DS203- 2021 Project

New Section

Installing required libraries

```
!pip install yfinance
!pip install yahoofinancials
!pip install fastai
!pip install pmdarima
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dis
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/loc
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: yahoofinancials in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pytz in /usr/local/lib/python3.7/dist-packages (frc
Requirement already satisfied: fastai in /usr/local/lib/python3.7/dist-packages (1
Requirement already satisfied: spacy>=2.0.18 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: fastprogress>=0.2.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: bottleneck in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: nvidia-ml-py3 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: numexpr in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: torch>=1.0.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/d
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /usr/local/lib/python3.7/dis
```

```
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /usr/local/lib/python3.7
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /usr/local/lib/python3.7/dist
Requirement already satisfied: blis<0.5.0,>=0.4.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: importlib-metadata>=0.20 in /usr/local/lib/python3.
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/loc
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/lo
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages
```

Importing Basic Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pt
import yfinance as yf
from yahoofinancials import YahooFinancials
%matplotlib inline
from fastai.tabular import add_datepart
```

creating dataframe for stock and EDA

```
# stock for TATAMOTORS
stock = "TATAMOTORS.NS"
```

```
# accessing data
df = yf.download(stock, start='2012-01-01', end='2021-11-24', progress=False)
df.tail()
```

Date	Open	High	Low	Close	Adj Close	Volume
2021-11-16	506.899994	526.849976	506.200012	519.049988	519.049988	55897781
2021-11-17	520.250000	536.700012	520.250000	530.150024	530.150024	48463415
2021-11-18	531.450012	534.200012	501.299988	509.700012	509.700012	47197742
2021-11-22	512.250000	512.250000	478.399994	486.100006	486.100006	40044849
2021-11-23	484.399994	499.350006	477.000000	495.500000	495.500000	27553684

```
1 df.shape
```

```
(2434, 6)
```

```
1 df = df.reset_index()
```

```
1 # dropping features as they are highly correalted with the closing price
2 df = df.drop(['Open', 'High', 'Low', 'Adj Close' ], axis=1)
```

```
1 df.shape
```

```
(2434, 3)
```

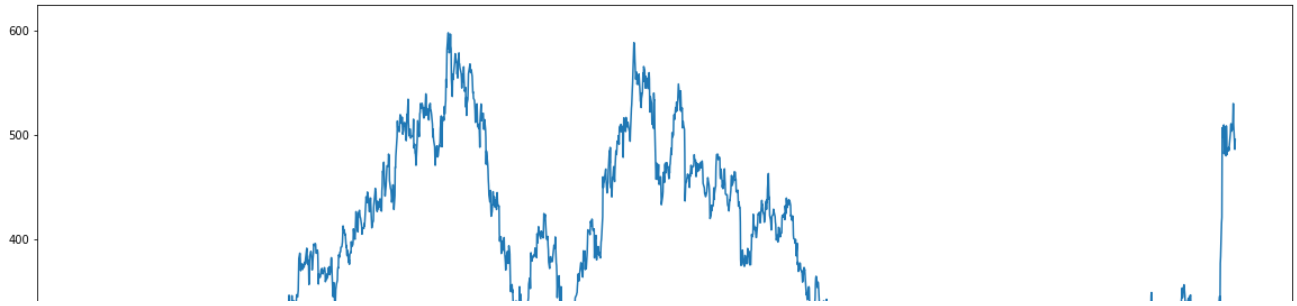
```
1 # checking for repeated rows
2 dates = df['Date']
3 df[dates.isin(dates[dates.duplicated()])].sort_index()
```

	Date	Close	Volume

```
1 df = df.drop_duplicates()
```

```
1 # plotting price movement
2 df['Date'] = pd.to_datetime(df.Date, format='%Y-%m-%d')
3 df.set_index('Date', inplace=True)
4  # plotting
5 pt.figure(figsize=(20,10))
6 pt.plot(df['Close'], label='Close Price movement')
```

```
[<matplotlib.lines.Line2D at 0x7f46bdf66c90>]
```



creating new features and understanding their nature

```
1 stock_price = df['Close'].to_frame()
2 stock_price['simple_MA_60'] = stock_price['Close'].rolling(60).mean()
3 stock_price['cumulative_MA'] = stock_price['Close'].expanding().mean()
4 stock_price['exponential_MA_60'] = stock_price['Close'].ewm(span=60).mean()
5 stock_price.dropna(inplace=True)
```

plots for Mean Averages for a roll back period for 60 days

```
1 stock_price[['Close', 'simple_MA_60','cumulative_MA','exponential_MA_60']].plot(label='
2
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46bde72610>
```

the exponential MA provides major thresholds and reverting points while simple and cumulative MA traces the price

adding features-

- Moving Averages (MA)
- Bollinger Bands (region enclosing +- std about MA)

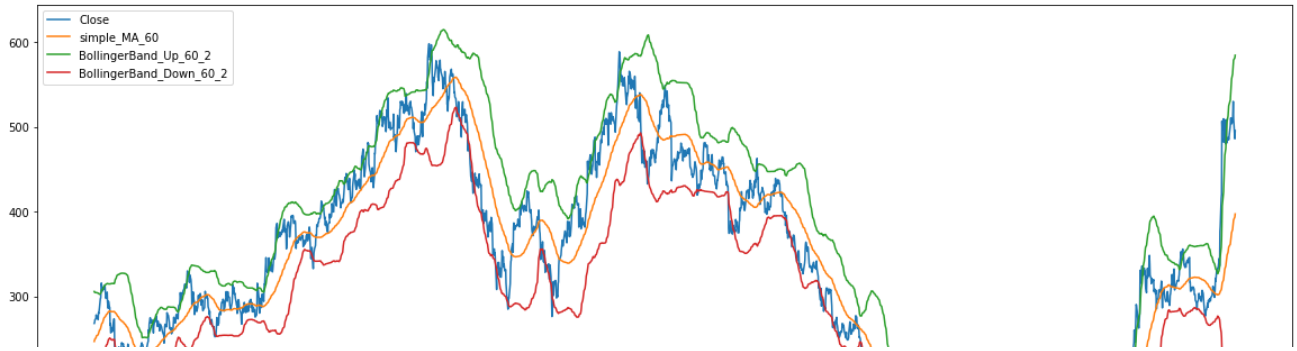
```
df['simple_MA_60'] = df['Close'].rolling(60).mean()
```

```
df['BollingerBand_Up_60_1'] = df['Close'].rolling(60).mean() + df['Close'].rolling(60).
df['BollingerBand_Down_60_1'] = df['Close'].rolling(60).mean() - df['Close'].rolling(60
df['BollingerBand_Up_60_2'] = df['Close'].rolling(60).mean() + 2*df['Close'].rolling(60
df['BollingerBand_Down_60_2'] = df['Close'].rolling(60).mean() - 2*df['Close'].rolling(
df['BollingerBand_Up_20_2'] = df['Close'].rolling(20).mean() + 2*df['Close'].rolling(20
df['BollingerBand_Down_20_2'] = df['Close'].rolling(20).mean() - 2*df['Close'].rolling(
df['BollingerBand_Up_20_1'] = df['Close'].rolling(20).mean() + df['Close'].rolling(20).
df['BollingerBand_Down_20_1'] = df['Close'].rolling(20).mean() - df['Close'].rolling(20
df['BollingerBand_Up_10_1'] = df['Close'].rolling(10).mean() + df['Close'].rolling(10).
df['BollingerBand_Down_10_1'] = df['Close'].rolling(10).mean() - df['Close'].rolling(10
df['BollingerBand_Up_10_2'] = df['Close'].rolling(10).mean() + 2*df['Close'].rolling(10
df['BollingerBand_Down_10_2'] = df['Close'].rolling(10).mean() - 2*df['Close'].rolling(
df = df.dropna()
```

Bollinger Bands Visualisation

```
df[['Close', 'simple_MA_60', 'BollingerBand_Up_60_2', 'BollingerBand_Down_60_2']].plot(
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46bde74750>
```




```
1 display(df)
```

Date	Close	Volume	simple_MA_60	BollingerBand_Up_60_1	BollingerBand_Down_
2012-03-28	268.272522	7597363	247.003970	276.459411	217.54
2012-03-29	269.064026	9912451	248.455072	276.776553	220.13
2012-03-30	272.329010	6411121	249.795693	277.285905	222.30
2012-04-02	273.862549	8010959	251.042322	277.881245	224.20
2012-04-03	278.265320	8305886	252.347489	278.556667	226.13
...	...	...	...	...	
2021-11-16	519.049988	55897781	382.536667	475.340635	289.73
2021-11-17	530.150024	48463415	386.483334	480.457986	292.50
2021-11-18	509.700012	47197742	390.260001	484.562195	295.95
2021-11-22	486.100006	40044849	393.745834	487.653327	299.83
2021-11-23	495.500000	27553684	397.340835	490.940525	303.74

2375 rows × 15 columns

Analyzing different Models

without using added features and history

```
1 data_without_history = df['Close'].to_frame()
2 display(data_without_history)
```

Date	Close
2012-03-28	268.272522
2012-03-29	269.064026
2012-03-30	272.329010
2012-04-02	273.862549
2012-04-03	278.265320
...	...
2021-11-16	519.049988
2021-11-17	530.150024
2021-11-18	509.700012
2021-11-22	486.100006
2021-11-23	495.500000

2375 rows × 1 columns

```
1 data_without_history = data_without_history.reset_index()
2 display(data_without_history)
```

	Date	Close
0	2012-03-28	268.272522
1	2012-03-29	269.064026
2	2012-03-30	272.329010
3	2012-04-02	273.862549
4	2012-04-03	278.265320
...	...	...
2370	2021-11-16	519.049988
2371	2021-11-17	530.150024
2372	2021-11-18	509.700012
2373	2021-11-22	486.100006
2374	2021-11-23	495.500000

2375 rows × 2 columns

```
1 date = data_without_history['Date'].to_frame()
```

adding date related features

```
1 add_datepart(data_without_history, 'Date')
2 data_without_history.drop('Elapsed', axis=1, inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/fastai/tabular/transform.py:63: FutureWarning
  for n in attr: df[prefix + n] = getattr(field.dt, n.lower())
```

creating new feature which will represent first and the last day of the week

```
1 data_without_history['mon_fri'] = 0
2 for i in range(0,len(data)):
3     if (data_without_history['Dayofweek'][i] == 0 or data_without_history['Dayofweek'][
4         data_without_history['mon_fri'][i] = 1
5     else:
6         data_without_history['mon_fri'][i] = 0
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  after removing the cwd from sys.path.
```

```
1 new_data = pd.concat((data_without_history,date),axis=1)
2 display(new_data)
```

	Close	Year	Month	Week	Day	Dayofweek	Dayofyear	Is_month_end	Is_mont
0	268.272522	2012	3	13	28	2	88	False	

```
train_size = int(0.7*new_data.shape[0])
test_size = int(0.3*new_data.shape[0])
print(train_size)
print(test_size)
```

```
1662
712
```

4	278.265320	2012	4	14	3	1	94	False	

```
train_data = new_data[:train_size]
train_data = train_data.drop(['Date'], axis=1)
test_data = new_data[train_size:]
test_data = test_data.drop(['Date'], axis=1)
X_train = train_data.drop(['Close'], axis=1)
Y_train = train_data['Close']
X_test = test_data.drop(['Close'], axis = 1)
Y_test = test_data['Close']
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(1662, 13)
(1662,)
(713, 13)
(713,)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,Y_train)
```

```
LinearRegression()
```

```
price_prediction_wh = model.predict(X_test)
rms=np.sqrt(np.mean(np.power((np.array(Y_test)-np.array(price_prediction_wh)),2)))
print(rms)
print(price_prediction_wh)
```

```
246.60354869944692
[394.563913 441.737074 446.614788 439.556011 ... 430.357968 430.838367 427.471709 429
```

```
new_data.set_index('Date', inplace=True)
```

Visualizing Prediction

```
test_data['Predictions'] = 0
test_data['Predictions'] = price_prediction_wh

test_data.index = new_data[train_size:].index
train_data.index = new_data[:train_size].index

pt.figure(figsize=(20,10))
pt.plot(train_data['Close'])
pt.plot(test_data[['Close', 'Predictions']])
pt.legend(['Train_close', 'Actual_close', 'Predicted_close'])
```

```
<matplotlib.legend.Legend at 0x7f46bac32c50>
```

KNN

```
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train_scaled)
X_test_scaled = scaler.fit_transform(X_test)
X_test = pd.DataFrame(X_test_scaled)

#using gridsearch to find the best parameter
params = {'n_neighbors':[4,5,6,7,8,9,10,11]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)

#fit the model and make predictions
model.fit(X_train,Y_train)
price_prediction_KNN_wh = model.predict(X_test)
```

```
rms_KNN=np.sqrt(np.mean(np.power((np.array(Y_test)-np.array(price_prediction_KNN)),2)))
rms
```

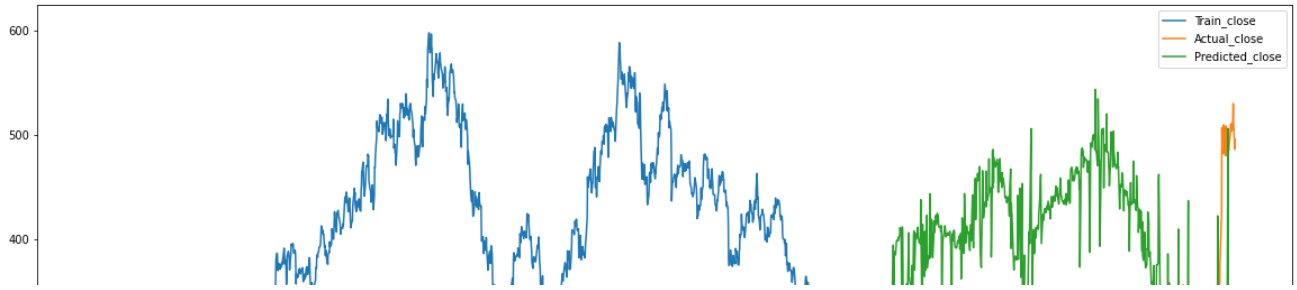
```
246.60354869944692
```

```
test_data['Predictions'] = 0
test_data['Predictions'] = price_prediction_KNN_wh

test_data.index = new_data[train_size:].index
train_data.index = new_data[:train_size].index

pt.figure(figsize=(20,10))
pt.plot(train_data['Close'])
pt.plot(test_data[['Close', 'Predictions']])
pt.legend(['Train_close', 'Actual_close', 'Predicted_close'])
```

```
<matplotlib.legend.Legend at 0x7f46bab737d0>
```



using features for past/ historical data

creating new historical features by shifting tha rows by 20, 40 and 60 days

```
data = df['Close'].to_frame()
for lag in [20,40,60]:
    shift = lag
    shifted = df.shift(shift)
    shifted.columns = [str.format("%s_shifted_by_%d" % (column ,shift)) for column in s
    data = pd.concat((data,shifted),axis=1)
```

```
data = data.dropna()
display(data)
```

Date	Close	Close_shifted_by_20	Volume_shifted_by_20	simple_MA_60_shifted_by
2012-06-25	243.587280	273.664673	8186826.0	282.237

```
1 data = data.reset_index()
```

```
1 data.shape
```

```
(2315, 47)
```

06-28

```
1 # data = data.reset_index()
2 # data['Date'] = pd.to_datetime(data.Date,format='%Y-%m-%d')
3 # data.index = data['Date']
4 # data = data.sort_index(ascending=True, axis=0)
5 # new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])
6 # for i in range(0,len(data)):
7 #      new_data['Date'][i] = data['Date'][i]
8 #      new_data['Close'][i] = data['Close'][i]
```

```
1 display(data)
```

	Date	Close	Close_shifted_by_20	Volume_shifted_by_20	simple_MA_60_shif
0	2012-	243.587280	273.664673	8186826.0	

```
1 date = data['Date'].to_frame()
```

1	2012-	244.081970	272.823700	12691931.0	

```
1
2 add_datepart(data, 'Date')
3 data.drop('Elapsed', axis=1, inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/fastai/tabular/transform.py:63: FutureWarning
  for n in attr: df[prefix + n] = getattr(field.dt, n.lower())
```

06-29

```
1 display(data)
```

	Close	Close_shifted_by_20	Volume_shifted_by_20	simple_MA_60_shifted_by_
0	243.587280	273.664673	8186826.0	282.237
1	244.081970	272.823700	12691931.0	282.316
2	236.661575	240.470718	40931062.0	281.918
3	237.799362	230.527359	37837266.0	281.347
4	239.877075	222.167038	24109675.0	280.529
...	...	...	...	
2310	519.049988	497.600006	103630901.0	316.090
2311	530.150024	509.600006	72322623.0	319.547
2312	509.700012	481.899994	57428637.0	322.536
2313	486.100006	486.899994	55444814.0	325.725
2314	495.500000	508.000000	52608672.0	329.306

2315 rows × 58 columns

Introducing features for Date

```
1 data['mon_fri'] = 0
2 for i in range(0,len(data)):
3     if (data['Dayofweek'][i] == 0 or data['Dayofweek'][i] == 4):
4         data['mon_fri'][i] = 1
5     else:
6         data['mon_fri'][i] = 0
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
```

```
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
```

```
1 display(data)
```

	Close	Close_shifted_by_20	Volume_shifted_by_20	simple_MA_60_shifted_by_
0	243.587280	273.664673	8186826.0	282.237
1	244.081970	272.823700	12691931.0	282.316
2	236.661575	240.470718	40931062.0	281.918
3	237.799362	230.527359	37837266.0	281.347
4	239.877075	222.167038	24109675.0	280.529
...	...	...	...	
2310	519.049988	497.600006	103630901.0	316.090
2311	530.150024	509.600006	72322623.0	319.547
2312	509.700012	481.899994	57428637.0	322.536
2313	486.100006	486.899994	55444814.0	325.725
2314	495.500000	508.000000	52608672.0	329.306

2315 rows × 59 columns

```
1 new_data = pd.concat((data,date),axis=1)
2 new_data
3 display(new_data)
```

	Close	Close_shifted_by_20	Volume_shifted_by_20	simple_MA_60_shifted_by
0	243.587280	273.664673	8186826.0	282.237
1	244.081970	272.823700	12691931.0	282.316
2	236.661575	240.470718	40931062.0	281.918
3	237.799362	230.527359	37837266.0	281.347
4	239.877075	222.167038	24109675.0	280.529

...

```
train_size = int(0.7*new_data.shape[0])
test_size = int(0.3*new_data.shape[0])
print(train_size)
print(test_size)
```

```
1620
694
```

Linear Regression and KNN will be able to predict data for 20 days in future atmost

```
train_data = new_data[:train_size]
train_data = train_data.drop(['Date'], axis=1)
test_data = new_data[train_size:]
test_data = test_data.drop(['Date'], axis=1)
X_train = train_data.drop(['Close'], axis=1)
Y_train = train_data['Close']
X_test = test_data.drop(['Close'], axis = 1)
Y_test = test_data['Close']
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(1620, 58)
(1620,)
(695, 58)
(695,)
```

Linear Regression

```
model = LinearRegression()
model.fit(X_train,Y_train)
```

```
LinearRegression()
```

```
price_prediction = model.predict(X_test)
rms=np.sqrt(np.mean(np.power((np.array(Y_test)-np.array(price_prediction)),2)))
print(rms)
print(type(price_prediction))
print(price_prediction)
```

```
63.89653090586305
<class 'numpy.ndarray'>
[163.666481 161.637592 162.261637 163.2093   ... 539.53674  522.08177  530.489868 539
```

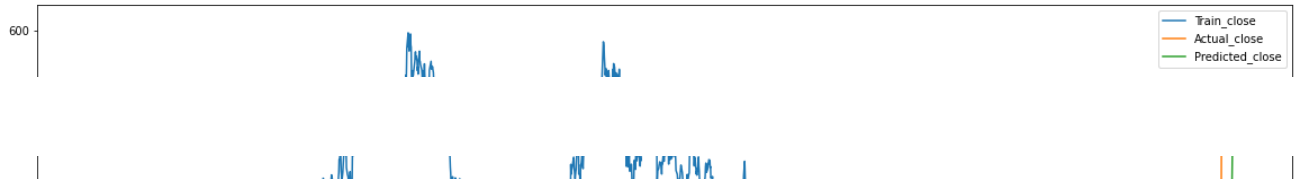
```
new_data.set_index('Date', inplace=True)
```

```
#new_data.set_index('Date', inplace=True)
test_data['Predictions'] = 0
test_data['Predictions'] = price_prediction

test_data.index = new_data[train_size:].index
train_data.index = new_data[:train_size].index

pt.figure(figsize=(20,10))
pt.plot(train_data['Close'])
pt.plot(test_data[['Close', 'Predictions']])
pt.legend(['Train_close', 'Actual_close', 'Predicted_close'])
```

```
<matplotlib.legend.Legend at 0x7f46bab85450>
```




```
1
```


KNN

```
# from sklearn import neighbors
# from sklearn.model_selection import GridSearchCV
# from sklearn.preprocessing import MinMaxScaler
# scaler = MinMaxScaler(feature_range=(0, 1))
```


```
X_train_scaled = scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train_scaled)
X_test_scaled = scaler.fit_transform(X_test)
X_test = pd.DataFrame(X_test_scaled)

#using gridsearch to find the best parameter
params = {'n_neighbors':[4,5,6,7,8,9,10,11]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)

#fit the model and make predictions
model.fit(X_train,Y_train)
price_prediction_KNN = model.predict(X_test)
```

```
rms_KNN=np.sqrt(np.mean(np.power((np.array(Y_test)-np.array(price_prediction_KNN)),2)))
rms
```

```
63.89653090586305
```

```
test_data['Predictions'] = 0
test_data['Predictions'] = price_prediction_KNN

test_data.index = new_data[train_size:].index
train_data.index = new_data[:train_size].index

pt.figure(figsize=(20,10))
pt.plot(train_data['Close'])
pt.plot(test_data[['Close', 'Predictions']])
pt.legend(['Train_close', 'Actual_close', 'Predicted_close'])
```

```
<matplotlib.legend.Legend at 0x7f46ba9ce190>
```

KNN is only able to capture the basic flow of price as with increased dimensions the performance of KNN model decreases

```
1
```

Using models which take in account for historical prices but does not require seperate features

Auto-Arima

```
1 from pmdarima import auto_arima
```

```
1 data = df.sort_index(ascending=True, axis=0)
2 #display(data)
```

```
1 train_size = int(0.7*data.shape[0])
2 test_size = int(0.3*data.shape[0])
3 print(train_size)
4 print(test_size)
```

```
1662
712
```

```
train_data = data[:train_size]
test_data = data[train_size:]
# only CLose Price is provided as data no other features
train = train_data['Close']
test = test_data['Close']
model = auto_arima(train, start_p=5, start_q=5,max_p=25, max_q=15, m=12,start_P=0, seas
model.fit(train)

```

```
Performing stepwise search to minimize aic
 ARIMA(5,1,5)(0,1,1)[12]             : AIC=inf, Time=44.62 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=12809.178, Time=0.12 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=12309.961, Time=0.78 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=5.35 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=12808.648, Time=0.10 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=12174.142, Time=1.69 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=inf, Time=20.21 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=inf, Time=6.93 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=12173.264, Time=1.33 sec
 ARIMA(0,1,0)(1,1,0)[12]             : AIC=12309.009, Time=0.50 sec
 ARIMA(0,1,0)(2,1,1)[12]             : AIC=inf, Time=14.23 sec
 ARIMA(0,1,0)(1,1,1)[12]             : AIC=inf, Time=4.30 sec
 ARIMA(0,1,1)(2,1,0)[12]             : AIC=12174.014, Time=2.23 sec
 ARIMA(1,1,1)(2,1,0)[12]             : AIC=inf, Time=17.76 sec
 ARIMA(0,1,0)(2,1,0)[12] intercept   : AIC=12175.259, Time=4.30 sec

Best model:  ARIMA(0,1,0)(2,1,0)[12]
Total fit time: 124.465 seconds
ARIMA(order=(0, 1, 0), scoring_args={}, seasonal_order=(2, 1, 0, 12),
      suppress_warnings=True, with_intercept=False)
```

```
predictions = model.predict(n_periods=test_size+1)
prediction_arima = pd.DataFrame(predictions,index = test_data.index,columns=['Predictio
```

```
rms=np.sqrt(np.mean(np.power((np.array(test_data['Close'])-np.array(prediction_arima['P
rms
```

```
226.27290877544834
```

The model prdicted initial downfall correctly which provided a good bound

```
pt.figure(figsize=(20,10))
pt.plot(train_data['Close'])
pt.plot(test_data['Close'])
pt.plot(prediction_arima['Prediction'])
pt.legend(['Train_close', 'Actual_close', 'Predicted_close'])
```

```
<matplotlib.legend.Legend at 0x7f46baa99390>
```

LSTM

The LSTM model is Recurrent Neural Network which considers the data from past and works through a loop. Neural Network induces non-linearity.

```
1 from keras.models import Sequential
2 from keras.layers import Dense, Dropout, LSTM
```

```
1 data = df.sort_index(ascending=True, axis=0)
```

```
1 #display(data)
```

```
1 data=data.reset_index()
2 new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])
3 for i in range(0,len(data)):
4     new_data['Date'][i] = data['Date'][i]
5     new_data['Close'][i] = data['Close'][i]
6
7 new_data.index = new_data.Date
8 new_data.drop('Date', axis=1, inplace=True)
9 display(new_data)
```

Date	Close
2012-03-28	268.273
2012-03-29	269.064
2012-03-30	272.329
2012-04-02	273.863
2012-04-03	278.265
...	...
2021-11-16	519.05
2021-11-17	530.15
2021-11-18	509.7
2021-11-22	486.1
2021-11-23	495.5

```
train_size = int(0.7*new_data.shape[0])
test_size = int(0.3*new_data.shape[0])
print(train_size)
print(test_size)
```

```
1662
712
```

```
dataset = new_data.values
dataset
```

```
array([[268.27252197265625],
       [269.06402587890625],
       [272.3290100097656],
       [273.862548828125],
       ...,
       [530.1500244140625],
       [509.70001220703125],
       [486.1000061035156],
       [495.5]], dtype=object)
```

```
train = dataset[0:train_size,:]
test = dataset[train_size:,:]

#converting dataset into x_train and y_train
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

# Training intances are lagged by 100 days here thus providing the hitorical element
x_train, y_train = [], []
for i in range(100,len(train)):
    x_train.append(scaled_data[i-100:i,0])
```

```
    y_train.append(scaled_data[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)

x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(units=100,activation = 'tanh', return_sequences=True, input_shape=(x_tra
model.add(LSTM(units=100))
# model.add(LSTM(50, activation='tanh',))
# model.add(LSTM(units=50))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=1, batch_size=1, verbose=2)

#predicting 246 values, using past 60 from the train data
inputs = new_data[len(new_data) - len(test) - 100:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(100,inputs.shape[0]):
    X_test.append(inputs[i-100:i,0])
X_test = np.array(X_test)

X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
```

```
1562/1562 - 68s - loss: 0.0025 - 68s/epoch - 43ms/step
```

```
rms=np.sqrt(np.mean(np.power((test-closing_price),2)))
rms
```

```
17.04918868275994
```

```
train = new_data[:train_size]
test = new_data[train_size:]
test['Predictions'] = closing_price
pt.figure(figsize = (20,10))
pt.plot(train['Close'])
pt.plot(test[['Close','Predictions']])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  This is separate from the ipykernel package so we can avoid doing imports until
[<matplotlib.lines.Line2D at 0x7f46bf491d90>,
 <matplotlib.lines.Line2D at 0x7f46bf491750>]
```

1