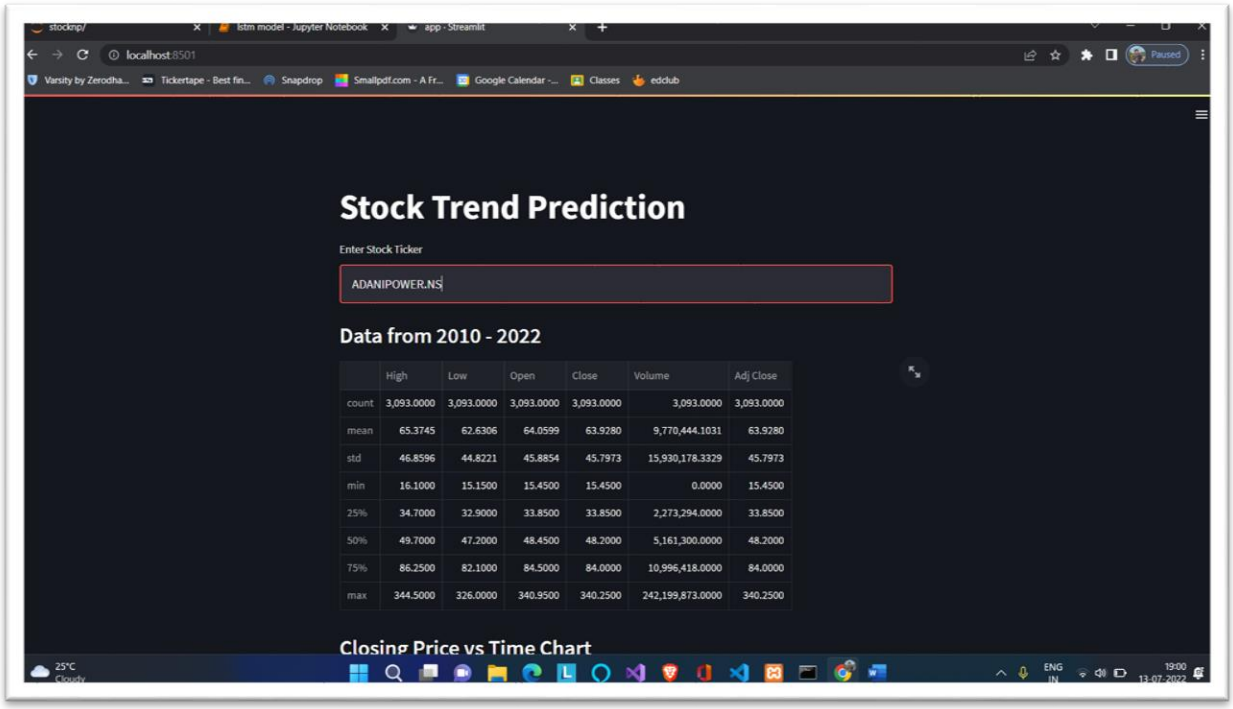
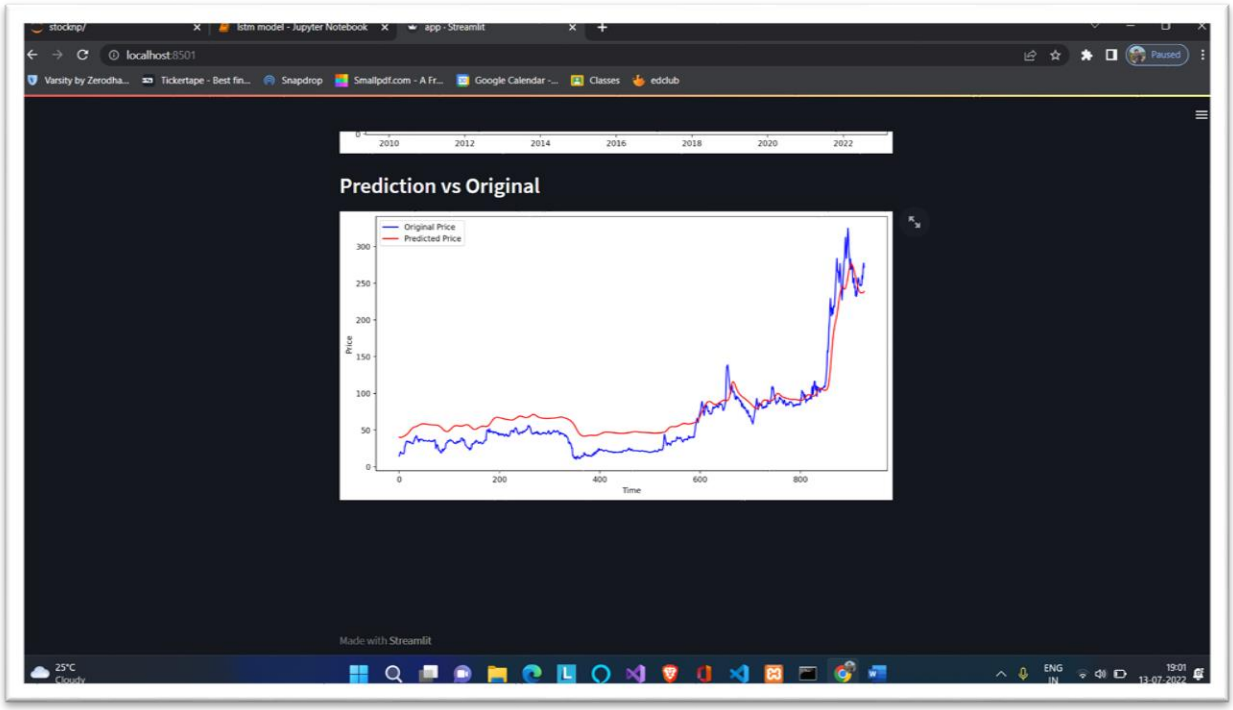
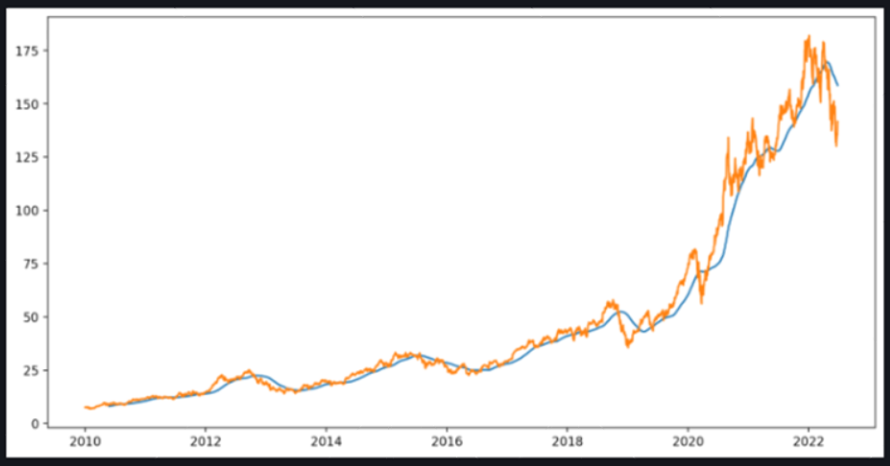
User Interface of the Project Stock Trend Prediction.



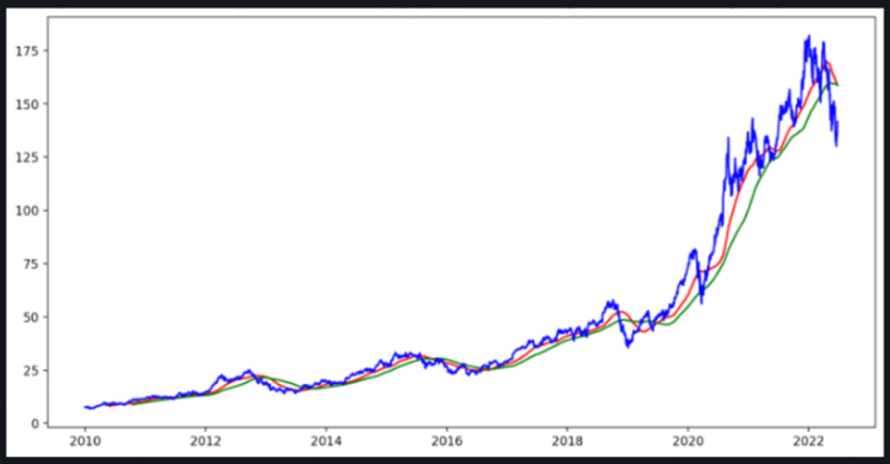
	High	Low	Open	Close	Volume	Adj Close
count	3,093.0000	3,093.0000	3,093.0000	3,093.0000	3,093.0000	3,093.0000
mean	65.3745	62.6306	64.0599	63.9280	9,770,444.1031	63.9280
std	46.8596	44.8221	45.8854	45.7973	15,930,178.3329	45.7973
min	16.1000	15.1500	15.4500	15.4500	0.0000	15.4500
25%	34.7000	32.9000	33.8500	33.8500	2,273,294.0000	33.8500
50%	49.7000	47.2000	48.4500	48.2000	5,161,300.0000	48.2000
75%	86.2500	82.1000	84.5000	84.0000	10,996,418.0000	84.0000
max	344.5000	326.0000	340.9500	340.2500	242,199,873.0000	340.2500



Closing Price vs Time Chart with 100MA

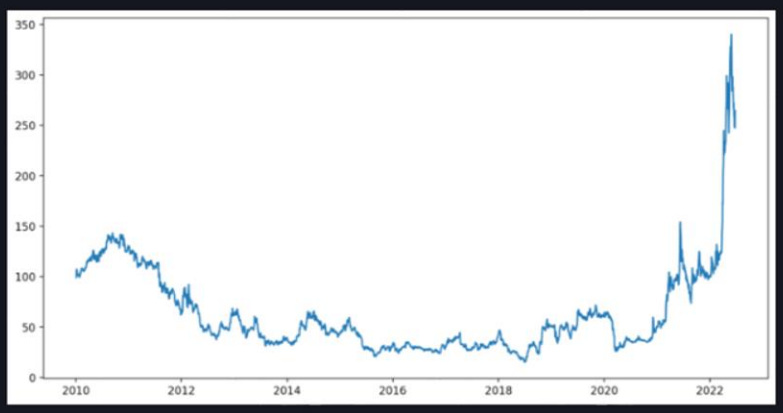


Closing Price vs Time Chart with 100MA & 200MA

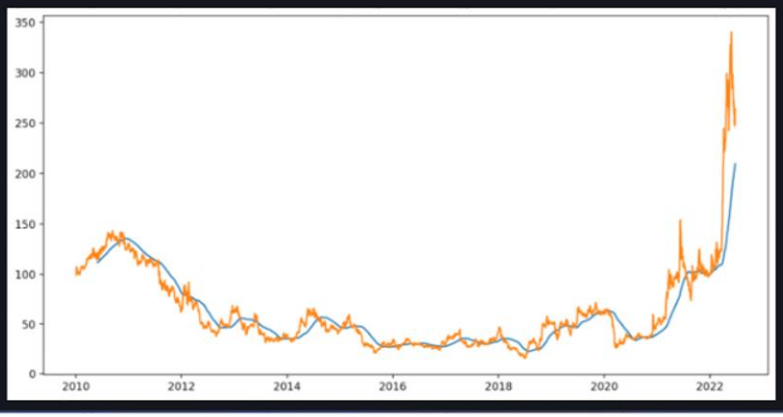


Prediction vs Original

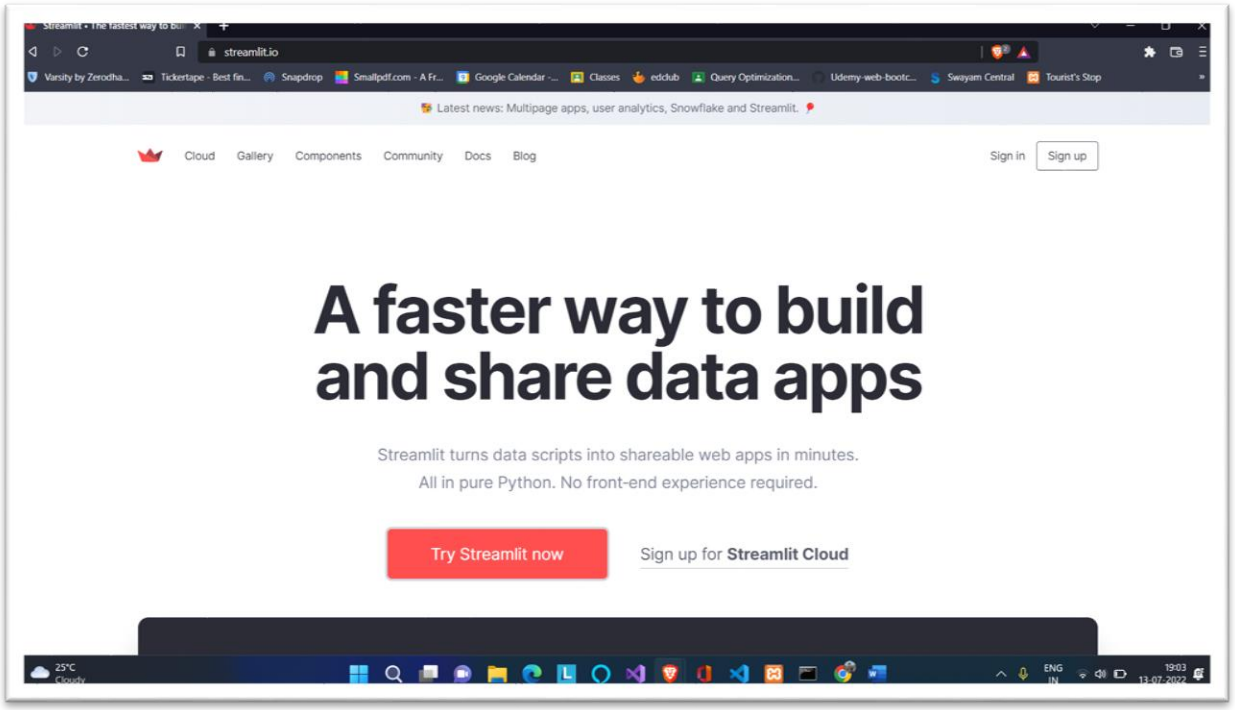
Closing Price vs Time Chart

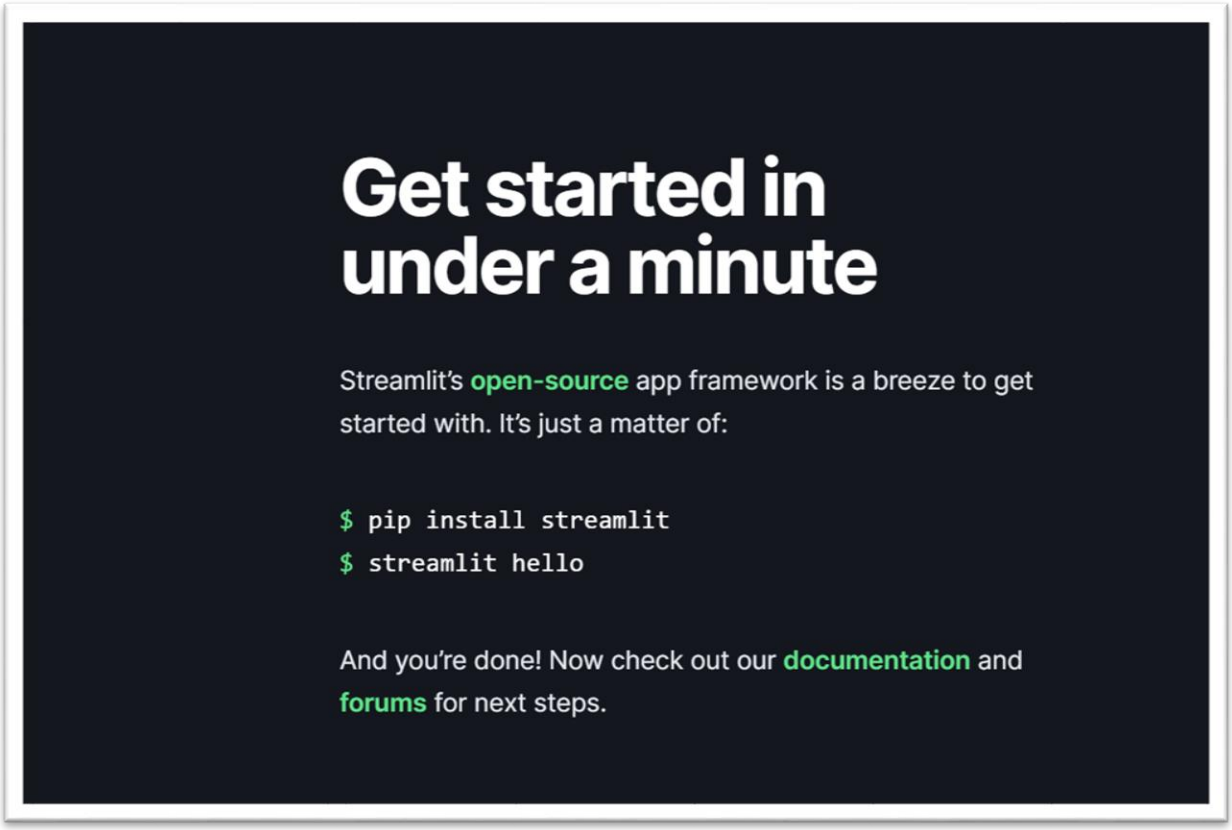


Closing Price vs Time Chart with 100MA



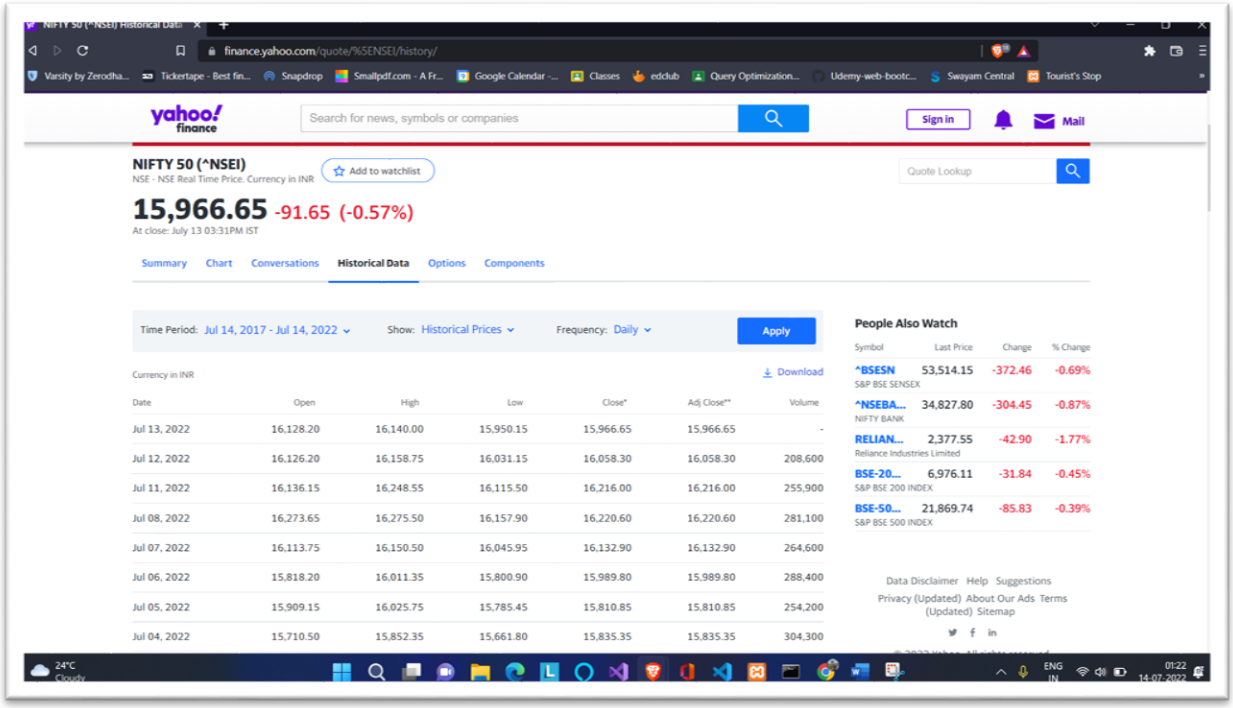
Used **Streamlit,** an open-source Python library that makes it easy to build beautiful custom web apps for Machine Learning and Data Science, It is compatible with major Python libraries such as scikit-learn, Keras, PyTorch, SymPy(latex), NumPy, pandas, Matplotlib etc.



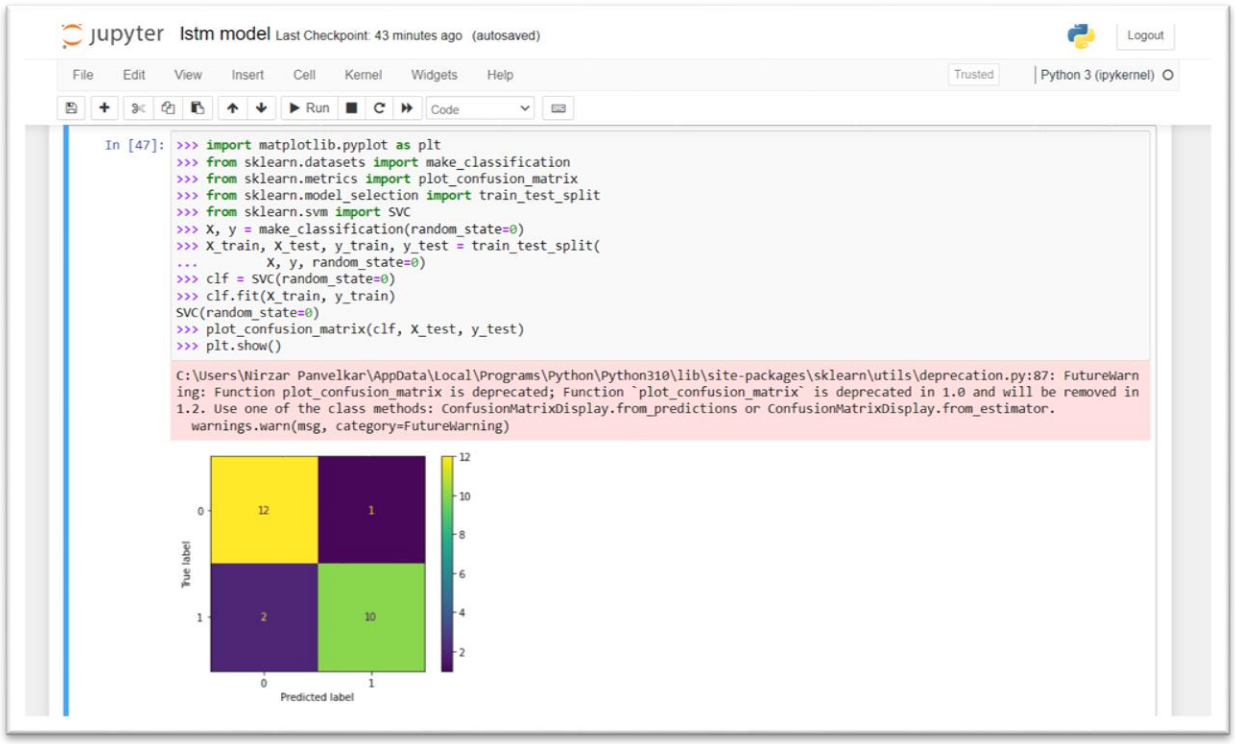


Dataset:

We have used Yahoo Finance for scraping the data into the project.



Confusion Matrix



KNN

In [139…
```
#importing libraries
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
```

In [140…
```
x_valid = x_test
y_valid = y_test
```

In [141…
```
#scaling data
x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)
x_valid_scaled = scaler.fit_transform(x_valid)
x_valid = pd.DataFrame(x_valid_scaled)
```

In [143…
```
#using gridsearch to find the best parameter
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)
```

In [144…
```
#fit the model and make predictions
model.fit(x_train,y_train)
preds = model.predict(x_valid)
```

In [145…
```
#rmse
rms=np.sqrt(np.mean(np.power((np.array(y_valid)-np.array(preds)),2)))
rms
```

Out[145]:
```
5.427252109207298
```

Prophet

```
#fit the model
model = Prophet()
model.fit(train)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this
<fbprophet.forecaster.Prophet at 0x7f1aff57a3c8>
```

```
#predictions
close_prices = model.make_future_dataframe(periods=len(valid))
forecast = model.predict(close_prices)
```

```
#rmse
forecast_valid = forecast['yhat'][train_len:]
rms=np.sqrt(np.mean(np.power((np.array(valid['y'])-np.array(forecast_valid)),2)))
rms
```

```
1.8844396476432457
```

Xg Boost

```
def xgb_predict(train, val):
    train = np.array(train)
    X, y = train[:, :-1], train[:, -1]
    model = XGBRegressor(objective="reg:squarederror", n_estimators=1000)
    model.fit(X, y)

    val = np.array(val).reshape(1, -1)
    pred = model.predict(val)
    return pred[0]
```

```
xgb_predict(train, test[0, 0])
```

```
23.835068
```

```
from sklearn.metrics import mean_squared_error


def validate(data, perc):
    predictions = []

    train, test = train_test_split(data, perc)

    history = [x for x in train]

    for i in range(len(test)):
        test_X, test_y = test[i, :-1], test[i, -1]

        pred = xgb_predict(history, test_X[0])
        predictions.append(pred)

        history.append(test[i])

    error = mean_squared_error(test[:, -1], predictions, squared=False)

    return error, test[:, -1], predictions
```

LSTM

```
# ML Model
```

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

```
model = Sequential()
model.add(LSTM(units = 50,activation = 'relu',return_sequences = True,
               input_shape =(x_train.shape[1],1)))
model.add(Dropout(0.2))




model.add(LSTM(units = 60,activation = 'relu',return_sequences = True))
model.add(Dropout(0.3))




model.add(LSTM(units = 80,activation = 'relu',return_sequences = True))
model.add(Dropout(0.4))



model.add(LSTM(units = 120,activation = 'relu'))
model.add(Dropout(0.5))


# dense layer
model.add(Dense(units = 1))
```