TABLE OF CONTENTS

1: Arima Time series model (Seasonal)

Forecasting People's Search Interest Over Time in Walmart

Introduction and Motivation

Walmart is one of the biggest retail corporations. One of the key factors to a successful giant retail business is to be a household name. In marketing, we consider factors like sales, profit, ad campaigns and so on. It is always interesting to see how people do analysis, forecast, and produce innovative marketing strategies. Here, monthly data of people's interest over time from 2004-2022(till April) has been collected from link, analyzed the data to understand where Walmart stands in terms of popularity, and it will be useful in strategy planning/decision making from marketing analytics.

Data Description

Data Link: [Click here]

Date-Data range from (Jan 2004 – Apr 2022)

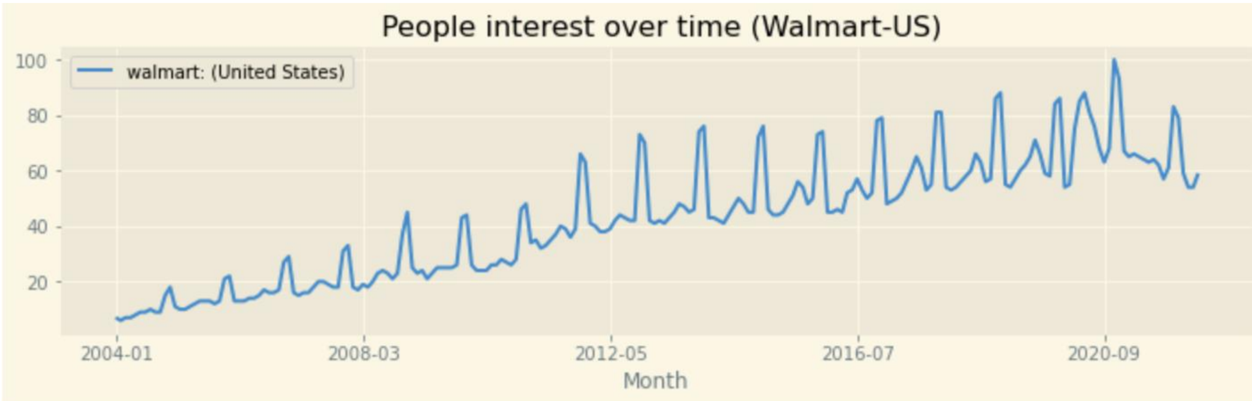
Search Interest over Time (Max value 100)- Numbers represent search interest relative to the highest point on the chart for the given region and time.

	Month	walmart: (United States)
0	2004-01	7
1	2004-02	6
2	2004-03	7
3	2004-04	7
4	2004-05	8
...	...	...
215	2021-12	79
216	2022-01	59
217	2022-02	54
218	2022-03	54
219	2022-04	59

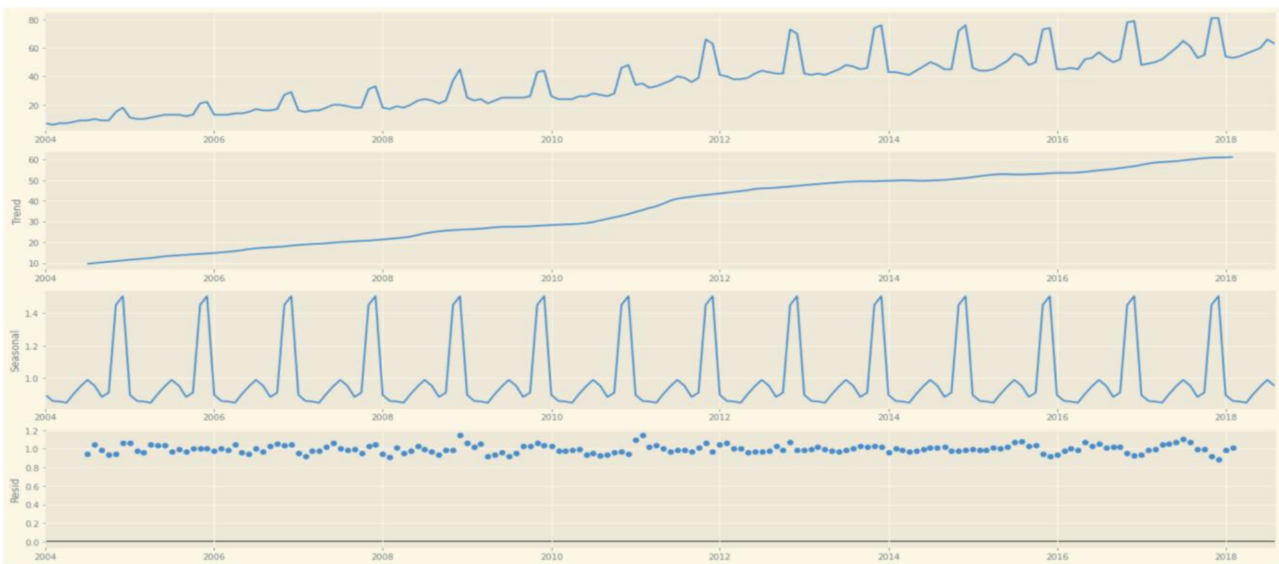
220 rows × 2 columns

Time Series Analysis

Before the analysis, let us split the data into train and test data to ensure that the test data is not biased. Here, data is split into train (80%) and test (20%). Now we will start analyzing the train data by plotting the continuous variable against date variable to understand people's interest over time in United states from Jan 2004 to Apr 2022. From the plot below we can see the data is seasonal and it suggests stationery. And there is a clear upward trend. And we notice that the people's interest over time is increasing in numbers. We can see the numbers are dropping by the end of every year.

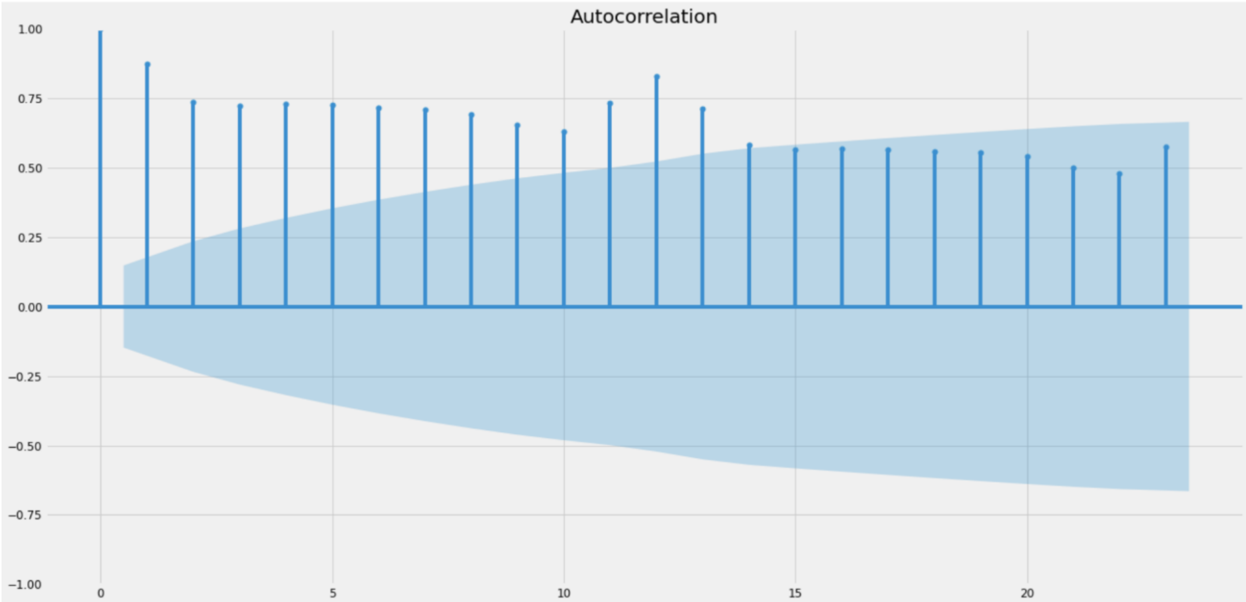


We use seasonal decompose plot below to understand trend, seasonality, and residual, we notice a clear upward trend, seasonality of the data looks good, and the residual shows high variability throughout the years. This insight is useful to make non- stationary data stationary.



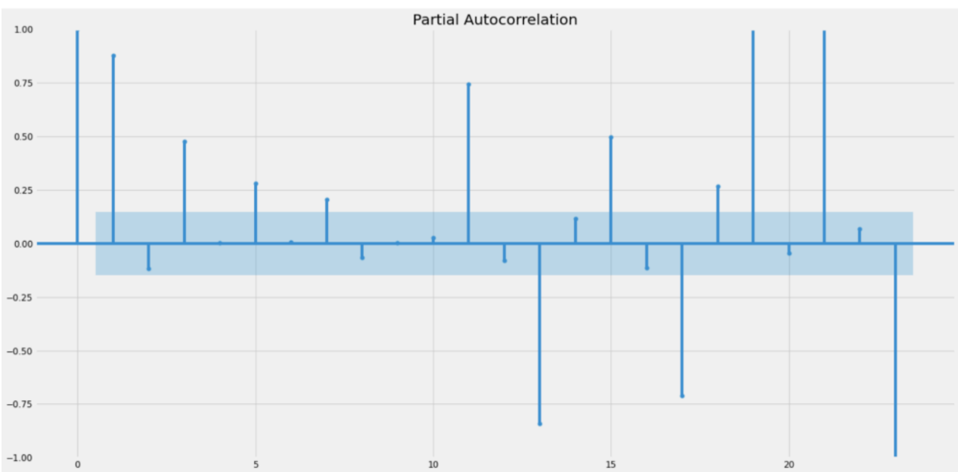
Box-Jenkins Method

Now, we know the data is stationary, we will move on to the next step which is identifying the ARMA model. We will plot ACF and PACF to identify possible ARMA models.



ACF plot suggests high auto correlation and it is challenging to determine with above plot. Intuitively I consider MA (0), MA (1), MA (2).

From the below PACF plot, I would suggest AR (1) though there are spikes as the order of the lags increasing.



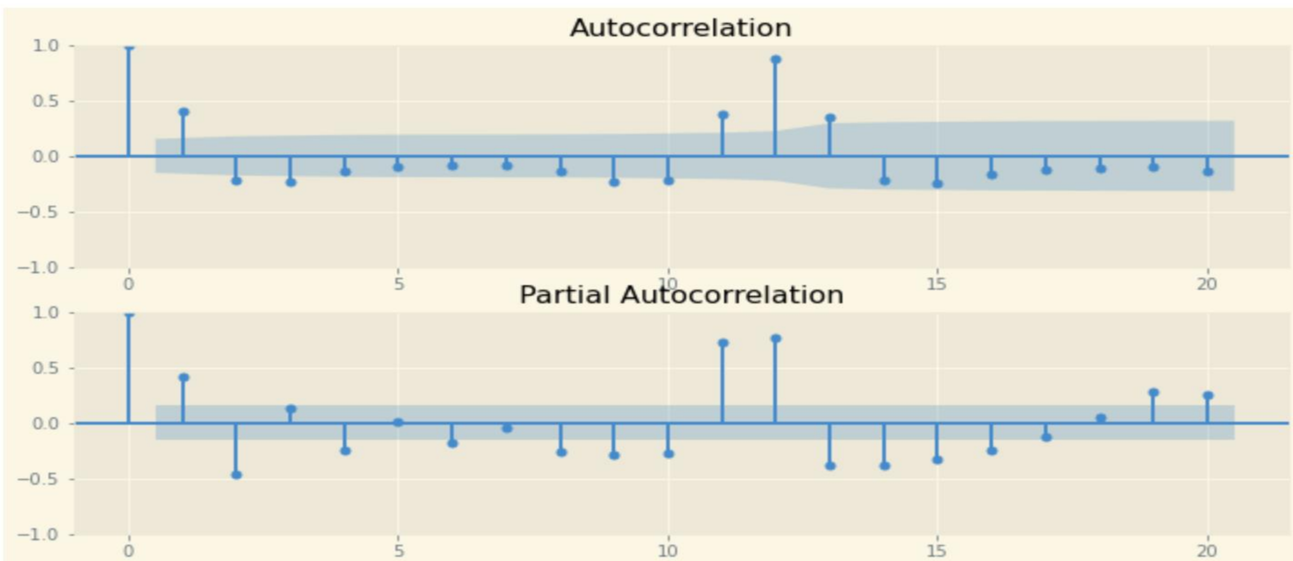
We are unsure weather the data is stationary from the above plot but let us validate the stationarity using the Augmented Dickey Fuller (ADF) Test. As the original data is not stationary, we have detrended the data and checked for stationarity but still the data is not stationary. So, we have taken difference of detrended data. Now the data looks stationary. We validate the stationarity of data when p-value is less than the significant level.

```
Is the data stationary ?
Test statistic = -0.727
P-value = 0.840
Critical values :
        1%: -3.471374345647024 - The data is not stationary with 99% confidence
        5%: -2.8795521079291966 - The data is not stationary with 95% confidence
        10%: -2.5763733302850174 - The data is not stationary with 90% confidence

 Is the de-trended data stationary ?
Test statistic = -2.808
P-value = 0.057
Critical values :
        1%: -3.4744158894942156 - The data is not stationary with 99% confidence
        5%: -2.880878382771059 - The data is not stationary with 95% confidence
        10%: -2.577081275821236 - The data is  stationary with 90% confidence

 Is the 12-lag differenced de-trended data stationary ?
Test statistic = -3.575
P-value = 0.006
Critical values :
        1%: -3.4779446621720114 - The data is  stationary with 99% confidence
        5%: -2.8824156122448983 - The data is  stationary with 95% confidence
        10%: -2.577901887755102 - The data is  stationary with 90% confidence
```

In order to determine ARIMA model we need to check our ACF and PACF plot. To check the SARMA model we need to consider seasonal trends along with the ARIMA model.



Given the ACF and PACF plot, here I suggested possible SARIMA models we could consider to train the model.

```
Possible ARIMA models suggested:
SARIMAX: [0, 0, 1] x [0, 1, 1, 12]
SARIMAX: [1, 0, 1] x [0, 1, 1, 12]
SARIMAX: [1, 0, 0] x [0, 1, 1, 12]
```

We feed our train data with p and q range from (0,2) to function from stats model package called SARIMAX () to check their AIC (Akaike information criterion) score. AIC measures "goodness of fit" of model. The lower the AIC, the better the model fits. We are going to choose the model with minimum AIC. Below images display the number of models generated with AIC and the best model.

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1807.307919165748
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1613.9808357272648
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:986.515988038335
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:954.5165571259184
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1113.013990077152
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1080.7864571076755
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:928.3180429694175
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:882.3920715351578
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:1582.6918115654464
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:1393.2746785980971
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:860.3902855198323
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:849.972020932584
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:973.3399676943767
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:963.670530855375
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:840.0521582057372
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:798.6481857746492
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:1268.6296895958453
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:1095.936273805662
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:728.3063635412338
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:729.4914624365174
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:812.5547702278935
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:814.1391265180854
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:729.5093856800219
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:731.4236603668292
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:1261.369033411508
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:1097.7380996855923
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:718.9681197497446
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:720.1498538909032
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:803.5135553078173
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:805.0643975083694
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:720.1421987398311
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:722.1380394243008
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:1278.9359504851784
```

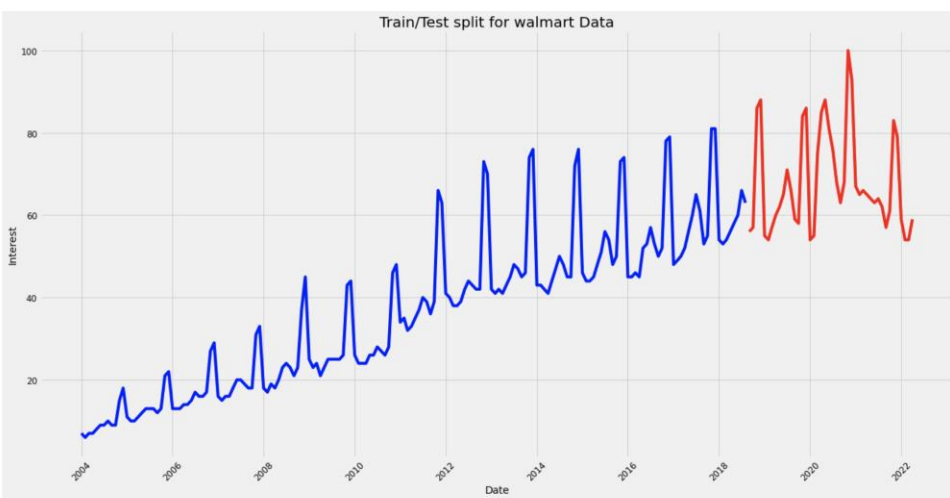
SARIMAX Results

Dep. Variable:	walmart: (United States)	**No. Observations:**	220
Model:	SARIMAX(1, 0, 1)x(0, 1, 1, 12)	**Log Likelihood**	-532.358
Date:	Sat, 30 Apr 2022	**AIC**	1072.716
Time:	23:18:40	**BIC**	1086.066
Sample:	01-01-2004	**HQIC**	1078.114
	- 04-01-2022		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.8501	0.031	27.552	0.000	0.790	0.911
ma.L1	0.2525	0.052	4.829	0.000	0.150	0.355
ma.S.L12	-0.2999	0.039	-7.598	0.000	-0.377	-0.223
sigma2	9.6560	0.495	19.508	0.000	8.686	10.626

Ljung-Box (L1) (Q):	0.41	**Jarque-Bera (JB):**	298.07
Prob(Q):	0.52	**Prob(JB):**	0.00
Heteroskedasticity (H):	4.52	**Skew:**	0.31
Prob(H) (two-sided):	0.00	**Kurtosis:**	8.83

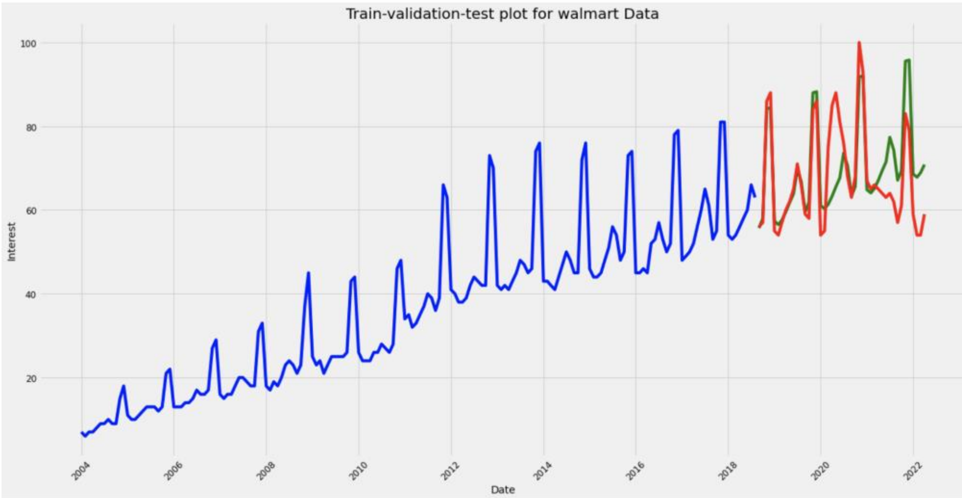
Before predicting test data, here is the visualization of train and validation data.



So far, we have trained our model with data from Jan 2004 to Aug 2018. With our trained model, we will predict the values from Sep 2018 to Apr 2022. Visualization of Actuals and Predictions with training data is plotted as below,

1.1 Time series Model (ARIMA)

Here we have trained our model with original data without differencing to check the goodness of fit.

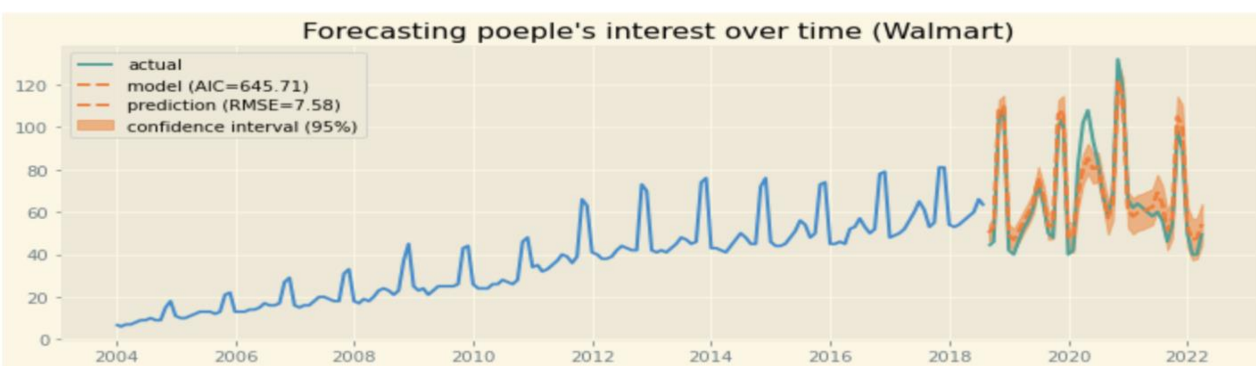


Here, the model looks good. As a next step, we will evaluate this model

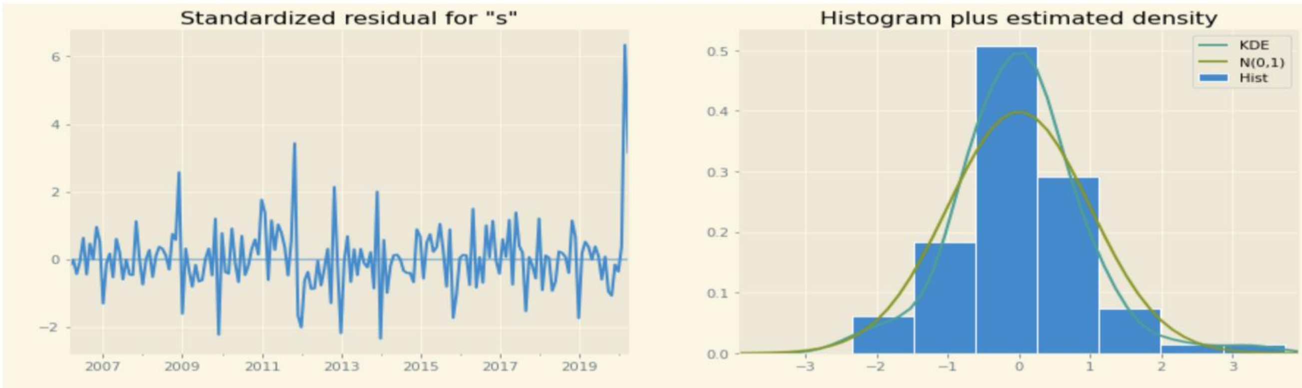
```
# performance evaluation
mse = mean_squared_error(validation, pred)
print('MSE: '+str(mse))
mae = mean_absolute_error(validation, pred)
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(validation, pred))
print('RMSE: '+str(rmse))
```

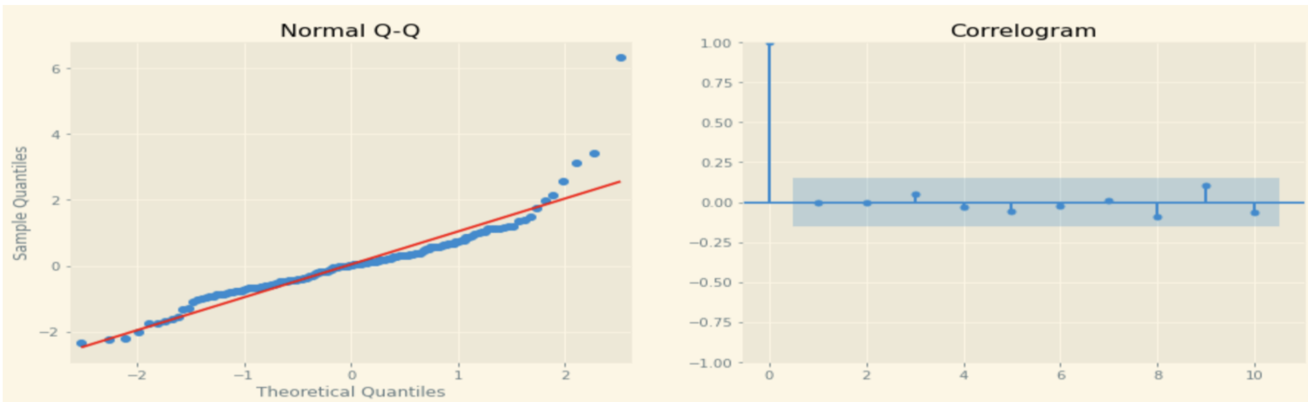
```
MSE: 74.30387229224179
MAE: 6.144261824086427
RMSE: 8.619969390446917
```

RMSE helps us understand the relationship between the actual and predicted values. The lower the AIC the better the model fits. The below forecast model looks good fit. Here the AIC is lesser than any other models.

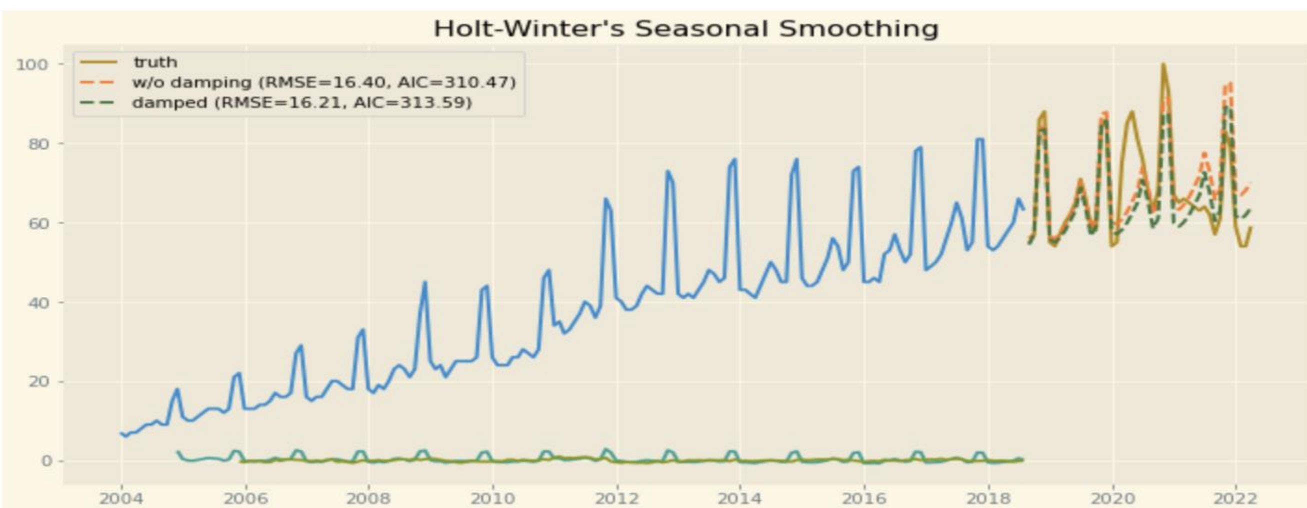


We are using plot_diagnostics() to show that the residuals are normally distributed with histogram and normal Q-Q. Even correlogram indicates that the residual is uncorrelated.





1.2 Exponential Smoothing



On the other hand, we have used holt-winters exponential smoothing to predict the seasonal component. Here we have plotted original data, without damping and with damping. AIC of damped data looks higher than the AIC of data without damping. In overall, we will go with SARIMA (1,0,1)X (0,1,1,12) has best fit model.

importing libraries and changing their name

import warnings

```
warnings.filterwarnings('ignore')
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import matplotlib
from pylab import rcParams
from sklearn.linear_model import LinearRegression
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa import api as smt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
import requests
import pandas as pd
import json
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
get_ipython().run_line_magic('matplotlib', 'inline')
plt.style.use('Solarize_Light2')
data=pd.read_csv('walmart.csv',header=1,parse_dates=True)
data
df = pd.DataFrame(data, columns=['Month','walmart: (United States)']).set_index('Month')
size=len(df)-int(len(df)*0.2)
train = df.iloc[:size, :]
test = df.iloc[size:, :]
train.index = pd.to_datetime(train.index)
test.index = pd.to_datetime(test.index)
```

```
pred = test.copy()

df.plot(figsize=(12,3));

plt.title("  People interest over time  (Walmart-US)");

 from pylab import rcParams

rcParams['figure.figsize'] = 20, 10

decomposition = sm.tsa.seasonal_decompose(train,  model = 'multiplicative')

fig = decomposition.plot()

plt.show()

train['z_data']  = (train['walmart:  (United  States)'] - train['walmart:  (United  States)'].rolling(window=12).mean())  / train['walmart:  (United States)'].rolling(window=12).std()

train['zp_data']  = train['z_data'] - train['z_data'].shift(12)

 def plot_rolling(df):

    fig, ax = plt.subplots(3,figsize=(12,  9))

    dftest = adfuller(train['walmart:  (United States)'], autolag='AIC')

    ax[0].plot(train.index,  train['walmart:  (United States)'],  label='raw data')

    ax[0].plot(train['walmart:   (United States)'].rolling(window=12).mean(),  label="rolling mean");

    ax[0].plot(train['walmart:   (United States)'].rolling(window=12).std(),  label="rolling std (x10)");

    ax[0].legend()

    ax[1].plot(train.index,  train['z_data'],  label="de-trended data")

    ax[1].plot(train['z_data'].rolling(window=12).mean(),   label="rolling mean");

    ax[1].plot(train['z_data'].rolling(window=12).std(),   label="rolling std (x10)");

    ax[1].legend()

    ax[2].plot(train.index,  train['zp_data'],  label="12 lag differenced de-trended data")

    ax[2].plot(train['zp_data'].rolling(window=12).mean(),   label="rolling mean");

    ax[2].plot(train['zp_data'].rolling(window=12).std(),   label="rolling std (x10)");

    ax[2].legend()

    plt.tight_layout()

    fig.autofmt_xdate()

##### import statsmodels.api  as sm

from statsmodels.api   import OLS

x, y = np.arange(len(decomposition.trend.dropna())),  decomposition.trend.dropna()
```

```
x = sm.add_constant(x)

model = OLS(y, x)

res = model.fit()

print(res.summary())

fig, ax = plt.subplots(1, 2, figsize=(12,6));

ax[0].plot(decomposition.trend.dropna().values, label='trend')

ax[0].plot([res.params.x1*i + res.params.const for i in np.arange(len(decomposition.trend.dropna()))])

ax[1].plot(res.resid.values);

ax[1].plot(np.abs(res.resid.values));

ax[1].hlines(0, 0, len(res.resid), color='r');

ax[0].set_title("Trend and Regression");

ax[1].set_title("Residuals");

from statsmodels.tsa.stattools import adfuller

print("Is the data stationary ?")

dftest = adfuller(train['walmart: (United States)'], autolag='AIC')

print("Test statistic = {:.3f}".format(dftest[0]))

print("P-value = {:.3f}".format(dftest[1]))

print("Critical values :")

for k, v in dftest[4].items():

  print("\t{}: {} - The data is {} stationary with {}% confidence".format(k, v, "not" if v<dftest[0] else "", 100-int(k[:-1])))

print("\n Is the de-trended data stationary ?")

dftest = adfuller(train['z_data'].dropna(), autolag='AIC')

print("Test statistic = {:.3f}".format(dftest[0]))

print("P-value = {:.3f}".format(dftest[1]))

print("Critical values :")

for k, v in dftest[4].items():

  print("\t{}: {} - The data is {} stationary with {}% confidence".format(k, v, "not" if v<dftest[0] else "", 100-int(k[:-1])))

print("\n Is the 12-lag differenced de-trended data stationary ?")

dftest = adfuller(train['zp_data'].dropna(), autolag='AIC')
```

```
print("Test statistic = {:.3f}".format(dftest[0]))

print("P-value = {:.3f}".format(dftest[1]))

print("Critical values :")

for k, v in dftest[4].items():

   print("\t{}: {} - The data is {} stationary with {}% confidence".format(k, v, "not" if v<dftest[0] else "", 100-int(k[:-1])))

 fig, ax = plt.subplots(2, figsize=(12,6))

ax[0] = plot_acf(train['z_data'].dropna(), ax=ax[0], lags=20)

ax[1] = plot_pacf(train['z_data'].dropna(), ax=ax[1], lags=20)

import statsmodels.api as sm

from statsmodels.api import OLS

x, y = np.arange(len(decomposition.trend.dropna())), decomposition.trend.dropna()

x = sm.add_constant(x)

model = OLS(y, x)

res = model.fit()

print(res.summary())

fig, ax = plt.subplots(1, 2, figsize=(12,6));

ax[0].plot(decomposition.trend.dropna().values, label='trend')

ax[0].plot([res.params.x1*i + res.params.const for i in np.arange(len(decomposition.trend.dropna()))])

ax[1].plot(res.resid.values);

ax[1].plot(np.abs(res.resid.values));

ax[1].hlines(0, 0, len(res.resid), color='r');

ax[0].set_title("Trend and Regression");

ax[1].set_title("Residuals");




len(df)-size

from statsmodels.tsa.holtwinters import ExponentialSmoothing

model = ExponentialSmoothing(train['walmart: (United States)'], trend="add", seasonal="add", seasonal_periods=12)

model2 = ExponentialSmoothing(train['walmart: (United States)'], trend="add", seasonal="add", seasonal_periods=12, damped=True)
```

```
fit = model.fit()

pred = fit.forecast(len(df)-size)

fit2 = model2.fit()

pred2 = fit2.forecast(len(df)-size)

sse1 = np.sqrt(np.mean(np.square(test['walmart:  (United States)'].fillna(0).values  - pred.values)))

sse2 = np.sqrt(np.mean(np.square(test['walmart:  (United States)'].fillna(0).values  - pred2.values)))

fig, ax = plt.subplots(figsize=(12, 6))

#ax.plot(train.index[0:], train.values[0:]);

ax.plot(test.index, test.values, label='Actuals');

ax.plot(test.index, pred, linestyle='--', color='#ff7823', label="w/o damping (RMSE={:0.2f}, AIC={:0.2f})".format(sse1,
fit.aic));

ax.plot(test.index, pred2, linestyle='--', color='#3c763d', label="damped (RMSE={:0.2f}, AIC={:0.2f})".format(sse2,
fit2.aic));

ax.legend();

ax.set_title("Holt-Winter's Seasonal Smoothing");

 #### SARIMA

from statsmodels.graphics.tsaplots  import plot_acf, plot_pacf

 # Stationary data

train['station'] = train['walmart:  (United States)'] - train['walmart:  (United States)'].rolling(12).mean()

fig, ax = plt.subplots(3, figsize=(12,6))

x = (train.station.dropna()  - train.station.dropna().shift(12)).dropna()

ax[0] = plot_acf(x, ax=ax[0], lags=25)

ax[1] = plot_pacf(x, ax=ax[1], lags=25)

ax[2].plot(x)

print("Possible ARIMA models suggested:")

print('SARIMAX: {} x {}'.format([0,0,1], [0,1,1,12]))

print('SARIMAX: {} x {}'.format([1,0,1], [0,1,1,12]))

print('SARIMAX: {} x {}'.format([1,0,0], [0,1,1,12]))

# set the typical ranges for p, d, q

p = d = q = range(0, 2)

#take all possible combination for p, d and q
```

```
pdq = list(itertools.product(p, d, q))

s_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

# To find optimal parameters for the model

min=0

for param in pdq:

   for p_s in s_pdq:

      try:

         mod = sm.tsa.statespace.SARIMAX(df, order = param, seasonal_order = p_s, enforce_stationary = False,enforce_invertibility=False)

         result = mod.fit()

         print('ARIMA{}x{}12 - AIC:{}'.format(param, p_s, result.aic))

         if min==0 or min>result.aic:

            min=result.aic

            best_param=param

            best_p_s=p_s

            best=result

      except:

         continue

#mod = sm.tsa.statespace.SARIMAX(df, order = best_param, seasonal_order = best_p_s, enforce_stationary = False,enforce_invertibility=False)

#result = mod.fit()

best.summary()

 from statsmodels.tsa.statespace.sarimax import SARIMAX


#train_st = train.ix[:-24, "station"]

#test_st = df.ix[-24:, "station"]

sarima_model = SARIMAX(train['station'], order=(1, 0, 1), seasonal_order=(0, 1, 1, 12), enforce_invertibility=False, enforce_stationarity=False)

sarima_fit = sarima_model.fit()

sarima_pred = sarima_fit.get_prediction("2018-9", "2022-04")

pred_mean=sarima_pred.predicted_mean

predicted_means = pred_mean.values + test['walmart: (United States)'].rolling(12).mean().fillna(0).values
```

```
predicted_intervals = sarima_pred.conf_int(alpha=0.05)

lower_bounds = predicted_intervals['lower station'] + test['walmart: (United States)'].rolling(12).mean().fillna(0).values

upper_bounds = predicted_intervals['upper station'] + test['walmart: (United States)'].rolling(12).mean().fillna(0).values

test['station'] = test['walmart: (United States)'] - test['walmart: (United States)'].rolling(12).mean()

sarima_rmse = np.sqrt(np.mean(np.square(test['station'].fillna(0).values - pred_mean.values)))

fig, ax = plt.subplots(figsize=(12, 4))

ax.plot(train['walmart: (United States)'].index, train['walmart: (United States)'].values);

ax.plot(test['station'].index, test['station'].values + test['walmart: (United States)'].rolling(12).mean().fillna(0).values, label='truth');

ax.plot(test['station'].index, predicted_means, color='#ff7823', linestyle='--', label="prediction (RMSE={:0.2f})".format(sarima_rmse));

ax.fill_between(test['station'].index, lower_bounds, upper_bounds, color='#ff7823', alpha=0.5, label="confidence interval (95%)");

ax.legend();

ax.set_title("Forecasting poeple's interest over time (Walmart)");

sarima_model = SARIMAX(train['station'], order=(1, 0, 1), seasonal_order=(0, 1, 1, 12), enforce_invertibility=False, enforce_stationarity=False)

sarima_fit = sarima_model.fit()

sarima_pred = sarima_fit.get_prediction("2018-09", "2022-04")

pred_mean=sarima_pred.predicted_mean

predicted_means = pred_mean.values + test['walmart: (United States)'].values

predicted_intervals = sarima_pred.conf_int(alpha=0.05)

lower_bounds = predicted_intervals['lower station'] + test['walmart: (United States)'].values

upper_bounds = predicted_intervals['upper station'] + test['walmart: (United States)'].values

test['station'] = test['walmart: (United States)'] - test['walmart: (United States)'].mean()

sarima_rmse = np.sqrt(np.mean(np.square(test['station'].fillna(0).values - pred_mean.values)))

fig, ax = plt.subplots(figsize=(12, 4))

ax.plot(train['walmart: (United States)'].index, train['walmart: (United States)'].values);

ax.plot(test['station'].index, test['station'].values + test['walmart: (United States)'].values, label='actual')

ax.plot(test['station'].index, predicted_means, color='#ff7823', linestyle='--', label="model (AIC={:0.2f})".format(sarima_fit.aic));
```

```
ax.plot(test['station'].index, predicted_means, color='#ff7823', linestyle='--', label="prediction (RMSE={:0.2f})".format(sarima_rmse));

ax.fill_between(test['station'].index, lower_bounds, upper_bounds, color='#ff7823', alpha=0.5, label="confidence interval (95%)");

ax.legend();

ax.set_title("Forecasting poeple's interest over time (Walmart)");

sarima_fit.plot_diagnostics(figsize=(15, 12))
```

2 Time series model (non-Seasonal)

Forecasting Dow Jones Monthly stock prices (1915-1968)

Introduction and Motivation

Dow Jones industrial average is one of the most influenced/watch index. It influences the investors to determine the overall direction of stock prices. For instance, it includes several actively traded company stocks. When Dow Jones index goes up then it is called Bullish, in contrast when Dow Jones index goes down it is called bearish. It means when it goes down, the value of stock prices goes dow n. Here we have Dow Jones data from 1915-1968.

Data Description

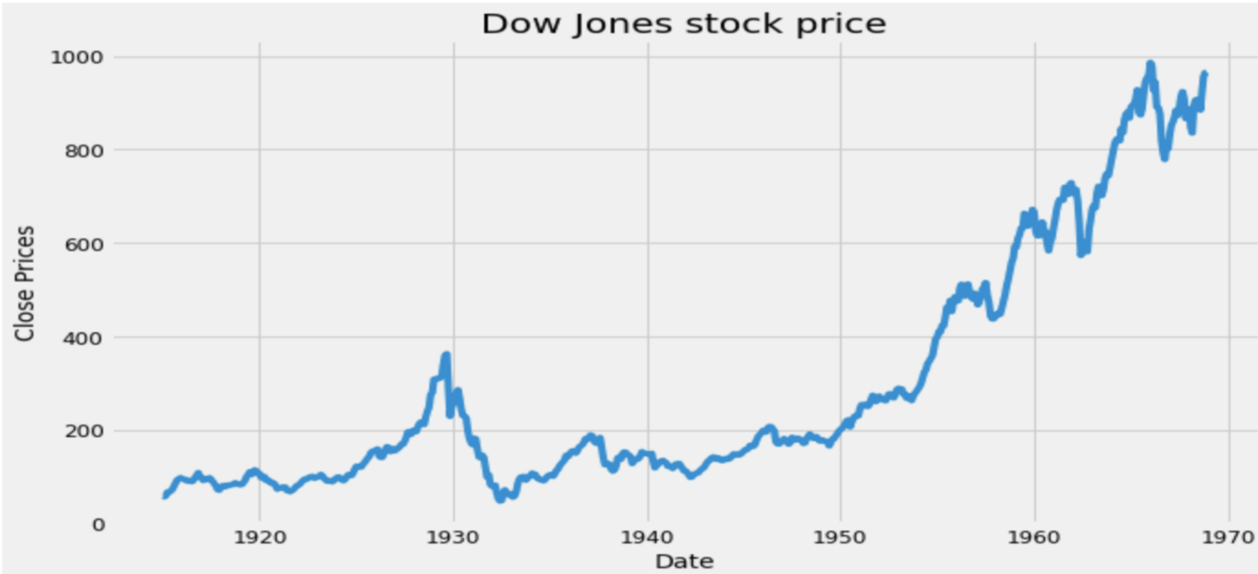
Data Link: [click here]

Date-Data range from (Jan 1915 – Dec 1968)

Stock Index - Monthly stock index data

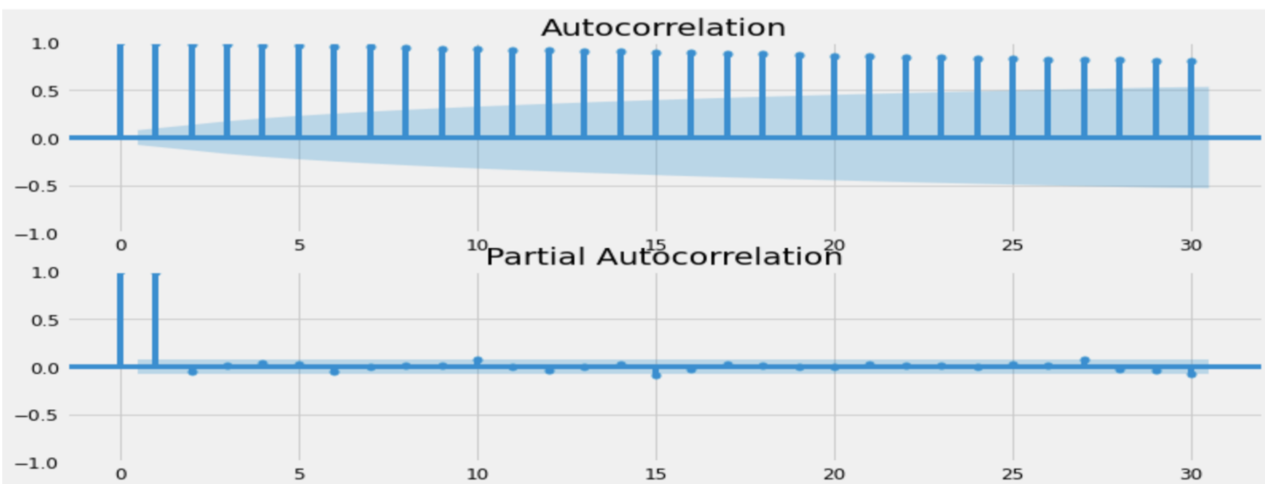
Time Series Analysis

Before the analysis, let us split the data into train and test data to ensure that the test data is not biased. Now we will start analyzing the train data by plotting the continuous variable against date variable to understand stock index over time in United states from Jan 1915 to Dec 1968. From the plot below, we can suggest the data is not seasonal. And there is a clear upward trend from 1940. To determine stationarity, we need to do further tests.



Box-Jenkins Method

Our ACF looks stationary and our PACF suggests AR(1). We will validate the stationarity with ADF(Augmented Dickey Fuller) test.

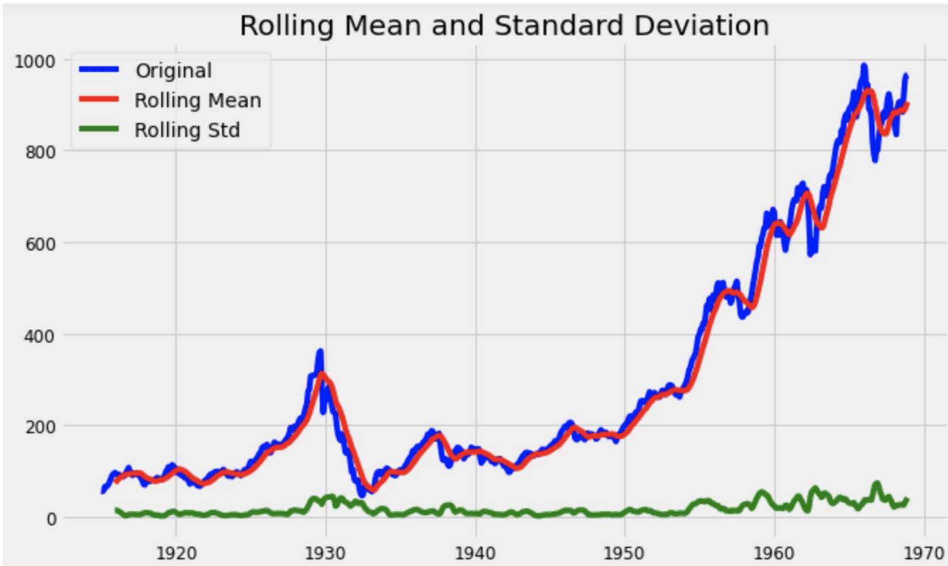



```
from pmdarima.arima.stationarity import ADFTest

# Test whether we should difference at the alpha=0.05
# significance level
adf_test = ADFTest(alpha=0.05)
p_val, should_diff = adf_test.should_diff(data)
print(p_val)
print(should_diff)
```

```
0.9560299866328014
True
```

From the ADF test, P-Value is significantly high and we need to do first order difference and check staionarity.




```
Results of dickey fuller test
Test Statistics                  1.405668
p-value                          0.997140
No. of lags used                10.000000
Number of observations used    637.000000
critical value (1%)             -3.440657
critical value (5%)             -2.866088
critical value (10%)            -2.569192
dtype: float64
```

Mean and Standard deviation is increasing in the above plot, indicating that our series isn't stationary.

We cannot rule out the Null hypothesis because the p-value is bigger than 0.05. Additionally, the test statistics exceed the critical values. As a result, the data is nonlinear.

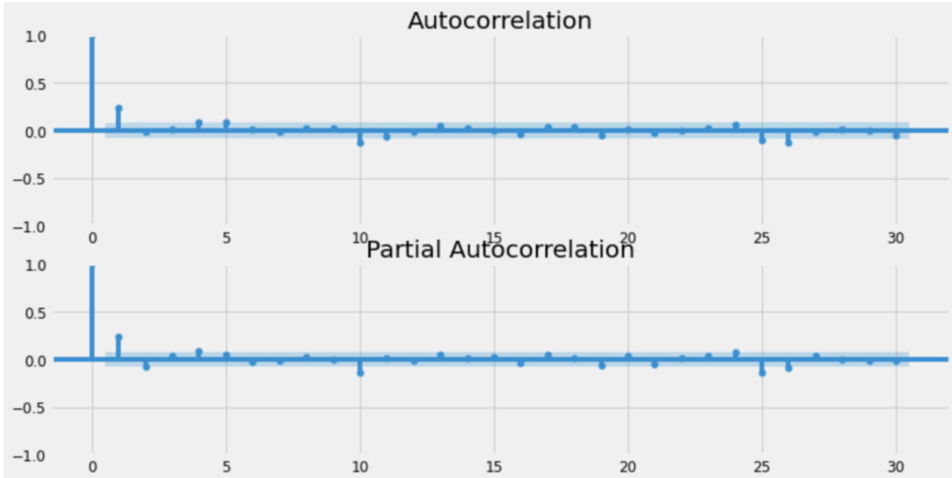
After taking the difference, we can see that the variance of the residual is around the mean and looks good.

```
from pmdarima.arima.stationarity import ADFTest

# Test whether we should difference at the alpha=0.05
# significance level
adf_test = ADFTest(alpha=0.05)
p_val, should_diff = adf_test.should_diff(diff)
print(p_val)
print(should_diff)
```

```
0.01
False
```

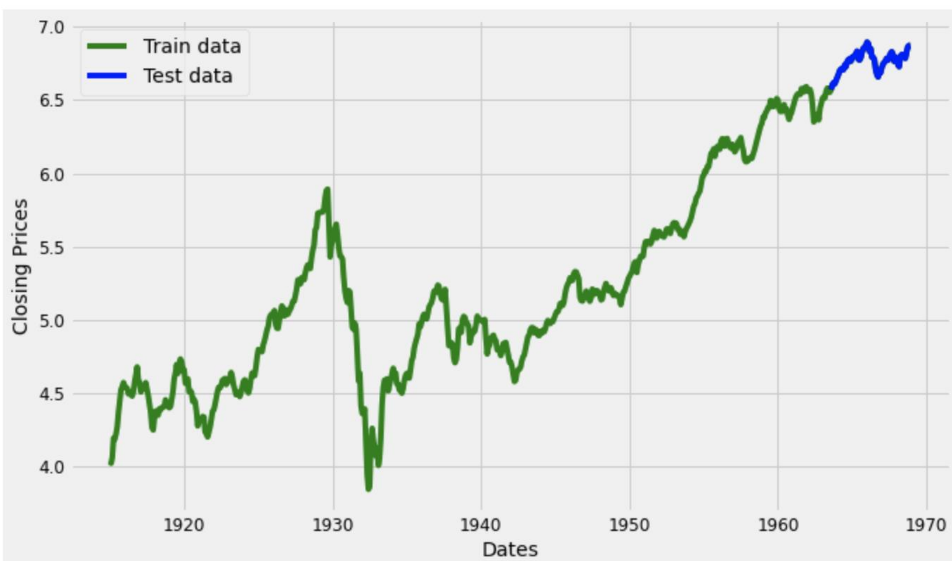
After ADF test with differenced data, p-value is lesser than the significant value. And it indicates no further differencing is required.



After plotting ACF and PACF of differenced data, following are the suggested models.

```
Possible ARIMA models given ACF and PACF
ARIMA : [0,1,0]
ARIMA : [1,1,0]
ARIMA : [0,1,1]
ARIMA : [1,1,1]
```

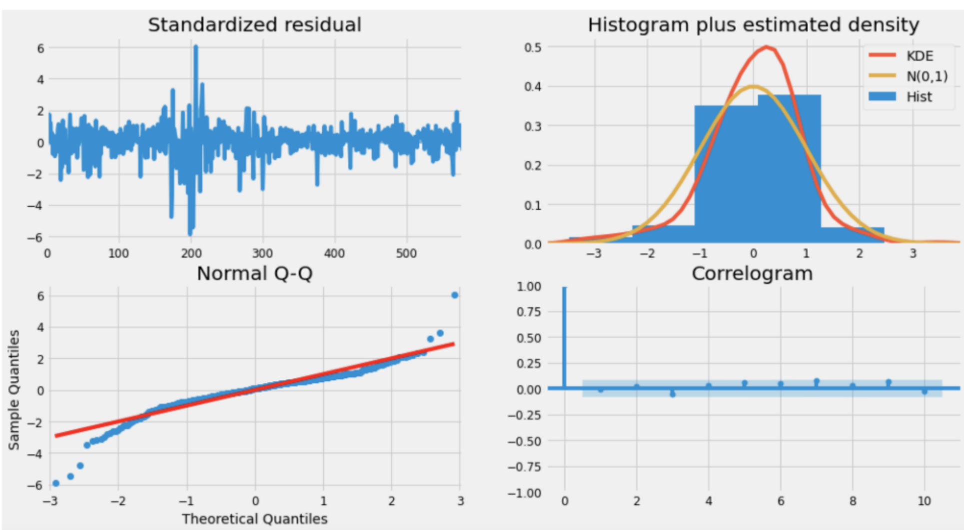
Now we will develop an ARIMA model and train data is using the stock's closing price from the train data. So, let us visualize the data by dividing it into training and test sets.



Here, we have list of potential models generated by the auto arima function with AIC.

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-1835.128, Time=0.12 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-1913.956, Time=0.04 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-1916.633, Time=0.10 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-1833.219, Time=0.06 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-1917.848, Time=0.19 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-1918.489, Time=0.31 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=-1920.045, Time=0.33 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=-1919.120, Time=0.11 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-1918.060, Time=0.45 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=-1919.777, Time=0.08 sec

Best model:  ARIMA(2,1,0)(0,0,0)[0] intercept
Total fit time: 1.803 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  580
Model:               SARIMAX(2, 1, 0)   Log Likelihood                 964.022
Date:                Fri, 29 Apr 2022   AIC                          -1920.045
Time:                        07:26:45   BIC                          -1902.600
Sample:                             0   HQIC                         -1913.243
                                - 580
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0029      0.002      1.427      0.154      -0.001       0.007
ar.L1          0.4034      0.025     15.897      0.000       0.354       0.453
ar.L2         -0.1177      0.025     -4.684      0.000      -0.167      -0.068
sigma2         0.0021   6.04e-05     34.689      0.000       0.002       0.002
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):              1196.30
Prob(Q):                              0.91   Prob(JB):                         0.00
Heteroskedasticity (H):               0.39   Skew:                            -0.73
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.89
```



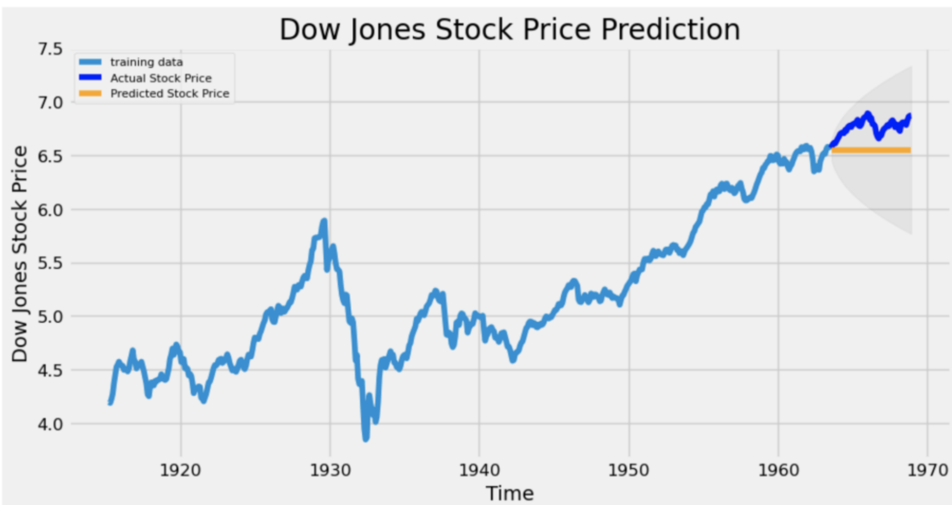
The above residuals plot suggests, variance is along zero and average of the variance fluctuates around zero. The histogram/density plot suggests a normal distribution with zero mean. Normal Q-Q suggests that the data is normally distributed. The residual errors are not autocorrelated, as shown by the Correlogram. Any autocorrelation would imply that the residual errors have a pattern that isn't explained by the model. As a result, the Auto ARIMA model assigned the values 0, 1, and 0 to, p, d, and q, respectively with minimum AIC.

2.1 Time Series Model (ARIMA)

```
#Modeling
# Build Model
model = ARIMA(train_data, order=(0,1,0))
fitted = model.fit()
print(fitted.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    DPS   No. Observations:                  580
Model:                 ARIMA(0, 1, 0)   Log Likelihood                 917.609
Date:                Fri, 29 Apr 2022   AIC                          -1833.219
Time:                        07:26:45   BIC                          -1828.857
Sample:                    04-01-1915   HQIC                         -1831.518
                         - 07-01-1963
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2         0.0025   7.12e-05     34.530      0.000       0.002       0.003
===================================================================================
Ljung-Box (L1) (Q):                  75.90   Jarque-Bera (JB):              1116.73
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.35   Skew:                            -0.84
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.59
===================================================================================
```

We chose ARIMA(0,1,0), as this model seems a good fit to train the data. AIC of this model is lower compared to other models.

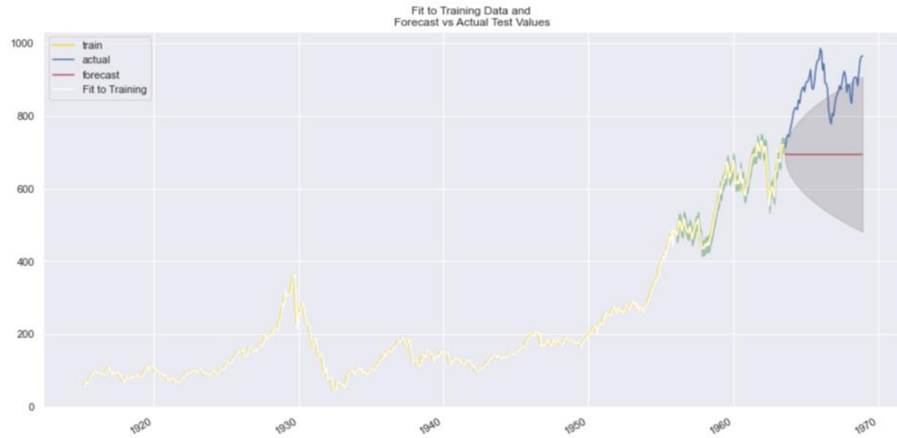


From the above forecast model, we can see that our prediction is closer to the actuals and within the confidence interval. Now it is time to evaluate our model. Our RMSE and MAPE looks satisfactory.

```
MSE: 0.05086938554691149
MAE: 0.2136721057622964
RMSE: 0.22554242515968362
MAPE: 0.03147002685436965
```

2.1 Time Series Model (ARIMA + GARCH)

With the residuals from our fitted ARIMA model we have identified P and Q for Garch model with ACF, PACF plot, ADF test and fitted the model. Below is the plot of ARIMA(0,1,0) + GARCH(1,1) model and the model is not significantly different from our ARIMA(0,1,0) model.



However our RMSE shows that the ARIMA+GARCH performance is slightly better than ARIMA model. But I prefer

```
#fc.summary_frame()["mean"]
# report performance
rmse = math.sqrt(mean_squared_error(test_data, fc.summary_frame()["mean"]))
print('RMSE: '+str(rmse))
```

```
RMSE: 0.18554802553761265
```

importing libraries

import warnings

warnings.filterwarnings('ignore')

import itertools

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.api as sm

import matplotlib

from pylab import rcParams

from sklearn.linear_model import LinearRegression

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

from statsmodels.tsa import api as smt

from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.tsa.stattools import adfuller

from sklearn.metrics import mean_squared_error, mean_absolute_error

import math

```
import numpy as np
from sklearn.metrics import mean_squared_error
from math  import sqrt
from statsmodels.tsa.arima.model   import ARIMA
from pmdarima.arima  import auto_arima
plt.style.use('fivethirtyeight')
matplotlib.rcParams['axes.labelsize']  = 14
matplotlib.rcParams['xtick.labelsize']  = 12
matplotlib.rcParams['ytick.labelsize']  = 12
matplotlib.rcParams['text.color']  = 'k'
#Extracting data into dataframe
data=pd.read_csv('dow_jones.csv',header=1,parse_dates=True)
data.columns=['Month','DPS']  #DPS- Dollar Per Share
data
data['Month']=pd.to_datetime(data['Month'])
# splitting into train and test dataset
size=len(data)-int(len(data)*0.2)
df, validation  = data[0:size], data[size:]
#checking for null values
data.isnull().sum()
 #checking for duplicate values
data[data.duplicated(keep=False)]
ata.set_index('Month', inplace=True)
data.head()
#plot stock price over years
plt.figure(figsize=(10,6))
plt.grid(True)
plt.xlabel('Date')
plt.ylabel('Close Prices')
plt.plot(data['DPS'])
plt.title('Dow Jones stock price')
```

```
plt.show()
#Lets Visualize Distribution of the dataset
df_close = data['DPS']
df_close.plot(kind='kde')
plot_acf(data)
 fig, ax = plt.subplots(2, figsize=(12,6))
ax[0] = plot_acf(data.dropna(), ax=ax[0], lags=30)
ax[1] = plot_pacf(data.dropna(), ax=ax[1], lags=30)
 plot_pacf(data)
from pmdarima.arima.stationarity  import ADFTest
adf_test = ADFTest(alpha=0.05)
p_val, should_diff = adf_test.should_diff(data)
print(p_val)
print(should_diff)
#Test for staionarity
def test_stationarity(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()
    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='green', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)
    print("Results of dickey fuller test")
    adft = adfuller(timeseries,autolag='AIC')
   output = pd.Series(adft[0:4],index=['Test Statistics','p-value','No. of lags used','Number of observations used'])
    for key,values in adft[4].items():
        output['critical value (%s)'%key] =  values
```

```
    print(output)
test_stationarity(df_close)
# Let's isolate the time series from the Trend and Seasonality.
 import statsmodels as sm
import statsmodels.api as sm
result = sm.tsa.seasonal_decompose(data,  model = 'multiplicative')
#fig =
result.plot()
#matplotlib.rcParams['figure.figsize']  = [30,5]
diff = data.diff()
diff.fillna(0,inplace=True)
plt.plot(diff)
plt.show()
from pmdarima.arima.stationarity  import ADFTest
# Test whether we should difference at the alpha=0.05
# significance level
adf_test = ADFTest(alpha=0.05)
p_val, should_diff = adf_test.should_diff(diff)
print(p_val)
print(should_diff)
 fig, ax = plt.subplots(2, figsize=(12,6))
ax[0] = plot_acf(diff, ax=ax[0], lags=30)
ax[1] = plot_pacf(diff, ax=ax[1], lags=30)
 #if not stationary then eliminate trend
#Eliminate trend
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
df_log = np.log(df_close)
moving_avg = df_log.rolling(12).mean()
std_dev = df_log.rolling(12).std()
plt.legend(loc='best')
```

```
plt.title('Moving Average')
plt.plot(std_dev, color ="black", label = "Standard Deviation")
plt.plot(moving_avg, color="red", label = "Mean")
plt.legend()
plt.show()
print('Possible ARIMA models given ACF and PACF')
print('ARIMA : [0,1,0]')
print('ARIMA : [1,1,0]')
print('ARIMA : [0,1,1]')
print('ARIMA : [1,1,1]')
#split data into train and training set
train_data, test_data = df_log[3:int(len(df_log)*0.9)], df_log[int(len(df_log)*0.9):]
plt.figure(figsize=(10,6))
plt.grid(True)
plt.xlabel('Dates')
plt.ylabel('Closing Prices')
plt.plot(df_log, 'green', label='Train data')
plt.plot(test_data, 'blue', label='Test data')
plt.legend()
model_autoARIMA = auto_arima(train_data, start_p=0, start_q=0,
        test='adf',       # use adftest to find optimal 'd'
        max_p=3, max_q=3, # maximum p and q
        m=1,              # frequency of series
        d=None,           # let model determine 'd'
        seasonal=False,   # No Seasonality
        start_P=0,
        D=0,
        trace=True,
        error_action='ignore',
        suppress_warnings=True,
        stepwise=True)
```

```
print(model_autoARIMA.summary())
model_autoARIMA.plot_diagnostics(figsize=(15,8))
plt.show()
#Modeling
# Build Model
model = ARIMA(train_data, order=(0,1,0))
fitted = model.fit()
print(fitted.summary())
 # Forecast
fitted.forecast(len(test_data), alpha=0.05)  # 95% conf
fc=fitted.get_forecast(len(test_data), alpha=0.05).summary_frame()
 fc_series = pd.Series(fc['mean'], index=test_data.index)
lower_series = pd.Series(fc['mean_ci_lower'], index=test_data.index)
upper_series = pd.Series(fc['mean_ci_upper'], index=test_data.index)
# Plot
plt.figure(figsize=(10,5), dpi=100)
plt.plot(train_data, label='training data')
plt.plot(test_data, color = 'blue', label='Actual Stock Price')
plt.plot(fc_series, color = 'orange',label='Predicted Stock Price')
plt.fill_between(lower_series.index, lower_series, upper_series, color='k', alpha=.05)
plt.title('Dow Jones Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Dow Jones Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()
# report performance
mse = mean_squared_error(test_data, fc['mean'])
print('MSE: '+str(mse))
mae = mean_absolute_error(test_data, fc['mean'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(test_data, fc['mean']))
```

```
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(fc['mean']  - test_data)/np.abs(test_data))
print('MAPE: '+str(mape))
# Importing required package
from arch import arch_model

# Building Residuals DataFrame
resid_df = train_data.copy()
resid_df["DPS_resid"] = resid_df["DPS"].shift(1).loc[resid_df.index]
resid_df.at[train_data.index[1]:train_data.index[-1],  "DPS_resid"] = fitted.resid**2

# Defining GARCH(2, 2) model
resid_model = arch_model(resid_df["DPS_resid"][1:], p = 1, q = 1, vol = "GARCH")
# Fitting (Training) the model
resid_model_results = resid_model.fit(last_obs = test_data.index[0], update_freq = 5)
# Displaying the model summary
resid_model_results.summary()
fitted.resid**2
fc   = fitted.get_forecast(len(test_data))
print(fc.summary_frame())
conf = fc.conf_int()

# Transforming the values back to normal
fc_series    = pd.Series(fc.predicted_mean.values, index=test_data.index)
lower_series = pd.Series(conf.iloc[:, 0].values, index=test_data.index)
upper_series = pd.Series(conf.iloc[:, 1].values, index=test_data.index)
# Values to test against the train set, see how the model fits
predictions = fitted.get_prediction(dynamic=False)
pred       = predictions.predicted_mean

# Confidence interval for the training set
```

```
conf_int   = predictions.conf_int()

low_conf   = pd.Series(conf_int.iloc[:,0], index=train_data.index)

upper_conf = pd.Series(conf_int.iloc[:,1], index=train_data.index)

sns.set(rc={'figure.figsize':(16, 8)})

# Plotting the training set, test set, forecast, and confidence interval.

plt.plot(train_data[1:],  label='train', color='gold')

plt.plot(test_data,  label='actual', color='b')

plt.plot(fc_series, label='forecast', color='r')

plt.fill_between(lower_series.index, lower_series, upper_series, color='k', alpha=.15)

# Plotting against the training data

pred[1:].plot(label='Fit to Training', color='w')

# Confidence interval for the fitted data

plt.fill_between(conf_int[-90:].index, conf_int[-90:].iloc[:,0], conf_int[-90:].iloc[:,1], color='g',alpha=.5)

plt.title('Fit to Training Data and \nForecast vs Actual Test Values')

plt.legend()

plt.show()
```

Conclusion

For Seasonal time series analysis, we have achieved best results using SARIMA (1,0,1) X (0,1,1,12) than with Holt winters exponential smoothing. For seasonal data, ARMA+GARCH was not giving better results. Hence, the ARMA+GARCH model is not considered for seasonal dataset. For Non-Seasonal time series analysis, we have achieved better results using ARIMA (0,1,0) + Garch (1,1) than with simple ARIMA (0,1,0) model.

References

[1] https://www.r-bloggers.com/2019/02/performanceanalytics-an-indespensible-quant-tool-for-any-investor/

[2] https://talksonmarkets.files.wordpress.com/2012/09/time-series-analysis-with-arima-e28093-arch013.pdf

[3] https://stats.stackexchange.com/questions/501291/model-specification-for-seasonal-arma-garch-model-using-rugarch