Analysis of Autocorrelation, Heteroskedasticity, Non-Stationarity and Over-Reliance on Normality For Time Series Data

Group No: 361 Submission for Group Work Project No.2

Joel Debo Adriko
Adrikodebojoel1@gmail.com

Gonye Ronald
ronny.gonye@gmail.com

Alper Ulku
alperulku1970@gmail.com

WorldQuant University

1. INTRODUCTION

In this working paper we analyze stock data for their autocorrelation, heteroskedasticity, non-stationarity and over-reliance on normality. We code and run tests for detecting each phenomena and suggested directions for mitigation or treatment of each phenomena to either overcome its adverse effects or making advantage of each, on the way to estimating future values of returns, volatility etc. of the data we desire.

2. AUTOCORRELATION

2.1. Definition and Description

According to Smith, autocorrelation is the mathematical representation of the degree of similarity between a given time series and a lagged version of itself over successive time intervals.

2.1.1. Causes of autocorrelation

- Leaving out relevant exogenous variables in the model.
- Catastrophic natural occurrences such as global pandemics like Covid 19 pandemic, strikes, accidents whose effects can be felt over prolonged periods of time.
- Tendencies of data extrapolation that emanate from traditions and other various forms of behavioral and customary patterns established in the past will often affect affairs in the present moment.
- Manipulation and transformation of data in the form of smoothing, sampling and other treatment methods tend to spread ripple effects over several periods.
- Interconnection of units in an economic system will make effects of a shock that hits one unit to be felt by other units of the same system through their linkages.

2.2. Diagnosis of Autocorrelation: Durbin-Watson Test

2.2.1. Diagnosis 1: Durbin-Watson Test

The Durbin-Watson Test is a measure of autocorrelation (also called serial correlation) in residuals from regression analysis. The DW test statistic is calculated using the following equation:

$$d \approx 2\,(1 - r)$$

$d =$ Durbin-Watson test statistic value

$r =$ Correlation coefficient

Assumptions

The residuals are independent and normally distributed with a mean of zero.

The residuals are stationary.

Hypotheses

H0: The is no autocorrelation among the residuals.

H1: The residuals are autocorrelated.

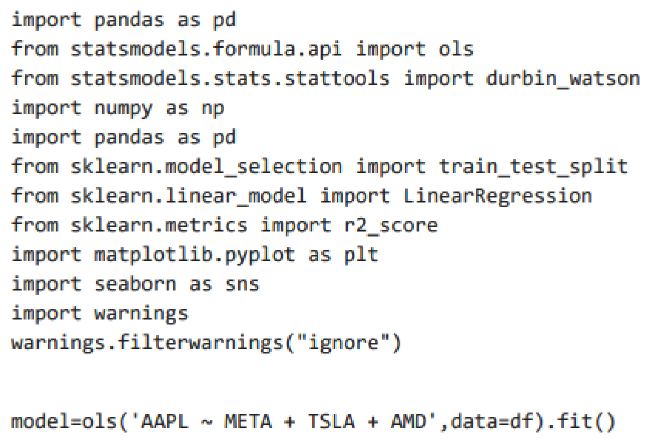
The test statistic has a range between 0 and 4 with the following interpretation:

a) There is no autocorrelation if the test statistic has value of 2.

b) The closer the test statistic is to 0, the more evidence of positive serial correlation.

c) A test statistic value of 4 indicates negative serial correlation.

d) Test statistic values between the range of 1.5 and 2.5 are considered normal (That would mean H0 is accepted, otherwise it is rejected in favor of H1. However, values outside of this range could indicate that autocorrelation would be problematic to the time series and regression models, hence calling for decisive ways to deal with the phenomenon.



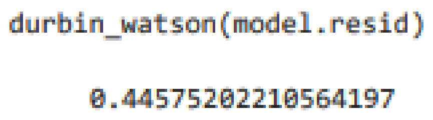
```
import pandas as pd
from statsmodels.formula.api import ols
from statsmodels.stats.stattools import durbin_watson
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")


model=ols('AAPL ~ META + TSLA + AMD',data=df).fit()
```

Code listing for OLS fit

<u>Regression Model of AAPL stock price against three regressor stocks</u>

To demonstate the Durbin-Watson test, stock prices data was used and AAPL was regressed on three chosen stocks namely TSLA, META and AMD. The regression model summary results were as follows:

```
                              OLS Regression Results
==============================================================================

Dep. Variable:                    AAPL   R-squared:                     0.790
Model:                             OLS   Adj. R-squared:                0.783
Method:                  Least Squares   F-statistic:                   123.8
Date:                 Sun, 18 Sep 2022   Prob (F-statistic):         2.23e-33
Time:                         05:41:04   Log-Likelihood:              -243.54
No. Observations:                  103   AIC:                           495.1
Df Residuals:                       99   BIC:                           505.6
Df Model:                            3
Covariance Type:             nonrobust
==============================================================================
                                                                            ==
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     30.9480      7.551      4.099      0.000      15.966      45.930
META           0.1515      0.013     11.224      0.000       0.125       0.178
TSLA           0.1820      0.019      9.632      0.000       0.144       0.219
AMD            0.1162      0.095      1.217      0.226      -0.073       0.306
==============================================================================
==
Omnibus:                        10.854   Durbin-Watson:                 0.446
Prob(Omnibus):                   0.004   Jarque-Bera (JB):              4.392
Skew:                           -0.219   Prob(JB):                      0.111
Kurtosis:                        2.088   Cond. No.                   1.12e+04
==============================================================================
```



```
durbin_watson(model.resid)

0.44575202210564197
```

Tiny code for checking DB independently

A Durbin-Watson test statistic calculated from Python is 0.44575. Since this is outside the range of 1.5 and 2.5, it is not normal. This is positive autocorrelation and its closeness to zero is problematic in the time series model.

2.2.2. Diagnosis 2: Ljung-Box test

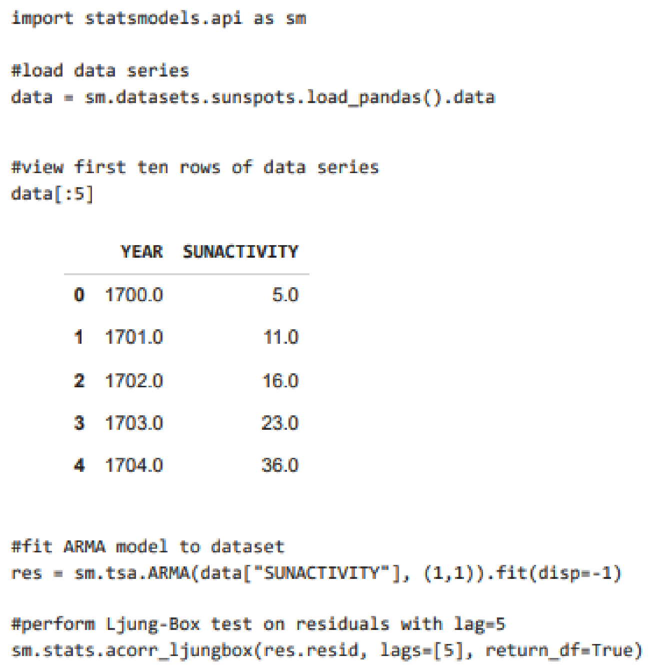
The Ljung-Box test checks the existence of autocorrelation in a time series.

<u>Hypotheses</u>

H0: The residuals are independently distributed.

H1: The residuals are not independently distributed; that is they exhibit serial correlation.

A p-value less than 0.05 leads to rejection of H0 meaning the residuals are not independently distributed or autocorrelation exists. Ljung-Box test performed on built in Python dataset called SUNACTIVITY produced results as follows:



```
import statsmodels.api as sm

#load data series
data = sm.datasets.sunspots.load_pandas().data


#view first ten rows of data series
data[:5]
```

	YEAR	SUNACTIVITY
0	1700.0	5.0
1	1701.0	11.0
2	1702.0	16.0
3	1703.0	23.0
4	1704.0	36.0

```
#fit ARMA model to dataset
res = sm.tsa.ARMA(data["SUNACTIVITY"], (1,1)).fit(disp=-1)

#perform Ljung-Box test on residuals with lag=5
sm.stats.acorr_ljungbox(res.resid, lags=[5], return_df=True)
```

Code listing for LB test

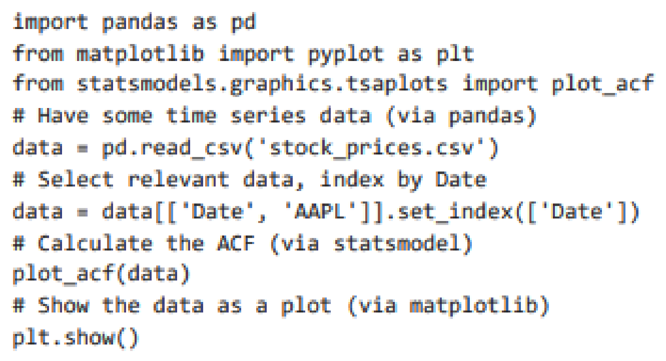
	lb_stat	lb_pvalue
30	456.726189	8.528403e-78

With lag 30, the test statistic is 456.726189 and the p-value of the test is 8.528403 x 10^{-78}, which is much less than 0.05. We therefore reject the null hypotheses and conclude that the residuals are not independent. That is to say autocorrelation exists.

2.3. ACF AND PACF ANALYSES FOR AAPL STOCK PRICES

Autocorrelation is an instrumental tool in identifying statistically significant relationships among observed values in linear data such as detecting patterns of periodicity, seasonality or other sources of influence through the use of autocorrelation function (ACF) and partial autocorrelation function (PACF) plots. These are also applied in technical analysis of stock price data.

ACFs are used to visualize time series movement and in predictive analysis as they display a number of features which represent data metrics.

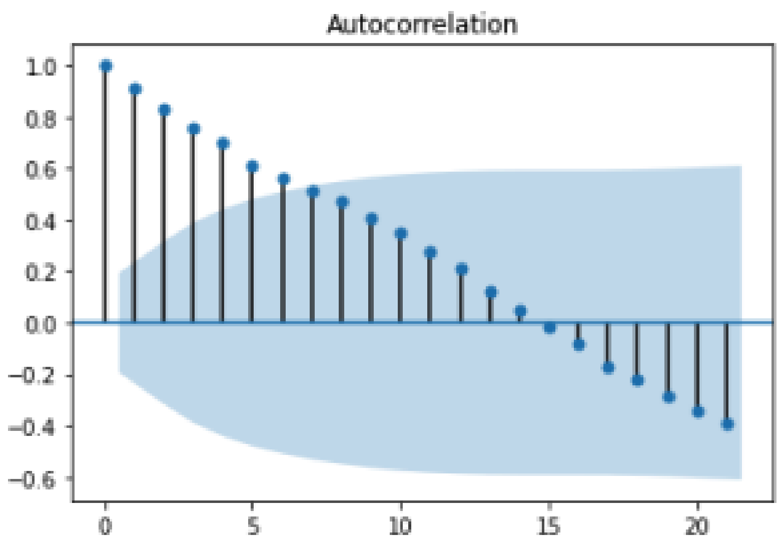


```
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
# Have some time series data (via pandas)
data = pd.read_csv('stock_prices.csv')
# Select relevant data, index by Date
data = data[['Date', 'AAPL']].set_index(['Date'])
# Calculate the ACF (via statsmodel)
plot_acf(data)
# Show the data as a plot (via matplotlib)
plt.show()
```

Code listing for plotting ACF

Typical ACF plot

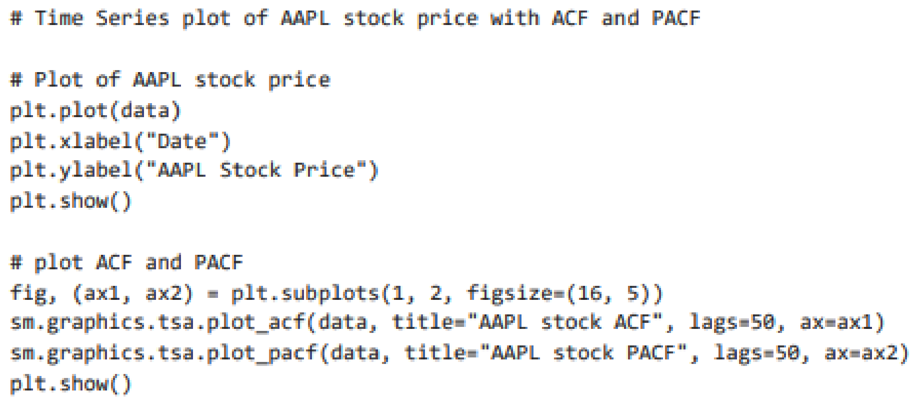
The diagram below illustrates an ACF Plot of the AAPL stock prices.



The region shaded in blue is the 95% confidence interval in which any value here has no significant correlation with the current or most recent stock price value. The top-marked vertical lines are the lags. These lines represent a specific number of previous values the time series is regressed on. On the AAPL autocorrelation function plot above, previous stock prices beyond day 7 have no significant correlation with the current stock price. The y-axis has the correlation scale from -1 to 1 and each lag has its own correlation value.

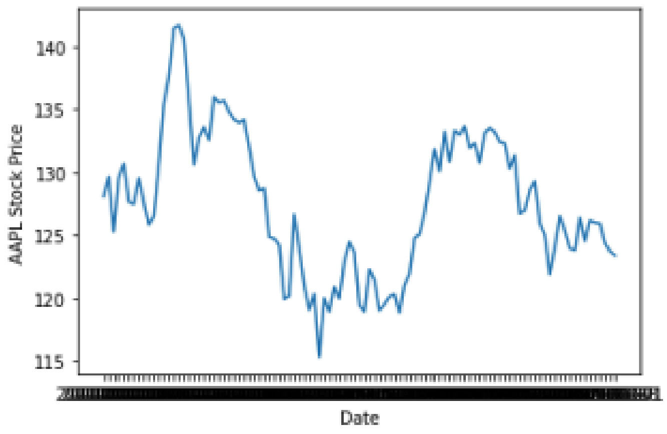
<u>Time Series plot of AAPL stock price with ACF and PACF</u>

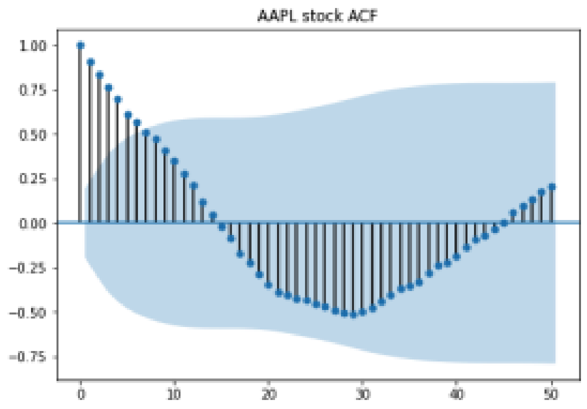
The plots below illustrate the use of autocorrelation in forecasting.

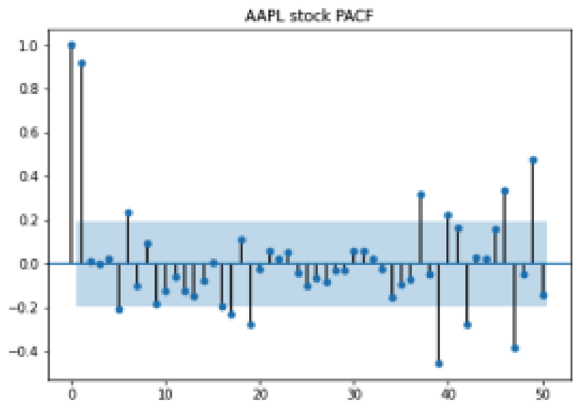


```
# Time Series plot of AAPL stock price with ACF and PACF

# Plot of AAPL stock price
plt.plot(data)
plt.xlabel("Date")
plt.ylabel("AAPL Stock Price")
plt.show()

# plot ACF and PACF
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 5))
sm.graphics.tsa.plot_acf(data, title="AAPL stock ACF", lags=50, ax=ax1)
sm.graphics.tsa.plot_pacf(data, title="AAPL stock PACF", lags=50, ax=ax2)
plt.show()
```

Code listing for plotting ACF and PACF for AAPL







However, as a forecasting technique, it also has shortcomings when it comes to data that generates massive amounts of noise. The noise data points will result in skewed correlation metrics that can falsify or fail to capture the desired influence of the autocorrelation model. (Fukushima, 1985; Willink, 2013).

In light of this, it should be noted that the level of autocorrelation should be ascertained before a time series model is adopted for use for fear that the phenomenon can lead to the following damages:

a) Autocorrelation leads to swelled standard errors and variances resulting in inefficient parameter estimation. There is bias in the determination of the model coefficients.

b) The forecasting power of the model is compromised.

2.4. Treatment of autocorrelation in time series modeling

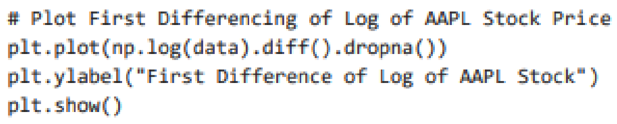
If autocorrelation problem is deemed serious enough in time series modeling, then the following options are explored to maintain the integrity of the model's influence and forecasting power:

1. Addition of lagged endogenous and/or exogenous variables to the model to treat positive serial correlation.

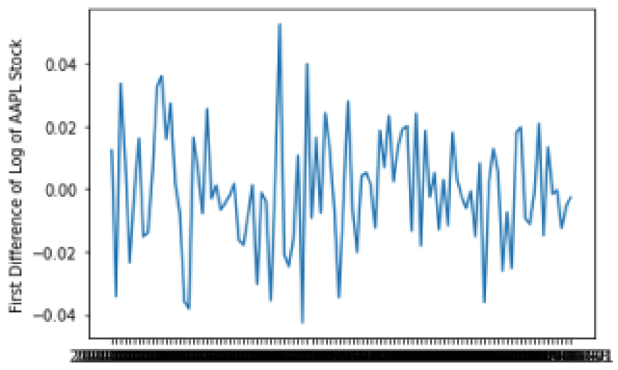
2. Addition of seasonal dummy variables to the model to address seasonal correlation

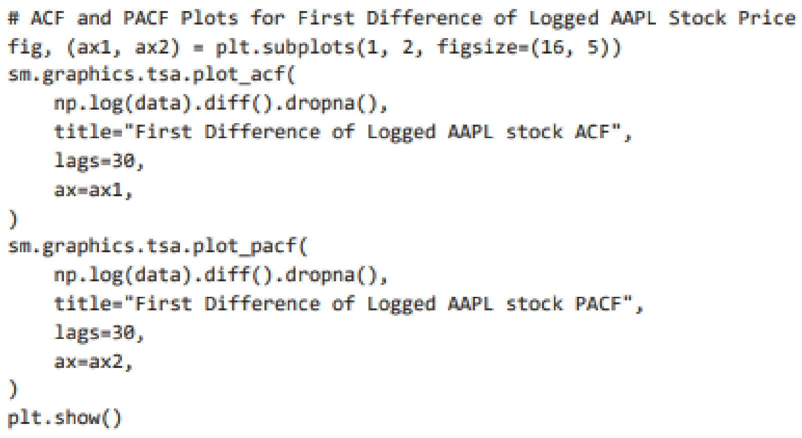
3. For negative serial correlation, checks for over-differenced variables are done. Otherwise with proper differencing order, negative serial correlation is done away with as illustrated below:

2.4.1. Treatment by differencing



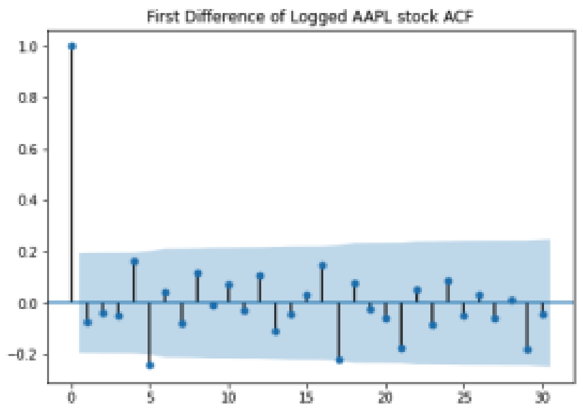
```
# Plot First Differencing of Log of AAPL Stock Price
plt.plot(np.log(data).diff().dropna())
plt.ylabel("First Difference of Log of AAPL Stock")
plt.show()
```

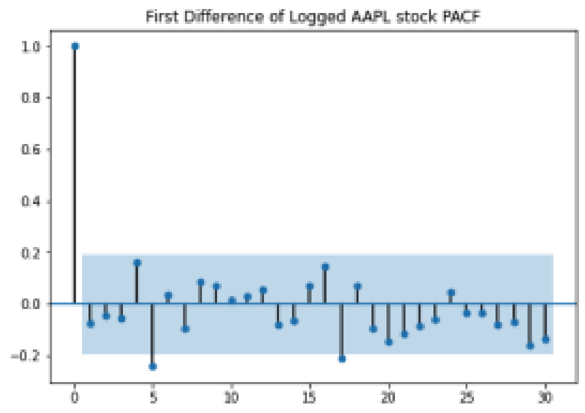




```
# ACF and PACF Plots for First Difference of Logged AAPL Stock Price
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 5))
sm.graphics.tsa.plot_acf(
    np.log(data).diff().dropna(),
    title="First Difference of Logged AAPL stock ACF",
    lags=30,
    ax=ax1,
)
sm.graphics.tsa.plot_pacf(
    np.log(data).diff().dropna(),
    title="First Difference of Logged AAPL stock PACF",
    lags=30,
    ax=ax2,
)
plt.show()
```

Code listing for differencing





From the ACF and PACF of logged AAPL stock price data, it is clearly evident that with first order differencing, all the lagged values are now in the 95% confidence interval region where serial correlation of the values is statistically insignificant. Differencing helps in stabilizing the time series mean by removing the changes in the time series level. This affirms the notion that identifying the right order of differencing is a salve to problematic autocorrelation in time series modeling.

3. NON-STATIONARITY

3.1. Definition

According to Iordanova, non-stationarity is a situation whereby data points in datasets have mean, variances and covariances that change with time. In other words, non-stationarity is the opposite of stationarity. For stationarity the mean function is constant and independent of time, and similarly for stationarity the autocovariance is independent of time for each difference in time.

Non-stationarity is usually common in time series dataset. Time series dataset is a dataset whose data points have date or timestamp attached to them. Time series are common financial datasets such as stock price datasets.

The dataset that usually shows non-stationarity shows the following behaviours or a combination of them:

1. Trends

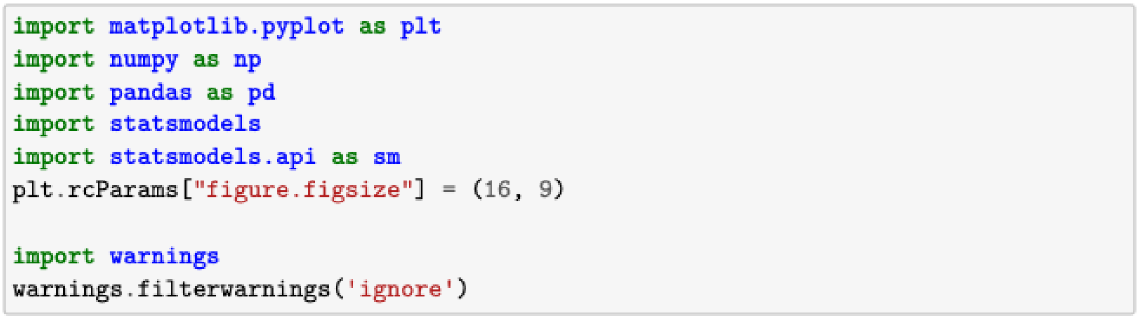
2. Seasonality (cycles)

3. Random walks

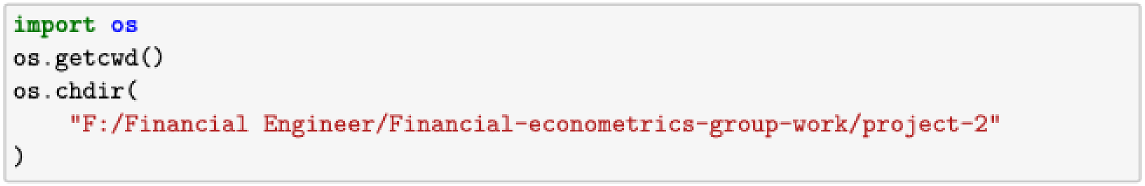
4. Combination of all the three above.

Below we look at each of the above and how to deal with them.

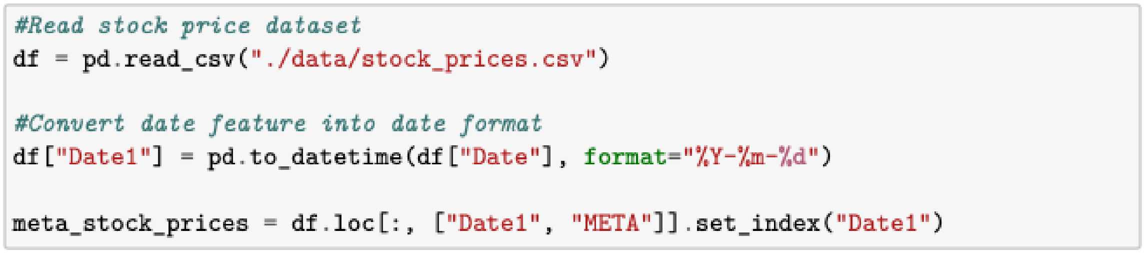
3.2. Demonstration of non-stationarity in meta stock prices



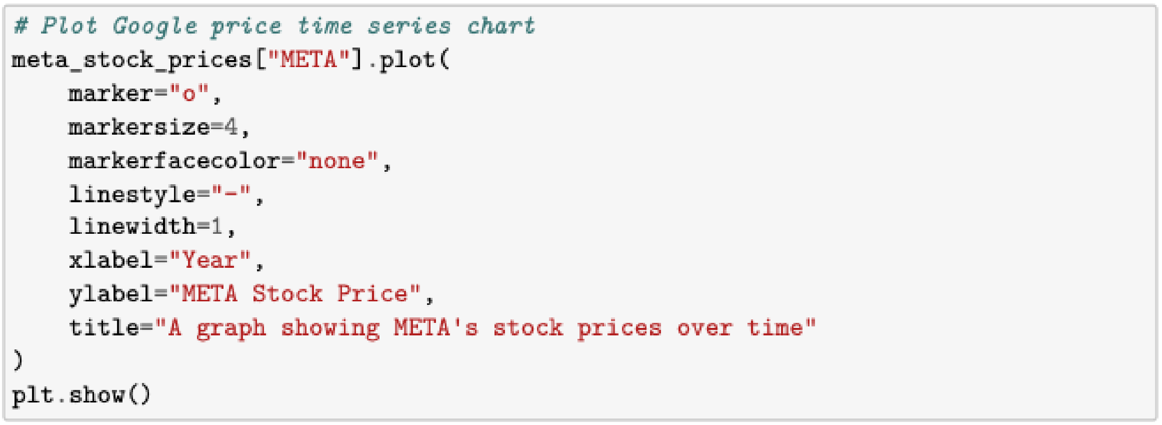
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels
import statsmodels.api as sm
plt.rcParams["figure.figsize"] = (16, 9)

import warnings
warnings.filterwarnings('ignore')
```



```python
import os
os.getcwd()
os.chdir(
    "F:/Financial Engineer/Financial-econometrics-group-work/project-2"
)
```

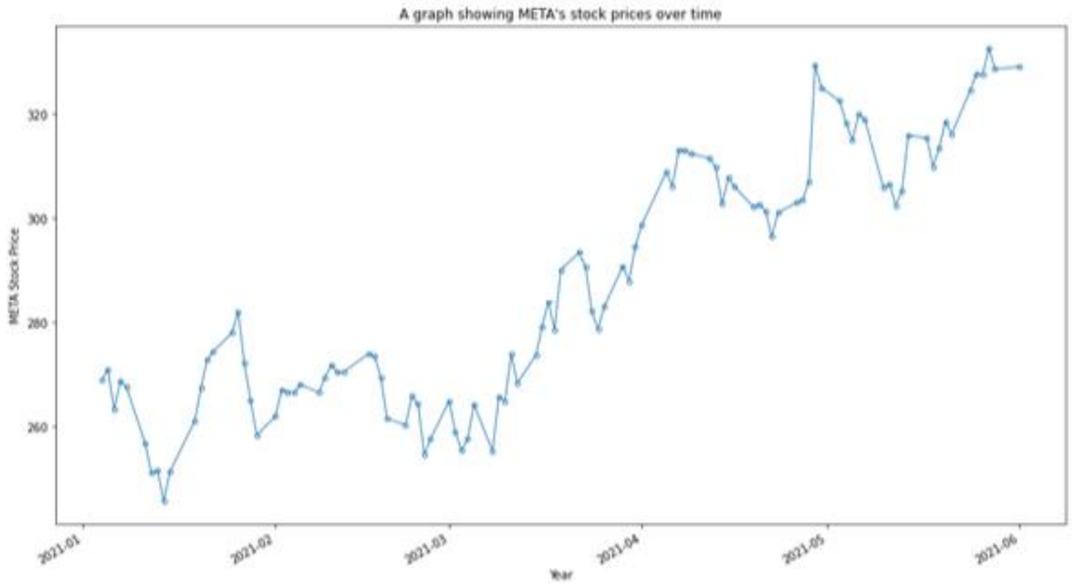


```python
#Read stock price dataset
df = pd.read_csv("./data/stock_prices.csv")

#Convert date feature into date format
df["Date1"] = pd.to_datetime(df["Date"], format="%Y-%m-%d")

meta_stock_prices = df.loc[:, ["Date1", "META"]].set_index("Date1")
```



```python
# Plot Google price time series chart
meta_stock_prices["META"].plot(
    marker="o",
    markersize=4,
    markerfacecolor="none",
    linestyle="-",
    linewidth=1,
    xlabel="Year",
    ylabel="META Stock Price",
    title="A graph showing META's stock prices over time"
)
plt.show()
```

Code listing for META stock data



META stock price of six months

3.2.1. Trend

The trend behaviour of a non-stationary dataset is where the feature of time series dataset grows over time. From the figure above, we can observe that META stock prices generally keep on increasing from January 2021 to June 2021. Therefore, there's a trend in this dataset.

3.2.2. Seasonality

Seasonality is same as cycles and can be defined as a change in a time series dataset over different seasons for example monthly changes. From the graph above, we can observe that there is a cyclic upward movement of the stock prices for example, from mid April-May and May- mid May. This shows that there's seasonality in META's stock prices.

3.2.3. Random Walk

Random walk is a situation whereby the next data values in the time series dataset depend on previous values including their shocks thereby creating a random movement. From the figure above we can observe that the proceeding data values seem influenced by the preceeding data values in that if there's decline in a day, there's a higher chance the next day is still in a decline.

3.2.4. The combination of the three behviours

We can see that the figure above has these three behaviours and therefore is a dataset that shows non-stationarity. In the section below we are going to look at the disadvantages of non-stationarity in dataset and how to handle the associated issues during financial data modeling and forecasting.

3.3. Issues with non-stationary dataset

According to Iordanova, as a rule, non-stationary datasets are unpredictable and cannot be modeled or forecasted. Iordanova continues to explain that building a model out of the non-stationary dataset may give results that show an existence of relationship between two variable when in actual sense such a relationship is inexistent. Therefore, to obtain consistent and reliable results, a non-stationary dataset needs to be trans- formed into stationary dataset. Below we look at these transformations.

3.4. Transforming non-stationary dataset into stationary dataset

There are several methods for transforming a non-stationary dataset into a stationary dataset, for example, according to Patterson and Mills, differencing or cointegration can be used to remove non-stationarity. Below we use the differencing method to remove non-stationarity in META's stock prices.

3.4.1. Trend and differencing method

We can use differencing method to remove trends in datasets. Differencing method involves generating the first-difference time series out of the existing non-stationary time series.

The first-difference can be written as below:

$$\nabla x_t = x_t - x_{t-1}$$

x_t= first observation
x_{t-1}= the subsequent observation after the first observation

∇x_t= first difference

One notable issue with differencing method (from the first-difference) above is that an observation is lost when a difference is taken.

3.4.2. Implementing differencing method (with moving average) on META stock prices

Below we now apply the differencing method to META stock prices. We do this by running two Moving Average (MA) models: one with original META stock prices dataset, then the other with first difference of META's stock prices. We then compare the results. But, first, let's have some brief concepts about Moving Average.

Moving Average, (MA) concepts According to Fernando, in technical financial analysis, mov- ing average is used to indicate shocks in price dataset. The moving average can be the model of choice if the shock in the price dataset has an initial impact that gradually fades in a finite time frame. A shock lasting only for a total time period of q has a moving average MA(q), mathematically written as follow:

$$W_t = W_t + \theta_1 W_{t-1} + \theta_2 W_{t-2} + \cdots + \theta_q W_{t-q}$$

$\theta_1, \theta_2, \cdots, \theta_q$= parameters
W_t= normally distributed white noise

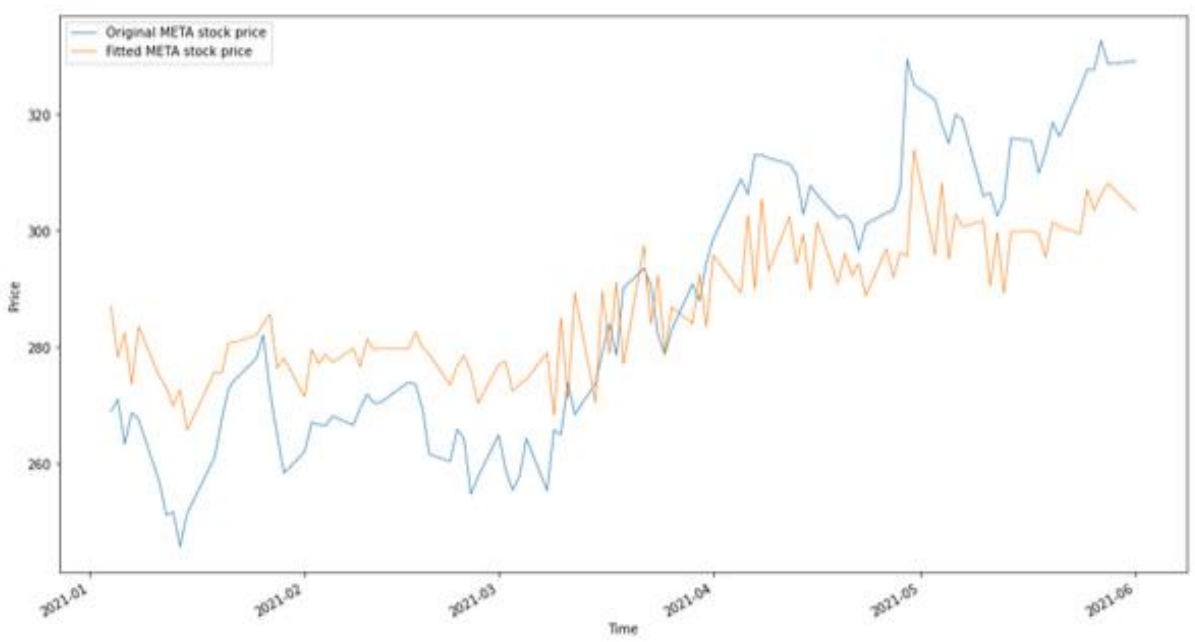
W_t has mean = 0 and variance $\sigma^2 = 1$.

We coded below for Moving Average [MA(1)] model with original META stock prices:

```
# MA(1) for original META stock prices
meta_original = statsmodels.tsa.arima.model.ARIMA(meta_stock_prices.META,␣
  ↪order=(0, 0, 1)).fit()

# Plot Original Meta prices vs fitted Meta stock prices
meta_original_residuals = meta_original.resid
```

```
meta_original_fitted_values = meta_stock_prices["META"] -␣
  ↪meta_original_residuals

meta_stock_prices["META"].plot(linewidth=0.8, label="Original META stock price")
meta_original_fitted_values.plot(linewidth=0.8, label="Fitted META stock price")
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.show()
```

Code listing Moving Average [MA(1)] model



Original vs. Moving Average [MA(1)] model fitted META Stock price chart

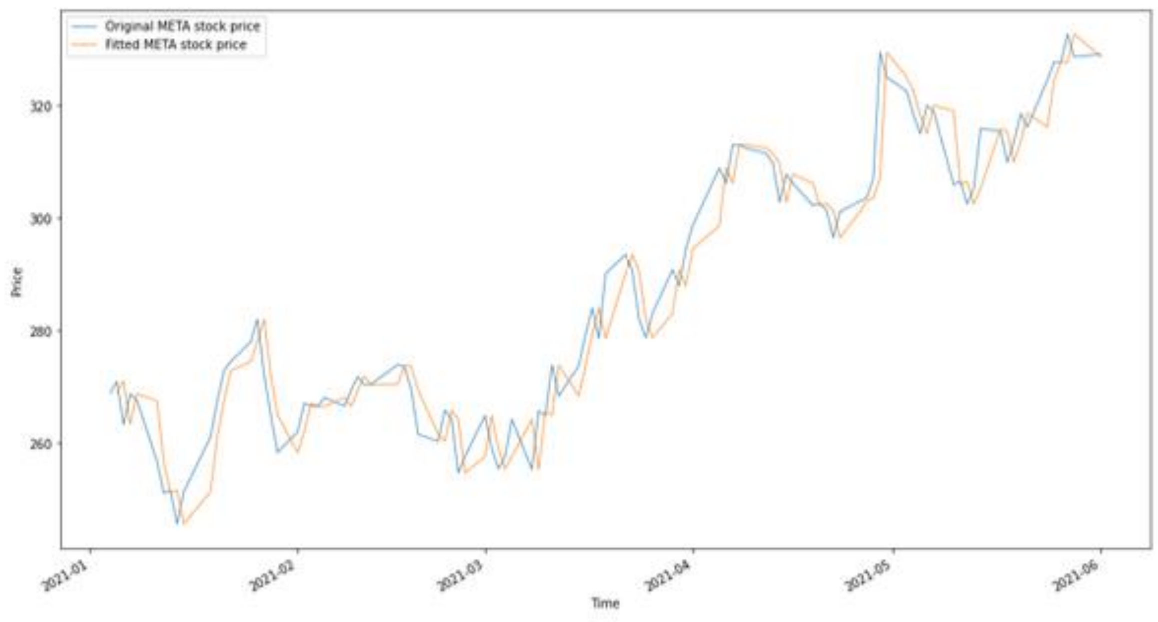
From the figure above, we can observe that the plot from fitted META stock price values doesn’t closely follow the plot from META stock prices. This is probably because the dataset we used was non-stationary.

Below we run a Moving Average [MA(1)] again but this time with the first difference of META's stock prices.

```
# MA(1) for first difference of META stock prices
meta_original = statsmodels.tsa.arima.model.ARIMA(meta_stock_prices.META,␣
 ↪order=(0, 1, 1)).fit()

# Plot Meta prices vs fitted Meta stock prices
#Note that we have to slice out the first element since first difference makes␣
 ↪us to lose one data value
meta_original_residuals = meta_original.resid[1:]
meta_original_fitted_values = (meta_stock_prices["META"] -␣
 ↪meta_original_residuals)[1:]
```

```
meta_stock_prices["META"].plot(linewidth=0.8, label="Original META stock price")
meta_original_fitted_values.plot(linewidth=0.8, label="Fitted META stock price")
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.show();
```

Code listing Moving Average [MA(1)] model with first difference



Original vs. Moving Average [MA(1)] model with first difference fitted META Stock price chart

From the above we can see that the curve from the plots of fitted META stock prices closely follow the curve from the plot of original META prices. This shows that transforming our dataset from non-stationarity state to stationarity state using the first differencing improves the model.

4. OVER-RELIANCE ON NORMALITY

4.1. Definitions and descriptions

Before we can discuss the issue of over-reliance normality in financial modeling, it’s important that we define what normality is and what non-normality is, and we do these below.

4.1.1. Normal and Non-normal distributions

According to Musselwhite and Wesolowski, a normal distribution (also known as a Gaussian distribution) is a hypothetical distribution that’s symmetrical and is used to make comparisons between scores or other types of statistical decisions (2). The normal distribution is a bell-shaped and has mean and median equal. The normal distribution can be constructed using the equation below.

$$X \sim N(\mu_x, \sigma_x^2)$$

Where:-

$$X = \text{normal random variable}$$

$$\mu_x = \text{expected value of X}$$

$$\sigma_x^2 = \text{variance of X}$$

A non-normal distribution is any distribution that doesn’t conform to the properties of normal distribution. These commonly include the distributions that show some skewness (as discussed in project 1 under skewness), fat tailed distributions, among others.

4.1.2. Over-reliance on non-normal distributions

“Over-reliance on normal-distribution” is situation whereby an analyst unnecessarily put emphasis on normal distribution that may negatively affect the results of the model without putting the emphasis on the causes of non-normality and finding alternative models.

Over-reliance on normality can happen at various level, and below we demonstrate the over-reliance on normality by trying to convert the variables by eliminating outliers instead of using methodssuch as robust regressions or weighted least square (WLS) regression. The dataset that we are using is US_dollar_index that we used in Project-1 to handle “SENSITIVITY TO OUTLIER” challenge

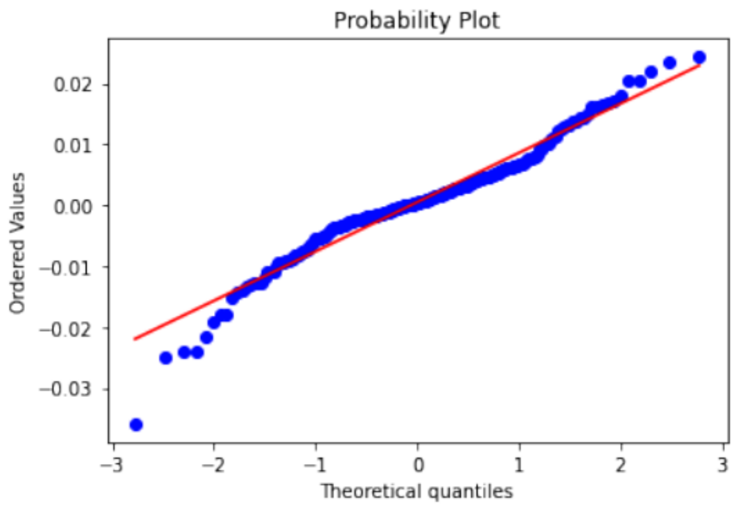
4.1.3. Demonstration

First we check for the normality of both exogeneous variable [S&P 500 Index daily return (US_STK)] and endogenous variable [Dollar Index daily return (DXY)] using QuantileQuantile (QQ) plot and Shapiro Wilk test.

```python
# Shapiro Wilk normality test for S&P 500 Index daily return
shapiro_test = shapiro(df_2["US_STK"])
print(
    "Shapiro W: {0} \nShapiro p-value {1}".format(
        shapiro_test.statistic, shapiro_test.pvalue
    )
)
#QQ-plot for S&P 500 Index daily return
probplot(df_2["US_STK"], dist="norm", plot=pylab)
pylab.show()
```

```
Shapiro W: 0.9578427672386169
Shapiro p-value 1.1078949455622933e-06
```

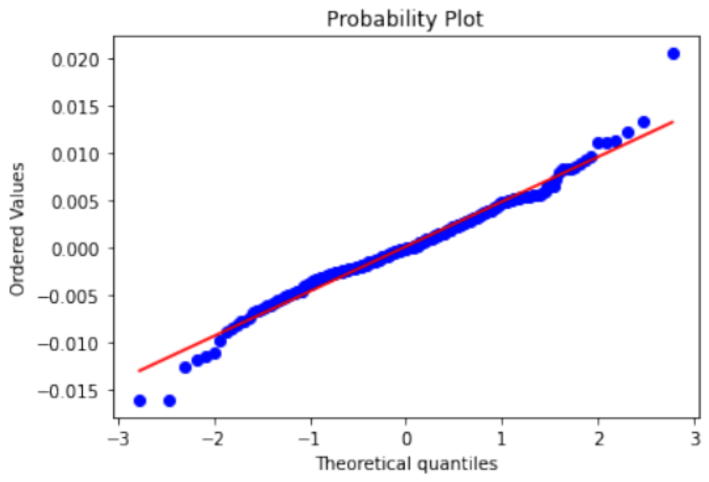
Code listing for Shapiro Wilk normality test



Q-Q plot Shapiro Wilk normality test

```
# Shapiro Wilk normality test for Dollar Index daily return
shapiro_test = shapiro(df_2["DXY"])
print(
    "Shapiro W: {0} \nShapiro p-value {1}".format(
        shapiro_test.statistic, shapiro_test.pvalue
    )
)
#QQ-plot for Dollar Index daily return
probplot(df_2["DXY"], dist="norm", plot=pylab)
pylab.show()
```

```
Shapiro W: 0.9772481322288513
Shapiro p-value 0.00048736194730736315
```



From the QQ-plots above, we can see that the two returns are not following normal distribution. Also from the Shapiro Wilk test the p-values are way below 0.05 so we can reject the null hypothesis (*H*0) that the two variables:Dollar Index daily return (DXY) and S&P 500 Index daily return (US_STK) follow a normal distribution. From Project-1 under "sensitivity to outliers" we saw that running an ordinary least square (OLS) regression gave results below.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    DXY   R-squared:                       0.002
Model:                            OLS   Adj. R-squared:                 -0.002
Method:                 Least Squares   F-statistic:                    0.4260
Date:                Mon, 19 Sep 2022   Prob (F-statistic):              0.515
Time:                        21:01:01   Log-Likelihood:                 983.15
No. Observations:                 250   AIC:                            -1962.
Df Residuals:                     248   BIC:                            -1955.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0002      0.000      0.572      0.568      -0.000       0.001
US_STK        -0.0239      0.037     -0.653      0.515      -0.096       0.048
==============================================================================
Omnibus:                       12.795   Durbin-Watson:                   1.890
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               30.817
Skew:                           0.049   Prob(JB):                     2.03e-07
Kurtosis:                       4.717   Cond. No.                         122.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

We know from Project-1 under “Sensitivity to outliers” that the non-gaussian distribution is due to the outliers. However, we could choose to over rely on normality by droping the outliers so that our dataset approximates normal distribution. Below includes the codes for dropping the outliers.

```
# drop 3% below and above
lower = df_2.DXY.quantile(.03)
upper = df_2.DXY.quantile(.97)
df_3 = df_2[df_2.loc[:,'DXY'] < upper]
df_3 = df_3[df_3.loc[:,'DXY'] > lower]
```

After dropping we test for normality through Shapiro-Wilk test.

```
shapiro_test = shapiro(df_3["DXY"])
print(
    "Shapiro W: {0} \nShapiro p-value {1}".format(
        shapiro_test.statistic, shapiro_test.pvalue
    )
)
```

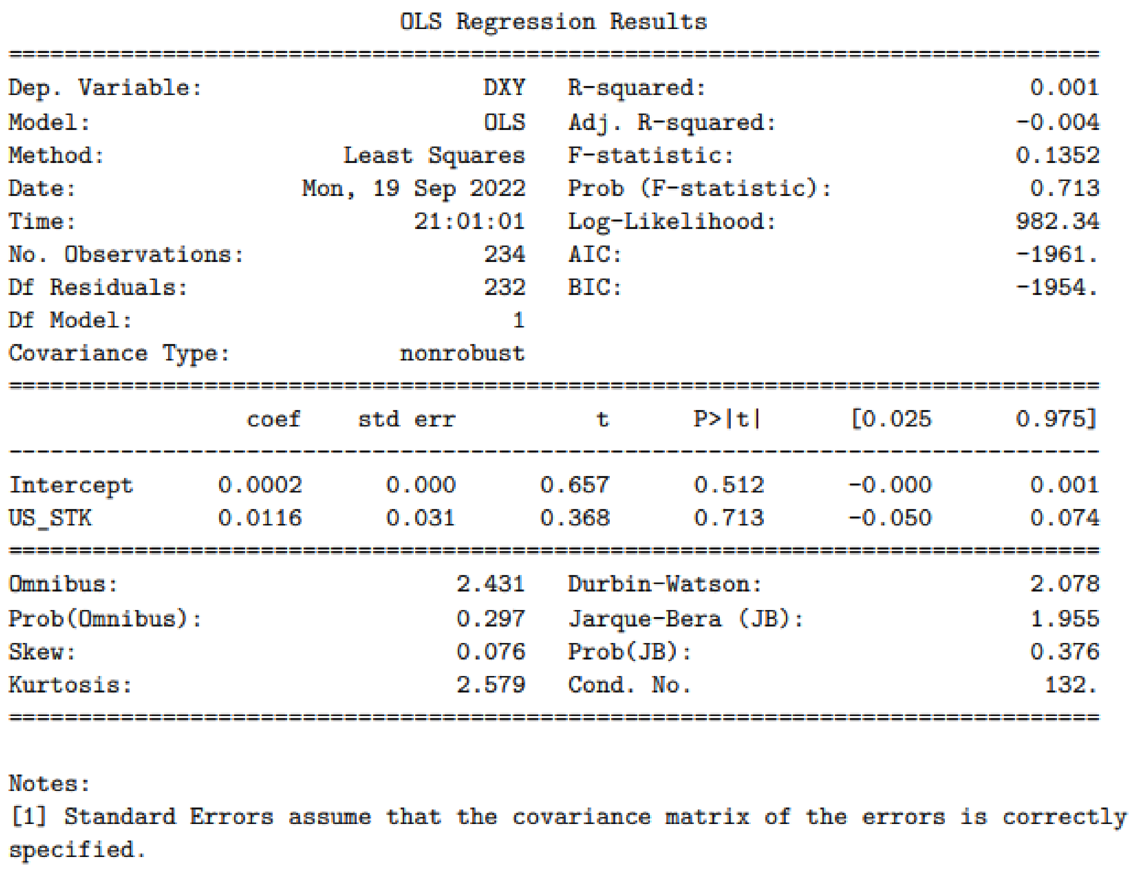
```
Shapiro W: 0.9920908212661743
Shapiro p-value 0.24137508869171143
```

From the test values we can see that the p-value is greater than 0.05 and therefore we accept the null hypothesis ($H0$) that the Dollar Index daily return (DXY) is normally distributed. However, let’s examine the damage.

4.2. Damage caused by over reliance to non-normality

We are going to rerun the OLS model, to see observable damages:

```
# OLS model
model_2 = smf.ols("DXY ~ US_STK", data=df_3).fit()
```

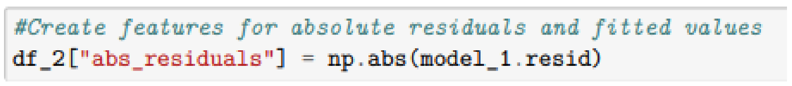


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    DXY   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.004
Method:                 Least Squares   F-statistic:                    0.1352
Date:                Mon, 19 Sep 2022   Prob (F-statistic):              0.713
Time:                        21:01:01   Log-Likelihood:                 982.34
No. Observations:                 234   AIC:                            -1961.
Df Residuals:                     232   BIC:                            -1954.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0002      0.000      0.657      0.512      -0.000       0.001
US_STK         0.0116      0.031      0.368      0.713      -0.050       0.074
==============================================================================
Omnibus:                        2.431   Durbin-Watson:                   2.078
Prob(Omnibus):                  0.297   Jarque-Bera (JB):                1.955
Skew:                           0.076   Prob(JB):                        0.376
Kurtosis:                       2.579   Cond. No.                         132.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

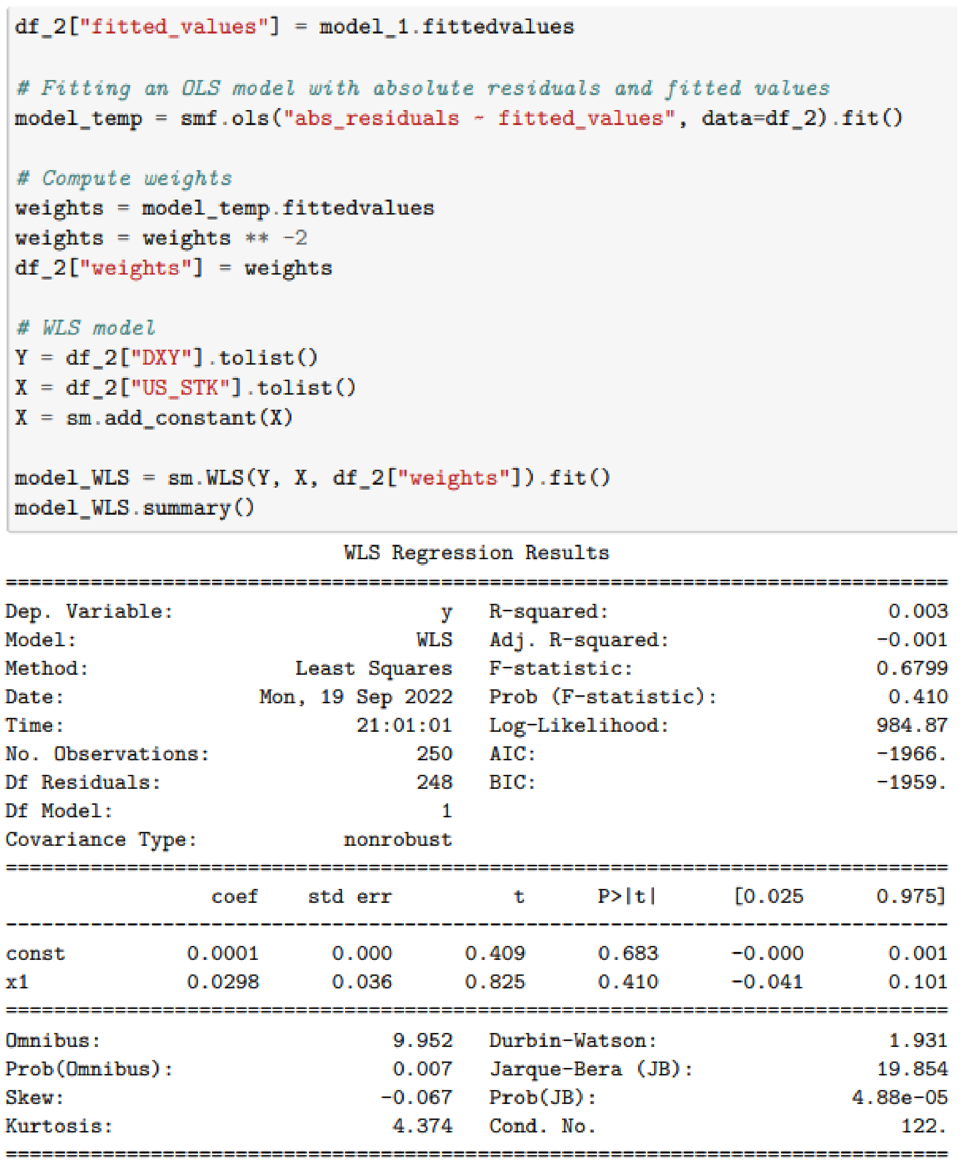
We can observe that the R-squared has dropped from 0.002 to 0.001 when we drop outliers. Similarly, we can also observe that the adjusted R-squared has dropped from -0.002 to -0.004. These two are the observable damages. However, there are also domain-specific damages; for example, removing outliers in financial return leads to wrong modeling since outliers play big part in investment especially if the outliers are purely due to shocks not errors in measurements.

4.3. Directions

Instead of dropping the outliers to normalize the variables, the alternative regression modelling that work well with outliers could be used; for example robust regression models such as Mestimation or Weighted Least Square (WLS) regressions can be used. By taking this approach we are focusing on the causes of non-normality than over-relying on normality. In project-1 under “Sentivity to outlier” we demonstrated how to solve this using WLS model, and for the purpose of this paper the model is below.



```python
#Create features for absolute residuals and fitted values
df_2["abs_residuals"] = np.abs(model_1.resid)
```



```
df_2["fitted_values"] = model_1.fittedvalues

# Fitting an OLS model with absolute residuals and fitted values
model_temp = smf.ols("abs_residuals ~ fitted_values", data=df_2).fit()

# Compute weights
weights = model_temp.fittedvalues
weights = weights ** -2
df_2["weights"] = weights

# WLS model
Y = df_2["DXY"].tolist()
X = df_2["US_STK"].tolist()
X = sm.add_constant(X)

model_WLS = sm.WLS(Y, X, df_2["weights"]).fit()
model_WLS.summary()
```

```
                            WLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.003
Model:                            WLS   Adj. R-squared:                 -0.001
Method:                 Least Squares   F-statistic:                    0.6799
Date:                Mon, 19 Sep 2022   Prob (F-statistic):              0.410
Time:                        21:01:01   Log-Likelihood:                 984.87
No. Observations:                 250   AIC:                            -1966.
Df Residuals:                     248   BIC:                            -1959.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0001      0.000      0.409      0.683      -0.000       0.001
x1             0.0298      0.036      0.825      0.410      -0.041       0.101
==============================================================================
Omnibus:                        9.952   Durbin-Watson:                   1.931
Prob(Omnibus):                  0.007   Jarque-Bera (JB):               19.854
Skew:                          -0.067   Prob(JB):                     4.88e-05
Kurtosis:                       4.374   Cond. No.                         122.
==============================================================================
```

We can observe that using WLS improves the R-squared value from 0.002 to 0.003. This is also three times better than dropping the outliers leave alone the domain specific issues.

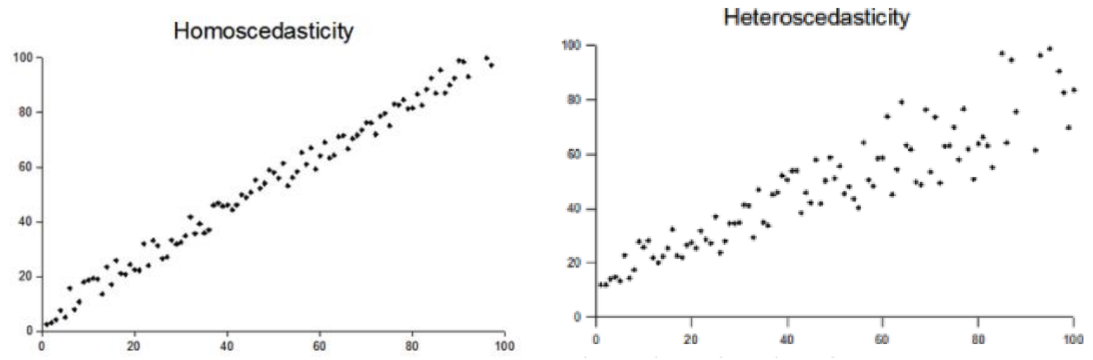
5. HETEROSKEDASTICITY

5.1. Definition:

We consider the linear regression equation,

$$y_i = x_i\beta_i + \epsilon_i,\ i = 1, \ldots, N,$$

where the dependent random variable y_i equals the deterministic variable x_i times coefficient β_i plus a random disturbance term ε_i that has mean zero.



Graphical interpretation of heteroskedasticity

The disturbances are accepted homoscedastic if the variance of ε_i is constant σ^2; otherwise, they are heteroscedastic. In particular, the disturbances are heteroscedastic if the variance of ε_i depends on i or on the value of x_i . They are said to be heteroscedastic if $\sigma_i^2 = x_i\,\sigma^2$ so the variance is proportional to the value of x.

5.2. Description:

According to White, a sequence of random variables is said to be homoscedastic in statistics if each of its random variables has a fixed variance. Another name for this is homogeneity of variance. Heteroscedasticity is the complementary concept meaning variance can change with respect to the independent variable.

When a variable is actually heteroscedastic, assuming it to be homoscedastic leads to either unbiased but ineffective point estimates or biased estimates of standard errors or may also cause an overestimation of the goodness of fit as determined by the Pearson coefficient.

As it invalidates statistical tests of significance that presume that the modeling mistakes all have the same variance, the presence of heteroscedasticity is a key concern in regression analysis and the analysis of variance. The generalized least squares estimator should be used instead of the ordinary least squares estimator since it is more effective and remains unbiased in the presence of heteroscedasticity.

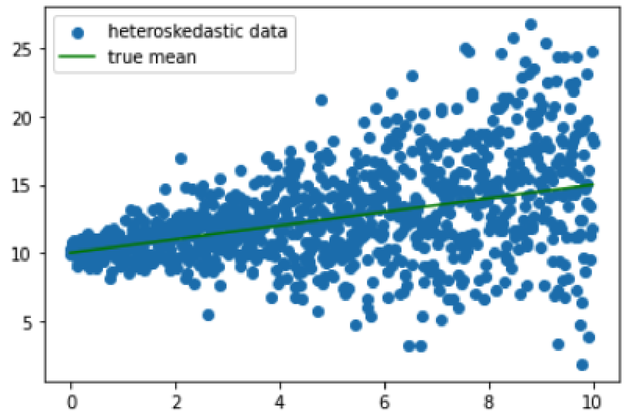
For Engle's research on regression analysis in the presence of heteroscedasticity, which resulted in the creation of the autoregressive conditional heteroscedasticity (ARCH) modeling technique, the econometrician Robert Engle won the 2003 Nobel Memorial Prize for Economics.

5.3. Demonstration, Diagram and Diagnosis of Heteroskedasticity

With the python code below, according to Helm et.al, we generate random data with heteroskedasticity:

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

rng = np.random.RandomState(42)
x = np.linspace(start=0, stop=10, num=1000)
X = x[:, np.newaxis]
y_true_mean = 10 + 0.5 * x
y_heteroskedastic = y_true_mean + rng.normal(loc=0, scale=0.5 + 0.5 * x, size=x.shape[0])

plt.scatter(X,y_heteroskedastic, label='heteroskedastic data')
plt.plot(X,y_true_mean, '-g', label='true mean')
plt.legend()
plt.show()
```

Code for creating heteroskedastic data `y_heteropskedastic`



Plot of heteroskedastic data `y_heteroskedastic`

There are numerous tools for detection of heteroskedasticity. Breusch-Pagan test is one of them. The Breusch-Pagan test is employed to work out whether or not or not heteroscedasticity is present in a regression model. The check uses the subsequent null and various hypotheses:

- Null Hypothesis (H0): Homoscedasticity is present (the residuals are distributed with equal variance)
- Alternative Hypothesis (HA): Heteroscedasticity is present (the residuals aren't distributed with equal variance) If the p-value of the check is a smaller amount than some significance level (i.e. $\alpha = .05$) then we have a tendency to reject the null hypothesis and conclude that heteroscedasticity is gift within the regression model.

We need to run Breusch-Pagan (BP) test and check the p-value of the test for the null hypothesis of data contains heteroskedasticity. If the p-value is leass than 0.05 than heteroskedasticity is detected. Making use of the code presented by Zach, we coded the BP-test as below:

```
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

df = pd.DataFrame(y_heteroskedastic)
df2 = pd.DataFrame(X)
df = pd.concat([df,df2], axis=1)

model = smf.ols('y_heteroskedastic ~ X', data=df).fit()
names = ['Lagrange multiplier statistic', 'p-value', 'f-value', 'f p-value']
test = sms.het_breuschpagan(model.resid, model.model.exog)

lzip(names, test)
```

Code for BP Test

output is:

```
[('Lagrange multiplier statistic', 10.673778282992686),
 ('p-value', 0.0010866493368004045),
 ('f-value', 11.710226310110727),
 ('f p-value', 0.0009081751078028031)]
```

Result of BP test

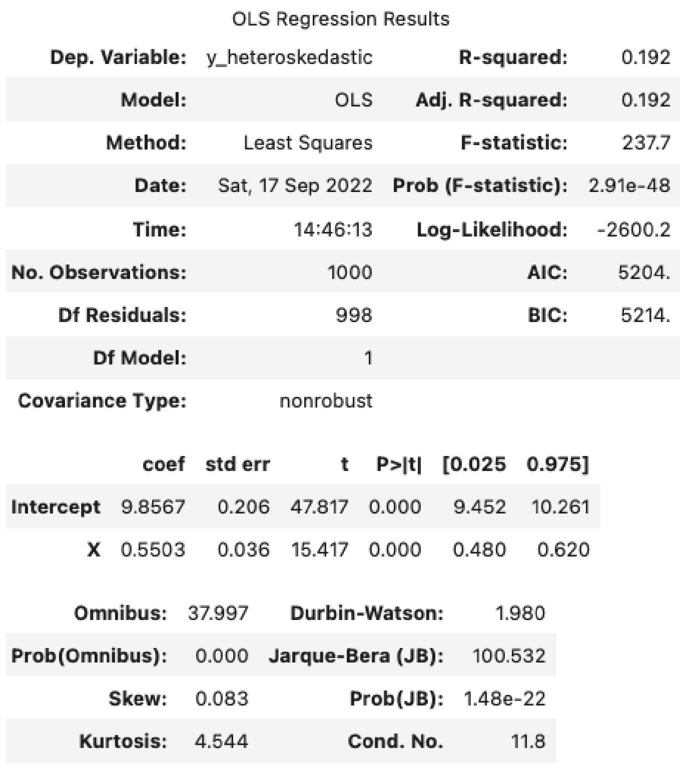
Since p-value is 0.001, we can see that heteroskedasticity is present in data "*y_heteroskedastic*".

5.4. Damage Caused by Heteroskedasticity

According to White, one of the assumptions of the classical statistical regression model is that there's no heteroscedasticity. Breaking this assumption means the Gauss–Markov theorem doesn't apply, that means that OLS estimators aren't the most effective estimators and their variance isn't the bottom of all alternative unbiased estimators. Here is the code for OLS implementation of our heteroskedastic data. Let us observe how the damage's made by varying variance:

```
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from statsmodels.compat import lzip


df = pd.DataFrame(y_heteroskedastic)
df2 = pd.DataFrame(X)
df = pd.concat( [df,df2], axis=1)
#df.columns = ['index','Y-HET','X']
df.columns = ['Y-HET','X']

model = smf.ols('y_heteroskedastic ~ X', data=df).fit()
model.summary()
```

Code for OLS Implementation



OLS Regression Results

Dep. Variable:	y_heteroskedastic	**R-squared:**	0.192
Model:	OLS	**Adj. R-squared:**	0.192
Method:	Least Squares	**F-statistic:**	237.7
Date:	Sat, 17 Sep 2022	**Prob (F-statistic):**	2.91e-48
Time:	14:46:13	**Log-Likelihood:**	-2600.2
No. Observations:	1000	**AIC:**	5204.
Df Residuals:	998	**BIC:**	5214.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	9.8567	0.206	47.817	0.000	9.452	10.261
X	0.5503	0.036	15.417	0.000	0.480	0.620

Omnibus:	37.997	**Durbin-Watson:**	1.980
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	100.532
Skew:	0.083	**Prob(JB):**	1.48e-22
Kurtosis:	4.544	**Cond. No.**	11.8

Results for OLS

Note that f-statistic is low, R^2 value is very low and Jacque Bera result reveals that data was overly non-normal, so we showed that OLS assumptions and Gauss-Markov Theorem can not be applied here.

Heteroscedasticity doesn't cause normal statistical procedure constant estimates to be biased, though it will cause normal statistical procedure estimates of the variance of the coefficients to be biased, probably on top of or below verity of population variance.

Biased commonplace errors cause biased logical thinking, thus results of hypothesis tests square measure probably wrong. for instance, if OLS is performed on a heteroscedastic information set, yielding biased commonplace error estimation, a research worker would possibly fail to reject a null hypothesis at a given significance level, once that null hypothesis was really a typical the particular population (making a type II error).

Under bound assumptions, the OLS computer contains a traditional straight-line distribution once properly normalized and targeted (even once the information doesn't come back from a standard distribution). This result's accustomed justify employing a Gaussian distribution, or a chi sq. distribution (depending on however the check datum is calculated), once conducting a hypothesis check. this is even underneath heteroscedasticity. a lot of exactly, the OLS computer within the presence of heteroscedasticity is asymptotically traditional, once properly normalized and targeted, with a variance-covariance matrix that differs from the case of homoscedasticity.

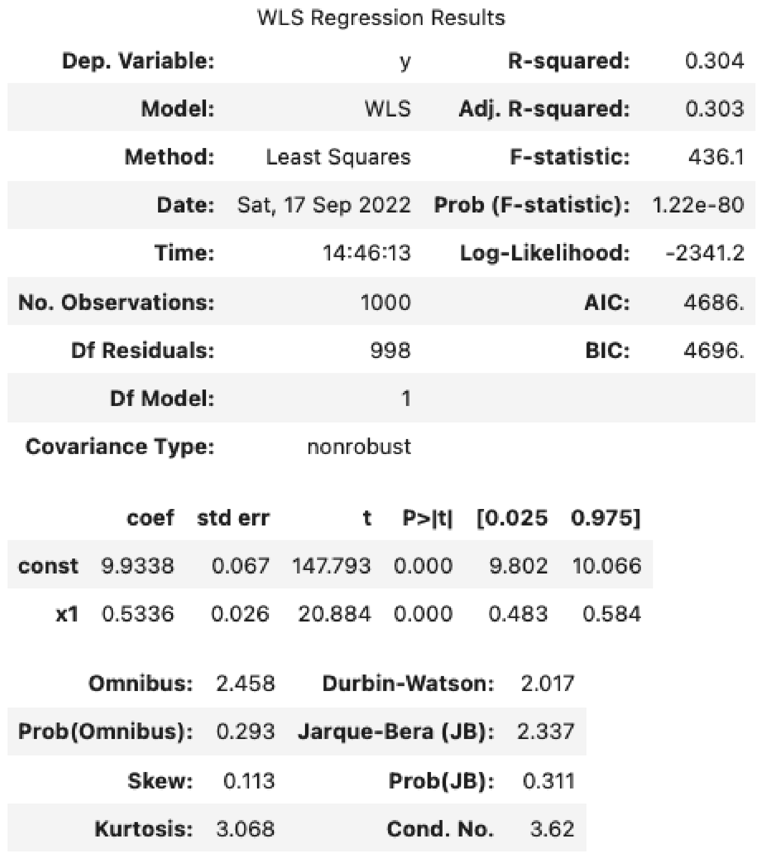
According to Giles, for any non-linear model, heteroscedasticity has a lot of severe consequences: the maximum likelihood estimates (MLE) of the parameters will be biased and inconsistent unless the developed model properly takes the precise kind of heteroscedasticity into consideration.

5.5. Directions for overcoming Heteroskedasticity

According to Tofallis, applying a Weighed Least Squares estimation method overcomes heteroskedasticity and its consequences, within which OLS is applied to reworked or weighted values of X and Y. The weights vary over observations, typically betting on the ever-changing error variances. As a result, the weights become directly associated with the magnitude of the independent variable. Let us implement WLS with the following code:

```
# WLS regression result
import statsmodels.api as sm
# Add Absolute residuals and fitted values to dataset columns
df["abs_residuals"] = np.abs(model.resid)
df["fitted_values"] = model.fittedvalues

# Fit OLS model with absolute residuals and fitted values
model_temp = smf.ols("abs_residuals ~ fitted_values", data=df).fit()

# Compute weights and add it to the data_set column
weights = model_temp.fittedvalues
weights = weights ** -2
df["weights"] = weights

# Fit WLS model
X_N = df['X'].tolist()
Y_N = df['Y-HET'].tolist()
X = sm.add_constant(X_N)  # add a intercept point
print(df)
model_WLS = sm.WLS(Y_N, X, df["weights"]).fit()
model_WLS.summary()
```

Code for WLS Implementation



WLS Regression Results

Dep. Variable:	y	**R-squared:**	0.304
Model:	WLS	**Adj. R-squared:**	0.303
Method:	Least Squares	**F-statistic:**	436.1
Date:	Sat, 17 Sep 2022	**Prob (F-statistic):**	1.22e-80
Time:	14:46:13	**Log-Likelihood:**	-2341.2
No. Observations:	1000	**AIC:**	4686.
Df Residuals:	998	**BIC:**	4696.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	9.9338	0.067	147.793	0.000	9.802	10.066
x1	0.5336	0.026	20.884	0.000	0.483	0.584

Omnibus:	2.458	**Durbin-Watson:**	2.017
Prob(Omnibus):	0.293	**Jarque-Bera (JB):**	2.337
Skew:	0.113	**Prob(JB):**	0.311
Kurtosis:	3.068	**Cond. No.**	3.62

Results for WLS (second iteration)

As a result, Weighed Least Squares method came effective in overcoming the heteroskedasticity of data. R^2 value improved around %50 and normality of the data is improved as Jacque-Bera test results significantly reduced. However after the WLS transformation and regression operation, Jacque-Bera result reduced significantly and data became much nearer to a normal distributed one and behaved better for estimation.

6. CONCLUSION

In this working paper we analyzed stock data for their autocorrelation, heteroskedasticity, non-stationarity. We had run tests the Augmented Dickey-Fuller, Durbin-Watson and Breusch-Pagan tests for detecting each one. We suggested remedies for each of the phenomena to overcome the adverse effects, as we aim to estimate future values of returns, volatility of the said data.

7. REFERENCES

Adriko, J.D., Ronald, G., and Ulku, A., "Project 1| Group 361." World Quant University | Group Project, 7 Sept. 2022.

"Econometric Theory." *Wikibooks, Open Books for an Open World*, https://en.m.wikibooks.org/wiki/Econometric_Theory.

Engle, Robert F. "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation." *Econometrica*, vol. 50, no. 4, 1982, p. 987., https://doi.org/10.2307/1912773.

Fernando, Jason. "Moving Average (MA): Purpose, Uses, Formula, and Examples." Investopedia, Investopedia, 8 Sept. 2022, https://www.investopedia.com/terms/m/movingaverage.asp.

Fisher, Walter D., and Carl F. Christ. "Econometric Models and Methods." *Journal of Farm Economics*, vol. 49, no. 4, 1967, p. 952., https://doi.org/10.2307/1236960.

Giles, Dave. "Robust Standard Errors for Nonlinear Models." *Robust Standard Errors for Nonlinear Models*, 1 Jan. 1970, https://davegiles.blogspot.com/2013/05/robust-standard-errors-for-nonlinear.html.

Helm, et al. "How to Generate Heteroskedastic Data for Linear Regression Analysis given Y." *Cross Validated*, 1 Nov. 1963, https://stats.stackexchange.com/questions/221716/how-to-generate-heteroskedastic-data-for-linear-regression-analysis-given-y.

Iordanova, Tzveta. "An Introduction to Non-Stationary Processes." Investopedia, 8 Feb. 2022, https://www.investopedia.com/articles/trading/07/stationary.asp

Musselwhite, Dorothy J., and Brian C. Wesolowski. "Normal Distribution." The SAGE Encyclopedia of Educational Research, Measurement, and Evaluation, 2018, https://doi.org/10.4135/9781506326139.n476.

Smith, Tim. "What Is Autocorrelation?" *Investopedia*, Investopedia, 8 July 2022, https://www.investopedia.com/terms/a/autocorrelation.asp.

Patterson, Kerry, and Terence C. Mills. Palgrave Handbook of Econometrics. Palgrave Macmillan, 2006.

White, Halbert. “A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity.” *Econometrica*, vol. 48, no. 4, 1980, p. 817., https://doi.org/10.2307/1912934.

Willink, Tricia J. “Limits on Estimating Autocorrelation Matrices from Mobile Mimo Measurements.” *International Journal of Antennas and Propagation*, vol. 2013, 2013, pp. 1–6., https://doi.org/10.1155/2013/345908.

Zach. “How to Perform a Durbin-Watson Test in Python.” *Statology*, 21 Jan. 2021, https://www.statology.org/durbin-watson-test-python/.

Zach. “How to Perform a Breusch-Pagan Test in Python.” *Statology*, 31 Dec. 2020, https://www.statology.org/breusch-pagan-test-python/.