**SAMPLE CODE**

**UserSide views.py**

from django.shortcuts import render

from django.conf import settings

# Create your views here.

from django.shortcuts import render,HttpResponse

from django.contrib import messages

from .forms import UserRegistrationForm

from .models import UserRegistrationModel

# Create your views here.

def UserRegisterActions(request):

if request.method == 'POST':

form = UserRegistrationForm(request.POST)

if form.is\_valid():

print('Data is Valid')

form.save()

messages.success(request, 'You have been successfully registered')

form = UserRegistrationForm()

return render(request, 'UserRegistrations.html', {'form': form})

else:

messages.success(request, 'Email or Mobile Already Existed')

print("Invalid form")

else:

form = UserRegistrationForm()

return render(request, 'UserRegistrations.html', {'form': form})

def UserLoginCheck(request):

if request.method == "POST":

loginid = request.POST.get('loginid')

pswd = request.POST.get('pswd')

print("Login ID = ", loginid, ' Password = ', pswd)

try:

check = UserRegistrationModel.objects.get(loginid=loginid, password=pswd)

status = check.status

print('Status is = ', status)

if status == "activated":

request.session['id'] = check.id

request.session['loggeduser'] = check.name

request.session['loginid'] = loginid

request.session['email'] = check.email

print("User id At", check.id, status)

return render(request, 'users/UserHomePage.html', {})

else:

messages.success(request, 'Your Account Not at activated')

return render(request, 'UserLogin.html')

except Exception as e:

print('Exception is ', str(e))

pass

messages.success(request, 'Invalid Login id and password')

return render(request, 'UserLogin.html', {})

def UserHome(request):

return render(request, 'users/UserHomePage.html', {})

def user\_machine\_learning(request):

from .utility import stock\_predictions

result = stock\_predictions.start\_process()

#result = result.to\_html

return render(request, 'users/ml\_results.html', {'results': result})

def user\_future\_prediction(request):

from .utility.FuturePredections import FuturePredImpl

obj = FuturePredImpl()

rslt = obj.startFuturePrediction()

import pandas as pd

rslt = pd.DataFrame(rslt)

rslt = rslt.to\_html

return render(request, 'users/futures.html', {'data': rslt})

def view\_dataset(request):

import pandas as pd

path = settings.MEDIA\_ROOT + "\\" + "AMZN.csv"

df = pd.read\_csv(path)

df = df.to\_html

return render(request,'users/view\_dataset.html', {'data':df})

**models.py**

from django.db import models

# Create your models here.

class UserRegistrationModel(models.Model):

name = models.CharField(max\_length=100)

loginid = models.CharField(unique=True, max\_length=100)

password = models.CharField(max\_length=100)

mobile = models.CharField(unique=True, max\_length=100)

email = models.CharField(unique=True, max\_length=100)

locality = models.CharField(max\_length=100)

address = models.CharField(max\_length=1000)

city = models.CharField(max\_length=100)

state = models.CharField(max\_length=100)

status = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.loginid

class Meta:

db\_table = 'UserRegistrations'

from django.db import models

# Create your models here.

**Forms.py**

from django import forms

from .models import UserRegistrationModel

class UserRegistrationForm(forms.ModelForm):

name = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[a-zA-Z]+'}), required=True, max\_length=100)

loginid = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[a-zA-Z]+'}), required=True, max\_length=100)

password = forms.CharField(widget=forms.PasswordInput(attrs={'pattern': '(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,}',

'title': 'Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters'}),

required=True, max\_length=100)

mobile = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[56789][0-9]{9}'}), required=True,

max\_length=100)

email = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,}$'}),

required=True, max\_length=100)

locality = forms.CharField(widget=forms.TextInput(), required=True, max\_length=100)

address = forms.CharField(widget=forms.Textarea(attrs={'rows': 4, 'cols': 22}), required=True, max\_length=250)

city = forms.CharField(widget=forms.TextInput(

attrs={'autocomplete': 'off', 'pattern': '[A-Za-z ]+', 'title': 'Enter Characters Only '}), required=True,

max\_length=100)

state = forms.CharField(widget=forms.TextInput(

attrs={'autocomplete': 'off', 'pattern': '[A-Za-z ]+', 'title': 'Enter Characters Only '}), required=True,

max\_length=100)

status = forms.CharField(widget=forms.HiddenInput(), initial='waiting', max\_length=100)

class Meta():

model = UserRegistrationModel

fields = '\_\_all\_\_'

stock\_predictions.py

import time

import numpy as np

import pandas as pd

import datetime

import os

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

import math

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import StandardScaler

import xgboost as xgb

from sklearn.metrics import accuracy\_score

from django.conf import settings

def parser(x):

return datetime.datetime.strptime(x,'%m/%d/%Y')

# FEATURE GENERATION

def get\_technical\_indicators(dataset): # function to generate feature technical indicators

# Create 7 and 21 days Moving Average

dataset['ma7'] = dataset['Close'].rolling(window=7).mean()

dataset['ma21'] = dataset['Close'].rolling(window=21).mean()

# Create MACD

dataset['26ema'] = dataset['Close'].ewm(span=26).mean()

dataset['12ema'] = dataset['Close'].ewm(span=12).mean()

dataset['MACD'] = (dataset['12ema'] - dataset['26ema'])

# Create Bollinger Bands

dataset['20sd'] = dataset['Close'].rolling(window=20).std()

dataset['upper\_band'] = (dataset['Close'].rolling(window=20).mean()) + (dataset['20sd'] \* 2)

dataset['lower\_band'] = (dataset['Close'].rolling(window=20).mean()) - (dataset['20sd'] \* 2)

# Create Exponential moving average

dataset['ema'] = dataset['Close'].ewm(com=0.5).mean()

# Create Momentum

dataset['momentum'] = (dataset['Close'] / 100) - 1

return dataset

def plot\_technical\_indicators(dataset, last\_days):

plt.figure(figsize=(16, 10), dpi=100)

shape\_0 = dataset.shape[0]

xmacd\_ = shape\_0 - last\_days

dataset = dataset.iloc[-last\_days:, :]

x\_ = range(3, dataset.shape[0])

x\_ = list(dataset.index)

# Plot first subplot

plt.subplot(2, 1, 1)

plt.plot(dataset['ma7'], label='MA 7', color='g', linestyle='--')

plt.plot(dataset['Close'], label='Closing Price', color='b')

plt.plot(dataset['ma21'], label='MA 21', color='r', linestyle='--')

plt.plot(dataset['upper\_band'], label='Upper Band', color='c')

plt.plot(dataset['lower\_band'], label='Lower Band', color='c')

plt.fill\_between(x\_, dataset['lower\_band'], dataset['upper\_band'], alpha=0.35)

plt.title('Technical indicators for Amazon - last {} days.'.format(last\_days))

plt.ylabel('USD')

plt.legend()

# Plot second subplot

plt.subplot(2, 1, 2)

plt.title('MACD')

plt.plot(dataset['MACD'], label='MACD', linestyle='-.')

plt.hlines(15, xmacd\_, shape\_0, colors='g', linestyles='--')

plt.hlines(-15, xmacd\_, shape\_0, colors='g', linestyles='--')

plt.plot(dataset['momentum'], label='Momentum', color='b', linestyle='-')

plt.legend()

plt.show()

def get\_fourier(dataset):

data\_FT = dataset[['Date', 'Close']]

close\_fft = np.fft.fft(np.asarray(data\_FT['Close'].tolist()))

close\_fft = np.fft.ifft(close\_fft)

close\_fft

fft\_df = pd.DataFrame({'fft':close\_fft})

fft\_df['absolute'] = fft\_df['fft'].apply(lambda x: np.abs(x))

fft\_df['angle'] = fft\_df['fft'].apply(lambda x: np.angle(x))

fft\_list = np.asarray(fft\_df['fft'].tolist())

fft\_list\_m10= np.copy(fft\_list); fft\_list\_m10[100:-100]=0

dataset['Fourier'] = pd.DataFrame(fft\_list\_m10).apply(lambda x: np.abs(x))

#dataset['absolute'] = dataset['Fourier'].apply(lambda x: np.abs(x))

return dataset

def get\_feature\_importance\_data(data\_income):

data = data\_income.copy()

y = data['Close']

X = data.iloc[:, 1:19]

train\_samples = int(X.shape[0] \* 0.65)

X\_train = X.iloc[:train\_samples]

X\_test = X.iloc[train\_samples:]

y\_train = y.iloc[:train\_samples]

y\_test = y.iloc[train\_samples:]

return (X\_train, y\_train), (X\_test, y\_test)

def start\_process():

import datetime

path = os.path.join(settings.MEDIA\_ROOT, 'AMZN.csv')

dataset\_ex\_df = pd.read\_csv(path, header=0, parse\_dates=[0], date\_parser=parser)

print(dataset\_ex\_df[['Date', 'Close']].head(3))

print('There are {} number of days in the dataset.'.format(dataset\_ex\_df.shape[0]))

plt.figure(figsize=(14, 5), dpi=100)

plt.plot(dataset\_ex\_df['Date'], dataset\_ex\_df['Close'], label='Amazon stock')

plt.vlines(datetime.date(2016, 4, 20), 0, 270, linestyles='--', colors='gray', label='Train/Test data cut-off')

plt.xlabel('Date')

plt.ylabel('USD')

plt.title('Figure 2: Amazon stock price')

plt.legend()

plt.show()

# FEATURE GENERATION

dataset\_TI\_df = get\_technical\_indicators(dataset\_ex\_df)

print(dataset\_TI\_df.head())

# Plot Technical

plot\_technical\_indicators(dataset\_TI\_df, 400)

data\_FT = dataset\_ex\_df[['Date', 'Close']]

close\_fft = np.fft.fft(np.asarray(data\_FT['Close'].tolist()))

fft\_df = pd.DataFrame({'fft': close\_fft})

fft\_df['absolute'] = fft\_df['fft'].apply(lambda x: np.abs(x))

fft\_df['angle'] = fft\_df['fft'].apply(lambda x: np.angle(x))

plt.figure(figsize=(14, 7), dpi=100)

fft\_list = np.asarray(fft\_df['fft'].tolist())

for num\_ in [3, 6, 9, 100]:

fft\_list\_m10 = np.copy(fft\_list);

fft\_list\_m10[num\_:-num\_] = 0

plt.plot(np.fft.ifft(fft\_list\_m10), label='Fourier transform with {} components'.format(num\_))

plt.plot(data\_FT['Close'], label='Real')

plt.xlabel('Days')

plt.ylabel('USD')

plt.title('Figure 3: Amazon (close) stock prices & Fourier transforms')

plt.legend()

plt.show()

dataset\_TI\_df = get\_fourier(dataset\_ex\_df)

print(dataset\_TI\_df.head(30))

from collections import deque

items = deque(np.asarray(fft\_df['absolute'].tolist()))

items.rotate(int(np.floor(len(fft\_df) / 2)))

plt.figure(figsize=(10, 7), dpi=80)

plt.stem(items)

plt.title('Figure 4: Components of Fourier transforms')

plt.show()

# Arima Model

from statsmodels.tsa.arima\_model import ARIMA

from pandas import DataFrame

from pandas import datetime

series = data\_FT['Close']

model = ARIMA(series, order=(5, 1, 0))

model\_fit = model.fit(disp=0)

print(model\_fit.summary())

from pandas.plotting import autocorrelation\_plot

autocorrelation\_plot(series)

plt.figure(figsize=(10, 7), dpi=80)

plt.show()

from pandas import read\_csv

from pandas import datetime

from pandas import DataFrame

from statsmodels.tsa.arima\_model import ARIMA

from sklearn.metrics import mean\_squared\_error

X = series.values

size = int(len(X) \* 0.66)

train, test = X[0:size], X[size:len(X)]

history = [x for x in train]

predictions = list()

for t in range(len(test)):

model = ARIMA(history, order=(5, 1, 0))

model\_fit = model.fit(disp=0)

output = model\_fit.forecast()

yhat = output[0]

#print(f'Future values {yhat} Data is {t}')

predictions.append(yhat)

obs = test[t]

history.append(obs)

dataset\_TI\_df['ARIMA'] = pd.DataFrame(predictions)

error = mean\_squared\_error(test, predictions)

rmse = math.sqrt(error)

print('Test MSE: %.3f' % error)

# Plot the predicted (from ARIMA) and real prices

# plt.figure(figsize=(12, 6), dpi=100)

# plt.plot(test, color='black', label='Real')

# plt.plot(predictions, color='yellow', label='Predicted')

# plt.xlabel('Days')

# plt.ylabel('USD')

# plt.title('Figure 5: ARIMA model on Amazon stock')

# plt.legend()

# plt.show()

print(dataset\_ex\_df.head(8))

print('Total dataset has {} samples, and {} features.'.format(dataset\_ex\_df.shape[0], dataset\_ex\_df.shape[1]))

# Get training and test data

(X\_train\_FI, y\_train\_FI), (X\_test\_FI, y\_test\_FI) = get\_feature\_importance\_data(dataset\_TI\_df)

regressor = xgb.XGBRegressor(gamma=0.0, n\_estimators=200, base\_score=0.7, colsample\_bytree=1, learning\_rate=0.05)

xgbModel = regressor.fit(X\_train\_FI, y\_train\_FI, \

eval\_set=[(X\_train\_FI, y\_train\_FI), (X\_test\_FI, y\_test\_FI)], \

verbose=False)

eval\_result = regressor.evals\_result()

training\_rounds = range(len(eval\_result['validation\_0']['rmse']))

plt.scatter(x=training\_rounds, y=eval\_result['validation\_0']['rmse'], label='Training Error')

plt.scatter(x=training\_rounds, y=eval\_result['validation\_1']['rmse'], label='Validation Error')

plt.xlabel('Iterations')

plt.ylabel('RMSE')

plt.title('Training Vs Validation Error')

plt.legend()

plt.show()

fig = plt.figure(figsize=(8, 8))

plt.xticks(rotation='vertical')

plt.bar([i for i in range(len(xgbModel.feature\_importances\_))], xgbModel.feature\_importances\_.tolist(),

tick\_label=X\_test\_FI.columns)

plt.title('Figure 6: Feature importance of the technical indicators.')

plt.show()

# LSTM

# 1. take dataframe and drop na

dataset\_lstm\_df = dataset\_TI\_df.drop(columns='Date')

dataset\_lstm\_df.head(7)#1. take dataframe and drop na

dataset\_lstm\_df = dataset\_TI\_df.drop(columns='Date')

print(dataset\_lstm\_df.head(7))

print('Total dataset has {} samples, and {} features.'.format(dataset\_lstm\_df.shape[0],dataset\_lstm\_df.shape[1]))

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

from keras.layers import Flatten

# creating test, train and validate trains

train, validate, test = np.split(dataset\_lstm\_df.sample(frac=1), [int(.6 \* len(dataset\_lstm\_df)), int(.8 \* len(dataset\_lstm\_df))])

open\_training = train.iloc[:, 1:2].values

# normalise

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0, 1))

open\_training = scaler.fit\_transform(open\_training)

# convert to right shape

features\_set\_1 = []

labels\_1 = []

for i in range(60, 450):

features\_set\_1.append(open\_training[i - 60:i, 0])

labels\_1.append(open\_training[i, 0])

# Code ref: https://github.com/LiamConnell/deep-algotrading

features\_set\_1, labels\_1 = np.array(features\_set\_1), np.array(labels\_1)

features\_set\_1 = np.reshape(features\_set\_1, (features\_set\_1.shape[0], features\_set\_1.shape[1], 1))

# training it

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(features\_set\_1.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(Flatten())

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=['mean\_absolute\_error'])

model.fit(features\_set\_1, labels\_1, epochs=100, batch\_size=32, validation\_data=(features\_set\_1, labels\_1))

# TESTING THE MODEL

open\_testing\_processed = test.iloc[:, 1:2].values

# convert test data to right format

open\_total = pd.concat((train['Open'], test['Open']), axis=0)

test\_inputs = open\_total[len(open\_total) - len(test) - 60:].values

# scaling data

test\_inputs = test\_inputs.reshape(-1, 1)

test\_inputs = scaler.transform(test\_inputs)

test\_features = []

for i in range(60, 151):

test\_features.append(test\_inputs[i - 60:i, 0])

test\_features = np.array(test\_features)

test\_features.shape

test\_features = np.reshape(test\_features, (test\_features.shape[0], test\_features.shape[1], 1))

rslt\_dict = {}

# make predictions

predictions = model.predict(test\_features)

predictions = scaler.inverse\_transform(predictions)

plt.figure(figsize=(10, 6))

plt.plot(open\_testing\_processed, color='pink', label='Actual Stock Price')

plt.plot(predictions, color='yellow', label='Predicted Stock Price')

rslt\_dict.update({'actual': open\_testing\_processed, 'predictions': predictions})

plt.title('Actual Value vs Predicted in Futures')

plt.xlabel('Date')

plt.ylabel('Predicted Price')

plt.legend()

plt.show()

rslt\_dict.update({'error': error})

rslt\_dict.update({'rmse': rmse})

#rslt\_dict = pd.DataFrame(rslt\_dict)

return rslt\_dict

Future Predictions

from django.conf import settings

import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.api as sm

import itertools

class FuturePredImpl:

df = ''

def \_\_init\_\_(self):

path = settings.MEDIA\_ROOT + "\\" + "AMZN.csv"

self.df = pd.read\_csv(path)

def startFuturePrediction(self):

import datetime

df = self.df[['Date', 'Close']]

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

#dp = pd.to\_datetime(df['Date'], format='%Y-%m-%d')

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

# print(dp.min(), dp.max())

df = df.groupby(dp)['Close'].sum().reset\_index()

df = df.set\_index('Date')

df.index

y = df['Close'].resample('MS').mean()

y['2018':]

import itertools

p = d = q = range(0, 2)

pdq = list(itertools.product(p, d, q))

seasonal\_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

# print('Examples of parameter combinations for Seasonal ARIMA...')

# print('SARIMAX: {} x {}'.format(pdq[1], seasonal\_pdq[1]))

# print('SARIMAX: {} x {}'.format(pdq[1], seasonal\_pdq[2]))

# print('SARIMAX: {} x {}'.format(pdq[2], seasonal\_pdq[3]))

# print('SARIMAX: {} x {}'.format(pdq[2], seasonal\_pdq[4]))

import statsmodels.api as sm

import itertools

for param in pdq:

for param\_seasonal in seasonal\_pdq:

try:

mod = sm.tsa.statespace.SARIMAX(y,

order=param,

seasonal\_order=param\_seasonal,

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

# print('ARIMA{}x{}12 - AIC:{}'.format(param, param\_seasonal, results.aic))

except Exception as ex:

print("Exception is ",str(ex))

continue

import statsmodels.api as sm

mod = sm.tsa.statespace.SARIMAX(y,

order=(1, 1, 1),

seasonal\_order=(1, 1, 0, 12),

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

# print(type(results))

pred\_uc = results.get\_forecast(steps=80)

pred\_ci = pred\_uc.conf\_int()

ax = y.plot(label='observed', figsize=(14, 7))

pred\_uc.predicted\_mean.plot(ax=ax, label='Future Forecast')

ax.fill\_between(pred\_ci.index,

pred\_ci.iloc[:, 0],

pred\_ci.iloc[:, 1], color='k', alpha=.5)

ax.set\_xlabel('Date')

ax.set\_ylabel('Price')

plt.legend()

plt.show()

return pred\_ci