**Project Report:**

Following libraries are used in the project.  
import streamlit as st

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

import matplotlib.pyplot as plt

import matplotlib

from sklearn.model\_selection import train\_test\_split, GridSearchCV

import seaborn as sns

import pandas as pd

import warnings

import numpy as np

from sklearn.metrics import accuracy\_score

warnings.filterwarnings("ignore", category=FutureWarning)

st.set\_option('deprecation.showPyplotGlobalUse', False)

First of all we created streamlit UI elements (Title, sidebar, checkbox, file uploader and a drop down for classifiers).

# Title

st.title("Stock Market Dashboard")

#sidebar

sideBar = st.sidebar

display = sideBar.checkbox('Display Dataset')

uploaded\_file = sideBar.file\_uploader("Upload the dataset")

classifier = sideBar.selectbox('Which Classifier do you want to use?',('SVM' , 'KNN' , 'Random Forest'))

if uploaded\_file is not None:

if uploaded\_file.name.endswith('.csv'):

df=pd.read\_csv(uploaded\_file)

if display:

st.dataframe(df)

if classifier == 'SVM':

SVM(df)

elif classifier == 'Random Forest':

random\_forest(df)

elif classifier == 'KNN':

KNN(df)

elif uploaded\_file.name.endswith('.xlsx'):

df = pd.read\_excel(uploaded\_file)

if display:

st.dataframe(df)

if classifier == 'SVM':

SVM(df)

elif classifier == 'Random Forest':

random\_forest(df)

elif classifier == 'KNN':

KNN(df)

else:

sideBar.write("Please upload csv or excel files only")

**Data Preparation:**

We updated the date column as an index like this

df.index = pd.to\_datetime(df['Date'])

# drop The original date column

df = df.drop(['Date'], axis='columns')

**Explanatory variables:**

These variables are used to predict the value response variable. The prediction variables are kept in X variable such as ‘Open – Close’ and ‘High – Low’. These can be understood as indicators based on which the algorithm will predict tomorrow’s trend.

# Create predictor variables

df['Open-Close'] = df.Open - df.Close

df['High-Low'] = df.High - df.Low

# Store all predictor variables in a variable X

X = df[['Open-Close', 'High-Low']]

**Target variables:**

The target variable is the outcome which the machine learning model will predict based on the explanatory variables. y is a target dataset.If tomorrow’s price is greater than today’s price then we will buy the particular Stock else we will have no position in the. We will store +1 for a buy signal and 0 for a no position in y.

# Target variables

y = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)

**Dataset splitting into Training and Testing:**

split\_percentage = 0.8

split = int(split\_percentage\*len(df))

# Train data set

X\_train = X[:split]

y\_train = y[:split]

# Test data set

X\_test = X[split:]

y\_test = y[split:]

**Model prediction and strategy implementation:**

We have used the following three classifiers for prediction.

1. **SVM**

# Support vector classifier

cls = SVC().fit(X\_train, y\_train)

1. **KNN**

knn = KNeighborsClassifier()

# Create a dictionary of all values we want to test for n\_neighbors

params\_knn = {'n\_neighbors': np.arange(1, 25)}

# Use gridsearch to test all values for n\_neighbors

knn\_gs = GridSearchCV(knn, params\_knn, cv=5)

# Fit model to training data

knn\_gs.fit(X\_train, y\_train)

1. **Random Forest**

# Create a new random forest classifier

rf = RandomForestClassifier()

# Dictionary of all values we want to test for n\_estimators

params\_rf = {'n\_estimators': [110,130,140,150,160,180,200]}

# Use gridsearch to test all values for n\_estimators

rf\_gs = GridSearchCV(rf, params\_rf, cv=5)

# Fit model to training data

rf\_gs.fit(X\_train, y\_train)

**Accuracy Calculation for the above classifiers:**

from sklearn import metrics

#Calculating the accuracy

test\_score = round(rf\_gs.score(X\_test, y\_test), 2)

train\_score = round(rf\_gs.score(X\_train, y\_train), 2)

accuracy=test\_score

st.write("Accuracy: ", accuracy.round(2))

We predicted the (buy or sell) using predict() function. And then we calculated the daily returns and strategy returns. We also cacluated the cumulative returns and strategy returns.

df['Predicted\_Signal'] = cls.predict(X)

# Calculate daily returns

df['Return'] = df.Close.pct\_change()

# Calculate strategy returns

df['Strategy\_Return'] = df.Return \*df.Predicted\_Signal.shift(1)

# Calculate Cumulutive returns

df['Cum\_Ret'] = df['Return'].cumsum()

# Plot Strategy Cumulative returns

df['Cum\_Strategy'] = df['Strategy\_Return'].cumsum()

Then we plot the Cumulative return and strategy returns using streamlit line charts.

plt.style.use('seaborn-darkgrid')

df=df.dropna() # Dropping NaN rows

chart\_data = pd.DataFrame(

df,

columns=['Cum\_Ret', 'Cum\_Strategy'])

st.line\_chart(chart\_data)# Dsiplaying Line chart