Submitted To:

Mr. Sami Ullah

I191669 | I191660 | K

Zohaib Bashir | Ali Raza

MLOPS

Assignment 2

# Code Explainantion:

The Python script in StockPredictionA2.py defines a Flask web application with a machine learning model to predict the closing price of a stock. Here is a step-by-step breakdown of what the code is doing:

* Importing required libraries:
* The first lines of code import the required libraries such as Flask, Pandas, Scikit-Learn, NumPy, and Joblib. These libraries are used to create the Flask web application and build the machine learning model to predict stock prices.
* Loading the dataset:
* The next few lines of code load the stock price dataset from a CSV file named result.csv. It is read using the pandas library and is split into training and testing sets in the ratio of 80:20.
* Defining input and target variables:
* The input features for the machine learning model are defined as 'Open', 'High', 'Low', and 'Volume'. The target variable to be predicted is defined as 'Close'.
* Creating polynomial features of degree 2:
* In the next step, a PolynomialFeatures object is created with a degree of 2. This object is used to transform the input features into higher order polynomials, which is useful in capturing non-linear relationships between the input features and the target variable.
* Training the machine learning model:
* The LinearRegression model is trained on the transformed input features and target variable using the fit() method. The model is then used to predict the 'Close' values for the test data.
* Saving the trained model:
* The trained model is then saved using the joblib library, which is a utility for saving and loading Python objects.
* Defining the Flask routes:
* The Flask application is defined with three routes:
* '/' - the home page of the web application that shows a chart of the stock prices and predicted stock prices using the trained machine learning model.
* '/input' - this page shows a form where users can input the stock price features for which they want to predict the stock price.
* '/predict' - this page handles the prediction of stock prices. It takes the user input from the form and passes it to the trained model to predict the stock price.
* Implementing the routes:
  + The '/' route displays the chart of stock prices and predicted prices by rendering an HTML template with the chart data and predicted prices. The predicted prices are calculated using the trained model and are passed as arguments to the template.
  + The '/input' route displays an HTML template with a form where users can input the stock price features.
  + The '/predict' route takes the user input from the form and passes it to the trained model to predict the stock price. The predicted price is then rendered in an HTML template along with an output message.
  + Running the Flask application:
  + Finally, the application is run using the run() method, specifying the host and port number. If debug mode is enabled, Flask will print out detailed error messages and restart the application automatically if there are any code changes.

# Front End:

## Prediction.html:

The body of the HTML file contains a conditional statement that displays the predicted closed stock value if there is a prediction available, otherwise it displays an output message.

The curly braces with percentage signs surrounding the code is a Jinja template syntax which allows the embedding of dynamic content within the HTML template. In this case, the template is expecting two variables to be passed to it: prediction and output. If the prediction variable is zero or not available, then the output variable will be displayed. Otherwise, if the prediction variable is available, then the predicted closed stock value will be displayed.

## Index.html:

This file defines the layout and content of a web page for a stock exchange dashboard. The page displays a High Low Chart for a given set of data and provides some metrics related to the chart, including R-squared, Mean Squared Error, and Mean Absolute Error. The page also has a form with a "Next" button that allows the user to input data for prediction. The chart is created using the Highcharts library and the page is styled using Bootstrap CSS.

## Input.html:

This html file allows users to input data related to a stock and predict its closing price. The form has four input fields for the opening price of the stock, stock volume, maximum stock price, and minimum stock price. There is also a "Predict" button that, when clicked, will send the data to a server-side script for processing and prediction. The form uses CSS styling to make it look visually appealing and user-friendly.

# Setup:

JENKINS

For this assignment we utilized the cloud variant of Jenkins using AWS (Amazon Web Services). Using the cloud version we were able to run our jobs and our application from everywhere if the Amazon ec2 instance was up and running. Due to the free tier of AWS, we have to change webhook of GitHub every time as the public IP is not static. For AWS we used windows for that we needed PuTTy, the list of commands for putty are as follows

*sudo yum update*

*sudo wget -O /etc/yum.repos.d/jenkins.repo \*

*https://pkg.jenkins.io/redhat-stable/jenkins.repo*

*sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key*

*sudo yum upgrade*

*sudo amazon-linux-extras install java-openjdk11*

*sudo yum install jenkins*

*sudo systemctl enable jenkins*

*sudo systemctl start jenkins*

*sudo cat /var/lib/jenkins/secrets/initialAdminPassword*

*9c409b7a27c74c6e95c60f43ed222432*

*sudo yum install git*

*sudo yum install docker*

*sudo usermod -G docker -a jenkins*

*sudo systemctl start docker*

*sudo systemctl enable docker*

*sudo reboot*

What we did in Jenkins is we created a job that occurs/tells us whenever a push happens it will notify us. Whenever a commit is made in GitHub it is reflected on Jenkins due to the webhook.