Import Modules

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

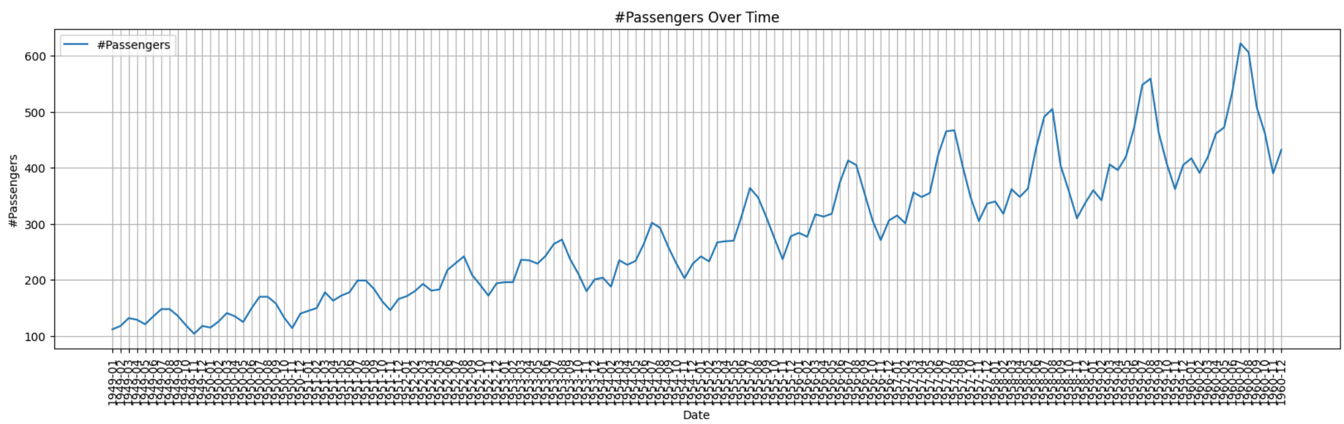
Load the Dataset

```
df = pd.read_csv('./AirPassengers.csv')
df.set_index('Month', inplace=True)
df.head()
```

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

Exploratory Data Analysis

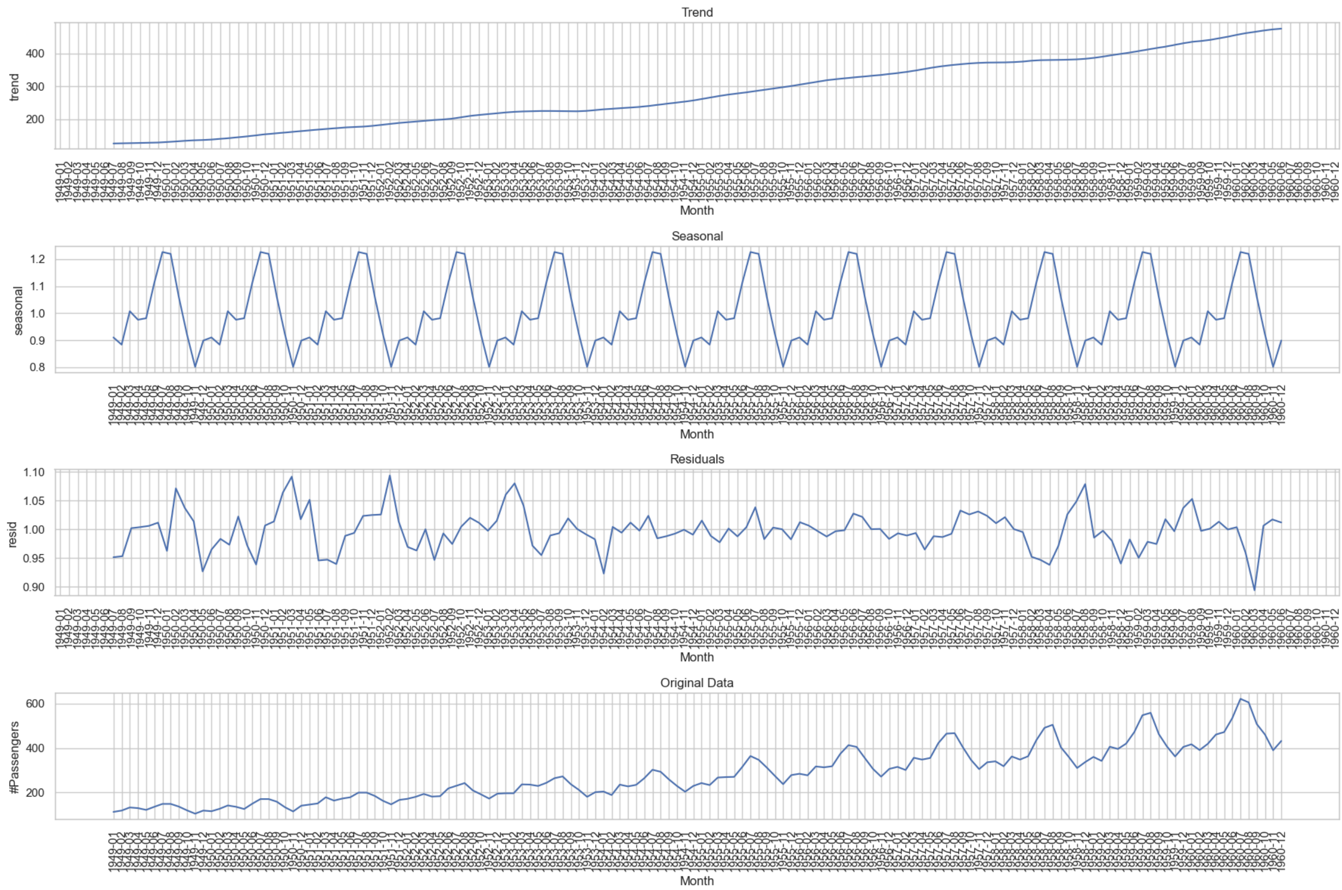
```
plt.figure(figsize=(20, 5))
plt.plot(df.index, df['#Passengers'], label='#Passengers')
plt.xlabel('Date')
plt.ylabel('#Passengers')
plt.title('#Passengers Over Time')
plt.legend()
plt.grid(True)
plt.xticks(rotation=90)
plt.show()
```




```
# perform seasonal decomposition
result = seasonal_decompose(df['#Passengers'], model='multiplicative', period=12)
```

```
# plot the components in the graph
sns.set(style='whitegrid')

plt.figure(figsize=(18,12))
```

```
plt.figure(figsize=(18,12))

# trend component
plt.subplot(411)
sns.lineplot(data=result.trend)
plt.title('Trend')
plt.xticks(rotation=90)

# seasonal component
plt.subplot(412)
sns.lineplot(data=result.seasonal)
plt.title('Seasonal')
plt.xticks(rotation=90)

# Residuals component
plt.subplot(413)
sns.lineplot(data=result.resid)
plt.title('Residuals')
plt.xticks(rotation=90)

# Original data
plt.subplot(414)
sns.lineplot(data=df['#Passengers'])
plt.title('Original Data')
plt.xticks(rotation=90)

plt.tight_layout()
plt.show()
```




```
seasonal_period = 12
```

```
from statsmodels.tsa.stattools import adfuller # Augmented Dickey-Fuller Test

result = adfuller(df['#Passengers'], autolag='AIC') # Akaike Information Criterion
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

```
ADF Statistic: 0.8153688792060441
p-value: 0.9918802434376409
```

```
# first order differencing
result = adfuller(df['#Passengers'].diff().dropna(), autolag='AIC')
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

```
ADF Statistic: -2.8292668241700034
p-value: 0.05421329028382497
```

```
# second order differencing
result = adfuller(df['#Passengers'].diff().diff().dropna(), autolag='AIC')
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

```
ADF Statistic: -16.38423154246852
p-value: 2.732891850014085e-29
```

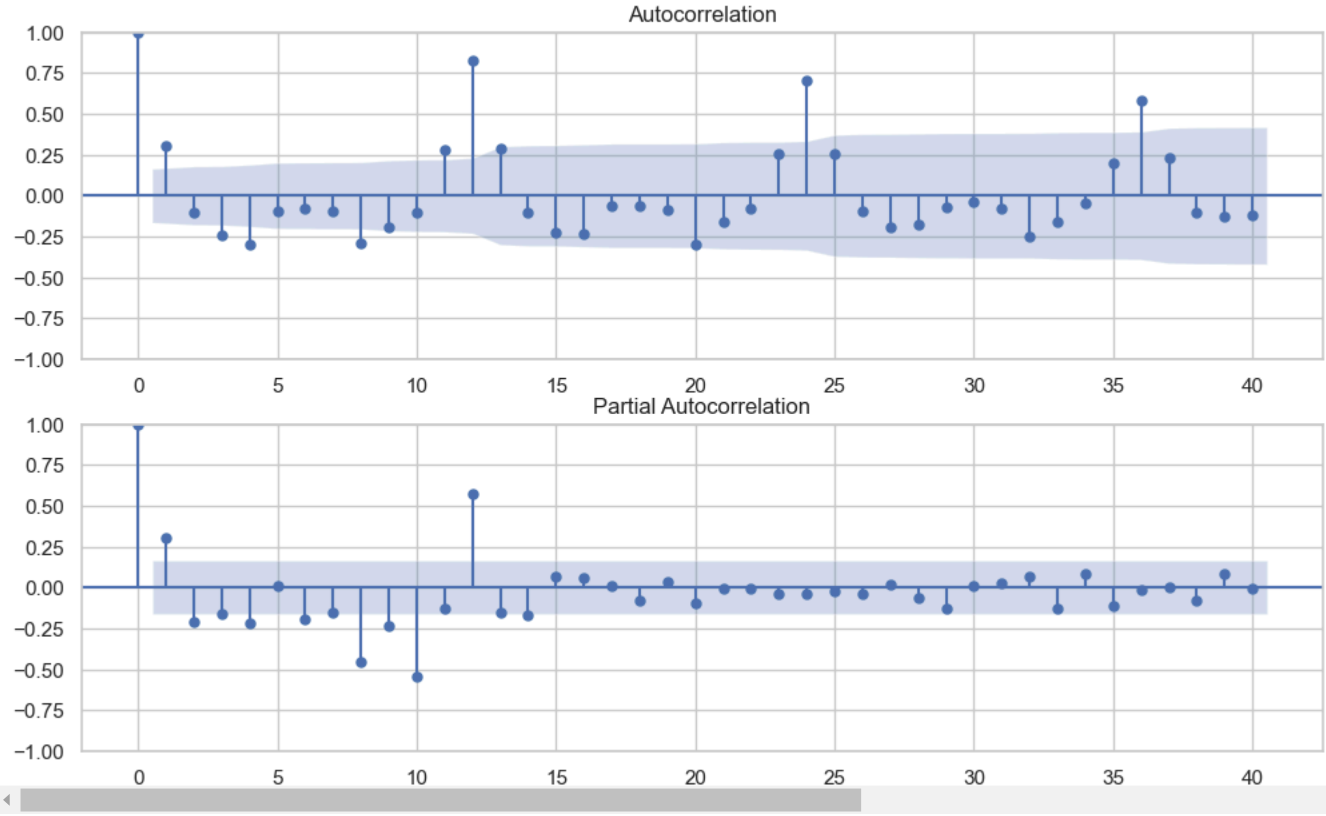
```
# plot the differencing values
fig, (ax1, ax2, ax3) = plt.subplots(3)

ax1.plot(df)
ax1.set_title('Original Time Series')
ax1.axes.xaxis.set_visible(False)

ax2.plot(df.diff())
ax2.set_title('1st Order Differencing')
ax2.axes.xaxis.set_visible(False)

ax3.plot(df.diff().diff())
ax3.set_title('2nd Order Differencing')
ax3.axes.xaxis.set_visible(False)

plt.show()
```


```
# the time series becomes stationary after first order differencing
```

Define Parameters for ARIMA

```
# p = 0 # MA - Moving Average - PACF
# d = 1 # order of differencing - I
# q = 0 # AR - Auto Regressive - ACF
```

```
fig, ax = plt.subplots(2, 1, figsize=(12, 7))
sm.graphics.tsa.plot_acf(df.diff().dropna(), lags=40, ax=ax[0])
sm.graphics.tsa.plot_pacf(df.diff().dropna(), lags=40, ax=ax[1])
plt.show()
```




```
p = 2 # pacf
d = 1 # 1st order difference
q = 1 # acf
```

```
P = 1
D = 0
Q = 3
```

Model Training

```
# define the arima model
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = SARIMAX(df['#Passengers'], order=(p, d, q), seasonal_order=(P, D, Q, seasonal_period))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                #Passengers   No. Observations:          144
Model:             SARIMAX(2, 1, 1)x(1, 0, [1, 2, 3], 12)   Log Likelihood        -563.224
Date:                                    Mon, 30 Sep 2024   AIC                   1142.448
Time:                                            18:26:03   BIC                   1166.151
Sample:                                        01-01-1949   HQIC                  1152.080
                                             - 12-01-1960
Covariance Type:                                      opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.6244      0.101      6.168      0.000       0.426       0.823
ar.L2          0.1947      0.100      1.951      0.051      -0.001       0.390
ma.L1         -0.9675      0.039    -24.632      0.000      -1.045      -0.891
ar.S.L12       0.9619      0.036     26.615      0.000       0.891       1.033
ma.S.L12      -0.1127      0.126     -0.898      0.369      -0.359       0.133
ma.S.L24       0.1355      0.129      1.053      0.292      -0.117       0.388
ma.S.L36       0.0049      0.147      0.033      0.973      -0.284       0.294
sigma2       124.2089     14.750      8.421      0.000      95.299     153.119
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):                16.14
Prob(Q):                              0.93   Prob(JB):                         0.00
Heteroskedasticity (H):               3.99   Skew:                             0.18
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.61
===================================================================================

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

Forecasting

```
# forecast for next 2 years
forecast_steps = 24
forecast = fitted_model.get_forecast(steps=forecast_steps)

# create the date range for the forecasted values
forecast_index = pd.date_range(start=df.index[-1], periods=forecast_steps+1, freq='M')[1:].strftime('%Y-%m') # remove start date
```

```
# create a forecast dataframe
forecast_df = pd.DataFrame({
    "Forecast": list(forecast.predicted_mean),
    "Lower CI": list(forecast.conf_int().iloc[:, 0]),
    "Upper CI": list(forecast.conf_int().iloc[:, 1])
}, index=forecast_index)

forecast_df.head()
```

	Forecast	Lower CI	Upper CI
1961-01	446.711482	424.867844	468.555119
1961-02	423.325499	397.191175	449.459823
1961-03	456.418435	426.807576	486.029294
1961-04	491.562749	459.538919	523.586579
1961-05	505.131996	471.260404	539.003589

```
# plot the forecast values

plt.figure(figsize=(12, 6))
plt.plot(df['#Passengers'], label='Historical Data')
plt.plot(forecast_df['Forecast'], label='Forecast Data')
plt.fill_between(forecast_df.index, forecast_df['Lower CI'], forecast_df['Upper CI'], color='k', alpha=0.1)
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.title('Air Passengers Forecast')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```