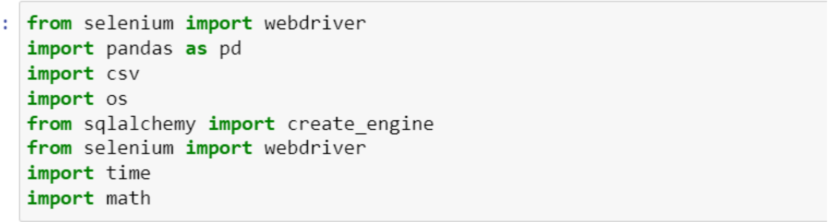
Documentation

Objective:

1. To build a EPS database which stores the analysts forecasts and real EPS information for 29 companies
2. Scraping the data to retrieve the following in the form of tables from the Estimize.com for all available quarters in 2022, 2021, and 2020.
   - Company Basic Information
   - Company Forecasts
   - Analysts Information
   - Stocks Covered
   - Pending Estimates
   - Scored Estimates

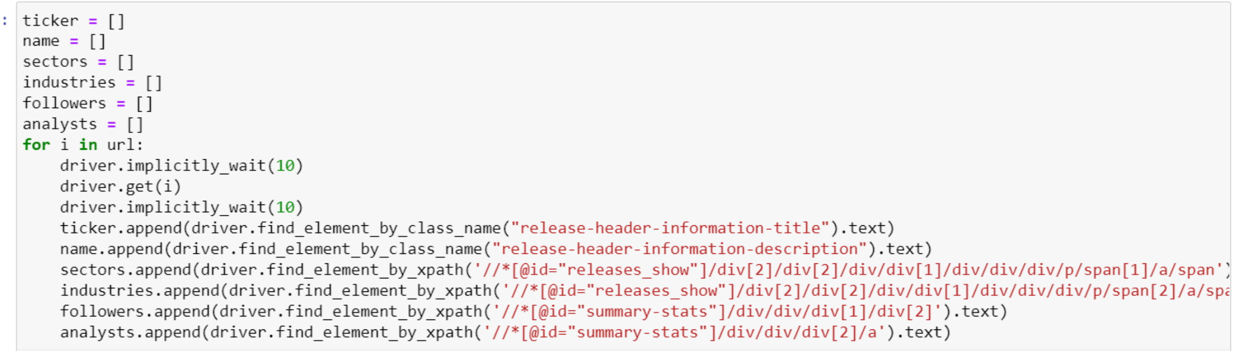
Importing all the necessary Files:



```
from selenium import webdriver
import pandas as pd
import csv
import os
from sqlalchemy import create_engine
from selenium import webdriver
import time
import math
```

- Chrome WebDriver is installed and used here in order to run the file.

Company Basic Information:

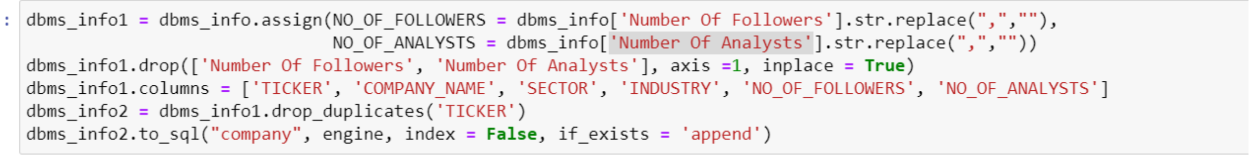


```
ticker = []
name = []
sectors = []
industries = []
followers = []
analysts = []
for i in url:
    driver.implicitly_wait(10)
    driver.get(i)
    driver.implicitly_wait(10)
    ticker.append(driver.find_element_by_class_name("release-header-information-title").text)
    name.append(driver.find_element_by_class_name("release-header-information-description").text)
    sectors.append(driver.find_element_by_xpath('//*[@id="releases_show"]/div[2]/div[2]/div/div[1]/div/div/div/p/span[1]/a/span')
    industries.append(driver.find_element_by_xpath('//*[@id="releases_show"]/div[2]/div[2]/div/div[1]/div/div/div/p/span[2]/a/spa
    followers.append(driver.find_element_by_xpath('//*[@id="summary-stats"]/div/div/div[1]/div[2]').text)
    analysts.append(driver.find_element_by_xpath('//*[@id="summary-stats"]/div/div/div[2]/a').text)
```

- The data is scraped using Selenium.
- The xpath is chosen as unique identifier for scraping the data of all columns in Company Basic information table.

We obtain a Data Frame which contains the Basic Information of 29 Companies. It includes the following columns:

- Ticker_list
- Company Name
- Sectors
- Industries
- Number Of Followers
- Number Of Analysts



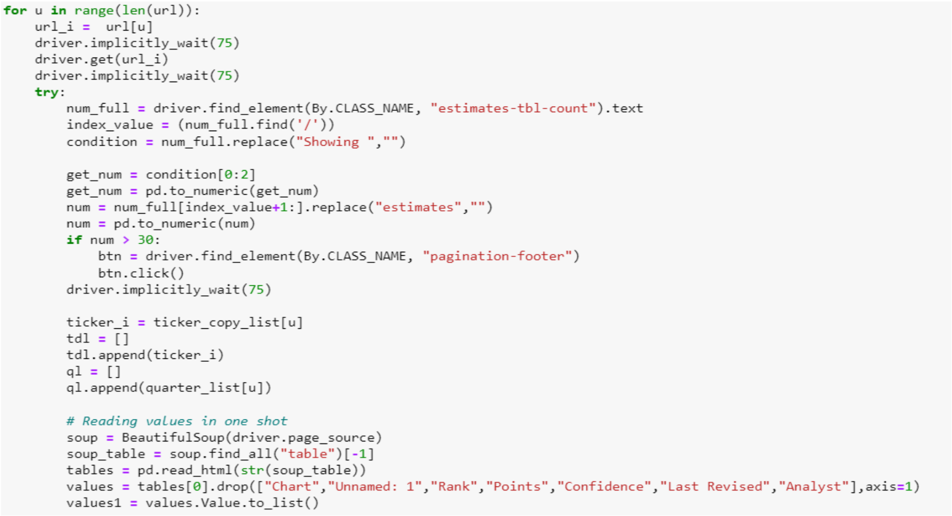
```
: dbms_info1 = dbms_info.assign(NO_OF_FOLLOWERS = dbms_info['Number Of Followers'].str.replace(",",""),
                                NO_OF_ANALYSTS = dbms_info['Number Of Analysts'].str.replace(",",""))
  dbms_info1.drop(['Number Of Followers', 'Number Of Analysts'], axis =1, inplace = True)
  dbms_info1.columns = ['TICKER', 'COMPANY_NAME', 'SECTOR', 'INDUSTRY', 'NO_OF_FOLLOWERS', 'NO_OF_ANALYSTS']
  dbms_info2 = dbms_info1.drop_duplicates('TICKER')
  dbms_info2.to_sql("company", engine, index = False, if_exists = 'append')
```

- The ',' is removed from the 'Number Of Followers' and 'Number Of Analysts' columns and then the data is cleaned in order to run the tables properly in MYSQL.

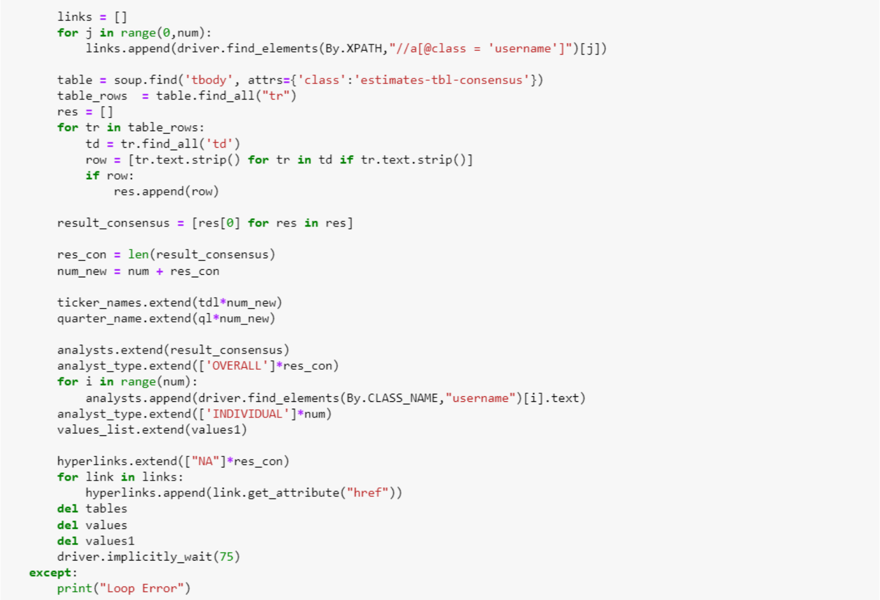
Company Forecasts:

This table contains the following information for 29 companies for all available quarters in 2022, 2021, and 2020:

- Ticker
- Quarter_Year
- Analyst
- Value
- Analyst_Type
- URL



```
for u in range(len(url)):
    url_i =  url[u]
    driver.implicitly_wait(75)
    driver.get(url_i)
    driver.implicitly_wait(75)
    try:
        num_full = driver.find_element(By.CLASS_NAME, "estimates-tbl-count").text
        index_value = (num_full.find('/'))
        condition = num_full.replace("Showing ","")

        get_num = condition[0:2]
        get_num = pd.to_numeric(get_num)
        num = num_full[index_value+1:].replace("estimates","")
        num = pd.to_numeric(num)
        if num > 30:
            btn = driver.find_element(By.CLASS_NAME, "pagination-footer")
            btn.click()
        driver.implicitly_wait(75)

        ticker_i = ticker_copy_list[u]
        tdl = []
        tdl.append(ticker_i)
        ql = []
        ql.append(quarter_list[u])

        # Reading values in one shot
        soup = BeautifulSoup(driver.page_source)
        soup_table = soup.find_all("table")[-1]
        tables = pd.read_html(str(soup_table))
        values = tables[0].drop(["Chart","Unnamed: 1","Rank","Points","Confidence","Last Revised","Analyst"],axis=1)
        values1 = values.Value.to_list()
```



```
        links = []
        for j in range(0,num):
            links.append(driver.find_elements(By.XPATH,"//a[@class = 'username']")[j])

        table = soup.find('tbody', attrs={'class':'estimates-tbl-consensus'})
        table_rows  = table.find_all("tr")
        res = []
        for tr in table_rows:
            td = tr.find_all('td')
            row = [tr.text.strip() for tr in td if tr.text.strip()]
            if row:
                res.append(row)

        result_consensus = [res[0] for res in res]

        res_con = len(result_consensus)
        num_new = num + res_con

        ticker_names.extend(tdl*num_new)
        quarter_name.extend(ql*num_new)

        analysts.extend(result_consensus)
        analyst_type.extend(['OVERALL']*res_con)
        for i in range(num):
            analysts.append(driver.find_elements(By.CLASS_NAME,"username")[i].text)
        analyst_type.extend(['INDIVIDUAL']*num)
        values_list.extend(values1)

        hyperlinks.extend(["NA"]*res_con)
        for link in links:
            hyperlinks.append(link.get_attribute("href"))
        del tables
        del values
        del values1
        driver.implicitly_wait(75)
    except:
        print("Loop Error")
```

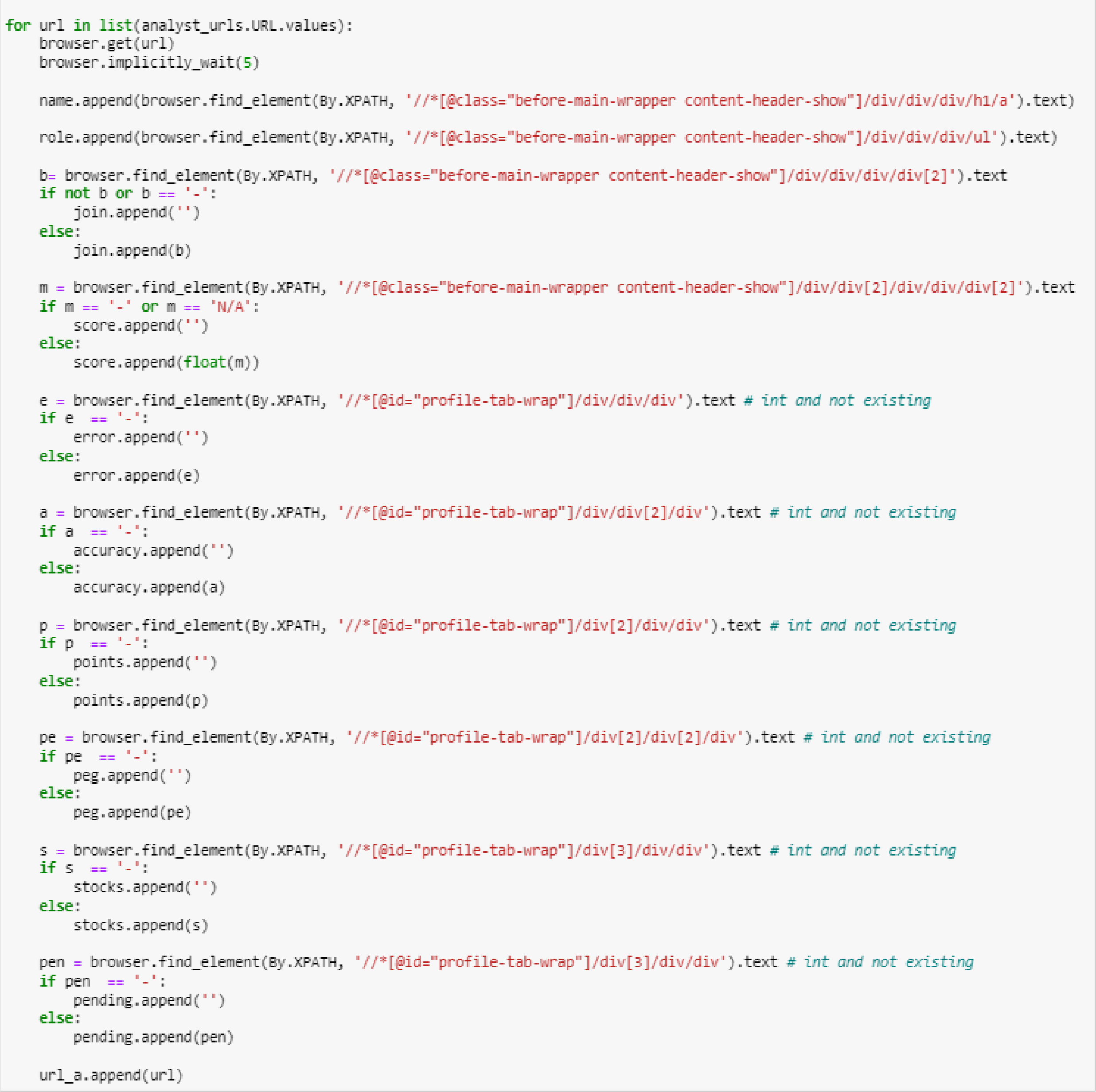
- In order to obtain all the Analysts Information - Beautiful Soup and Selenium is used.
- The 'values' column is Extracted using Beautiful Soup, whereas Selenium is used for other data extraction.
- The Analyst_Type has 'Overall' and 'Individual' content.
- Here Overall is assigned to Reported Earnings, Estimize Consensus, Estimize Mean and Wall Street Consensus, whereas Individual is assigned to all the individual Analysts.

Analyst Information:

This table contains the following columns:

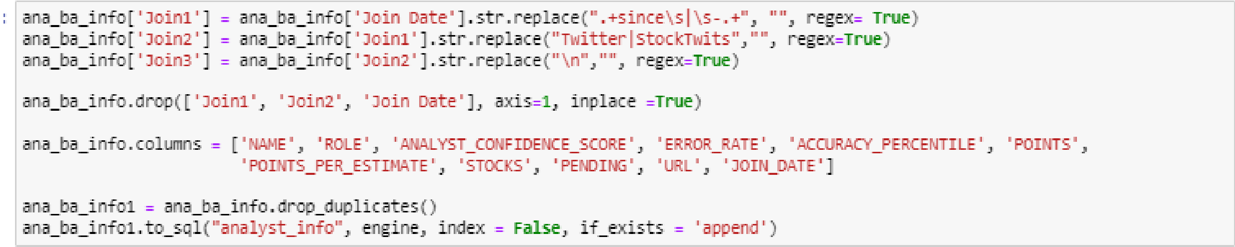
- Name
- Roles

- Join Date
- Analyst Confidence Score
- Error Rate
- Accuracy Percentile
- Points
- Points/Estimate
- Stocks
- Pending
- URL



```
for url in list(analyst_urls.URL.values):
    browser.get(url)
    browser.implicitly_wait(5)

    name.append(browser.find_element(By.XPATH, '//*[@class="before-main-wrapper content-header-show"]/div/div/div/h1/a').text)

    role.append(browser.find_element(By.XPATH, '//*[@class="before-main-wrapper content-header-show"]/div/div/div/ul').text)

    b= browser.find_element(By.XPATH, '//*[@class="before-main-wrapper content-header-show"]/div/div/div/div[2]').text
    if not b or b == '-':
        join.append('')
    else:
        join.append(b)

    m = browser.find_element(By.XPATH, '//*[@class="before-main-wrapper content-header-show"]/div/div[2]/div/div/div[2]').text
    if m == '-' or m == 'N/A':
        score.append('')
    else:
        score.append(float(m))

    e = browser.find_element(By.XPATH, '//*[@id="profile-tab-wrap"]/div/div/div').text # int and not existing
    if e  == '-':
        error.append('')
    else:
        error.append(e)

    a = browser.find_element(By.XPATH, '//*[@id="profile-tab-wrap"]/div/div[2]/div').text # int and not existing
    if a  == '-':
        accuracy.append('')
    else:
        accuracy.append(a)

    p = browser.find_element(By.XPATH, '//*[@id="profile-tab-wrap"]/div[2]/div/div').text # int and not existing
    if p  == '-':
        points.append('')
    else:
        points.append(p)

    pe = browser.find_element(By.XPATH, '//*[@id="profile-tab-wrap"]/div[2]/div[2]/div').text # int and not existing
    if pe  == '-':
        peg.append('')
    else:
        peg.append(pe)

    s = browser.find_element(By.XPATH, '//*[@id="profile-tab-wrap"]/div[3]/div/div').text # int and not existing
    if s  == '-':
        stocks.append('')
    else:
        stocks.append(s)

    pen = browser.find_element(By.XPATH, '//*[@id="profile-tab-wrap"]/div[3]/div/div').text # int and not existing
    if pen  == '-':
        pending.append('')
    else:
        pending.append(pen)

    url_a.append(url)
```

- The data is scraped using Selenium. Here the XPath is chosen to scrape the data form each analyst



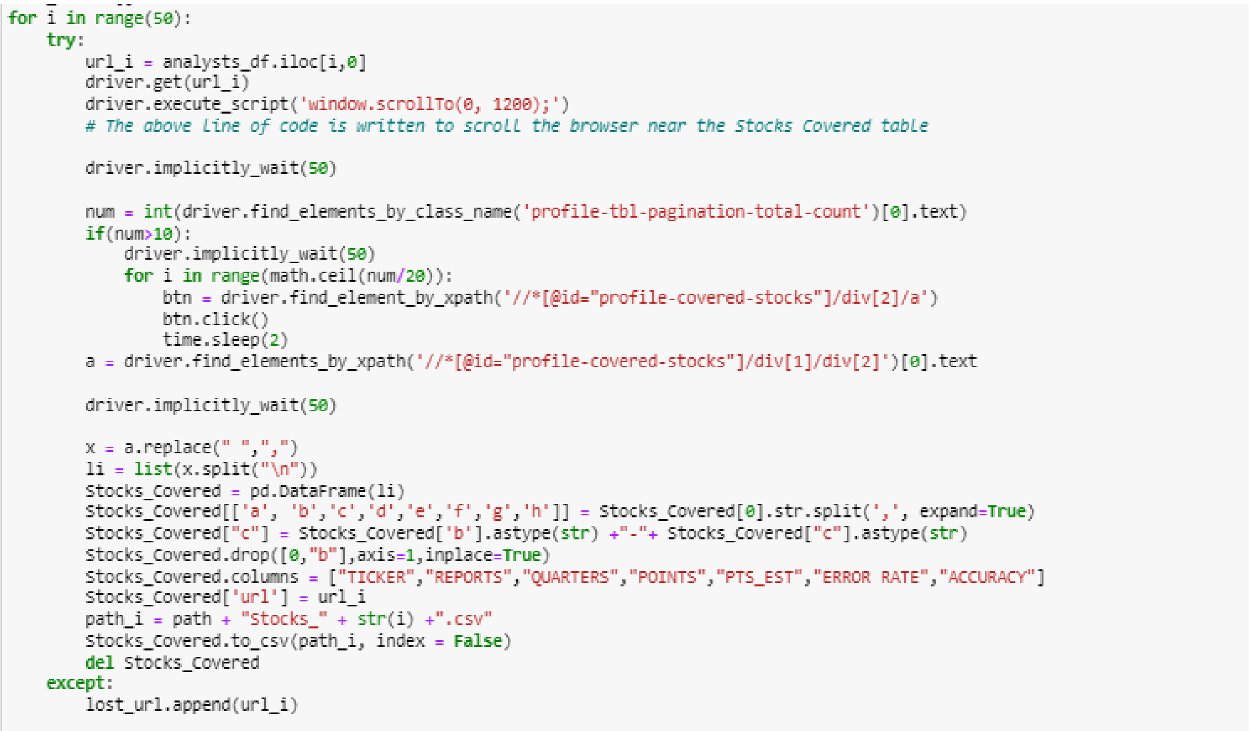
```
ana_ba_info['Join1'] = ana_ba_info['Join Date'].str.replace(".+since\s|\s-.+", "", regex= True)
ana_ba_info['Join2'] = ana_ba_info['Join1'].str.replace("Twitter|StockTwits","", regex=True)
ana_ba_info['Join3'] = ana_ba_info['Join2'].str.replace("\n","", regex=True)

ana_ba_info.drop(['Join1', 'Join2', 'Join Date'], axis=1, inplace =True)

ana_ba_info.columns = ['NAME', 'ROLE', 'ANALYST_CONFIDENCE_SCORE', 'ERROR_RATE', 'ACCURACY_PERCENTILE', 'POINTS',
                       'POINTS_PER_ESTIMATE', 'STOCKS', 'PENDING', 'URL', 'JOIN_DATE']

ana_ba_info1 = ana_ba_info.drop_duplicates()
ana_ba_info1.to_sql("analyst_info", engine, index = False, if_exists = 'append')
```

The above snippet shows data – preprocessing for Analyst information table. After Data cleaning the DataFrame is brought into MYSQL.

Stocks Covered:

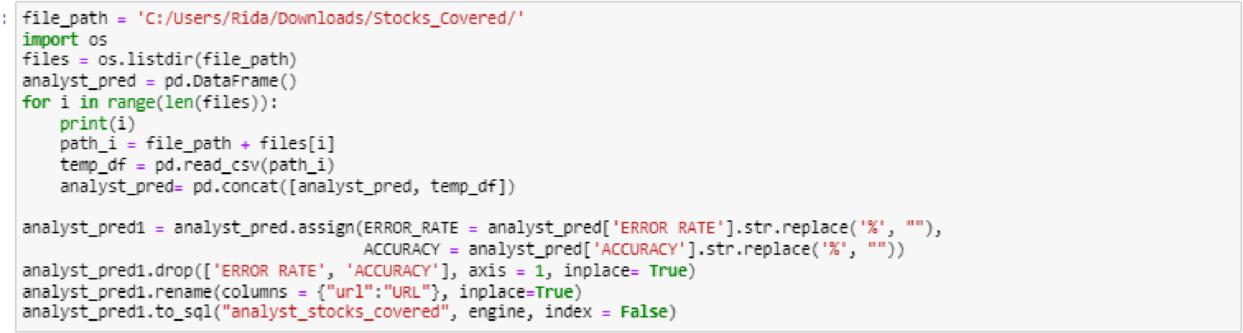
This table contains the following:

- Ticker
- Reports
- Quarters
- Points
- PTS_EST
- URL
- Error Rate



```
for i in range(50):
    try:
        url_i = analysts_df.iloc[i,0]
        driver.get(url_i)
        driver.execute_script('window.scrollTo(0, 1200);')
        # The above Line of code is written to scroll the browser near the Stocks Covered table

        driver.implicitly_wait(50)

        num = int(driver.find_elements_by_class_name('profile-tbl-pagination-total-count')[0].text)
        if(num>10):
            driver.implicitly_wait(50)
            for i in range(math.ceil(num/20)):
                btn = driver.find_element_by_xpath('//*[@id="profile-covered-stocks"]/div[2]/a')
                btn.click()
                time.sleep(2)
        a = driver.find_elements_by_xpath('//*[@id="profile-covered-stocks"]/div[1]/div[2]')[0].text

        driver.implicitly_wait(50)

        x = a.replace(" ",",")
        li = list(x.split("\n"))
        Stocks_Covered = pd.DataFrame(li)
        Stocks_Covered[['a', 'b','c','d','e','f','g','h']] = Stocks_Covered[0].str.split(',', expand=True)
        Stocks_Covered["c"] = Stocks_Covered['b'].astype(str) +"-"+ Stocks_Covered["c"].astype(str)
        Stocks_Covered.drop([0,"b"],axis=1,inplace=True)
        Stocks_Covered.columns = ["TICKER","REPORTS","QUARTERS","POINTS","PTS_EST","ERROR RATE","ACCURACY"]
        Stocks_Covered['url'] = url_i
        path_i = path + "Stocks_" + str(i) +".csv"
        Stocks_Covered.to_csv(path_i, index = False)
        del Stocks_Covered
    except:
        lost_url.append(url_i)
```

- The data is scraped using Selenium.
- The xpath and class is chosen as unique identifier for scraping the data of all columns in Company Basic information table.
- Due to the constant interruption of 403 Error, we decided to collect the lost URLS and run the code again, in order to retrieve all the information. The information is collected in lost_url list.
- Scrolling the browser till the stocks covered table was required in order to fetch the information. Otherwise it would show “loading”.
- The table has **12745** records



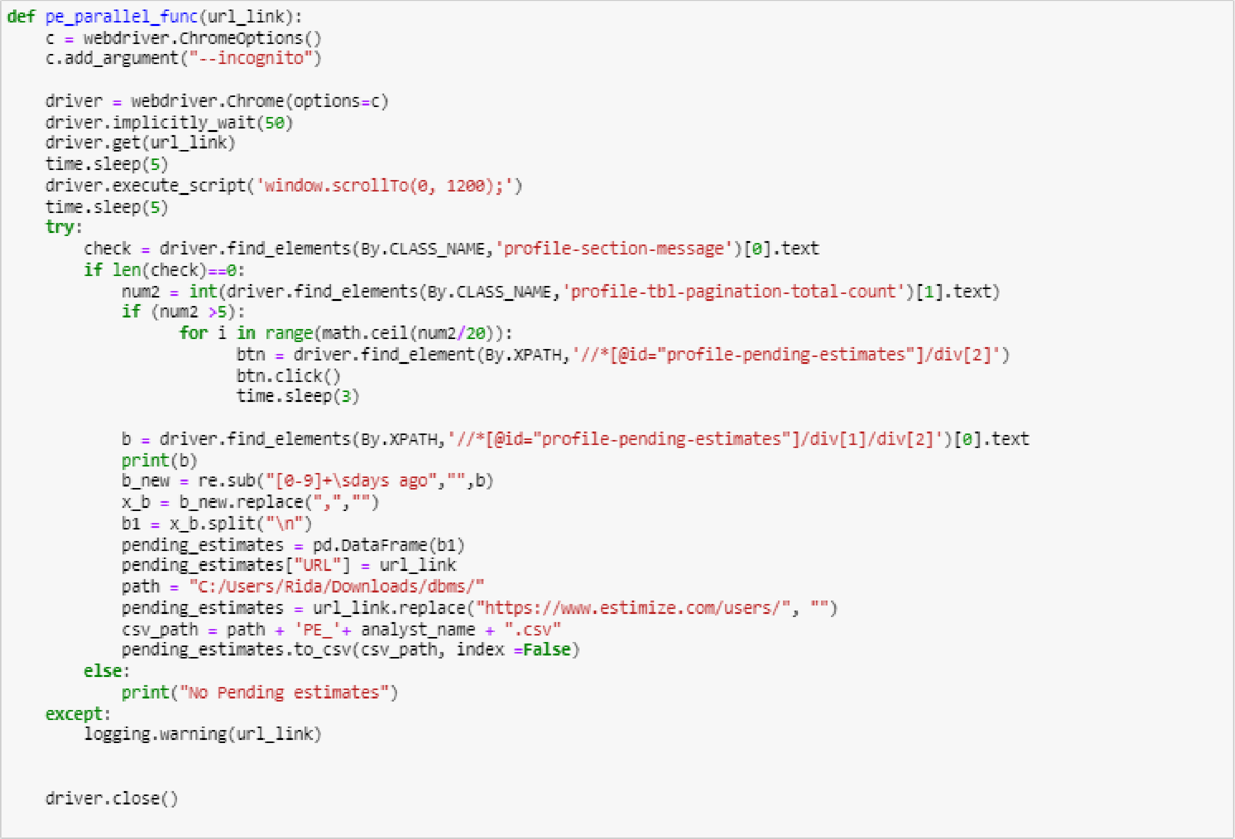
```
file_path = 'C:/Users/Rida/Downloads/Stocks_Covered/'
import os
files = os.listdir(file_path)
analyst_pred = pd.DataFrame()
for i in range(len(files)):
    print(i)
    path_i = file_path + files[i]
    temp_df = pd.read_csv(path_i)
    analyst_pred= pd.concat([analyst_pred, temp_df])

analyst_pred1 = analyst_pred.assign(ERROR_RATE = analyst_pred['ERROR RATE'].str.replace('%', ""),
                                    ACCURACY = analyst_pred['ACCURACY'].str.replace('%', ""))
analyst_pred1.drop(['ERROR RATE', 'ACCURACY'], axis = 1, inplace= True)
analyst_pred1.rename(columns = {"url":"URL"}, inplace=True)
analyst_pred1.to_sql("analyst_stocks_covered", engine, index = False)
```

The above snippet shows data – preprocessing for Stocks Covered table. After Data cleaning the Dataframe is brought into MYSQL.

Pending Estimates:

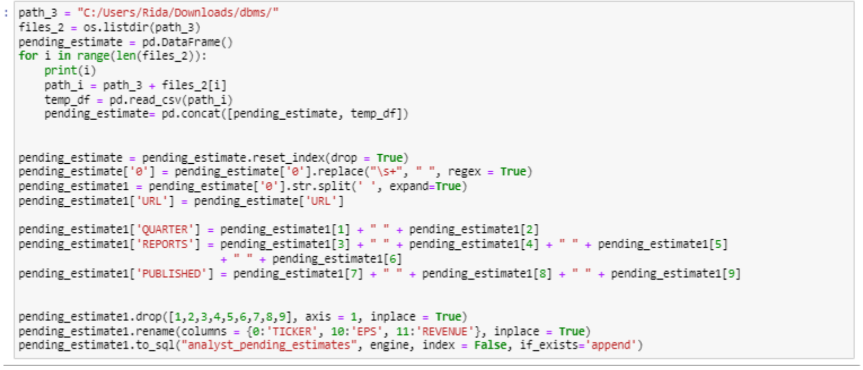
The DataFrame Pending estimates includes the following information:

- TICKER
- EPS
- Revenue
- URL
- Quarter
- Reports
- Published



```python
def pe_parallel_func(url_link):
    c = webdriver.ChromeOptions()
    c.add_argument("--incognito")

    driver = webdriver.Chrome(options=c)
    driver.implicitly_wait(50)
    driver.get(url_link)
    time.sleep(5)
    driver.execute_script('window.scrollTo(0, 1200);')
    time.sleep(5)
    try:
        check = driver.find_elements(By.CLASS_NAME,'profile-section-message')[0].text
        if len(check)==0:
            num2 = int(driver.find_elements(By.CLASS_NAME,'profile-tbl-pagination-total-count')[1].text)
            if (num2 >5):
                  for i in range(math.ceil(num2/20)):
                        btn = driver.find_element(By.XPATH,'//*[@id="profile-pending-estimates"]/div[2]')
                        btn.click()
                        time.sleep(3)

            b = driver.find_elements(By.XPATH,'//*[@id="profile-pending-estimates"]/div[1]/div[2]')[0].text
            print(b)
            b_new = re.sub("[0-9]+\sdays ago","",b)
            x_b = b_new.replace(",","")
            b1 = x_b.split("\n")
            pending_estimates = pd.DataFrame(b1)
            pending_estimates["URL"] = url_link
            path = "C:/Users/Rida/Downloads/dbms/"
            pending_estimates = url_link.replace("https://www.estimize.com/users/", "")
            csv_path = path + 'PE_'+ analyst_name + ".csv"
            pending_estimates.to_csv(csv_path, index =False)
        else:
            print("No Pending estimates")
    except:
        logging.warning(url_link)


    driver.close()
```

- Here 'check' is used to see which analyst has pending estimates.
- If the analyst has no estimates we get the message "No Pending estimates".
- Here Parallel Function is used to fetch the data quickly.
- Not every analyst has Pending estimates. Hence this table has **2478** records



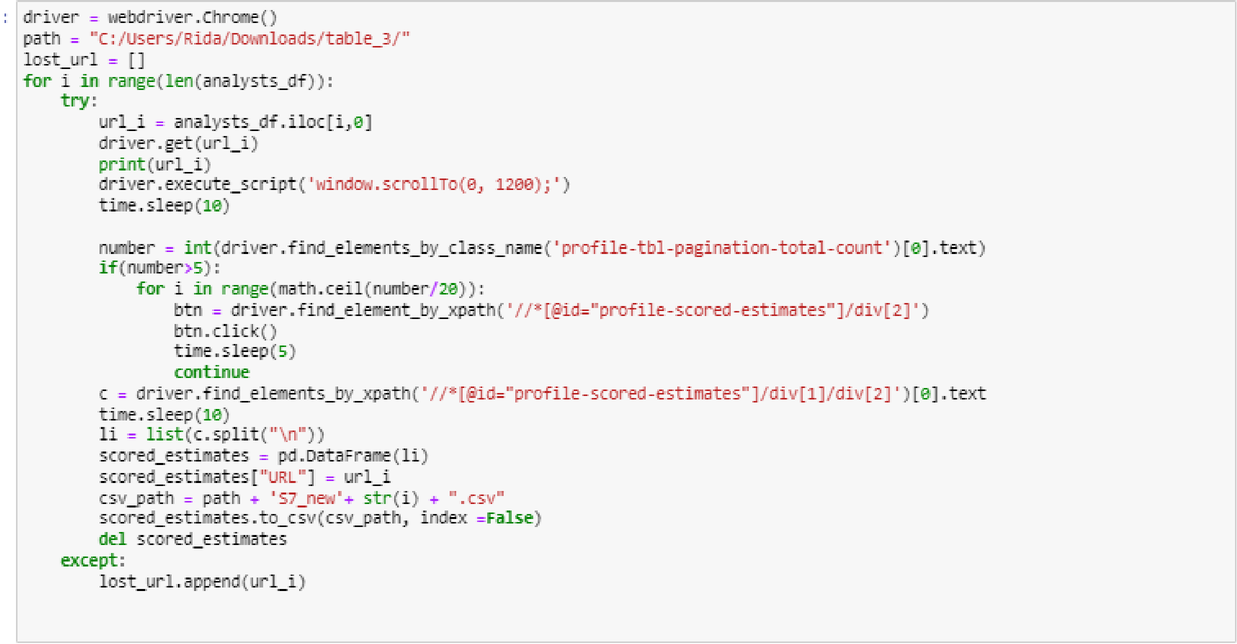
```python
path_3 = "C:/Users/Rida/Downloads/dbms/"
files_2 = os.listdir(path_3)
pending_estimate = pd.DataFrame()
for i in range(len(files_2)):
    print(i)
    path_i = path_3 + files_2[i]
    temp_df = pd.read_csv(path_i)
    pending_estimate= pd.concat([pending_estimate, temp_df])


pending_estimate = pending_estimate.reset_index(drop = True)
pending_estimate['0'] = pending_estimate['0'].replace("\s+", " ", regex = True)
pending_estimate1 = pending_estimate['0'].str.split(' ', expand=True)
pending_estimate1['URL'] = pending_estimate['URL']

pending_estimate1['QUARTER'] = pending_estimate1[1] + " " + pending_estimate1[2]
pending_estimate1['REPORTS'] = pending_estimate1[3] + " " + pending_estimate1[4] + " " + pending_estimate1[5]
                               + " " + pending_estimate1[6]
pending_estimate1['PUBLISHED'] = pending_estimate1[7] + " " + pending_estimate1[8] + " " + pending_estimate1[9]


pending_estimate1.drop([1,2,3,4,5,6,7,8,9], axis = 1, inplace = True)
pending_estimate1.rename(columns = {0:'TICKER', 10:'EPS', 11:'REVENUE'}, inplace = True)
pending_estimate1.to_sql("analyst_pending_estimates", engine, index = False, if_exists='append')
```

The above snippet shows data – preprocessing for Pending estimates table. After Data cleaning the Dataframe is brought into MYSQL.

Scored Estimates:

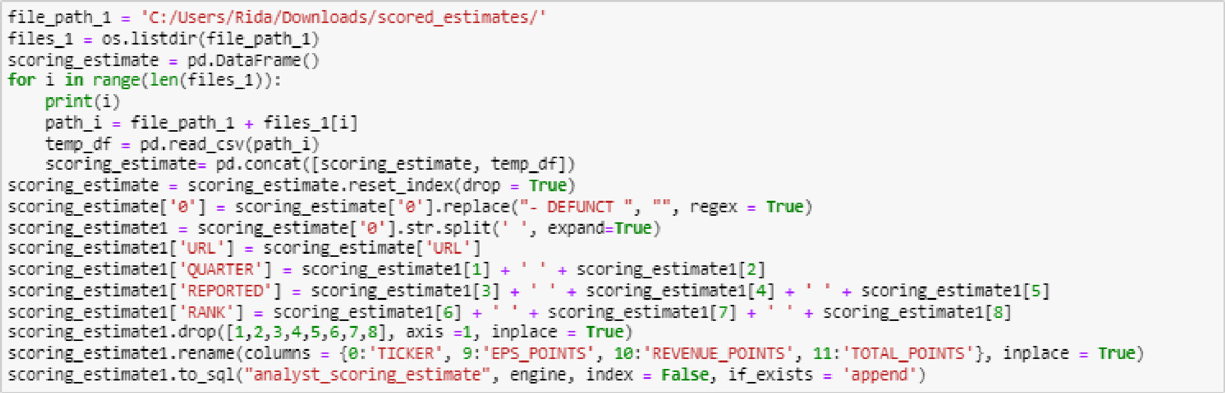
This Scored Estimates table contains the following data:

- Ticker
- EPS_Points
- Revenue_points
- Total_Points
- URL
- Quarter
- Reported
- Rank



```
driver = webdriver.Chrome()
path = "C:/Users/Rida/Downloads/table_3/"
lost_url = []
for i in range(len(analysts_df)):
    try:
        url_i = analysts_df.iloc[i,0]
        driver.get(url_i)
        print(url_i)
        driver.execute_script('window.scrollTo(0, 1200);')
        time.sleep(10)

        number = int(driver.find_elements_by_class_name('profile-tbl-pagination-total-count')[0].text)
        if(number>5):
            for i in range(math.ceil(number/20)):
                btn = driver.find_element_by_xpath('//*[@id="profile-scored-estimates"]/div[2]')
                btn.click()
                time.sleep(5)
                continue
        c = driver.find_elements_by_xpath('//*[@id="profile-scored-estimates"]/div[1]/div[2]')[0].text
        time.sleep(10)
        li = list(c.split("\n"))
        scored_estimates = pd.DataFrame(li)
        scored_estimates["URL"] = url_i
        csv_path = path + 'S7_new'+ str(i) + ".csv"
        scored_estimates.to_csv(csv_path, index =False)
        del scored_estimates
    except:
        lost_url.append(url_i)
```

- This table has **113805** Records.
- The data is scraped using Selenium.
- The xpath is chosen as unique identifier for scraping the data of all columns in Company Basic information table.



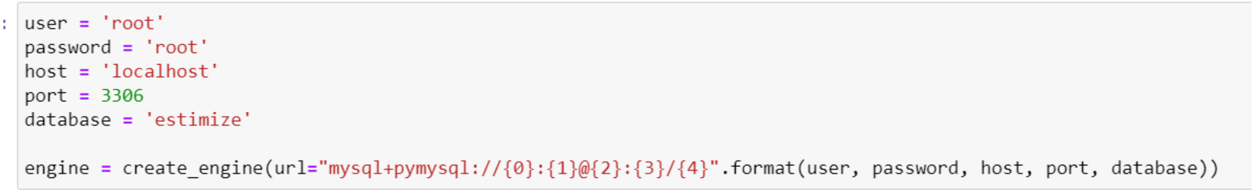
```python
file_path_1 = 'C:/Users/Rida/Downloads/scored_estimates/'
files_1 = os.listdir(file_path_1)
scoring_estimate = pd.DataFrame()
for i in range(len(files_1)):
    print(i)
    path_i = file_path_1 + files_1[i]
    temp_df = pd.read_csv(path_i)
    scoring_estimate= pd.concat([scoring_estimate, temp_df])
scoring_estimate = scoring_estimate.reset_index(drop = True)
scoring_estimate['0'] = scoring_estimate['0'].replace("- DEFUNCT ", "", regex = True)
scoring_estimate1 = scoring_estimate['0'].str.split(' ', expand=True)
scoring_estimate1['URL'] = scoring_estimate['URL']
scoring_estimate1['QUARTER'] = scoring_estimate1[1] + ' ' + scoring_estimate1[2]
scoring_estimate1['REPORTED'] = scoring_estimate1[3] + ' ' + scoring_estimate1[4] + ' ' + scoring_estimate1[5]
scoring_estimate1['RANK'] = scoring_estimate1[6] + ' ' + scoring_estimate1[7] + ' ' + scoring_estimate1[8]
scoring_estimate1.drop([1,2,3,4,5,6,7,8], axis =1, inplace = True)
scoring_estimate1.rename(columns = {0:'TICKER', 9:'EPS_POINTS', 10:'REVENUE_POINTS', 11:'TOTAL_POINTS'}, inplace = True)
scoring_estimate1.to_sql("analyst_scoring_estimate", engine, index = False, if_exists = 'append')
```

- Data Processing for Scored Estimate Table includes erasing extra test from the main columns and then bifurcating them into columns to obtain a proper table.
- We have defined the primary keys inside SQL and foreign keys as well.

Data Description:

Table	Shape (rows * columns)
Company	29 * 6
Company forecasts	37431 * 6
Analyst Basic info	8959 * 11
Analyst Stocks covered	12745 * 7
Analyst Pending Estimates	2478 * 7
Analyst Scoring Estimates	113805 * 8

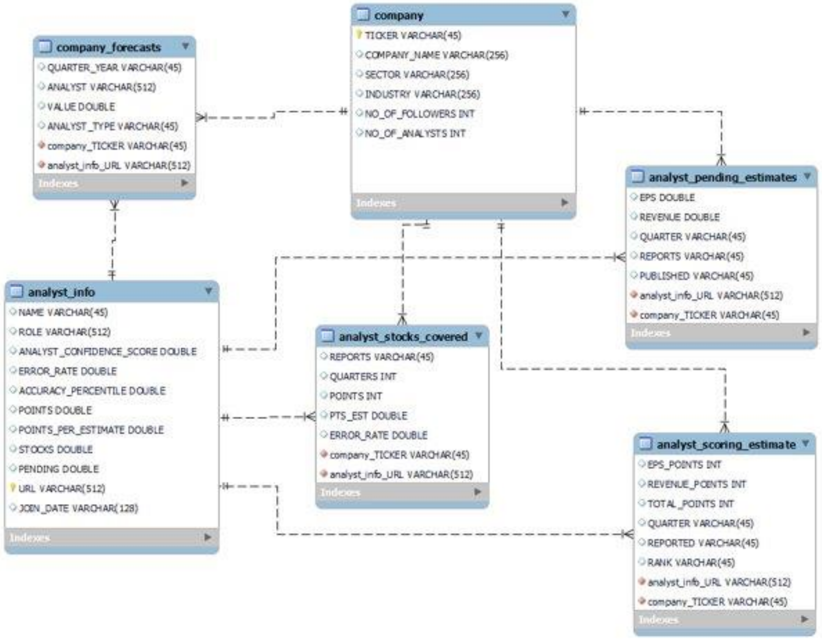
Connecting with MYSQL Database:



```python
user = 'root'
password = 'root'
host = 'localhost'
port = 3306
database = 'estimize'

engine = create_engine(url="mysql+pymysql://{0}:{1}@{2}:{3}/{4}".format(user, password, host, port, database))
```

- We have used 'sqlalchemy' package to create an engine and connect to the database.

Entity Relationship Model



Data Analysis (Mysql Part):

Query 1:

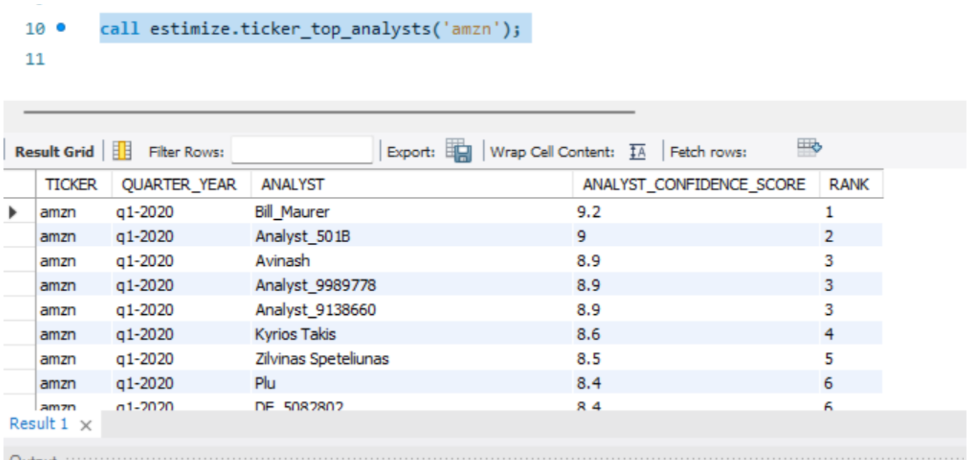
Given a ticker, how many analysts have made estimations for its EPS? Rank them by their confidence score, total points, error rate or accuracy percentile?

---QUERY---

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ticker_top_analysts`(IN ticker_name VARCHAR(255))

BEGIN

SELECT

   cf.TICKER,cf.QUARTER_YEAR,cf.ANALYST,ai.ANALYST_CONFIDENCE_SCORE, dense_rank()

OVER ( partition by TICKER,QUARTER_YEAR order by ANALYST_CONFIDENCE_SCORE desc )

AS 'RANK' FROM

   estimize.company_forecasts cf INNER JOIN

   analyst_info ai ON cf.URL = ai.URL

WHERE ai.ANALYST_CONFIDENCE_SCORE IS NOT NULL

      AND cf.TICKER = ticker_name;

END
```

call estimize.ticker_top_analysts('amzn');

--OUTPUT---



```
call estimize.ticker_top_analysts('amzn');
```

TICKER	QUARTER_YEAR	ANALYST	ANALYST_CONFIDENCE_SCORE	RANK
amzn	q1-2020	Bill_Maurer	9.2	1
amzn	q1-2020	Analyst_501B	9	2
amzn	q1-2020	Avinash	8.9	3
amzn	q1-2020	Analyst_9989778	8.9	3
amzn	q1-2020	Analyst_9138660	8.9	3
amzn	q1-2020	Kyrios Takis	8.6	4
amzn	q1-2020	Zilvinas Speteliunas	8.5	5
amzn	q1-2020	Plu	8.4	6
amzn	q1-2020	DE_5082802	8.4	6

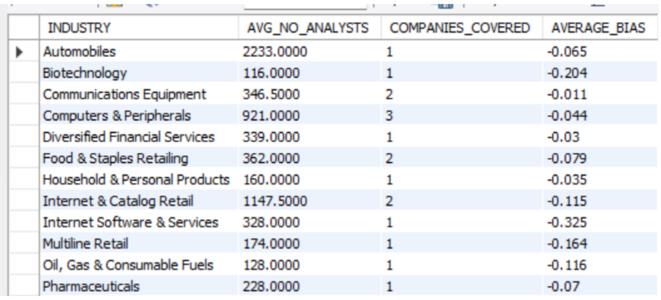
Query 2:

Given a industry, how many companies are covered, the average number of analysts, the average bias between the Estimize Consensus and the Reported Earnings?

--QUERY---

create temporary table industry_avg_counts
(select ind.INDUSTRY, avg(ind.ANALYSTS_PER_COMPANY) AVG_NO_ANALYSTS, count(distinct ind.TICKER) as COMPANIES_COVERED from
(SELECT cf.TICKER, count(distinct cf.URL) as ANALYSTS_PER_COMPANY, c.INDUSTRY FROM estimize.company_forecasts cf inner join company c
on cf.TICKER=c.TICKER where cf.ANALYST_TYPE = 'INDIVIDUAL' group by cf.TICKER) ind group by ind.INDUSTRY);
create temporary table est_consensus(SELECT cf.TICKER,cf.QUARTER_YEAR,cf.VALUE as EST_CONS, c.INDUSTRY FROM estimize.company_forecasts cf left join company c on cf.TICKER=c.TICKER where cf.ANALYST_TYPE = 'OVERALL' and cf.ANALYST like '%Estimize Consensus%');
create temporary table est_reported
(SELECT cf.TICKER,cf.QUARTER_YEAR, cf.VALUE as EST_REP, c.INDUSTRY FROM estimize.company_forecasts cf left join company c
on cf.TICKER=c.TICKER where cf.ANALYST_TYPE = 'OVERALL' and cf.ANALYST like '%Reported Earnings%');
create temporary table industry_avg_bias
(select bt.INDUSTRY, round(avg(bt.BIAS),3) as 'AVERAGE_BIAS' from
(select ec.*, er.EST_REP, round(ec.EST_CONS - er.EST_REP,3) as 'BIAS' from est_consensus ec inner join est_reported er on ec.TICKER=er.TICKER and ec.QUARTER_YEAR = er.QUARTER_YEAR) bt group by bt.INDUSTRY);
select iac.INDUSTRY,iac.AVG_NO_ANALYSTS, iac.COMPANIES_COVERED,iab.AVERAGE_BIAS from industry_avg_counts iac left join industry_avg_bias iab on iac.INDUSTRY = iab.INDUSTRY;

--OUTPUT---



INDUSTRY	AVG_NO_ANALYSTS	COMPANIES_COVERED	AVERAGE_BIAS
Automobiles	2233.0000	1	-0.065
Biotechnology	116.0000	1	-0.204
Communications Equipment	346.5000	2	-0.011
Computers & Peripherals	921.0000	3	-0.044
Diversified Financial Services	339.0000	1	-0.03
Food & Staples Retailing	362.0000	2	-0.079
Household & Personal Products	160.0000	1	-0.035
Internet & Catalog Retail	1147.5000	2	-0.115
Internet Software & Services	328.0000	1	-0.325
Multiline Retail	174.0000	1	-0.164
Oil, Gas & Consumable Fuels	128.0000	1	-0.116
Pharmaceuticals	228.0000	1	-0.07

Query 3:

Which company have the largest number of analysts with confidence score greater

than 7?

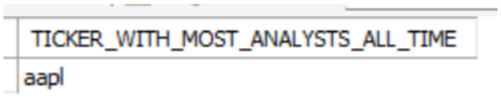
--QUERY---

create temporary table ticker_analyst_count (select cfa.TICKER, count(distinct cfa.URL) as NO_OF_ANALYSTS from (SELECT cf.TICKER,cf.QUARTER_YEAR,cf.ANALYST, cf.URL, ai.ANALYST_CONFIDENCE_SCORE FROM estimize.company_forecasts cf INNER JOIN analyst_info ai ON cf.URL = ai.URL WHERE ai.ANALYST_CONFIDENCE_SCORE > 7) cfa group by cfa.TICKER order by NO_OF_ANALYSTS desc);

set @max_analysts = (select max(NO_OF_ANALYSTS) from ticker_analyst_count);

select TICKER as TICKER_WITH_MOST_ANALYSTS_ALL_TIME from ticker_analyst_count where NO_OF_ANALYSTS = @max_analysts;

--OUTPUT---



TICKER_WITH_MOST_ANALYSTS_ALL_TIME
aapl

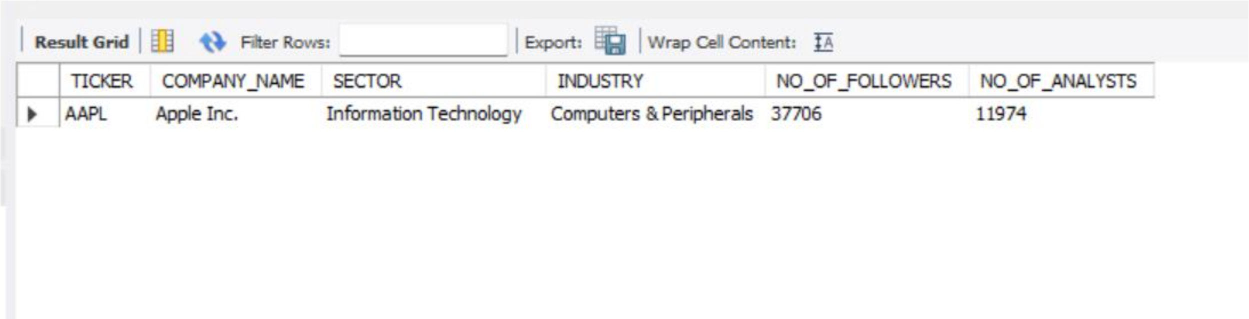
Query 4:

Who has the largest number of followers?

--QUERY---

select * from company where NO_OF_FOLLOWERS = (select max(NO_OF_FOLLOWERS) from company);

--OUTPUT---



TICKER	COMPANY_NAME	SECTOR	INDUSTRY	NO_OF_FOLLOWERS	NO_OF_ANALYSTS
AAPL	Apple Inc.	Information Technology	Computers & Peripherals	37706	11974

Regression Model (Bonus Question):

Objective: To identify the error rate of an analyst with respect to different scores of an analyst.

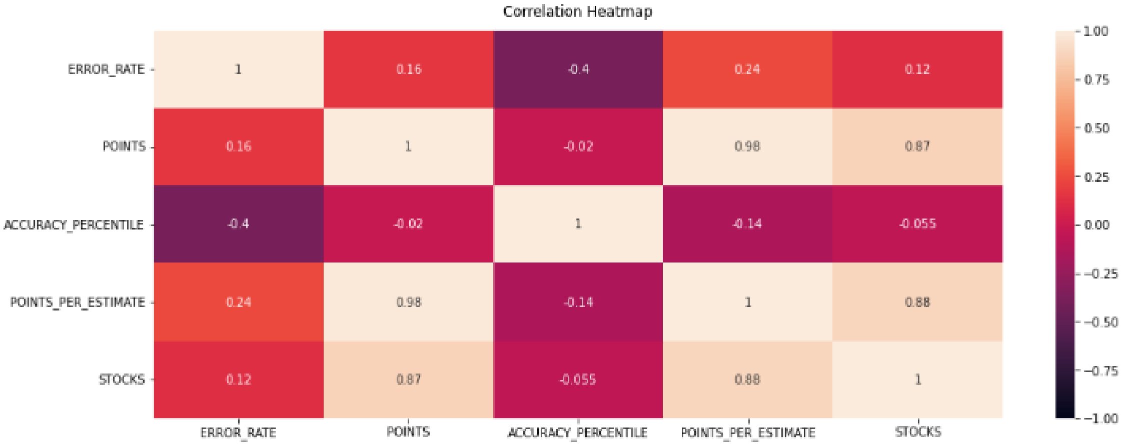
Independent Variables:

1. Points
2. Accuracy Percentile
3. Points per estimate
4. Stocks.

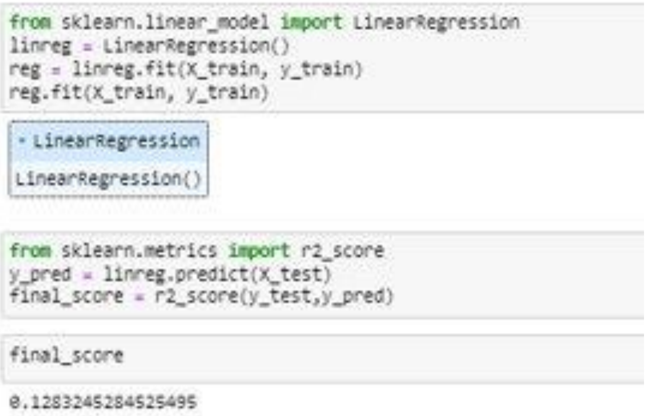
Target Variable:

1. Error rate

Correlation Heatmap:



LINEAR REGRESSION



```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
reg = linreg.fit(X_train, y_train)
reg.fit(X_train, y_train)
```

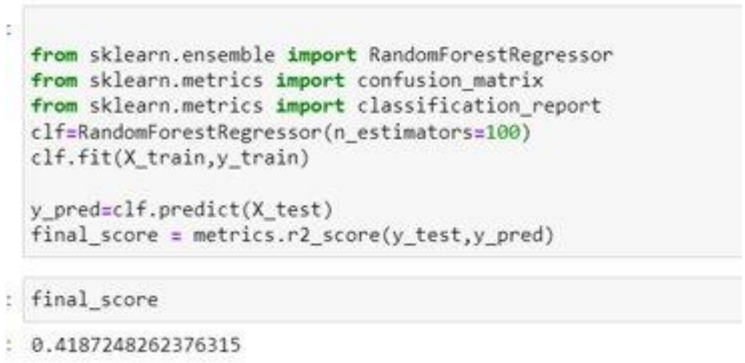
```
▸ LinearRegression
LinearRegression()
```

```
from sklearn.metrics import r2_score
y_pred = linreg.predict(X_test)
final_score = r2_score(y_test,y_pred)
```

```
final_score
```

```
0.1283245284525495
```

Random Forest

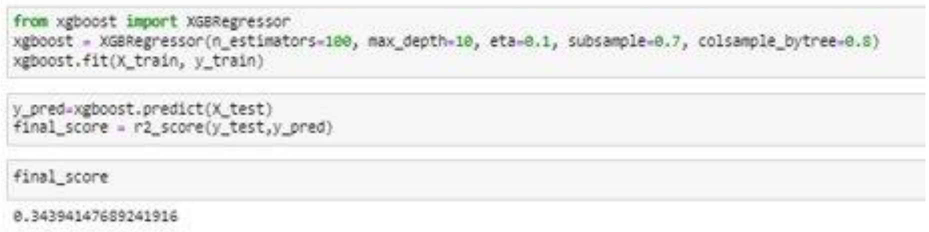


```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
clf=RandomForestRegressor(n_estimators=100)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
final_score = metrics.r2_score(y_test,y_pred)
```

```
final_score
```

```
0.4187248262376315
```

XGBRegressor

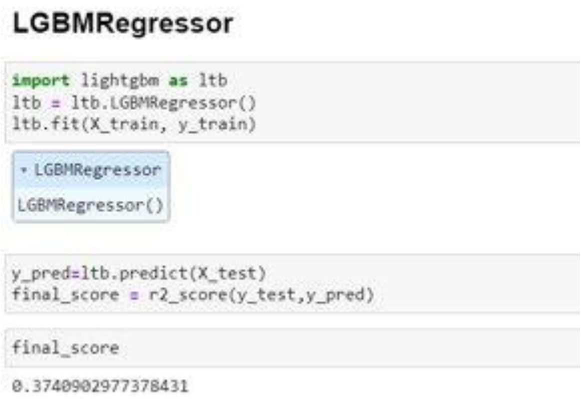


```
from xgboost import XGBRegressor
xgboost = XGBRegressor(n_estimators=100, max_depth=10, eta=0.1, subsample=0.7, colsample_bytree=0.8)
xgboost.fit(X_train, y_train)
```

```
y_pred=xgboost.predict(X_test)
final_score = r2_score(y_test,y_pred)
```

```
final_score
```

```
0.34394147689241916
```



LGBMRegressor

```
import lightgbm as ltb
ltb = ltb.LGBMRegressor()
ltb.fit(X_train, y_train)
```

```
▾ LGBMRegressor

LGBMRegressor()
```

```
y_pred=ltb.predict(X_test)
final_score = r2_score(y_test,y_pred)
```

```
final_score
```

```
0.3740902977378431
```

MODEL	R2 score	Mean Squared error
Linear Regression	0.128	0.021
Random Forest	0.418	0.014
Light Gradient Boosting	0.37	0.0155
XGBoost	0.35	0.016