = decomposition.plot()
fig.set_size_inches(12, 8)
plt.show()
```




```
data['Passengers_diff'] = data['#Passengers'].diff()
data.dropna(inplace=True)

plt.figure(figsize=(12,6))
plt.plot(data['Passengers_diff'], label='Differenced Series')
plt.title("First Order Differenced Passenger Series")
plt.legend()
plt.show()
```




```
print("""Confirm stationarity after differencing.""")
```

```
Confirm stationarity after differencing.
```

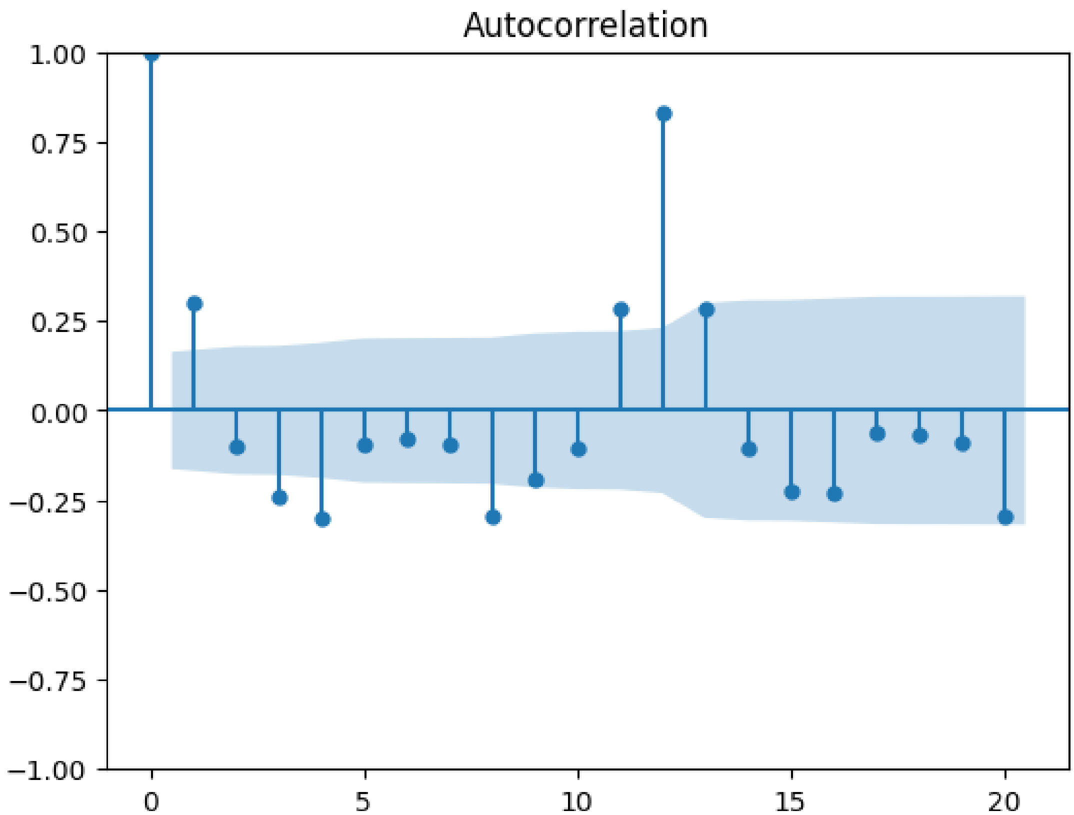
```
adf_test(data['Passengers_diff'])
```

```
ADF Statistic: -2.8292668241699923
p-value: 0.054213290283826474
Critical Value (1%): -3.4816817173418295
Critical Value (5%): -2.8840418343195267
Critical Value (10%): -2.578770059171598
```

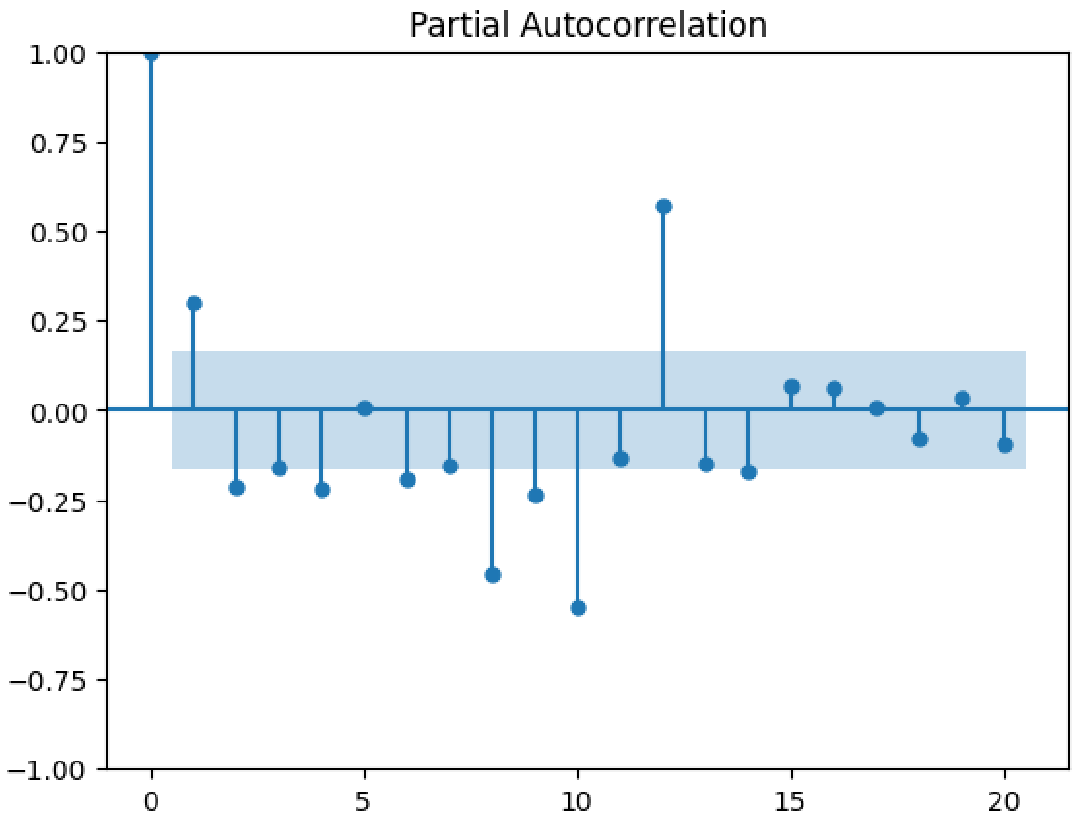
```
print("""Identify AR (p) and MA (q) values.""")
```

```
Identify AR (p) and MA (q) values.
```

```
plot_acf(data['Passengers_diff'], lags=20)
plt.show()
```




```
plot_pacf(data['Passengers_diff'], lags=20)
plt.show()
```




```
print("""Chosen Parameters:

p = 1

d = 1

q = 1""")
```

```
Chosen Parameters:

p = 1

d = 1

q = 1
```

```
print("""Split data to evaluate forecasting accuracy.""")
```

```
Split data to evaluate forecasting accuracy.
```

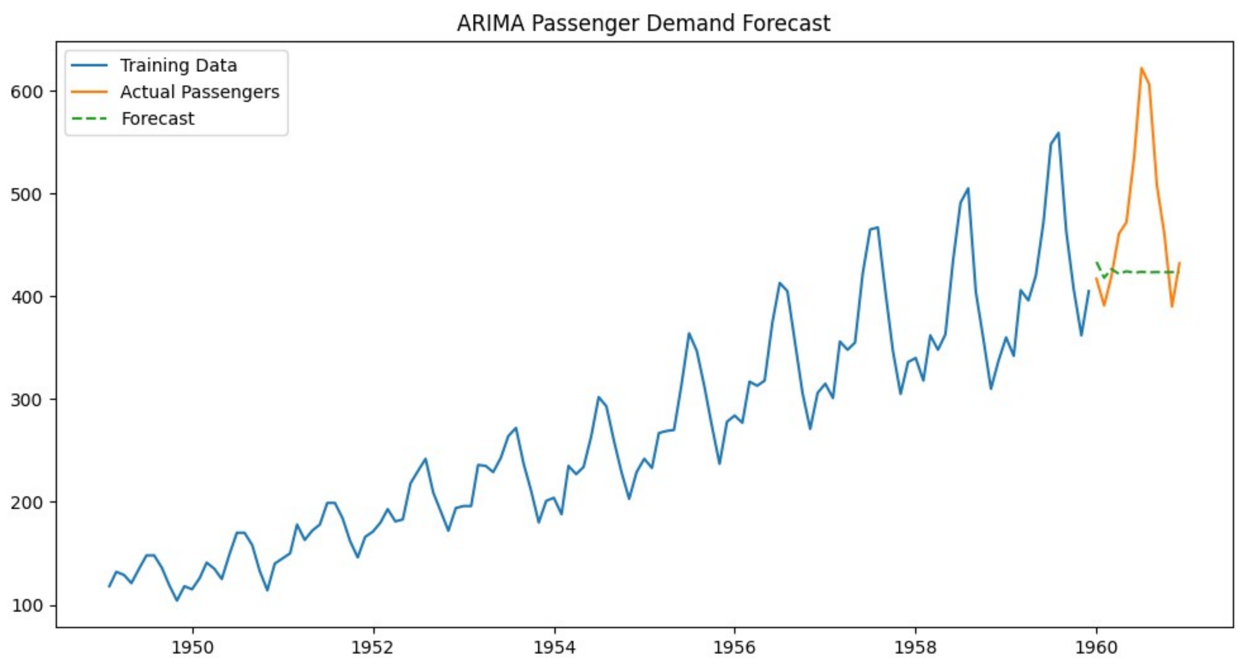
```
train = data['#Passengers'][:-12]
test = data['#Passengers'][-12:]


print("""Train ARIMA(1,1,1) model using training data.""")
```

Train ARIMA(1,1,1) model using training data.

```
model = ARIMA(train, order=(1,1,1))
model_fit = model.fit()

model_fit.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results                                
==============================================================================
========
Dep. Variable:            #Passengers   No. Observations:                  
131
Model:                 ARIMA(1, 1, 1)   Log Likelihood                
-621.194
Date:                Fri, 06 Feb 2026   AIC                           
1248.388
Time:                        14:08:41   BIC                           
1256.990
Sample:                    02-01-1949   HQIC                          
1251.883
                         - 12-01-1959                                         

Covariance Type:                  opg                                         

==============================================================================
========
                 coef    std err          z      P>|z|      [0.025      
0.975]
------------------------------------------------------------------------------
--------
ar.L1         -0.5454      0.100     -5.461      0.000      -0.741      
-0.350
ma.L1          0.9290      0.050     18.528      0.000       0.831       
1.027
sigma2       822.0448     96.591      8.511      0.000     632.730    
1011.360
==============================================================================
=============
Ljung-Box (L1) (Q):                   0.32   Jarque-Bera (JB):               
2.39
Prob(Q):                              0.57   Prob(JB):                       
0.30
Heteroskedasticity (H):               6.79   Skew:                          
-0.27
Prob(H) (two-sided):                  0.00   Kurtosis:                       
3.40
==============================================================================
```

```
=============

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

print("""Generate forecasts and compare with actual values.""")

Generate forecasts and compare with actual values.

forecast = model_fit.forecast(steps=12)
forecast.index = test.index

plt.figure(figsize=(12,6))
plt.plot(train, label='Training Data')
plt.plot(test, label='Actual Passengers')
plt.plot(forecast, label='Forecast', linestyle='--')
plt.legend()
plt.title("ARIMA Passenger Demand Forecast")
plt.show()
```




```
print("""Evaluate model performance using MAE and RMSE.""")

Evaluate model performance using MAE and RMSE.

mae = mean_absolute_error(test, forecast)
rmse = np.sqrt(mean_squared_error(test, forecast))
```

```
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Mean Absolute Error (MAE): 66.23731104483657
Root Mean Squared Error (RMSE): 91.21414644331603
```

```
print("""Forecast passenger demand for the next 12 months.""")
```

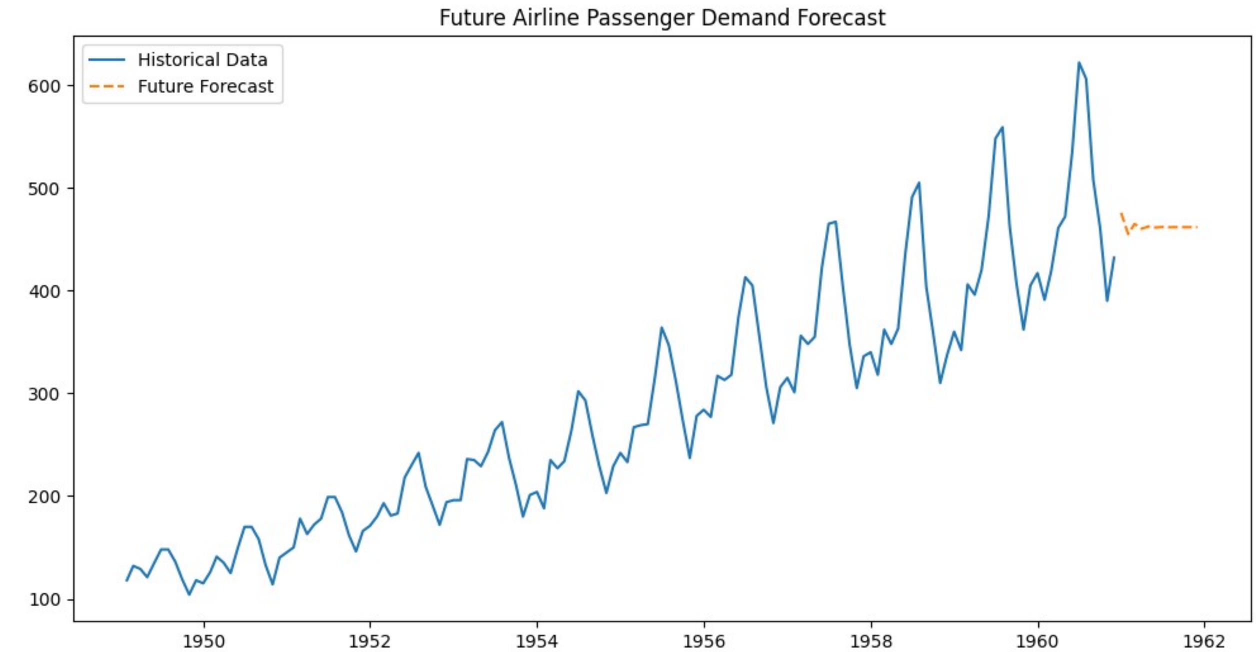
```
Forecast passenger demand for the next 12 months.
```

```
final_model = ARIMA(data['#Passengers'], order=(1,1,1))
final_model_fit = final_model.fit()
```

```
future_forecast = final_model_fit.forecast(steps=12)
future_forecast
```

```
1961-01-01    475.785430
1961-02-01    454.965300
1961-03-01    464.865347
1961-04-01    460.157839
1961-05-01    462.396276
1961-06-01    461.331891
1961-07-01    461.838010
1961-08-01    461.597349
1961-09-01    461.711784
1961-10-01    461.657369
1961-11-01    461.683244
1961-12-01    461.670940
Freq: MS, Name: predicted_mean, dtype: float64
```

```
plt.figure(figsize=(12,6))
plt.plot(data['#Passengers'], label='Historical Data')
plt.plot(future_forecast, label='Future Forecast', linestyle='--')
plt.legend()
plt.title("Future Airline Passenger Demand Forecast")
plt.show()
```


Future Airline Passenger Demand Forecast
Historical Data
Future Forecast
600
500
400
300
200
100
1950
1952
1954
1956
1958
1960
1962