<u>Multivariate Modeling</u>

DATS 6450 (Spring 2020)

Instructor: Dr. Reza Jafari

<u>*Final Project Report*</u>
<u>*Time Series Analysis of*</u>
<u>*Personal Consumption Expenditure*</u>
<u>*(US Economy)*</u>

Submitted by: Amna Gul

Date: Apr 22nd, 2020

Table of Contents:

<u>Abstract:</u>
For the purpose of time series analysis, monthly personal expenditure data of US population/economy was used. The end goal was to apply different time-series models (Average, Naïve, Drift, Simple Exponential Smoothing, Holt-Linear, Holt-Winter, Linear Regression and Auto Regressive Moving Average (ARMA)) on the data set and then choose the one that best forecasts the future values. It turned out that ARMA(1,0) predicted values that most closely resembled the actual data.

<u>Introduction:</u>
From macro-economic perspective Personal Consumption Expenditure serves as a key indicator of Gross Domestic Product (GDP) of a country and hence serves as an important measure of economic growth. So in this project we would try to predict Personal Consumption Expenditure (PCE) of US Population using different models.

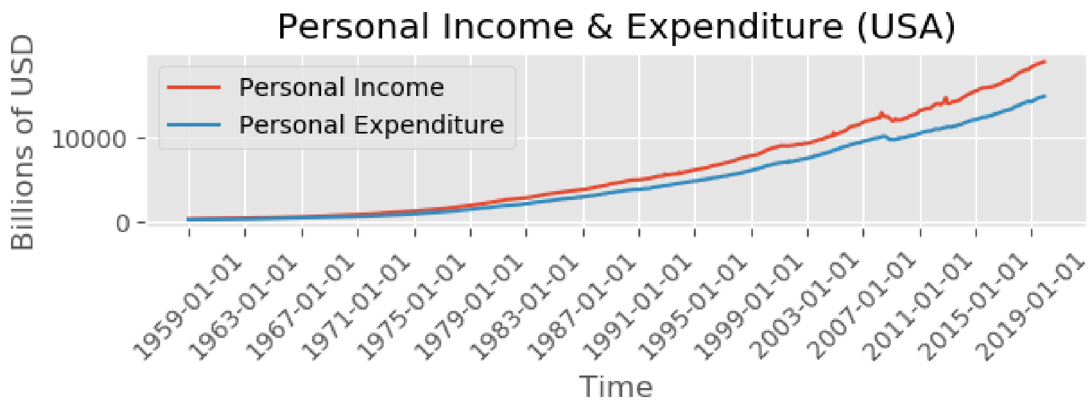
<u>Description of Data set:</u>
Source for the data set that we are going to use is Federal Reserve Economic Data (FRED). We have downloaded 3 separate monthly time series files. A brief description of each file is given below:

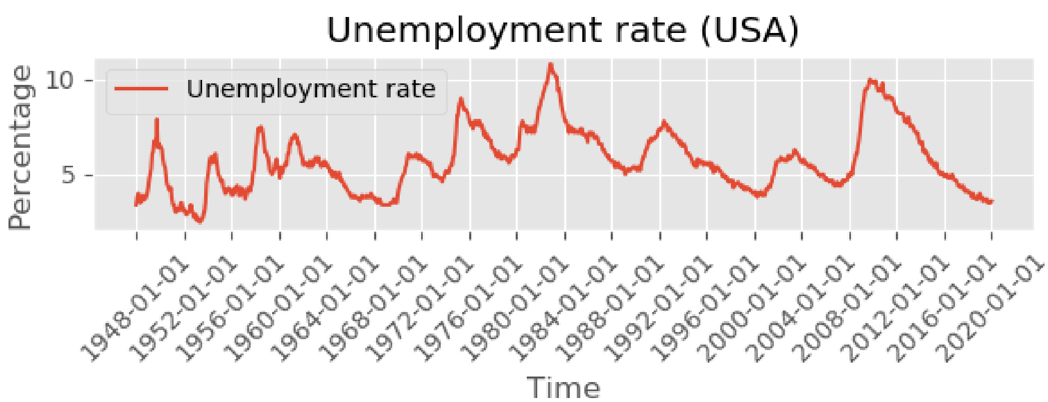
<u>Personal Income</u>: Income that person receives in return for their provision of land, labor, capital (Unit: Billions of USD)
Personal Consumption Expenditures: Money spent by Americans on their everyday goods and services (Unit: Billions of USD)
Unemployment Rate: Number of unemployed as percentage of labor force.

<u>Dependent and Independent Variables</u>:
Personal Consumption Expenditure (PCE) is the dependent variable whereas Personal Income (PI) and Unemployment rate (UNRATE) are independent variables.

<u>Raw Data:</u>

Time series plot of these files is given below (UNRATE has been plotted separately because of difference in scales with PI and PCE). We can see that the variables have different starting and ending dates e.g. PI and PCE data starts in 1959 whereas UNRATE starts in 1948. Similarly PI and PCE data ends in 2019 whereas UNRATE data ends in year 2020.

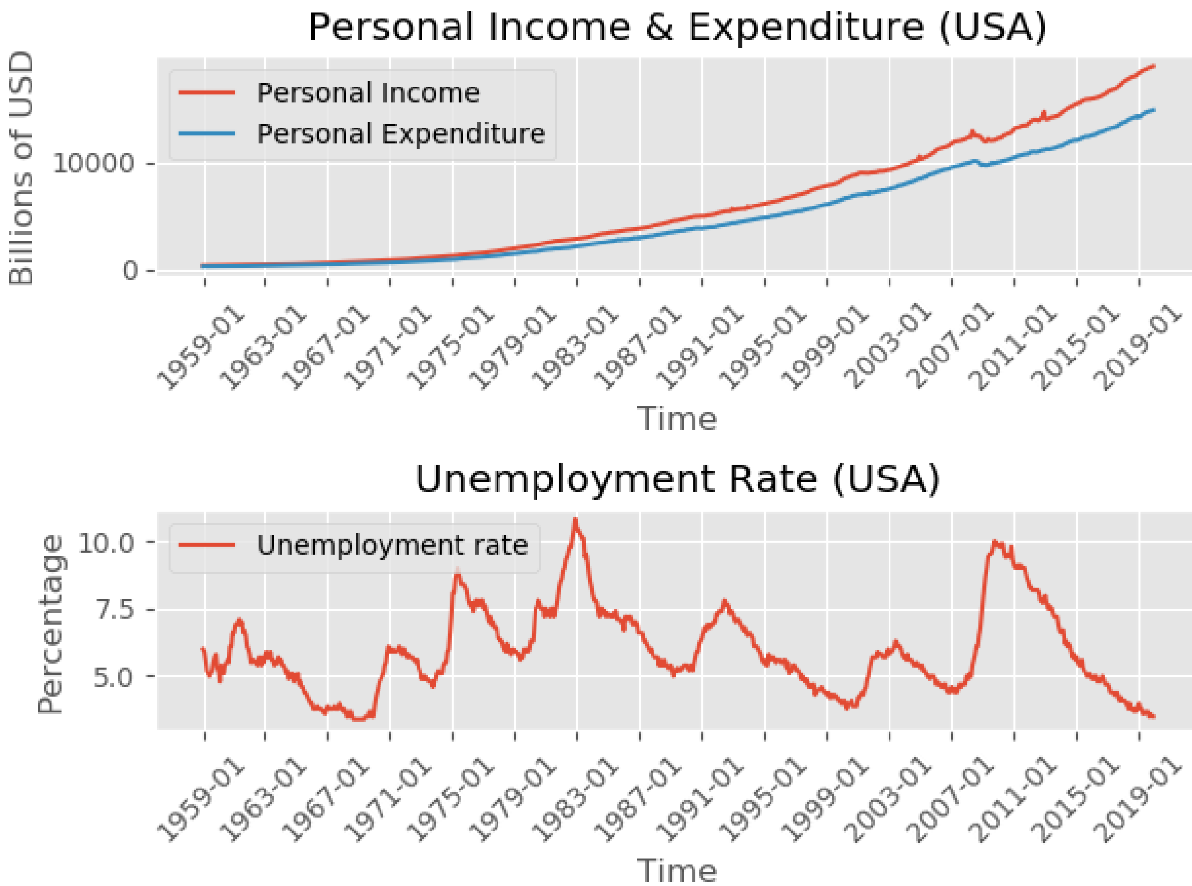




<u>Merged Data</u>:

The discrepancy in starting and ending dates for each variable might create problems for us in long run hence we must bring them all on the same page.

So we merged and combine the three separate files into one, in such a way so that they have a common start and end point, and following graphs are resultantly obtained which are very similar to previous ones except that now starting and ending dates match for PCE, PI and UNRATE.



Preprocessing:

Before applying any models on data set, first step is to explore how many and what kind of observations do we have in data set.

Total number of observations is 732 and total columns (excluding time index) are 3.

```
Number of rows and columns in dataframe are: (732, 3)
```

The first 5 rows look like this.

	PI	PCE	UNRATE
Date_Month			
1959-01	391.8	306.1	6.0
1959-02	393.7	309.6	5.9
1959-03	396.5	312.7	5.6
1959-04	399.9	312.2	5.2
1959-05	402.4	316.1	5.1

The last 5 rows look like this.

	PI	PCE	UNRATE
Date_Month			
2019-08	18731.7	14682.4	3.7
2019-09	18786.4	14707.8	3.5
2019-10	18797.5	14740.7	3.6
2019-11	18881.6	14806.0	3.5
2019-12	18922.3	14852.6	3.5

Then we ensure that

1. Data set is clean (there are no missing values or NaN in our data set)

```
Rows containing missing (NaN) values are: Empty DataFrame
Columns: [PI, PCE, UNRATE]
Index: []
```

- The same can be confirmed using df.info() where the 2nd column gives us "Non-Null Count" which is exactly equal to number of rows in data set.

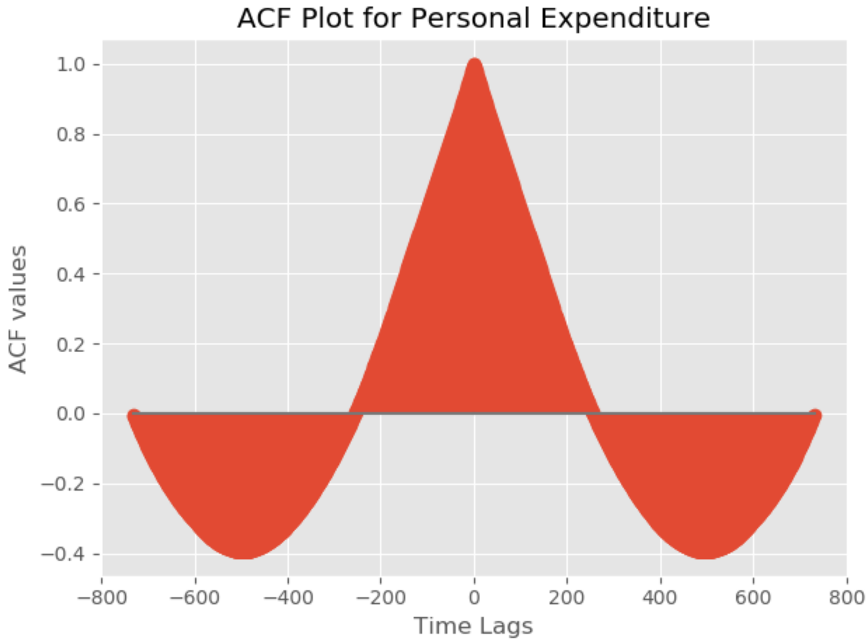
```
Index: 732 entries, 1959-01 to 2019-12
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   PI      732 non-null    float64
 1   PCE     732 non-null    float64
 2   UNRATE  732 non-null    float64
dtypes: float64(3)
memory usage: 22.9+ KB
```

2. Time steps are equal

```
The first date in our data set is: 1959-01
The last date in our data set is: 2019-12
So we have monthly data for  61 years.
Consequently, 61*12 =  732 months which is exactly equal to
the number of rows we have in our dataframe.
```

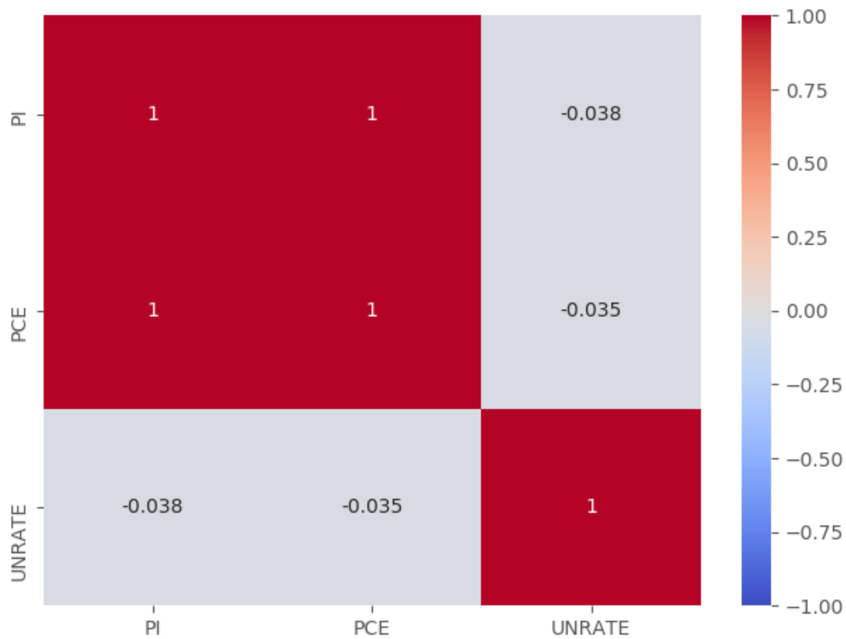
<u>ACF of dependent variable:</u>
Autocorrelation measures the linear relationship between lagged values of a time series. It represents the dependence of a data point with a previous data point (or, a future data point). If autocorrelation is "strong", then it means that we can infer more accurately the value of a variable for some future time. If the autocorrelation is "weak", our predictions would not be that accurate.

ACF plot of dependent variable (PCE) is clearly decaying with increasing number of lags meaning that our predictions about the near future would be more accurate than predictions about the distant future.



Correlation Matrix:

With help of *seaborn's* heatmap and *pearson's correlation coefficient* the following matrix is obtained indicating strong linear relationship between PCE and PI whereas (almost) zero relationship between PCE and UNRATE.



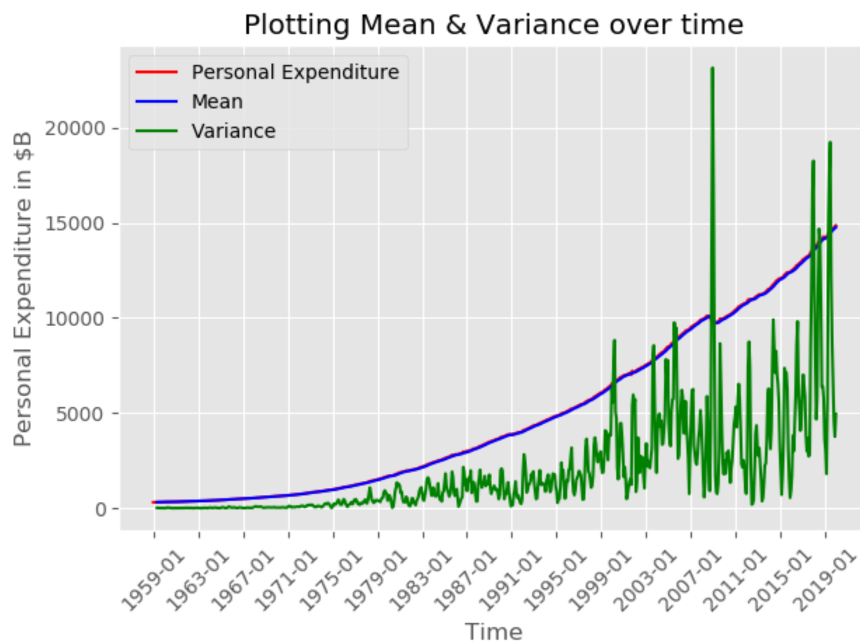
Stationarity:

For forecasting and prediction-making we apply different kinds of models on our data set but the prerequisite for most of these models is Stationarity of data. A given time-series is stationary if basic data-statistics (mean and variance) don't change with time.

There are two methods to identify stationarity in data set,

1. Visual Inspection:
   a. We will plot our time-series data and then by looking at graphs we will determine if the mean & variance are constant (stationary) or varying (non-stationary) with time. But it is important to remember that visual inspection method are prone to subjective results.

- From graph below, it is self-evident that mean and variance of data set are changing considerably with the change in time. Hence it is very likely that data is non-stationary.



2. Augmented Dickey-Fuller (ADF) Test using Python's *statsmodels*:
   a. It is an objective/statistical test that outputs a p-value. If
   H0: P-value > 0.05 then our data is non-stationary
   H1: P-value < 0.05 then our data is stationary

- From results of ADF test (given below), p-value of 1 i.e. greater than 0.05 is obtained (hence failing to reject Null Hypothesis) confirming that PCE is non-stationary.

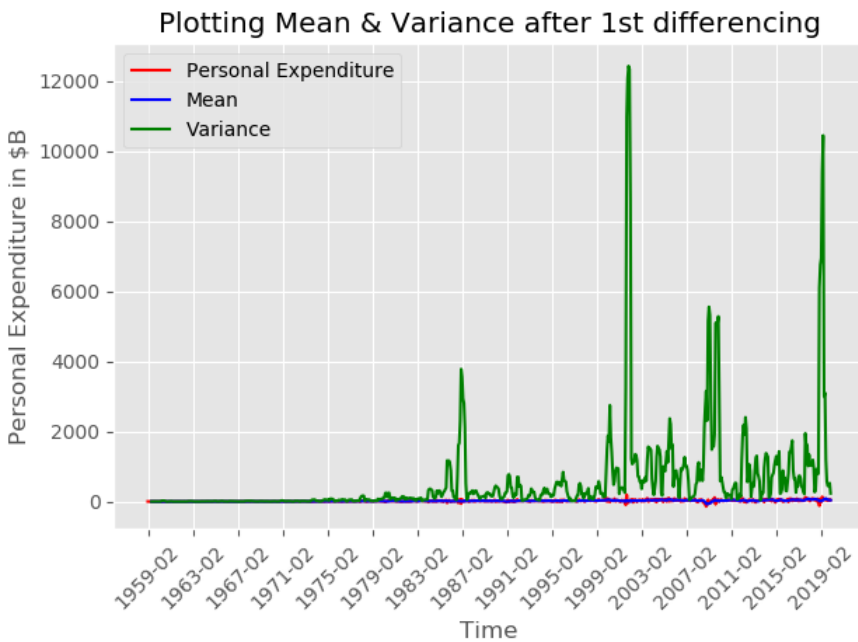
```
ADF Statistic: 4.765965
p-value: 1.000000
Critical Values:
    1%: -3.439
    5%: -2.866
    10%: -2.569
```

Now that we are sure about Non-Stationarity of PCE, detrending is required to convert it into stationary data.

<u>Detrending the dependent variable:</u>
To tackle the matter of non-stationarity, we first apply first order difference technique to convert upward moving trend (varying mean values) into a constant.

Here are the visual results. Mean looks constant but there is still some variation left in variance.

However, surprisingly the p-value of 0.0222 (i.e. less than 0.05) obtained from ADF test of detrended data suggests stationarity, meaning no further action (transformation) required. Therefore for time series models that require stationarity, we will use the detrended (differenced) data set. But it is important to mention here that once a model has been applied on detrended data and predictions obtained, they must be reverse transformed to obtain real predicted values. During the process of first order differencing, the first observation i.e. January 1959 is lost and hence the graph below starts from February 1959.




```
ADF Statistic: -3.162749
p-value: 0.022245
Critical Values:
    1%: -3.439
    5%: -2.866
    10%: -2.569
```

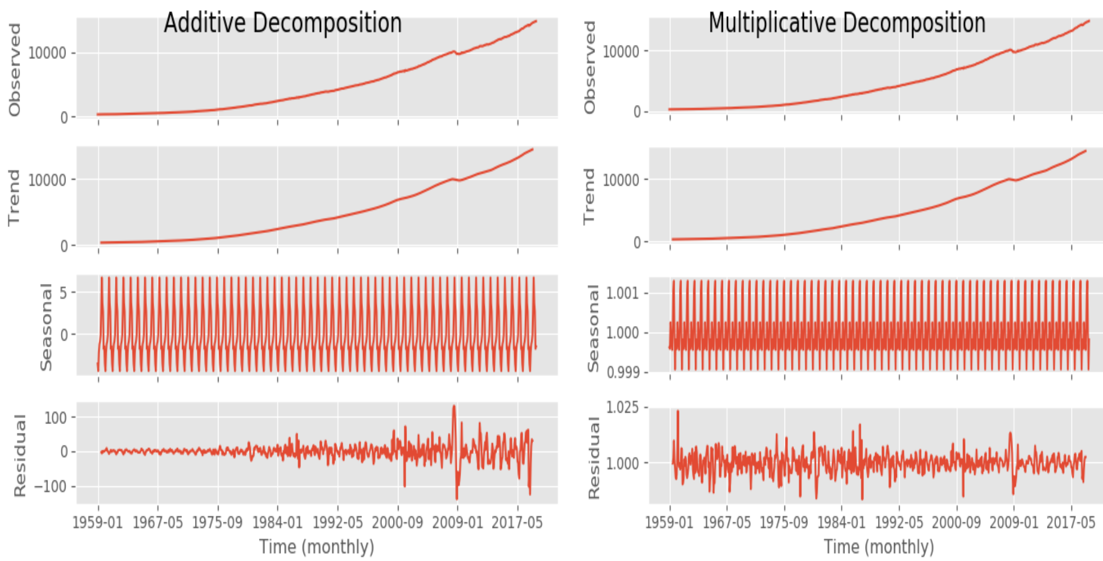
<u>Time Series Decomposition:</u>

A time series can be decomposed into 3 different components namely trend, seasonality and residual.

- Trend: is the increasing or decreasing value in the series.
- Seasonality: is the repeating short-term cycle in the series.
- Noise: is the random variation in the series.

These components can be combined in either of 2 ways:
Additively: y(t) = trend + seasonality + residual
Multiplicatively: y(t) = trend * seasonality * residual

Usually, if variance in original graph is constant throughout, it means time series is additive else multiplicative. For our data set we know that variance is not exactly constant so multiplicative model should be chosen. Another reason to pick multiplicative model over additive is that range of residuals is smaller for multiplicative than that of additive.



Holt-Winters method & other simple forecasting methods:

In this section we are going to apply four simple (Average, Naïve, Drift, SES) and 2 advanced forecasting methods (Holt-Linear, Holt-Winter) and then evaluate their performance on basis of residual diagnostics. In next section, first a brief description of each method is given for easier understanding followed by their prediction graphs. Average, Naïve and SES give flat forecasts whereas Drift, Holt-Linear and Holt-Winter methods give non-flat forecasts.

Four Simple Forecasting Methods:

Average Method:

It is useful for forecasting short-term trends. This method assumes that all values are of equal importance and gives them equal weights.
Forecasted value = Average of all historical values in data set

$$\hat{y}_{T+h|T} = \frac{y_1 + y_2 + \ldots + y_T}{T}$$

Naïve Method:

It is also known as Random Walk forecasting because it is optimal when data follow random walk. Forecasted value = Value of last observation in data set

$$\hat{y}_{T+h|T} = y_T.$$

Drift Method:

Unlike average and naïve forecast that give us flat forecasts for the future, Drift method can increase or decrease with time because it captures the linear trend in data.

Basically, we take the first and last point from our data set (ignoring all middle values), draw a line between them and extend (extrapolate) it in the future to make prediction.

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h\left(\frac{y_T - y_1}{T-1}\right).$$

Simple Exponential Smoothing(SES):

Here, forecasts are produced using weighted averages of historical observations with weights getting smaller exponentially as observations get older. It is most suitable for data with no trend or seasonal pattern. The rate at which weights decrease is controlled by $0<\alpha<1$. If alpha is close to 0, more weight is given to observations from distant past. If alpha is close to 1, more weight is given to most recent observations (for this project alpha=0.8 was used). The other parameter involved in SES is l_0. The optimum alpha and l_0 can be found by minimizing Sum Square Error (SSE). Substituting alpha and l_0, we are ready to predict as given below:

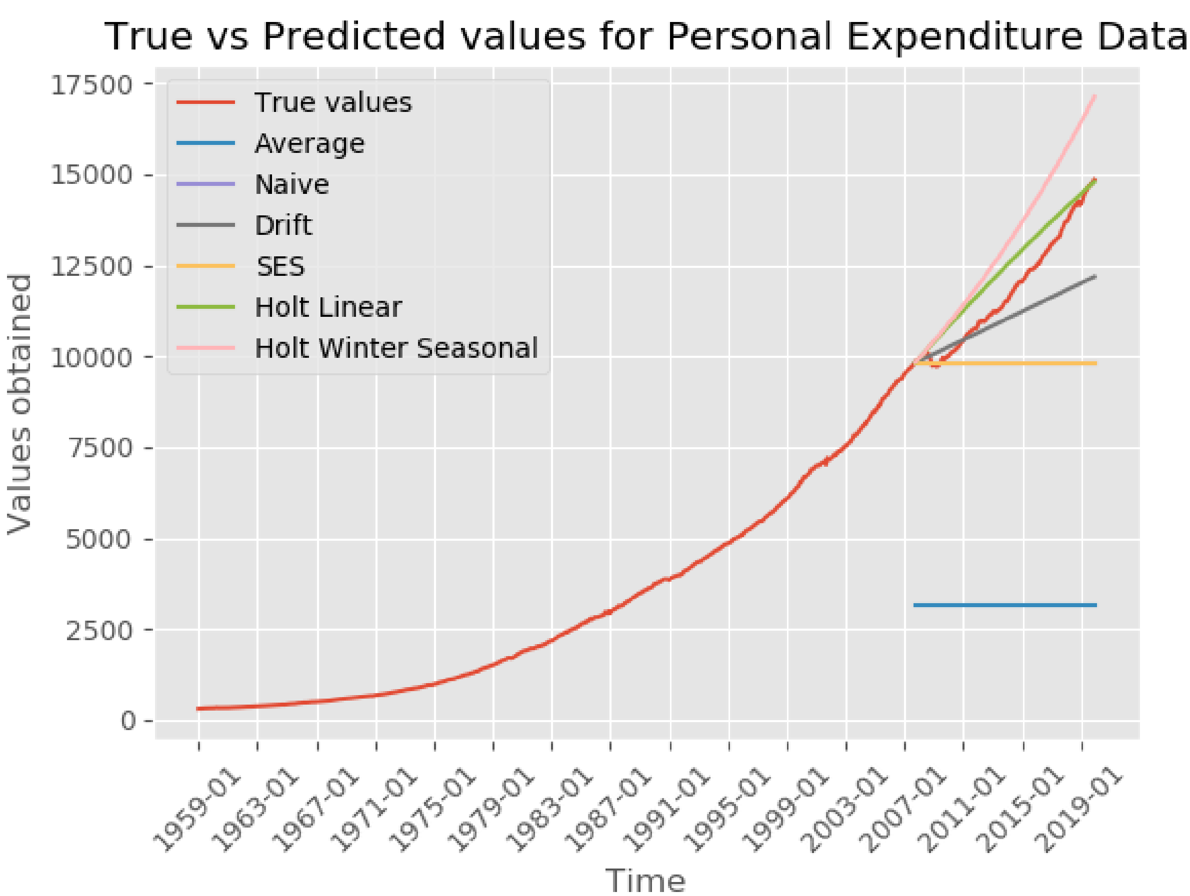
$$\begin{aligned}
\hat{y}_{2|1} &= \alpha y_1 + (1-\alpha)\ell_0 \\
\hat{y}_{3|2} &= \alpha y_2 + (1-\alpha)\hat{y}_{2|1} \\
\hat{y}_{4|3} &= \alpha y_3 + (1-\alpha)\hat{y}_{3|2} \\
&\vdots \\
\hat{y}_{T|T-1} &= \alpha y_{T-1} + (1-\alpha)\hat{y}_{T-1|T-2} \\
\hat{y}_{T+1|T} &= \alpha y_T + (1-\alpha)\hat{y}_{T|T-1}.
\end{aligned}$$

Two Advanced Forecasting Methods:

Holt's Linear Trend method:
It is an extension of SES method that allows the forecasting of data with trend.

Holt-Winter Seasonal method:
It is an extension of Holt's Linear Trend method that allows forecasting of data that has trend as well as seasonality.

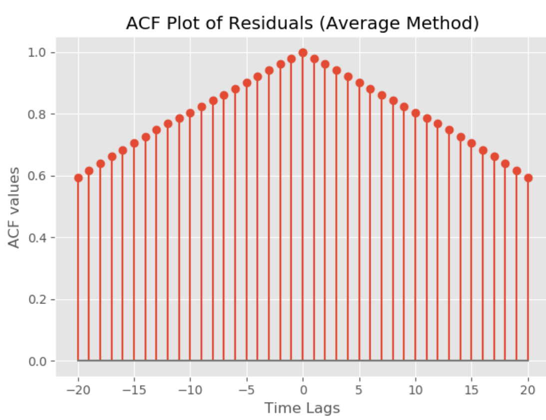
From graph plotted below we observe that the values forecasted by Naïve and SES were almost the same. That is why they are overlapping in the graph. Similarly since our dataset lacks seasonality component, hence Holt-Winter Seasonal forecasts are not as close to original values (test set) as Holt-Linear forecasts which captures trend component fully.

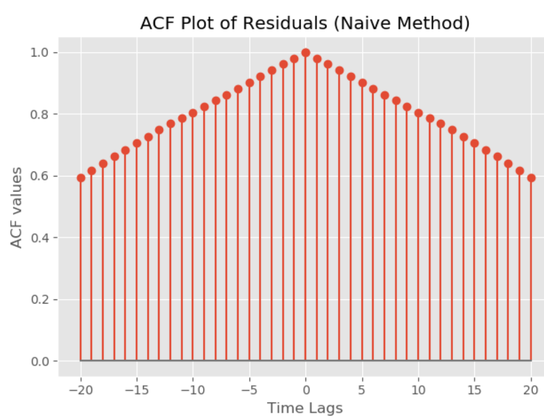


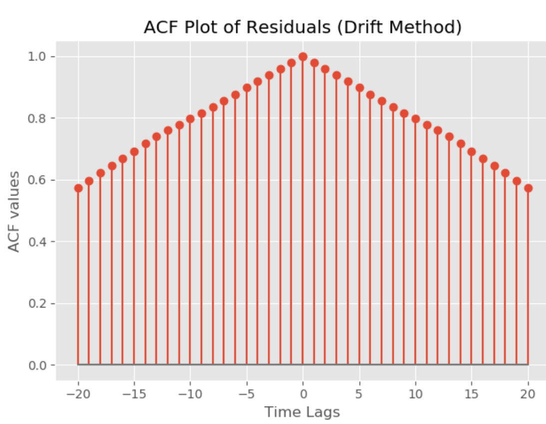
```
Holt Linear method summary
                        ExponentialSmoothing Model Results
================================================================================
Dep. Variable:                     endog   No. Observations:                  585
Model:              ExponentialSmoothing   SSE                         171772.865
Optimized:                          True   AIC                           3334.155
Trend:                    Multiplicative   BIC                           3356.013
Seasonal:                           None   AICC                          3334.349
Seasonal Periods:                   None   Date:                 Thu, 23 Apr 2020
Box-Cox:                           False   Time:                         17:14:09
Box-Cox Coeff.:                     None
==============================================================================
```

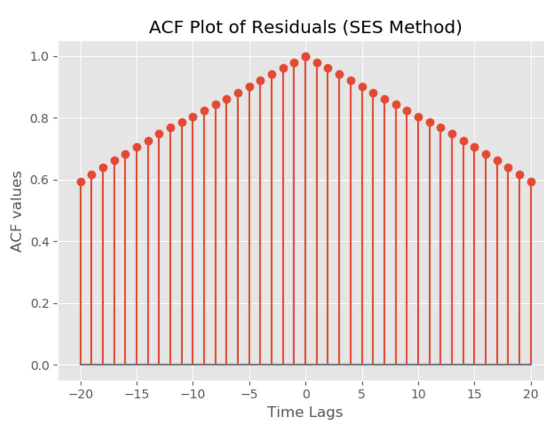
<u>ACF Plots of Residuals:</u>

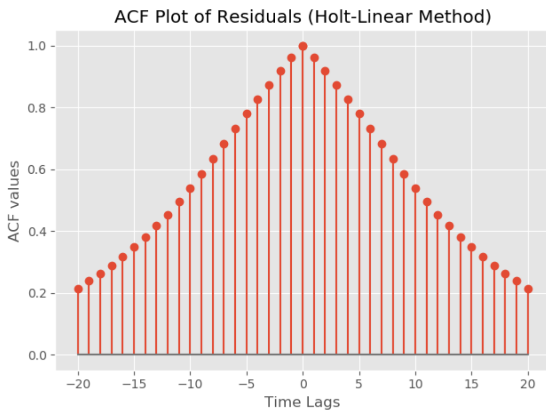
It is easy to observe that of all the methods, ACF (residuals) graph of Holt-Linear is the one that has the steepest descent (hence captured the most information). The same result is confirmed by RMSE values as well (given below).

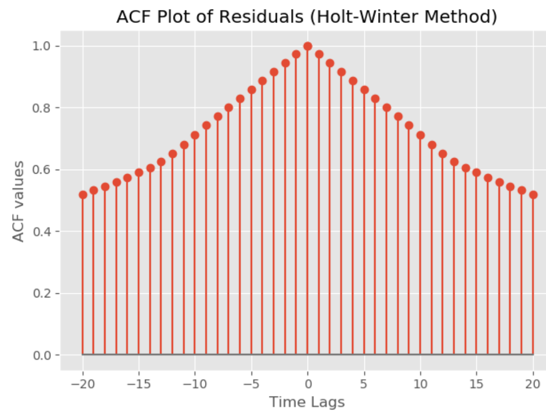













```
Root Mean Square Error of Average Method is:  8775.239139258058
Root Mean Square Error of Naive Method is:  2500.95956598268
Root Mean Square Error of Drift Method is:  1162.2532935142763
Root Mean Square Error of SES Method is:  2509.598250501314
Root Mean Square Error of Holt-Linear Method is:  703.4889569382652
Root Mean Square Error of Holt-Winter Method is:  1504.4441310081697
```

Mean & Variance of residuals (Holt-Linear Method):
The mean of residuals is not zero hence it is a biased estimator.

```
Mean of residuals for Holt-Linear method is :  -628.349431230998
Variance of residuals for Holt-Linear method is :  100073.70480576938
```

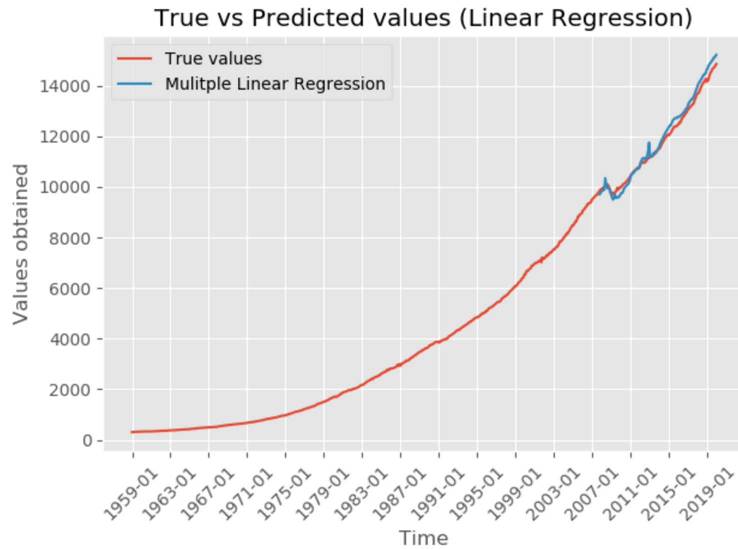
Feature Selection & Linear Regression (LR) Model:

A basic assumption of LR model is that dependent variable has a linear relationship with independent variables. We already established this between PCE and PI using Pearson's coefficient in Correlation Matrix. Equation of multiple linear regression model with 2 independent variables is:

$$y = \beta 0 + \beta 1 x 1 + \beta 2 x 2 + error$$

We already know y (PCE) and x1, x2 (PI, UNRATE) values from our dataset, but the coefficients β0 (y-intercept), β1 (slope) and β1 need to be estimated. Error represents the deviation of predicted value from the true value.

Before applying Linear Regression model, data is split into train (80%) and test (20%) sets. Then using statsmodel's OLS package, true and predicted values obtained from Linear regression model are obtained. Predicted values are very close to actual values indicating this is a good model (relative to simple forecasting methods that we saw in previous section).

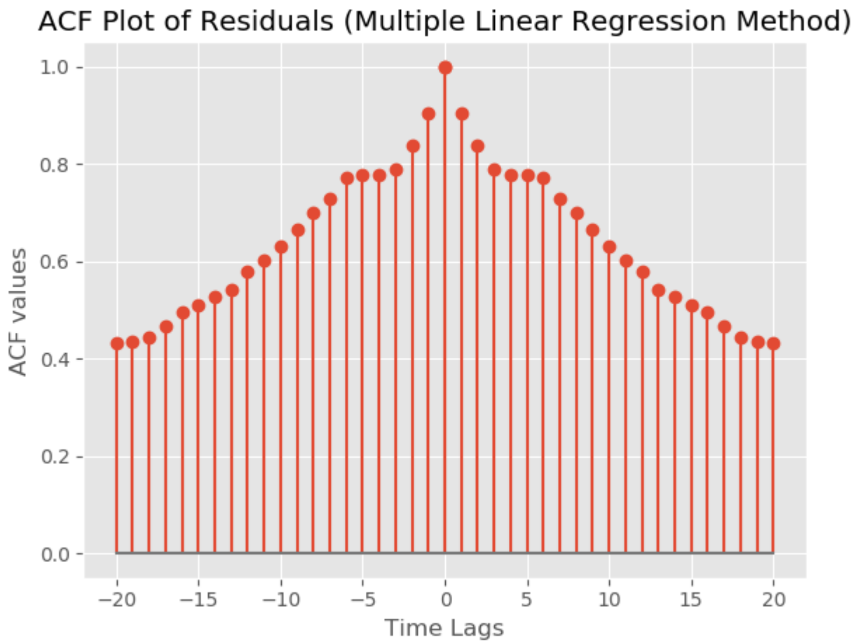



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    PCE   R-squared:                       0.999
Model:                            OLS   Adj. R-squared:                  0.999
Method:                 Least Squares   F-statistic:                 2.828e+05
Date:                Sat, 18 Apr 2020   Prob (F-statistic):               0.00
Time:                        14:16:47   Log-Likelihood:                -3446.1
No. Observations:                 585   AIC:                             6898.
Df Residuals:                     582   BIC:                             6911.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      2.7411     16.632      0.165      0.869     -29.925      35.407
PI             0.8064      0.001    742.690      0.000       0.804       0.809
UNRATE       -13.8607      2.568     -5.397      0.000     -18.905      -8.816
==============================================================================
Omnibus:                       42.761   Durbin-Watson:                   0.085
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               83.412
Skew:                           0.455   Prob(JB):                     7.72e-19
Kurtosis:                       4.610   Cond. No.                     2.42e+04
==============================================================================
```

- R-square (proportion of variation explained by the regression model) and Adjusted R-square values are almost 1
- AIC is 6898 and BIC is 6911 (the lower the better)

Mean, Variance, Q-value, RMSE and ACF plot of residuals are given below. Since mean is not 0 hence this is biased estimator:

```
Mean, Variance, Q-value & RMSE of prediction error (OLS model) are -110.31, 44255.00, 1203.63 &
 237.54 respectively.
```



Feature Selection:

For feature selection we need to look at p-values of F-test and T-test but first let's state Null and Alternate (opposite of Null) hypothesis to make things more clear.

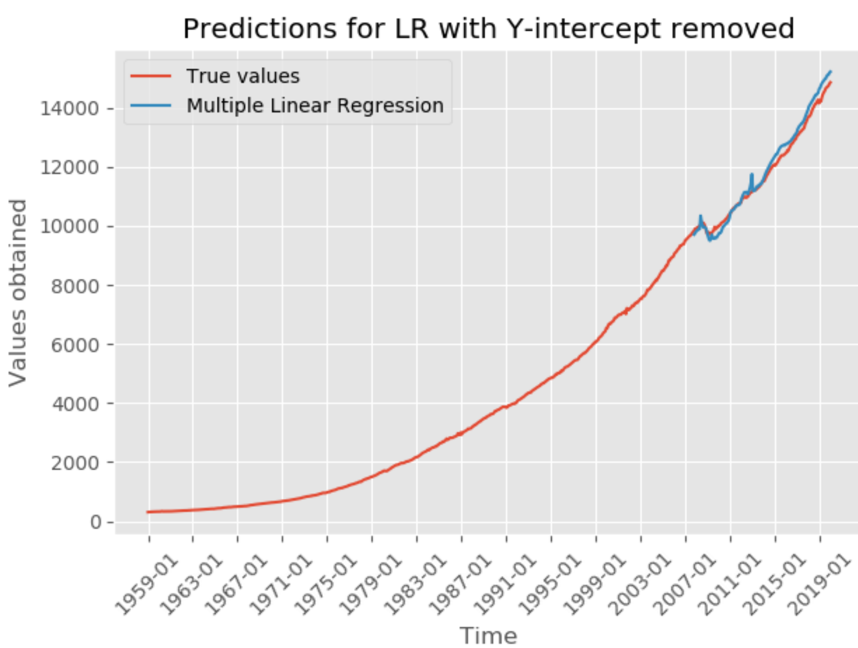
<u>F-test:</u>
H0: Model performed as good as the y-intercept only model (Independent features are irrelevant)
H1: Model performed better than the y-intercept only model (Independent features are relevant)

<u>T-test:</u>
H0: There is no relationship between feature x1 and y (β1 equals zero)
H1: There is a relationship between feature x1 and y (β1 not equal to zero)

Considering our threshold value to be 0.05 for both F-test and T-Test and since Null hypothesis is rejected in all the cases except for y-intercept, we conclude the following:

- Our model performed better than y-intercept only model (p-value for F-statistic = 0)
- There is a significant relationship between features (PI and UNRATE) and target variable (PCE) with p-values for T-test equal to 0 and 0 respectively and confidence intervals not containing 0.
- **Y-intercept should be removed from the model because p-value for T-test is greater than 0.05 (i.e. 0.869) and also because the confidence interval contains 0.**

So we are going to run LR model one more time. But this time, removing y-intercept from the model and here are the results.

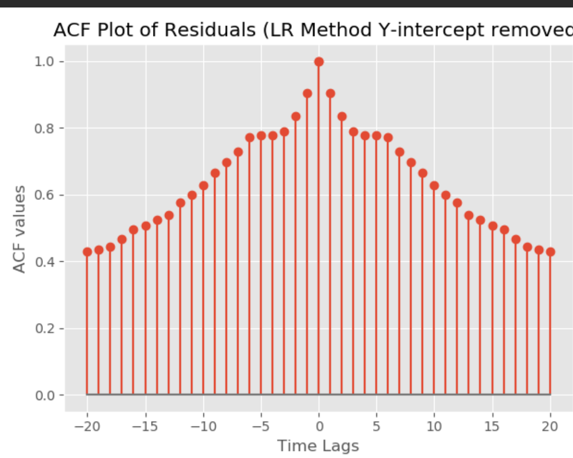


```
                           OLS Regression Results
=======================================================================================
Dep. Variable:                    PCE   R-squared (uncentered):                   1.000
Model:                            OLS   Adj. R-squared (uncentered):              1.000
Method:                 Least Squares   F-statistic:                          6.560e+05
Date:                Wed, 22 Apr 2020   Prob (F-statistic):                        0.00
Time:                        10:39:07   Log-Likelihood:                         -3446.2
No. Observations:                 585   AIC:                                      6896.
Df Residuals:                     583   BIC:                                      6905.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
PI             0.8065      0.001    809.397      0.000       0.805       0.808
UNRATE       -13.4622      0.865    -15.572      0.000     -15.160     -11.764
==============================================================================
Omnibus:                       42.344   Durbin-Watson:                   0.085
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               81.901
Skew:                           0.454   Prob(JB):                     1.64e-18
Kurtosis:                       4.592   Cond. No.                     1.24e+03
==============================================================================
```

- R-square (proportion of variation explained by the regression model) and Adjusted R-square values are again 1
- AIC is 6896 and BIC is 6905. (Less than the results we got from model that included y-intercept)

Mean, Variance, Q-value, RMSE and ACF plot of new residuals are exactly the same as before. Since mean is not 0 hence this is biased estimator:

```
Mean, Variance, Q-value & RMSE of prediction error (OLS model with y-intercept removed) are -110.31, 44255.00, 1203.63 &
 237.54 respectively.
```



ARMA model:

Stationarity is a mandatory requirement/assumption for ARMA model so instead of original PCE values , the first order differenced (stationary) data set will be used for prediction this time. Once predictions have been obtained, they will be reversed transformed to get the original values. Also the residuals of ARMA model will be passed through Chi-Square test to confirm if they are white or not.

$\chi 2$Test for whiteness of residuals:

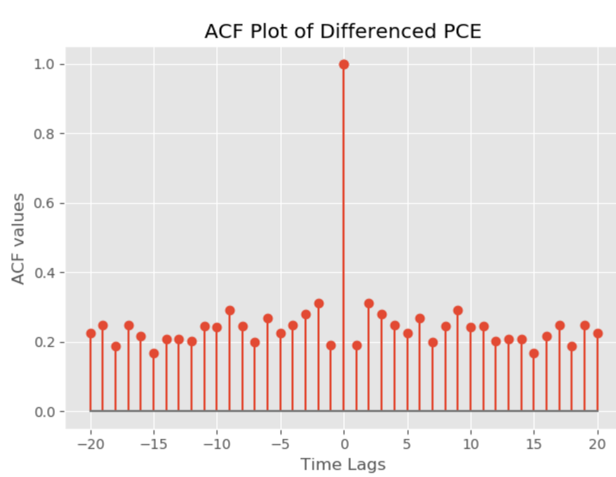
Let $Re(\tau)$ be the ACF of residuals. Then, Q can be calculated as:

$$Q = N * \sum_{\tau=1}^{h} Re(\tau)^2$$

Where N is the number of observations and h is the number of maximum lags. The above Q needs to be compared versus the critical Qc which can be found from the $\chi 2$ table.

- H0: The residuals are uncorrelated, hence it is white. This can be shown when Q<Qc or when the p-value is higher than the threshold.
- HA: The residuals are correlated, hence it is NOT white. This can be shown when Q>Qc or when the p-value is lower than threshold.

ACF of Differenced Data and GPAC table:



ACF values with lag 20 were used to compute GPAC table. A pattern of constant column and a row of zeros can be seen at k=2 and j=0. But it failed Chi-square test and so does k=1 and j=2.

	1	2	3	4	5	6	7	8
0	0.19	0.285	0.207	0.127	0.076	0.126	0.042	0.082
1	1.64	0.159	0.039	0.004	-0.131	0.101	-0.204	0.006
2	0.898	-0.021	0.027	1.318	-0.123	0.049	-0.185	3.113
3	0.888	1.303	0.347	0.451	-0.092	-0.196	-0.112	0.047
4	0.905	-13.84	20.465	0.497	-0.339	0.067	-0.144	0.382
5	1.198	1.702	0.95	-0.936	-0.603	-0.914	-0.196	-0.029
6	0.737	0.951	2.55	-2.699	-1.711	0.987	-0.246	-0.949
7	1.233	0.078	0.855	0.342	1.152	0.1	0.714	0.173

So I tried all possible combinations in 8x8 matrix and out of those only (0,1), (1,0) and (0,2) passed the Chi-Square test for whiteness. All the remaining orders either produced error or failed Chi-Square test. Estimated parameters, their standard deviation and confidence intervals are given below for these 3 orders.

ARMA (0,1):

```
********* Results for ARMA(0,1) start here *************
The MA coefficient b0 is: 0.22457888125045183

Confidence Interval (95%) for estimated paramters are:
                0        1
ma.L1.PCE  0.16161  0.287548

The estimated covariance matrix for for model is:
           ma.L1.PCE
ma.L1.PCE   0.001032

The estimated variance & standard deviation of prediction error :
 677.2772437190439 26.02455078803559

Mean of ARMA residuals -34.24655512896841
Variance of ARMA residuals 1677.1895991926606
Q (32.709644326761364) < Q_c (36.19086912927004) hence residuals pass the whiteness test confirming that model suggested by
  LM is a good model.
```

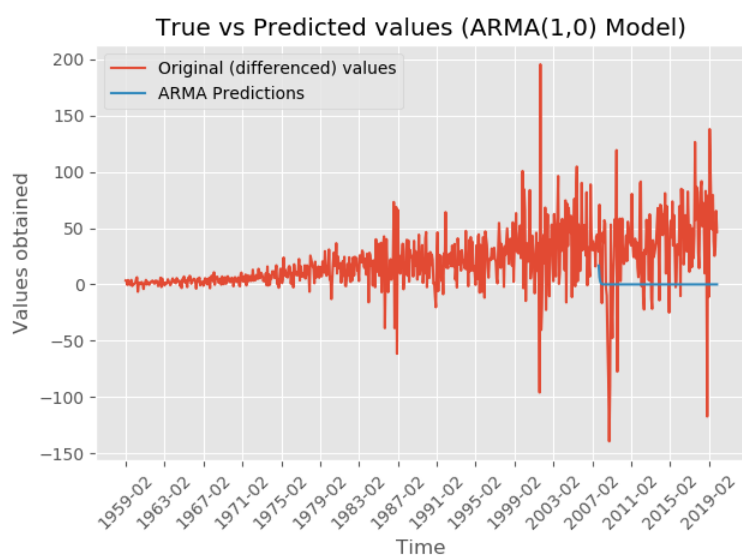
ARMA (1,0):

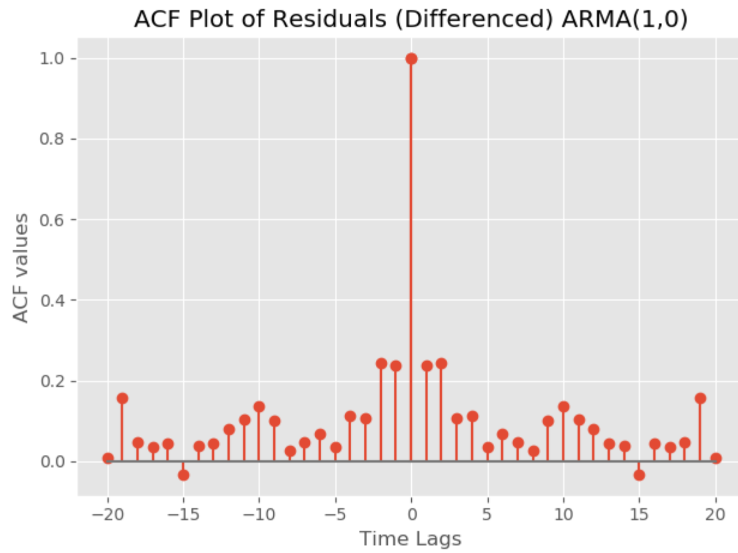
```
The estimated covariance matrix for for model is:
          ar.L1.PCE
ar.L1.PCE    0.00147

The estimated variance & standard deviation of prediction error :
 630.8746899122251 25.11721899240091

Mean of ARMA residuals -34.11808155375666
Variance of ARMA residuals 1677.5326864173746
RMSE of ARMA residuals 53.30643652811699
Q (33.93600844900642) < Q_c (36.19086912927004) hence residuals pass the whiteness test confirming that suggested model is
 good.
The AR coefficient a0 is: 0.3790581123579692

                              ARMA Model Results
==============================================================================
Dep. Variable:                    PCE   No. Observations:                  584
Model:                     ARMA(1, 0)   Log Likelihood               -2711.293
Method:                       css-mle   S.D. of innovations             25.117
Date:                Fri, 24 Apr 2020   AIC                           5426.586
Time:                        18:07:20   BIC                           5435.326
Sample:                    02-01-1959   HQIC                          5429.992
                         - 09-01-2007
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.PCE      0.3791      0.038      9.887      0.000       0.304       0.454
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            2.6381           +0.0000j            2.6381            0.0000
-----------------------------------------------------------------------------
Confidence Interval (95%) for estimated paramters are:
                  0         1
ar.L1.PCE  0.303916  0.454201
```

ARMA (0,2):

```
*********  Results for ARMA(0,2) start here *************
The MA coefficient b0 is: 0.097040276495199
The MA coefficient b1 is: 0.3396807997649558

Confidence Interval (95%) for estimated paramters are:
                   0         1
ma.L1.PCE  0.009656  0.184425
ma.L2.PCE  0.266744  0.412617

The estimated covariance matrix for for model is:
            ma.L1.PCE  ma.L2.PCE
ma.L1.PCE   0.001988  -0.000598
ma.L2.PCE  -0.000598   0.001385

The estimated variance & standard deviation of prediction error :
 597.9207998045093 24.45241909923248

Mean of ARMA residuals -34.12383647800151
Variance of ARMA residuals 1674.8806160050804
Q (33.9971940231835) < Q_c (34.805305734705065) hence residuals pass the whiteness test confirming that model suggested by
  LM is a good model.
```

Since out of all the models passing Chi-square test, the simplest (one with the least parameters) should be chosen so I will pick ARMA(1,0) because it has least number of parameters and estimated variance and standard deviation of prediction error is less than ARMA(0,1). Mean of residuals is not 0 so it is biased estimator. ACF plot of residuals and forecasted values (with differenced and original data respectively) are plotted below:



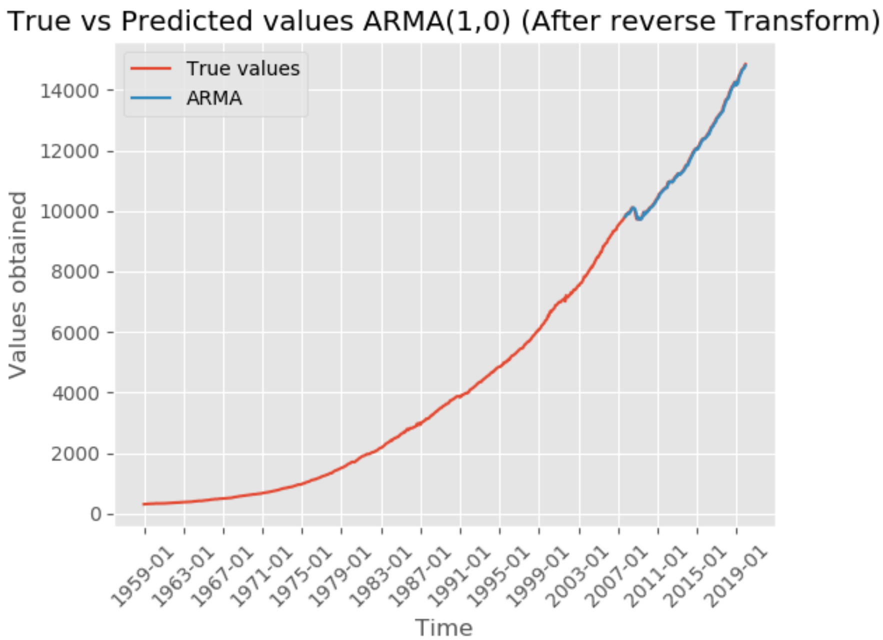


In ACF plot of ARMA(1,0) residuals, we see two spikes at lag 1 and 2 that have values greater than 0.2 so let's create a 15x15 GPAC table from these residuals.

Since we already know the three ARMA orders that pass Chi-Square test within 8x8 range so we would start looking for orders/patterns beyond that range in GPAC table below.

From patterns visible below and adding them to original ARMA(1,0), I got ARMA (10,0), (11,0) and (16,0). The AIC and BIC values did decrease slightly but the residuals obtained, failed Chi-Square test so these orders turned out to be inadmissible.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.23800	0.19900	0.01300	0.04800	-0.01900	0.03600	0.02300	-0.00700	0.09400	0.09400	0.02600	-0.00400	-0.01800	0.00700	-0.06900
1	1.02700	0.18400	-0.71600	0.05300	0.07300	0.04800	0.03400	0.30100	0.10100	0.06900	0.03900	-0.13200	-0.01900	-0.17300	-0.06500
2	0.43300	0.48300	0.04500	-0.05400	-0.09700	0.06600	0.23500	-0.19900	0.11700	0.05300	-0.07400	0.06900	-0.11400	0.01500	-0.08400
3	1.07900	0.54800	0.66200	-0.21700	-0.18200	-0.06000	-0.09000	0.03000	0.10300	0.09100	0.06700	-0.11800	-0.15900	-0.67700	-0.08800
4	0.32100	0.72200	-1.82600	1.47300	-0.24400	0.45800	-0.11700	0.38900	0.10000	[illegible]	0.37100	-0.36300	-0.35900	0.28500	[illegible]
5	1.90100	0.46400	0.36400	-0.88400	-1.19300	-0.07800	-0.58400	0.02200	0.08400	0.30200	-0.28000	0.40200	0.00200	0.28500	-191.54100
6	0.70100	-0.07000	1.71600	-1.67000	-1.41900	10.35200	-0.57200	2.09800	0.07000	0.22000	0.47900	0.40400	-74.18500	0.28400	-0.29000
7	0.63800	19.78800	1.96800	-0.37500	0.88400	-0.10400	0.19700	0.30200	[illegible]	0.17800	-0.02300	-0.09500	-0.01100	0.16800	-0.28000
8	3.41900	1.69900	1.41500	3.22900	0.81200	1.58000	0.41200	0.84200	-0.14800	0.15400	-0.78100	-0.09100	-1.43500	0.16000	-0.31400
9	1.25100	-0.90400	0.32700	-0.19700	0.30200	-1.83600	3.93500	0.71300	0.59700	0.05300	-0.19600	0.34900	-0.13500	0.03100	0.04900
10	0.79300	-0.21100	-0.09200	0.33500	-0.44900	-0.43700	1.09200	0.20600	0.27700	1.49400	-0.12500	-0.01700	-0.06100	0.20000	-0.02400
11	0.69600	-0.47600	-1.08000	0.00200	-1.26000	-1.29400	1.15200	-0.83600	-0.32100	0.86600	-0.19800	0.45700	-0.09100	0.23900	-0.08800
12	0.61200	1.49900	-1.07000	-665.36800	-1.25600	0.90900	1.20300	-3.07100	-5.19000	0.83500	-1.35700	-0.65200	-0.68700	0.24600	6.99800
13	0.90800	3.31000	-9.36900	-3.88400	-2.69400	3.74100	1.10000	2.36700	2.30500	0.53400	0.98900	2.81700	0.83600	-0.29900	0.35600
14	-0.85900	0.20400	1.27300	2.57700	3.47700	1.27200	0.41100	0.47000	1.51300	-1.00700	-1.63800	1.25500	3.06800	1.90900	0.76600



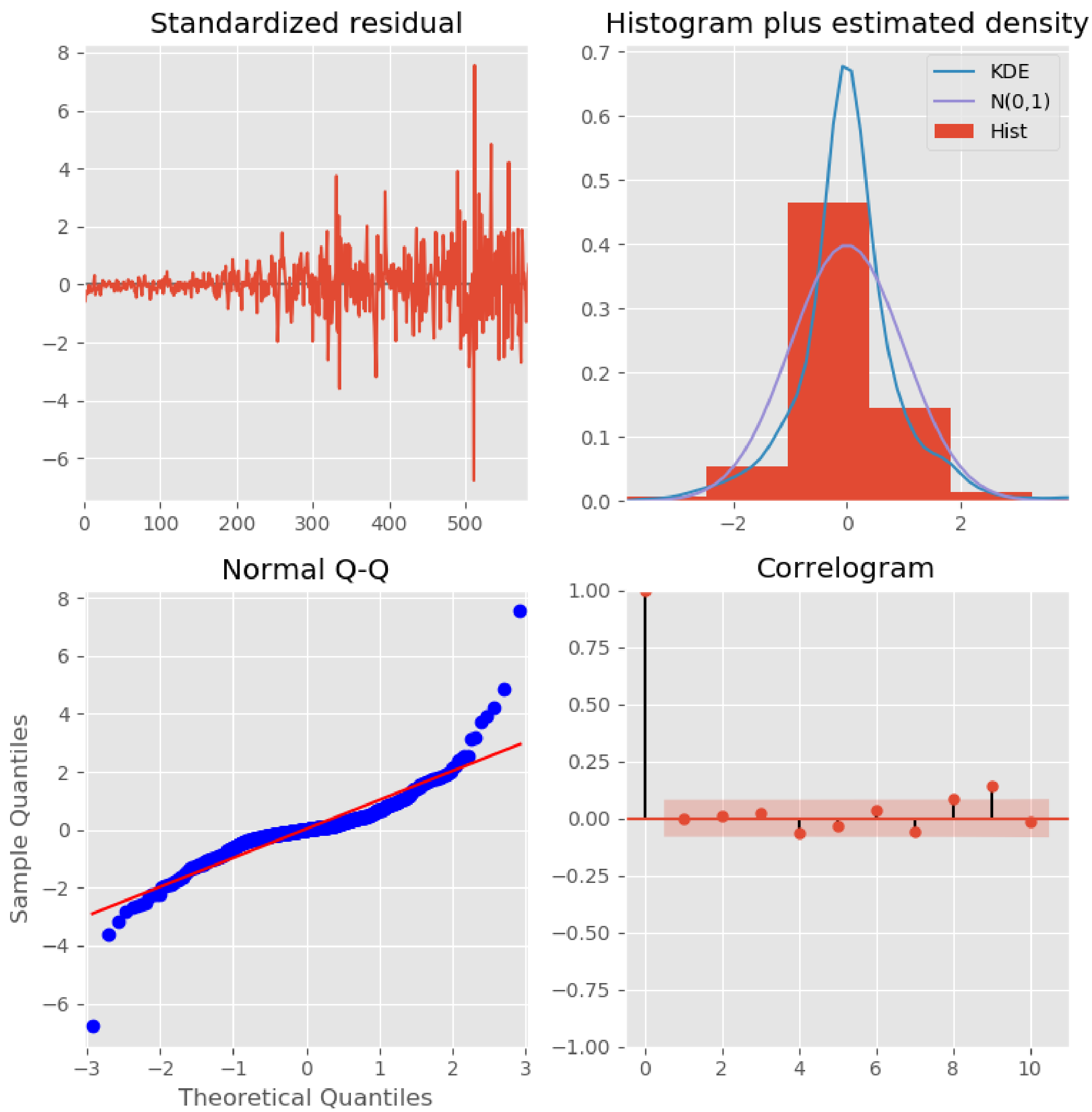
<u>Auto AR Integrated MA (ARIMA) Model:</u>

It is an extension of ARMA model. Auto ARIMA seeks to identify the most optimal parameters and then chooses the one with least AIC and BIC values. When trained over raw PCE data, following results are obtained:

Best ARIMA model is (2,1,2). AIC and BIC values are the second best (after Holt-Linear) we have seen so far. P-values of T-test and confidence intervals for all estimated parameters indicate that they are significant. However p-values of T-test and confidence intervals for 'y-intercept' indicate it is un-important (tried model again with y-intercept removed but it did not result in significant difference).

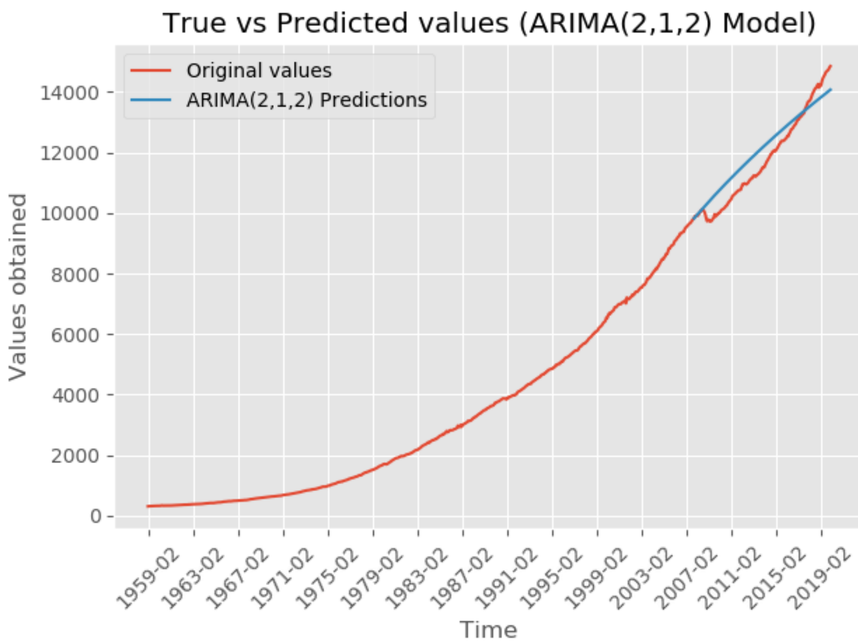
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  585
Model:               SARIMAX(2, 1, 2)   Log Likelihood               -2487.323
Date:                Thu, 23 Apr 2020   AIC                           4986.646
Time:                        22:49:42   BIC                           5012.865
Sample:                             0   HQIC                          4996.865
                                - 585
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0958      0.144      0.665      0.506      -0.186       0.378
ar.L1          1.1443      0.055     20.697      0.000       1.036       1.253
ar.L2         -0.1499      0.055     -2.705      0.007      -0.258      -0.041
ma.L1         -1.5374      0.052    -29.828      0.000      -1.638      -1.436
ma.L2          0.6140      0.048     12.899      0.000       0.521       0.707
sigma2       291.8439      8.058     36.218      0.000     276.051     307.637
===================================================================================
Ljung-Box (Q):                       84.35   Jarque-Bera (JB):              3009.44
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):              44.14   Skew:                             0.54
Prob(H) (two-sided):                  0.00   Kurtosis:                        14.07
===================================================================================
```

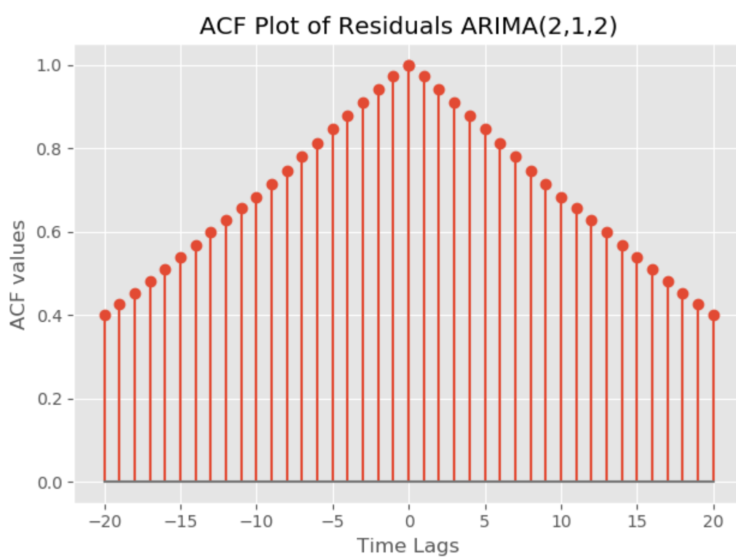
Given below are some diagnostics about the trained model. The top right and bottom left graphs show that training residuals are approximately normally distributed. The top left and bottom right graphs show that variance in training residuals increases with increasing time and lags values respectively.



<u>Computing the error metrics on test set:</u>
From results below, we see that values predicted by ARIMA coincide with real values only at few points.

```
Mean of ARIMA residuals 317.99146297460777
Variance of ARIMA residuals 192047.31500673303
RMSE of ARIMA residuals 541.4479527447346
```






```
ARIMA Residuals fail Chi-Square test.
```

Summary of Model Comparison:

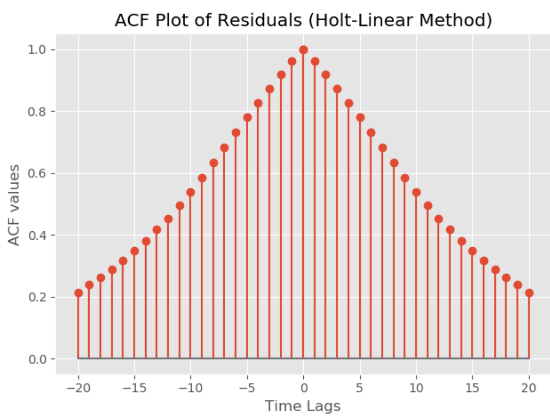
Picking the Best Model:

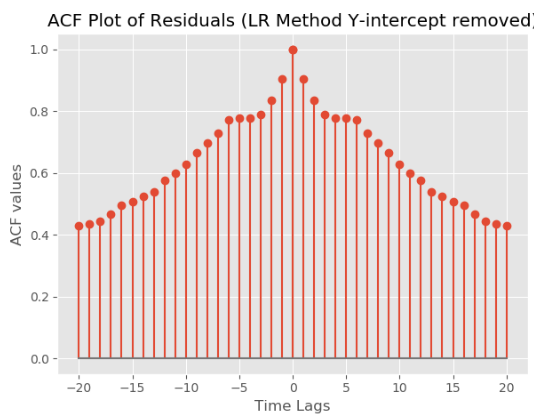
Based on **AIC and BIC values Holt-Linear is the winner** but based on **Prediction errors ARMA(1,0) performed the best**.

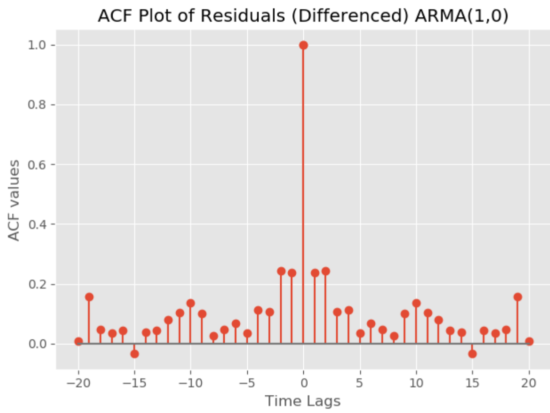
	Model Training		Prediction Errors		
	AIC	**BIC**	**Mean**	**Variance**	**RMSE**
Linear Regression without intercept	6896	6905	-110	44255	237
Holt-Linear	**3334**	**3356**	-628	100073	703
ARMA(1,0)	5426	5435	**-34**	**1677**	**53**
ARIMA	4986	5012	317	192047	541

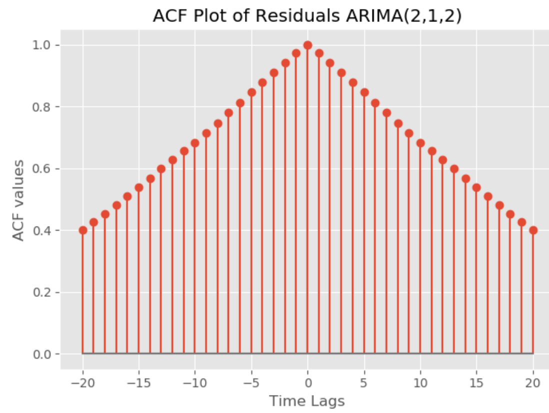
Comparison via ACF Plots:

Residual ACF plot of ARMA model seems closest to that of White Noise.



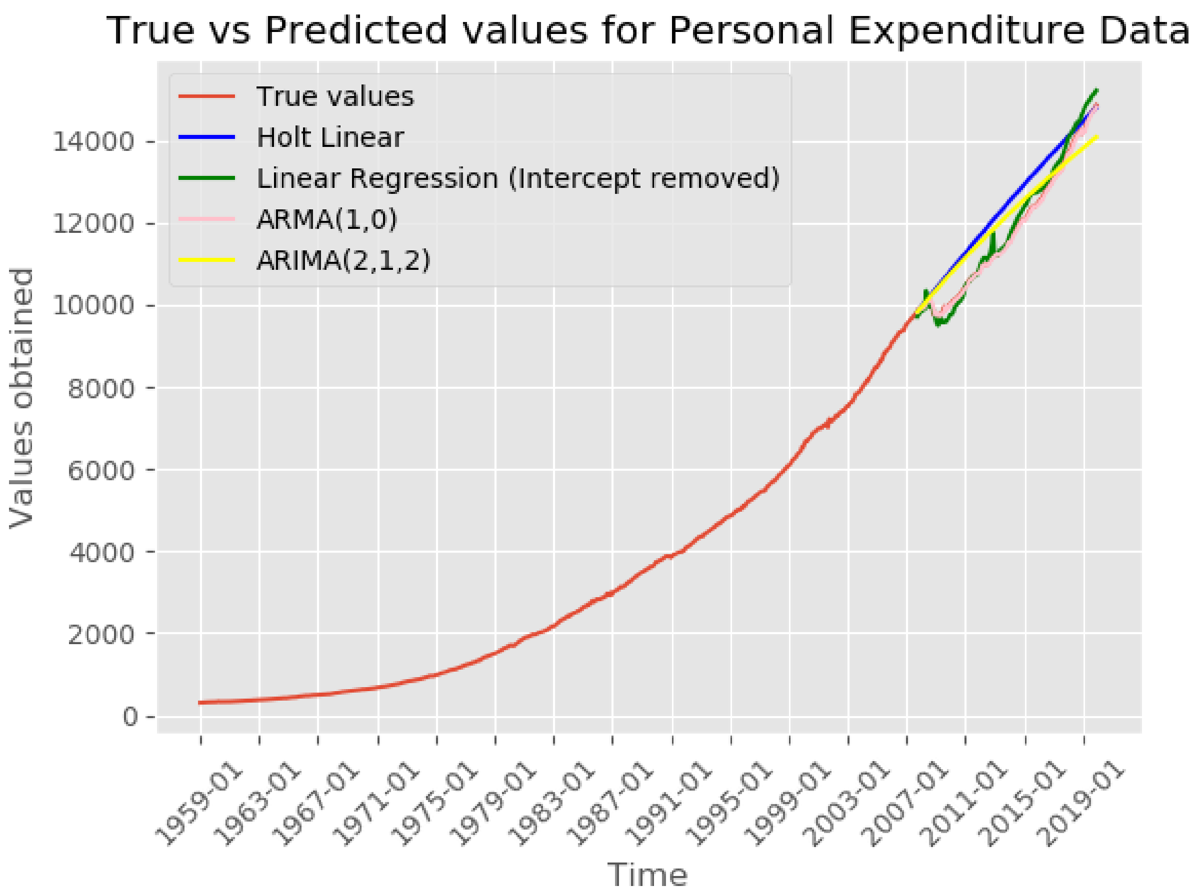






Comparison via True vs Predicted Forecasts:

ARMA(1,0) follows closely the red line whereas others deviate.



Conclusion:

ARMA(1,0) gives the best results because mean, variance, RMSE of residuals is the least. Also ACF of residual looks closest to white noise and predicted values follow almost exactly the same path as original values.

References:

Lecture Slides of Professor Reza Jafari
https://otexts.com/fpp2/
https://fred.stlouisfed.org/

Appendix:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import seaborn as sns
import statsmodels.formula.api as sm_ols        # for ols
import statsmodels.api as sm   # for ARMA
from sklearn.model_selection import train_test_split
from Toolbox import ADF_test, correlation_coefficient_cal, autocorrelation_cal, plot_ACF, visual_stationary_check,
autocorrelation_cal_k_lags
from Toolbox import holt_linear, holt_winter_seasonal, avg_method_hstep, ses_method_hstep, drift_method_hstep,
naive_method_hstep
from Toolbox import phi_kk, calc_ARMA, autocorrelation_cal_k_lags, plot_ACF_title, RMSE_calc
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy.stats import chi2
np.random.seed(42)
import warnings
warnings.filterwarnings("ignore")


## ******************************************************************************************************
# RAW DATA
## ******************************************************************************************************

# Dependent variable
personal_consump_exp = pd.read_csv('./Data/PCE.csv')
# Independent variables
unempl_rate = pd.read_csv('./Data/UNRATE.csv')
personal_income = pd.read_csv('./Data/PI.csv')



# plotting features and target variables against time
```

```
plt.subplot(2,1,1)
plt.plot(personal_income.DATE, personal_income.PI, label='Personal Income')
plt.plot(personal_consump_exp.PCE, label='Personal Expenditure')
plt.title('Personal Income & Expenditure (USA)')
plt.xlabel('Time')
plt.xticks(personal_income.DATE[::48], rotation=45)
plt.ylabel('Billions of USD')
plt.legend()

plt.subplot(2,1,2)
plt.plot(unempl_rate.DATE, unempl_rate.UNRATE, label='Unemployment rate')
plt.title('Unemployment rate (USA)')
plt.xlabel('Time')
plt.xticks(unempl_rate.DATE[::48], rotation=45)
plt.ylabel('Percentage')
plt.legend()
plt.show()

## ************************************************************************************************
# MERGED DATA
## ************************************************************************************************

# From plots above, we observe that starting and ending period for all columns is not same so need to fix that.
merged_inner = pd.merge(left=personal_income, right=personal_consump_exp, left_on='DATE', right_on='DATE')
merged_inner = pd.merge(left=merged_inner, right=unempl_rate, left_on='DATE', right_on='DATE')      # all values
start from 1959-01-01 and end on 2019-12-01

# Since we have monthly instead of daily data so creating a new column that contains only the month and year of
date
merged_inner['Date_Month'] = pd.to_datetime(merged_inner['DATE']).dt.to_period('M')
merged_inner.head()

# Now that we have a new Date_Month column, removing old DATE column
del merged_inner['DATE']
merged_inner.head()

# converting df to time series object by setting index col = Date_Month for easier manipulation e.g. in plots
# Period data type must 1st be converted into string or number type before setting it as index col for df
merged_inner['Date_Month'] = merged_inner['Date_Month'].astype(str)
merged_inner.set_index('Date_Month', inplace=True)

# creating a copy of combined dataframe so that original data remains safe
df = merged_inner.copy()
PCE = df.PCE



# Plotting the same plot as above but with similar starting and ending dates & formatted values on x-axis for time
plt.subplot(2,1,1)
plt.plot(df.PI, label='Personal Income')
plt.plot(df.PCE, label='Personal Expenditure')
plt.title('Personal Income & Expenditure (USA)')
plt.xlabel('Time')
```

```
plt.xticks(df.index[::48], rotation=45)
plt.ylabel('Billions of USD')
plt.legend()

plt.subplot(2,1,2)
plt.plot(df.UNRATE, label='Unemployment rate')
plt.title('Unemployment Rate (USA)')
plt.xlabel('Time')
plt.xticks(df.index[::48], rotation=45)
plt.ylabel('Percentage')
plt.legend()

plt.tight_layout()
plt.show()



# ----------------------------------------------------------------------
# Make sure that your dataset has at least 500 samples or more.
# ----------------------------------------------------------------------

print('Number of rows and columns in dataframe are:', df.shape, '\n')
N = len(df)     # total obs in df

# 1st 5 rows
df.head()
# last 5 rows
df.tail()



# ----------------------------------------------------------------------
# You must make sure that the dataset does not misses any samples. And the dataset is clean.
# ----------------------------------------------------------------------
null_rows = df[df.isna().any(axis=1)]
print('Rows containing missing (NaN) values are:', null_rows)       # Nil

df.info()

# ----------------------------------------------------------------------
# Make sure the time steps are equal.
# ----------------------------------------------------------------------

print('The first date in our data set is:', df.index[0])
print('The last date in our data set is:', df.index[-1])
print('So we have monthly data for ', 2019 - 1959 + 1, 'years. \nConsequently, 61*12 = ', 61*12, 'months which is
exactly '
'equal to \nthe number of rows we have in our dataframe.')


# ----------------------------------------------------------------------
# 8 (f)- Plot the ACF of the original dataset and see if the ACF decays.
# ----------------------------------------------------------------------
```

```
# all lags i.e. 732
acf = autocorrelation_cal(df.PCE)
a = plot_ACF(acf)

# ----------------------------------------------------------------------
# Correlation Matrix with seaborn heatmap and pearson’s correlation coefficient.
# ----------------------------------------------------------------------

sns.heatmap(df.corr(), vmin=-1, cmap='coolwarm', annot=True)
plt.show()

print('From heatmap, it is observed that there is a strong correlation between Personal Income & Personal
Expenditure '
'whereas these two variable have hardly any correlation with Unemployment. Hence, Unemployment should be
dropped from dataset'
'because apparently it does not hold any predictive power.')


# ----------------------------------------------------------------------
# SPLITTING DATA SET INTO TRAIN & TEST SET
# ----------------------------------------------------------------------
# dividing whole dataset into train & test set

train, test = train_test_split(df, test_size=0.2, shuffle=False)

# ----------------------------------------------------------------------
# CHECKING STATIONARITY OF DEPENDENT VARIABLE
# ----------------------------------------------------------------------


# visually checking if mean & var are constant
visual_stationary_check(df.PCE, df.index)

# objective test to confirm sty
adf_result = ADF_test(df.PCE)   # non-sty with p-value of 1


# ----------------------------------------------------------------------
# APPLYING FIRST ORDER DIFFERENCING TO TACKLE NON-STATIONARITY
# ----------------------------------------------------------------------

# to detrend, we are going to apply first order differencing technique
a = np.array((df.PCE)).copy()
PCE_differenced = a[1:] - a[0: len(a)-1]

# creating a separate df for differenced/Stationary data, it starts from Feb 1959 (instead of Jan 1959)
sty_df = pd.DataFrame(PCE_differenced, index=df.index[1:])
sty_df.rename(columns={0:'PCE'}, inplace=True)


# visually checking if mean & var are constant
visual_stationary_check(sty_df.PCE, sty_df.index)
```

```
# objective test to confirm sty
adf_result = ADF_test(sty_df.PCE)   # sty with p-value of 0.02


# # # log transformation
# log_df = np.log(a)
# adf_result = ADF_test(log_df)       # 0.07 hence failed ADF test
#
# # 1st order diff of log transformation
# PCE_log_diff = log_df[1:] - log_df[0: len(log_df)-1]
# ADF_test(PCE_log_diff)

# -----------------------------------------------------------------------
# REVERSE TRANSFORMATION OF DETRENDED DATA
# -----------------------------------------------------------------------

# reverse_diff = np.array(df.PCE.iloc[0: N-1]) + np.array(sty_df.PCE.iloc[0: ])   # 731 obs starting with Feb 1959
# -----------------------------------------------------------------------
# TIME SERIES DECOMPOSITION (original )
# -----------------------------------------------------------------------

result = seasonal_decompose(df.PCE, model='additive', freq=12)  # data collected every month hence freq=12
result.plot()
plt.suptitle('Additive Decomposition', fontsize=16 )
plt.xlabel('Time (monthly)')
plt.show()

result = seasonal_decompose(df.PCE, model='multiplicative', freq=12)
result.plot()
plt.suptitle('Multiplicative Decomposition', fontsize=16)
plt.xlabel('Time (monthly)')
plt.show()


print('************* Results for simple & advanced forecasting methods start here **********')

# # -----------------------------------------------------------------------
# 6- Holt-Winters method: Using the Holt-Winters method try to find the best fit using the train dataset and
# make a prediction using the test set.
# # -----------------------------------------------------------------------

l = len(test.PCE)


avg_forecast = avg_method_hstep(train.PCE)
# print('Forecast for Average Method', avg_forecast)

naive_forecast = naive_method_hstep(train.PCE)
# print('Forecast for Naive Method', naive_forecast)

drift_forecast = drift_method_hstep(train.PCE, l)
# print('Forecast for Drift Method', drift_forecast)
```

```
ses_forecast = ses_method_hstep(train.PCE)
# print('Forecast for SES Method', ses_forecast[-1])
# print(ses_forecast)



holt_linear_forecast = holt_linear(train.PCE, test.PCE)

holt_seasonal_forecast = holt_winter_seasonal(train.PCE, test.PCE)

# creating prediction plot
empty_list = [None for i in train.PCE]
plt.plot(df.PCE, label='True values')
plt.plot(empty_list + [avg_forecast]*l, label='Average')
plt.plot(empty_list + [naive_forecast]*l, label='Naive')
plt.plot(empty_list + drift_forecast, label='Drift')
plt.plot(empty_list + [ses_forecast[-1]]*l, label='SES')
plt.plot(empty_list + list(holt_linear_forecast), label='Holt Linear')
plt.plot(empty_list + list(holt_seasonal_forecast), label='Holt Winter Seasonal')
plt.legend()
plt.xlabel('Time')
plt.xticks(df.index[::48], rotation=45)
plt.ylabel('Values obtained')
plt.title('True vs Predicted values for Personal Expenditure Data')
plt.show()


# calculating residuals
residuals_avg   = test.PCE - [avg_forecast]*l
residuals_naive = test.PCE - [naive_forecast]*l
residuals_drift = test.PCE - drift_forecast
residuals_ses = test.PCE - [ses_forecast[-1]]*l
residuals_holt_linear = test.PCE - list(holt_linear_forecast)
residuals_holt_winter = test.PCE - list(holt_seasonal_forecast)

# ACF of Residuals
acf_residuals_avg = autocorrelation_cal_k_lags(residuals_avg, 20)
plot_ACF_title(acf_residuals_avg, 'Residuals (Average Method)')

acf_residuals_naive = autocorrelation_cal_k_lags(residuals_naive, 20)
plot_ACF_title(acf_residuals_naive, 'Residuals (Naive Method)')

acf_residuals_drift = autocorrelation_cal_k_lags(residuals_drift, 20)
plot_ACF_title(acf_residuals_drift, 'Residuals (Drift Method)')

acf_residuals_ses = autocorrelation_cal_k_lags(residuals_ses, 20)
plot_ACF_title(acf_residuals_ses, 'Residuals (SES Method)')

acf_residuals_holt_linear = autocorrelation_cal_k_lags(residuals_holt_linear, 20)
plot_ACF_title(acf_residuals_holt_linear, 'Residuals (Holt-Linear Method)')

acf_residuals_holt_winter = autocorrelation_cal_k_lags(residuals_holt_winter, 20)
plot_ACF_title(acf_residuals_holt_winter, 'Residuals (Holt-Winter Method)')
```

```
# RMSE of residuals
RMSE_avg = RMSE_calc(residuals_avg)
RMSE_naive = RMSE_calc(residuals_naive)
RMSE_drift = RMSE_calc(residuals_drift)
RMSE_ses = RMSE_calc(residuals_ses)
RMSE_holt_linear = RMSE_calc(residuals_holt_linear)    # has least RMSE= 703
RMSE_holt_winter = RMSE_calc(residuals_holt_winter)

print('Root Mean Square Error of Average Method is: ', RMSE_avg)
print('Root Mean Square Error of Naive Method is: ', RMSE_naive)
print('Root Mean Square Error of Drift Method is: ', RMSE_drift)
print('Root Mean Square Error of SES Method is: ', RMSE_ses)
print('Root Mean Square Error of Holt-Linear Method is: ', RMSE_holt_linear)
print('Root Mean Square Error of Holt-Winter Method is: ', RMSE_holt_winter)

# mean & variance of residuals for Holt-Linear method
print('Mean of residuals for Holt-Linear method is : ', np.mean(residuals_holt_linear))    # -628 biased
print('Variance of residuals for Holt-Linear method is : ', np.var(residuals_holt_linear))




# # ----------------------------------------------------------------------
# 8- Develop the multiple linear regression model that represent the dataset. Check the accuracy of the developed model.
# a. You need to include the complete regression analysis into your report.
# b. Hypothesis tests like F-test, t-test
# c. AIC, BIC, RMSE, R-squared and Adjusted R-squared
# d. ACF of residuals.
# e. Q-value
# f. Variance and mean of the residuals.
# # ----------------------------------------------------------------------


ols_model = sm_ols.ols("PCE ~ PI + UNRATE", data=train).fit()
ols_model_summary = ols_model.summary()
print(ols_model_summary)

predictions_ols = ols_model.predict(test)
# print(predictions_ols)
prediction_error_ols = test['PCE'] - predictions_ols

# creating prediction plot
empty_list = [None]*len(train)
plt.plot(df.PCE, label='True values')
plt.plot(empty_list + list(predictions_ols), label='Multiple Linear Regression')
plt.legend()
plt.xlabel('Time')
plt.xticks(df.index[::48], rotation=45)
plt.ylabel('Values obtained')
plt.title('True vs Predicted values (Linear Regression)')
```

```
plt.show()

# acf of Residuals
acf_residuals_ols = autocorrelation_cal_k_lags(prediction_error_ols, 20)
plot_ACF_title(acf_residuals_ols, 'Residuals (Multiple Linear Regression Method)')

mean_residuals_ols = np.mean(prediction_error_ols)
var_residuals_ols = np.var(prediction_error_ols)
RMSE_residuals_ols = np.sqrt(np.mean(prediction_error_ols**2))

# Q-value for OLS
T = len(test)
h = 20
Q_ols = T * np.sum(acf_residuals_ols[1:h]**2)

print('Mean, Variance, Q-value & RMSE of prediction error (OLS model) are {:.2f}, {:.2f}, {:.2f} & {:.2f}
respectively.'.format(mean_residuals_ols, var_residuals_ols, Q_ols, RMSE_residuals_ols))

## RERUN OLS WITH Y-INTERCEPT REMOVED

rerun_ols_model = sm_ols.ols("PCE ~ 0 + PI + UNRATE", data=train).fit()
rerun_model_summary = rerun_ols_model.summary()
print(rerun_model_summary)

rerun_predictions_ols = rerun_ols_model.predict(test)
# print(predictions_ols)
rerun_prediction_error_ols = test['PCE'] - rerun_predictions_ols

# creating prediction plot
empty_list = [None]*len(train)
plt.plot(df.PCE, label='True values')
plt.plot(empty_list + list(rerun_predictions_ols), label='Multiple Linear Regression')
plt.legend()
plt.xlabel('Time')
plt.xticks(df.index[::48], rotation=45)
plt.ylabel('Values obtained')
plt.title('Predictions for LR with Y-intercept removed')
plt.show()

# acf of Residuals
rerun_acf_residuals_ols = autocorrelation_cal_k_lags(rerun_prediction_error_ols, 20)
plot_ACF_title(rerun_acf_residuals_ols, 'Residuals (LR Method Y-intercept removed)')

rerun_mean_residuals_ols = np.mean(rerun_prediction_error_ols)
rerun_var_residuals_ols = np.var(rerun_prediction_error_ols)
rerun_RMSE_residuals_ols = np.sqrt(np.mean(rerun_prediction_error_ols**2))

# Q-value for OLS
T = len(test)
h = 20
Q_ols_rerun = T * np.sum(rerun_acf_residuals_ols[1:h]**2)

print('Mean, Variance, Q-value & RMSE of prediction error (OLS model with y-intercept removed) are {:.2f}, {:.2f},
```

```
{:.2f} & {:.2f} respectively.'.format(mean_residuals_ols, var_residuals_ols, Q_ols, RMSE_residuals_ols))

# ---------------------------------------------------------------------
# ARMA MODEL
# ---------------------------------------------------------------------

# using 1st order differenced (Stationary PCE) values for ARMA model
# calc ACF for GPAC
acf_sty_df = autocorrelation_cal_k_lags(sty_df.PCE, 20)     # lag 20
plot_ACF_title(acf_sty_df, 'Differenced PCE')


# creating a GPAC with j=8 & k=8
phi = []
for nb in range(8):
    for na in range(1,9):
        phi.append(phi_kk(nb, na,  acf_sty_df))


gpac = np.array(phi).reshape(8,8)
gpac_df = pd.DataFrame(gpac)
cols = np.arange(1,9)
gpac_df.columns = cols
print(gpac_df)
print()

sns.heatmap(gpac_df, annot=True)
plt.xlabel('AR process (k)')
plt.ylabel('MA process (j)')
plt.title('Heatmap of GPAC (ARMA process)')
plt.show()

# ****************************************************************************************
# estimate the corresponding parameters for the AR and MA part.
# ****************************************************************************************
def ARMA_estimates(na, nb):

    train_arma, test_arma = train_test_split(sty_df.PCE, test_size=0.2, shuffle=False)

    # ARMA parameter estimatation
    arma_model = sm.tsa.ARMA(train_arma, (na, nb)).fit(trend='nc', disp=0)

    for i in range(na):
        print('The AR coefficient a{}'.format(i), 'is:', arma_model.params[i])
    for i in range(nb):
        print('The MA coefficient b{}'.format(i), 'is:', arma_model.params[i+na])
    print()

    print(arma_model.summary())

    print('Confidence Interval (95%) for estimated paramters are:\n' , arma_model.conf_int(alpha=0.05))
    print()
```

```
# ***************************************************************************************
# 5- Display the estimated covariance matrix.
# ***************************************************************************************

print('The estimated covariance matrix for for model is: \n' , arma_model.cov_params())
print()

# ***************************************************************************************
# 6- Display the estimated variance of the prediction error.
# ***************************************************************************************

print('The estimated variance & standard deviation of prediction error :\n' , arma_model.sigma2,
np.sqrt(arma_model.sigma2))
print()

# ***************************************************************************************
# 8- Plot the true Sales value versus the estimated Sales value.
# ***************************************************************************************

# Prediction
# model_hat = arma_model.predict(start=test_arma.index[0], end=len(train_arma)+len(test_arma)-1)
model_hat = arma_model.predict(start=test_arma.index[0], end=test_arma.index[-1])


# ADF_test(sty_df.PCE)

# creating prediction plot
empty_list = [None]*len(train_arma)
plt.plot(sty_df.PCE, label= 'Original (differenced) values')
plt.plot(empty_list + list(model_hat), label='ARMA Predictions')
plt.legend()
plt.xlabel('Time')
plt.xticks(sty_df.index[::48], rotation=45)
plt.ylabel('Values obtained')
plt.title('True vs Predicted values (ARMA({},{}) Model)'.format(na,nb))
plt.show()


# ***************************************************************************************
# 9- Plot the ACF of the residuals.
# ***************************************************************************************

residuals_arma = pd.DataFrame(  model_hat - sty_df.PCE[len(train_arma):])
a = np.array(residuals_arma[0])
# print(a)
a = np.delete(a, -1)

print('Mean of ARMA residuals', a.mean())
print('Variance of ARMA residuals', a.var())
print('RMSE of ARMA residuals', np.sqrt(np.mean(a**2)))
```

```
    acf_residuals_arma = autocorrelation_cal_k_lags(a, 20)      #  lag 20
    title = 'Residuals (Differenced) ARMA({},{})'.format(na,nb)
    # N = len(test_arma)
    plot_ACF_title(acf_residuals_arma, title)

    # ***********************************************************************************************
    # 10 - Find Q value.
    # ***********************************************************************************************
    N = len(test_arma)
    Q_arma = N * (np.sum(acf_residuals_arma[1:]**2))

    # ***********************************************************************************************
    # 11 - Are the residuals errors white? Knowing the DOF and alfa = .01 perform a   test and check if the residuals pass the whiteness test.

    # Q must be less than Qc
    # ***********************************************************************************************

    DOF = 20 - na - nb
    # define probability
    alpha = 0.01

    # retrieve value <= probability
    Q_critical = chi2.ppf(1-alpha, DOF)

    if Q_arma < Q_critical:
        print('Q ({}) < Q_c ({}) hence residuals pass the whiteness test confirming that suggested model is good.'.format(Q_arma, Q_critical))
    else:
        print('Residuals fail Chi-Square test.')

    # Reverse transforming the ARMA predictions
    reverse_diff_arma_predictions = np.array(df.PCE.iloc[584: 731]) + np.array(model_hat[0: ])   # 731 obs starting with Feb 1959

    # creating prediction plot
    empty_list = [None]*len(train)
    plt.plot(df.PCE, label='True values')
    plt.plot(empty_list + list(reverse_diff_arma_predictions), label='ARMA')
    plt.legend()
    plt.xlabel('Time')
    plt.xticks(df.index[::48], rotation=45)
    plt.ylabel('Values obtained')
    plt.title('True vs Predicted values ARMA({},{}) (After reverse Transform)'.format(na,nb))
    plt.show()

    return residuals_arma, reverse_diff_arma_predictions

print('*********  Results for ARMA(1,0) start here *************')

residuals_arma, reverse_diff_arma_predictions = ARMA_estimates(1,0)
# reverse_diff_arma_predictions = ARMA_estimates(1,0)
# print(reverse_diff_arma_predictions)
```

```
residuals_arma_reverse_transformed = df.PCE[584:731] - reverse_diff_arma_predictions

# calc ACF for ARMA residuals after reverse transforming data
acf_sty_df = autocorrelation_cal_k_lags(residuals_arma_reverse_transformed, 20)     # lag 20
plot_ACF_title(acf_sty_df, 'ARMA (Reverse Transformed) residuals')



print('*********  Results for ARMA(0,1) start here *************')
ARMA_estimates(0,1)
print()

print('*********  Results for ARMA(0,2) start here *************')
residuals_arma = ARMA_estimates(0,2)
print()

# ARMA(2,2) RAISES ERROR
# print('*********  Results for ARMA(2,2) start here *************')
# residuals_arma = ARMA_estimates(2,2)
# print()

# PASSING ARMA(1,0)'S RESIDUAL to GPAC
def create_gpac(ts):
   acf_sty_df = autocorrelation_cal_k_lags(ts, 30)     # lag 30
   # plot_ACF_title(acf_sty_df, 'Residuals GPAC')


   # creating a GPAC with j=8 & k=8
   phi = []
   for nb in range(15):
       for na in range(1,16):
           phi.append(phi_kk(nb, na,  acf_sty_df))


   gpac = np.array(phi).reshape(15,15)
   gpac_df = pd.DataFrame(gpac)
   cols = np.arange(1,16)
   gpac_df.columns = cols
   print('GPAC of residuals ARMA(1,0)')
   print(gpac_df)
   print()

   return gpac_df

gpac_arma_residuals = create_gpac(np.array(residuals_arma[0]))

print('*********  Results for ARMA(10,0) start here *************')
ARMA_estimates(10,0)
print('*********  Results for ARMA(11,0) start here *************')
ARMA_estimates(11,0)
print('*********  Results for ARMA(16,0) start here *************')
ARMA_estimates(16,0)
```

```
### ARIMA MODEL
print('****************** Auto ARIMA Model results start here *************')
import pmdarima as pm

# fitting arima model
arima_model = pm.auto_arima(train.PCE, test='adf')
print(arima_model.summary())
arima_model.plot_diagnostics(figsize=(8,8))
plt.show()

# predicting
arima_model_hat = arima_model.predict(test.shape[0])  # predict N steps into the future


# creating prediction plot
empty_list = [None]*len(train)
plt.plot(df.PCE, label= 'Original values')
plt.plot(empty_list + list(arima_model_hat), label='ARIMA(2,1,2) Predictions')
plt.legend()
plt.xlabel('Time')
plt.xticks(sty_df.index[::48], rotation=45)
plt.ylabel('Values obtained')
plt.title('True vs Predicted values (ARIMA(2,1,2) Model)')
plt.show()


# ********************************************************************************************
# 9- Plot the ACF of the residuals.
# ********************************************************************************************

residuals_arima = pd.DataFrame( arima_model_hat - test.PCE)
a = np.array(residuals_arima.PCE)
# print(a)
# a = np.delete(a, -1)

# print(len(residuals_arima))
# print(len(test))

print('Mean of ARIMA residuals', a.mean())
print('Variance of ARIMA residuals', a.var())
print('RMSE of ARIMA residuals', np.sqrt(np.mean(a**2)))


acf_residuals_arima = autocorrelation_cal_k_lags(a, 20)      #  lag 20
title = 'Residuals ARIMA(2,1,2)'
# N = len(test_arma)
plot_ACF_title(acf_residuals_arima, title)

# ********************************************************************************************
# 10 - Find Q value.
# ********************************************************************************************
N = len(test)
```

```
Q_arima = N * (np.sum(acf_residuals_arima[1:]**2))

# ************************************************************************************************
# 11 - Are the residuals errors white? Knowing the DOF and alfa = .01 perform a   test and check if the residuals pass the whiteness test.

# Q must be less than Qc
# ************************************************************************************************

DOF = 20 - 2 - 2
# define probability
alpha = 0.01

# retrieve value <= probability
Q_critical = chi2.ppf(1-alpha, DOF)

if Q_arima < Q_critical:
    print('Q ({}) < Q_c ({}) hence residuals pass the whiteness test confirming that suggested model is good.'.format(Q_arma, Q_critical))
else:
    print('ARIMA Residuals fail Chi-Square test.')


# print('****************** Auto ARIMA Model with y-intercept removed *************')
# arima_model = pm.auto_arima(train.PCE, test='adf', with_intercept=False)
# print(arima_model.summary())
# arima_model.plot_diagnostics(figsize=(8,8))
# plt.show()

#
# plotting just holt-linear, LR & ARMA(0,1), ARIMA(2,1,2) predictions for final comparison
# creating prediction plot
empty_list = [None for i in train.PCE]
plt.plot(df.PCE, label='True values')
plt.plot(empty_list + list(holt_linear_forecast), color='blue', label='Holt Linear')
plt.plot(empty_list + list(rerun_predictions_ols), color='green', label='Linear Regression (Intercept removed)')
plt.plot(empty_list + list(reverse_diff_arma_predictions), color='pink', label='ARMA(1,0)')
plt.plot(empty_list + list(arima_model_hat), color='yellow', label='ARIMA(2,1,2)')
plt.legend()
plt.xlabel('Time')
plt.xticks(df.index[::48], rotation=45)
plt.ylabel('Values obtained')
plt.title('True vs Predicted values for Personal Expenditure Data')
plt.show()
```