Importing necessary libraries

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

Loading the dataframe

```
df= pd.read_csv("AirPassengers.csv")
df
```

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121
...	...	...
139	1960-08	606
140	1960-09	508
141	1960-10	461
142	1960-11	390
143	1960-12	432

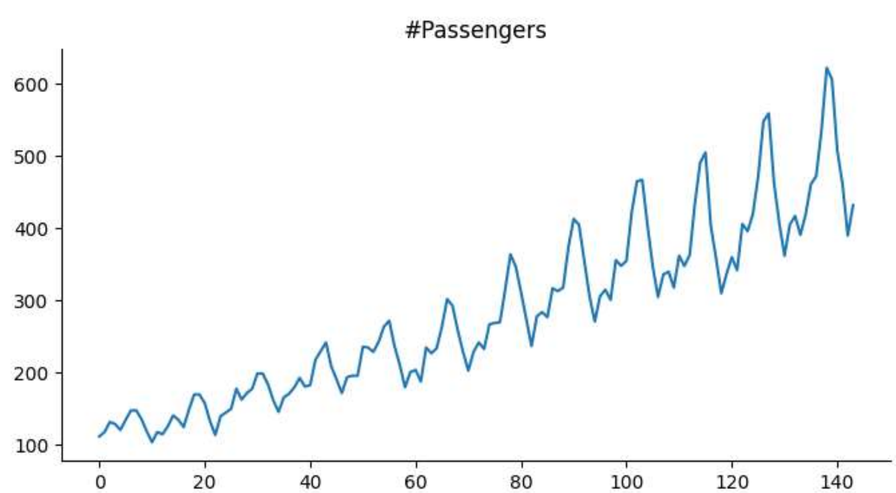
144 rows × 2 columns

Next steps: Generate code with df View recommended plots

Passengers

```
# @title #Passengers

from matplotlib import pyplot as plt
df['#Passengers'].plot(kind='line', figsize=(8, 4), title='#Passengers')
plt.gca().spines[['top', 'right']].set_visible(False)
```



```
df.columns
```

```
Index(['Month', '#Passengers'], dtype='object')
```

```
df["Month"]= pd.to_datetime(df["Month"])
df
```

	Month	#Passengers
0	1949-01-01	112
1	1949-02-01	118
2	1949-03-01	132
3	1949-04-01	129
4	1949-05-01	121
...	...	...
139	1960-08-01	606
140	1960-09-01	508
141	1960-10-01	461
142	1960-11-01	390
143	1960-12-01	432

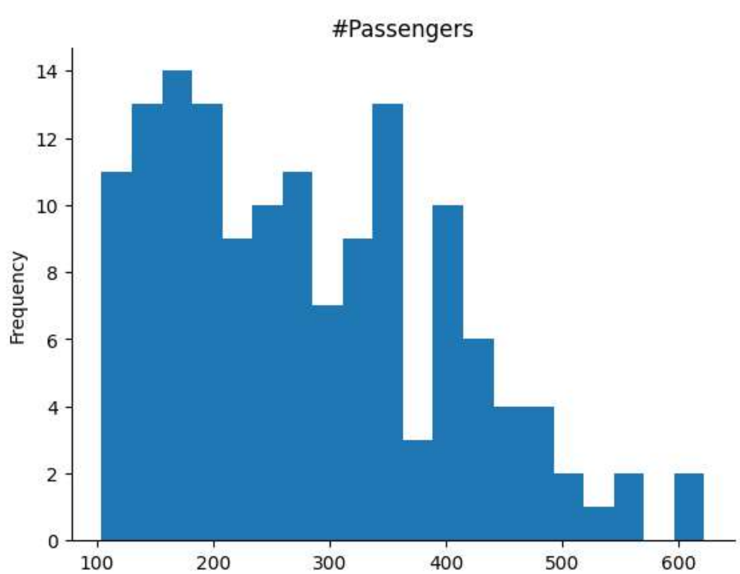
144 rows × 2 columns

Next steps: Generate code with df View recommended plots

Passengers

```
# @title #Passengers

from matplotlib import pyplot as plt
df['#Passengers'].plot(kind='hist', bins=20, title='#Passengers')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
df.columns
```

```
Index(['Month', '#Passengers'], dtype='object')
```

```
df.dtypes
```

```
Month          datetime64[ns]
#Passengers             int64
dtype: object
```

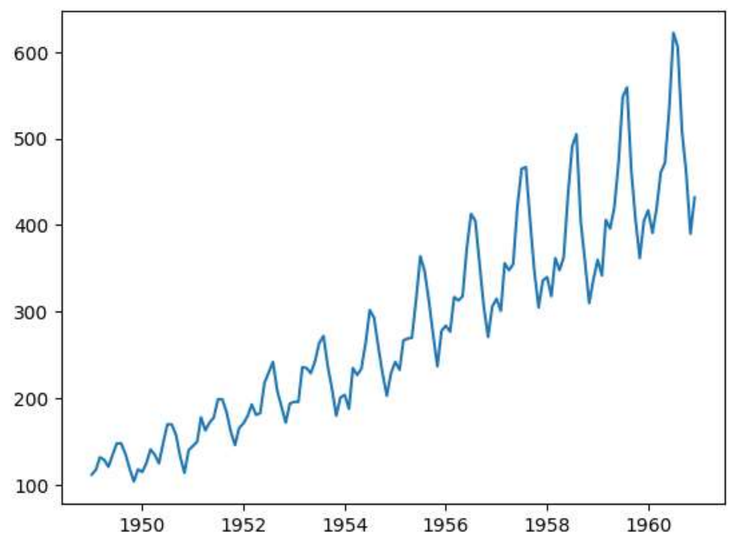
```
df.set_index("Month",inplace=True)
df
```

	#Passengers
Month	
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121
...	...
1960-08-01	606
1960-09-01	508
1960-10-01	461
1960-11-01	390
1960-12-01	432

144 rows × 1 columns

```
plt.plot(df['#Passengers'])
```

```
[<matplotlib.lines.Line2D at 0x7ac647b87670>]
```



After running this code, you can examine the adf, pvalue, and other variables to determine whether the time series df is stationary or not. A small p-value and a test statistic significantly smaller than the critical values indicate that the time series is likely stationary.

```
from statsmodels.tsa.stattools import adfuller
adf,pvalue,usedlag_, nobs_, critical_values, icbest_ = adfuller(df)
```

```
print(pvalue) #if pvalue > 0.05 then data is not stationary
```

```
0.991880243437641
```

Adding a new column named "year" to the DataFrame df, containing the year values extracted from the index. This can be useful for time series analysis, where you may want to analyze data based on the year component.

```
df["year"]= [d.year for d in df.index]
df
```

Month	#Passengers	year
1949-01-01	112	1949
1949-02-01	118	1949
1949-03-01	132	1949
1949-04-01	129	1949
1949-05-01	121	1949
...	...	...
1960-08-01	606	1960
1960-09-01	508	1960
1960-10-01	461	1960
1960-11-01	390	1960
1960-12-01	432	1960

144 rows × 2 columns

Next steps: Generate code with df View recommended plots

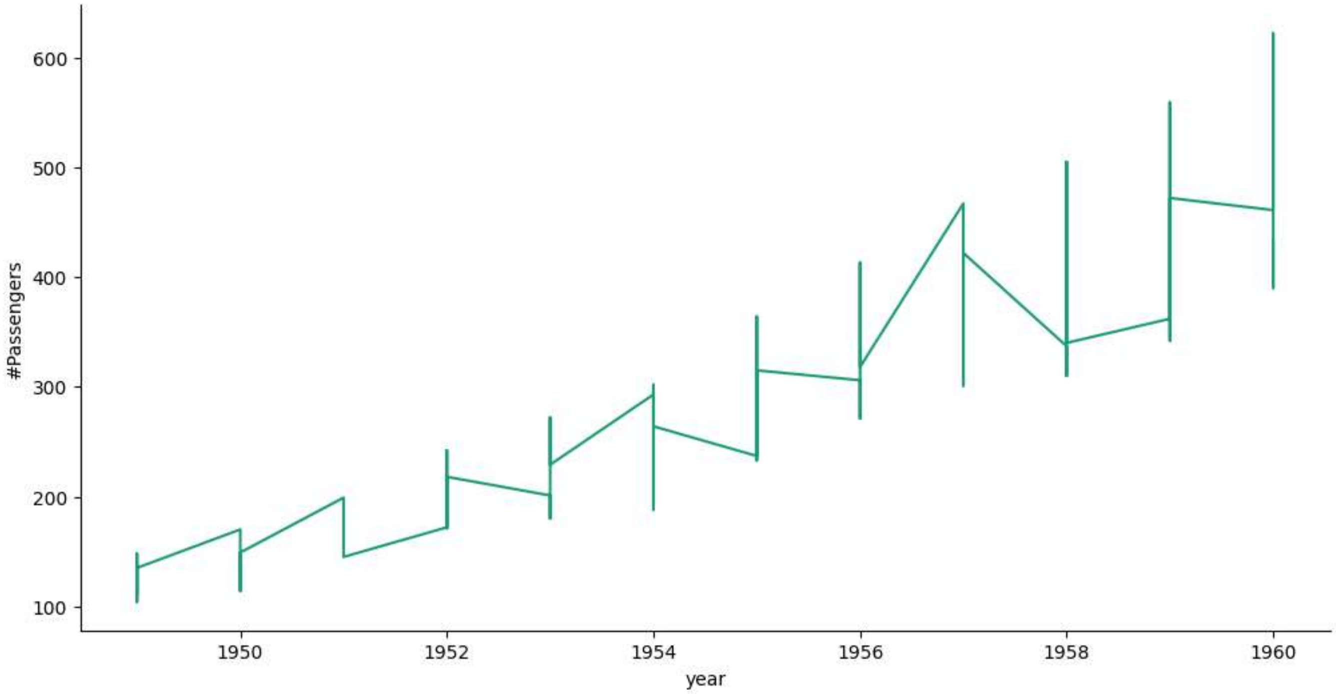
year vs #Passengers

```
# @title year vs #Passengers

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['year']
  ys = series['#Passengers']

  plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = df.sort_values('year', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('year')
_ = plt.ylabel('#Passengers')
```



Adding a new column named "month" to the DataFrame df, containing the month abbreviations extracted from the index. This can be useful for analyzing data based on the month component, such as seasonal patterns or monthly trends.

```
df["month"]= [d.strftime('%b') for d in df.index]
df
```

	#Passengers	year	month
Month			
1949-01-01	112	1949	Jan
1949-02-01	118	1949	Feb
1949-03-01	132	1949	Mar
1949-04-01	129	1949	Apr
1949-05-01	121	1949	May
...	...	...	...
1960-08-01	606	1960	Aug
1960-09-01	508	1960	Sep
1960-10-01	461	1960	Oct
1960-11-01	390	1960	Nov
1960-12-01	432	1960	Dec

144 rows × 3 columns

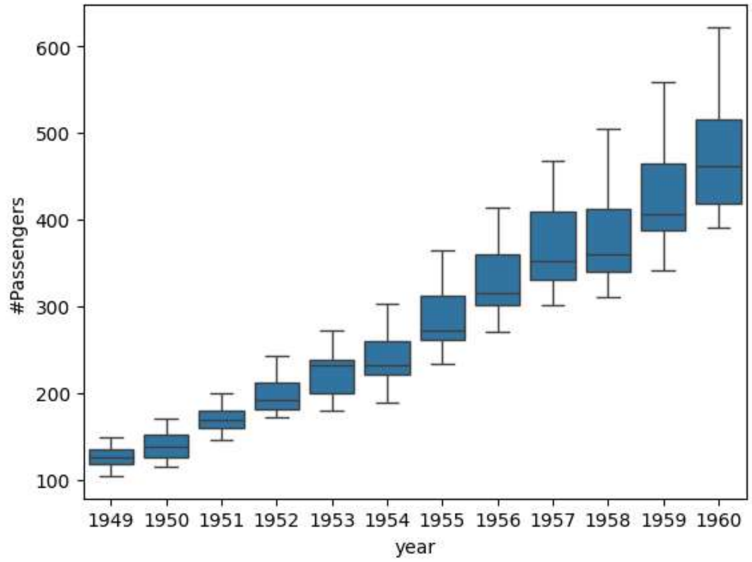
Next steps: Generate code with df View recommended plots

```
years= df['year'].unique()
years
```

```
array([1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959,
       1960])
```

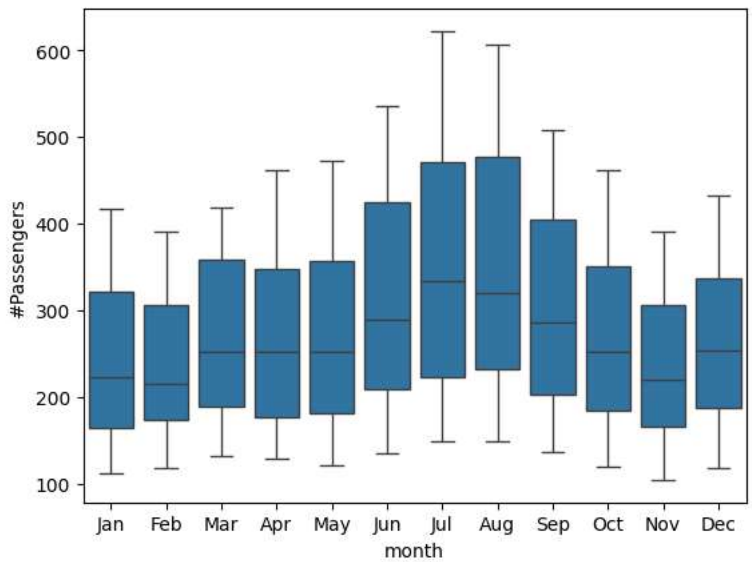
```
sns.boxplot(x='year',y='#Passengers',data=df)
```

```
<Axes: xlabel='year', ylabel='#Passengers'>
```




```
sn.boxplot(x='month', y='#Passengers', data=df)
```

```
<Axes: xlabel='month', ylabel='#Passengers'>
```



After executing this code, the variable decompose will contain the results of the seasonal decomposition, including the trend, seasonal, and residual components

```
from statsmodels.tsa.seasonal import seasonal_decompose
decompose= seasonal_decompose(df['#Passengers'],
                              model='additive'
                              )
```

```
trend= decompose.trend
seasonal=decompose.seasonal
residual=decompose.resid
```

```
trend
```

```
Month
1949-01-01    NaN
1949-02-01    NaN
1949-03-01    NaN
1949-04-01    NaN
1949-05-01    NaN
             ..
```

```
1960-08-01   NaN
1960-09-01   NaN
1960-10-01   NaN
1960-11-01   NaN
1960-12-01   NaN
Name: trend, Length: 144, dtype: float64
```

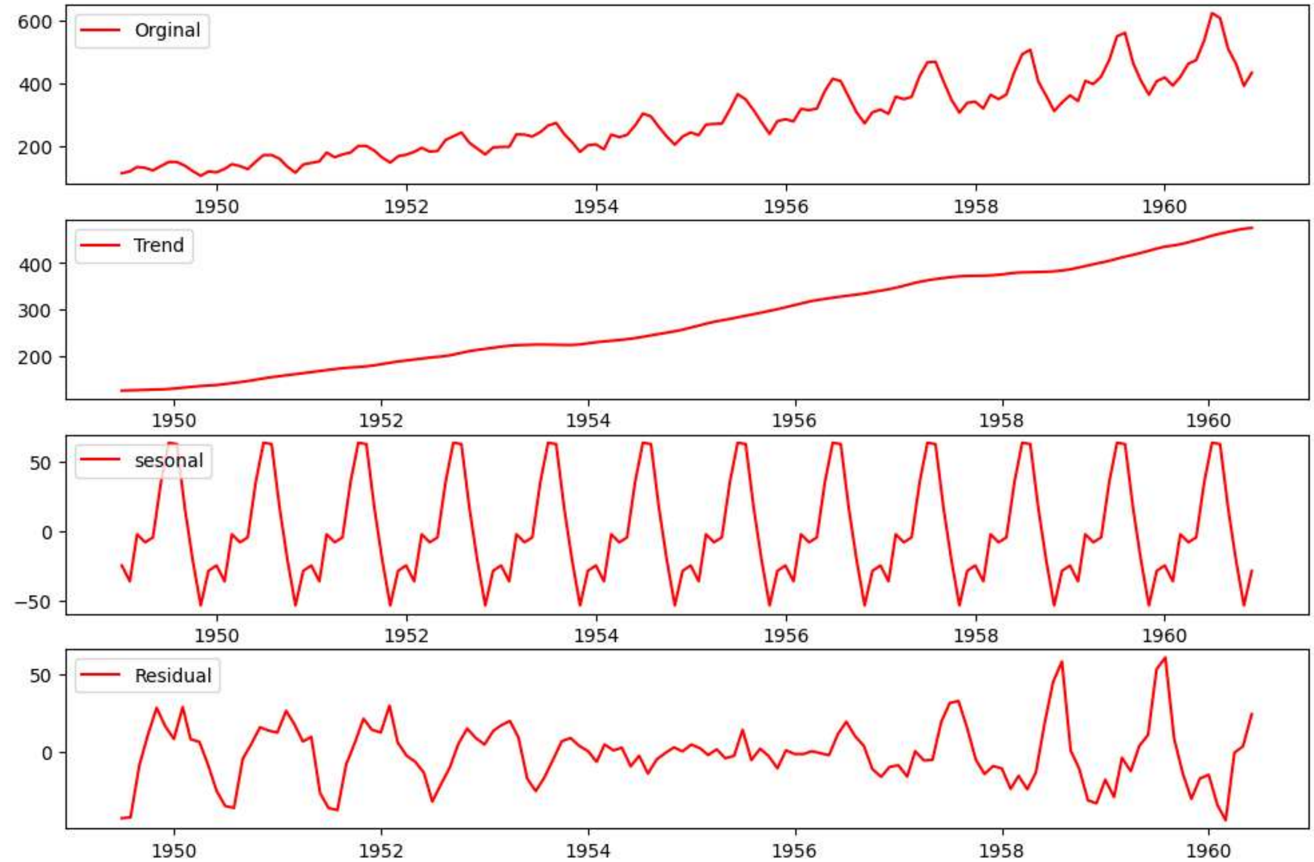
```
seasonal
```

```
Month
1949-01-01   -24.748737
1949-02-01   -36.188131
1949-03-01    -2.241162
1949-04-01    -8.036616
1949-05-01    -4.506313
                ...
1960-08-01    62.823232
1960-09-01    16.520202
1960-10-01   -20.642677
1960-11-01   -53.593434
1960-12-01   -28.619949
Name: seasonal, Length: 144, dtype: float64
```

```
residual
```

```
Month
1949-01-01   NaN
1949-02-01   NaN
1949-03-01   NaN
1949-04-01   NaN
1949-05-01   NaN
              ..
1960-08-01   NaN
1960-09-01   NaN
1960-10-01   NaN
1960-11-01   NaN
1960-12-01   NaN
Name: resid, Length: 144, dtype: float64
```

```
plt.figure(figsize=(12,8))
plt.subplot(411)
plt.plot(df["#Passengers"],label="Orginal",color='red')
plt.legend(loc='upper left')
plt.subplot(412)
plt.plot(trend,label="Trend",color='red')
plt.legend(loc='upper left')
plt.subplot(413)
plt.plot(seasonal,label="sesonal",color='red')
plt.legend(loc='upper left')
plt.subplot(414)
plt.plot(residual,label="Residual",color='red')
plt.legend(loc='upper left')
plt.show()
```




```
!pip install pmdarima
from pmdarima.arima import auto_arima
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 10.1 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.3.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (3.0.8)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.25.2)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.5.3)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.11.4)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (0.14.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (2.0.7)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (67.7.2)
Requirement already satisfied: packaging>=17.1 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (23.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2023.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->pmdarima) (3.2
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmdarima) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.4->statsmodels>=0.13.2->pmdarima) (1.16.
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
```

After executing this code, the arima_model variable will contain the best-fitting ARIMA model selected by the auto_arima function. This model can then be used for forecasting or further analysis of the time series data.

```
arima_model=auto_arima(df["#Passengers"], start_p=1, d=1, q=1,
                       max_p=5, max_d=5, max_q=5, m=12,
                       start_P=0, D=1, start_Q=0, max_P=5, max_D=5, max_Q=5,
                       seasonal=True,
                       trace=True,
                       error_action='ignore',
                       supress_warning=True,
                       stepwise=True, n_fits=50)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,2)(0,1,0)[12]             : AIC=1024.160, Time=0.22 sec
```

```
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=1031.508, Time=0.04 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=1020.393, Time=0.20 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=1021.003, Time=0.24 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=1020.393, Time=0.07 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=1019.239, Time=0.42 sec
 ARIMA(1,1,0)(3,1,0)[12]             : AIC=1020.582, Time=1.23 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=inf, Time=5.71 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=1020.493, Time=1.08 sec
 ARIMA(1,1,0)(3,1,1)[12]             : AIC=inf, Time=10.64 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=1032.120, Time=0.69 sec
 ARIMA(2,1,0)(2,1,0)[12]             : AIC=1021.120, Time=1.21 sec
 ARIMA(1,1,1)(2,1,0)[12]             : AIC=1021.032, Time=1.81 sec
 ARIMA(0,1,1)(2,1,0)[12]             : AIC=1019.178, Time=1.33 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=1020.425, Time=0.38 sec
 ARIMA(0,1,1)(3,1,0)[12]             : AIC=1020.372, Time=2.17 sec
 ARIMA(0,1,1)(2,1,1)[12]             : AIC=inf, Time=7.36 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=1020.327, Time=1.54 sec
 ARIMA(0,1,1)(3,1,1)[12]             : AIC=inf, Time=15.47 sec
 ARIMA(0,1,2)(2,1,0)[12]             : AIC=1021.148, Time=1.11 sec
 ARIMA(1,1,2)(2,1,0)[12]             : AIC=1022.805, Time=0.95 sec
 ARIMA(0,1,1)(2,1,0)[12] intercept   : AIC=1021.017, Time=0.85 sec

Best model:  ARIMA(0,1,1)(2,1,0)[12]
Total fit time: 54.806 seconds
```

```
arima_model.summary()
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	144
Model:	SARIMAX(0, 1, 1)x(2, 1, [], 12)	**Log Likelihood**	-505.589
Date:	Thu, 15 Feb 2024	**AIC**	1019.178
Time:	17:17:37	**BIC**	1030.679
Sample:	01-01-1949 - 12-01-1960	**HQIC**	1023.851
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.3634	0.074	-4.945	0.000	-0.508	-0.219
ar.S.L12	-0.1239	0.090	-1.372	0.170	-0.301	0.053
ar.S.L24	0.1911	0.107	1.783	0.075	-0.019	0.401
sigma2	130.4480	15.527	8.402	0.000	100.016	160.880

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	4.59
Prob(Q):	0.92	**Prob(JB):**	0.10
Heteroskedasticity (H):	2.70	**Skew:**	0.15
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.87

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

The code, x_train will contain the training set data, and x_test will contain the test set data.

```
size=int(len(df)*.66)
x_train, x_test=df[0:size], df[size:len(df)]
```

```
x_train.shape
```

```
(95, 3)
```

```
x_test.shape
```

```
(49, 3)
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

The code, you'll get a summary of the SARIMAX model, which can be used to interpret the results and assess the model's goodness of fit. This summary provides valuable insights into the model's performance and can help in making informed decisions about forecasting.

```
model=SARIMAX(x_train["#Passengers"],
             order=(0,1,1),
             seasonal_order=(2,1,1,12))
result=model.fit()
result.summary()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so
  self._init_dates(dates, freq)
```

SARIMAX Results

Dep. Variable:	#Passengers	**No. Observations:**	95
Model:	SARIMAX(0, 1, 1)x(2, 1, 1, 12)	**Log Likelihood**	-300.269
Date:	Thu, 15 Feb 2024	**AIC**	610.537
Time:	17:17:41	**BIC**	622.571
Sample:	01-01-1949 - 11-01-1956	**HQIC**	615.368
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.3201	0.103	-3.115	0.002	-0.522	-0.119
ar.S.L12	0.6847	0.613	1.116	0.264	-0.517	1.887
ar.S.L24	0.3142	0.127	2.476	0.013	0.066	0.563
ma.S.L12	-0.9812	5.504	-0.178	0.859	-11.769	9.806
sigma2	78.6460	384.747	0.204	0.838	-675.444	832.736

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	2.56
Prob(Q):	0.95	**Prob(JB):**	0.28
Heteroskedasticity (H):	1.69	**Skew:**	0.42
Prob(H) (two-sided):	0.18	**Kurtosis:**	2.83

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

```
start_index=0
end_index=len(x_train)-1
train_prediction=result.predict(start_index, end_index)
train_prediction
```

```
1949-01-01      0.000000
1949-02-01    111.998298
1949-03-01    117.999818
1949-04-01    131.999574
1949-05-01    129.000091
                 ...
1956-07-01    419.543859
1956-08-01    398.687816
1956-09-01    365.414676
1956-10-01    320.670003
1956-11-01    274.819838
Freq: MS, Name: predicted_mean, Length: 95, dtype: float64
```

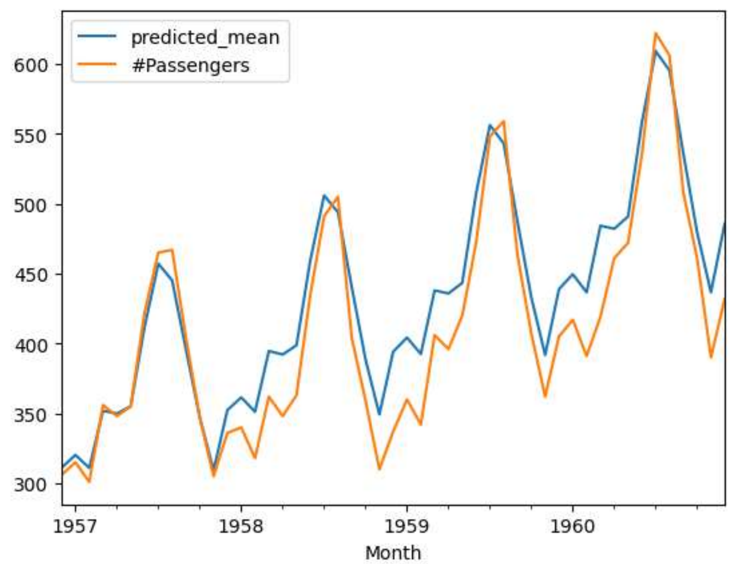
```
st_index=len(x_train)
ed_index=len(df)-1
predction=result.predict(st_index,ed_index)
predction
```

```
1956-12-01    311.113955
1957-01-01    320.267623
1957-02-01    310.945643
1957-03-01    351.862586
1957-04-01    349.886437
1957-05-01    355.071049
1957-06-01    411.895842
1957-07-01    457.099797
1957-08-01    445.091982
1957-09-01    395.832743
1957-10-01    347.111833
1957-11-01    309.227105
1957-12-01    352.333602
1958-01-01    361.447190
1958-02-01    351.163790
1958-03-01    394.593660
1958-04-01    392.118063
1958-05-01    398.685822
1958-06-01    459.531384
1958-07-01    505.841142
1958-08-01    493.916972
```

```
1958-09-01    440.452106
1958-10-01    388.466224
1958-11-01    349.234778
1958-12-01    394.111802
1959-01-01    404.188789
1959-02-01    392.517595
1959-03-01    437.956087
1959-04-01    435.774400
1959-05-01    443.347163
1959-06-01    507.204772
1959-07-01    556.220844
1959-08-01    543.094740
1959-09-01    486.983137
1959-10-01    432.849507
1959-11-01    391.789658
1959-12-01    438.819128
1960-01-01    449.543135
1960-02-01    436.619692
1960-03-01    484.222961
1960-04-01    482.085585
1960-05-01    490.781013
1960-06-01    557.964160
1960-07-01    609.180597
1960-08-01    595.257844
1960-09-01    536.012741
1960-10-01    479.382796
1960-11-01    436.647969
1960-12-01    485.707436
Freq: MS, Name: predicted_mean, dtype: float64
```

```
predction.plot(legend=True)
x_test['#Passengers'].plot(legend=True)
```

```
<Axes: xlabel='Month'>
```




```
import math
from sklearn.metrics import mean_squared_error
```

After executing this code, you'll get the RMSE values for both the training and test sets, which can be used to evaluate the performance of your model. Lower RMSE values indicate better model performance.

```
trainScore=math.sqrt(mean_squared_error(x_train['#Passengers'],train_prediction))
testScore=math.sqrt(mean_squared_error(x_test["#Passengers"],predction))
trainScore,testScore
```

```
(16.142751214053153, 29.49357226833754)
```

```
forcast=result.predict(start=len(df),
                       end=(len(df)-1)+3*12,
                       typ="levels").rename('Forecust')
```

```
plt.figure(figsize=(12,8))
plt.plot(x_train["#Passengers"],label="Training",color='green')
plt.plot(x_test["#Passengers"],label="Test",color='blue')
plt.plot(forcast,label="Forecast",color="red")
plt.legend(loc="upper left")
```

```
<matplotlib.legend.Legend at 0x7ac63beb98d0>
```