Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Report on Mini Project

Time Series Analysis (DJ19DSC5012)
AY: 2021-22

Data is the new 'Oil'

Daksh Dholakia	60009200040
Ojas Karmarkar	60009200042
Tanishka Kothari	60009200052

Guided By
Prof. Pradnya Saval

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

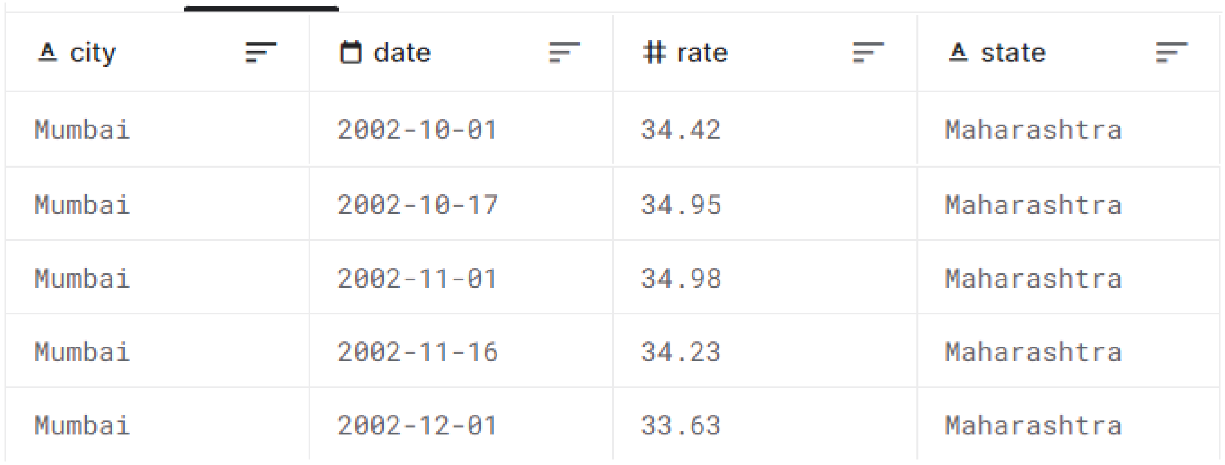
Given the variations in the fuel prices recently , we chose the dataset of petrol prices in Mumbai from the year 2002 - 2020.

Various factors affect the fuel prices in our country , such as **Tax Rates , Mismatch of Supply & Demand , Cost of Crude Oil.**

To study these variations , irregularities in the fuel prices correct time series model(s) and techniques must be used.

This report includes the entire process of data cleaning to fitting the model and forecasting the possible fuel prices based on the historical data available.

CHAPTER 2: Data Description



city	date	rate	state
Mumbai	2002-10-01	34.42	Maharashtra
Mumbai	2002-10-17	34.95	Maharashtra
Mumbai	2002-11-01	34.98	Maharashtra
Mumbai	2002-11-16	34.23	Maharashtra
Mumbai	2002-12-01	33.63	Maharashtra

The original dataset contained 4 columns namely ,

1. **City**
2. **Date**
3. **Rate (In Rupees)**
4. **State.**

Columns **City** and **State** represent the same variable (**Rate**) at a single time over different geographical locations.

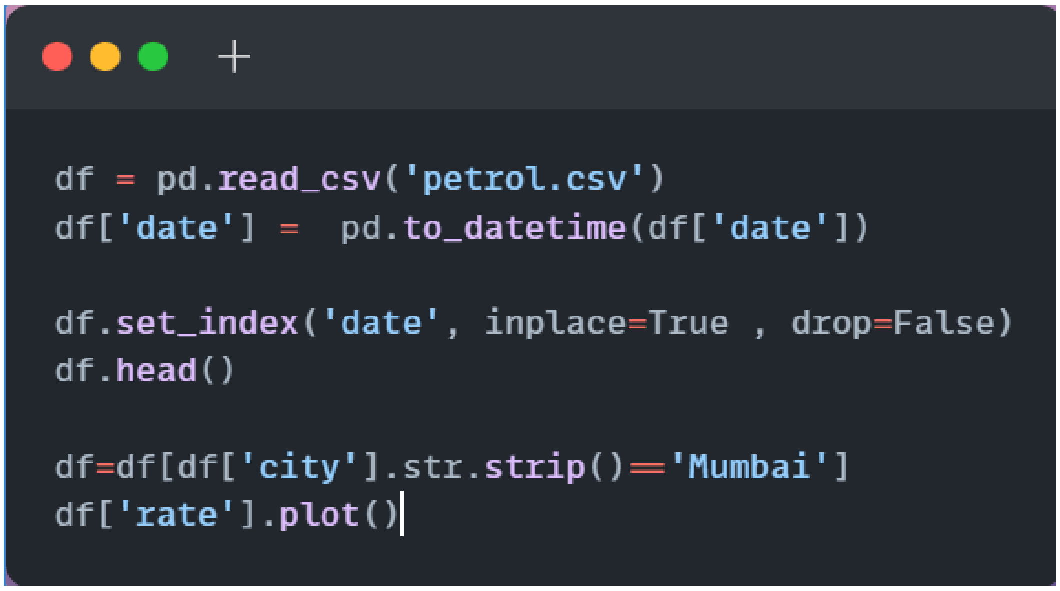
Thus the data used is **Cross Sectional Data**.

CHAPTER 3: Objective

Objective of the above Time series data includes :

1. Data **analysis** and **cleaning**
2. Finding out **trends** in data
3. Checking for **Seasonality**
4. **Smoothening** the data
5. Identifying **parameters** of the model
6. Fitting the right **model**
7. Calculating **accuracy matrix**
8. **Forecasting** future prices

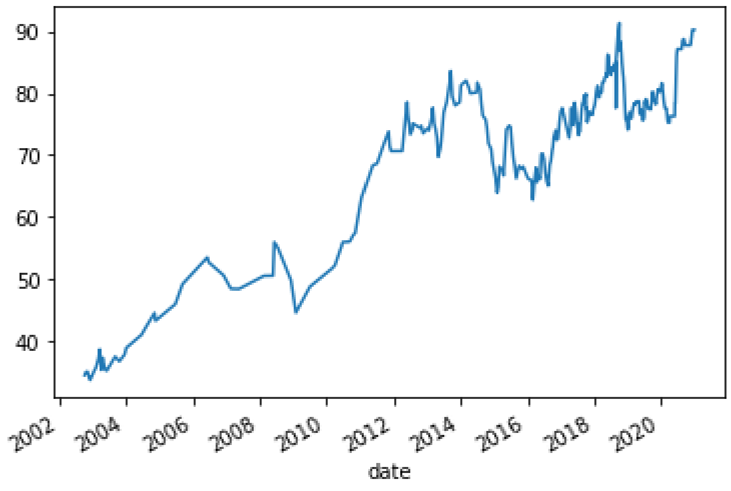
CHAPTER 4: Data Cleaning



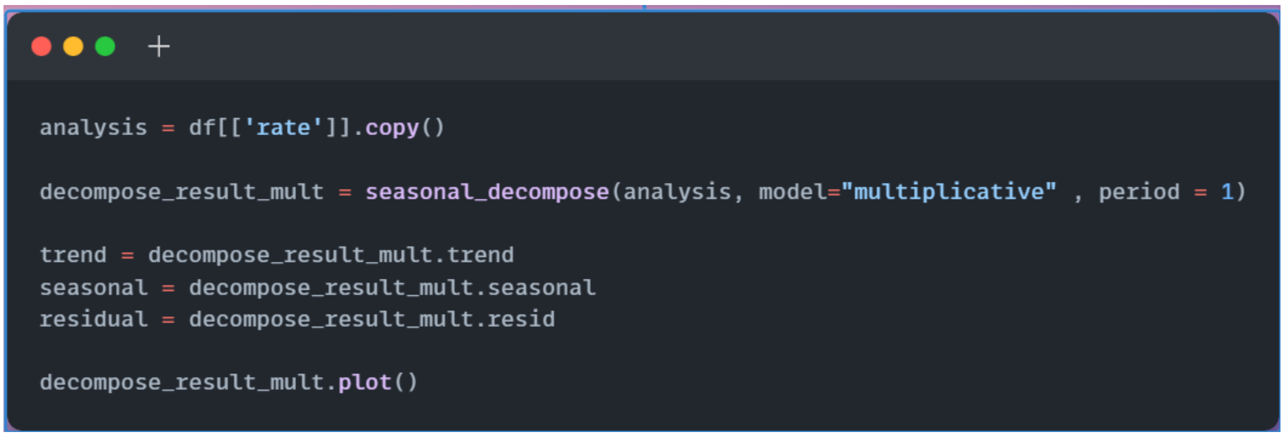
```
df = pd.read_csv('petrol.csv')
df['date'] =  pd.to_datetime(df['date'])

df.set_index('date', inplace=True , drop=False)
df.head()

df=df[df['city'].str.strip()=='Mumbai']
df['rate'].plot()
```

Petrol.csv contains rates of petrol in various cities and states at the **same timestamp**.

This, when plotted , looks very cluttered , hence the new data contains prices of only **Mumbai , Maharashtra**.



CHAPTER 5: Data Decomposition



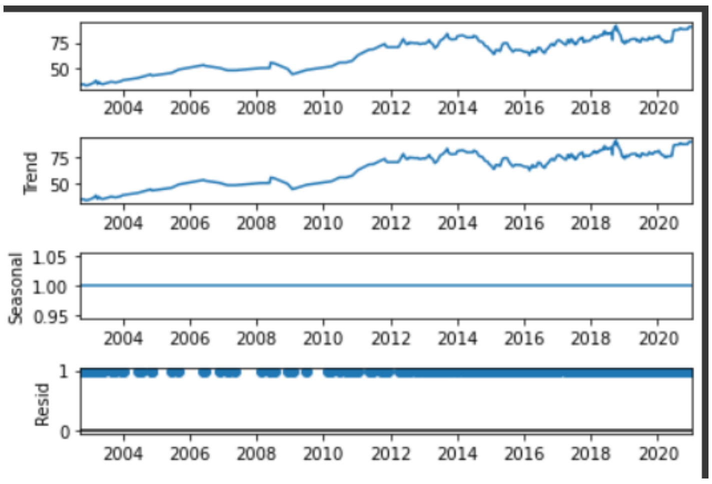
```
analysis = df[['rate']].copy()

decompose_result_mult = seasonal_decompose(analysis, model="multiplicative" , period = 1)

trend = decompose_result_mult.trend
seasonal = decompose_result_mult.seasonal
residual = decompose_result_mult.resid

decompose_result_mult.plot()
```

Any Time Series data consists of 3 components :

1. **Trend**
2. **Seasonal**
3. **Residual**

Above code decomposes the time series data into those 3 components over a **period of 1 year.**

This helps in Identifying seasonality & trend if present.

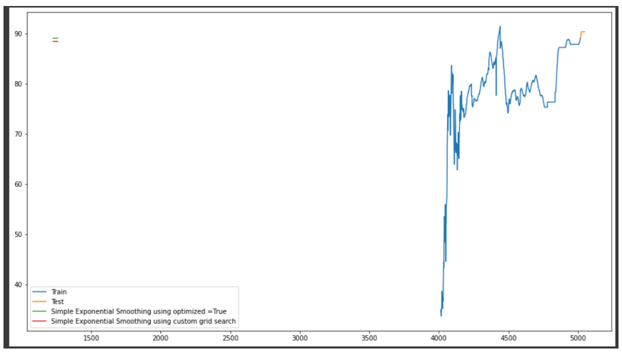


CHAPTER 6: Data Smoothing

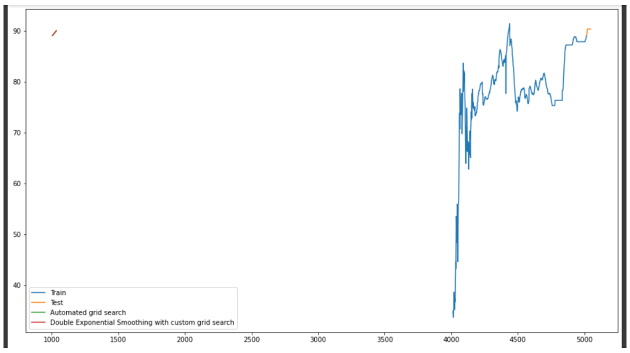
Smoothing is a method we can use to create a function to remove irregularities in data and attempt to capture significant patterns.

Methods used for smoothing of data :

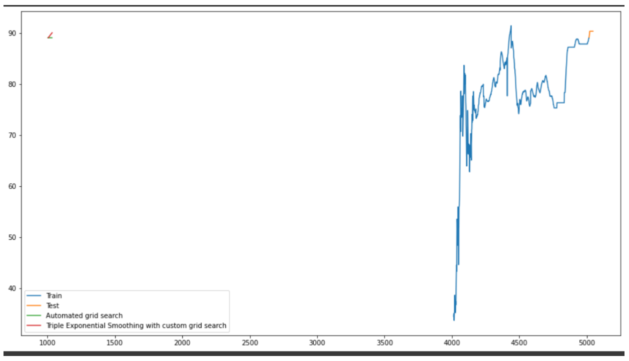
1. **Simple Exponential Smoothing (SEE)**
2. **Double Exponential Smoothing (DEE)**
3. **Triple Exponential Smoothing (TEE)**



(SEE)



(DEE)

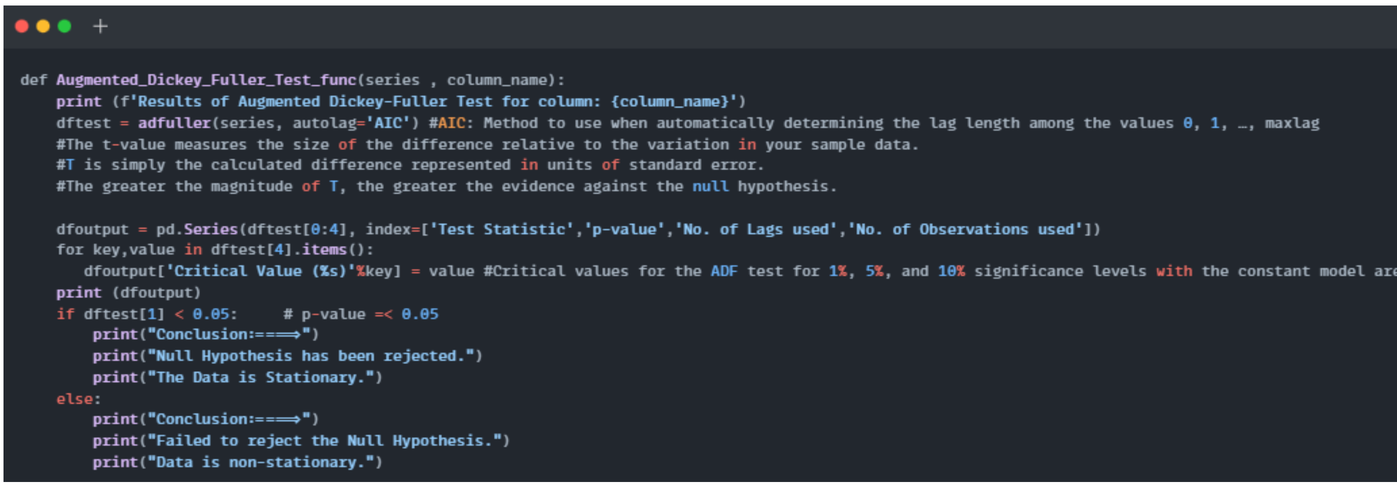


(TEE)

CHAPTER 7: Testing Stationarity

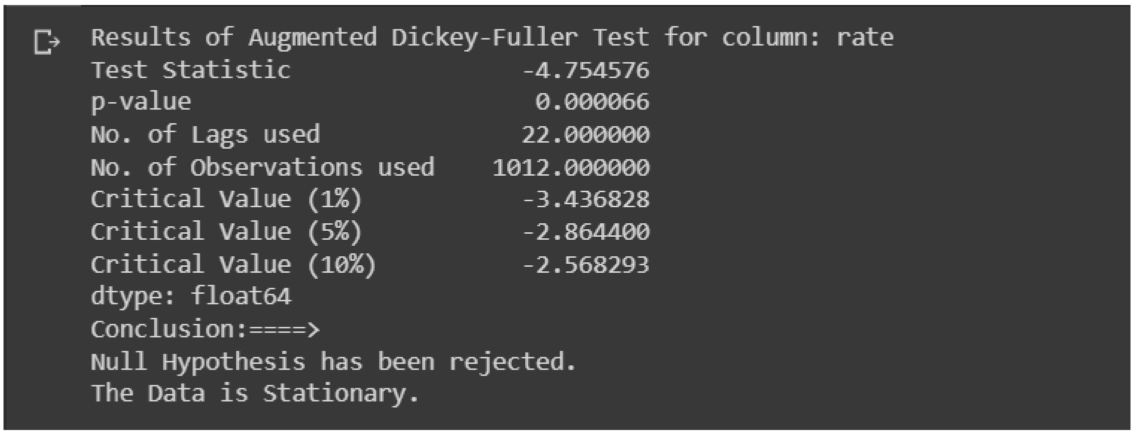
Stationarity deals with unit roots. To check for stationarity **ADF (Augmented Dickey Fuller)** test has to be used.

Checking for unit roots is mandatory because not all models can be applied to a TS that contains unit roots.



```
def Augmented_Dickey_Fuller_Test_func(series , column_name):
    print (f'Results of Augmented Dickey-Fuller Test for column: {column_name}')
    dftest = adfuller(series, autolag='AIC') #AIC: Method to use when automatically determining the lag length among the values 0, 1, …, maxlag
    #The t-value measures the size of the difference relative to the variation in your sample data.
    #T is simply the calculated difference represented in units of standard error.
    #The greater the magnitude of T, the greater the evidence against the null hypothesis.

    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','No. of Lags used','No. of Observations used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value #Critical values for the ADF test for 1%, 5%, and 10% significance levels with the constant model are
    print (dfoutput)
    if dftest[1] < 0.05:     # p-value =< 0.05
        print("Conclusion:====>")
        print("Null Hypothesis has been rejected.")
        print("The Data is Stationary.")
    else:
        print("Conclusion:====>")
        print("Failed to reject the Null Hypothesis.")
        print("Data is non-stationary.")
```

We reject the null hypothesis since **p-value < 0.05 and test-stat < critical values**.

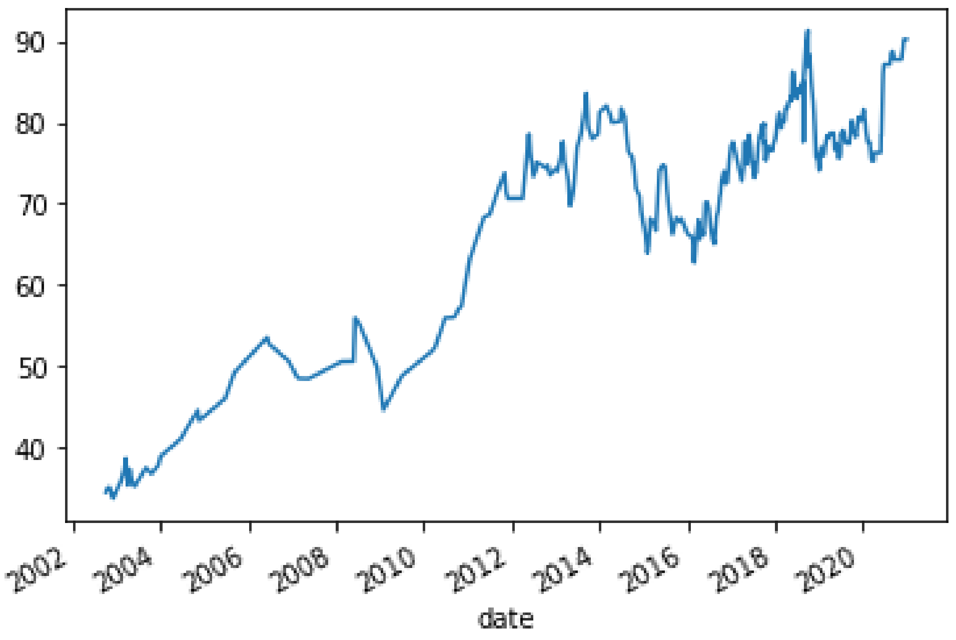


```
Results of Augmented Dickey-Fuller Test for column: rate
Test Statistic                  -4.754576
p-value                          0.000066
No. of Lags used                22.000000
No. of Observations used      1012.000000
Critical Value (1%)             -3.436828
Critical Value (5%)             -2.864400
Critical Value (10%)            -2.568293
dtype: float64
Conclusion:====>
Null Hypothesis has been rejected.
The Data is Stationary.
```

CHAPTER 8: WHY? Justification of TS

Reasons for classifying it as a TS data :

1. The fuel rates are continuously increasing with **some irregularities**.
2. The **data is continuous** i.e. there exists rate for every timestamp
3. There is a **slight seasonality** , over the period of **1 year**.
4. This data can be decomposed into the **three components** of TS



CHAPTER 9 : Implementation & Forecasting

Various methods and algorithms can be applied for implementing a time series model and interpretation of its forecast generated.
From implementation we can interpret that forecasts generated by models are able to capture irregularity in data and give somewhat accurate predictions.

Some methods implemented by us are :

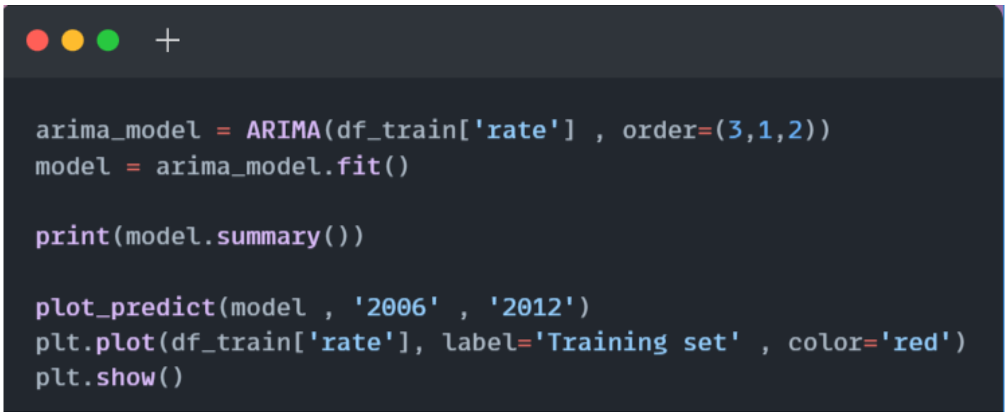
1. **Autoregressive Integrated Moving Average(ARIMA) Model**
   Autoregressive integrated moving average—also called ARIMA(p,d,q)—is a forecasting equation that can make time series stationary with the help of differencing and log techniques when required. A time series that should be differentiated to be stationary is an integrated (d) (I) series. Lags of the stationary series are classified as autoregressive (p), which is designated in (AR) terms. Lags of the forecast errors are classified as moving averages (q), which are identified in (MA) terms.

2. **Convolutional Neural Network(CNN) Model**
   We have seen examples on using CNN for sequence prediction. If we consider Dow Jones Industrial Average (DJIA) as an example, we may build a CNN with 1D convolution for prediction. This makes sense because a 1D convolution on a time series is roughly computing its moving average or using digital signal processing terms, applying a filter to the time series. It should provide some clues about the trend.

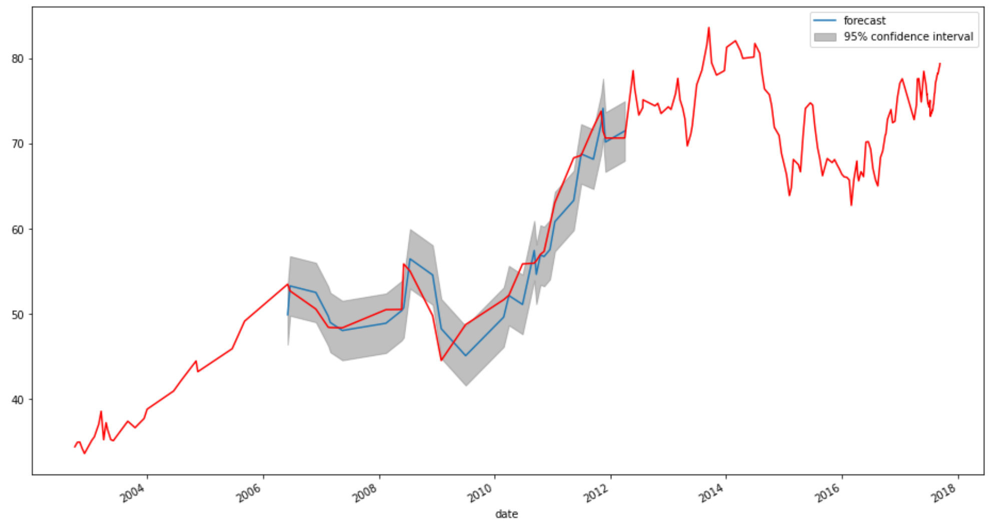
3. **Autoregressive Conditional Heteroskedasticity(ARCH) Model**
   Autoregressive models can be developed for univariate time-series data that is stationary (AR), has a trend (ARIMA), and has a seasonal

component (SARIMA). But, these Autoregressive models do not model a change in the variance over time. The error terms in the stochastic processes generating the time series were homoscedastic, i.e. with constant variance. There are some time series where the variance changes consistently over time. In the context of a time series in the financial domain, this would be called increasing and decreasing volatility.

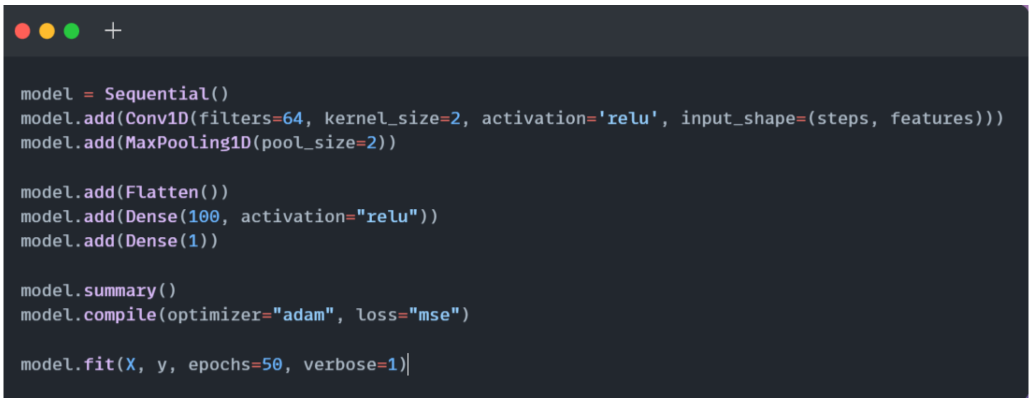
ARIMA :



```
arima_model = ARIMA(df_train['rate'] , order=(3,1,2))
model = arima_model.fit()

print(model.summary())

plot_predict(model , '2006' , '2012')
plt.plot(df_train['rate'], label='Training set' , color='red')
plt.show()
```

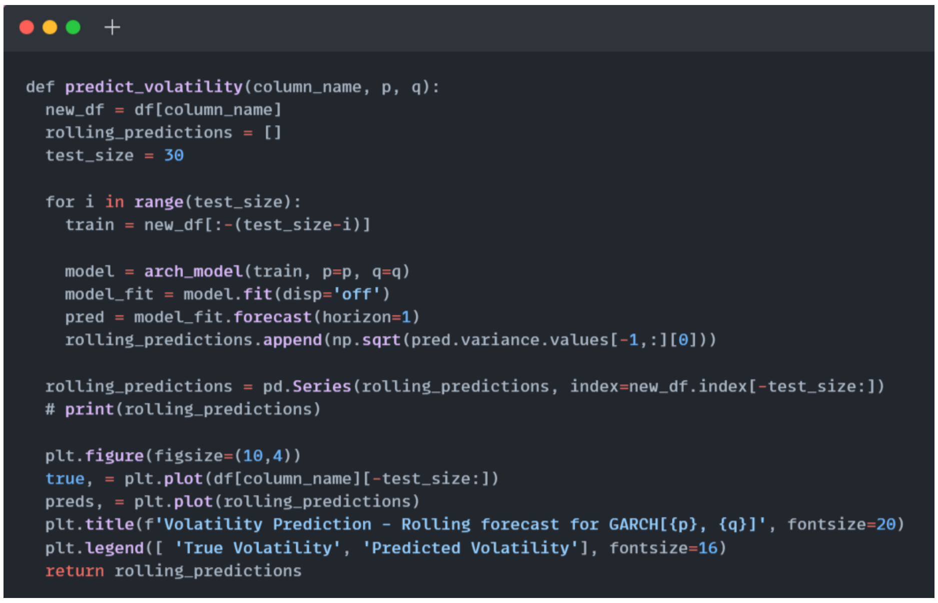


CNN :

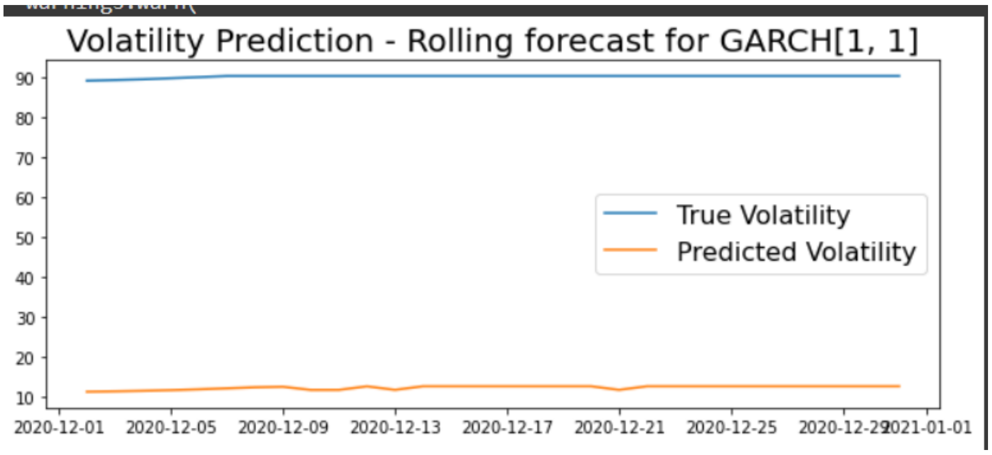


```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(steps, features)))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())
model.add(Dense(100, activation="relu"))
model.add(Dense(1))

model.summary()
model.compile(optimizer="adam", loss="mse")

model.fit(X, y, epochs=50, verbose=1)
```

ARCH/GARCH :

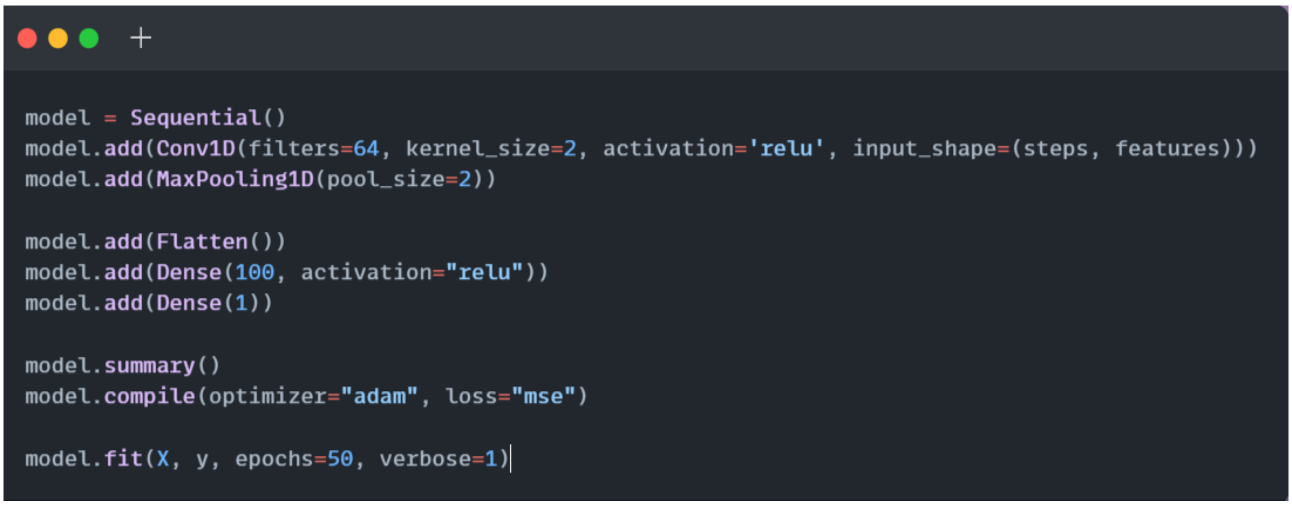


```
def predict_volatility(column_name, p, q):
  new_df = df[column_name]
  rolling_predictions = []
  test_size = 30

  for i in range(test_size):
    train = new_df[:-(test_size-i)]

    model = arch_model(train, p=p, q=q)
    model_fit = model.fit(disp='off')
    pred = model_fit.forecast(horizon=1)
    rolling_predictions.append(np.sqrt(pred.variance.values[-1,:][0]))

  rolling_predictions = pd.Series(rolling_predictions, index=new_df.index[-test_size:])
  # print(rolling_predictions)

  plt.figure(figsize=(10,4))
  true, = plt.plot(df[column_name][-test_size:])
  preds, = plt.plot(rolling_predictions)
  plt.title(f'Volatility Prediction - Rolling forecast for GARCH[{p}, {q}]', fontsize=20)
  plt.legend([ 'True Volatility', 'Predicted Volatility'], fontsize=16)
  return rolling_predictions
```



CHAPTER 10 : Reasons for selection of Model

For this TS data , **CNN** model was used , based on various metrics like rmse , mse etc.

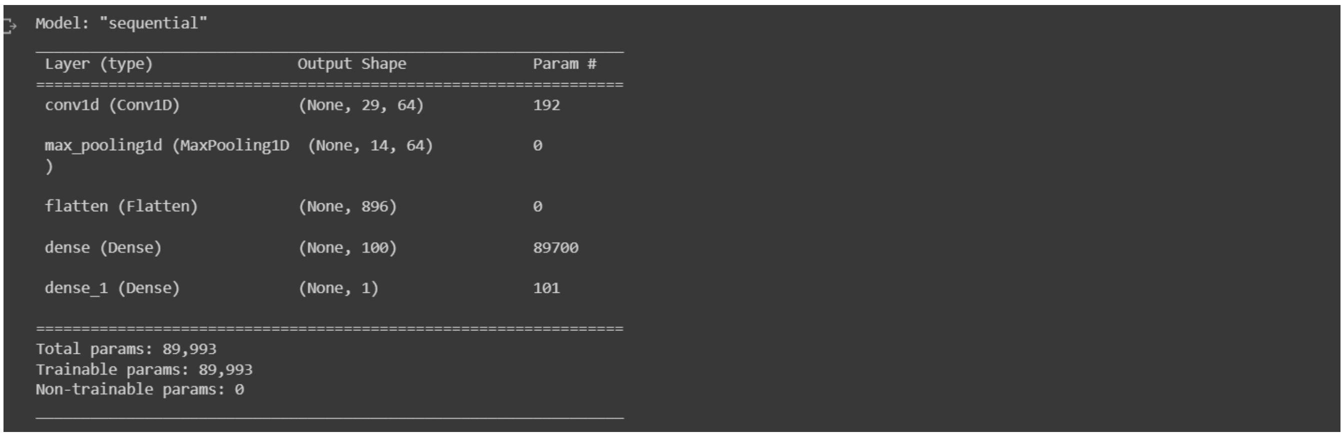


```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(steps, features)))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())
model.add(Dense(100, activation="relu"))
model.add(Dense(1))

model.summary()
model.compile(optimizer="adam", loss="mse")

model.fit(X, y, epochs=50, verbose=1)
```

Reasons for **CNN** model :

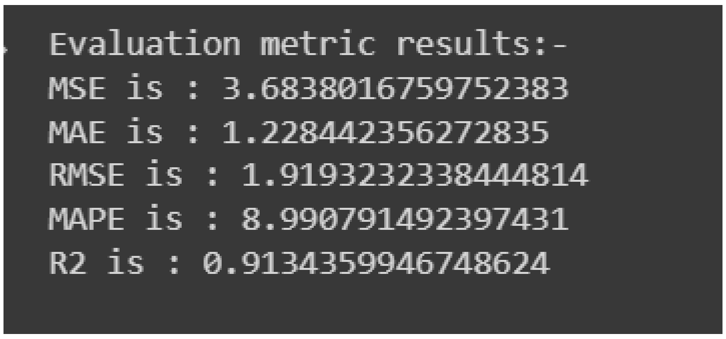
1. The fuel prices data was **not very volatile** , since the use of ARCH/GARCH model .
2. Seasonality too was not present , thus **SARIMA is not used**.
3. **Data was stationary** (No Unit roots). Thus an AR/MA or ARIMA or CNN model can be fitted.
4. CNN and ARIMA were both accurate , but CNN performed better while forecasting.

Model Summary of CNN :



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 29, 64)            192

 max_pooling1d (MaxPooling1D  (None, 14, 64)           0
 )

 flatten (Flatten)           (None, 896)               0

 dense (Dense)               (None, 100)               89700

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 89,993
Trainable params: 89,993
Non-trainable params: 0
_________________________________________________________________
```

Error Metrics for CNN model :



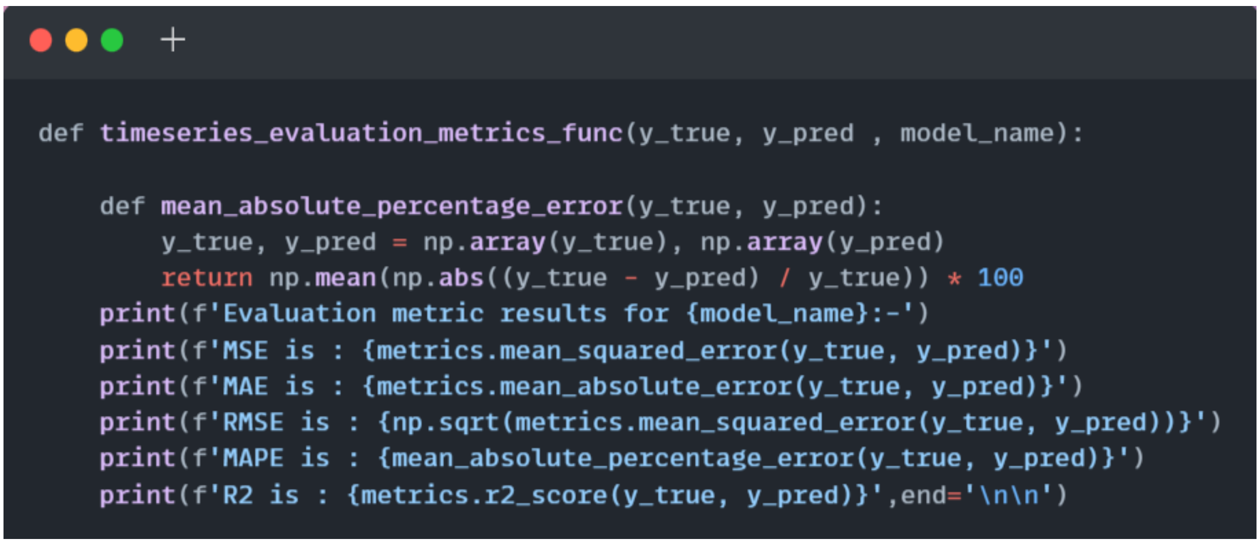
```
Evaluation metric results:-
MSE is : 3.6838016759752383
MAE is : 1.228442356272835
RMSE is : 1.9193232338444814
MAPE is : 8.990791492397431
R2 is : 0.9134359946748624
```

CHAPTER 11 : Comparative Analysis

The Forecasts of ARIMA model were compared on different error parameters such as :

1. Mean Error (ME)
2. Root Mean Squared Error (RMSE)
3. Mean Absolute Error (MAE)
4. Mean Percentage Error (MAE)
5. Mean Absolute Percentage Error (MAPE)

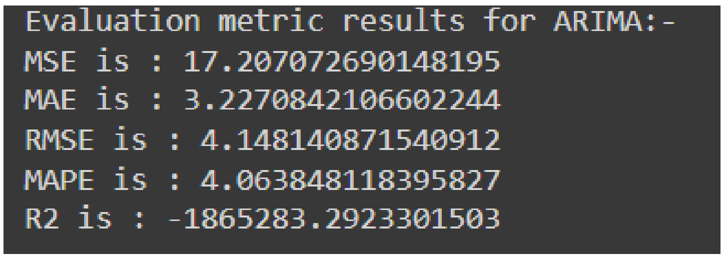
This function below is common function used by different model to evaluate the performance of each model



```python
def timeseries_evaluation_metrics_func(y_true, y_pred , model_name):

    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    print(f'Evaluation metric results for {model_name}:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(y_true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error(y_true, y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',end='\n\n')
```

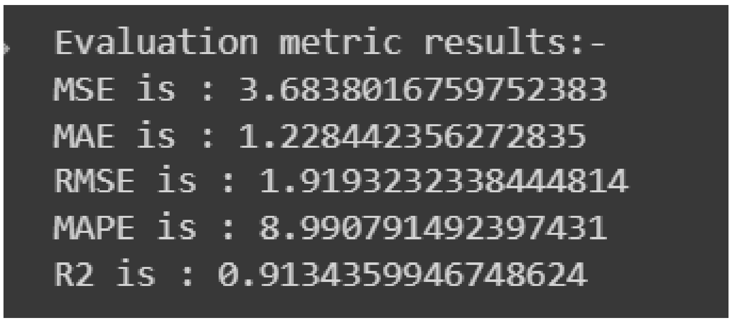
Comparison of CNN , ARIMA & ARCH were done on the above mentioned error metrics & based on that the final model was selected for prediction.

Error Metrics for ARIMA :



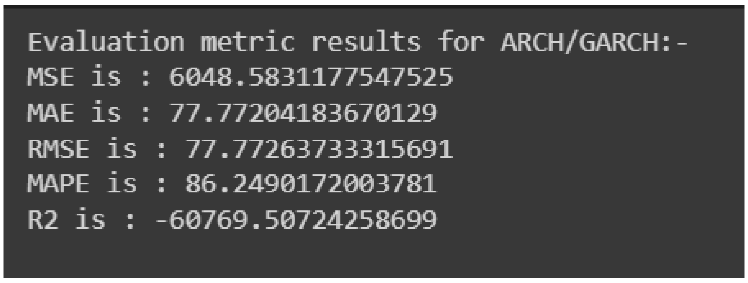
```
Evaluation metric results for ARIMA:-
MSE is : 17.207072690148195
MAE is : 3.2270842106602244
RMSE is : 4.148140871540912
MAPE is : 4.063848118395827
R2 is : -1865283.2923301503
```

Error Metrics for CNN :



```
Evaluation metric results:-
MSE is : 3.6838016759752383
MAE is : 1.228442356272835
RMSE is : 1.9193232338444814
MAPE is : 8.990791492397431
R2 is : 0.9134359946748624
```

Error Metrics for ARCH :



```
Evaluation metric results for ARCH/GARCH:-
MSE is : 6048.5831177547525
MAE is : 77.77204183670129
RMSE is : 77.77263733315691
MAPE is : 86.2490172003781
R2 is : -60769.50724258699
```

Thus the comparative analysis shows that CNN performs the best , whereas ARCH performs the worst.

CHAPTER 12 & 13 : Conclusion & Colab

The Data obtained from **kaggle** seemed to be cluttered when plotted. Thus data of only one city **(Mumbai)** was considered for further **fitting , modelling and predicting.**

The final selected model was **CNN.**

The results of the **CNN** model were compared on different error measurements.

GOOGLE COLAB LINK 👇

https://colab.research.google.com/drive/1XhJjnycDzrycraJMojQMlHtm9lzw_Yvv?usp=sharing

CHAPTER 14 & 15 : Future Scope & References

This type of TS data can be used for :

1. **Price change with respect to a global price change**
2. **Analysing how the govt change overtime affected the prices**
3. **Inspecting for high inflation rates**
4. **Checking for changes in tax rates**
5. **Irregularities in such data can point to a major global event at that timestamp.**

REFERENCES :

1. Fuel Price In India | Kaggle
2. Regression Metrics for Machine Learning - MachineLearningMastery.com
3. An overview of time series forecasting models | by Davide Burba | Towards Data Science
4. Statsmodels
5. Smoothing of time series | Statistical Software for Excel