Time Series Analysis Project

Contents:

1. Data Description

2. Exploratory Data Analysis
   - Boxplots
   - Seasonal Plots

3. Seasonal Decomposition of Time Series
   - Trends
   - Seasonality
   - Random Component

4. Some Hypothesis Tests
   - Augmented Dickey-Fuller Test
   - Ljung Box Test

5. Model Fittings
   - **ARIMA Model**
   - Fitting
   - Summary Table
   - Forecasting
   - Diagnostic Plots for Residuals
   - Performance Summary and Model Adequacy
   - **SARIMA Model**
   - Fitting
   - Summary Table
   - Forecasting
   - Diagnostic Plots for Residuals
   - Performance Summary and Model Adequacy

6. Results and Conclusion

INTRODUCTION:

- **Briefly introducing the project and its Objectives**:

  The project aims to develop a forecasting model for predicting the total number of monthly passengers for an airline. The dataset used for this project spans from January 1949 to December 1960. The primary objective is to create a reliable predictive model that can capture patterns, trends, and seasonality in the historical passenger data and generate accurate forecasts for future months. By analyzing the historical data and identifying relevant patterns, the model aims to assist the airline in making informed decisions related to capacity planning, resource allocation, and overall operational strategies.

- **Explaining the importance of time series analysis in the context of this project:**

  Time series analysis is essential for forecasting monthly airline passengers due to its ability to identify patterns, trends, and seasonality in historical data. Accurate forecasts help airlines optimize operations, allocate resources efficiently, plan for seasonal variations, and make informed strategic decisions. This analysis aids in mitigating risks, enhancing customer satisfaction, and improving overall operational efficiency.

Data Description:

This is the dataset about the Monthly total number of air passengers for an airline, from January 1949 to December 1961. We aim to find the trend and seasonal components in our dataset and fit a time series model to our dataset to forecast the future revenue of the store.

```
# Printing the first five observations from the dataset

df.head()
```

	Month	Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

Initiating with a concise portrayal of our dataset, we embarked on a comprehensive examination to identify instances of null values within the dataset. proceeded to construct a summary table specifically delineating the attributes of the Revenue column.

```
# Checkiing for the null values

df.isnull().sum()
```

```
Month         0
Passengers    0
dtype: int64
```

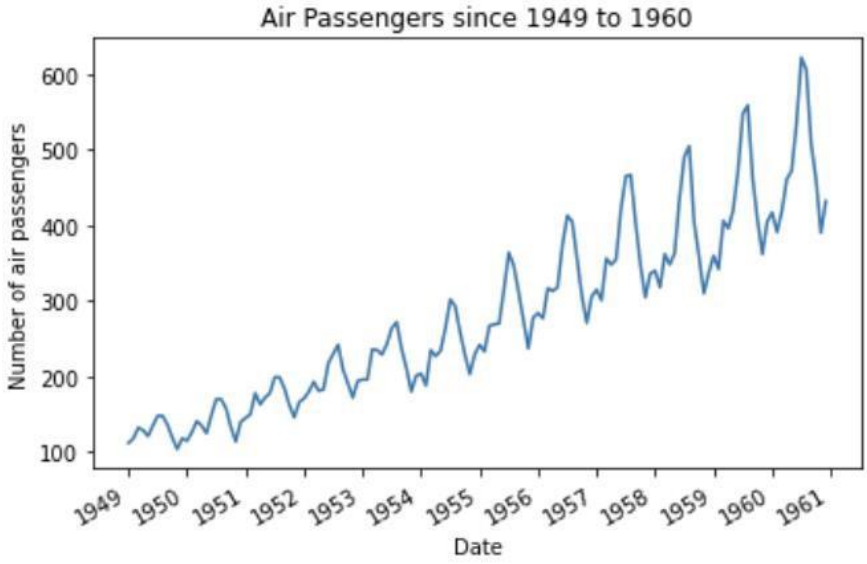
```
# Getting the summary of Air Passenger
round(df['Passengers'].describe(),2)
```

```
count    144.00
mean     280.30
std      119.97
min      104.00
25%      180.00
50%      265.50
75%      360.50
max      622.00
Name: Passengers, dtype: float64
```

From the above summary tables, we conclude we don’t have any missing values in our dataset, and from the summary table, we get 8 number summary values of sales.

Exploratory Data Analysis:

Here is the time series plot of the Air Passenger dataset from 1949 to 1960.



Decomposition Of Time Series:

Time series decomposition is a technique used to break down a time series dataset into its underlying components: trend, seasonality, and residual (or noise). This process allows you to better understand the individual patterns and behaviors that contribute to the overall time series data. Decomposition is particularly useful for gaining insights, making forecasts, and identifying anomalies.

The decomposition of a time series involves three main components:

Trend:

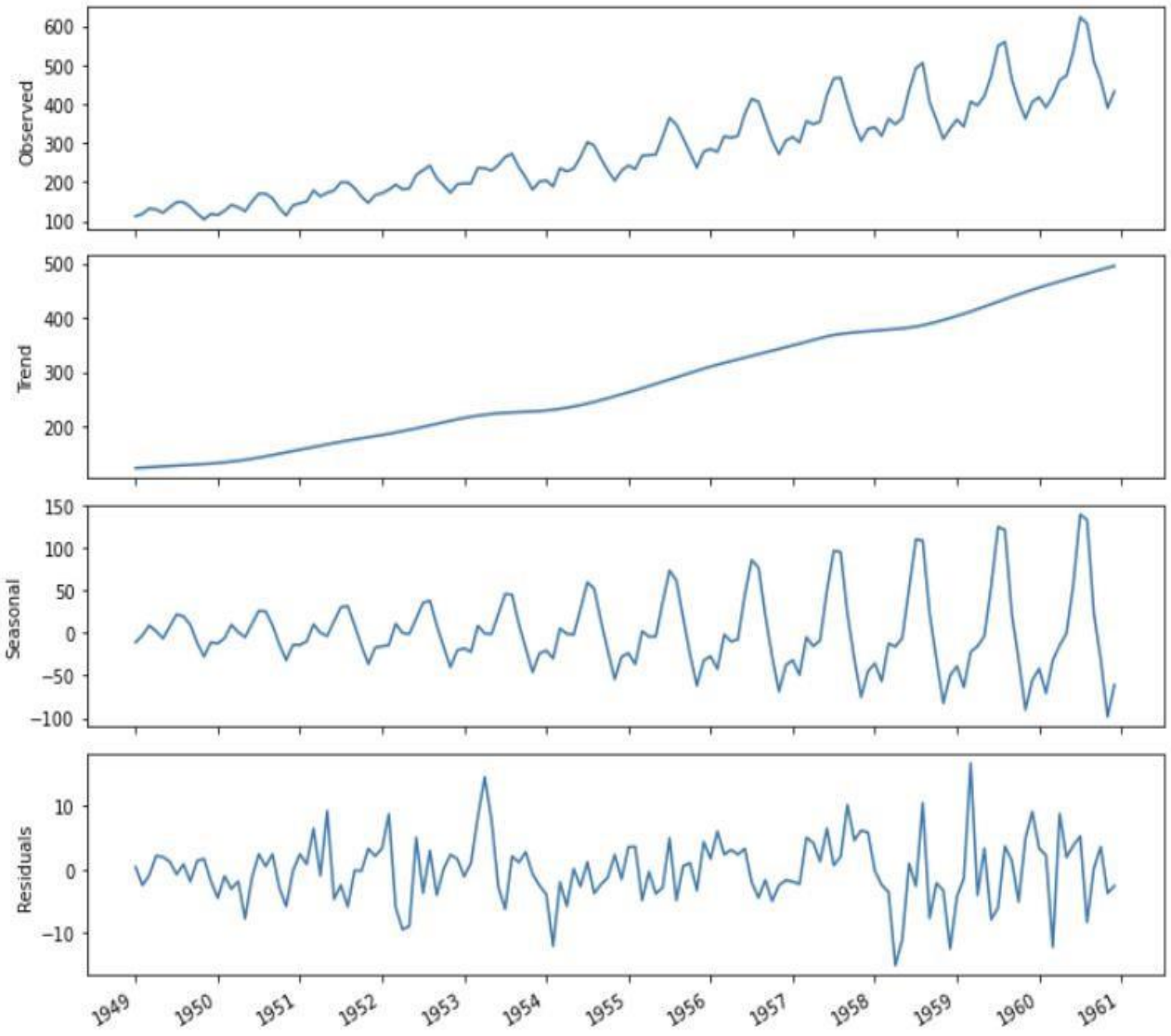
The trend component represents the long-term movement or direction in the data. It captures the overall upward or downward movement of the time series over an extended period. Trends can be linear, exponential, or more complex.

Seasonality:

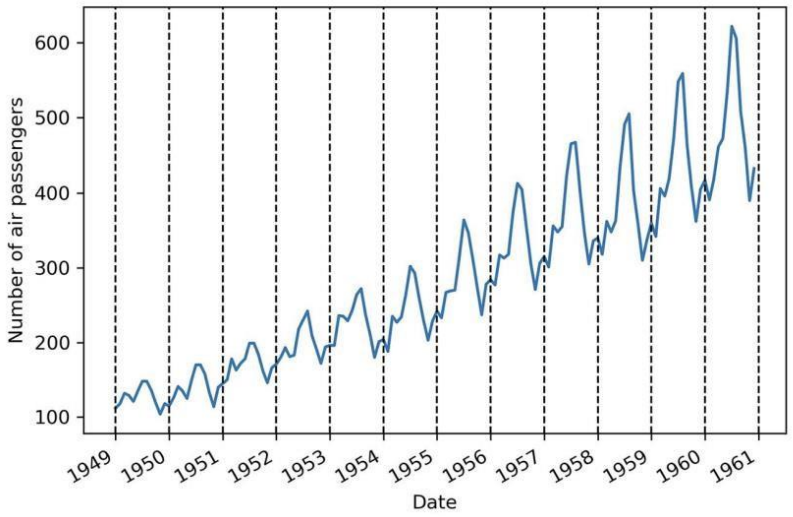
Seasonality refers to the recurring patterns or cycles within the data that repeat at fixed intervals, such as daily, monthly, or yearly. It represents the systematic variations that occur due to factors like holidays, seasons, or economic cycles.

Residual (Noise):

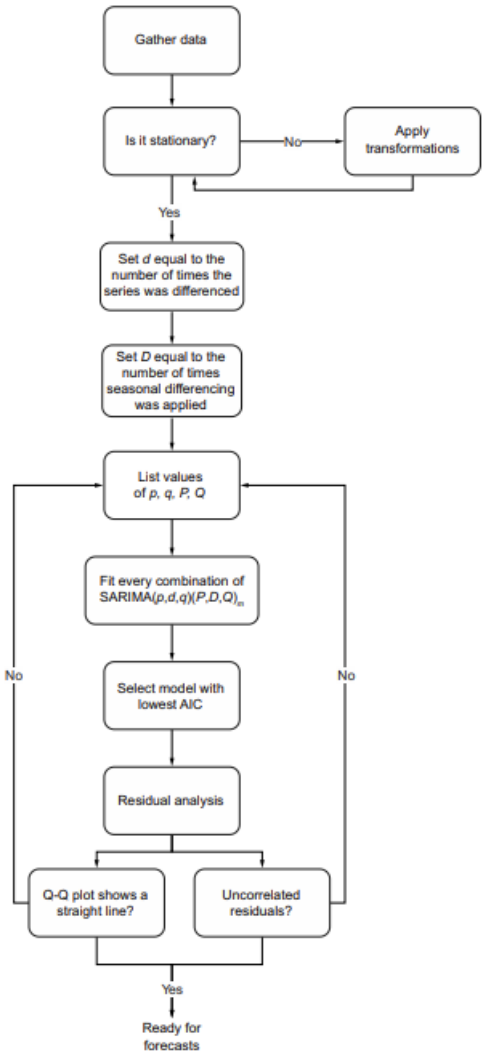
The residual component contains the random fluctuations or noise in the data that cannot be explained by the trend and seasonality. It represents the unexplained variations and any irregular or unpredictable behavior present in the time series.



Identifying seasonal patterns in a time series:



General modelling procedure for the SARIMA mode:



Some Hypothesis Tests:

Augmented Dickey-Fuller Test:

The augmented Dickey-Fuller (ADF) test helps us determine if a time series is stationary.

Hypotheses

The ADF test verifies the following null hypothesis: there is a unit root present in a time series. The alternative hypothesis is that there is no unit root, and therefore the time series is stationary. The result of this test is the ADF statistic, which is a negative number. The more negative it is, the stronger the rejection of the null hypothesis. In its implementation in Python, the p-value is also returned. If its value is less than 0.05, we can also reject the null hypothesis and say the series is stationary.

Ljung Box Test:

The Ljung (pronounced Young) Box test (sometimes called the modified boxPierce, or just the Box test) is a way to test for the absence of autocorrelation, up to a specified lag k.

Ljung Box Test Hypotheses:

The Ljung-Box test uses the following hypotheses:

H_0: The residuals are independently distributed.

H_A: The residuals are not independently distributed; they exhibit serial correlation.

Ideally, we would like to fail to reject the null hypothesis. That is, we would like to see the p-value of the test be greater than 0.05 because this means the residuals for our time series model are independent, which is often an assumption we make when creating a model.

Test Statistic:

The test statistic for the Ljung-Box test is as follows: $Q =$

$n(n+2) \Sigma pk2 / (n-k)$

where:

n = sample size

Σ = a fancy symbol that means “sum” and is taken as the sum of 1 to *h*, where *h* is the number of lags being tested.

$\mathbf{p_k}$ = sample autocorrelation at lag *k.*

Model Fitting:

Once we have a stationary series, we must determine whether there is autocorrelation or not. Remember that a random walk is a series whose first difference is stationary and uncorrelated. The ADF test takes care of the stationarity portion, but we'll need to use the autocorrelation function to determine if the series is correlated or not.

Auto Correlation Function (ACF):

Autocorrelation is the relationship between two values in a time series. To put it another way, the time series data are correlated, hence the word. "Lags" are the term for these kinds of connections. When a characteristic is measured on a regular basis, such as daily, monthly, or yearly, time-series data is created. The number of intervals between two measurements is known as the lag. For example, there is a one-second lag between current and past observations. The lag grows to two if you go back to another interval, and so on. The observations at y_t and y_{t-k} are separated by k time units in mathematical terms. The lag is denoted by K. Depending on the nature of the data, this lag can be measured in days, quarters, or years. When k=1, you're evaluating observations that are next to each other. There is a correlation with each latency. The autocorrelation function (ACF) evaluates the correlation between observations in a time series over a given range of lags. Corr(y_t,y_{t-k}), k=1,2,…. gives the ACF for the time series y.

Partial Autocorrelation Function (PACF):

The partial autocorrelation function, like the ACF, indicates only the association between two data that the shorter lags between those observations do not explain. The partial autocorrelation for lag 3 is, for example, merely the correlation that lags 1 and 2 do not explain. In other words, the partial correlation for each lag is the unique correlation between the two observations after the intermediate correlations have been removed. As previously stated, the autocorrelation function aids in determining the qualities of a time series. The partial autocorrelation function (PACF), on the other hand, is more beneficial during the definition phase for an autoregressive model. Partial autocorrelation plots can be used to specify regression models with time series data as well as Auto-Regressive Integrated Moving Average (ARIMA) models.

ARIMA Model:

An autoregressive integrated moving average – ARIMA model is a generalization of a simple autoregressive moving average – ARMA model. Both of these models are used to forecast or predict future points in the time-series data. ARIMA is a form of regression analysis that indicates the strength of a dependent variable relative to other changing variables.

The components of the ARIMA model:

Autoregressive (AR):

In an autoregression model, we forecast the variable of interest using a linear combination of past values of that variable. The term autoregression indicates that it is a regression of the variable against itself. That is, we use lagged values of the target variable as our input variables to forecast values for the future. An autoregression model of order p will look like this:

$$x_t = \mu + \phi_t x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_p x_{t-p} + a_t$$

where, x_{t-1}, x_{t-2}, …, xt-p are previous values of the process and a random shock at. μ and ϕ's are unknown parameters.

In the above equation, the currently observed value of X is a linear function of its past p values. [0, p] are the regression coefficients that are determined after training. There are some standard methods to determine optimal values of p one of which is, analysing Autocorrelation and Partial Autocorrelation function plots.

PACF and ACF plot helps us to determine what can be the possible value of p and that model is known as AR(p) model. The AR (AutoRegressive) model assumes that the time series data is stationary. Stationary is a fundamental assumption for the application of AR models.

Integrated (I):

The "I" part of ARIMA is about the process of differencing the time series data to make it stationary. Stationary means that the statistical properties of the data (like mean and variance) don't change over time. The order of differencing (d) determines how many times the differencing operation is applied to make the data stationary. In some cases, one level of differencing is sufficient, while in other cases, higher levels of differencing may be needed to achieve stationarity. **Moving Average (MA):**

In the MA model, the future values of the time series are modeled as linear combinations of the current and past white noise error terms. The "q" in the ARIMA (p, d, q) represents the order of the moving average, which specifies how many past error terms are considered in the model. The general form of the MA (q) model can be expressed as follows:

$$x_t = \mu + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \ldots - \theta_q a_{t-q}$$

The MA model is called a "Moving Average" model because it uses the moving average of past error terms to predict future values of the time series. The model captures the immediate impact of the recent error terms on the current values.

The ARIMA forecasting equation for a stationary time series is a linear (i.e., regression-type) equation in which the predictors consist of lags of the dependent variable and/or lags of the forecast errors.

A nonseasonal ARIMA model is classified as an **"ARIMA (p, d, q)"** model, where: **p:** number of autoregressive terms**, d:** number of nonseasonal differences needed for stationarity, and **q:** number of lagged forecast errors in the prediction equation**.** *In terms of y, the general forecasting equation is:* $\hat{y}_t = \mu + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}$

Forecasting with an ARIMA(p,d,q) model:

We'll first model the dataset using an ARIMA(p,d,q) model. That way, we can compare its performance to the SARIMA(p,d,q)(P,D,Q)m model. Following the general modelling procedure we outlined before, we'll first test for stationarity. Again, we use the ADF test.

```
In [10]: ad_fuller_result = adfuller(df['Passengers'])

         print(f'ADF Statistic: {ad_fuller_result[0]}')
         print(f'p-value: {ad_fuller_result[1]}')

         ADF Statistic: 0.8153688792060488
         p-value: 0.991880243437641
```

This prints out an ADF statistic of 0.82 and a p-value of 0.99. Therefore, we cannot reject the null hypothesis and the series is not stationary.

We'll difference the series and test for stationarity again.

```
In [11]: df_diff = np.diff(df['Passengers'], n=1)

         ad_fuller_result = adfuller(df_diff)

         print(f'ADF Statistic: {ad_fuller_result[0]}')
         print(f'p-value: {ad_fuller_result[1]}')
```

```
ADF Statistic: -2.8292668241700056
p-value: 0.054213290283824704
```

This returns an ADF statistic of –2.83 and a p-value of 0.054. Again, we cannot reject the null hypothesis, and differencing the series once did not make it stationary.

Therefore, we'll difference it again and test for stationarity.

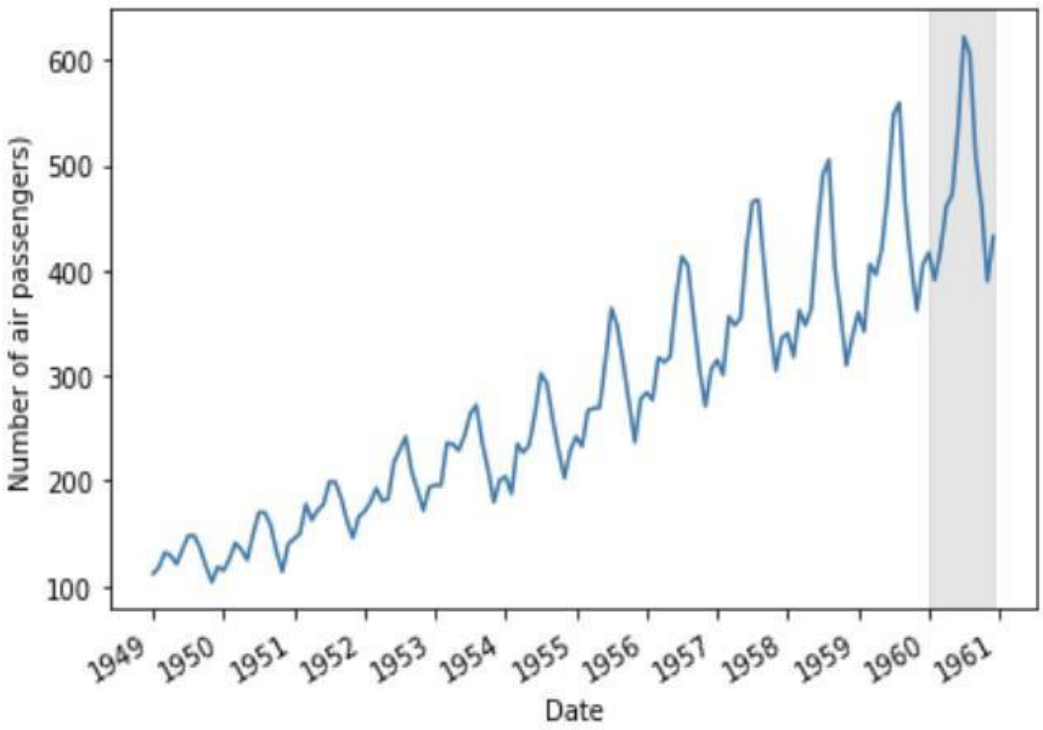
```
In [13]: df_diff2 = np.diff(df_diff, n=1)

         ad_fuller_result = adfuller(df_diff2)

         print(f'ADF Statistic: {ad_fuller_result[0]}')
         print(f'p-value: {ad_fuller_result[1]}')
```

```
ADF Statistic: -16.384231542468548
p-value: 2.732891850013967e-29
```

This returns an ADF statistic of –16.38 and a p-value of 2.73 × 10–29. Now we can reject the null hypothesis, and our series is considered to be stationary. Since the series was differenced twice to become stationary, d = 2.

Now we can define a range of possible values for the parameters p and q and fit all unique ARIMA(p,d,q) models. We'll specifically choose a range from 0 to 12 to allow the ARIMA model to go back 12 timesteps in time. Since the data is sampled monthly and we know it is seasonal, we can hypothesize that the number of air passengers in January of a given year is likely predictive of the number of air passengers in January of the following year. Since these two points are 12 timesteps apart, we'll allow the values of p and q to vary from 0 to 12 in order to potentially capture this seasonal information in the
ARIMA(p,d,q) model. Finally, since we are working with an ARIMA model, we'll set P, D, and Q to 0.

Splitting the Train set and test set split for the air passenger dataset. The shaded area represents the testing period, which corresponds to the full year of 1960, as our goal is to forecast a year of monthly air passengers.



We set the parameters P, D, Q, and m, even though we are working with an ARIMA model. This is because we are going to define an optimize. SARIMA function that will then be reused. We set P, D, and Q to 0 because a SARIMA(p,d,q)(0,0,0)m model is equivalent to an ARIMA(p,d,q) model.

In [18]:
```
def optimize_ARIMA(endog: Union[pd.Series, list], order_list: list, d: int) -> pd.DataFrame:

    results = []

    for order in tqdm_notebook(order_list):
        try:
            model = SARIMAX(endog, order=(order[0], d, order[1]), simple_differencing=False).fit(disp=False
        except:
            continue

        aic = model.aic
        results.append([order, aic])

    result_df = pd.DataFrame(results)
    result_df.columns = ['(p,q)', 'AIC']

    #Sort in ascending order, lower AIC is better
    result_df = result_df.sort_values(by='AIC', ascending=True).reset_index(drop=True)

    return result_df
```

Defining a function to select the best ARIMA model:

With the function ready, we can launch it using the train set and get the ARIMA model with the lowest AIC. Despite the fact that we are using the optimize_SARIMA function, we are still fitting an ARIMA model because we specifically set P, D, and Q to 0. For the train set, we'll take all data points but the last twelve, as they will be used

In [20]:
```
ps = range(0, 13, 1)
qs = range(0, 13, 1)
Ps = [0]
Qs = [0]

d = 2
D = 0
s = 12

ARIMA_order_list = list(product(ps, qs, Ps, Qs))

train = df['Passengers'][:-12]

ARIMA_result_df = optimize_SARIMA(train, ARIMA_order_list, d, D, s)
ARIMA_result_df
```

```
  0%|          | 0/169 [00:00<?, ?it/s]
```

for the test set.

Out[20]:

	(p,q,P,Q)	AIC
0	(11, 3, 0, 0)	1016.840035
1	(11, 4, 0, 0)	1019.035337
2	(11, 5, 0, 0)	1020.378996
3	(12, 0, 0, 0)	1020.782069
4	(11, 1, 0, 0)	1021.023930
...	...	...
164	(5, 0, 0, 0)	1281.732157
165	(3, 0, 0, 0)	1300.282335
166	(2, 0, 0, 0)	1302.913196
167	(1, 0, 0, 0)	1308.152194
168	(0, 0, 0, 0)	1311.919269

169 rows × 2 columns

This returns a DataFrame where the model with the lowest AIC is a SARIMA(11,2,3)(0,0,0)12 model, which is equivalent to an ARIMA(11,2,3) model. As you can see, allowing the order p to vary from 0 to 12 was beneficial for the model, as the model with the lowest AIC takes into account the past 11 values of the series, since p = 11. We will see if this is enough to capture seasonal information from the series, and we will compare the performance of the ARIMA model to the SARIMA model.

Below is the summary table of **ARIMA (11,2,3):**

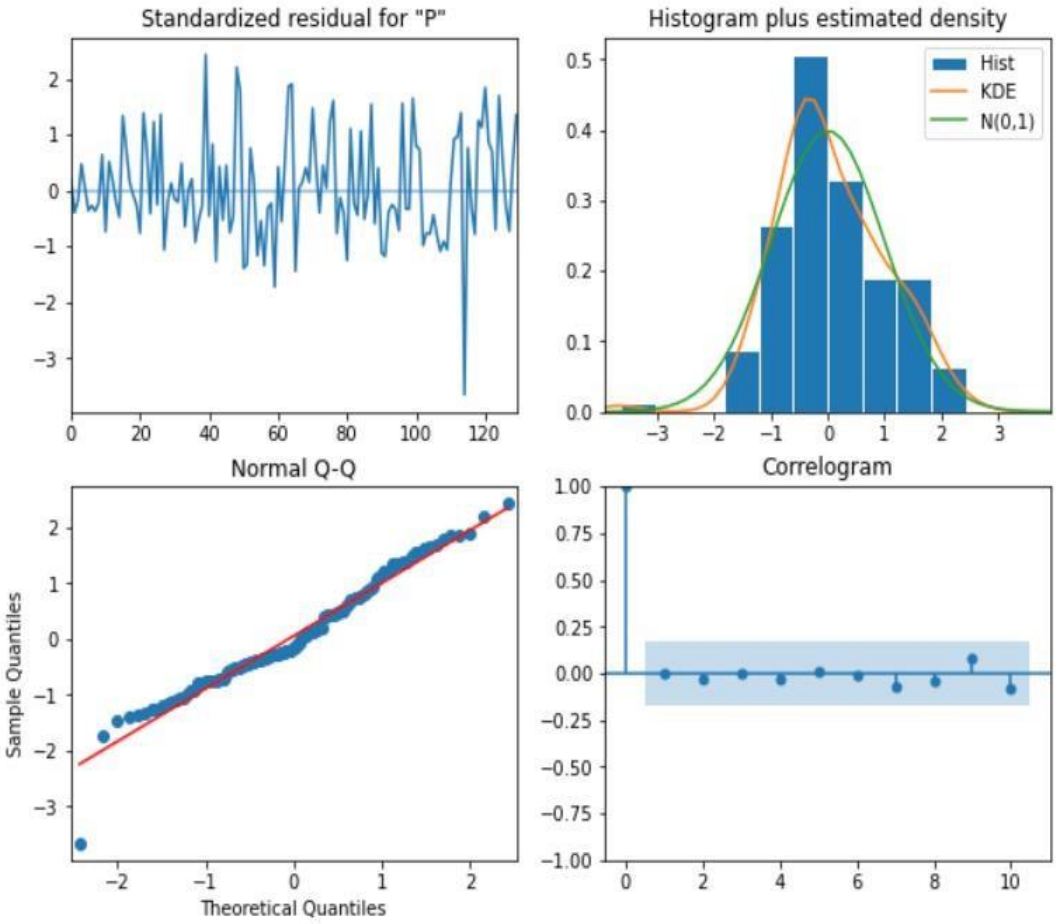
SARIMAX Results

Dep. Variable:	Passengers	No. Observations:	132
Model:	SARIMAX(11, 2, 3)	Log Likelihood	-493.420
Date:	Wed, 02 Aug 2023	AIC	1016.840
Time:	15:29:26	BIC	1059.853
Sample:	0	HQIC	1034.318
	- 132		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.8240	0.100	-8.220	0.000	-1.021	-0.628
ar.L2	-0.9630	0.049	-19.747	0.000	-1.059	-0.867
ar.L3	-0.8520	0.087	-9.752	0.000	-1.023	-0.681
ar.L4	-0.9538	0.047	-20.361	0.000	-1.046	-0.862
ar.L5	-0.8332	0.092	-9.103	0.000	-1.013	-0.654
ar.L6	-0.9501	0.043	-22.203	0.000	-1.034	-0.866
ar.L7	-0.8360	0.089	-9.405	0.000	-1.010	-0.662
ar.L8	-0.9624	0.049	-19.602	0.000	-1.059	-0.866
ar.L9	-0.8250	0.086	-9.601	0.000	-0.993	-0.657
ar.L10	-0.9582	0.031	-30.928	0.000	-1.019	-0.897
ar.L11	-0.8087	0.095	-8.469	0.000	-0.996	-0.622
ma.L1	-0.3338	0.136	-2.455	0.014	-0.600	-0.067
ma.L2	0.2204	0.160	1.373	0.170	-0.094	0.535
ma.L3	-0.2912	0.141	-2.063	0.039	-0.568	-0.015
sigma2	104.4098	17.694	5.901	0.000	69.731	139.089

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	3.83
Prob(Q):	0.96	Prob(JB):	0.15
Heteroskedasticity (H):	2.23	Skew:	-0.02
Prob(H) (two-sided):	0.01	Kurtosis:	3.84

From the above summary table of ARIMA (11,2,3), it is visible that all the parameters of the model are non-zero.



Residuals' diagnostics of the ARIMA(11,2,3) model. In the top-left plot, the residuals have no trend with a variance that seems fairly constant over time, which resembles the behaviour of white noise. The top-right plot shows the distribution of the residuals, which approaches a normal distribution, despite the unusual peak. This is further confirmed by the Q-Q plot at the bottom left, which displays a fairly straight line that lies on y = x. Finally, the correlogram in the bottom-right plot shows no significant autocorrelation coefficients after lag 0, which is exactly like white noise. From this analysis, the residuals resemble white noise.

The next step is to run the Ljung-Box test on the residuals to make sure that they are independent and uncorrelated.

In [39]:
```
residuals = ARIMA_model_fit.resid

pvalue = acorr_ljungbox(residuals, np.arange(1, 11, 1))

print(pvalue)
```

```
      lb_stat  lb_pvalue
1    6.555609   0.010455
2    6.682527   0.035392
3    6.962313   0.073109
4    7.753990   0.101017
5    8.397350   0.135654
6    8.681574   0.192294
7    9.493147   0.219161
8    9.731599   0.284372
9    9.742065   0.371767
10  11.536975   0.317239
```

The returned p-values are all greater than 0.05 except for the first two values. This means that, according to the Ljung-Box test, we reject the null hypothesis with a 5% chance of being wrong, since we set our significance boundary to 0.05. However, the third value and onwards are all greater than 0.05, so we reject the null hypothesis, concluding that the residuals are uncorrelated starting at lag 3.

This is an interesting situation to dissect, because the graphical analysis of the residuals leads us to conclude that they resemble white noise, but the LjungBox test points to some correlation at lags 1 and 2. This means that our ARIMA model is not capturing all the information from the data.

we wish to predict a full year of monthly air passengers, using the last 12 months of data as our test set. The baseline model is the naive seasonal forecast, where we simply use the number of air passengers for each month of 1959 as a forecast for each month of 1960.

We can append the forecasts from our ARIMA(11,2,3) model to the test DataFrame.

Out[25]:

	Month	Passengers	naive_seasonal	ARIMA_pred
132	1960-01	417	360	422.353117
133	1960-02	391	342	410.643933
134	1960-03	419	406	461.834233
135	1960-04	461	396	457.812177
136	1960-05	472	420	481.684310
137	1960-06	535	472	531.085259
138	1960-07	622	548	606.149563
139	1960-08	606	559	615.467593
140	1960-09	508	463	525.627301
141	1960-10	461	407	467.133879
142	1960-11	390	362	425.074090
143	1960-12	432	405	467.353795

Defining a function to select the best SARIMA model:

In [19]:
```
def optimize_SARIMA(endog: Union[pd.Series, list], order_list: list, d: int, D: int, s: int) -> pd.DataFrame:

    results = []

    for order in tqdm_notebook(order_list):
        try:
            model = SARIMAX(
                endog,
                order=(order[0], d, order[1]),
                seasonal_order=(order[2], D, order[3], s),
                simple_differencing=False).fit(disp=False)
        except:
            continue

        aic = model.aic
        results.append([order, aic])

    result_df = pd.DataFrame(results)
    result_df.columns = ['(p,q,P,Q)', 'AIC']

    #Sort in ascending order, lower AIC is better
    result_df = result_df.sort_values(by='AIC', ascending=True).reset_index(drop=True)

    return result_df
```

With the function ready, we can launch it using the train set and get the SARIMA model with the lowest AIC. Despite the fact that we are using the optimize_SARIMA function, we are still fitting an ARIMA model because we specifically set P, D, and Q to

0. For the train set, we'll take all data points but the last twelve, as they will be used for the test set.

In [21]:
```
ps = range(0, 4, 1)
qs = range(0, 4, 1)
Ps = range(0, 4, 1)
Qs = range(0, 4, 1)

SARIMA_order_list = list(product(ps, qs, Ps, Qs))

train = df['Passengers'][:-12]

d = 1
D = 1
s = 12

SARIMA_result_df = optimize_SARIMA(train, SARIMA_order_list, d, D, s)
SARIMA_result_df
```

```
  0%|          | 0/256 [00:00<?, ?it/s]
```

Out[21]:

	(p,q,P,Q)	AIC
0	(2, 1, 1, 2)	892.249548
1	(2, 1, 2, 1)	893.636711
2	(2, 1, 1, 3)	894.100533
3	(1, 0, 1, 2)	894.289353
4	(0, 1, 1, 2)	894.992176
...	...	...
251	(0, 0, 2, 0)	906.940147
252	(3, 2, 0, 3)	907.181875
253	(0, 0, 3, 2)	907.472510
254	(0, 0, 3, 0)	908.742583
255	(0, 0, 0, 3)	908.781405

Once the function is done running, we find that the SARIMA(2,1,1)(1,1,2)12 model has the lowest AIC, which is a value of 892.24. We can fit this model again on the training set to perform residual analysis.

Forecasting with a SARIMA(p,d,q)(P,D,Q)m model:

we used an ARIMA(11,2,3) model to forecast the number of monthly air passengers. Now we'll fit a SARIMA model and see if it performs better than the ARIMA model.

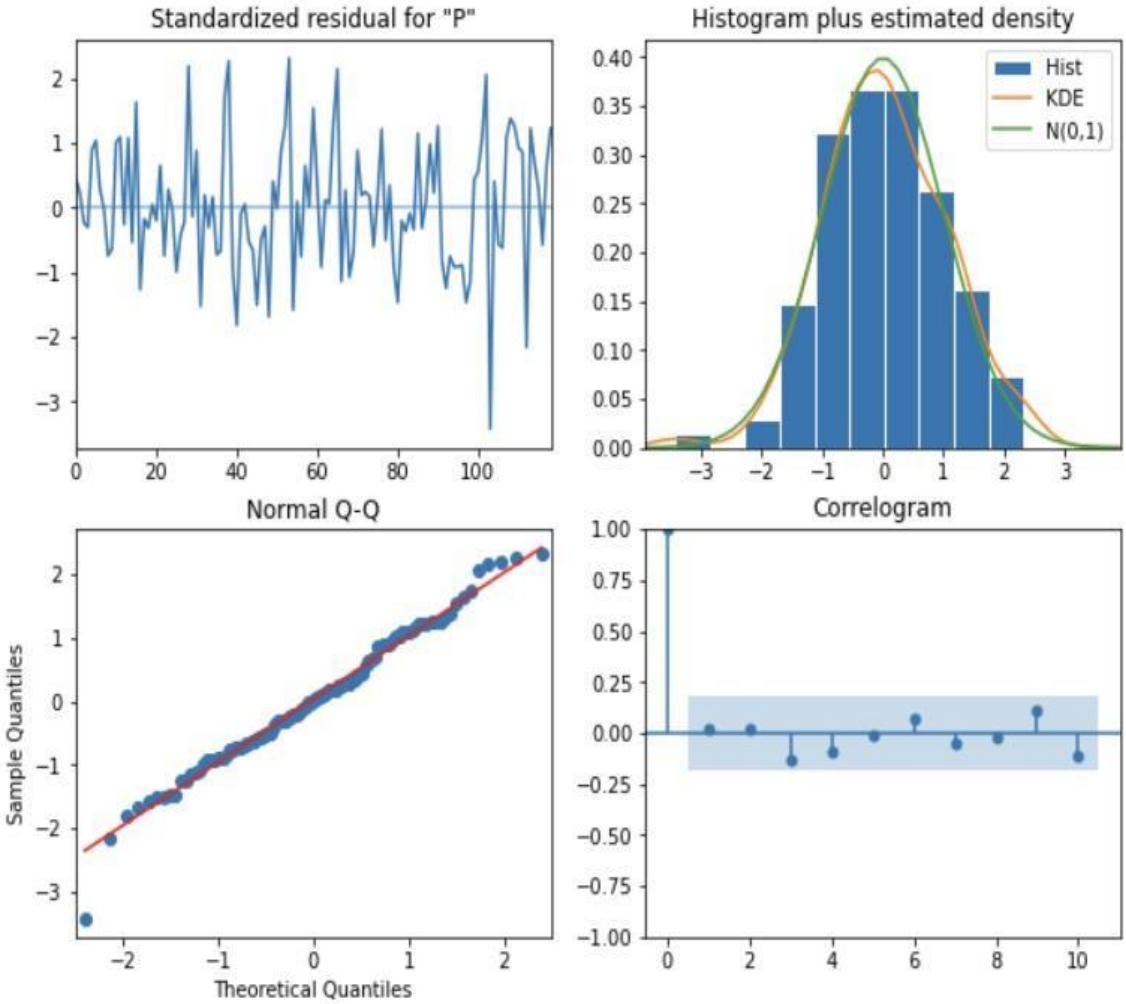
SARIMAX Results

Dep. Variable:	Passengers	No. Observations:	132
Model:	SARIMAX(2, 1, 1)x(1, 1, [1, 2], 12)	Log Likelihood	-439.125
Date:	Fri, 11 Aug 2023	AIC	892.250
Time:	22:25:17	BIC	911.703
Sample:	0	HQIC	900.149
	- 132		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-1.2660	0.085	-14.904	0.000	-1.433	-1.100
ar.L2	-0.3396	0.078	-4.381	0.000	-0.492	-0.188
ma.L1	0.9994	0.272	3.673	0.000	0.466	1.533
ar.S.L12	0.9985	0.110	9.064	0.000	0.783	1.214
ma.S.L12	-1.3293	1.360	-0.978	0.328	-3.994	1.335
ma.S.L24	0.3569	0.409	0.874	0.382	-0.444	1.158
sigma2	79.2979	97.163	0.816	0.414	-111.138	269.734

Ljung-Box (L1) (Q):	0.06	Jarque-Bera (JB):	0.91
Prob(Q):	0.81	Prob(JB):	0.63
Heteroskedasticity (H):	1.57	Skew:	-0.09
Prob(H) (two-sided):	0.16	Kurtosis:	3.39

From the above summary table of SARIMA(2,1,1)(1,1,2)12), it is visible that all the parameters of the model are non-zero.



Residuals' diagnostics of the SARIMA(2,1,1)(1,1,2)12 model. The top-left plot shows that the residuals do not exhibit a trend or a change in variance. The top-right plot shows that the residuals' distribution is very close to a normal distribution. This is further supported by the Q-Q plot at the bottom left, which displays a fairly straight line that lies on y = x. Finally, the correlogram at the bottom right shows no significant coefficients after lag 0. Therefore, everything leads to the conclusion that the residuals resemble white noise.

The results show that our residuals are completely random, which is exactly what we are looking for in a good model. The final test to determine whether we can use this model for forecasting or not is the Ljung-Box test.

In [37]:
```
residuals = SARIMA_model_fit.resid

pvalue = acorr_ljungbox(residuals, np.arange(1, 11, 1))

print(pvalue)
```

```
     lb_stat  lb_pvalue
1   0.004294   0.947756
2   0.744149   0.689303
3   1.018199   0.796848
4   1.224264   0.874088
5   1.435030   0.920443
6   1.711729   0.944212
7   2.306488   0.940951
8   2.719260   0.950722
9   2.735459   0.973867
10  4.976940   0.892713
```

The returned p-values are all greater than 0.05. Therefore, we do not reject the null hypothesis, and we conclude that the residuals are independent and uncorrelated, just like white noise. Our model has passed all the tests from the residuals analysis, and we are ready to use it for forecasting. Again, we'll forecast the number of monthly air passengers for the year of 1960 to compare the predicted values to the observed values in the test set.

In [29]:
```
SARIMA_pred = SARIMA_model_fit.get_prediction(132, 143).predicted_mean

test['SARIMA_pred'] = SARIMA_pred
test
```

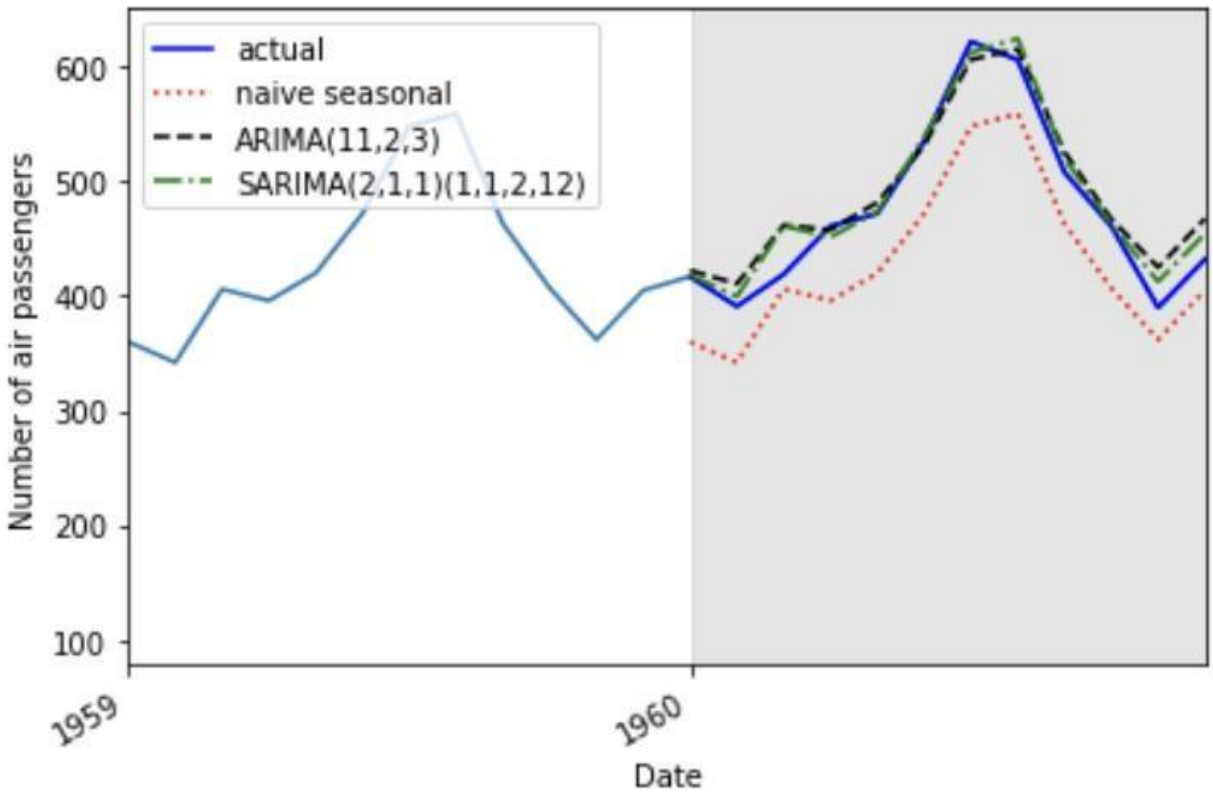
Out[29]:

	Month	Passengers	naive_seasonal	ARIMA_pred	SARIMA_pred
132	1960-01	417	360	422.353117	418.518709
133	1960-02	391	342	410.643933	399.537816
134	1960-03	419	406	461.834233	461.277148
135	1960-04	461	396	457.812177	451.393708
136	1960-05	472	420	481.684310	473.687649
137	1960-06	535	472	531.085259	538.791211
138	1960-07	622	548	606.149563	612.439831
139	1960-08	606	559	615.467593	624.596710
140	1960-09	508	463	525.627301	520.137160
141	1960-10	461	407	467.133879	462.795787
142	1960-11	390	362	425.074090	412.655111
143	1960-12	432	405	467.353795	454.149914

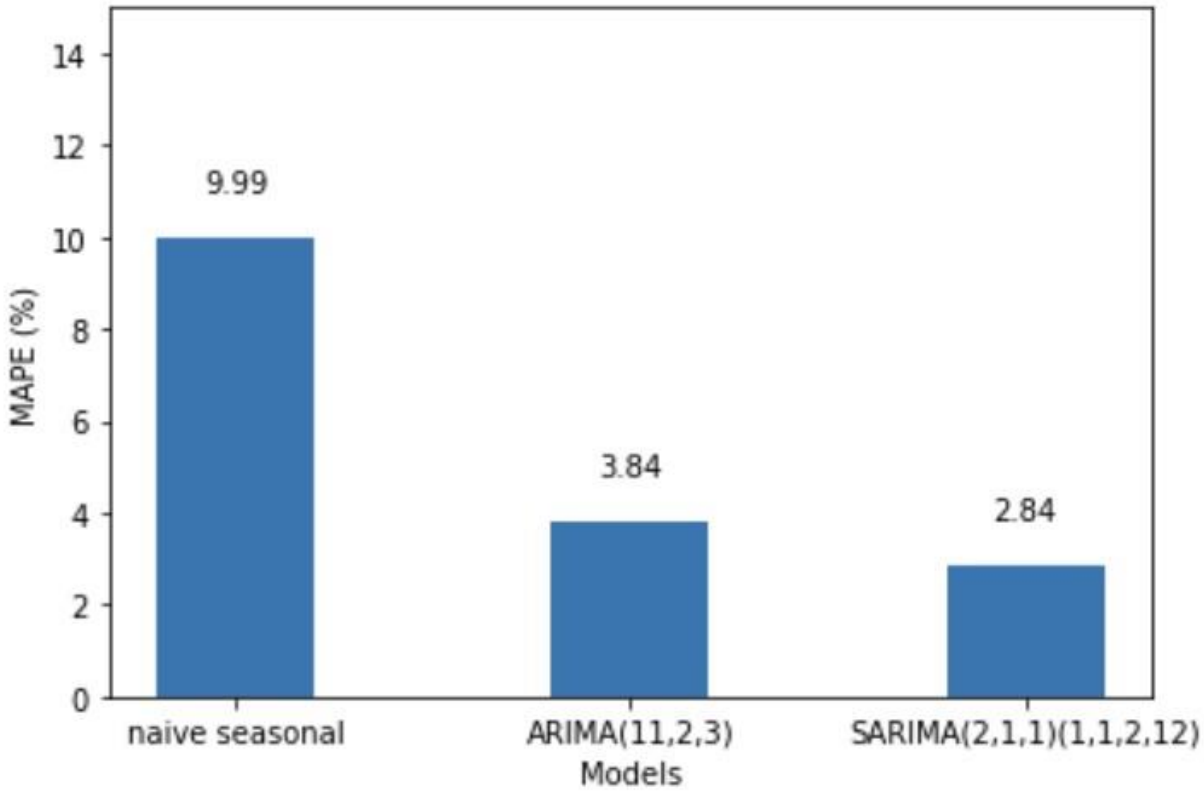
Comparing the performance of each forecasting method:

We can now compare the performance of each forecasting method: the naive seasonal forecasts, the ARIMA model, and the SARIMA model. We'll use the mean absolute percentage error (MAPE) to evaluate each model. We can first visualize the forecasts against the observed values of the test set

Forecasts of the number of monthly air passengers. The shaded area designates the test set. You can see that the curves coming from the ARIMA and SARIMA models almost obscure the observed data, which is indicative of good predictions.



We can measure the MAPE of each model and display it in a bar plot.



The MAPE of all forecasting methods. You can see that the best-performing model is the SARIMA model, since it has the lowest MAPE of all methods.

we can see that our baseline achieves a MAPE of 9.99%. The ARIMA model produced forecasts with a MAPE of 3.85%, and the SARIMA model scored a MAPE of 2.85%. A MAPE closer to 0 is indicative of better predictions, so the SARIMA model is the bestperforming method for this situation. This makes sense, since our dataset had clear seasonality, and the SARIMA model is built to use the seasonal properties of time series to make forecasts.

Results:

Model	*MAPE*
Naïve Seasonal	*9.99%*
ARIMA	*3.85%*
SARIMA	*2.85%*

.