# analyticsC apstone project

Le Zhou - MS of Data Science
Dr. Christelle Scharff

lz25261n@pace.edu

https://github.com/50dollarsbuddy/CS668_capstone

# A prediction model that identify how many R&D cost is good for IBM's stock price

# Research and development(R&D)

A direct expenditures relating to a company's efforts to develop, design, and enhance its products, services, technologies or processes

**Amazon (AMZN), $42.74 billion**

**Alphabet (GOOG, GOOGL), $27.57 billion**

**Microsoft (MSFT), $19.27 billion**

# Introduction

Accurate prediction is hard in chaotic commercial system. However, the logic of analysing financial report give a new idea to revisit the process that people evaluate a company by accounting experience. Accounting is the process of recording financial transactions pertaining to a business and historic standard which is designed by exports. We could build a relation to quickly analysis their accounting's change. It is hard to find an explainable mathematics model between R&D and company's developing potential, but it is reasonable for the relation and exploring its application.

# Personal motivation

1. IBM's identity(top 500, high technology, dow jones index)
2. R&D management
3. A relationship between R&D and IBM's stock price(value and future)

# Literature review

1. K-Nearest Neighbor regression with Principal component analysis for financial Time series Prediction
   (1)PCA for reducing redundancy information and extracting essential features
   (2)set data set by time series
   (3) K-NN regression

2. A machine learning Aproach for Stock Price Prediction
   (1)structural support vector machines(SSVMs) - non-linearly separated data
   (2)Data mining and machine learning approaches can be incorporated into business intelligence(BI) systems to help users for decision support

3. Stock Price Predictiion Bases on Machine Learning Approaches
   (1)Logisttic regression + support vector machine

# Summary

**Data set**

Financial data need a suitable format

01.

02.

**variable(features)**

Focus on principal component

**model**

Non-linear issues
cross validation
Complex model

03.

04.

**evaluation**

R-Squared
MSE
MAE

# Solution

01. Time series analysis

02. R*D, Nasdaq_value, IBM's stock price

03. Random forest, decision tree Voting regression

04. evaluation

R-Squared
MSE
MAE

# Build a forecast model between R&D and stock price

**Feature**

- R&D(research and development expense)
- Nasdaq index - reflect the whole market's trend
- IBM's stock price

**Model**

- Decision tree regression
- Random forest regression

**Optimize Model**

- Ensemble method
- Voting  regression

**Data format**

- Time series data -  it is a collection of observations obtained through repeated measurement over time.

**The result will indicate the relation between R&D and IBM's stock price, which remind users to have a better budget in R&D for next year**

# Data set

# 1. R&D from IBM's financial report(quarterly)

IBM's financial report – [Yahoo Finance]

```
df
```

| | name | ttm | 06/30/2021 | 03/31/2021 |
|---|---|---|---|---|
| 0 | TotalRevenue | 74402000000.00 | 18745000000.00 | 17729000000.00 |
| 1 | \tOperatingRevenue | 74402000000.00 | 18745000000.00 | 17729000000.00 |
| 2 | CostOfRevenue | 38241000000.00 | 9741000000.00 | 9525000000.00 |
| 3 | GrossProfit | 36161000000.00 | 9004000000.00 | 8204000000.00 |
| 4 | OperatingExpense | 28102000000.00 | 6857000000.00 | 6658000000.00 |
| ... | ... | ... | ... | ... |
| 56 | TotalUnusualItemsExcludingGoodwill | (270000000.00) | (65000000.00) | (45000000.00) |
| 57 | TotalUnusualItems | (270000000.00) | (65000000.00) | (45000000.00) |
| 58 | NormalizedEBITDA | 13777000000.00 | 3578000000.00 | 2902000000.00 |
| 59 | TaxRateForCalcs | 0.06 | 0.15 | 0.40 |
| 60 | TaxEffectOfUnusualItems | (15990667.62) | (9555000.00) | (18000000.00) |

61 rows × 146 columns

```
df = df.T
df = df.drop('ttm')
df =df.drop('name')
df_RD = df.iloc[:,10]
df_RD = df_RD.dropna()
data = {'Date':df_RD.index,
        'R&D':df_RD.values}
df_RD = pd.DataFrame(data)
```

Feature: date, R&D(109 rows)

```
df_RD
```

| | Date | R&D |
|---|---|---|
| 0 | 06/30/2021 | 1657000000.00 |
| 1 | 03/31/2021 | 1630000000.00 |
| 2 | 12/31/2020 | 1611000000.00 |
| 3 | 09/30/2020 | 1515000000.00 |
| 4 | 06/30/2020 | 1582000000.00 |
| ... | ... | ... |
| 104 | 06/30/1995 | 9.74e+08 |
| 105 | 03/31/1995 | 9.13e+08 |
| 106 | 12/31/1994 | 1118000000.00 |
| 107 | 09/30/1994 | 1.053e+09 |
| 108 | 06/30/1994 | 1.091e+09 |

109 rows × 2 columns

# 2. IBM's stock price

Yahoo finance API - yfinance

Import finance as yf
Import pandas as pd
Ibm = yf.Ticker('ibm')
Ibm_his =ibm.history(period='max')

```
ibm = yf.Ticker('ibm')
ibm_his = ibm.history(period="max")
ibm_his
```

| Date | Open | High | Low | Close | Volume | Dividends | Stock Splits |
|---|---|---|---|---|---|---|---|
| 1962-01-02 | 1.733500 | 1.733500 | 1.714022 | 1.714022 | 407940 | 0.0 | 0.0 |
| 1962-01-03 | 1.714023 | 1.729006 | 1.714023 | 1.729006 | 305955 | 0.0 | 0.0 |
| 1962-01-04 | 1.729006 | 1.729006 | 1.711027 | 1.711776 | 274575 | 0.0 | 0.0 |
| 1962-01-05 | 1.709528 | 1.709528 | 1.675068 | 1.678065 | 384405 | 0.0 | 0.0 |
| 1962-01-08 | 1.676567 | 1.676567 | 1.633117 | 1.646601 | 572685 | 0.0 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 2021-12-16 | 123.510002 | 126.639999 | 123.480003 | 125.930000 | 7280500 | 0.0 | 0.0 |
| 2021-12-17 | 125.870003 | 128.639999 | 125.209999 | 127.400002 | 10379000 | 0.0 | 0.0 |
| 2021-12-20 | 125.720001 | 127.199997 | 124.699997 | 127.059998 | 4941400 | 0.0 | 0.0 |
| 2021-12-21 | 127.660004 | 129.339996 | 127.660004 | 128.970001 | 4856000 | 0.0 | 0.0 |
| 2021-12-22 | 129.059998 | 129.789993 | 127.599998 | 129.750000 | 3883634 | 0.0 | 0.0 |

```
df_IBM_stockprice = df_IBM_stockprice[['Date','Close']]
df_IBM_stockprice = df_IBM_stockprice.rename({'Close': 'stockprice'}, axis=1)
df_IBM_stockprice['Date'] = pd.to_datetime(df_IBM_stockprice['Date'])
df_IBM_stockprice
```

| | Date | stockprice |
|---|---|---|
| 0 | 1994-06-30 | 14.041587 |
| 1 | 1994-07-01 | 13.623327 |
| 2 | 1994-07-05 | 13.444073 |
| 3 | 1994-07-06 | 13.623327 |
| 4 | 1994-07-07 | 13.563576 |
| ... | ... | ... |
| 6886 | 2021-11-03 | 121.539200 |
| 6887 | 2021-11-04 | 120.849998 |
| 6888 | 2021-11-05 | 123.610001 |
| 6889 | 2021-11-08 | 124.540001 |
| 6890 | 2021-11-09 | 120.849998 |

6891 rows × 2 columns

# 3. NASDAQ composite index

Yahoo finance API - yfinance

```
Import finance as yf
Import pandas as pd
naq = yf.Ticker('^IXIC')
his_naq = naq.history(period="max")
```

```
[9] naq = yf.Ticker('^IXIC')
    his_naq = naq.history(period="max")
```

```
his_naq
```

| Date | Open | High | Low | Close | Vo |
|---|---|---|---|---|---|
| 1971-02-05 | 100.000000 | 100.000000 | 100.000000 | 100.000000 | |
| 1971-02-08 | 100.839996 | 100.839996 | 100.839996 | 100.839996 | |
| 1971-02-09 | 100.760002 | 100.760002 | 100.760002 | 100.760002 | |
| 1971-02-10 | 100.690002 | 100.690002 | 100.690002 | 100.690002 | |
| 1971-02-11 | 101.449997 | 101.449997 | 101.449997 | 101.449997 | |
| ... | ... | ... | ... | ... | |
| 2021-12-16 | 15629.080078 | 15633.190430 | 15119.490234 | 15180.429688 | 489047 |
| 2021-12-17 | 15036.769531 | 15288.780273 | 14960.370117 | 15169.679688 | 761632 |
| 2021-12-20 | 14933.000000 | 15007.299805 | 14860.040039 | 14980.940430 | 457675 |
| 2021-12-21 | 15140.429688 | 15349.059570 | 15015.030273 | 15341.089844 | 454623 |
| 2021-12-22 | 15319.200195 | 15525.973633 | 15303.063477 | 15521.892578 | 365708 |

```
df_Nasdaq = df_Nasdaq.dropna()
df_Nasdaq = df_Nasdaq[['Date', 'Close']]
df_Nasdaq['Date'] = pd.to_datetime(df_Nasdaq['Date'])
df_Nasdaq =df_Nasdaq.rename({'Close': 'Nasdaq_value'}, axis=1)
df_Nasdaq
```

| | Date | Nasdaq_value |
|---|---|---|
| **0** | 1971-02-05 | 100.000000 |
| **1** | 1971-02-08 | 100.839996 |
| **2** | 1971-02-09 | 100.760002 |
| **3** | 1971-02-10 | 100.690002 |
| **4** | 1971-02-11 | 101.449997 |
| **...** | ... | ... |
| **12799** | 2021-11-03 | 15811.580078 |
| **12800** | 2021-11-04 | 15940.309570 |
| **12801** | 2021-11-05 | 15971.589844 |
| **12802** | 2021-11-08 | 15982.360352 |
| **12803** | 2021-11-09 | 15886.540039 |

12804 rows × 2 columns

# Merge by R&D's date and drop null value

```
df_1 = pd.merge(df_RD, df_Nasdaq, how="outer", on=['Date'])
df_1
```

| | Date | R&D | Nasdaq_value |
|---|---|---|---|
| **0** | 1994-06-30 | 1.091e+09 | 705.960022 |
| **1** | 1994-09-30 | 1.053e+09 | 764.289978 |
| **2** | 1994-12-31 | 1118000000.00 | NaN |
| **3** | 1995-03-31 | 9.13e+08 | 817.210022 |
| **4** | 1995-06-30 | 9.74e+08 | 933.450012 |
| **...** | ... | ... | ... |
| **12831** | 2021-11-03 | NaN | 15811.580078 |
| **12832** | 2021-11-04 | NaN | 15940.309570 |
| **12833** | 2021-11-05 | NaN | 15971.589844 |
| **12834** | 2021-11-08 | NaN | 15982.360352 |
| **12835** | 2021-11-09 | NaN | 15886.540039 |

12836 rows × 3 columns

```
df_2 = pd.merge(df_1, df_IBM_stockprice, how="outer", on=['Date'])
df_2
```

| | Date | R&D | Nasdaq_value | stockprice |
|---|---|---|---|---|
| **0** | 1994-06-30 | 1.091e+09 | 705.960022 | 14.041587 |
| **1** | 1994-09-30 | 1.053e+09 | 764.289978 | 16.640774 |
| **2** | 1994-12-31 | 1118000000.00 | NaN | NaN |
| **3** | 1995-03-31 | 9.13e+08 | 817.210022 | 19.628345 |
| **4** | 1995-06-30 | 9.74e+08 | 933.450012 | 22.944551 |
| **...** | ... | ... | ... | ... |
| **12831** | 2021-11-03 | NaN | 15811.580078 | 121.539200 |
| **12832** | 2021-11-04 | NaN | 15940.309570 | 120.849998 |
| **12833** | 2021-11-05 | NaN | 15971.589844 | 123.610001 |
| **12834** | 2021-11-08 | NaN | 15982.360352 | 124.540001 |
| **12835** | 2021-11-09 | NaN | 15886.540039 | 120.849998 |

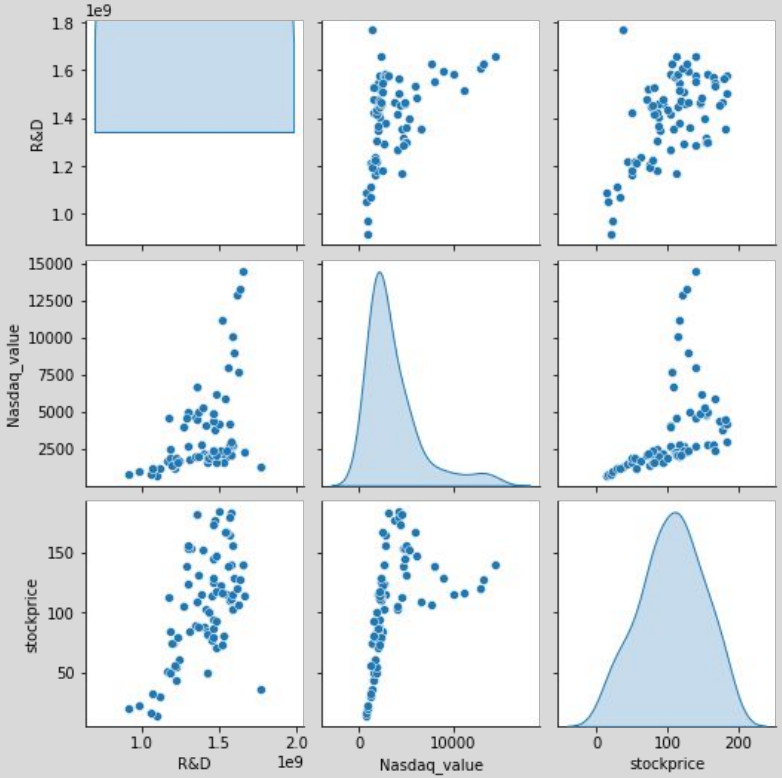
12836 rows × 4 columns

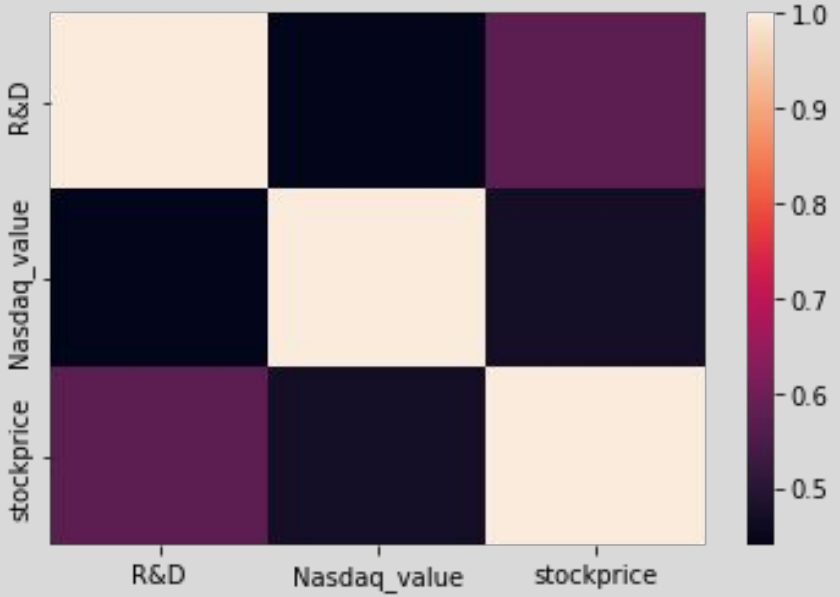
```
[ ] df_IBM = df_2.dropna()

df_IBM['R&D'] = pd.to_numeric(df_IBM['R&D'], downcast='float')

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
A value is trying to be set on a copy of a slice from a DataFram
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/
  """Entry point for launching an IPython kernel.
```

```
df_IBM
```

| | Date | R&D | Nasdaq_value | stockprice |
|---|---|---|---|---|
| **0** | 1994-06-30 | 1.091000e+09 | 705.960022 | 14.041587 |
| **1** | 1994-09-30 | 1.053000e+09 | 764.289978 | 16.640774 |
| **3** | 1995-03-31 | 9.130000e+08 | 817.210022 | 19.628345 |
| **4** | 1995-06-30 | 9.740000e+08 | 933.450012 | 22.944551 |
| **9** | 1996-09-30 | 1.115000e+09 | 1226.920044 | 29.756214 |
| **...** | ... | ... | ... | ... |
| **104** | 2020-06-30 | 1.582000e+09 | 10058.769531 | 115.458893 |
| **105** | 2020-09-30 | 1.515000e+09 | 11167.509766 | 116.319313 |
| **106** | 2020-12-31 | 1.611000e+09 | 12888.280273 | 120.344170 |
| **107** | 2021-03-31 | 1.630000e+09 | 13246.870117 | 127.399620 |
| **108** | 2021-06-30 | 1.657000e+09 | 14503.950195 | 140.143402 |

77 rows × 4 columns


1e9
R&D
Nasdaq_value
stockprice
1e9


R&D
Nasdaq_value
stockprice
R&D
Nasdaq_value
stockprice

# Import method and set independent variable and dependent variable

```
x = df_IBM.iloc[:,1:3].values
y = df_IBM.iloc[:,3].values
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

# Decision tree regression with Cross Validation

```
xtrain_DTR, xtest_DTR, ytrain_DTR, ytest_DTR=train_test_split(x, y, random_state=33, test_size=0.19)
```

```
regressor = DecisionTreeRegressor()

regressor.fit(xtrain_DTR,ytrain_DTR)
```

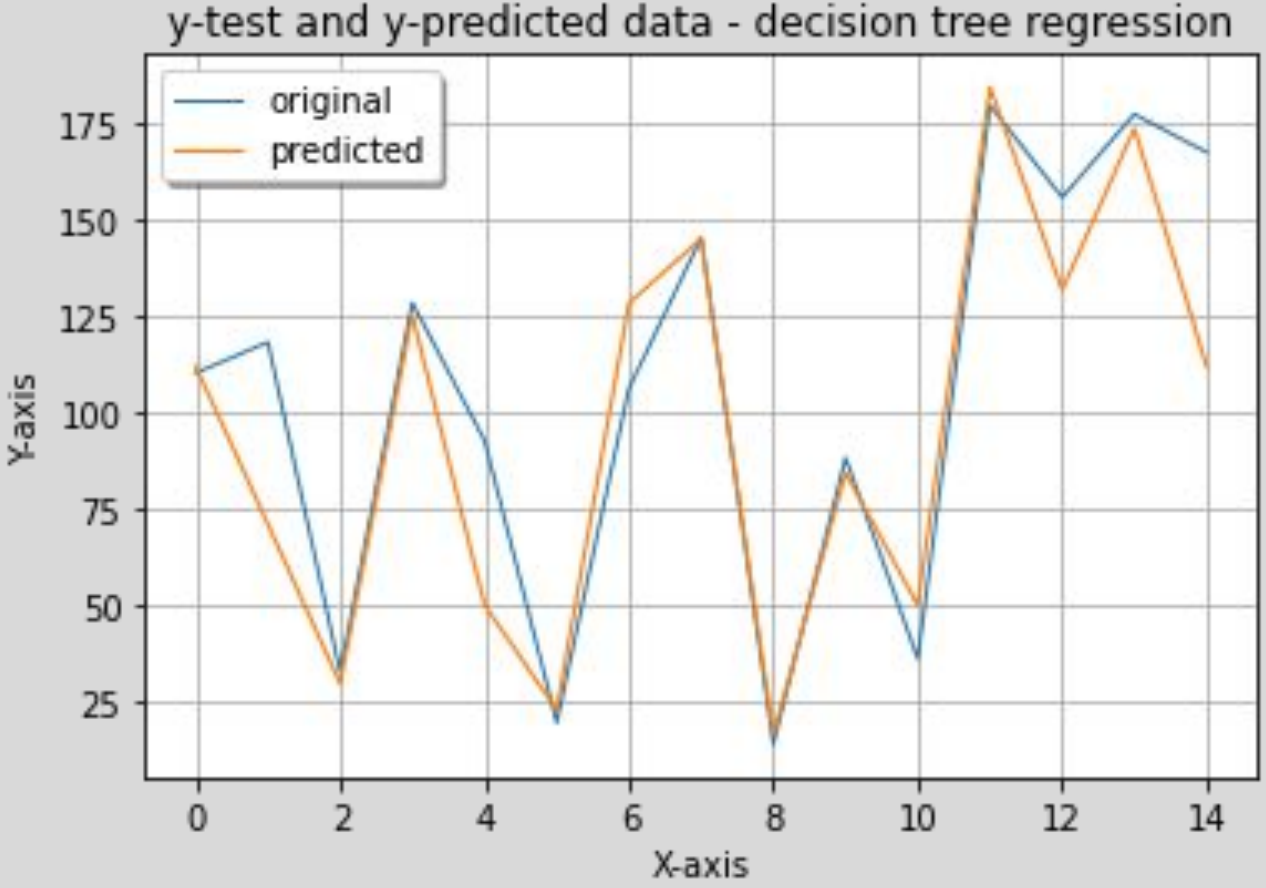
```
DecisionTreeRegressor()
```

```
score_DTR = regressor.score(xtest_DTR,ytest_DTR)
print('R-squared:', score_DTR)
```

```
R-squared: 0.8140956617200699
```

```
ypred_DTR = regressor.predict(xtest_DTR)
mse_DTR = mean_squared_error(ytest_DTR, ypred_DTR)
rmse_DTR = mean_absolute_error(ytest_DTR, ypred_DTR)
print('MSE: ', mse_DTR)
print('RMSE: ', rmse_DTR)
```

```
MSE:  562.8930318412687
RMSE:  15.423040399999998
```


y-test and y-predicted data - decision tree regression
original
predicted
175
150
125
100
75
50
25
Y-axis
0
2
4
6
8
10
12
14
X-axis

# Random forest regression with Cross Validation

```
xtrain_RFR, xtest_RFR, ytrain_RFR, ytest_RFR=train_test_split(x, y, test_size=0.15)
```

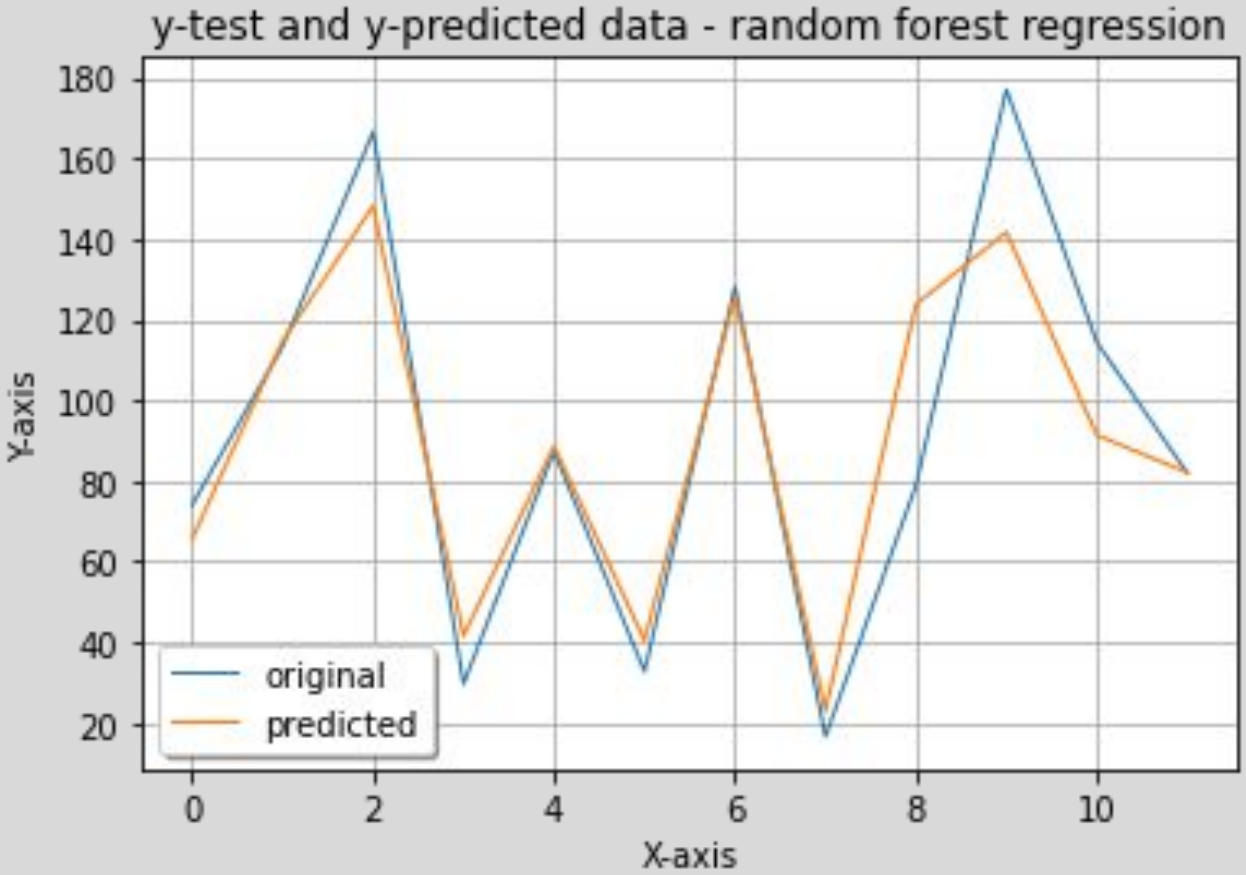
```
rfr = RandomForestRegressor()
```

```
rfr.fit(xtrain_RFR, ytrain_RFR)

score_RFR = rfr.score(xtrain_RFR, ytrain_RFR)
print("R-squared:", score_RFR)
```

```
R-squared: 0.9535096541587306
```

```
ypred_RFR = rfr.predict(xtest_RFR)

mse_RFR = mean_squared_error(ytest_RFR, ypred_RFR)
print("MSE: ", mse_RFR)
print("RMSE: ", mse_RFR*(1/2.0))
```

```
MSE:  371.75072653418755
RMSE:  185.87536326709377
```


y-test and y-predicted data - random forest regression
180
160
140
120
100
80
60
40
20
Y-axis
0
2
4
6
8
10
X-axis
original
predicted

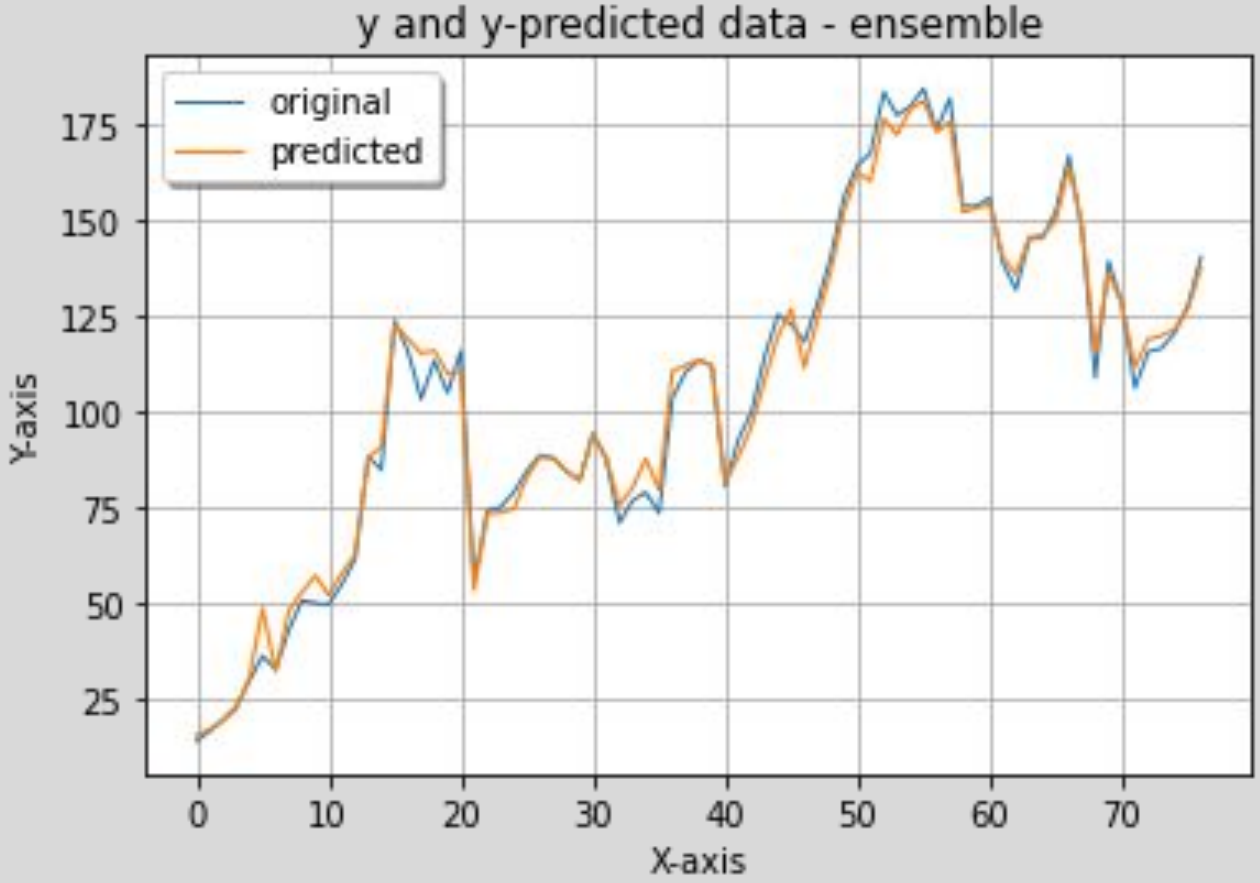
# Voting regression(ensemble method)

```
from sklearn.ensemble import VotingRegressor
```

```
er = VotingRegressor([('DTR',regressor),('RFR',rfr)])
```

```
er.fit(x,y)
score_er = er.score(x, y)
ypred_er = er.predict(x)
mse_er = mean_squared_error(y, ypred_er)
rmse_er = mean_absolute_error(y, ypred_er)
print("R-squared:", score_er)
print('MSE: ', mse_er)
print('RMSE: ', rmse_er)
```

```
R-squared: 0.9913489040193093
MSE:  16.447809293776036
RMSE:  3.045853476883111
```


y and y-predicted data - ensemble
original
predicted
175
150
125
100
75
50
25
Y-axis
0
10
20
30
40
50
60
70
X-axis

# Evaluation of Different models

| | Decision tree | Random forest | Voting regression |
|---|---|---|---|
| R-squared | 0.81 | 0.95 | 0.99 |
| MSE | 562.89 | 371.75 | 16.45 |
| RMSE | 15.42 | 185.88 | 3.05 |

# conclusion

1. Time series data is a way to load massive accounting data in regression model.

2. Decision tree regression and random forest regression can treat non-linear issues and less feature.

3. Random forest also is a suitable addition by voting model.  It often adapt in same conditions because random forest is a bagging of decision tree.

4. The model reach the intended result that accuracy is 99%. R&D could build an accurate relationship with IBM's stock price, which is a workable method to predict the company's value then guide R&D management

# Thank you

# Reference

[1] Tang, L., Pan, H., & Yao, Y. (2018). K-nearest neighbor regression with principal component analysis for financial time series prediction. *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence - ICCAI 2018*. https://doi.org/10.1145/3194452.3194467
[2] Sen, J., Mehtab, S., & Dutta, A. (2021). Stock price prediction using machine learning and LSTM-based deep learning models. https://doi.org/10.36227/techrxiv.15103602.v1
[3] Kogan, S., Levin, D., Routledge, B. R., Sagi, J. S., & Smith, N. A. (2009). Predicting risk from financial reports with regression. *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics on - NAACL '09*. https://doi.org/10.3115/1620754.1620794
[4] Wang, H. (2020). Stock price prediction based on machine learning approaches. *Proceedings of the 3rd International Conference on Data Science and Information Technology*. https://doi.org/10.1145/3414274.3414275
[5] Sen, J., Mehtab, S., & Dutta, A. (2021). Stock price prediction using machine learning and LSTM-based deep learning models. https://doi.org/10.36227/techrxiv.15103602.v1