In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from  statsmodels.tsa.seasonal import seasonal_decompose
from  statsmodels.tsa.seasonal import STL
from statsmodels.tsa.stattools import adfuller
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Extract the data into Excel files
data=pd.read_excel("C:/Users/Asus/Downloads/Airline_Passenger1.xlsx")
```

In [ ]:

In [9]:
```
data.shape
```

Out[9]: (105, 2)

In [10]:
```
#Top head
data.head()
```

Out[10]:

	Date	Passengers
0	2016-01-01	2815
1	2016-02-01	2672
2	2016-03-01	2755
3	2016-04-01	2721
4	2016-05-01	2946

In [7]:
```
#Set the 'Date' column as the index.
data.set_index('Date', inplace=True)
```

In [26]:
```
data.head()
```

Out[26]:

	Passengers
Date	
2016-01-01	2815
2016-02-01	2672
2016-03-01	2755
2016-04-01	2721
2016-05-01	2946

In [13]:
```
data.columns
```

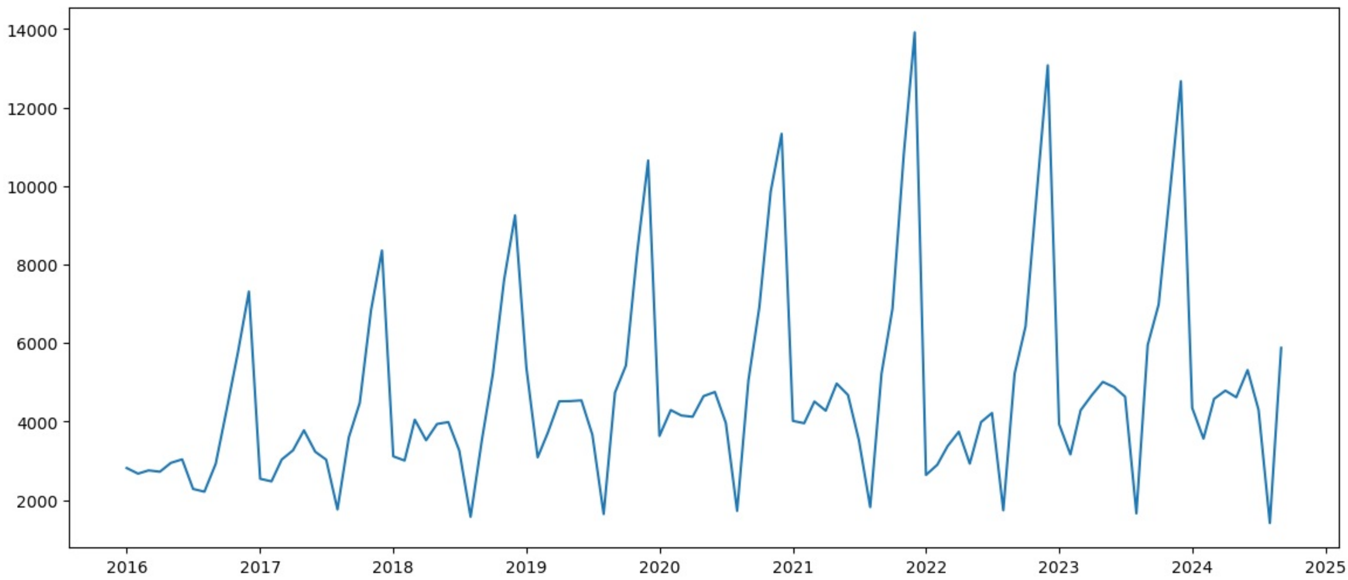
Out[13]: Index(['Passengers'], dtype='object')

In [18]:
```
data.describe(include='all')
```

Out[18]:

	Date	Passengers
count	105	105.000000
mean	2020-05-01 10:30:51.428571392	4761.152381
min	2016-01-01 00:00:00	1413.000000
25%	2018-03-01 00:00:00	3113.000000
50%	2020-05-01 00:00:00	4217.000000
75%	2022-07-01 00:00:00	5221.000000
max	2024-09-01 00:00:00	13916.000000
std	NaN	2553.502601

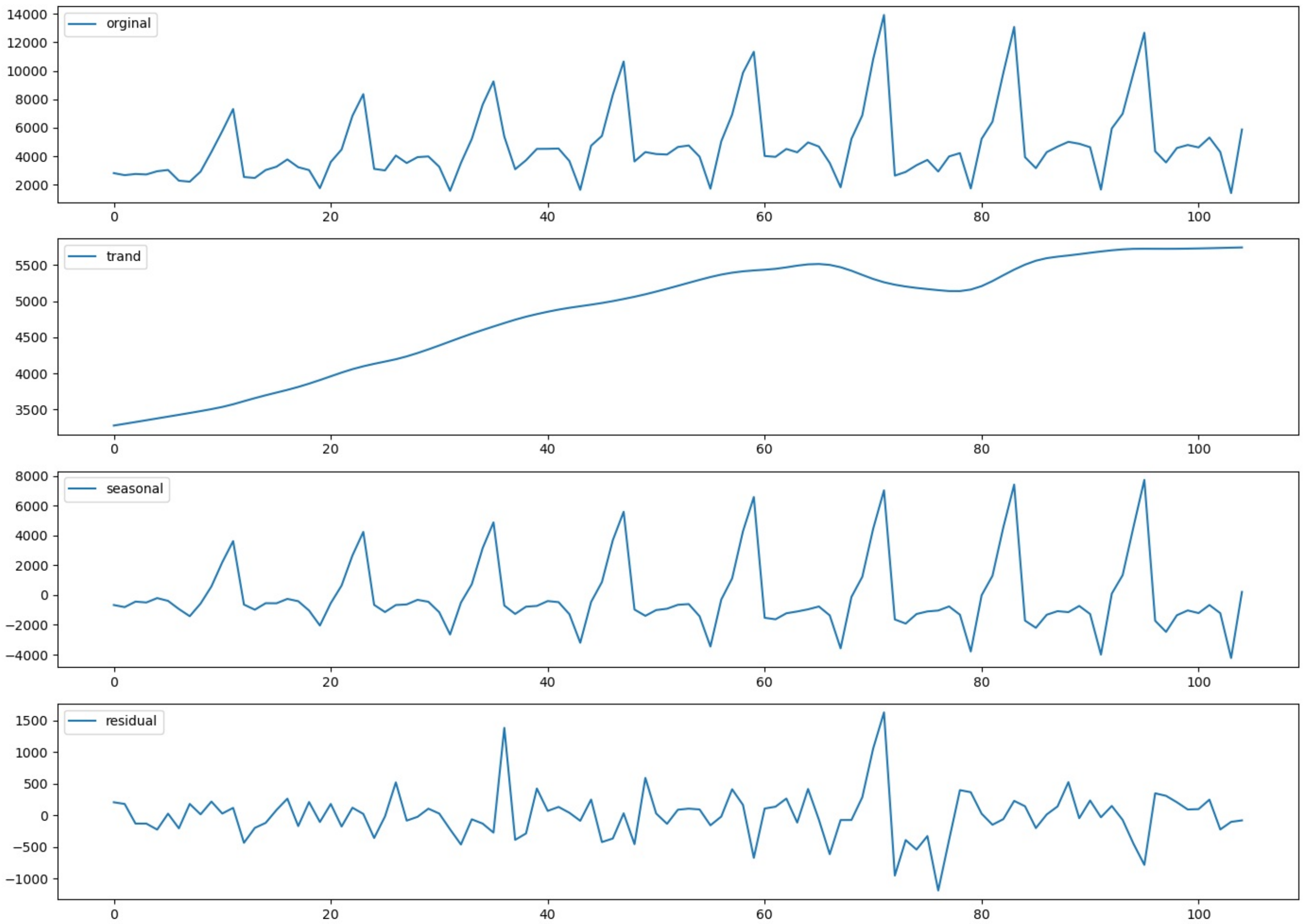
In [27]:
```
# Identified trends in passenger numbers across years
plt.figure(figsize=(14,6))
plt.plot(data.index,data['Passengers'])
plt.show()
```



In [ ]:

In [24]:

```python
stl=STL(data['Passengers'],period=12)

result=stl.fit()

plt.figure(figsize=(14,10))
plt.subplot(411)
plt.plot(result.observed,label='orginal')
plt.legend(loc='upper left')
plt.subplot(412)
plt.plot(result.trend,label='trand')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(result.seasonal,label='seasonal')
plt.legend(loc='upper left')
plt.subplot(414)
plt.plot(result.resid,label='residual')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



- The STL decomposition confirms strong long-term growth in airline passenger demand
  - Trend- Trend shows steady long-term growth in passengers.
  - Seasonality-Seasonality reveals consistent yearly peaks and troughs.
  - Residuals- Residuals capture small random fluctuations, mostly near zero

In [ ]:

In [ ]:

- Test the time series for stationarity with ADFuller and KPSS

In [11]:
```
test=adfuller(data['Passengers'])

print(f'adf stats:{round(test[0],3)}')
print(f'p_valu:{round(test[1],3)}')
print(f'critical values')
for key,value in test[4].items():#critical values
  print(f'{key}:{round(value,3)}')
if test[1]<=0.05:
  print(f'data is stationary')
else:
  print(f'data is non stationary')
```

```
adf stats:-1.834
p_valu:0.364
critical values
1%:-3.503
5%:-2.893
10%:-2.584
data is non stationary
```

- Series of data is not stationary

In [ ]:

- The data was not stationary, so it needs to be transformed before forecasting

In [4]:
```
data['Passenger1']=data['Passengers']-data['Passengers'].shift(12)
data
```

Out[4]:

	Date	Passengers	Passenger1
0	2016-01-01	2815	NaN
1	2016-02-01	2672	NaN
2	2016-03-01	2755	NaN
3	2016-04-01	2721	NaN
4	2016-05-01	2946	NaN
...	...	...	...
100	2024-05-01	4618	-392.0
101	2024-06-01	5312	438.0
102	2024-07-01	4298	-335.0
103	2024-08-01	1413	-246.0
104	2024-09-01	5877	-74.0

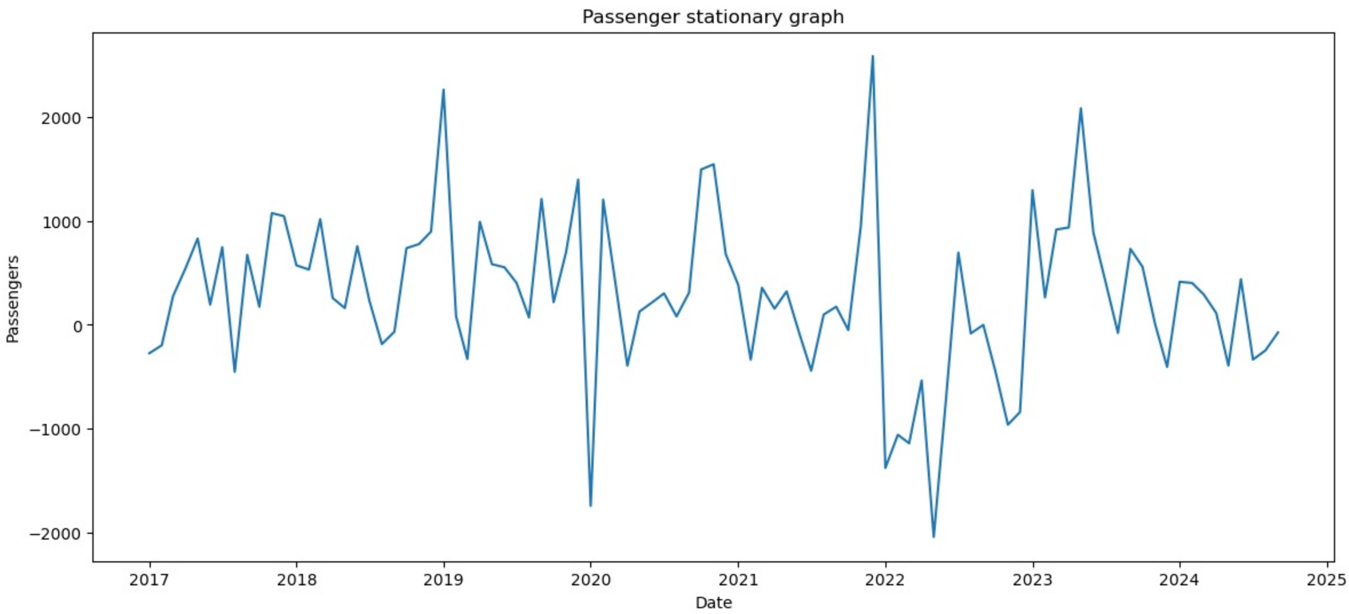
105 rows × 3 columns

In [5]:
```
stationary=adfuller(data['Passenger1'].dropna())

print(f'adf stats:{round(stationary[0],3)}')
print(f'p_valu:{round(stationary[1],3)}')
print(f'critical values')
for key,value in stationary[4].items():#critical values
  print(f'{key}:{round(value,3)}')
if stationary[1]<=0.05:
  print(f'data is stationary')
else:
  print(f'data is non stationary')
```

```
adf stats:-7.627
p_valu:0.0
critical values
1%:-3.504
5%:-2.894
10%:-2.584
data is stationary
```

- Applied seasonal differencing of order 12 to achieve stationarity by removing yearly seasonality in the monthly passenger data.

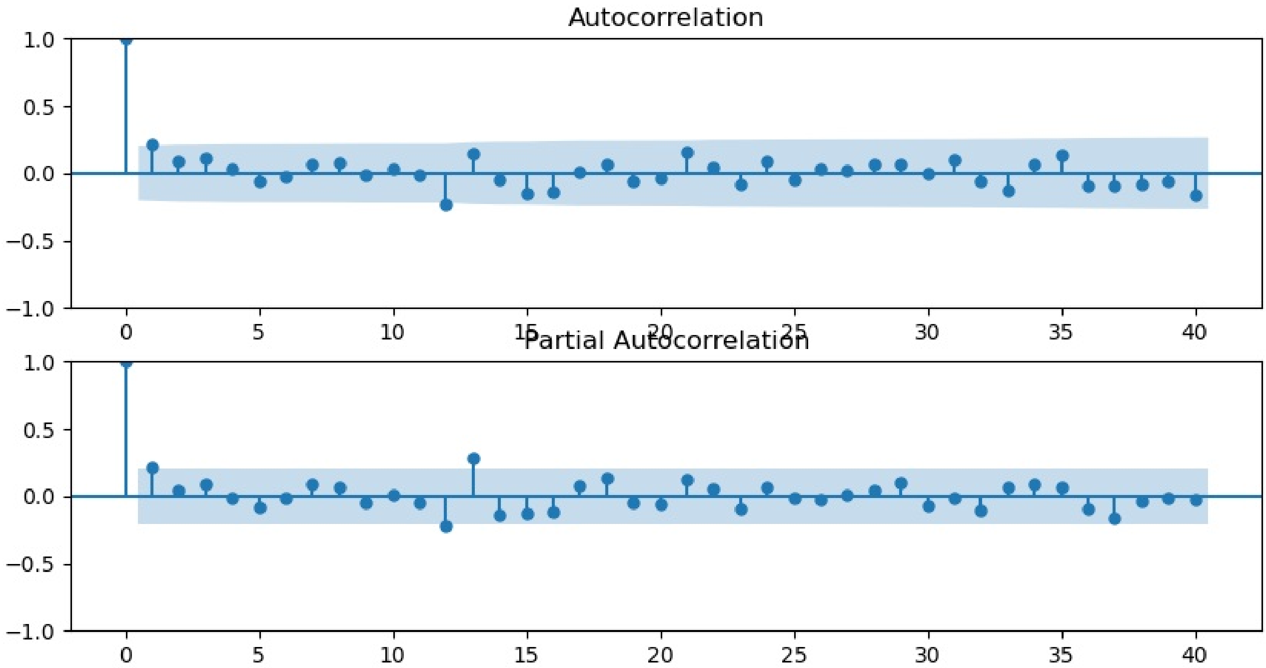
In [11]:
```
# see passenger Graph After Stationarity Transformation
plt.figure(figsize=(14,6))
plt.plot(data.index,data['Passenger1'])
plt.xlabel('Date')
plt.title('Passenger stationary graph')
plt.ylabel('Passengers')
plt.show()
```



In [ ]:

Final Steps on Autocorrelation and Partial Autocorrelation

In [32]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
import statsmodels.api as sm


fig = plt.figure(figsize=(10,5))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(data['Passenger1'].dropna(),lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(data['Passenger1'].dropna(),lags=40,ax=ax2)
```



- ▪ The ACF plot shows a gradual geometric decay or sine-wave pattern after the first few lags.
  - The PACF plot typically shows a sharp cutoff after lag 1 (or a small number of lags)
    - Since airline passenger data is known to have 12-month seasonality, you will also see strong spikes at lag 12, 24, etc., in both ACF and PACF
      - Based on the plots, a model like SARIMA(p,d,q)(P,D,Q,12) is usually appropriate, with p chosen from PACF cutoff and q from ACF cutoff.
      - p=1, d=1, q=0 or 1

In [ ]:

```

```

In [ ]:

```
#p=1, d=1, q=0 or 1
```

Forcasting

In [21]:

```
x=data['Passenger1'].dropna()
```

In [33]:

```
#split stationary data
train_data , test_data=x[:-30],x[-30:]
```

In [37]:

```
x
```

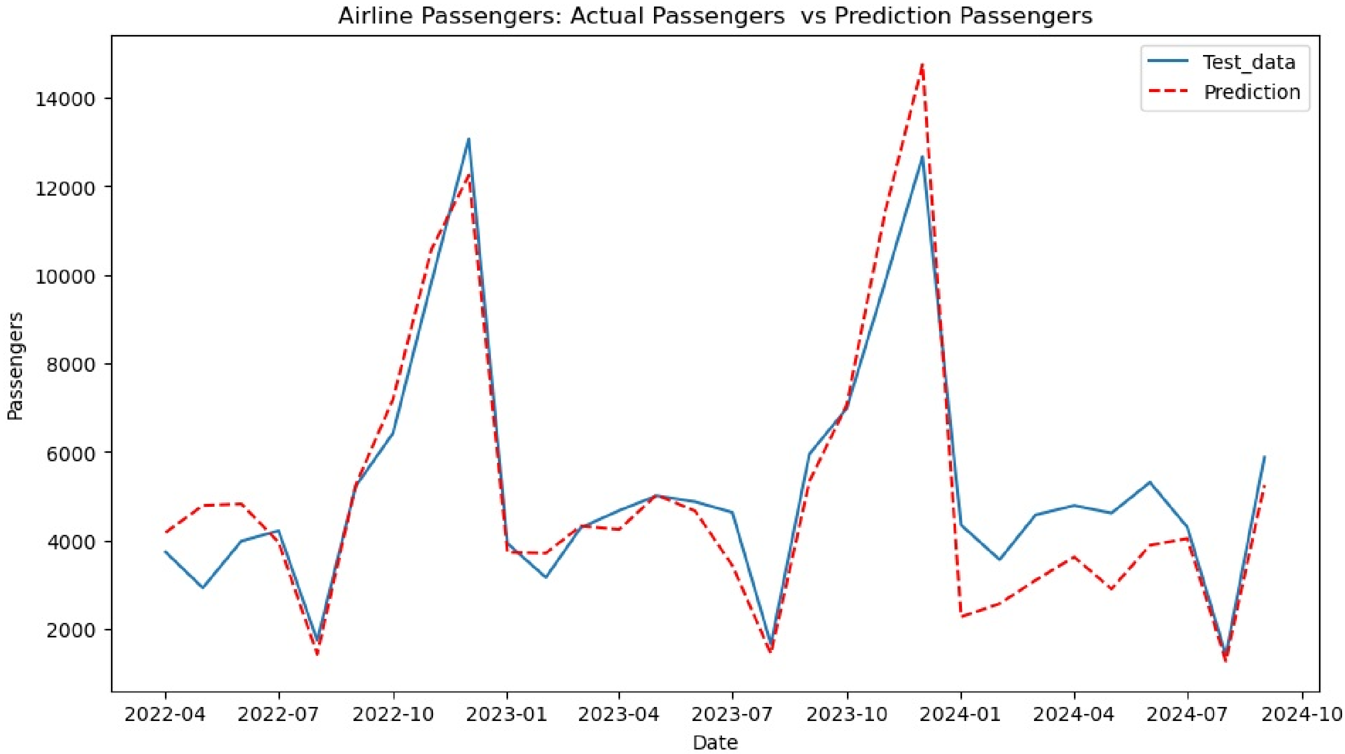
Out[37]:

```
Date
2017-01-01   -274.0
2017-02-01   -197.0
2017-03-01    276.0
2017-04-01    545.0
2017-05-01    830.0
              ...
2024-05-01   -392.0
2024-06-01    438.0
2024-07-01   -335.0
2024-08-01   -246.0
2024-09-01    -74.0
Name: Passenger1, Length: 93, dtype: float64
```

In [31]:

```
##see the rms score and ploting graph how mouch corect train and test data

from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error

model=SARIMAX(train_data,order=(0, 1, 1),seasonal_order=(0,1,1,12))
model_fit=model.fit()


prediction=model_fit.predict(start=len(train_data),
                              end=len(train_data) + len(test_data)-1,dynamic=False)

predictions_original = data['Passengers'].iloc[len(train_data)-12:len(train_data)+len(test_data)-12] + predicti
actual_test_data = data['Passengers'].iloc[-30:]
```

```
##ploting
plt.figure(figsize=(11,6))
plt.plot(actual_test_data.index , actual_test_data  , label='Test_data')
plt.plot(actual_test_data.index,predictions_original, c='r',linestyle='--',label='Prediction')
plt.title('Airline Passengers: Actual Passengers  vs Prediction Passengers')
plt.xlabel('Date')
plt.ylabel('Passengers')
plt.legend()
plt.show()

rmse=round(np.sqrt(mean_squared_error(test_data,prediction)),2)
print('RMSE:',rmse)
```




```
RMSE: 954.69
```

- Conclusion:

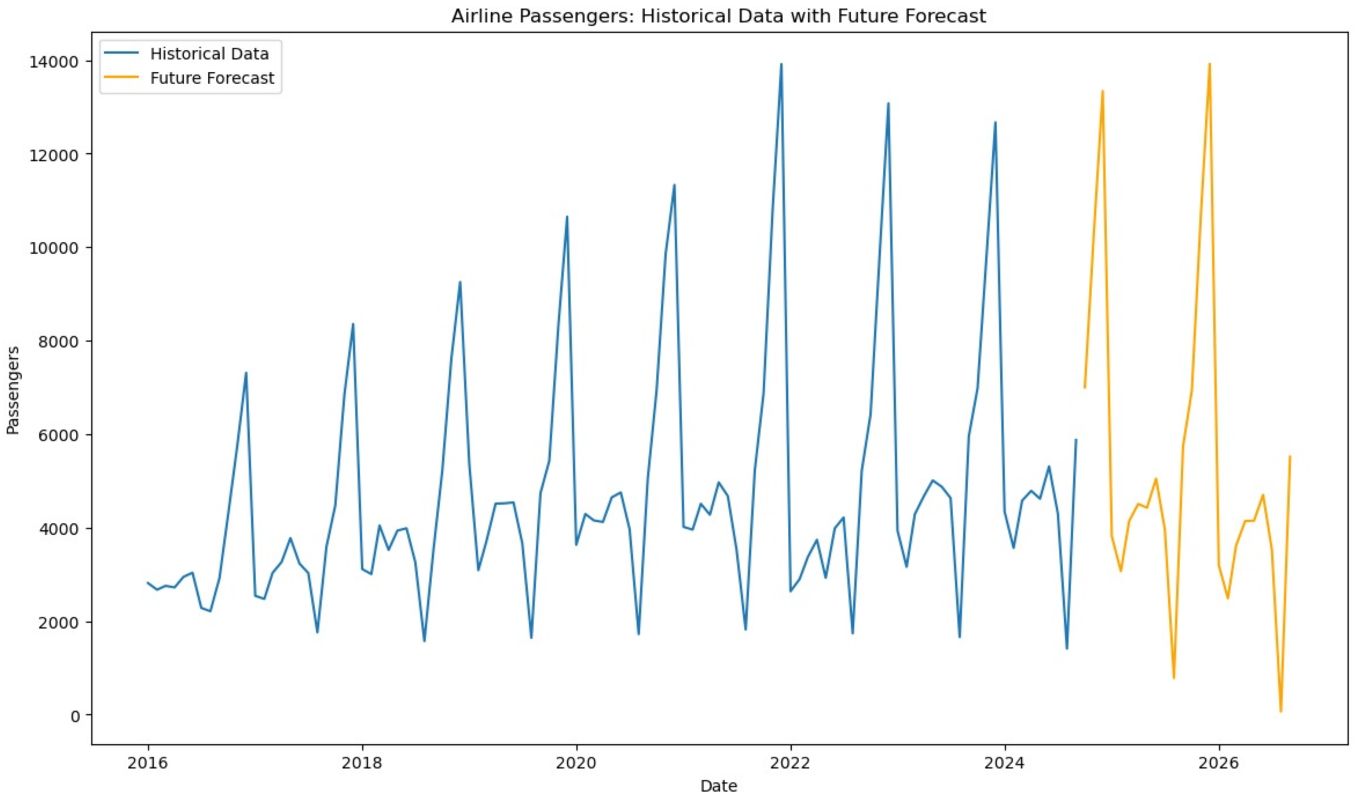
  The model's accuracy was evaluated using RMSE, which measures the average error between actual and predicted values. A lower RMSE indicates better performance. The comparison plot shows that the forecast closely follows the actual trend and seasonality, confirming the model's reliability.

In [ ]:

last step forcasting future in 24 months

In [35]:

```
# Generate future dates and placeholder DataFrame
future_dates = [data.index[-1] + pd.DateOffset(months=x) for x in range(1, 25)]

# Predict differenced values for the future
diff_preds = model_fit.predict(start=len(data), end=len(data) + len(future_dates) - 1, dynamic=True)

# Reverse differencing in a compact way
forecasted = []
for i, diff in enumerate(diff_preds):
    last_year_val = data['Passengers'].iloc[-12 + i] if i < 12 else forecasted[i - 12]
    forecasted.append(last_year_val + diff)

# Create forecast series
forecast_series = pd.Series(forecasted, index=future_dates)

# Plot
plt.figure(figsize=(14, 8))
plt.plot(data['Passengers'], label='Historical Data')
plt.plot(forecast_series, color='orange', label='Future Forecast')
plt.title('Airline Passengers: Historical Data with Future Forecast')
plt.xlabel('Date')
plt.ylabel('Passengers')
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js