

# Cloud-based Machine Learning Prediction of Stock Price Website

by

Leung Wing Hin, Leo

**Submitted in partial fulfillment of the requirements for the degree of**

**Bachelor of Science (Honours) in Computer Science**

**Hong Kong Baptist University**

**April, 2022**

## **Declaration**

I hereby declare that all the work done in this Final Year Project is of my independent effort. I also certify that I have never submitted the idea and product of this Final Year Project for academic or employment credits.

---

Leung Wing Hin, Leo

**Acceptance Page**  
Hong Kong Baptist University  
Computer Science Department

We hereby recommend that the Final Year Project submitted by Leung Wing Hin, Leo entitled " Cloud-based Machine Learning Prediction of Stock Price Website" be accepted in partial fulfillment of the requirements for the degree of Bachelor of Science (Honours) in Computer Science.

\_\_\_\_\_  
Dr. Han, Bo Prof.  
Supervisor

\_\_\_\_\_  
Prof. LEUNG, Yiu Wing  
Observer

Date:\_\_\_\_\_

Date:\_\_\_\_\_

## Contents

Declaration .....	2
Acceptance Page.....	3
.....	3
Chapter 1. Introduction.....	5
1.    Background .....	5
2.    Linear regression Model.....	6
3.    MLP neural network model.....	6
4.    Objective .....	7
Chapter 2 Application Overview .....	8
1.    System Architecture .....	8
1.        Website architecture.....	8
2.        Cloud Architecture.....	15
Chapter 3 Method and its Implementation .....	19
1.    Implement a linear regression model.....	19
2.    Implement a MLP neural network model.....	24
3.    Cloud system infrastructure as code.....	27
4.    Website Implementation.....	33
Chapter 5Performance .....	47
1. Evaluate model performance .....	47
Chapter 6 Discussions .....	49
1. Improvement and Limitation .....	49
Limitation .....	49
Improvement.....	50
Chapter 7 Conclusion .....	51
References .....	51
APPENDICES.....	54
1.    System Deployment Guide.....	54

## Chapter 1. Introduction

### 1. Background

Machine learning is an application of AI, that made the systems can learn and improve from experience without being explicitly programmed (*What is machine learning? A definition. 2022*). It focusses on developing program to access data and learn by itself (*What is machine learning? A definition. 2022*). In today world machine learning is widely use in different field, for example in medical field it is use for classification to determine and label disease for medical use (*Machine learning in healthcare: 12 real-world use cases – nix united 2021*). In this final year project, we will focus on using machine learning to predict stock market.

Predicting stock market with machine learning is very popular task among machine learning, it is also one of the most challenging tasks due to stock market can be affected by different situation for example political situation, country economy and natural disaster. However, because of machine learning method is now more advance and data are more available now, we can still use machine learning to predict stock market price or trend to give us some insight. Variation of markets are using various techniques such as ML, Deep Learning, Neural Networks, AI to analyses and predict stock price (*Soni et al., 2022*). Although prediction stock price from machine learning maybe not accurate, but it can enable investor to properly anticipate situation of the stock base on past and present data (*Soni et al., 2022*). For example, give them a direction to make decision to prevent money loss and improve profits (*Soni et al., 2022*).

Therefore, in the following section, I will introduce two machine learning method I will use in attempt to try to predict stock price which are linear regression and multilayer perceptron (*MLP*) neural network to predict stock price.

## 2. Linear regression Model

Because stock data is a form of time series data (*Soni et al., 2022*). It can be using a method call time series analysis to conduct analysis of data. Time series Analysis method depends on forecast and projection of the discrete time data (*Soni et al., 2022*). And linear regression is suitable for this situation. Linear Regression is trying to model relationship between two variables by fitting a linear equation on the data (*Linear Regression*). For example, in stock data we can have technical indication for example simple moving average and relative strength index and the relationship with the stock price. One variable will be explanatory variable and other one is a dependent variable (*Linear Regression*). There are some prerequisites to use the model, first it needs to determine is the data have the relation between the variables of interest (*Linear Regression*). In which stock data is in this situation. Therefore, we can use linear regression in this use case. The linear regression equation is  $Y=a+bX$ , X is the explanatory variable and Y is the dependent variable(*Linear Regression*).

To use the linear regression model first we have to define the dependent variable and independent variable. Than plug into the training with linear regression method. After the model is finished, test the model on test data and use scatter plot to determine the strength of the relationship between two variables (*Linear Regression*). Also, we can use correlation coefficient which is value between 1 and -1 indicate strength of the two variables in the data (*Linear Regression*).

## 3. MLP neural network model

MLP full name is multi-layer perceptron it is feed forward neural network. It is a deep learning method, which use artificial neural network as their structure (*Techopedia, 2017*). It has several layers of input nodes connected as a directed graph between input and output layer (*Techopedia, 2017*). The working principal of multi-layer perceptron, the input layer receives the input signal that need to process, then task such as prediction and classification will be performed by outer layer

(Abirami & Chitra, 2019). MLP is designed for solve problems that are not linearly separable, majority of the use case are pattern classification, recognition, prediction and approximation (Abirami & Chitra, 2019). The Fig. 1. show below is a representation of a multi-layer perceptron with a single hidden layer (Abirami & Chitra, 2019).

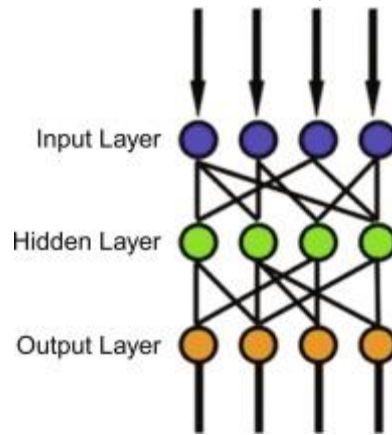


Fig. 3. representation of a multi-layer perceptron with a single hidden layer (Abirami & Chitra, 2019).

#### 4. Objective

In this final year project, the objective will be using linear regression and multi-layer perceptron to explore the possibility of predicting stock price. As I previously discussed in the above paragraph, although prediction of stock price can be affected by multiple things for example company performance, economy, and policy. It can provide a expectation for the user for the following trend to better plan for their trading and minimize the loss. The whole website will have three functions, first it will get stock price data from yahoo finance, then it will process the data to filter out row that are empty and create features and finally it will train the model. The website will be deployed on AWS cloud as the base infrastructure. AWS is a cloud service provider that offer 200 services from data center around the world (*What is AWS* ). Their services include computing, machine learning and more (*What is AWS* ).

## Chapter 2 Application Overview

### 1. System Architecture

#### 1. Website architecture

In the system architecture first, we will talk about the website architecture first. The website will have three functions first it will gather the stock data that the user enters, then the function will download data from yahoo finance and store it as csv. After the website gather data, the next function will be processing the data. Including create feature for predicting and clear row that have zero value. After process the data, the data will send to the model for prediction and the result will display to the function. Also, the website will be have login and logout function to let the user create their own account to store their prediction record.

Here are all the function requirements:

Req.ID	Requirement Title	Target Users	Priority
REQ-1	Login	All users	Must
REQ-2	Logout	All users	Must
REQ-3	Create account	All users	Must
REQ-4	Get data from yahoo finance	All users	Must
REQ-5	Predict stock data using MLP model	All users	Must
REQ-6	Predict stock data using linear regression model	Consumer	Must

Here are all the requirement Description

Item	Description
Requirement ID	REQ-1
Requirement Title	Login



Priority	Must
Functional Requirement Description	The function can authenticate username and password correctly and let user login. The function can alert user if they enter incorrectly value.
Frequency of Use	Daily

Item	Description
Requirement ID	REQ-2
Requirement Title	Logout.
Priority	Must
Functional Requirement Description	The function can logout the current user.
Frequency of Use	Daily

Item	Description
Requirement ID	REQ-3
Requirement Title	Create account
Priority	Must
Functional Requirement Description	The function can let user create an account to access to the model prediction page and use the model to predict. The function can store user information into the database
Frequency of Use	Daily

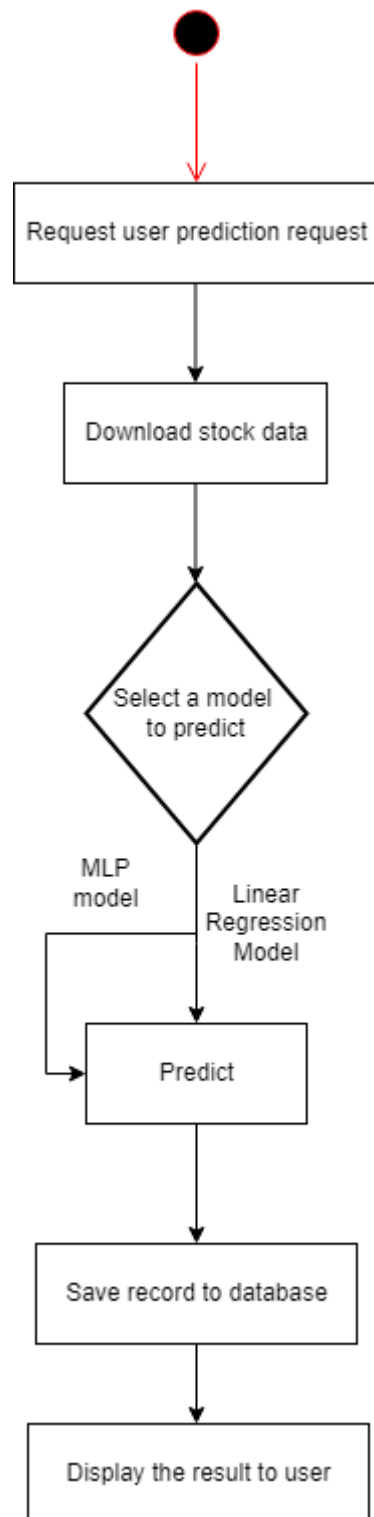
Item	Description
Requirement ID	REQ-4
Requirement Title	Reset account password
Priority	Must
Functional Requirement Description	The user can reset their password of their account. Change the user's password in the database
Frequency of Use	Daily

Item	Description
Requirement ID	REQ-5
Requirement Title	Predict stock data using MLP model
Priority	Must
Functional Requirement Description	read the stored csv make it to pandas data frame drop all the unnecessary columns insert technical indicators

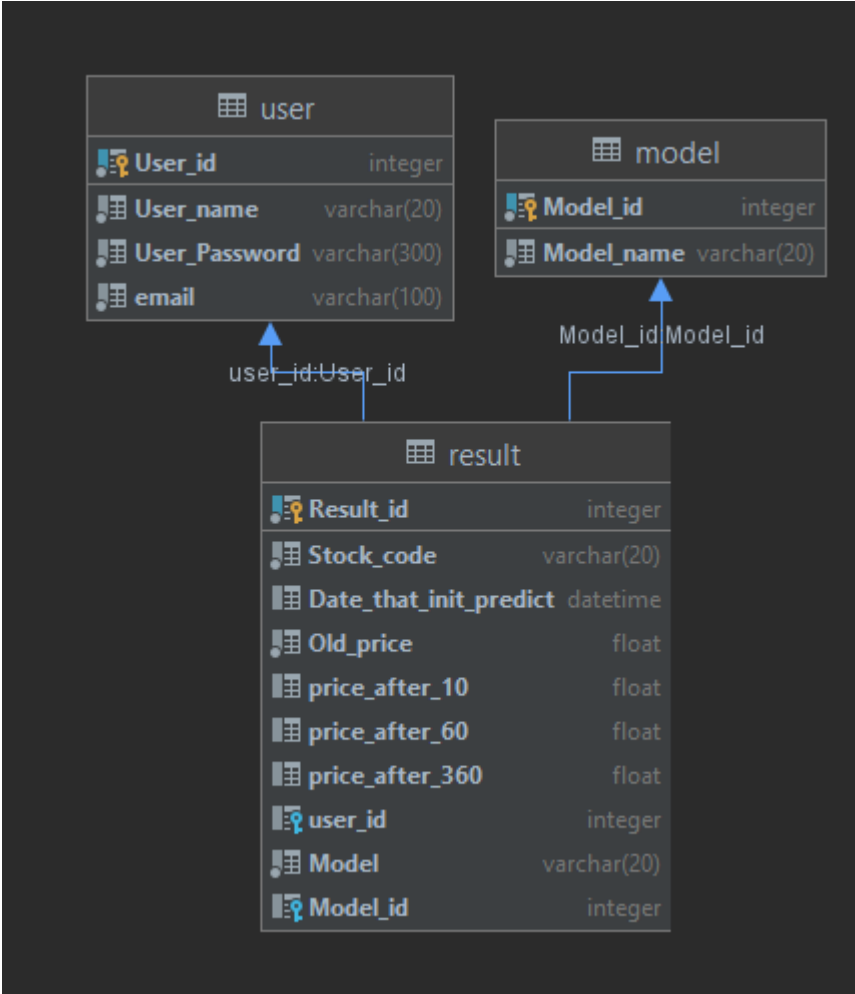
	create features use model to predict.
Frequency of Use	Daily

Item	Description
Requirement ID	REQ-7
Requirement Title	Predict stock data using linear regression model
Priority	Must
Functional Requirement Description	read the stored csv make it to pandas data frame drop all the unnecessary columns insert technical indicators create features use model to predict.
Frequency of Use	Daily

Here is the flow chart of the website.



Here is the database diagram:



Here is the database entity description.

Table Name	Field Name	Field Format	Field Length	Description	Mandatory	Primary Key	Foreign Key
user	User_name	Varchar	20	Username	Y	N	N/A
user	User_id	Integer	20	User id	Y	Y	N/A
user	email	Varchar	20	User's email	Y	N	N/A
user	User_Password	Integer	300	User's password	Y	N	N/A

Table Name	Field Name	Field Format	Field Length	Description	Mandatory	Primary Key	Foreign Key
------------	------------	--------------	--------------	-------------	-----------	-------------	-------------

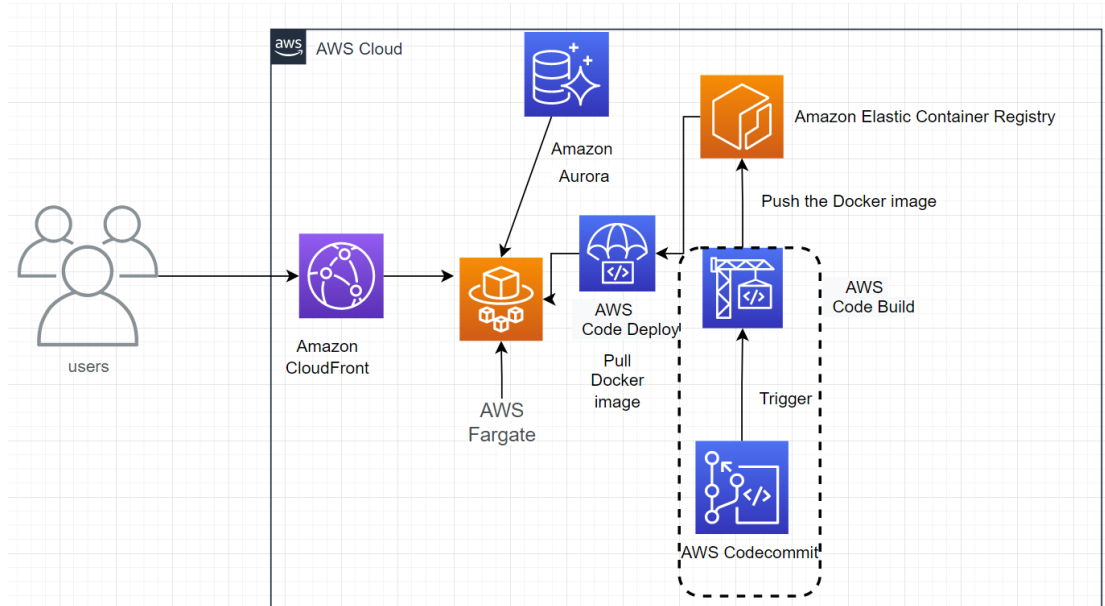
result	Result_ID	Integer	N/A	Payment Detail ID	Y	Y	N/A
result	Stock_code	Integer	N/A	Stock code	Y	N	N/A
result	Date_that_init_predict	Datetime	N/A	Date that do the prediction	Y	N	N/A
result	Old_price	float	N/A	Price during date of prediction	Y	N	N/A
result	Price_after_10	Float	N/A	Price after 10 days	Y	N	N/A
result	Price_after_60	Float	N/A	Price after 60 days	Y	Y	N/A
result	Price_after_365	Float	N/A	Price after 365 days	Y	N	N/A
result	user_id	Integer	N/A	User id that do the prediction	Y	N	User_id
result	Model	Varchar	20	Name of the model that predict.	Y	N	N/A
result	Model_id	Integer	N/A	ID of the model that do the prediction.	Y	N	Model_id

Table Name	Field Name	Field Format	Field Length	Description	Mandatory	Primary Key	Foreign Key
model	Model_id	Integer	N/A	Id of the model	Y	Y	N/A

model	Model_name	Varchar	20	Name of the model	Y	N	N/A
-------	------------	---------	----	-------------------	---	---	-----

## 2. Cloud Architecture

In the infrastructure wise, we will be deploying on the AWS cloud. Here is the architecture diagram.



The workflow of the system is as follow first user will upload his source code to AWS code commit, the code repository than will trigger AWS code build to build docker image that we need for deployment, after that it will push the docker image to Amazon Elastic Container Registry, After docker image pushed to the container registry than it will trigger AWS code deploy will pull the image than deploy to AWS Fargate which is a serverless container service. The endpoint of the AWS Fargate will serve behind Amazon CloudFront which is a caching service to improve latency and performance. And Amazon Aurora is a serverless MySQL database which server as the database for the application. Therefore, the system architecture can automatically sync, upload code to the repository and deploy the website. In the next paragraph I will introduce each services accordingly.

I will introduce the architecture's service from right to left which is the most beginning of the phase of getting the code which is the AWS CodeCommit. AWS CodeCommit is a secure, highly scalable and managed source control service that can host private GIT repositories (*AWS codecommit / managed source control service*). The reason why I choose code commit not using publicly available service like GITHUB. Because AWS code commit is better integrated to the AWS eco system, for example I can use IAM access to limit which service can have pull and push right. But if I use GITHUB it will be trouble some to implement all the right to other cloud services as I need to enter token for each of them. With code commit, I can just change the IAM policy than cloud services can pull the source code from it. Furthermore, the most important factor is due to it sit inside the AWS network the speed performance to pull from outside source like GITHUB, the connection is more secure due to it automatically encrypt your files in transit and rest.

Secondly, after getting the code is the build phase, it will build a docker image and push it to the Amazon Elastic Container Registry the AWS CodeBuild being a fully managed continuous integration service that compiles source code run tests and produce software packages that are ready to deploy (*AWS CodeBuild – Fully Managed Build Service*). It also doesn't need user to provision, manage and scale your own build servers (*AWS CodeBuild – Fully Managed Build Service*). I use AWS CodeBuild in this architecture because it enables the architecture to build and upload docker image automatically, which will decrease the workload of the developer plus it also doesn't require user to provision.

Thirdly after the build phase is complete, successfully create the docker image. It will upload to a service call Amazon Container Registry, it is a fully managed container registry allow user to upload container images and share it (*Amazon Elastic Container Registry (Amazon ECR)*). It is also have HTTPS protocol to automatically encryption and it is a saleable and durable architecture (*Amazon Elastic Container Registry (Amazon ECR)*). I choose this service instead of other free service like docker hub, the reason is very similar with choosing AWS CodeCommit, because of better incorporation with the AWS ecosystem, faster and more stable internet connection and better security.



Fourthly is the deployment phase, this phase it will use AWS CodeDeploy to deploy to the computing service to host the website. AWS CodeDeploy is a fully managed deployment service that automate software deployment on various compute service such as Amazon EC2, AWS fargate and AWS Lambda (*AWS codedeploy / automated software deployment*). It can help you to cut down down time of the deployment and handle update of the application (*AWS codedeploy / automated software deployment*). Not only that it also allow you to track the software deployment process and the deployment process can be easily stop and rollback(*AWS codedeploy / automated software deployment*). I choose AWS CodeDeploy as the service I use during the deployment phase, first is it can automatically deploy to the computing service in this case we are using AWS Fargate, automatically deploy not only can cut down the workload of the developer, but also less error prone, manually deploy and clicking button can easily have error, for example if the developer is tired he will easily miss click and have the wrong configuration. Also when you manually deploy it is very hard to debug and rollback to the previous version, for example if you need to revert into the old version, you have to use command to revert it back and manually redeploy it again. But with this service it will just automatically rollback if it has error and show to the user what is the problem. This is very important for the developer to debug and keep the down time as slim as possible. Apart from that it is also very easy to adopt to different computing services for deployment.

Fifthly, is the AWS Fargate which is the hosting and the computing service that the architecture use. AWS Fargate is a serverless compute service for container, it can uses to deploy and manage your application, the underlying infrastructure will solve the scaling, patching, securing and managing servers (*n.d., AWS Fargate Serverless compute for containers*).Also AWS can improve security through workload isolation, the containers are run in their own dedicated environment (*AWS Fargate Serverless compute for containers*). Apart from this we can also have a cheaper price than using traditional virtual machine, it scales closely match your specific requirements, there is no over-provisioning and paying for additional server (*n.d., AWS Fargate Serverless compute for containers*). In one of their AWS blog article Fargate has a 87% saving

on EC2 (virtual machine) equivalent (*Theoretical cost optimization by Amazon ECS launch type: Fargate vs EC2*). I choose this service instead of traditional virtual machine, because of the factor of I don't need to manage patching, security and update of the container which cut down the workload, also the cost is cheaper than traditional virtual machine. Also the security of the service.

Sixthly is the caching which serve the user, for the caching we will use Amazon Cloud Front, it is a service that will securely deliver content with low latency and high speed. It can reduce latency due to it delivering data through 310 plus global dispersed points of presence with automated network mapping and routing (*Content Delivery Network (CDN) - Amazon CloudFront*). It also can improve the security of the content that serve through CloudFront, due to it is use AWS Shield standard to defend against DDoS attack at no additional charge (*Content Delivery Network (CDN) - Amazon CloudFront*). Choosing AWS CloudFront for caching service can improve user experience on the website due to lower latency, but also, I can have protection against DDoS.

Lastly will be the database that we are using, the service that the architecture will be use is Amazon Aurora, it is a relational database built for cloud performance and provide availability, security and reliability at 1/10 the cost of the commercial-grade databases (n.d., *Amazon Aurora*). It run five times faster than standard MySQL database and three times faster than standard PostgreSQL database (n.d., *Amazon Aurora*). It is fully managed, also features a fault-tolerant, and able to self-healing, point-in-time recovery, and replication across three Availability Zones (n.d., *Amazon Aurora*). Also, able to create read replicas to deal with workload (n.d., *Amazon Aurora*). In this use case We will use Amazon Aurora Serverless, it is also highly available will all Aurora features. With serverless it is even more cost because we only need to pay the capacity consumed, not have to pay for the database instance.

## Chapter 3 Method and its Implementation

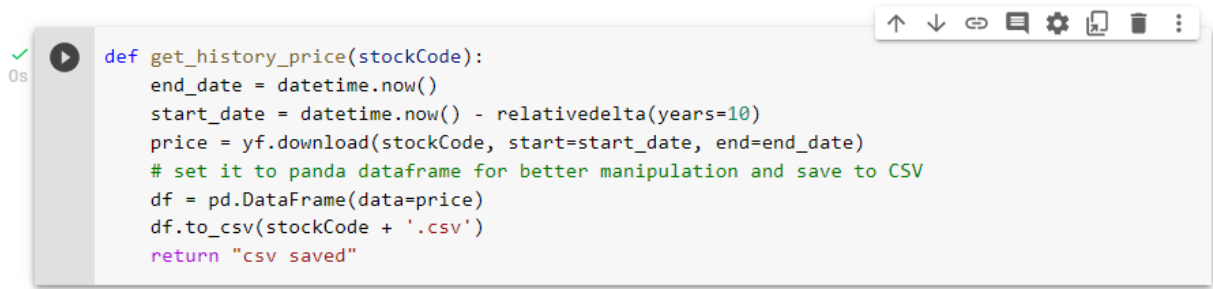
### 1. Implement a linear regression model

As in the objective part, the goal is to use linear regression model to explore the possibility of predicting the stock price. To implement the linear regression model, I will use a programming language call Python which is a high-level programming language and very popular use in area like data analysis and machine learning. And use scikit-learn machine learning library, scikit-learn is a python module integrating wide-range of state-of-the-art machine learning algorithms (Pedregosa et al.). This package can bring machine learning by using general purpose high-level language (Pedregosa et al.). This will be main two tools to create and implement a linear regression model. The model will be use two features, which are exponential moving average in period of 10 and close price as and next closing price as the dependent variable. To predict closing price after 10 days, 60 days and 360 days.

To train the model, first have to import multiple libraries that need to import. Here will import yfinance to download stock data from yahoo finance, we need to import sci-kit learn linear regression module to train linear regression model, pandas\_ta to create technical indicator, other module such as numpy and pandas are for data processing.

First we will create a function call get\_history\_price, this function will be use for get stock data, it will get the last 10 years of data, than convert to pandas data frame and save it as csv.

#### ▼ get stock code data function

A screenshot of a code editor window. On the left, there is a green checkmark and the text '0s'. The code is written in Python and defines a function 'get\_history\_price' that takes 'stockCode' as an argument. The function uses 'datetime.now()' to get the current date and time, and 'relativedelta(years=10)' to calculate the start date 10 years ago. It then uses 'yf.download()' to fetch stock data from Yahoo Finance. The data is converted to a pandas DataFrame and saved to a CSV file. Finally, it returns the string 'csv saved'.

```
def get_history_price(stockCode):
    end_date = datetime.now()
    start_date = datetime.now() - relativedelta(years=10)
    price = yf.download(stockCode, start=start_date, end=end_date)
    # set it to panda dataframe for better manipulation and save to CSV
    df = pd.DataFrame(data=price)
    df.to_csv(stockCode + '.csv')
    return "csv saved"
```

After we successfully download the stock data, we should do some data processing to clean up the data and prepare training split for further training. First we have to define what is the forecast day you want to do, for example I want to do 10 day forecast, than I will set forecast\_day to 10 and read the csv that we have downloaded.

```
# number of days need to forecast in the future
forecast_day = int(10)
df = pd.read_csv(df)
```

After we load the csv using pandas, we have to set the dataframe index as Date column. than use drop\_na function to drop all columns that have null value in them. We also need to drop unnecessary columns such as open price, high price, low price, adjusted close and volume.\

```
# set date as index and data type as date time
df['Date'] = pd.to_datetime(df.Date)
df.index = df['Date']
df.dropna()

# drop all column except the closing price axis=1 mean colmun
df.drop(['Open', 'High', 'Low', 'Adj Close', 'Volume'], axis=1, inplace=True)
```

After we drop and clean the data. We need to calculate the technical indicator of exponential moving average. Therefore, we will use panda\_ta library to calculate the exponential moving average in the length of 10.

```
# use panda ta library to generate exponential moving average in range of 10 days
df.ta.ema(close='Close', length=10, append=True)
```

After doing processing the data and calculate the moving average. We will prepare the data for training. We will first remove forecast day row to 0 for training. Therefore, we will use pandas shift function to remove last 10 day row for training as next\_close. After that we will set the exponential moving average and close price as feature. Set X as the features (independent variable) and Y as the close (dependent variable). And split the data which 70% use for training and 30% use for testing.]

```
# make forecast day row is 0 to train,for example last
df['NextClose'] = df['Close'].shift(-forecast_day)

# independent variable
features_to_fit = ['close', 'EMA_10']
```

For the train model, we will call sci-kit learn library linear regression object than fit the training data that we provide to train the model.

```
# Training, return future price
model = LinearRegression()

model.fit(X_train, Y_train)
```

Finally, will be the prediction part, in the prediction first we will get the to day price, is by using pandas iloc function to select the last row, than add the EMA\_10 and next close column to it to predict next 10 day closing price.

```
prediction = df_today[features_to_fit]
print(df_today)

future_price = model.predict(prediction)
```

Whole code will be here:

```
def prediction_fucnction(df):
    # number of days need to forecast in the future
    forecast_day = int(10)
    df = pd.read_csv(df)
    # modify data drop all columns except the closing price

    # set date as index and data type as date time
    df['Date'] = pd.to_datetime(df.Date)
    df.index = df['Date']
    df.dropna()

    # drop all column except the closing price axis=1 mean colmun
    df.drop(['Open', 'High', 'Low', 'Adj Close', 'Volume'], axis=1, inplace=True)

    # use panda ta library to generate exponential moving average in range of 10 days
    df['EMA_10'] = df.ta.ema(close='Close', length=10, append=True)

    # make forecast day row is 0 to train, for example last
    df['NextClose'] = df['Close'].shift(-forecast_day)

    # independent variable
    features_to_fit = ['close', 'EMA_10']

    df.fillna(0, inplace=True)

    # this is for sci kit learn trianing
    X = df[features_to_fit]
    Y = df['NextClose']

    # train the model
    # split data for training, 70% for traing and 30 % for training
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
    # Training, return future price
    model = LinearRegression()

    model.fit(X_train, Y_train)

    # Price after n days, shift mean make the last -forecast_day data as NAN
    df_today = df.iloc[-1:, :].copy() # get last row [ 0:3( 1 - 4 row), 0:3( 1-4 column)]

    prediction = df_today[features_to_fit]
    print(df_today)

    future_price = model.predict(prediction)

    # the actual data

    return future_price
prediction_fucnction('MSFT.csv')
```

In the website implementation, there will be some differences, to implement of to the website, because the above function is for training the model, if each time the function will run again to train a new model it

will take too much resource also the prediction will not stable. Therefore, in the website implementation, I use joblib, to package the model, it is a tools to provide lightweight pipelining in Python (*Running python functions as pipeline jobs*). Which allow me to package the model. Therefore in the website backend the code will be jilbab to load the model that we trained before and predict 10 days 60 days and 365 days closing price. Here is the whole source code.

```
def prediction_fucnction(df):
    df = pd.read_csv(df)
    # modify data drop all columns except the closing price
    # set date as index and data type as date time
    df['Date'] = pd.to_datetime(df.Date)
    df.index = df['Date']
    df.dropna()

    # drop all column except the closing price axis=1 mean column
    df.drop(['Open', 'High', 'Low', 'Adj Close', 'Volume'], axis=1, inplace=True)

    # use panda ta library to generate exponential moving average in range of 10 days
    df.ta.ema(close='Close', length=10, append=True)

    # Price after n days, shift mean make the last -forecast_day data as NAN
    df_today = df.iloc[-1:, :].copy() # get last row [ 0:3( 1 - 4 row),0:3( 1-4 column)]

    # independent variable
    features_to_fit = ['Close', 'EMA_10']

    df.fillna(0, inplace=True)

    # Training, testing to plot graphs
    model_10days=joblib.load('app/machine_learning_prediction/LR_10_Days')
    model_60_days=joblib.load('app/machine_learning_prediction/LR_60_Days')
    model_365_days=joblib.load('app/machine_learning_prediction/LR_365_Days')

    prediction = df_today[features_to_fit]

    future_price_10_days= model_10days.predict(prediction)
    future_price_60_days= model_60_days.predict(prediction)
    future_price_365_days= model_365_days.predict(prediction)

    return future_price_10_days, future_price_60_days, future_price_365_days
```

## 2. Implement a MLP neural network model

In this we will talk about how to implement a MLP neural network model. Luckily sci-kit learn also have MLP neural network included in it. Therefore, in this part we will also use sci-kit learn library for training the model. The implementation steps in data processing and prepare data for training is the same as the linear regression model. But the difference is in the machine learning method we use. In here machine learning method will import MLPRegressor to use the MLP regression training, the solver will be choose to use adam, which refer to stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba (*Sklearn.neural\_network.Mlpregressor*). The reason why I choose to use this solver because our dataset of stock data is well over 1000 rows, and adam solver is very suitable for processing large dataset in both training time and validation score according to the document of sci-kit learn (*Sklearn.neural\_network.Mlpregressor*). Also in the random state I type a random number in it. This is to ensure the random number generation of the weight and bias initiation will be the same, make the prediction result will be reproduceable. Apart from that the technical indicator will not be exponential moving average it will be relative strength index, because of want to explore if different kind of technical indicator and machine learning method will be difference in prediction result.



Here is the full code:

```
def prediction_fucnction_mlp(df):
    # number of days need to forecast in the future
    forecast_day = int(10)
    df = pd.read_csv(df)
    # modify data drop all columns except the closing price

    # set date as index and data type as date time
    df['Date'] = pd.to_datetime(df.Date)
    df.index = df['Date']
    df.dropna()

    # drop all column except the closing price axis=1 mean colmun
    df.drop(['Open', 'High', 'Low', 'Adj Close', 'Volume'], axis=1, inplace=True)

    # use panda ta library to generate exponential moving average in range of 10 days
    df['RSI'] = df.ta.rsi(close='Close', length=10, append=True)

    # Price after n days, shift mean make the last -forecast_day data as NAN
    df_today = df.iloc[-1:, :].copy() # get last row [ 0:3( 1 - 4 row),0:3( 1-4 column)]

    # make forecast day row is 0 to train,for example last
    df['NextClose'] = df['Close'].shift(-forecast_day)

    # independent variable
    features_to_fit = ['Close', 'RSI']

    df.fillna(0, inplace=True)

    # this is for sci kit learn trianing
    X = df[features_to_fit]
    Y = df['NextClose']

    # train the model
    # split data for training, 70% for traing and 30 % for training
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)

    # Traning, testing to plot graphs
    mlp = MLPRegressor(solver='adam', random_state=42, activation="relu")
    mlp.fit(X_train, Y_train)

    prediction = df_today[features_to_fit]
    print(prediction)
    future_price = mlp.predict(prediction)

    return future_price
prediction_fucnction_mlp('MSFT.csv')
```

In the website implementation, we also use joblib to package the model for the backend, this is to ensure not to use too much resource in the container and the result is reproduceable and stable.

Here is the website backend code.

```
def prediction_fucnction_mlp(df):
    df = pd.read_csv(df)
    # modify data drop all columns except the closing price

    # set date as index and data type as date time
    df['Date'] = pd.to_datetime(df.Date)
    df.index = df['Date']
    df.dropna()

    # drop all column except the closing price axis=1 mean colmun
    df.drop(['Open', 'High', 'Low', 'Adj Close', 'Volume'], axis=1, inplace=True)

    # use panda ta library to generate rsi in range of 10 days
    df['RSI'] = df.ta.rsi(close='Close', length=10, append=True)

    # Price after n days, shift mean make the last -forecast_day data as NAN
    df_today = df.iloc[-1:, :].copy() # get last row [ 0:3( 1 - 4 row), 0:3( 1-4 column)]

    # independent variable
    features_to_fit = ['Close', 'RSI']

    df.fillna(0, inplace=True)

    # Use model to predict the stock price
    model_10days = joblib.load('app/machine_learning_prediction/MLP_model_10_Days')
    model_60_days = joblib.load('app/machine_learning_prediction/MLP_model_60_Days')
    model_365_days = joblib.load('app/machine_learning_prediction/MLP_model_365_Days')

    prediction = df_today[features_to_fit]

    future_price_10_days = model_10days.predict(prediction)
    future_price_60_days = model_60_days.predict(prediction)
    future_price_365_days = model_365_days.predict(prediction)
    return future_price_10_days, future_price_60_days, future_price_365_days
```

### 3. Cloud system infrastructure as code

In the cloud system, we will use AWS Cloud Development Kit as my tool to provision and deploy my cloud infrastructure. AWS Cloud development kit is an open-source software development framework to define the cloud application resource that need to deploy (*AWS Cloud Development Kit*). Therefore, I can avoid error prone manual deployment and allow me to quicker deployment.

First is to create a VPC network which have cidr mask of 16.

And with 2 Public Subnet and 2 private subnet which have cidr mask of 24. And 2 nat gateway for private subnet to access to outside network.

Here is the code:

```
from aws_cdk import (aws_ec2 as ec2,
                      core
                      )
# create vpc with 4 subnet
class VPC_stack(core.Stack):
    def __init__(self, scope: core.Construct, construct_id: str, **kwargs):
        super().__init__(scope, construct_id, **kwargs)
        self.vpc = ec2.Vpc(self, "Infra_VPC",
                           cidr="10.0.0.0/16",
                           max_azs=2,
                           # 2 public and private subnet with 2 nat i
                           subnet_configuration=[
                               ec2.SubnetConfiguration(
                                   name='Public-Subnet',
                                   subnet_type=ec2.SubnetType.PUBLIC,
                                   cidr_mask=24
                               ),
                               ec2.SubnetConfiguration(
                                   name='Private-Subnet',
                                   subnet_type=ec2.SubnetType.PRIVATE
                                   cidr_mask=24)
                           ],
                           nat_gateways=2
        )
```

Secondly we will create the database, here we will provision a Amazon Aurora serverless database cluster, with rules such as stop after 10 minutes to idle, minimal capacity units is 8 and max capacity unit is 32. deploy in private VPC for security and have a security group to allow port 3306 for MySQL access.

Here is the code:

```
from aws_cdk import (aws_rds as rds,
                      aws_ec2 as ec2,
                      core
)
from cdk.vpc_construct.fyp_vpc import VPC_stack

class Database_Stack(core.Stack):
    def __init__(self, scope: core.Construct, construct_id: str, props: ec2.Vpc, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)
        auroa_sec_group = ec2.SecurityGroup(self, "AuroraSecurityGroup",
                                           vpc=props,
                                           description="Allow database access",
                                           allow_all_outbound=True)

        auroa_sec_group.add_ingress_rule(ec2.Peer.any_ipv4(), ec2.Port.tcp(3306), "allow mysql access")

        self.cluster = rds.ServerlessCluster(self, "AuroraCluster",
                                           engine=rds.DatabaseClusterEngine.AURORA_MYSQL,
                                           scaling=rds.ServerlessScalingOptions(
                                               auto_pause=core.Duration.minutes(10),
                                               # default is to pause after 10 minutes of idle time
                                               min_capacity=rds.AuroraCapacityUnit.ACUC_8,
                                               # default is 2 Aurora capacity units (ACUs)
                                               max_capacity=rds.AuroraCapacityUnit.ACUC_32
                                           ),
                                           vpc=props,
                                           vpc_subnets={"subnet_type": ec2.SubnetType.PRIVATE},
                                           security_groups=[auroa_sec_group]
                                           )

        self.database_endpoint = self.cluster.cluster_endpoint
```

Thirdly we will provision the repository, build phase and deployment phase of the cloud architecture. first, we will provision a Amazon Code Pipeline pipe line to connect all the phases. After that we create an Amazon Elastic Container registry to store the docker image. Here is the code.

```

super().__init__(scope, construct_id, **kwargs)
# Code Pipeline
pipeline = codepipeline.Pipeline(self, "Flask_App_Pipeline",
                                pipeline_name="Flask_App_Pipeline"
                                )

# this block of code is to set artifact for out out put
source_output = codepipeline.Artifact(artifact_name="source_output")
docker_output = codepipeline.Artifact(artifact_name="Docker")

# create container repo
self.container_repository = ecr.Repository(
    scope=self,
    id="containter_repo",
    repository_name="container_repo"
)

```

After that we will enter to the the phase of getting the code, first we will create an AWS CodeCommit repository, it will store the flask website code and the docker file. Than we will add the action of get\_souce\_from\_codecommit, add it as source stage. This will trigger the AWS CodeCommit whenever there are code upload to the master branch.

Here is the code.

```

# code commit repo, repo name and description is defined.
self.codecommit_repo = codecommit.Repository(
    scope=self,
    id="flask-repo",
    repository_name="flask-repo",
    description="flask app and docker code"
)

# code commit source action get source code from code commit repo. And the branch is master.
get_source = codepipeline_actions.CodeCommitSourceAction(
    action_name="get_source_from_codeCommit",
    repository=self.codecommit_repo,
    output=source_output,
    branch="master"
)

# pipeline add first stage get repo code. This stage is source stage which mean
pipeline.add_stage(
    stage_name="Source",
    actions=[get_source]
)

```

After that we will enter build stage, first we will create codebuild project which will build docker according to the docker according to the buildspec file, the build spec file is a file with command that will order CodeBuild to build docker and push to the ECR repository. then it also allow Code Build to push and pull from the ECR repository. Finally, we will add it to Code Pipeline build stage to build the docker and push to ECR. this will trigger when ever a code is pushed to Code Commit repository. Here is the code of CDK and build spec.

```
# define docker build spec, build spec is set of command that is use for building the environment and docker.
buildspec_docker = codebuild.BuildSpec.from_source_filename("buildspec.yml")

# define action of build docker, this will set up the which image should use and the environment variable to
# get the ecr image.

build_docker = codebuild.PipelineProject(
    self,
    "Build Docker",
    environment=dict(
        build_image=codebuild.LinuxBuildImage.AMAZON_LINUX_2_3,
        privileges=True),
    environment_variables={
        'REPO_ECR': codebuild.BuildEnvironmentVariable(
            value=self.container_repository.repository_uri),
    },
    build_spec=buildspec_docker
)

# define role for building docker and grant push pull right to repo.
build_docker.add_to_role_policy(iam.PolicyStatement(
    effect=iam.Effect.ALLOW,
    actions=["ecr:BatchCheckLayerAvailability", "ecr:GetDownloadUrlForLayer", "ecr:BatchGetImage"],
    resources=[
        f"arn:{core.Stack.of(self).partition}:ecr:{core.Stack.of(self).region}:{core.Stack.of(self).account}:repository/*", ))

self.container_repository.grant_pull_push(build_docker.role)

# stage in pipeline

# this add build docker stage in the pipeline also know as build stage.
pipeline.add_stage(
    stage_name="DockerBuild",
    actions=[
        codepipeline_actions.CodeBuildAction(
            action_name="Build_docker_and_push_to_ECR",
            project=build_docker,
            input=source_output,
            outputs=[docker_output])
    ]
)
```

Finally, is the deployment stage, in the deployment stage we first provision an ECS cluster, than the cluster will define task which host the docker container, the container have following specification, first the memory is 512 mb, than the cpu will be 256, which is roughly about 0.25 cpu, also it will expose port 8000. We also will set the cluster will always have 2 containers and have auto scaling when cpu utilization more than 50 %. Max scaling container will be 6 containers. We will also add a load balancer to the cluster to allocate the workload. After provisioning the cluster we will create a CodeDeploy application, set the deployment

policy to Time based Canary, which every 5 minutes it will upgrade 20% of the cluster's container to new version. And finally add to the CodePipeline deployment stage, add the CodeDeploy application and point to the ECS fargate service deploy. Here is the code.

```
# define deploy stage.
application = codedeploy.EcsApplication(
    self,
    "FLASK_APPLICATION",
    application_name="FLASK_APPLICATION"
)

# create ecs cluster
cluster = ecs.Cluster(self, "Cluster",
                      vpc=props
                      )

# create task definition
task_definiton = ecs.FargateTaskDefinition(
    self, "TaskDef",
)

# task definition policy
task_definiton.add_to_task_role_policy(
    iam.PolicyStatement(
        effect=iam.Effect.ALLOW,
        resources=["*"],
        actions=[
            "ecr:GetAuthorizationToken",
            "ecr:BatchCheckLayerAvailability",
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
        ]
    )
)

# create container
task_definiton.add_container("WebContainer",
                             image=ecs.ContainerImage.from_ecr_repository(
                                 self.container_repository),
                             memory_limit_mib=512,
                             cpu=256,
                             port_mappings=[ecs.PortMapping(container_port=8000)]
                             )
```

```

# cluster Service
self.cluster_service = ecs.FargateService(
    self,
    "ECS-FARGATE-SERVICE",
    cluster=cluster,
    task_definition=task_definition,
    desired_count=2
)

scaling = self.cluster_service.auto_scale_task_count(max_capacity=6,
                                                    min_capacity=2)
scaling.scale_on_cpu_utilization("cpu_scaling", target_utilization_percent=50)

self.load_balancer = elbv2.ApplicationLoadBalancer(self, "LB_FOR_FARGATE",
                                                    vpc=props,
                                                    vpc_subnets={"subnet_type": ec2.SubnetType.PUBLIC},
                                                    internet_facing=True
                                                    )
listener = self.load_balancer.add_listener("Listener", port=8000)
listener.add_targets("ECS-FARGATE",
                    port=8000,
                    targets=[self.cluster_service])

cfn_deployment_config = codedeploy.CfnDeploymentConfig(self, "MyCfnDeploymentConfig",
                                                    compute_platform="ECS",
                                                    deployment_config_name="deploymentConfigName",
                                                    traffic_routing_config=codedeploy.CfnDeploymentConfig.TrafficRoutingConfigProperty(
                                                        type="TimeBasedCanary",
                                                        # the properties below are optional
                                                        time_based_canary=codedeploy.CfnDeploymentConfig.TimeBasedCanaryProperty(
                                                            canary_interval=5,
                                                            canary_percentage=20
                                                        )
                                                    ),
                                                    )

ecs_deployment_group = codedeploy.EcsDeploymentGroup.from_ecs_deployment_group_attributes(self,
                                                                                          "ecs_deployment_group",
                                                                                          application=application,
                                                                                          deployment_group_name="Fargate_Deployment_Group",
                                                                                          deployment_config=cfn_deployment_config
                                                                                          )

pipeline.add_stage(
    stage_name="deploy_to_ecs",
    actions=[
        codepipeline_actions.EcsDeployAction(
            action_name='Deploy_to_ECS',
            service=self.cluster_service,
            image_file=codepipeline.ArtifactPath(docker_output, "imagedefinitions.json"),
        )
    ]
)

```



Finally, is the CloudFront caching service, that all we need is to point the CloudFront distribution to the cluster load balancer. Here is the code.

```
from aws_cdk import (aws_cloudfront as cloudfront,
                      aws_elasticloadbalancingv2 as elbv2,
                      aws_cloudfront_origins as origins,
                      core
)

# this will be create a cloudfront distribution with a elb as entry point

class cloudfront_dist(core.Stack):
    def __init__(self, scope: core.Construct, construct_id: str, props: elbv2.ApplicationLoadBalancer, **kwargs):
        super().__init__(scope, construct_id, **kwargs)
        self.cloud_dist = cloudfront.Distribution(self, "myDist",
                                                  default_behavior=cloudfront.BehaviorOptions(
                                                      origin=origins.LoadBalancerV2Origin(props))
                                                  )
```

#### 4. Website Implementation

For the website implementation we will use flask framework, flask is a micro MVC framework for building website. The website will have 4 pages, home page, login page, register page and prediction page. I will discuss each backend one by one and show one front end as example because front end is very similar. In this flask website, we also use blueprint to better manage each section of the website. Each backend folder is separated and register as their own module in the `__init__.py`. For better debugging and management.

First let talks the home page, the home page is a page that tell the user what the website is about and navigate the user. Here is the backend code.

```
from flask import render_template
from app.Home import bp

@bp.route('/', methods=['GET'])
def home():
    return render_template('Home/home.html')
```

The front-end code of the website, we first will develop a webpage that form as base page to serve content that is frequently use, for example navigation bar. The home page will extend from the base page, and add it new content on top of it. In this case it will be the description of the machine learning model and the link to use the prediction page. Here is the code of the base page.

```
{% extends 'bootstrap/base.html' %}

{% block head %}
    {{ super() }}
    <link href="{{ url_for('static', filename='bootstrap-responsive.css') }}" rel="stylesheet">
    <style type="text/css">
        body {
            padding-top: 60px;
            padding-bottom: 40px;
        }
    </style>
    <link href="{{ url_for('static', filename='bootstrap.css') }}" rel="stylesheet">
{% endblock %}

{% block navbar %}
    <div class="navbar navbar-fixed-top">
        <div class="navbar-inner">
            <div class="container">
                <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </a>
                <a class="brand" href="/">Machine Learning Stock Prediction</a>
                <div class="nav-collapse">
                    <ul class="nav">
                        <li class="active"><a href="/">Home</a></li>
                        <li><a href="/ml_predict">Prediction</a></li>
                    </ul>
                    <ul class="nav navbar-nav navbar-right">
                        {% if current_user.is_anonymous %}
                            <li>
                                <a href="/login">Login</a>
                            </li>
                        {% else %}
                            <li><a>Hello! {{ current_user.User_name }} &nbsp; &nbsp;</a></li>
                            <li class="">
                                <a href="/logout" methods="POST">Log Out</a>
                            </li>
                        {% endif %}
                    </ul>
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

Here is the code of the Home page.

```
{% extends "base.html" %}

{% block title %}
    Stock Prediction - Home
{% endblock %}

{% block content %}
    {% with messages = get_flashed_messages(with_categories=True) %}
        {% if messages %}
            {% for category, message in messages %}
                <div class="alert alert-{{ category }}">
                    {{ message }}
                </div>
            {% endfor %}
        {% endif %}
    {% endwith %}
    <div class="container">

        <!-- Main hero unit for a primary marketing message or call to action -->
        <div class="hero-unit">
            <h1>Stock prediction with machine learning</h1>
            <p>Would you like to know the future stock price? In this website we offer you two machine learning model
                for you to predict stock price.</p>
            <p><a class="btn btn-primary btn-large" href="/ml_predict">Click here to try! >></a></p>
        </div>

        <!-- Example row of columns -->
        <div class="row">
            <div class="span4">
                <h2>First model - Linear Regression</h2>
                <p>Linear regression model use set of predictor variable to predict the outcome variable, in this model
                    the independent variable will be exponential moving average in 10 days periods
                    and closing price of the day to predict the dependent variable of the next closing price</p>
                <p><a class="btn" href="/ml_predict">Click here go to predict >></a></p>
            </div>
            <div class="span4">
                <h2>Second model - Multilayer perceptron </h2>
                <p>Multilayer perceptron is a feed forward neural network, it includes input layer, output layer and
                    hidden layer. It have multiple use cases such as pattern classification and prediction
                    it is design to solve problems that are not linearly separable</p>
                <p><a class="btn" href="/ml_predict">Click here go to predict >></a></p>
            </div>
        </div>
    </div>
{% endblock %}
```

Next page will be Login page, this page allows the user to login and register an account. The backend function of route login, which will redirect to the login page. It will compare user's username and password to the database and authenticate it, if user information is correct, he will be authenticated and allow to login and user the machine learning service. Otherwise, he needs to check his information again or register an account. The login function is complete by flask-login library. Here is the backend code.

```

@bp.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('Home_Page.home'))
    form = loginform(csrf_enabled=False)
    if request.method == 'POST' and form.validate_on_submit():
        user_name = form.username.data
        password = form.password.data
        user = User.query.filter_by(User_name=user_name).first()
        if not user or not check_password_hash(user.User_Password, password):
            flash('Please check your login details and try again.', category='danger')
            return redirect(url_for('login_page.login'))
        else:
            login_user(user)
            flash('Welcome ' + user.User_name, category='success')
            return redirect(url_for('Home_Page.home'))
    return render_template('Login/login.html', form=form)

```

The route /logout is the logout backend function and it allow user to logout from the website. The backend is use the logout function from flask-login library here is the code.

```

@login_required
@bp.route('/logout')
def logout():
    user_id = current_user.get_id()
    logout_user()
    flash('Success logout', category='success')
    return redirect(url_for('Home_Page.home'))

```

The route /register, is the register backend function and allow user to register an account. The backend function will check if the user is login or not, if he is login then he will redirect to the home page. Once the user in the register page he will enter it account information in the form. When they submit, it will log the user in the database and successfully register an account.

Here is the code.

```
@bp.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('Home_Page.home'))
    form = RegistrationForm()
    if form.validate_on_submit():
        user = User(User_name=form.username.data, email=form.email.data,
                    User_Password=generate_password_hash(form.password.data, method='sha256'))
        db.session.add(user)
        db.session.commit()
        flash('Congratulations, you are now a registered user!')
        return redirect(url_for('login_page.login'))
    return render_template('Login/register.html', title='Register', form=form)
```

This is the code form.py, which include all the form that we need to use in the Login Module.

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, PasswordField, ValidationError
from wtforms.validators import DataRequired, EqualTo, Email
from model import User

class loginform(FlaskForm):
    username = StringField('Account Name', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Submit')

class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    password2 = PasswordField(
        'Repeat Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Register')
    def validate_username(self, username):
        user = User.query.filter_by(User_name=username.data).first()
        if user is not None:
            raise ValidationError('Please use a different username.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user is not None:
            raise ValidationError('Please use a different email address.')

class ResetPasswordForm(FlaskForm):
    password = PasswordField('Password', validators=[DataRequired()])
    password2 = PasswordField(
        'Repeat Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Request Password Reset')
```

Lastly is the machine learning page, the machine learning page is allow the user with a account, able to use the MLP model and linear regression model to predict the stock price. It also will display the prediction result. Since we already talk about the machine learning function, I will talk about the backend. It allows the user select which model to predict stock and store the result to the database. When the user enter this page, he will enter the stock code and choose the model that he want to use in the form. After he click submit it will download the stock data, and use the model he select to predict the data. After prediction is complete it will show the result on the table and a flash message will appear. The result will also record in the data base.

This is the backend code.

```
@login_required
@op.route('/ml_predict', methods=['GET', 'POST'])
def prediction():
    form = ml_form(csrf_enabled=False)
    form.ml_model.choices = [(model.Model_id, model.Model_name) for model in Model.query.all()]
    prediction_history=[]
    if current_user.is_authenticated:
        prediction_history = Result.query.filter_by(user_id=current_user.User_id) # query all the history prediction
        if request.method == 'POST' and form.validate_on_submit():
            stock_code = form.stock_code.data
            get_model_name = Model.query.filter_by(
                Model_id=form.ml_model.data).first()
            if get_model_name.Model_name == "linear regression":
                # first get stock csv first
                get_history_price(stock_code)
                # train and predict price of stock
                # today_price
                today_price = yf.Ticker(form.stock_code.data)
                price = today_price.info['regularMarketPrice']
                # prediction with prediction and mark to database
                future_price = prediction_fuction(stock_code + '.csv')
                prediction_record = Result(stock_code=stock_code, Old_price=price,
                    price_after_10=round(int(future_price[0]), 3),
                    price_after_60=round(int(future_price[1]), 3),
                    price_after_360=round(int(future_price[2]), 3), Model="linear regression", Model_id=form.ml_model.data, user_id=current_user.User_id,
                    Date_that_init_predict=datetime.now())
                # user_id=current_user.User_id
                db.session.add(prediction_record)
                db.session.commit()
                os.remove(stock_code + '.csv')
                flash('Success predict stock ' + str(stock_code) + ' price after 10 days will be $' + str(
                    round(int(future_price[0]), 3)), category='success')
                return redirect('/ml_predict')
            #
            elif get_model_name.Model_name == "MLP prediction":
                # first get stock csv first
                get_history_price(stock_code)
                # train and predict price of stock
                # today_price
                today_price = yf.Ticker(form.stock_code.data)
                price = today_price.info['regularMarketPrice']
                # prediction with prediction and mark to database
                future_price = prediction_fuction_mlp(stock_code + '.csv')
                prediction_record = Result(stock_code=stock_code, Old_price=price,
                    price_after_10=round(int(future_price[0]), 3),
                    price_after_60=round(int(future_price[1]), 3),
                    price_after_360=round(int(future_price[2]), 3), Model="MLP prediction",
                    Date_that_init_predict=datetime.now(), Model_id=form.ml_model.data,
                    user_id=current_user.User_id)
                db.session.add(prediction_record)
                db.session.commit()
                os.remove(stock_code + '.csv')
                flash('Success predict stock ' + str(stock_code) + ' price after 10 days will be $' + str(
                    round(int(future_price[0]), 3)),
                    category='success')
                return redirect('/ml_predict')
            else:
                flash('Please register an account to use prediction!', category='danger')
                return redirect(url_for('Home_Page.home'))
    return render_template('machine_learning_prediction/prediction.html',
        prediction_history=prediction_history, form=form)
```

This is the front-end code; front end is mainly to display the database data of the prediction record and send option on which model to train and get which stock data.

```
{% extends "base.html" %}
{% import "bootstrap/wtf.html" as wtf %}

{% block title %}
    Stock - Prediction
{% endblock %}

{% block content %}
    {% with messages = get_flashed_messages(with_categories=True) %}
    {% if messages %}
        {% for category, message in messages %}
            <div class="alert alert-{{ category }}">
                {{ message }}
            </div>
        {% endfor %}
    {% endif %}
    {% endwith %}
    <div class="container">
        <div class="row">
            <div class="col-md-6">
                <h1 class="display-4">Make prediction</h1>
                <div class="card">
                    <div class="card-body">
                        <form action="" method="POST">
                            {{ form.csrf_token() }}
                            {{ form.ml_model.label }}
                            <div class="form-group">
                                {{ form.ml_model(class="form-control", placeholder="Please select a prediction") }}
                            </div>
                            {{ form.stock_code.label }}
                            <div class="form-group">
                                {{ form.stock_code(class="form-control", placeholder="Please enter a stock name") }}
                            </div>
                            {{ form.submit(class="btn btn-primary") }}
                        </form>
                    </div>
                </div>
            </div>
            <div class="col-md-6">
                <br/>
                <br/>
                <br/>
                <table class="table table-bordered table-hover">
                    <thead>
                        <tr>
                            <th>Result ID</th>
                            <th>Stock Code</th>
                            <th>Date of prediction</th>
                            <th>Price during date of prediction</th>
                            <th>Price after 10 days</th>
                            <th>Price after 60 days</th>
                            <th>Price after 365 days</th>
                            <th>Model</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for record in prediction_histroy %}
                            <tr>
                                <td>{{ record.Result_id }}</td>
                                <td>{{ record.Stock_code }}</td>
                                <td>{{ record.Date_that_init_predict }}</td>
                                <td>{{ record.Old_price }}</td>
                                <td>{{ record.price_after_10 }}</td>
                                <td>{{ record.price_after_60 }}</td>
                                <td>{{ record.price_after_360 }}</td>
                                <td>{{ record.Model }}</td>
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </div>
{% endblock %}
```



For the database schema, database schema will be using flask-sqlalchemy ORM mapper to implement. This is the code that create all database table.

```
from app import db
from flask_login import UserMixin

class User(db.Model, UserMixin):
    User_id = db.Column(db.Integer(), unique=True, primary_key=True, nullable=False)
    User_name = db.Column(db.String(20), nullable=False)
    User_Password=db.Column(db.String(300), nullable=False)
    result=db.relationship('Result', backref='user', lazy=True)
    email=db.Column(db.String(100), nullable=False)

    def get_id(self):
        return (self.User_id)

    def __repr__(self):
        return '<User %r>' % self.User_name

class Result(db.Model):
    Result_id = db.Column(db.Integer(), unique=True, primary_key=True, nullable=False)
    Stock_code=db.Column(db.String(20), nullable=False)
    Date_that_init_predict=db.Column(db.DateTime)
    Old_price=db.Column(db.Float(), nullable=False)
    price_after_10=db.Column(db.Float())
    price_after_60=db.Column(db.Float())
    price_after_360=db.Column(db.Float())
    user_id=db.Column(db.Integer, db.ForeignKey('user.User_id'))
    Model=db.Column(db.String(20), nullable=False)
    Model_id=db.Column(db.Integer, db.ForeignKey('model.Model_id'))
    def __repr__(self):
        return '<Result %r>' % self.Result_id

class Model(db.Model):
    Model_id = db.Column(db.Integer(), unique=True, primary_key=True, nullable=False)
    Model_name=db.Column(db.String(20), nullable=False)
    result_model_id=db.relationship('Result', backref='model', lazy=True)
    def __repr__(self):
        return '<Model %r>' % self.Model_name
```

Lastly is the docker file, the docker file will be used for creating docker container to serve the website on the cloud. This is the docker file, this docker file will copy all the code of the website and machine learning model file. Than install require dependency and library, and run gunicorn to host the website and expose port 8000.

```
#pull a python image
FROM smizy/scikit-learn:latest

#Setup working directory
WORKDIR /app

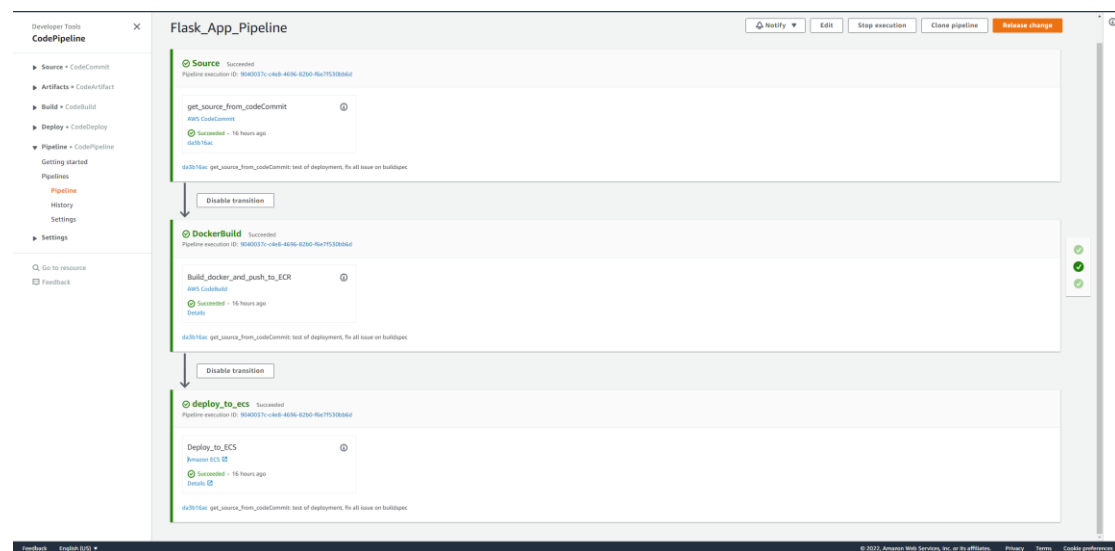
#add all files from current directory
ADD . /app
RUN rm -rf venv

##create a interpreter for cffi
RUN apk add --no-cache --virtual .pynacl_deps build-base python3-dev libffi-dev
RUN apk update && apk add libxml2 libxslt-dev
#update tools
RUN apk -U upgrade
RUN pip install --upgrade setuptools
RUN apk add --no-cache libffi-dev openssl-dev
RUN apk add --no-cache bash

#CD to app
RUN python -m venv venv
RUN source venv/bin/activate
#install dependency
RUN pip install --trusted-host pypi.python.org -r requirements_docker.txt
#expose port to 80 and 8000
EXPOSE 8000
#run gunicorn command
CMD bash ./script.sh
```

## Chapter 4 Work done

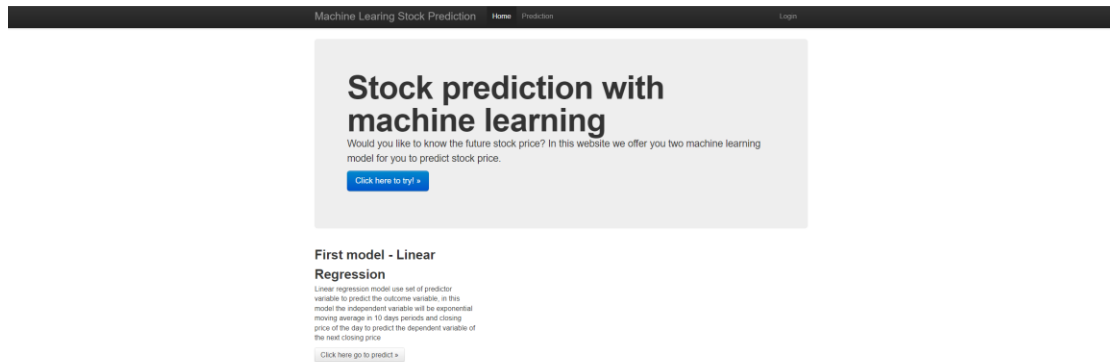
CodeCommit, CodeBuild and CodeDeploy able to store code, build docker, push to the ECR repository and Deploy to the ECR Fargate cluster from updated code. Which also show that the cdk code deployment is functional.



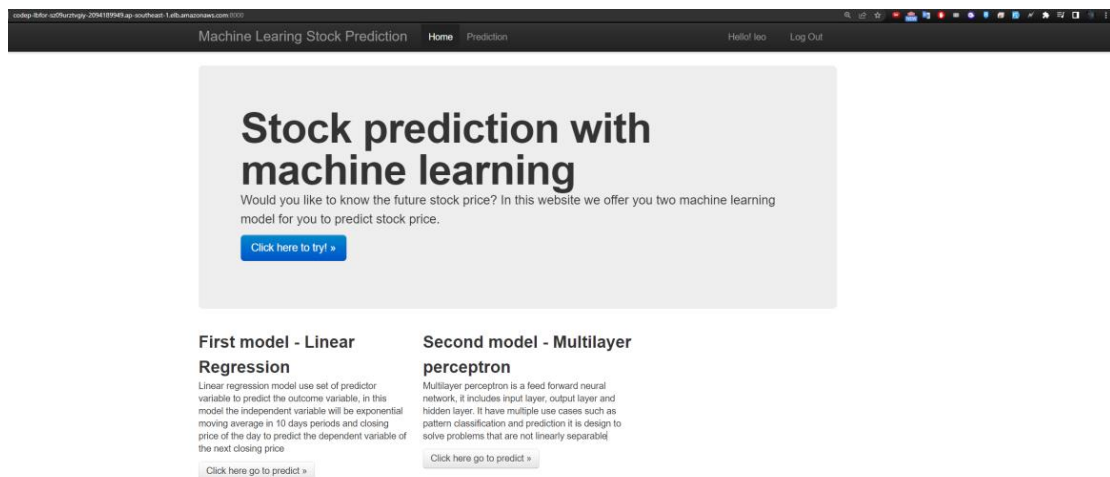
For example, I deleted the MPL description in the home page. Here is the code commit history.



After deployment complete is updated with the following change. It don't have the MPL model description.



ECR Faregate can serve the website.



[Home](#) [Prediction](#) [Login](#)

## Register

Username

Tester 2

Email

leungwinghin09@gmail.com

Password

\*\*\*\*\*

Repeat Password

\*\*\*\*\*

Register

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane is expanded, showing the 'sys' database. The 'sys.sysusers' table is selected. The main pane displays a query result for the 'sys.sysusers' table. The query is: `SELECT * FROM sys.sysusers`. The result set contains two rows:

id	name	sysname	email
1	admin	sa	sa@sql.com
2	Tester2	sa	sa@sql.com

User can login and use the machine model to successfully predict the stock code.

Machine Learning Stock Prediction

HomePrediction

LogoutadminLogout

Success predict stock AAPL price after 10 days will be \$171

Make prediction

Select a machine learning prediction

linear regression

Stock Code

Please enter a stock name

Submit

Result ID	Stock Code	Date of prediction	Price during date of prediction	Price after 10 days	Price after 60 days	Price after 365 days	Model
1	AAPL	2022-04-03 10:33:16	174.31	171.0	162.0	105.0	linear regression

Machine Learning Stock Prediction

HomePrediction

LogoutadminLogout

Success predict stock NVDA price after 10 days will be \$202

Make prediction

Select a machine learning prediction

linear regression

Stock Code

Please enter a stock name

Submit

Result ID	Stock Code	Date of prediction	Price during date of prediction	Price after 10 days	Price after 60 days	Price after 365 days	Model
1	AAPL	2022-04-03 10:33:16	174.31	171.0	162.0	105.0	linear regression
2	NVDA	2022-04-03 10:33:53	267.12	262.0	234.0	57.0	MLP prediction

Database record of prediction result.

SCHEMAS

Filter objects

sys

Tables

model

result

sys\_config

user

Columns

Indexes

Foreign Keys

Triggers

Views

Stored Procedures

Functions

Limit to 1000 rows

SELECT \* FROM sys.result;

Result Grid

Filter Rows

Edit

Export/Import

Wrap Cell Contents

Result_id	Stock_code	Date_that_init_predict	Old_price	price_after_10	price_after_60	price_after_360	user_id	Model	Model_id
1	AAPL	2022-04-03 10:33:16	174.31	171	162	105	1	linear regression	1
2	NVDA	2022-04-03 10:33:53	267.12	262	234	57	1	MLP prediction	2

## Chapter 5 Performance

### 1. Evaluate model performance

In this part I will evaluate both linear regression and MLP Regression model performance. We will use three metrics which are  $r^2$  score, mean absolute error and mean squared error.

$R^2$  score also known as R squared, is a metric that tells the performance of the model (*Evaluation metrics for your regression model 2021*).  $R^2$  square calculates how much regression line is better than a mean line. To interpret  $R^2$  score, as the regression line moves toward perfection the  $R^2$  score will move toward 1 (*Evaluation metrics for your regression model 2021*). Therefore,  $R^2$  score closer to 1 is better, for example if  $r^2$  score is 0.8 it means the model can explain 80% of the variance of data (*Evaluation metrics for your regression model 2021*).

Secondly is the mean absolute error, it calculates the difference between actual and predicted values (*Evaluation metrics for your regression model 2021*). Then take the average, to indicate how close is the predicted result to real data. The smaller mean absolute value, the more fit to the real data (*Evaluation metrics for your regression model 2021*). The advantage of MAE is it is a robust outlier and gets the same unit as output variable, the disadvantage of it is not differentiable (*Evaluation metrics for your regression model 2021*).

Finally, is the mean squared error, mean squared error basically is the same as mean absolute error, but here we are finding the distance between real data and the prediction, the smaller is better (*Evaluation metrics for your regression model 2021*). The advantage of it is being differentiable, can easily be used as a loss function/ However, it is penalizing if the dataset has outliers therefore it is not as robust as MAE (*Evaluation metrics for your regression model 2021*).

To get the score of both models, we simply call sci-kit learn library to calculate. Here is the code of two models. We will use the test data to test the score.

```
# Training, testing to plot graphs
mlp = MLPRegressor(solver='adam', random_state=42, activation="relu")
mlp.fit(X_train, Y_train)
y_pred = mlp.predict(X_test)
print("mean_squared_error:", mean_squared_error(Y_test, y_pred))
print("Mean Absolute Error:", mean_absolute_error(Y_test, y_pred))
print("r2 score:", r2_score(Y_test, y_pred))
prediction = df_today[features_to_fit]
future_price = mlp.predict(prediction)
```

```
# Training, return future price
model = LinearRegression()

model.fit(X_train, Y_train)

y_pred = model.predict(X_test)
print("mean_squared_error:", mean_squared_error(Y_test, y_pred))
print("Mean Absolute Error:", mean_absolute_error(Y_test, y_pred))
print("r2 score:", r2_score(Y_test, y_pred))
```

The result is here.

Linear Regression score:

```
mean_squared_error: 390.36847952310427
Mean Absolute Error: 5.504789115047587
r2 score: 0.9459114234421986
```

MLP Regressor model score:

```
mean_squared_error: 786.3591273180792
Mean Absolute Error: 5.9163363091517525
r2 score: 0.8891236598875829
```

As the result show after using the testing data for evaluation, linear regression model has better performance than MLP model as three score are lower than MLP model.



## Chapter 6 Discussions

### 1. Improvement and Limitation

#### Limitation

- Data problem

Current dataset is from the free public website of yahoo finance, free stock data quality is not that good due to it maybe not survivorship bias free(*Chan, 2017*). User predicting single stock have not much of a problem, but when user decide to use the machine learning to predict exchange traded fund or mutual fund than it will have problem, because those two investment product contain more than one stock. And the free data may have survivorship bias, due to it contain more than one stock. Survivorship-bias mean the tendency to evaluate the performance of current stocks or funds in the market as a representative comprehensive sample without considering those who have gone bankrupt is known as survivor bias(*Chen, 2021*).

Moreover, the free stock data may not have adjusted the split and dividend in the price(*Chen, 2021*). Lastly the price we get from yahoo finance is not the best open and close price from the auction price from the primary exchange(*Chen, 2021*). Using consolidated open or close price may inflate performance (*Chen, 2021*). Therefore the free data we are using may contribute to decrease the model performance.

- Lacking in machine learning type

In this project due to time constraint, we only explore linear regression and MLP machine learning method. We should explore different method to evaluate performance.

- Features should be more and use optimization technic to do better.

Features should include more technical indicators and use optimization technic to make it better.

#### Improvement

- Get quality data.

If we decide to further development and explore the possibility of stock prediction. We can purchase survivor-bias ship free data from data vendor for example CSI data and Quandl.

- Use different machine learning method

We should explore other different machine learning method, for example LSTM method is also very popular and Arima model. Therefore, we can evaluate different performance and decide which is more suitable for the task. For example in paper

- Need to improve feature engineering to improve model performance.

Feature engineering is to leverage data to create new variables that are not in the training. It can generate new features for both supervised and unsupervised learning, with the goal of making data transformations simpler and faster while also improving model accuracy.

For example, in paper “Short-term stock market price trend prediction using a comprehensive deep learning system” by Jingyi Shen and M. Omair Shafiq. Their attempt on machine learning model predicting short-term stock market price, their model outperform is due to their comprehensive feature engineering (*Shen & Shafiq, 2020*). Therefore we have to carefully select which feature we should use train the model to avoid downgrade of performance and complexity(*Shen & Shafiq, 2020*). In their work they use recursive feature elimination to ensure the selected feature is effective(*Shen & Shafiq, 2020*).

## Chapter 7 Conclusion

In conclusion, this report explains the whole project of “Cloud-based Machine Learning Prediction of Stock Price Website”. In the report, I have introduced the background, the method for the project and how to implement it, explain the system architecture and limitation and improvement ways for the project.

This project is my first experience on machine learning, include how to prepare data, how to select proper method to solve the task and how to train the model. I here to thank you my supervisor for giving me advice on how to do the project and direction. Learning how to use sci-kit learn to train a model and different machine learning method will be useful for my future career.

## References

Abirami, S., & Chitra, P. (2019, October 23). *Energy-efficient edge based real-time healthcare support system*. Advances in Computers. Retrieved March 30, 2022, from <https://www.sciencedirect.com/science/article/abs/pii/S0065245819300506?via%3Dihub>

Amazon. (n.d.). *AWS Cloud Development Kit*. Amazon. Retrieved April 1, 2022, from <https://aws.amazon.com/cdk/>

Amazon. (n.d.). Content Delivery Network (CDN) - Amazon CloudFront. Retrieved April 1, 2022, from <https://aws.amazon.com/cloudfront/>

*AWS CodeBuild – Fully Managed Build Service*. (n.d.). Retrieved March 31, 2022, from <https://aws.amazon.com/codebuild/>

*AWS codecommit / managed source control service*. (n.d.). Retrieved March 31, 2022, from <https://aws.amazon.com/codecommit/>

*AWS codedeploy / automated software deployment*. (n.d.). Retrieved

March 31, 2022, from <https://aws.amazon.com/codedeploy/>

AWS Editorial Team. (2008). *Theoretical cost optimization by Amazon ECS launch type: Fargate vs EC2*. Amazon. Retrieved April 1, 2022, from <https://aws.amazon.com/blogs/containers/theoretical-cost-optimization-by-amazon-ecs-launch-type-fargate-vs-ec2/>

AWS Fargate Serverless compute for containers. (2003). Retrieved April 1, 2022, from <https://aws.amazon.com/fargate/>

*Evaluation metrics for your regression model*. Analytics Vidhya. (2021, May 19). Retrieved April 2, 2022, from <https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/>

Linear Regression. (n.d.). Retrieved March 30, 2022, from <http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm>

*Machine learning in Healthcare: 12 real-world use cases – nix united*. NIX United – Custom Software Development Company in US. (2021, December 21). Retrieved March 29, 2022, from <https://nix-united.com/blog/machine-learning-in-healthcare-12-real-world-use-cases-to-know/>

*Multilayer Perceptron*. Multilayer Perceptron - an overview | ScienceDirect Topics. (n.d.). Retrieved March 30, 2022, from <https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron>

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (n.d.). *Scikit-Learn: Machine learning in Python*. *Journal of Machine Learning Research*. Retrieved April 1, 2022, from <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>

*Running python functions as pipeline jobs*. Joblib. (n.d.). Retrieved April

1, 2022, from <https://joblib.readthedocs.io/en/latest/>

(n.d.).(n.d.). Amazon Elastic Container Registry (Amazon ECR). Retrieved April 1, 2022, from <https://aws.amazon.com/ecr/>

*Sklearn.neural\_network.Mlpregressor*. scikit. (n.d.). Retrieved April 1, 2022, from [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPRegressor.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html)

Soni, P., Tewari, Y., & Krishnan, D. (2022). Machine learning approaches in stock price prediction: A systematic review. *Journal of Physics: Conference Series*, 2161(1), 012065. <https://doi.org/10.1088/1742-6596/2161/1/012065>

Techopedia. (2017, March 30). *What is a multilayer Perceptron (MLP)? - definition from Techopedia*. Techopedia.com. Retrieved March 30, 2022, from <https://www.techopedia.com/definition/20879/multilayer-perceptron-mlp>

What is AWS . (n.d.). Retrieved from [https://aws.amazon.com/what-is-aws/?nc1=f\\_cc](https://aws.amazon.com/what-is-aws/?nc1=f_cc)

*What is machine learning? A definition*. Expert.ai. (2022, March 14). Retrieved March 29, 2022, from <https://www.expert.ai/blog/machine-learning-definition/>

Chen, J. (2021, October 31). *Survivorship bias definition*. Investopedia. Retrieved April 2, 2022, from <https://www.investopedia.com/terms/s/survivorshipbias.asp>

Chan, E. P. (2017). *Machine trading: Deploying computer algorithms to conquer the markets*. John Wiley & Sons, Inc.

Shen, J., & Shafiq, M. O. (2020). Short-term stock market price trend prediction using a comprehensive deep learning system. *Journal of Big Data*, 7(1). <https://doi.org/10.1186/s40537-020-00333-6>

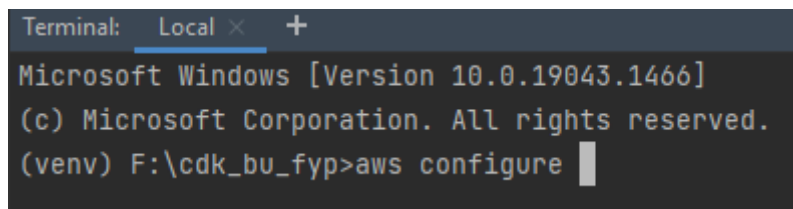
*Amazon Aurora. (n.d.). Retrieved April 4, 2022, from <https://aws.amazon.com/rds/aurora/>*

## **APPENDICES**

### **1. System Deployment Guide**

Please first register an AWS account, install aws cli and docker for deployment.

1. In command line type "aws configure" to configure your AWS account first.



```
Terminal: Local x +
Microsoft Windows [Version 10.0.19043.1466]
(c) Microsoft Corporation. All rights reserved.
(venv) F:\cdk_bu_fyp>aws configure
```

2. Please enter your AWS account ID and the region you want to deploy in CDK code folder, app.py, in the env variable, account parameter enter your account ID and region enter your desired region.

```
from aws_cdk import core
from cdk.vpc_construct.fyp_vpc import VPC_stack
from cdk.Database.Auroa_Serverless import Database_Stack
from cdk.Pipeline.Pipeline_Construct import PipelineConstruct
from cdk.Cloudfront.cloudfront_cache import cloudfront_dist
import os

app = core.App()
env = core.Environment(account="Your account ID", region="Your desired region")

vpc_construct = VPC_stack(
    app,
    "infraVpc",
    env=env
)

database_construct = Database_Stack(
    app,
    "auroraDB",
    props=vpc_construct.vpc,
    env=env
)

database_construct.add_dependency(vpc_construct)

pipeline_construct = PipelineConstruct(
    app,
    "CodePipelineFlask",
    env=env,
    props=vpc_construct.vpc
)

pipeline_construct.add_dependency(database_construct)

cloudfront = cloudfront_dist(
    app,
    "CloudFrontELB",
    env=env,
    props=pipeline_construct.pipeline_construct
)
```

3. In CDK folder ,please enter pip install -r requirement.txt in CDK folder to install all necessary dependency.

```
(venv) F:\cdk_bu_fyp>pip install -r requirements.txt
Requirement already satisfied: attrs==21.2.0 in f:\cdk_bu_fyp\venv\lib\site-packages (from -r requirements.txt (line 1)) (21.2.0)
Requirement already satisfied: aws-cdk.assets in f:\cdk_bu_fyp\venv\lib\site-packages (from -r requirements.txt (line 2)) (1.149.0)
Requirement already satisfied: aws-cdk.aws-acmpca in f:\cdk_bu_fyp\venv\lib\site-packages (from -r requirements.txt (line 3)) (1.149.0)
```

4. After installing all dependency, please enter `cdk deploy --all` in command line to deploy the cloud architecture.

```
(venv) F:\cdk_bu_ftp>cdk deploy --all
* Synthesis time: 8.34s

InfraVpc
InfraVpc: deploying...
InfraVpc: creating CloudFormation changeset...
InfraVpc | 0/25 | 7:20:45 PM | REVIEW_IN_PROGRESS | AWS::CloudFormation::Stack | InfraVpc User Initiated
InfraVpc | 0/25 | 7:20:50 PM | CREATE_IN_PROGRESS | AWS::CloudFormation::Stack | InfraVpc User Initiated
InfraVpc | 0/25 | 7:20:54 PM | CREATE_IN_PROGRESS | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
InfraVpc | 0/25 | 7:20:55 PM | CREATE_IN_PROGRESS | AWS::EC2::EIP | Infra_VPC/Public-SubnetSubnet1/EIP (InfraVPCPublicSubnetSubnet1EIP94983810)
InfraVpc | 0/25 | 7:20:55 PM | CREATE_IN_PROGRESS | AWS::EC2::VPC | Infra_VPC (InfraVPCB0239CA5)
InfraVpc | 0/25 | 7:20:55 PM | CREATE_IN_PROGRESS | AWS::EC2::InternetGateway | Infra_VPC/IGW (InfraVPCIGW06778985)
InfraVpc | 0/25 | 7:20:55 PM | CREATE_IN_PROGRESS | AWS::EC2::EIP | Infra_VPC/Public-SubnetSubnet2/EIP (InfraVPCPublicSubnetSubnet2EIP0F9CF0C4)
InfraVpc | 0/25 | 7:20:55 PM | CREATE_IN_PROGRESS | AWS::EC2::VPC | Infra_VPC (InfraVPCB0239CA5) Resource creation Initiated
InfraVpc | 0/25 | 7:20:56 PM | CREATE_IN_PROGRESS | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata) Resource creation Initiated
InfraVpc | 1/25 | 7:20:56 PM | CREATE_COMPLETE | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
InfraVpc | 1/25 | 7:20:56 PM | CREATE_IN_PROGRESS | AWS::EC2::InternetGateway | Infra_VPC/IGW (InfraVPCIGW06778985) Resource creation Initiated
InfraVpc | 1/25 | 7:21:10 PM | CREATE_IN_PROGRESS | AWS::EC2::EIP | Infra_VPC/Public-SubnetSubnet1/EIP (InfraVPCPublicSubnetSubnet1EIP94983810) Resource creation Initiated
InfraVpc | 1/25 | 7:21:10 PM | CREATE_IN_PROGRESS | AWS::EC2::EIP | Infra_VPC/Public-SubnetSubnet2/EIP (InfraVPCPublicSubnetSubnet2EIP0F9CF0C4) Resource creation Initiated
InfraVpc | 2/25 | 7:21:11 PM | CREATE_COMPLETE | AWS::EC2::EIP | Infra_VPC/Public-SubnetSubnet1/EIP (InfraVPCPublicSubnetSubnet1EIP94983810)
InfraVpc | 3/25 | 7:21:11 PM | CREATE_COMPLETE | AWS::EC2::EIP | Infra_VPC/Public-SubnetSubnet2/EIP (InfraVPCPublicSubnetSubnet2EIP0F9CF0C4)
InfraVpc | 4/25 | 7:21:12 PM | CREATE_COMPLETE | AWS::EC2::VPC | Infra_VPC (InfraVPCB0239CA5)
InfraVpc | 5/25 | 7:21:13 PM | CREATE_COMPLETE | AWS::EC2::InternetGateway | Infra_VPC/IGW (InfraVPCIGW06778985)
InfraVpc | 6/25 | 7:21:13 PM | CREATE_IN_PROGRESS | AWS::EC2::RouteTable | Infra_VPC/Public-SubnetSubnet1/RouteTable (InfraVPCPublicSubnetSubnet1RouteTable05788CA2)
InfraVpc | 7/25 | 7:21:13 PM | CREATE_IN_PROGRESS | AWS::EC2::RouteTable | Infra_VPC/Public-SubnetSubnet2/RouteTable (InfraVPCPublicSubnetSubnet2RouteTable0546C3048)
InfraVpc | 8/25 | 7:21:13 PM | CREATE_IN_PROGRESS | AWS::EC2::RouteTable | Infra_VPC/Private-SubnetSubnet1/RouteTable (InfraVPCPrivateSubnetSubnet1RouteTable040EBED4)
InfraVpc | 9/25 | 7:21:13 PM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Infra_VPC/Private-SubnetSubnet2/Subnet (InfraVPCPrivateSubnetSubnet2Subnet0B28BF13)
InfraVpc | 10/25 | 7:21:13 PM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Infra_VPC/Public-SubnetSubnet2/Subnet (InfraVPCPublicSubnetSubnet2Subnet06872505)
InfraVpc | 11/25 | 7:21:13 PM | CREATE_IN_PROGRESS | AWS::EC2::RouteTable | Infra_VPC/Private-SubnetSubnet1/RouteTable (InfraVPCPrivateSubnetSubnet1RouteTable0478BC54)
InfraVpc | 12/25 | 7:21:13 PM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Infra_VPC/Private-SubnetSubnet1/Subnet (InfraVPCPrivateSubnetSubnet1Subnet0ADEAC91C)
InfraVpc | 13/25 | 7:21:14 PM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Infra_VPC/Public-SubnetSubnet1/Subnet (InfraVPCPublicSubnetSubnet1Subnet0F08168)
InfraVpc | 14/25 | 7:21:15 PM | CREATE_IN_PROGRESS | AWS::EC2::RouteTable | Infra_VPC/Public-SubnetSubnet1/RouteTable (InfraVPCPublicSubnetSubnet1RouteTable05788CA2) Resource creation Initiated
InfraVpc | 15/25 | 7:21:15 PM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Infra_VPC/Private-SubnetSubnet2/Subnet (InfraVPCPrivateSubnetSubnet2Subnet0B28BF13) Resource creation Initiated
InfraVpc | 16/25 | 7:21:15 PM | CREATE_IN_PROGRESS | AWS::EC2::RouteTable | Infra_VPC/Public-SubnetSubnet2/RouteTable (InfraVPCPublicSubnetSubnet2RouteTable0546C3048) Resource creation Initiated
InfraVpc | 17/25 | 7:21:15 PM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Infra_VPC/Private-SubnetSubnet1/Subnet (InfraVPCPrivateSubnetSubnet1Subnet0ADEAC91C) Resource creation Initiated
```

5. type y and press enter to deploy the database

```
* Total time: 208.91s

auroraDB
This deployment will make potentially sensitive changes according to your current security approval level (--require-approval broadening).
Please confirm you intend to make the following modifications:

Security Group Changes


|   | Group                           | Dir | Protocol   | Peer            |
|---|---------------------------------|-----|------------|-----------------|
| + | \${AuroraSecurityGroup.GroupId} | In  | TCP 3306   | Everyone (IPv4) |
| + | \${AuroraSecurityGroup.GroupId} | Out | Everything | Everyone (IPv4) |


(NOTE: There may be security-related changes not in this list. See https://github.com/aws/aws-cdk/issues/1299)

Do you wish to deploy these changes (y/n)?
```

```
Do you wish to deploy these changes (y/n)? y
auroraDB: deploying...
auroraDB: creating CloudFormation changeset...
auroraDB | 0/7 | 7:24:47 PM | REVIEW_IN_PROGRESS | AWS::CloudFormation::Stack | auroraDB User Initiated
auroraDB | 0/7 | 7:24:53 PM | CREATE_IN_PROGRESS | AWS::CloudFormation::Stack | auroraDB User Initiated
auroraDB | 0/7 | 7:24:57 PM | CREATE_IN_PROGRESS | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
auroraDB | 0/7 | 7:24:57 PM | CREATE_IN_PROGRESS | AWS::RDS::DBSubnetGroup | AuroraCluster/Subnets/Default (AuroraClusterSubnetsF3E9E6AD)
auroraDB | 0/7 | 7:24:58 PM | CREATE_IN_PROGRESS | AWS::EC2::SecurityGroup | AuroraSecurityGroup (AuroraSecurityGroup75F699F6)
auroraDB | 0/7 | 7:24:58 PM | CREATE_IN_PROGRESS | AWS::SecretsManager::Secret | AuroraCluster/Secret (AuroraClusterSecret8E4F28C8)
auroraDB | 0/7 | 7:24:58 PM | CREATE_IN_PROGRESS | AWS::RDS::DBSubnetGroup | AuroraCluster/Subnets/Default (AuroraClusterSubnetsF3E9E6AD) Resource creation Initiated
```

6. After Aurora Database have complete please go to AWS console, and search for Secrets Manager, in the Secrets Manager click AuroraClusterSecert, in there please click retrieve secret, and copy the host, username and password.



```
Do you wish to deploy these changes (y/n)? y+
Invalid choice: y+
Do you wish to deploy these changes (y/n)? y
auroraDB: deploying...
auroraDB: creating CloudFormation changeset...
auroraDB | 0/7 | 7:24:47 PM | REVIEW_IN_PROGRESS | AWS::CloudFormation::Stack | auroraDB User Initiated
auroraDB | 0/7 | 7:24:53 PM | CREATE_IN_PROGRESS | AWS::CloudFormation::Stack | auroraDB User Initiated
auroraDB | 0/7 | 7:24:57 PM | CREATE_IN_PROGRESS | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
auroraDB | 0/7 | 7:24:57 PM | CREATE_IN_PROGRESS | AWS::RDS::DBSubnetGroup | AuroraCluster/Subnets/Default (AuroraClusterSubnetsF3E9E6AD)
auroraDB | 0/7 | 7:24:58 PM | CREATE_IN_PROGRESS | AWS::EC2::SecurityGroup | AuroraSecurityGroup (AuroraSecurityGroup78F699F6)
auroraDB | 0/7 | 7:24:58 PM | CREATE_IN_PROGRESS | AWS::SecretsManager::Secret | AuroraCluster/Secret (AuroraClusterSecret8E4F2BC8)
auroraDB | 0/7 | 7:24:58 PM | CREATE_IN_PROGRESS | AWS::RDS::DBSubnetGroup | AuroraCluster/Subnets/Default (AuroraClusterSubnetsF3E9E6AD) Resource creation Initiated
auroraDB | 0/7 | 7:24:59 PM | CREATE_IN_PROGRESS | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata) Resource creation Initiated
auroraDB | 1/7 | 7:24:59 PM | CREATE_COMPLETE | AWS::RDS::DBSubnetGroup | AuroraCluster/Subnets/Default (AuroraClusterSubnetsF3E9E6AD)
auroraDB | 2/7 | 7:24:59 PM | CREATE_COMPLETE | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
auroraDB | 2/7 | 7:24:59 PM | CREATE_IN_PROGRESS | AWS::SecretsManager::Secret | AuroraCluster/Secret (AuroraClusterSecret8E4F2BC8) Resource creation Initiated
auroraDB | 3/7 | 7:25:00 PM | CREATE_COMPLETE | AWS::EC2::SecurityGroup | AuroraSecurityGroup (AuroraSecurityGroup78F699F6) Resource creation Initiated
auroraDB | 3/7 | 7:25:02 PM | CREATE_IN_PROGRESS | AWS::EC2::SecurityGroup | AuroraSecurityGroup (AuroraSecurityGroup78F699F6) Resource creation Initiated
auroraDB | 4/7 | 7:25:04 PM | CREATE_COMPLETE | AWS::RDS::DBCluster | AuroraCluster (AuroraCluster23D869C0)
auroraDB | 4/7 | 7:25:06 PM | CREATE_IN_PROGRESS | AWS::RDS::DBCluster | AuroraCluster (AuroraCluster23D869C0) Resource creation Initiated
auroraDB | 4/7 | 7:25:08 PM | CREATE_IN_PROGRESS | AWS::RDS::DBCluster | AuroraCluster (AuroraCluster23D869C0)
4/7 Currently in progress: auroraDB, AuroraCluster23D869C0
auroraDB | 5/7 | 7:28:41 PM | CREATE_COMPLETE | AWS::RDS::DBCluster | AuroraCluster (AuroraCluster23D869C0)
auroraDB | 5/7 | 7:28:42 PM | CREATE_IN_PROGRESS | AWS::SecretsManager::SecretTargetAttachment | AuroraCluster/Secret/Attachment (AuroraClusterSecretAttachmentD88032DA)
auroraDB | 5/7 | 7:28:44 PM | CREATE_IN_PROGRESS | AWS::SecretsManager::SecretTargetAttachment | AuroraCluster/Secret/Attachment (AuroraClusterSecretAttachmentD88032DA) Resource creation Initiated
auroraDB | 6/7 | 7:28:44 PM | CREATE_COMPLETE | AWS::SecretsManager::SecretTargetAttachment | AuroraCluster/Secret/Attachment (AuroraClusterSecretAttachmentD88032DA)
auroraDB | 7/7 | 7:28:45 PM | CREATE_COMPLETE | AWS::CloudFormation::Stack | auroraDB
Deployment time: 241.97s
```

AWS Secrets Manager

AWS Secrets Manager > Secrets

Secrets

Secrets

Filter secrets by name, description, tag key, tag value, or primary Region

1

Store a new secret

Secret name	Description	Last retrieved (UTC)
AuroraClusterSecret8E4F2BC8-2FnL41HVXMj	Generated by the CDK for stack: auroraDB	4/2/2022

Secret value

Retrieve and view the secret value.

Close Edit

Key/value

Plaintext

Secret key	Secret value
dbClusterIdentifier	auroradb-auroracluster23d869c0-g2kpuhzw35rl
password	b.d3ZFDNfAdtaXr-EW3W=OA5G1oOFF
engine	mysql
port	3306
host	auroradb-auroracluster23d869c0-g2kpuhzw35rl.cluster-cssae9afjapw.ap-southeast-1.rds.amazonaws.com
username	admin

7. Please enter username, password and hostname in the code folder, config.py at SQLALCHEMY\_DATABASE\_URI\

```
class Config(object):
    DEBUG = True
    SECRET_KEY = "YTGLBygn6C"
    WTF_CSRF_SECRET_KEY = "I7dz4PD7T6"
    STATIC_FOLDER = 'machine_learning_hkbu/app/static'
    # database URI
    SQLALCHEMY_DATABASE_URI = 'mysql+pymysql://user_name:password@host_name/sys'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

8. After entering the database uri. In the code folder type, pip install -r requirements.txt to install all necessary dependency and “docker build -t ‘your\_image\_name’:‘you\_tag’ .” to build the docker image for deployment

```
(venv) F:\machine_learning_hkbu>docker build -t ml_website:latest .
[+] Building 7.7s (6/17)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/smizy/scikit-learn:latest
=> [ 1/13] FROM docker.io/smizy/scikit-learn:latest@sha256:4a174d02b8096f1ce9037a9b2d9ce91579977cbfb423f6238bc8228f14d72113
=> [internal] load build context
=> => transferring context: 2.60MB
=> CACHED [ 2/13] WORKDIR /app
=> [ 3/13] ADD . /app
```

9. Please click yes to build the pipeline and ECS cluster.

+	*	Allow	ecr:BatchCheckLayerAvailability ecr:BatchGetImage ecr:GetAuthorizationToken ecr:GetDownloadUrlForLayer logs:CreateLogStream logs:PutLogEvents
+	*	Allow	ecr:GetAuthorizationToken
+	arn:\${AWS::Partition}:codebuild:ap-southeast-1:169747529889:report-group/\${BuildDocker28279A32}-*	Allow	codebuild:BatchPutCodeCoverages codebuild:BatchPutTestCases codebuild:CreateReport codebuild:CreateReportGroup codebuild:UpdateReport
+	arn:\${AWS::Partition}:codepipeline:ap-southeast-1:169747529889:\${FlaskAppPipeline41691881}	Allow	codepipeline:StartPipelineExecution
+	arn:\${AWS::Partition}:ecr:ap-southeast-1:169747529889:repository/*	Allow	ecr:BatchCheckLayerAvailability ecr:BatchGetImage ecr:GetDownloadUrlForLayer
+	arn:\${AWS::Partition}:logs:ap-southeast-1:169747529889:log-group:/aws/codebuild/\${BuildDocker28279A32} arn:\${AWS::Partition}:logs:ap-southeast-1:169747529889:log-group:/aws/codebuild/\${BuildDocker28279A32}:*	Allow	logs:CreateLogGroup logs:CreateLogStream logs:PutLogEvents

**Security Group Changes**

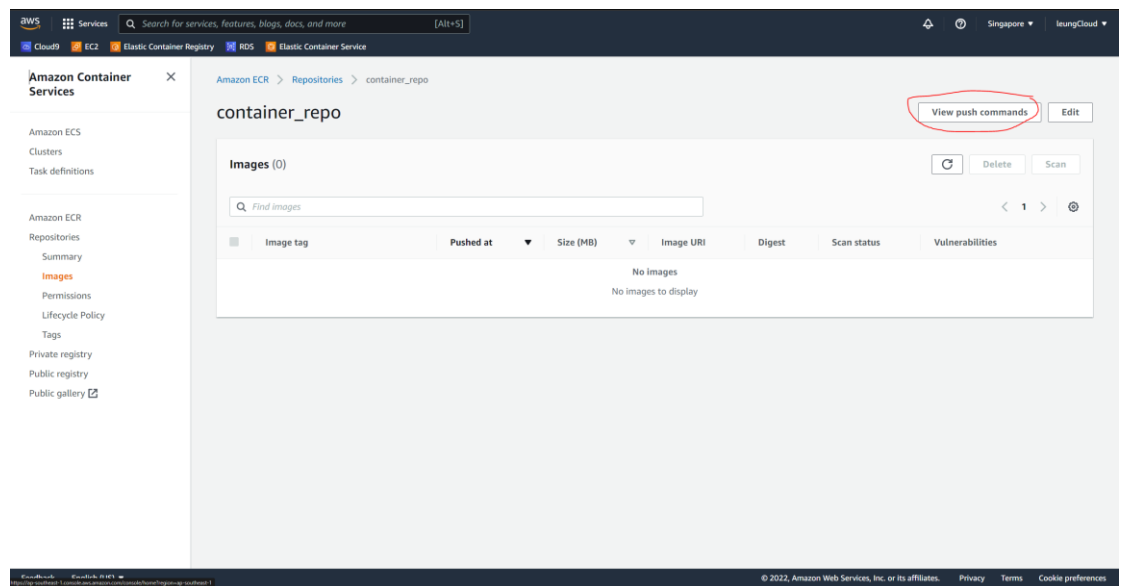
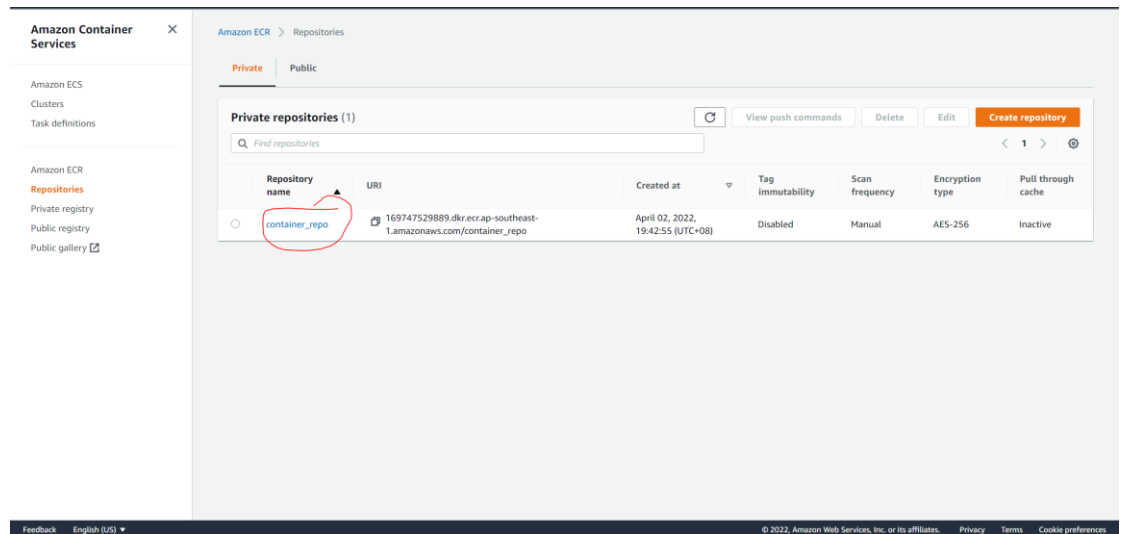
	Group	Dir	Protocol	Peer
+	\${ECS-FARGATE-SERVICE/SecurityGroup.GroupId}	In	TCP 8000	\${LB_FOR_FARGATE/SecurityGroup.GroupId}
+	\${ECS-FARGATE-SERVICE/SecurityGroup.GroupId}	Out	Everything	Everyone (IPv4)
+	\${LB_FOR_FARGATE/SecurityGroup.GroupId}	In	TCP 8000	Everyone (IPv4)
+	\${LB_FOR_FARGATE/SecurityGroup.GroupId}	Out	TCP 8000	\${ECS-FARGATE-SERVICE/SecurityGroup.GroupId}

(NOTE: There may be security-related changes not in this list. See <https://github.com/aws/aws-cdk/issues/1299>)

Do you wish to deploy these changes (y/n)? ☐

```
CodePipelineFlask: deploying...
CodePipelineFlask: creating CloudFormation changeset...
CodePipelineFlask | 0/41 | 7:42:34 PM | REVIEW_IN_PROGRESS | AWS::CloudFormation::Stack | CodePipelineFlask User Initiated
CodePipelineFlask | 0/41 | 7:42:45 PM | CREATE_IN_PROGRESS | AWS::CloudFormation::Stack | CodePipelineFlask User Initiated
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::IAM::Role | Flask_App_Pipeline/Role (FlaskAppPipelineRole2FC2E034)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::ElasticLoadBalancingV2::TargetGroup | LB_FOR_FARGATE/Listener/ECS-FARGATEGroup (LBFORFARGATEListenerECSFARGATEGroup20410370)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::EC2::SecurityGroup | LB_FOR_FARGATE/SecurityGroup (LBFORFARGATESecurityGroup23776747)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::IAM::Role | Flask_App_Pipeline/deploy_to_ecs/Deploy_to_ECS/CodePipelineActionRole (FlaskAppPipelineDeploytoecsDeploytoECS)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::CodeDeploy::DeploymentConfig | MycfnDeploymentConfig
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::ECS::Cluster | Cluster (ClusterEB0386A7)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::IAM::Role | Build Docker/Role (BuildDockerRoleC5DC9F1D)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::IAM::Role | Flask_App_Pipeline/EventsRole (FlaskAppPipelineEventsRoleFF079903)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::IAM::Role | Flask_App_Pipeline/source/get_source_from_codecommit/CodePipelineActionRole (FlaskAppPipelineSourcegetsourcefromcodecommit)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::IAM::Role | TaskDef/ExecutionRole (TaskDefExecutionRole84775C97)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::KMS::Key | Flask_App_Pipeline/ArtifactsBucketEncryptionKey (FlaskAppPipelineArtifactsBucketEncryptionKey48BD0C2F)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::IAM::Role | Flask_App_Pipeline/DockerBuild/Build_docker_and_push_to_ECR/CodePipelineActionRole (FlaskAppPipelineDockerBuild)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::CodeDeploy::Application | FLASK_APPLICATION (FLASKAPPLICATIONAFCEA358)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::ECR::Repository | container_repo (containerrepoC8B46920)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::EC2::SecurityGroup | ECS-FARGATE-SERVICE/SecurityGroup (ECSFARGATESERVICESecurityGroup148136CE)
CodePipelineFlask | 0/41 | 7:42:52 PM | CREATE_IN_PROGRESS | AWS::CodeCommit::Repository | flask-repo (flaskrepo2CE2936)
```

10. In the AWS console please type Elastic Container Registry and go to container\_repo we created. Click view push command and follow the instruction to upload the docker image to the registry.



11. first we enter

“aws ecr get-login-password --region ap-southeast-1 | docker  
login --username AWS --password-stdin  
“your\_account\_ID.dkr.ecr.ap-southeast-1.amazonaws.com”  
to login to the repository.

```
(venv) F:\machine_learning_hkbu>aws ecr get-login-password --region ap-southeast-1 | docker login --username AWS --password-stdin 169747529889.dkr.ecr.ap-southeast-1.amazonaws.com  
Login Succeeded
```

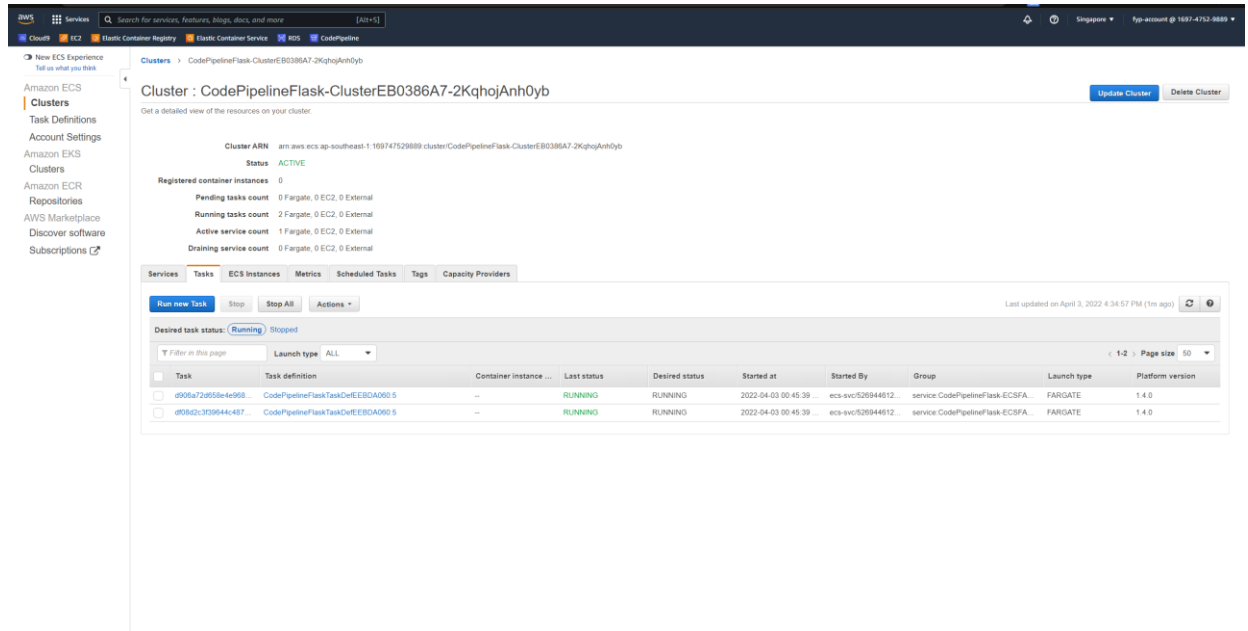
12. After that we will retag the image using the command “docker  
tag your\_container\_name:latest 169747529889.dkr.ecr.ap-  
southeast-1.amazonaws.com/container\_repo:latest”

```
(venv) F:\machine_learning_hkbu>docker tag ml_website:latest 169747529889.dkr.ecr.ap-southeast-1.amazonaws.com/container_repo:latest
```

13. enter docker push command provided by the ecr repo to push the  
image.

```
(venv) F:\machine_learning_hkbu>docker push 169747529889.dkr.ecr.ap-southeast-1.amazonaws.com/container_repo:latest  
The push refers to repository [169747529889.dkr.ecr.ap-southeast-1.amazonaws.com/container_repo]  
3b207b84fb49: Pushed  
4ad7dce5bafc: Pushed  
3fd2a4b67f22: Pushed  
fdab5dd512ae: Pushed  
75bb68426ebc: Pushed  
27a031e14058: Pushed  
a33adeeb2173: Pushed  
9bd3b3b2bb61: Pushed  
9efc3b77faa8: Pushed  
82eae9d39791: Pushed  
bfe859d5cc8b: Pushed  
8d6732e6fceb: Pushed  
33929e5a3f42: Pushed  
69525a95d291: Pushed  
e4b4c921cfcb: Pushed  
99d461f1e50a: Pushed  
be01fd55114f: Pushed  
6977a6d0f134: Pushed  
8e207b409db3: Pushed  
latest: digest: sha256:b4d39737b7cbdef51d824706c463a34aeb009054c0551b6de061fde348372801 size: 4303
```

14. After you push the docker to the ECR repo, the ECS cluster will get the image from the repo and start deployment of the website. After that it will successfully deploy with two task and able to access the website.



Cluster : CodePipelineFlask-ClusterEB0386A7-2KqhojAnh0yb

Get a detailed view of the resources on your cluster.

Cluster ARN: arn:aws:ecs:ap-southeast-1:169747529689:cluster:CodePipelineFlask-ClusterEB0386A7-2KqhojAnh0yb

Status: ACTIVE

Registered container instances: 0

Pending tasks count: 0 Fargate, 0 EC2, 0 External

Running tasks count: 2 Fargate, 0 EC2, 0 External

Active service count: 1 Fargate, 0 EC2, 0 External

Draining service count: 0 Fargate, 0 EC2, 0 External

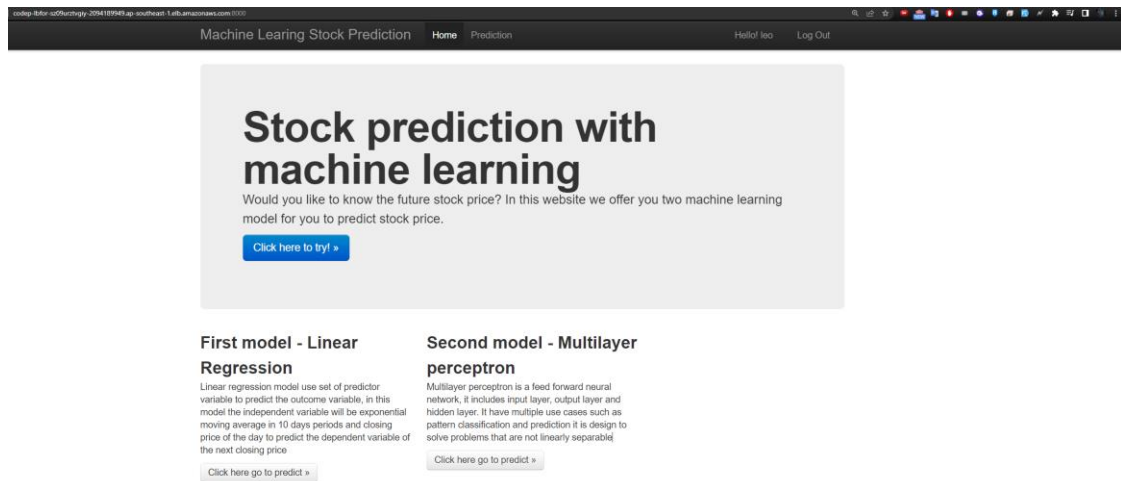
Services | **Tasks** | ECS Instances | Metrics | Scheduled Tasks | Tags | Capacity Providers

Run new Task | Stop | Stop All | Actions

Desired task status: Running | Stopped

Filter in this page | Launch type: ALL

Task	Task definition	Container instance	Last status	Desired status	Started at	Started By	Group	Launch type	Platform version
d905a72d558e4e968...	CodePipelineFlaskTaskDefEEBCA0605	--	RUNNING	RUNNING	2022-04-03 00:45:39	ecs-ecv926944812...	service:CodePipelineFlask-EC2SFA...	FARGATE	1.4.0
d0862c3f5944c487...	CodePipelineFlaskTaskDefEEBCA0605	--	RUNNING	RUNNING	2022-04-03 00:45:39	ecs-ecv926944812...	service:CodePipelineFlask-EC2SFA...	FARGATE	1.4.0



Machine Learning Stock Prediction | Home | Prediction

Hello! leo | Log Out

## Stock prediction with machine learning

Would you like to know the future stock price? In this website we offer you two machine learning model for you to predict stock price.

[Click here to try!](#)

### First model - Linear Regression

Linear regression model use set of predictor variable to predict the outcome variable, in this model the independent variable will be exponential moving average in 10 days periods and closing price of the day to predict the dependent variable of the next closing price

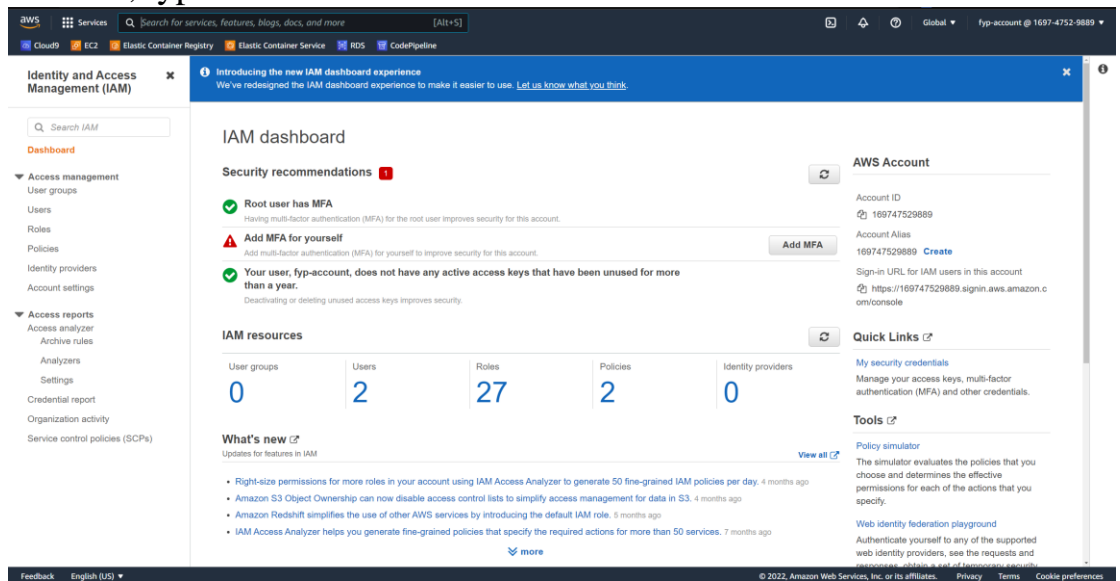
[Click here go to predict](#)

### Second model - Multilayer perceptron

Multilayer perceptron is a feed forward neural network, it includes input layer, output layer and hidden layer. It have multiple use cases such as pattern classification and prediction it is design to solve problems that are not linearly separable

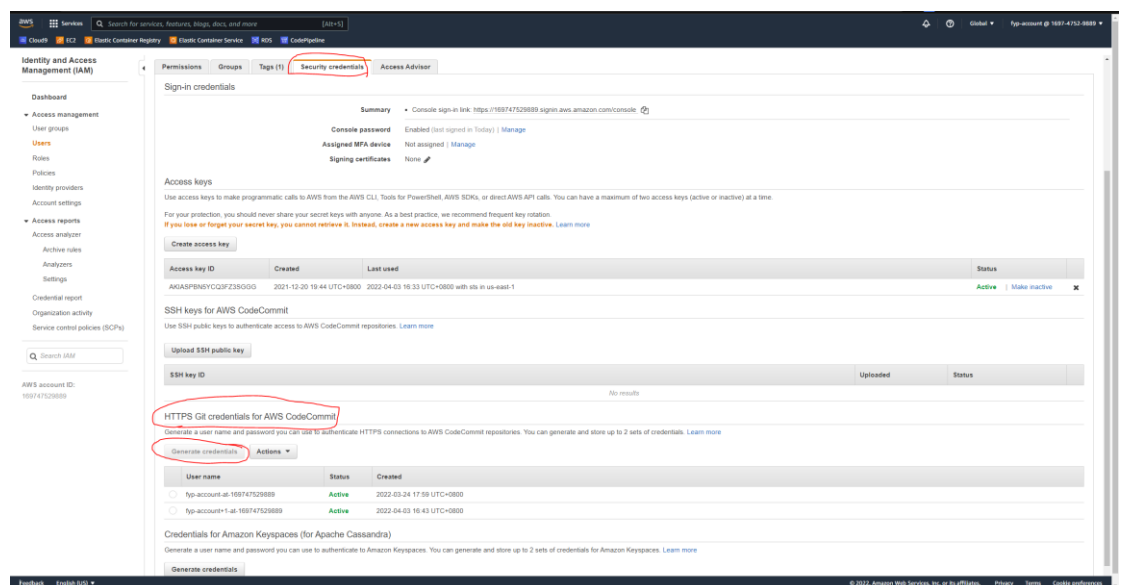
[Click here go to predict](#)

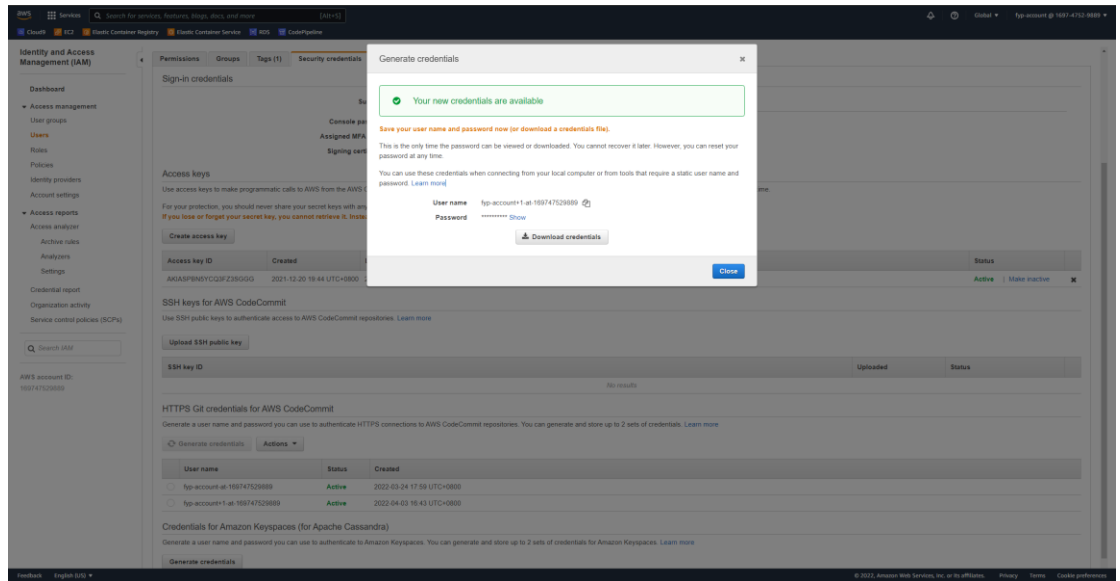
15. To upload code to the code commit repo, first go to AWS console, type IAM.



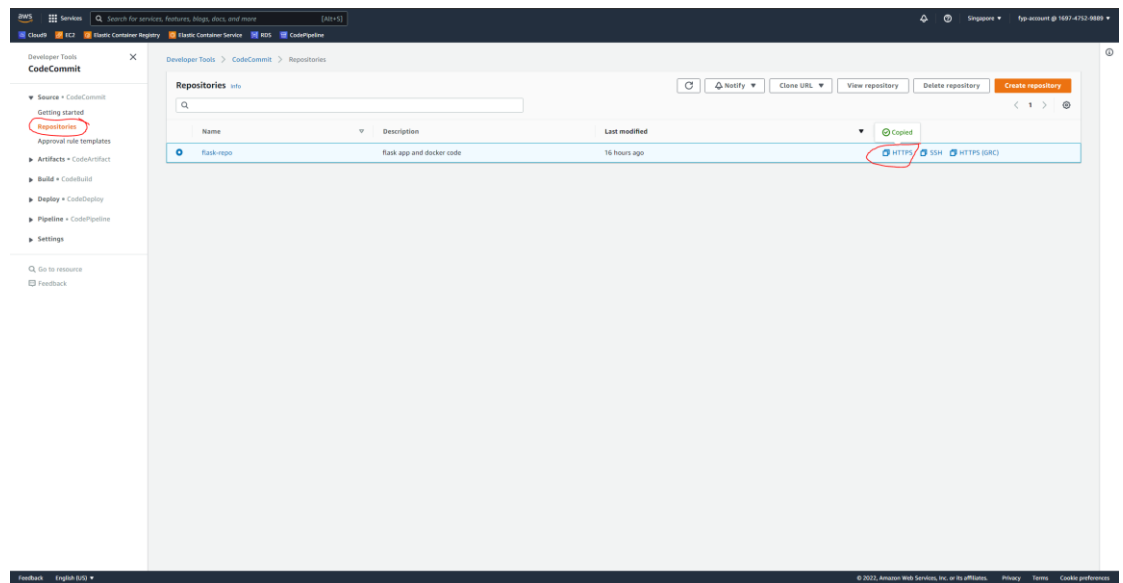
16. In IAM section click “User” and click your user.

17. Go to security credentials, in the section “HTTPS Git credentials for AWS CodeCommit”, Click “Generate credentials”. And note down the username and password.

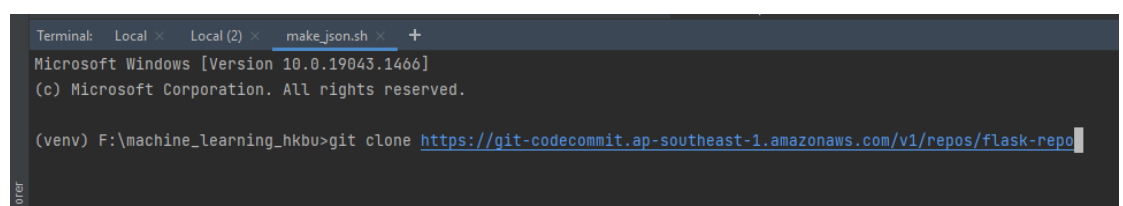




18. In CodeCommit page click the repository, click HTTPS to clone the repository.



19. in command line type git clone “your repository url”





20. in command line type git remote add origin “your repository url”

```
(venv) F:\machine_learning_hkbu>git remote add origin https://git-codecommit.ap-southeast-1.amazonaws.com/v1/repos/flask-repo
```

21. after you add the file and committed. use git push origin master to push to the repository.

```
(venv) F:\machine_learning_hkbu>git push origin master
```

22. Successfully pushed.

