

# **Student F1 VISA Prediction Based on Student's Profile**



## **NAME OF THE PARTICIPANTS**

Suhas Siddarajgari Tellatakula	11626111
Sai Tejesh Gonemadatala	11608671
Sai Praneeth Reddy Avula	11582402
Sai Rohith Varma Kantem	11606743

## **UNDER THE GUIDANCE**

**Dr. Zeenat Tariq**

PROF. OF THE DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
University Of North Texas, Denton

# INDEX

TOPICS	PAGE NO
INTRODUCTION	3
MOTIVATION	3
SIGNIFICANCE	3
OBJECTIVE	3
RELATED WORK	4
DATASET	5
ARCHITECTURE DIAGRAM	6
DETAILED FEATURES	7
WORKFLOW	8
IMPLEMENTATION	9
DATA PRE-PROCESSING	27
ANALYSIS AND PRELIMINARY RESULTS	37
MACHINE LEARNING	39
RESULTS	45
FINAL COMPARISON & PREDICTION CHECK	65
PROJECT MANAGEMENT	67
REFERENCES / BIBLIOGRAPHY	69

## **Introduction / Abstract**

Every year thousands of students appear for visa interviews to study in universities abroad. And only a handful of them clears the visa interview. The student would never know the reason for rejection. Often the visa officer evaluates the student's profile and issues a visa. If there are any drawbacks in the student profile then there is a high chance that the student might face rejection. The application we developed will evaluate students' profiles based on the criteria like student scores, work experience, Family background, etc., and outputs the chance of getting a visa approved in percentage. The application also suggests where the student is lacking. Thus, helping in building the student profile to increase the chance of getting approved.

## **Motivation :**

There is a tremendous increase in the number of students planning to pursue higher studies in a foreign country. But among thousands of applications, only a few students clear the visa process and get their visas approved by their desired country's embassy. This usually happens because of a lack of knowledge of the criteria that the embassy used to evaluate the applicant's profile. To tackle this problem we are attempting to build an effective solution that accepts student profiles as inputs like their family background, academic test score, and other appropriate information and predicts the visa chances of the applicant.

## **Significance of the tool:**

There is no tool or solution available online where a person can check his / her F1 visa approval chances. Therefore , this tool will be highly helpful for such candidates as he / she can easily check the chances of visa approval. Based on the results given by this tool , candidate can prepare or update his/her profile so that one can increase the chances. In this way , the tool can help save candidates time and money by booking the slot after improving the profile.

## **Objectives:**

This tool is aimed to do the following things:

- To output a percentage value indicating the candidates' Visa chances.
- Tool must be able to eliminate null entered rows from the data set and exclude them from getting trained for the machine learning model.
- To run different machine algorithms and choose the best model.
- To analyze the Visa\_Candidates data and use Hadoop components to produce statistics for the data.

## **Features of the tool:**

- The main feature of this solution is to give the visa prediction chances for a candidate based on the machine learning model built on his/her features.
- One of the features is to analyze the given Visa\_candidates csv data using the different components like Hadoop HDFS , Hive , Hue , Solr , Lucene, Cassandra.
- To build 5 regression models on the split data and analyze the accuracy of predictions of these

models.

- Another feature is that the tool should remove any null entered rows and duplicate rows.

## **Related Work (Background):**

Many students approach consultancies for profile evaluation and spend a lot of money to identify drawbacks in their profiles. So, to address this problem we started researching about different possible ways to find a solution. Using Artificial Intelligence models and relevant datasets we can predict the chance of getting approved. But there isn't any suitable dataset that is readily available online. So we started looking for the dataset that is closest to our desired dataset and we found a dataset on Kaggle that matches our requirement. The name of the data set is "Graduate Admission Prediction". We took this dataset deleted irrelevant columns and added more columns that are required.

For the Machine Learning and Data science part , there were very few insights from the internet. Since there are not any projects directly related to our problem statement but few similar ones like "predicting the H1B Visa Status" was found . But this project is again a classification project while what we are trying to predict is regression problem . Therefore , we used this as a reference while doing this project.

## **References:**

1. <https://www.kaggle.com/code/aryantiwari123/graduate-admission-prediction>
2. <https://www.kaggle.com/datasets/saddamazyazy/go-to-college-dataset>
3. <https://www.kaggle.com/code/akhilkasare/h-1b-visa-prediction-using-machine-learning>
4. <https://www.kaggle.com/code/campusx/gre-admission-prediction/data>

## **Dataset:**

The dataset that we are working on has 17 columns of which 16 columns are regular columns that are used for prediction and one main column called Chance of visa approval is used to display the result. The dataset includes important columns like Gre and tofel scores, university ratings, CGPA, work experience, US relatives, no of times rejected earlier, and no of backlogs.

Our dataset has 400 records of which 300 records are used for training purposes and the rest 100 are used for testing purposes.

9	8 K. Sainesh I	308	101	2	3	4	7.9	0	2 YES	0	2638958	0 NO	1	0.68
10	9 K. Uma Ma	302	102	1	2	1.5	8	0	5 NO	1	2230885	0 YES	2	0.5
11	10 Kodali Kira	323	108	3	3.5	3	8.6	0	4 NO	0	1542780	1 NO	5	0.45
12	11 Kolli Siri	325	106	3	3.5	4	8.4	1	5 YES	1	1233102	0 NO	0	0.52
13	12 Korada Ra	327	111	4	4	4.5	9	1	0 YES	0	734263	0 NO	2	0.84
14	13 Korukondé	328	112	4	4	4.5	9.1	1	2 YES	0	2515795	0 NO	2	0.78
15	14 Kotnani He	307	109	3	4	3	8	1	3 YES	0	550990	0 YES	2	0.62
16	15 Mandalapu	311	104	3	3.5	2	8.2	1	3 YES	0	709689	0 NO	0	0.61
17	16 Mummare	314	105	3	3.5	2.5	8.3	0	5 NO	1	1902907	0 NO	3	0.54
18	17 Nalabothu	317	107	3	4	3	8.7	0	3 YES	0	2560049	0 NO	2	0.66
19	18 Perella Hei	319	106	3	4	3	8	1	2 YES	0	807407	0 YES	5	0.65
20	19 Prathik Pra	318	110	3	4	3	8.8	0	2 YES	0	618192	0 NO	3	0.63
21	20 Ravela Lok	303	102	3	3.5	3	8.5	0	0 NO	0	2598373	0 NO	2	0.62
22	21 Sattiraju Sl	312	107	3	3	2	7.9	1	0 YES	0	1830373	0 NO	3	0.64
23	22 Satya Srav	325	114	4	3	2	8.4	0	0 YES	0	1983331	0 NO	4	0.7
24	23 Sollem Var	328	116	5	5	5	9.5	1	0 YES	0	1734089	0 NO	3	0.94
25	24 Sravya Tar	334	119	5	5	4.5	9.7	1	0 YES	0	2722223	0 NO	1	0.95
26	25 Sree Harsh	336	119	5	4	3.5	9.8	1	0 YES	0	1260806	0 NO	5	0.97
27	26 Sree Keert	340	120	5	4.5	4.5	9.6	1	0 YES	0	2315711	0 NO	4	0.94
28	27 Suvari Pra	322	109	5	4.5	3.5	8.8	0	1 YES	0	1607884	0 NO	1	0.76
29	28 Toship Sor	298	98	2	1.5	2.5	7.5	1	7 NO	2	2737390	2 NO	4	0.44
30	29 Tripurala E	295	93	1	2	2	7.2	0	6 NO	2	1227548	0 YES	1	0.46
31	30 Venkata Ai	310	99	2	1.5	2	7.3	0	4 YES	1	1076436	1 YES	3	0.54
32	31 Venkata Ja	300	97	2	3	3	8.1	1	3 YES	0	2817288	0 NO	3	0.65
33	32 Yasaswini	327	103	3	4	4	8.3	1	1 YES	0	521975	0 NO	3	0.74
34	33 Yelisetty N	338	118	4	3	4.5	9.4	1	0 YES	0	2285204	0 NO	3	0.91
35	34 D. Prasann	340	114	5	4	4	9.6	1	0 YES	0	1426336	0 NO	2	0.9
36	35 A. Prasann	331	112	5	4	5	9.8	1	0 YES	0	1329579	0 NO	2	0.94
37	36 Bikkumalle	320	110	5	5	5	9.2	1	0 YES	0	2685463	0 NO	2	0.88

## Features And Implementation:

## Architecture/Model:

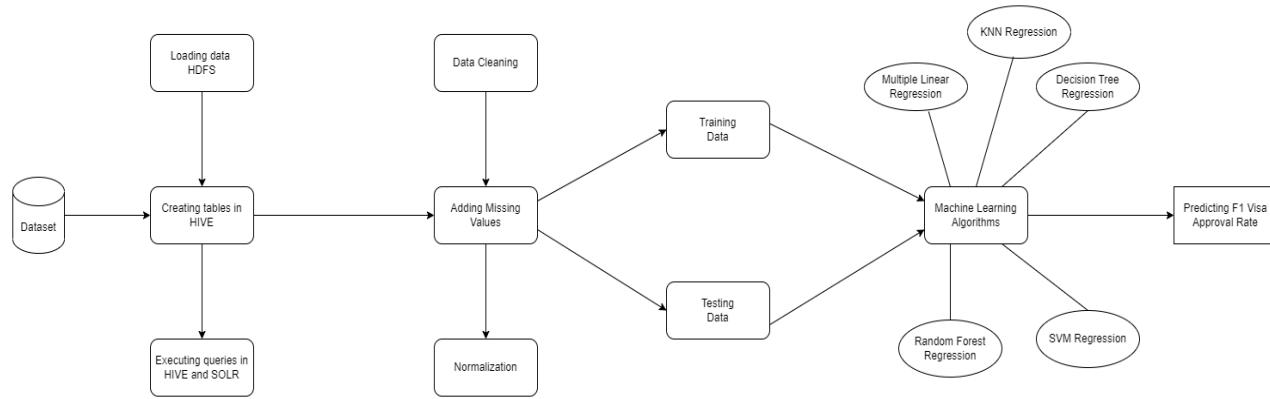


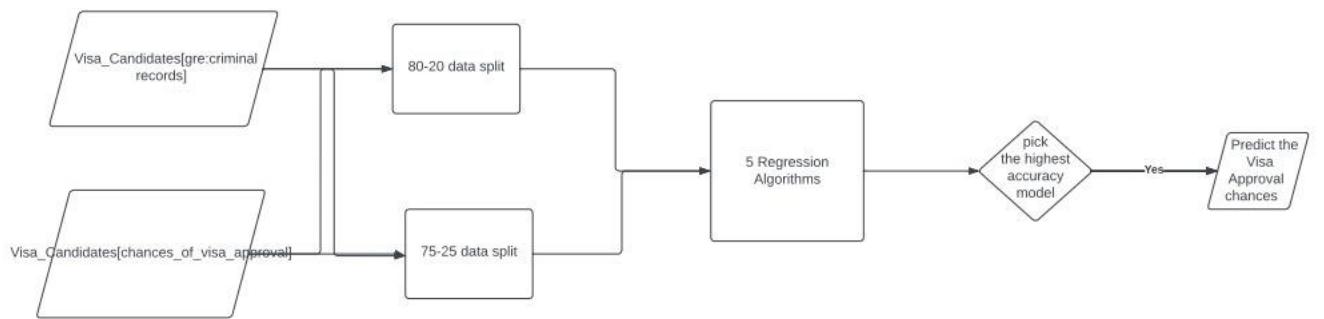
Fig 1. Architecture Model

## Evaluation models:

Firstly, we load the dataset into Hadoop Distributed File System (HDFS). From the data, we create tables in the hive by mentioning appropriate data types and Primary keys. Once the table is created, we can use queries to manipulate the data in these tables and can display results. For executing queries, we have used a Hadoop component called HIVE and SOLR which is a java-based search platform. With this, the data analysis part is completed and later we clean the data and pre-process the data to train ML models.

1. The first step is to fetch an appropriate dataset with a decent number of records that matches the requirement.
2. After fetching the dataset, the next step is to analyze the dataset. In this project, we have used Hadoop ecosystem components for data analysis. In data analysis, we load data into HDFS, create tables in the hive, and use HIVE and SOLR for executing queries
3. Next step after data analysis is data preprocessing. In this step, we clean the data I.e., we eliminate null values and remove any unwanted data. We add any missing value to the columns in the tables.
4. After the above step the dataset is ready, now using the dataset we train ML models by using 75% of available data, and the remaining 25% is used for testing purposes.
5. Furthermore, once the ML models are trained, we use different Machine learning algorithms like K- Nearest neighbor and Random Forest and select the best algorithm which gives the highest accuracy.
6. The last step is to display the chance of F1 visa approval to the student based on the student's profile and we'll suggest in which areas the student can improve his profile to increase the chances of getting approved.

## Detail design of Features:



- As mentioned earlier, to predict the F1 visa approval rate we take the student's profile like student CGPA, GRE and TOFEL scores, work experience, no of backlogs, and no of times previously refused as inputs. We are using HiveQL language to write queries and manipulate data in datasets.
- To visualize the results after analyzing the data we are using excel to generate meaningful pie charts and bar graphs. In later increments, we will use data visualization python libraries like matplotlib to represent results.
- Using our custom dataset, we are training and testing Machine Learning models. These ML models analyze student's profiles and identify the areas where a student is lacking. For instance, If the student has low GRE scores the application suggests reappearing for exams to get better scores.

## Workflow:

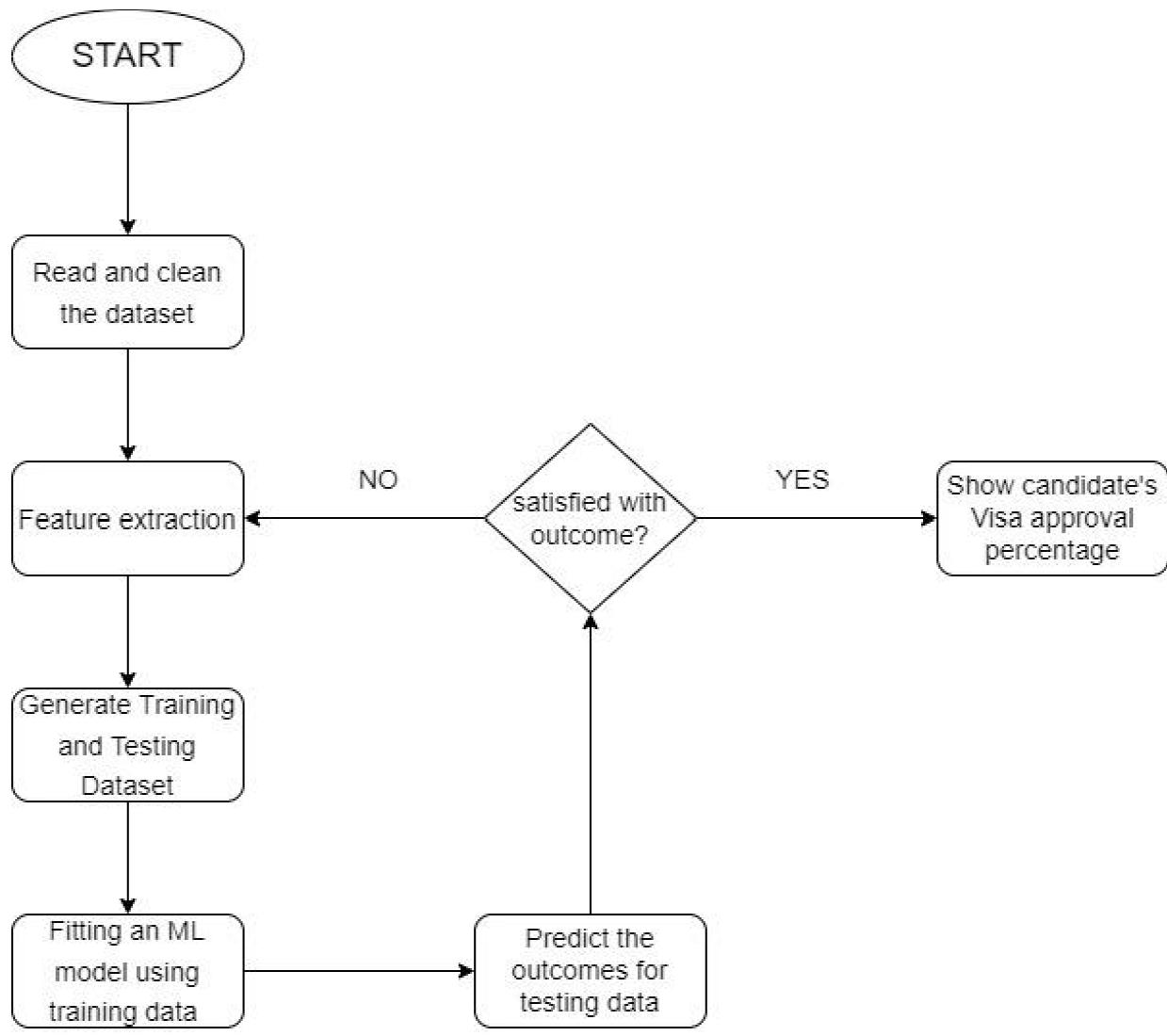


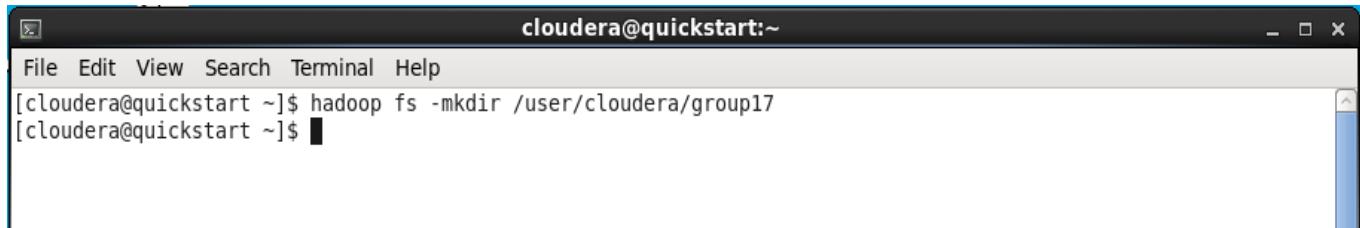
Fig 2. Work Flow Diagram

### **Workflow description:**

We start by collecting data, and after collecting data we read that read and clean using python during the cleaning phase we look for null values and eliminate those values from the data set. Once the data is cleaned and ready, we move to the feature extraction stage. In this phase, we transform some raw data into numerical values. The data thus available is split into two parts, One part is used for training purposes and another part is used for testing the ML models. After that using machine learning algorithms, we predict the outcomes of testing data. If the results are satisfactory then we proceed to the next step which is displaying the chances of getting a visa approved. If the results are not up to the mark, then we move back to the feature extraction step and reiterate the whole process until a satisfactory outcome is produced.

### **Implementation:**

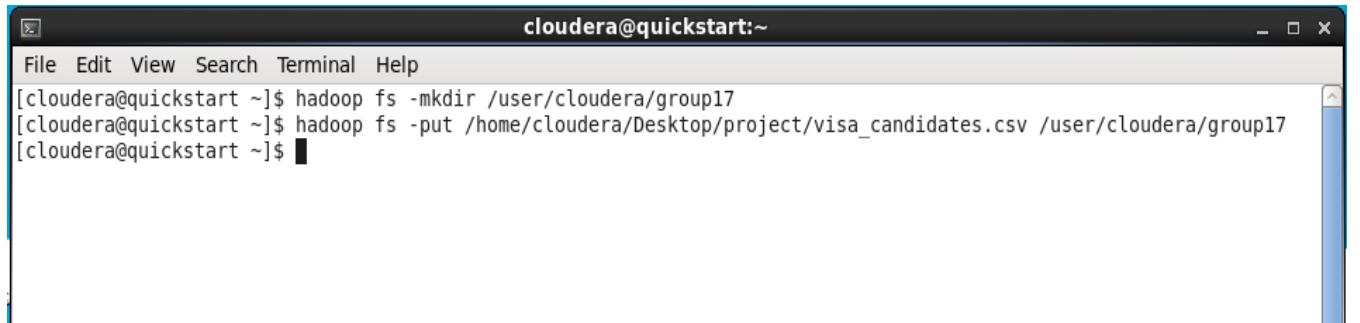
We Installed Virtual Machine on our host system, opened Cloudera in Virtual System, and worked on the Cloudera platform. Open a terminal in Cloudera.



A screenshot of a terminal window titled "cloudera@quickstart:~". The window has a standard Linux-style interface with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar. The terminal prompt is "cloudera@quickstart:~". The user has run the command "hadoop fs -mkdir /user/cloudera/group17", which creates a new directory named "group17" under the "/user/cloudera/" path. The command is followed by a new line character, indicating it is still running or completed.

```
cloudera@quickstart:~$ hadoop fs -mkdir /user/cloudera/group17
```

Now we have created a new directory named “group17” by using the “mkdir” function.



A screenshot of a terminal window titled "cloudera@quickstart:~". The window has a standard Linux-style interface with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar. The terminal prompt is "cloudera@quickstart:~". The user has run two commands: "hadoop fs -mkdir /user/cloudera/group17" to create a directory, and "hadoop fs -put /home/cloudera/Desktop/project/visa\_candidates.csv /user/cloudera/group17" to move a CSV file from the local desktop to the newly created HDFS directory. The command is followed by a new line character, indicating it is still running or completed.

```
cloudera@quickstart:~$ hadoop fs -mkdir /user/cloudera/group17
cloudera@quickstart:~$ hadoop fs -put /home/cloudera/Desktop/project/visa_candidates.csv /user/cloudera/group17
```

We have our source file “visa\_candidates.csv” on the Desktop. Now we are moving our source file from the desktop to the directory “group17” which we have created before.

Visualizing the data in “Hue” to check whether the Dataset loaded into HDFS successfully or not.

```
[cloudera@quickstart ~]$ hdfs dfs -cat /user/cloudera/group17/visa_candidates.csv | head -n 10
id,name,gre_score,toefl_score,university_rating,sop,lor,cgpa,research,no_of_backlogs,relevant_study,year_gaps,parents_annual_income,no_of_times_rejected,us_relatives,work_experience,criminal_records,chance_of_visa_approval
1,Alluri Udgith Reddy,337,118,4,4.5,4.5,9.65,1,0,YES,0,1690680,0,NO,1,0,0.92
2,Annem Naga Sujitha,324,107,4,4.4,4.5,8.87,1,0,YES,0,1207915,0,NO,0,0,0.76
3,Borra Kiran Kumar,316,104,3,3,3.5,8,1,0,YES,0,1761149,0,NO,4,0,0.72
4,Dadisetti V. S. Rajkumar,322,110,3,3.5,2.5,8.67,1,0,YES,0,1983861,0,NO,3,0,0.8
5,Dasari Sai Rishitha,314,103,2,2,3,8.21,0,3,YES,0,588408,0,NO,0,0,0.65
6,J. V. M. Amulya,330,115,5,4.5,3,9.34,1,0,YES,0,1807799,0,NO,3,0,0.9
7,Jampana Swamy Venkata Rama Raju,321,109,3,3,4,8.2,1,1,YES,0,890417,0,NO,1,0,0.75
8,K. Sailesh Kumar,308,101,2,3,4,7.9,0,2,YES,0,2638958,0,NO,1,0,0.68
9,K. Uma Maninder Reddy,302,102,1,2,1.5,8,0,0,NO,1,2230885,0,YES,2,0,0.5
10,Kodali Kiran,323,106,3,3,5.8,8.6,0,0,NO,0,1542780,1,NO,5,0,0.45
11,Kolla Giri,205,106,3,3,5.4,8,4,1,0,YES,1,2331180,0,NO,0,0,0.53
12,Korala Rakesh,327,111,4,4,5,9,1,0,YES,0,734263,0,NO,2,0,0.84
13,Korukonda Vivek,328,112,4,4,4,5.9,1,1,0,YES,0,2815795,0,NO,2,0,0.78
14,Kotnani Hari Chandana,397,199,3,4,3,6.1,0,0,YES,0,559990,0,2,0,0.62
15,Mandalapu Naganya Veena,311,104,3,3,3.5,2,8,2,1,0,YES,0,799880,0,NO,3,0,0.61
16,Mummareddy Harahini,314,105,3,3.5,2,5.8,5,0,0,NO,1,1982987,0,NO,3,0,0.54
17,Nalabothu Vasvi Krishna,317,107,3,4,3.8,7,0,0,YES,0,2568849,0,NO,2,0,0.66
18,Parella Hem Sri Avinash,219,106,3,4,3.8,7,1,1,YES,0,887407,0,YES,5,0,0.65
19,Prathik Prabhakar,318,110,3,4,3.8,8,0,0,YES,0,518192,0,NO,3,0,0.62
20,Ravula Lokesh,300,102,3,3,5,8,0,0,NO,0,2598873,0,NO,2,0,0.62
21,Sattiraju Shashank Arpp,312,107,3,3,2,7,9,1,0,YES,0,1838875,0,NO,2,0,0.64
22,Satya Sravya Peenapati,375,114,4,3,2,8,0,0,0,YES,0,1983331,0,NO,4,0,0.7
23,Sellu Vasvi Krishna,328,116,5,5,5.9,5,1,0,YES,0,1734068,0,NO,3,0,0.94
24,Sravya Tangirala,334,119,5,5,4,5.9,7,1,0,YES,0,2722223,0,NO,1,0,0.95
25,Sree Harsha Venkata Kopissetti,336,119,5,4,3,5.9,8,1,0,YES,0,1269896,0,NO,5,0,0.97
26,Sree Keerthana Kanchi,340,120,5,4,5,4.5,9,6,1,0,YES,0,2315711,0,NO,4,0,0.94
27,Suvvari Pradeep Kumar,322,109,5,4,5.3,5,8,8,0,1,YES,0,1697884,0,NO,1,0,0.76
cat: Unable to write to output stream.
[cloudera@quickstart ~]$
```

We have used the “cat” command and “head” command to display the first ten lines of the Data from our source file “visa\_candidates.csv”

```
[cloudera@quickstart ~]$ hdfs dfs -cat /user/cloudera/group17/visa_candidates.csv | tail -n 10
391,Guntaka Anil Reddy,314,102,2,2,2.5,8.24,0,3,YES,0,1011083,0,NO,1,0,0.64
392,Konduru Hemapushpika,318,106,3,2,3,8.65,0,3,YES,0,1256352,0,NO,5,0,0.71
393,G Varshita Reddy,326,112,4,4,3.5,9.12,1,0,YES,0,2375087,0,NO,4,0,0.84
394,Polamarasetti Manohar Babu,317,104,2,3,3,8.76,0,2,YES,0,2155515,0,NO,3,0,0.77
395,Chettipalli Nikhil,329,111,4,4.5,4,9.23,1,0,YES,0,2422167,0,YES,0,0,0.89
396,Ega Chakri Venkata Challa Reddy,324,110,3,3.5,3.5,9.04,1,0,YES,0,2042907,0,NO,5,0,0.82
397,Sarthak Majhi,325,107,3,3,3.5,9.11,1,0,YES,0,716638,0,NO,0,0,0.84
398,Aryasomayajula Bharadwaja Simha Somayajulu,330,116,4,5,4.5,9.45,1,0,YES,0,771776,0,NO,5,0,0.91
399,Lakshmi Prathyusha Adapa,312,103,3,3.5,4,8.78,0,4,YES,0,2235432,0,YES,1,0,0.67
400,B Roshini,333,117,4,5,4,9.66,1,0,YES,0,659175,0,NO,5,0,0.95
[cloudera@quickstart ~]$
```

Now we are visualizing the last ten lines from our source file by using the above “cat” and “tail” commands.

```
File Edit View Search Terminal Help
hive> create table visa_candidates (id INT,name STRING,gre_score INT,toefl_score INT,university_rating INT,sop FLOAT,lor FLOAT,cgpa FLOAT,research INT,no_of_backlogs INT,relevant_study STRING,year_gaps INT,parents_annual_income INT,no_of_times_rejected INT,us_relatives STRING,work_experience INT,criminal_records INT,chance_of_visa_approval FLOAT) row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 3.099 seconds
hive> load data local inpath '/home/cloudera/Desktop/project/visa_candidates.csv' into table visa_candidates;
Loading data to table default.visa_candidates
Table default.visa_candidates stats: [numFiles=1, totalSize=30743]
OK
Time taken: 1.256 seconds
hive>
```

We Create the tables in HIVE and load the entire dataset.

To load the Data into the table which we have created. HIVE provides us with the facility to load datasets from files that hold on HDFS.

```
File Edit View Search Terminal Help
hive> load data local inpath '/home/cloudera/Desktop/project/visa_candidates.csv' into table visa_candidates;
Loading data to table default.visa_candidates
Table default.visa_candidates stats: [numFiles=1, totalSize=30743]
OK
Time taken: 1.256 seconds
hive>
```

The Command used to load the data is:” LOAD DATA LOCAL INPATH WRITE INTO TABLE”;

Display data in Command Prompt in HIVE

```

cloudera@quickstart:~$ File Edit View Search Terminal Help
hive> alter table visa_candidates set tblproperties("skip.header.line.count"="1");
OK
Time taken: 0.295 seconds
hive> select * from visa_candidates;
OK
1 Alluri Udgith Reddy 337 118 4 4.5 4.5 9.65 1 0 YES 0 1690680 0 NO 1 0 0.92
2 Annem Naga Sujitha 324 107 4 4.0 4.5 8.87 1 0 YES 0 1207915 0 NO 0 0 0.76
3 Borra Kiran Kumar 316 104 3 3.0 3.5 8.0 1 0 YES 0 1761149 0 NO 4 0 0.72
4 Dadisetti V. S. Rajkumar 322 110 3 3.5 2.5 8.67 1 0 YES 0 1983861 0 NO 3 0 0.8
5 Dasari Sai Rishitha 314 103 2 2.0 3.0 8.21 0 3 YES 0 588408 0 NO 0 0 0.65
6 J. V. M. Amulya 330 115 5 4.5 3.0 9.34 1 0 YES 0 1807799 0 NO 3 0 0.9
7 Jampana Swamy Venkata Rama Raju 321 109 3 3.0 4.0 8.2 1 1 YES 0 899417 0 NO 1 0 0.75
8 K. Sailesh Kumar 308 101 2 3.0 4.0 7.9 0 2 YES 0 2638958 0 NO 1 0 0.68
9 K. Uma Maninder Reddy 302 102 1 2.0 1.5 8.0 0 0 NO 1 2230885 0 YES 2 0 0.5
10 Kodali Kiran 323 108 3 3.5 3.0 8.6 0 0 NO 0 1542788 1 NO 5 0 0.45
11 Koli Siri 325 106 3 3.5 4.0 8.4 1 0 YES 1 1233102 0 NO 0 0 0.52
12 Korada Rakesh 327 111 4 4.0 4.5 9.0 1 0 YES 0 734263 0 NO 2 0 0.84
13 Korukonda Vivek 328 112 4 4.0 4.5 9.1 1 0 YES 0 2515795 0 NO 2 0 0.78
14 Kotnani Hari Chandana 307 109 3 4.0 3.0 8.0 1 0 YES 0 550998 0 NO 2 0 0.62
15 Mandalapu Manogna Veena 311 104 3 3.5 2.0 8.2 1 0 YES 0 709688 0 NO 0 0 0.61
16 Mummarreddy Harshini 314 105 3 3.5 2.5 8.3 0 0 NO 1 1902967 0 NO 3 0 0.54
17 Nalabothu Vamsi Krishna 317 107 3 4.0 3.0 8.7 0 0 YES 0 2560049 0 NO 2 0 0.66
18 Perella Hem Sai Avinash 319 106 3 4.0 3.0 8.0 1 1 YES 0 867467 0 YES 5 0 0.65
19 Prathik Prabhakar 318 110 3 4.0 3.0 8.8 0 0 YES 0 618192 0 NO 3 0 0.63
20 Ravela Lokesh 303 102 3 3.0 8.5 9.0 0 0 NO 0 2598373 0 NO 2 0 0.62
21 Sattiraju Shashank Ampv 312 107 3 3.0 2.0 7.9 1 0 0 1830373 0 NO 3 0 0.64
22 Satya Sravya Peesapati 325 114 4 3.0 2.0 8.4 0 0 YES 0 1983331 0 NO 4 0 0.7
23 Sollem Vamsi Krishna 328 116 5 5.0 5.0 9.5 1 0 YES 0 1734089 0 NO 3 0 0.94
24 Sravya Tangirala 334 119 5 5.0 4.5 9.7 1 0 YES 0 2722223 0 NO 1 0 0.95
25 Sree Harsha Venkata Koppisetty 336 119 5 4.0 3.5 9.8 1 0 YES 0 1260806 0 NO 5 0 0.97
26 Sree Keerthana Kanchi 340 120 5 4.5 4.5 9.6 1 0 YES 0 2315711 0 NO 4 0 0.94
27 Suvarvi Pradeep Kumar 322 109 5 4.5 3.5 8.8 0 1 YES 0 1607884 0 NO 1 0 0.76
28 Toship Sonkusare 298 98 2 1.5 2.5 7.5 1 0 NO 2 2737398 2 NO 4 0 0.44
29 Tripurala Durga Aswin Kumar 295 93 1 2.0 2.0 7.2 0 4 NO 2 1227548 0 YES 1 0 0.46
30 Venkata Arun Vavilakolanu 310 99 2 1.5 2.0 7.3 0 4 YES 1 1076436 1 YES 3 0 0.54

```

Using “select” command we displayed the data that is loaded into the table by this we can confirm that the data is successfully loaded into the table

## Visualize un Hue

**Properties**

- Table: cloudera
- Location: 1 files
- Created: 10/30/2022 8:42 PM
- Type: Managed

**COLUMNS (18)**

Name	Type	Comment
1 id	int	Add a comment...
2 i name	string	Add a comment...
3 i gre_score	int	Add a comment...
4 i toefl_score	int	Add a comment...
5 i university_rating	int	Add a comment...

**SAMPLE**

id	name	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	relevant_study	year_gaps	parents_annual_income	no_of_times_rejected	us_relatives	work
1 NULL	name	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	YES	0	1690680	0	NO	1
2 1	Alluri Udgith Reddy	337	118	4	4.5	4.5	9.649999618530273	1	0	YES	0	1207915	0	NO	1
3 2	Annem Naga Sujitha	324	107	4	4	4.5	8.8699998559082	1	0	YES	0	1207915	0	NO	0

Hue - Table Browser - Mozilla Firefox

about:sessionrestore | Cloudera Live : Welcome | Solr Admin | Solr Admin | Hue - Table Browser | X

( quickstart.cloudera:8888/hue/metastore/table/default/visa\_candidates) | Search | ☆ 自 锁 家 回 三

Cloudera Cloudera Manager Getting Started

**HUE** Query Search data and saved documents... Jobs cloudera

	id	name	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	relevant_study	year_gaps	parents_annual_income	no_of_times_rejected	us_relatives	work_experience
38	37	Bommu Mohanti Reddy	299	106	2	4	4	8.399999618530273	0	1	YES	0	995084	0	NO	2
39	38	C Aditya	300	105	1	1	2	7.800000190734863	0	0	YES	0	1743961	0	NO	2
40	39	Duggirreddy Nikhil Reddy	304	105	1	3	1.5	7.5	0	5	NO	1	1674624	1	YES	1
41	40	Edara Mohitha	307	108	2	4	3.5	7.699999809265137	0	8	NO	2	2458927	0	NO	0
42	41	G Supriya Sai	308	110	3	3.5	3	8	1	0	NO	2	652191	0	NO	1
43	42	G. Naga Ramanjaney Reddy	316	105	2	2.5	2.5	8.199999809265137	1	1	YES	2	608221	0	NO	5
44	43	Ganti Haripriya	313	107	2	2.5	2	8.5	1	0	YES	1	2207218	0	NO	1
45	44	N Jahnavi	332	117	4	4.5	4	9.100000381469727	0	0	YES	0	2200027	0	NO	2
46	45	Krishna Vivek Vinakota	326	113	5	4.5	4	9.399999618530273	1	0	YES	0	2469560	0	NO	4
47	46	Mahati Madhra	322	110	5	5	4	9.100000381469727	1	0	YES	0	1905801	0	NO	5
48	47	P Rishi Kumar Srivatsa	329	114	5	4	5	9.300000190734863	1	0	YES	0	1549084	0	NO	0
49	48	Pabbathi Niharika	339	119	5	4.5	4	9.699999809265137	0	0	YES	0	1024258	0	NO	0
50	49	Peraval Manasa	321	110	3	3.5	5	8.850000381469727	1	0	YES	0	1336250	0	NO	2
51	50	Pinninti Rithvik Reddy	327	111	4	3	4	8.399999618530273	1	2	YES	0	2975122	0	NO	1
52	51	Sushma Pasupula Nagabhusanam	313	98	3	2.5	4.5	8.300000190734863	1	2	YES	0	2321817	0	NO	0
53	52	Raja Yagnesh	312	100	2	1.5	3.5	7.900000095367432	1	7	YES	2	2395045	0	YES	4
54	53	Ramireddy Ranapratap	334	116	4	4	3	8	1	2	YES	0	2924353	0	NO	5
55	54	Raveena Alluri	324	112	4	4	2.5	8.100000381469727	1	2	YES	0	632372	0	NO	2
56	55	Sanjana Bhogavalli	322	110	3	3	3.5	8	0	2	YES	0	1245120	0	NO	3
57	56	Satish Kari	320	103	3	3	3	7.699999809265137	0	5	YES	1	849227	0	NO	2
58	57	Sumanth Thota	316	102	3	2	3	7.400000095367432	0	4	YES	1	1543945	0	NO	0
59														NO	2	

Displaying the loaded dataset in HUE browser by which we can check the data that is loaded into the table we created is

### Finding Names and scores whose gre>320 and toefl>115

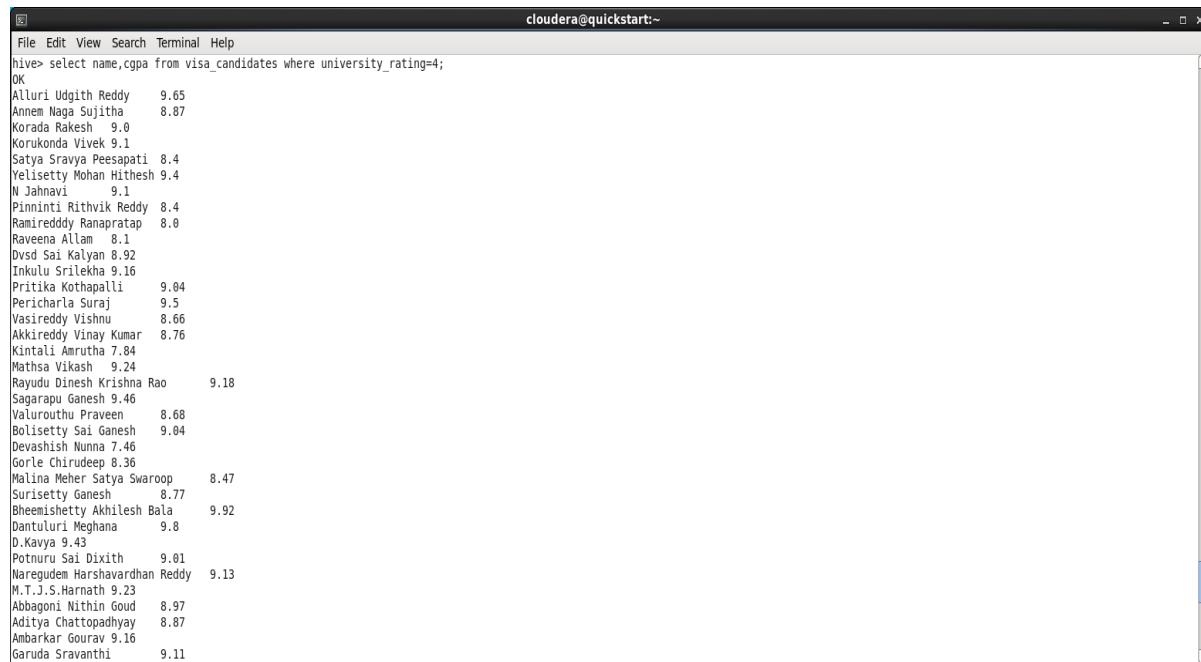
```
File Edit View Search Terminal Help
cloudera@quickstart:~
```

```
hive> select name, gre_score, toefl_score from visa_candidates where gre_score>320 AND toefl_score>115;
OK
Alluri Udgith Reddy      337    118
Sollem Vamsi Krishna    328    116
Sravya Tangirala        334    119
Sree Harsha Venkata Koppisetty 336    119
Sree Keerthana Kanchi   340    120
Yelisetty Mohan Hithesh 338    118
N Jahnavi                332    117
Pabbathi Niharika       339    119
Ramireddy Ranapratap   334    116
Rugved Jayavarapu       332    118
Pericharla Suraj         340    120
Mandalapu Purnachandra Rao 331    120
Mathsa Vikash            332    119
Sagarapu Ganesh          338    117
Soumya Gupta              331    116
D G Anjan Kumar          335    117
Duggirala Kusuma Neha Gandhi 334    119
Peddi Sai Pankaj          333    118
Yama Hemanth              326    116
B. S. N. S. Bhavya Sri   332    118
Bheemishetty Akhilesh Bala 340    120
Dantuluri Meghana        339    116
G Pranavi                 332    116
Unnati Goyal               334    117
Ambarkar Gourav            329    119
Jami Sai Likhita          335    118
Rodda Chanikya Reddy      336    118
Divya Pragya               340    120
Donkada Pratyusha          334    120
Gorla Chandrakleha        325    116
Kadiyala Jayasimha        338    120
Kata Akhil                 333    119
Kesiraju Abhishek          331    117
Kishan Kamlesh             330    116
Sai Praveen Kumar Salapu  331    119
Md Vazir Ali               331    116
```

We are listing out the people who secured “gre” score above 320 and “toefl” score above 115 by using the above queries.

The main purpose of executing this query to list the people with high GRE and TOEFL scores so they have the most visa acceptable.

## Finding names and cgpa for top universities



```
cloudera@quickstart:~$ hive> select name,cgpa from visa_candidates where university_rating=4;
OK
Alluri Udgith Reddy      9.65
Anmem Naga Sujitha      8.87
Korada Rakesh            9.0
Korukonda Vivek          9.1
Satya Sravya Peesapati   8.4
Yelisetty Mohan Hithesh  9.4
N Jahnavi                9.1
Pinninti Rithvik Reddy   8.4
Ramireddy Ranaprataap    8.8
Raveena Alam              8.1
Dvsd Sai Kalyan           8.92
Inkulu Srilekhya         9.16
Pritika Kothapalli       9.04
Perichala Suraj           9.5
Vasireddy Vishnu          8.66
Akkireddy Vinay Kumar     8.76
Kintali Amrutha           7.84
Mathsa Vikash             9.24
Rayudu Dinesh Krishna Rao 9.18
Sagarapu Ganesh           9.46
Valurouthu Praveen        8.68
Bolisetti Sai Ganesh      9.04
Devarashish Nunna          7.46
Gorle Chirudeep           8.36
Malina Meher Satya Swaroop 8.47
Surisetty Ganesh           8.77
Bheemisnetty Akhilash Bala 9.92
Dantuluri Meghana         9.8
D.Kavya                   9.43
Potnuru Sai Dixith        9.01
Naregudem Harshvardhan Reddy 9.13
M.T.J.S.Harnath            9.23
Abbagoni Nithin Goud      8.97
Aditya Chattopadhyay       8.87
Ambarkar Gourav             9.16
Garuda Sravanthi           9.11
```

We are finding the people who are form top rated universities by using the above queries. In querie we analyze the dataset and provided the cgpa and name of students from top rated

universities.

CGPA of the students also comes into picture when considering for the visa approval so we have taken this query for checking the candidates.

## Avg of visa approval



```
cloudera@quickstart:~$ hive> select avg(chance_of_visa_approval) from visa_candidates;
Query ID = cloudera_20221030153232_600d066b-c289-454f-9e55-9b9e502f6cff
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1664065500663_0001, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1664065500663_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1664065500663_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-10-30 15:33:12,189 Stage-1 map = 0%,  reduce = 0%
2022-10-30 15:33:29,934 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.41 sec
2022-10-30 15:33:46,311 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.98 sec
MapReduce Total cumulative CPU time: 4 seconds 980 msec
Ended Job = job_1664065500663_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1  Cumulative CPU: 4.98 sec  HDFS Read: 41215 HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 980 msec
OK
0.724349991297722
Time taken: 70.629 seconds, Fetched: 1 row(s)
hive>
```

We are finding the average visa approval rate by using above querie. This quire calls the mapper, reducer because the task executed through MapReduce process.

### Finding people with more criminal records

```
cloudera@quickstart:~
```

```
File Edit View Search Terminal Help
hive> select count(*) from visa_candidates where criminal_records>0;
Query ID = cloudera_20221030155151_db0d7221-ab42-44cd-9dd8-92cd50e19a4a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1664065500663_0002, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1664065500663_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1664065500663_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-10-30 15:51:25,815 Stage-1 map = 0%, reduce = 0%
2022-10-30 15:51:42,461 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.78 sec
2022-10-30 15:51:58,423 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 5.38 sec
MapReduce Total cumulative CPU time: 5 seconds 380 msec
Ended Job = job_1664065500663_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.38 sec HDFS Read: 41567 HDFS Write: 3 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 380 msec
OK
11
Time taken: 49.699 seconds, Fetched: 1 row(s)
hive>
```

In order for easy verification and approval of visa for any person their criminal record is checked all the time. Now here we are finding the number of people who have criminal record on their profile. For this task the above querie calls the MapReduce process.

People with more criminal records have the least visa approval rate so, A condition is checked that people with more criminal records the least chance of visa approval.

### Avg of parent's income

```
cloudera@quickstart:~
```

```
File Edit View Search Terminal Help
hive> select avg(parents_annual_income) from visa_candidates;
Query ID = cloudera_20221030155858_b99f6e60-ae19-468f-b67a-5150cb5eb793
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1664065500663_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1664065500663_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1664065500663_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-10-30 15:58:29,703 Stage-1 map = 0%, reduce = 0%
2022-10-30 15:58:44,728 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.04 sec
2022-10-30 15:59:00,674 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.59 sec
MapReduce Total cumulative CPU time: 4 seconds 590 msec
Ended Job = job_1664065500663_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.59 sec HDFS Read: 41326 HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 590 msec
OK
1683063.7964824121
Time taken: 47.716 seconds, Fetched: 1 row(s)
```

For some particular visa's like f1 and j1, we need check the income in their bank account on in their parent's bank account as most common of all these are "F1" visas which student visa so some has to sponsor them for their education. So, we are Finding the average income of parents of people who applied for visa. the above querie uses the MapReduce process to analyze the large data and provide accurate output.

## Finding the top ten people visa approval rate

```
cloudera@quickstart:~
```

```
File Edit View Search Terminal Help
hive> select name,chance_of_visa_approval from visa_candidates order by chance_of_visa_approval desc limit 10;
Query ID = cloudera_20221101143636_fafb3731-76e9-4f31-9b5b-8f443327083a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1664065500663_0005, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1664065500663_0005/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1664065500663_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-11-01 14:37:12,614 Stage-1 map = 0%,  reduce = 0%
2022-11-01 14:37:25,602 Stage-1 map = 100%,  reduce = 0%
2022-11-01 14:37:41,400 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.69 sec
MapReduce Total cumulative CPU time: 4 seconds 690 msec
Ended Job = job_1664065500663_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1  Cumulative CPU: 4.69 sec  HDFS Read: 39860 HDFS Write: 252 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 690 msec
OK
Sree Harsha Venkata Koppisetty 0.97
Bheemishetty Akhilesh Bala 0.97
Donkada Pratyusha 0.97
Divya Pragya 0.97
Siddani Mahesh 0.96
Potluri Sai Cherish 0.96
Dantuluri Meghana 0.96
Kata Akhil 0.96
Pericharla Suraj 0.96
Kunadharaju Sai Srinivasa Varma 0.96
Time taken: 44.843 seconds, Fetched: 10 row(s)
hive>
```

Based on the data we find the top people with visa approval rate for this task we used above queries which calls the MapReduce which retries the accurate data

To display the top 10 students with high visa approval rate so that they can get an idea about who are the highest visa approved students.

## Finding the last ten people visa approval rate

```
cloudera@quickstart:~  
File Edit View Search Terminal Help  
hive> select name,chance_of_visa_approval from visa_candidates order by chance_of_visa_approval limit 10;  
Query ID = cloudera_20221101144040_c8186f8c-3cb4-4488-8414-5af71826895c  
Total jobs = 1  
Launching Job 1 out of 1  
Number of reduce tasks determined at compile time: 1  
In order to change the average load for a reducer (in bytes):  
  set hive.exec.reducers.bytes.per.reducer=<number>  
In order to limit the maximum number of reducers:  
  set hive.exec.reducers.max=<number>  
In order to set a constant number of reducers:  
  set mapreduce.job.reduces=<number>  
Starting Job = job_1664065500663_0006, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1664065500663_0006/  
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1664065500663_0006  
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1  
2022-11-01 14:40:16,078 Stage-1 map = 0%,  reduce = 0%  
2022-11-01 14:40:29,856 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.06 sec  
2022-11-01 14:40:44,735 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.62 sec  
MapReduce Total cumulative CPU time: 4 seconds 620 msec  
Ended Job = job_1664065500663_0006  
MapReduce Jobs Launched:  
Stage-Stage-1: Map: 1  Reduce: 1  Cumulative CPU: 4.62 sec  HDFS Read: 39860 HDFS Write: 254 SUCCESS  
Total MapReduce CPU Time Spent: 4 seconds 620 msec  
OK  
Mukala Venkata Durga Kalyan      0.34  
Kalla Girish Pavan Kumar        0.34  
V.L.Rachana Priya                0.36  
Katukuri Pavankumar Reddy       0.36  
Mounica Satya Kavya Dakoju     0.38  
Brunji Manish Kumar              0.38  
Moningi Madhumita                0.39  
Kintali Amrutha                 0.42  
V.Anuhya Varma                  0.42  
Sahil Rana                      0.42  
Time taken: 45.354 seconds, Fetched: 10 row(s)  
hive>
```

Based on the data we find the least people with visa approval rate for this task we used above queries which calls the MapReduce which retries the accurate data

To display the least 10 students with high visa approval rate so that they can get an idea about who are the least visa approved students.

## Final query

```
cloudera@quickstart:~  
File Edit View Search Terminal Help  
hive> select name,chance_of_visa_approval from visa_candidates where gre_score>320 AND toefl_score>115 AND university_rating>4 AND cgpa>9 AND no_of_backlogs=0 AND year_gaps=0 AND work_experien  
ce>2;  
OK  
Soilem Vamsi Krishna    0.94  
Sree Harsha Venkata Koppisetti  0.97  
Sree Keerthana Kanchi    0.94  
D G Anjan Kumar 0.94  
Duggirala Kusum Neha Gandhi  0.94  
Peddi Sai Pankaj          0.92  
G Pranavi                0.94  
Donkada Pratyusha         0.97  
Kishan Kamlesh             0.93  
Moleti Reshma Sree        0.92  
Hemant Kumar Singh        0.94  
Veluru Anesh Kumar Chowdary 0.96  
Time taken: 0.126 seconds, Fetched: 12 row(s)  
hive>
```

Here we are displaying the data of people whose profile is perfectly suited for visa approval based on their GRE, TOEFL scores along with the rating of their university and CGPS secured.

## SOLR

Now we are performing several tasks on the visa\_candidates dataset using the SOLR tool



```
cloudera@quickstart:~$ solrctl instancedir --generate /home/cloudera/Desktop/project/visa_candidates
[cloudera@quickstart ~]$
```

The above command is used for creating a new instance for the “visa\_candidates” and you can see a folder is created on the name “visa\_candidates” in which all the schema and other configurations are stored.

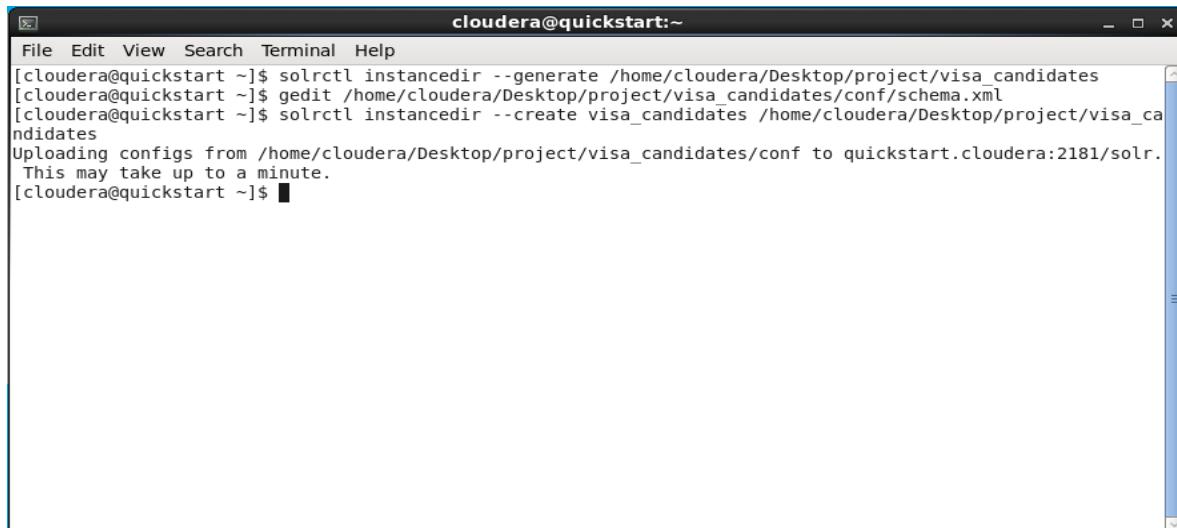
### Changing Schema



```
cloudera@quickstart:~$ solrctl instancedir --generate /home/cloudera/Desktop/project/visa_candidates
[cloudera@quickstart ~]$ gedit /home/cloudera/Desktop/project/visa_candidates/conf/schema.xml
```

We need to edit the schema because all are not present in the old schema to add new features we are changing the schema using gedit.

We will change the schema in the schema.xml



```
cloudera@quickstart:~$ solrctl instancedir --generate /home/cloudera/Desktop/project/visa_candidates
[cloudera@quickstart ~]$ gedit /home/cloudera/Desktop/project/visa_candidates/conf/schema.xml
[cloudera@quickstart ~]$ solrctl instancedir --create visa_candidates /home/cloudera/Desktop/project/visa_candidates
Uploading configs from /home/cloudera/Desktop/project/visa_candidates/conf to quickstart.cloudera:2181/solr.
This may take up to a minute.
[cloudera@quickstart ~]$
```

Here in the --create command the contents of the instance directory are uploaded to the zookeeper.

```

cloudera@quickstart:~$ solrctl instancedir --generate /home/cloudera/Desktop/project/visa_candidates
[cloudera@quickstart ~]$ gedit /home/cloudera/Desktop/project/visa_candidates/conf/schema.xml
[cloudera@quickstart ~]$ solrctl instancedir --create visa_candidates /home/cloudera/Desktop/project/visa_ca
ndidates
Uploading configs from /home/cloudera/Desktop/project/visa_candidates/conf to quickstart.cloudera:2181/solr.
This may take up to a minute.
[cloudera@quickstart ~]$ solrctl collection --create visa_candidates
[cloudera@quickstart ~]$

```

Here the “--create visa\_candidates” command a new collection is created in the solr which is named visa\_candidates

## Loading Data into SOLR

The screenshot shows the Solr Admin interface in Mozilla Firefox. The URL is `quickstart.cloudera:8983/solr/#/visa_candidates_shard1_replica1/documents`. The left sidebar shows various Cloudera services like Hue, Hadoop, HBase, Impala, Spark, Oozie, and Cloudera Manager. The main panel has a 'Request-Handler (qt)' dropdown set to 'update'. A 'Document type' dropdown is set to 'CSV'. The 'Documents(s)' section contains a list of 10 documents with fields such as id, name, gpa, score\_toefl, score\_university, rating, sop, cgpa, research, no\_of\_backlogs, relevant\_study\_year, gaps, parents, annual\_income, no\_of\_times\_rejected, us\_relatives, work\_experience, crm, marital\_records, chance\_of\_visa\_approval, and others. The right side shows the 'Status: SUCCESS' and the 'Response' JSON output.

```

Status: SUCCESS
Response:
{
  "responseHeader": {
    "status": 0,
    "QTime": 492
  }
}

```

For uploading the data into solr we follow these steps, Open solr and go to Core-Admin there you can find the visa\_candidates. Now select the visa\_candidates from the selector and make sure the document type is ‘CSV’ then upload all the data from the visa\_candidates dataset into the document section of solr and then click submit Document button for submitting.

## Displaying the data

The screenshot shows the Solr Admin interface in Mozilla Firefox. The URL is `http://quickstart.cloudera:8983/solr/#/visa_candidates_shard1_replica1/query`. The search query is `*;*`. The results show 483 documents. One document is highlighted, showing fields like id, name, gre\_score, toefl\_score, etc.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "start": "1",
      "q": "*;*",
      "_": "1667239673136",
      "wt": "json",
      "rows": "1000"
    }
  },
  "response": {
    "numFound": 483,
    "start": 1,
    "docs": [
      {
        "id": "ss",
        "name": "fff",
        "_version_": 1748175530635145200
      },
      {
        "id": "change.me",
        "title": [
          "change.me"
        ],
        "_version_": 1748176020542849000
      },
      {
        "id": "1",
        "name": "Alluri Udgith Reddy",
        "gre_score": 337,
        "toefl_score": 118,
        "university_rating": 4,
        "sop": 4.5,
        "spclcheck": null
      }
    ]
  }
}
```

Here we are listing all the data of visa-applied candidates, we used the above queries “\*;\*” we should add the id, name, gre\_score, .. etc. then the data will be retrieved. The above im is the output.

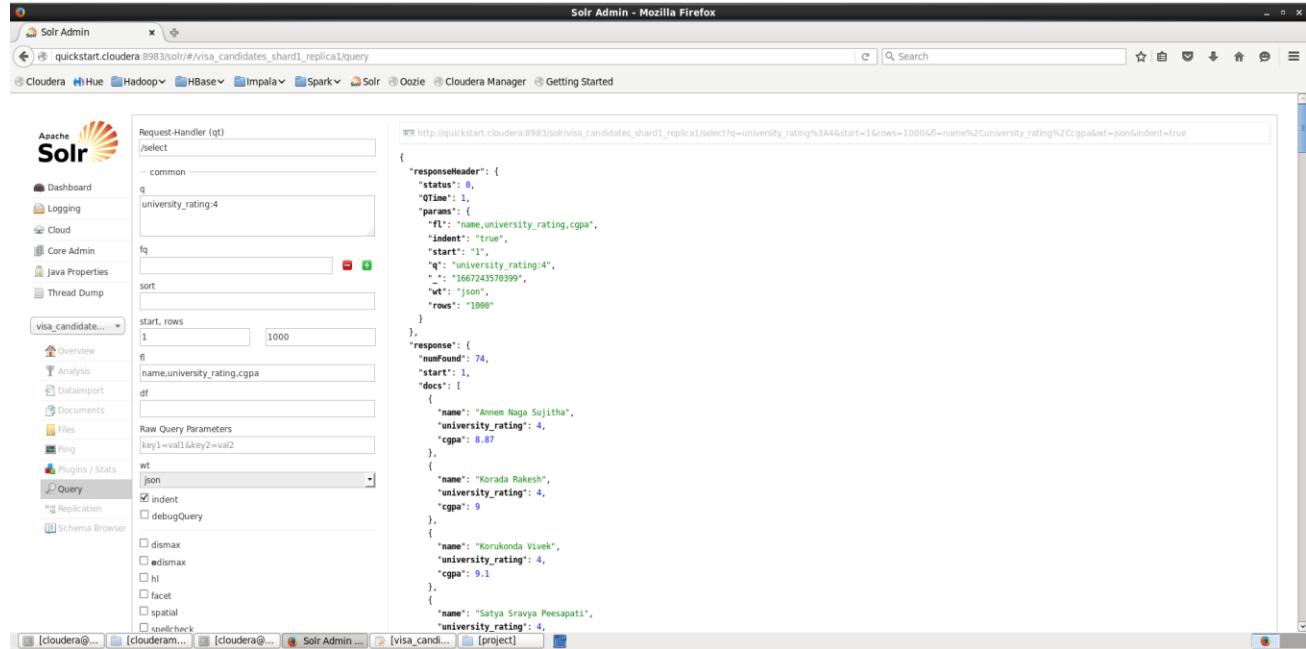
### Finding ‘gre’ and ‘toefl’ scores

The screenshot shows the Solr Admin interface in Mozilla Firefox. The URL is `http://quickstart.cloudera:8983/solr/#/visa_candidates_shard1_replica1/query`. The search query is `gre_score: {320 TO *} AND toefl_score: {115 TO *}`. The results show 44 documents. One document is highlighted, showing fields like name, gre\_score, toefl\_score, etc.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "q": "gre_score: {320 TO *} AND toefl_score: {115 TO *}",
      "fq": null,
      "sort": null,
      "start": "1",
      "rows": "1000",
      "fl": "name,gre_score,toefl_score",
      "df": null,
      "rawQuery": "key1=val1&key2=val2",
      "wt": "json",
      "indent": true,
      "debugQuery": false
    }
  },
  "response": {
    "numFound": 44,
    "start": 1,
    "docs": [
      {
        "name": "Soleem Vamsi Krishna",
        "gre_score": 320,
        "toefl_score": 116
      },
      {
        "name": "Sravya Tangirala",
        "gre_score": 334,
        "toefl_score": 119
      },
      {
        "name": "Sree Harsha Venkata Kopisetti",
        "gre_score": 336,
        "toefl_score": 119
      },
      {
        "name": "Sree Keerthana Kanchi",
        "gre_score": 340,
        "toefl_score": 120
      }
    ]
  }
}
```

Here we are displaying the score and names of people who secured marks above 320 in ‘GRE’ and above 115 in ‘TOEFL’. To display this we used gre\_scores and toefl\_scores and combined both relations with the ‘AND’ Operator.

## Finding University Rating

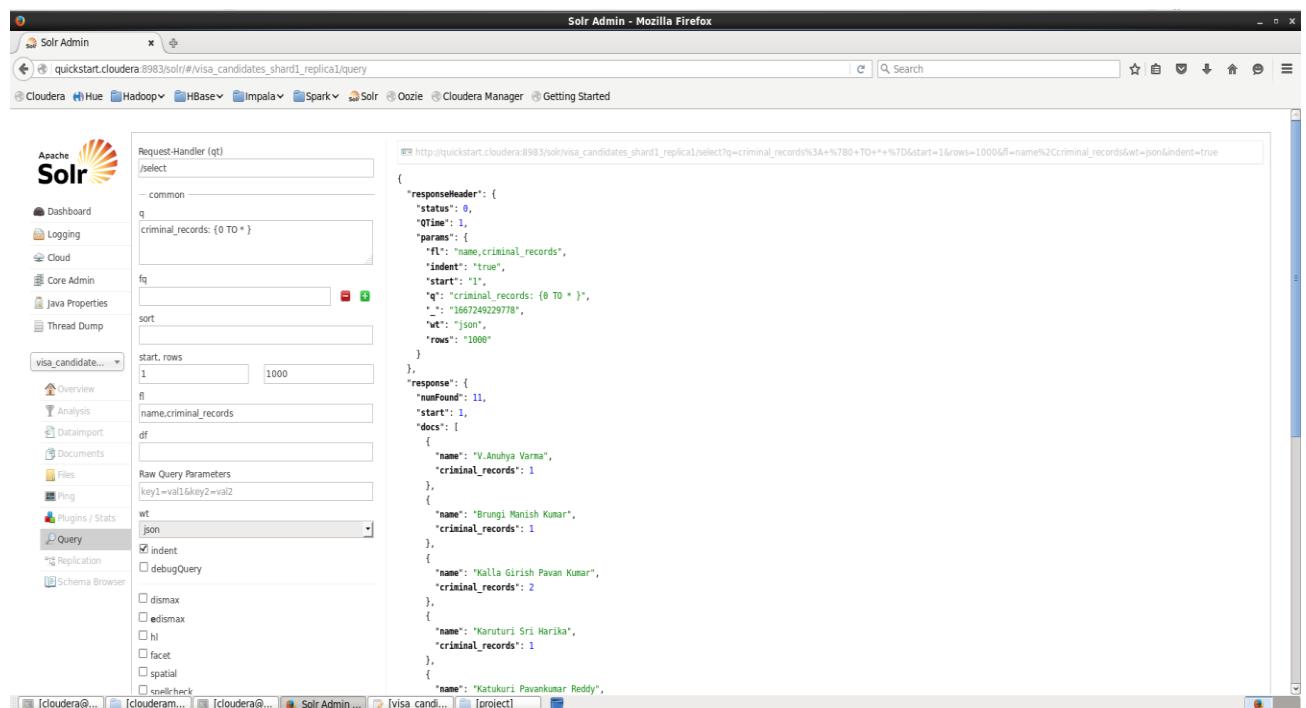


The screenshot shows the Apache Solr Admin interface in Mozilla Firefox. The URL is `http://quickstart.cloudera:8983/solr/#/visa_candidates_shard1_replica1/query`. The search query is `university_rating:4`. The results are displayed in JSON format, showing four documents with name, university rating, and cgsa fields. The first document is Anne Naga Sujitha with a university rating of 4 and a cgsa of 8.87.

```
[{"name": "Anne Naga Sujitha", "university_rating": 4, "cgsa": 8.87}, {"name": "Korada Rakesh", "university_rating": 4, "cgsa": 9.0}, {"name": "Korukonda Vivek", "university_rating": 4, "cgsa": 9.1}, {"name": "Satya Sravya Peesapati", "university_rating": 4, "cgsa": 9.0} ]
```

To display the university rating we should use the “university\_rating:4” query so the universities whose rating is 4 will be retrieved above image is the output.

## Total Criminal Records



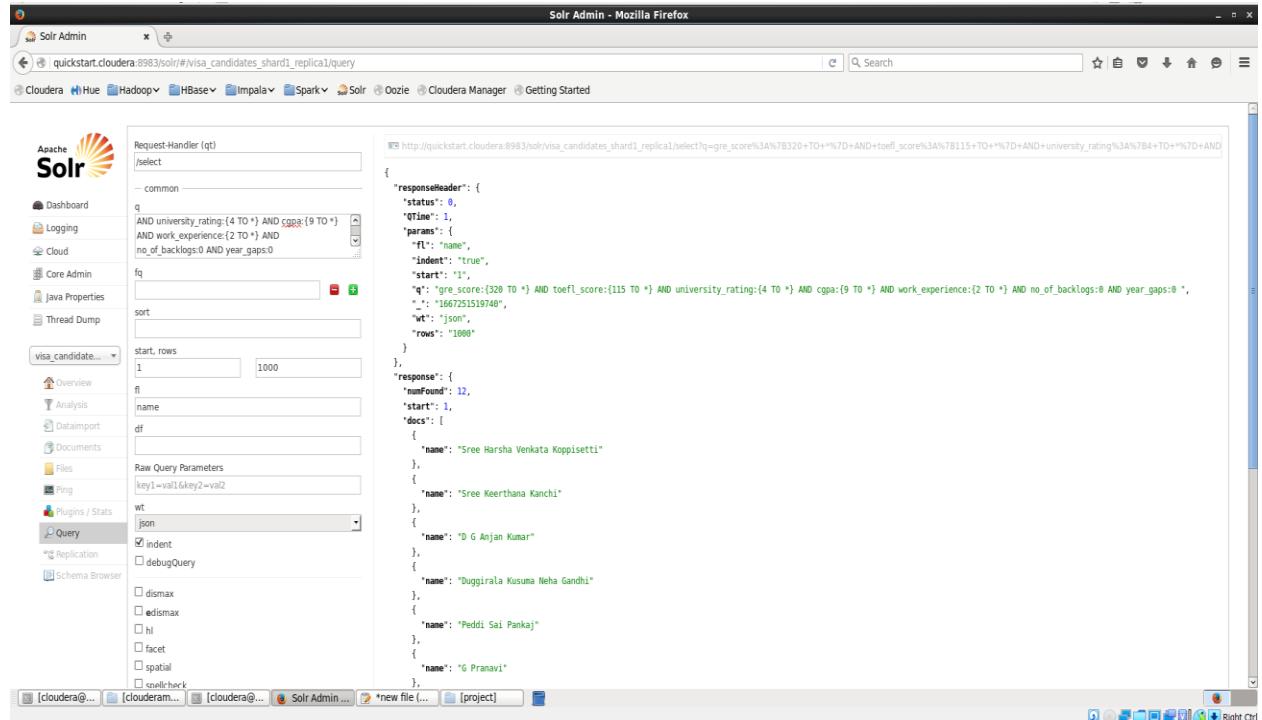
The screenshot shows the Apache Solr Admin interface in Mozilla Firefox. The URL is `http://quickstart.cloudera:8983/solr/#/visa_candidates_shard1_replica1/query`. The search query is `criminal_records: {0 TO *}`. The results are displayed in JSON format, showing three documents with name and criminal\_records fields. The first document is Anehuja Varma with one criminal record.

```
[{"name": "Anehuja Varma", "criminal_records": 1}, {"name": "Brungi Manish Kumar", "criminal_records": 1}, {"name": "Kalla Girish Pavan Kumar", "criminal_records": 2}, {"name": "Karuturi Sri Harika", "criminal_records": 1}, {"name": "Katukuri Pavankumar Reddy", "criminal_records": 0} ]
```

To display people who have criminal records we must use the “criminal\_records:{0 to \*}”

query , in which the values inside flower brackets are from min number to maximum number , here '\*' means topmost number in the dataset and this will provide the output with the people name and their criminal records count .

## Final Query



The screenshot shows the Solr Admin interface in Mozilla Firefox. The URL in the address bar is `http://quickstart.cloudera:8983/solr/visa_candidates_shard1_replica1/query`. The left sidebar shows various Solr configuration tabs like Dashboard, Logging, Cloud, Core Admin, Java Properties, Thread Dump, and a selected 'Query' tab. The main panel displays a search form with the following parameters:

- Request-Handler (qt): /select
- q: AND university\_rating:{4 TO \*} AND cgpa:{9 TO \*} AND work\_experience:{2 TO \*} AND no\_of\_backlogs:0 AND year\_gaps:0
- fq: (no input)
- sort: (no input)
- start, rows: start=1, rows=1000
- fl: name, df: (no input)
- Raw Query Parameters: key1=val1&key2=val2
- wt: json
- indent: checked
- debugQuery: unchecked
- checkboxes: dismax, edismax, hl, facet, spatial, spellcheck

The right panel shows the JSON response for the search query:

```

{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "fl": "name",
      "indent": "true",
      "start": "1",
      "q": "pre_score:[320 TO *] AND toefl_score:[115 TO *] AND university_rating:[4 TO *} AND cgpa:[9 TO *} AND work_experience:[2 TO *} AND no_of_backlogs:0 AND year_gaps:0",
      "wt": "json",
      "rows": "1000"
    }
  },
  "response": {
    "numFound": 12,
    "start": 1,
    "docs": [
      {
        "name": "Sree Harsha Venkata Kopisetti"
      },
      {
        "name": "Sree Keerthana Kanchi"
      },
      {
        "name": "D G Anjan Kumar"
      },
      {
        "name": "Duggirala Kusuma Neha Gandhi"
      },
      {
        "name": "Peddi Sai Pankaj"
      },
      {
        "name": "G Pranavi"
      }
    ]
  }
}

```

Here we are finding the names of people with best possible profile based on the uploaded data by the above command, from which we are sorting out the people who has university rating of 4 and above ,CGPA of 9 and above, work\_experience of 2 plus years with zero backlogs and no year\_gap.

## Cassandra

### Opening of Cassandra

```
C:\Windows\System32\cmd.exe - cqlsh
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\apache-cassandra-3.11.13\bin>cqlsh

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.13 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.

WARNING: pyreadline dependency missing. Install to enable tab completion.
cqlsh> CREATE KEYSPACE group17 with replication = {'class':'SimpleStrategy', 'replication_factor':3};

Warnings :
Your replication factor 3 for keyspace group17 is higher than the number of nodes 1

cqlsh>
```

We created a keyspace with the name group17 and the replication is done shown as above

### Creating of table

```
C:\Windows\System32\cmd.exe - cqlsh
cqlsh:group17> CREATE TABLE visa_candidates(id int PRIMARY KEY, name text, gre_score int, toefl_score int, university_rating int, sop float, lor float, cgpa float , research int, no_of_backlogs int, relevant_study text, year_gaps int, parents_annual_income int, no_of_times_rejected text, us_relatives text, work_experience int, criminal_record int, chance_of_visa_approval float);
cqlsh:group17>
```

Now we created a table with name visa\_candidates and described its attributes are shown as above.

## Inserting values into the table

```
C:\Windows\System32\cmd.exe - cqlsh
cqlsh:group17> Copy visa_candidates(id,name,gre_score,toefl_score,university_rating,sop,lor,cgpa,research,no_of_backlogs,relevant_study,year_gaps,parents_annual_income,no_of_times_rejected,us_relatives,work_experience,criminal_record,chance_of_visa_approval) from 'C:\Users\saite\Desktop\visa_candidates.csv' with DELIMITER =',' AND Header=TRUE;
Using 3 child processes

Starting copy of group17.visa_candidates with columns [id, name, gre_score, toefl_score, university_rating, sop, lor, cgpa, research, no_of_backlogs, relevant_study, year_gaps, parents_annual_income, no_of_times_rejected, us_relatives, work_experience, criminal_record, chance_of_visa_approval].
Process ImportProcess-13:: 305 rows/s; Avg. rate: 305 rows/s
TTraceback (most recent call last):
Process ImportProcess-14:
Process ImportProcess-15:
T File "C:\Python27\lib\multiprocessing\process.py", line 267, in _bootstrap
Traceback (most recent call last):
raceback (most recent call last):
    File "C:\Python27\lib\multiprocessing\process.py", line 267, in _bootstrap
File "C:\Python27\lib\multiprocessing\process.py", line 267, in _bootstrap
    self.run()
    File "C:\apache-cassandra-3.11.13\bin..\pylib\cqlshlib\copyutil.py", line 2339, in run
    self.run()
    self.run()
    self.run()
File "C:\apache-cassandra-3.11.13\bin..\pylib\cqlshlib\copyutil.py", line 2339, in run
    File "C:\apache-cassandra-3.11.13\bin..\pylib\cqlshlib\copyutil.py", line 2339, in run
    self.close()
    self.close()
    self.close()
    self.close()
File "C:\apache-cassandra-3.11.13\bin..\pylib\cqlshlib\copyutil.py", line 2343, in close
File "C:\apache-cassandra-3.11.13\bin..\pylib\cqlshlib\copyutil.py", line 2343, in close
    File "C:\apache-cassandra-3.11.13\bin..\pylib\cqlshlib\copyutil.py", line 2343, in close
    self._session.cluster.shutdown()
    self._session.cluster.shutdown()
    self._session.cluster.shutdown()
    self._session.cluster.shutdown()
File "C:\apache-cassandra-3.11.13\bin..\lib\cassandra-driver-internal-only-3.11.0-bb96859b.zip\cassandra-driver-3.11.0-bb96859b\cassandra\cluster.py", line 1259
, in shutdown
File "C:\apache-cassandra-3.11.13\bin..\lib\cassandra-driver-internal-only-3.11.0-bb96859b.zip\cassandra-driver-3.11.0-bb96859b\cassandra\cluster.py", line 1259
, in shutdown
    File "C:\apache-cassandra-3.11.13\bin..\lib\cassandra-driver-internal-only-3.11.0-bb96859b.zip\cassandra-driver-3.11.0-bb96859b\cassandra\cluster.py", line 1
259, in shutdown
    self.control_connection.shutdown()
```

Now after creating the table we have loaded the data into the table using the local CSV file which is present in our system, using the appropriate command.

Checking whether the data is loaded or not

visa_candidates										no_of_backlogs	no_of_times_rejected	parents_annual_income	relevant_study	research	
id	cgpa	chance_of_visa_approval	criminal_record	gre_score	lor	name	sop	toefl_score	university_rating	us_relatives	work_experience	year_gaps			
23	9.5	0.94	0	328	5	Sollem Vamsi Krishna	0	0	3.5	0	0	1734089	YES	1	
5	116	5	NO	3	0	Aditya Kanala	3	0	3.5	0	0	1504514	YES	0	
114	8.56	0.72	0	320	3.5	Ramireddy Ranapratap	2	0	4	0	0	2924353	YES	1	
4	110	2	NO	4	1	V L Santosh Tejaswi Resapu	4	0	5	0	0	852916	YES	0	
53	8	0.78	0	334	3	Anumula Shashidar Reddy	3	0	1	0	0	539501	YES	1	
4	116	4	NO	5	0	Oruganti Chakradhar Prabal Kumar	1	0	3	0	0	1947922	YES	1	
110	8.64	0.68	0	304	4	Appidi Harshavardhan Reddy	0	0	4	0	0	1682313	YES	1	
5	103	5	NO	4	0	Neelagiri Narayana Nisha Chandran	2	0	3	0	0	2952788	YES	1	
91	7.92	0.64	0	318	4	Garudadri Venkata Sai Praneeth	2	0	1	0	0	2696840	YES	0	
4	106	2	NO	1	0	K Sri Sundar Subhash Mahadev	2	0	4	0	0	2433437	YES	0	
128	8.71	0.78	0	319	2	Kata Akhil	0	0	5	0	0	1973361	YES	1	
2.5	112	3	YES	4	0	Deepak Swaminathan	4	0	3	0	0	2725368	YES	0	
363	9.23	0.91	0	338	5	Bheemishetty Akhilesh Bala	0	0	1	0	0	751416	YES	1	
4.5	115	5	NO	1	0	Doppa Meenakshi	2	0	3	0	0	1369917	YES	1	
251	8.57	0.74	0	320	2.5	Margapuri Pradyumna Rao	0	0	2	0	0	1681960	YES	1	
3	104	3	NO	3	0	Ananya Prabhu	0	0	4	0	0	2253642	YES	0	
310	8.6	0.7	0	308	3	Ganta Chirudeep	0	0	1	0	0	2624476	YES	1	
3.5	110	4	YES	2	0	Sanjana Bhogavalli	2	0	5	0	0	1245120	YES	0	
247	8.73	0.72	0	316	3.5	Peesa Sai	0	0	3	0	0	1381996	YES	0	
3	105	3	NO	0	0	Ambati Haarika	0	0	5	0	0	2621364	YES	1	
214	9.78	0.96	0	333	4.5	Parnem Rahul Reddy	0	0	4	0	0	1239769	YES	1	
5	119	5	YES	1	0	Dhanush Kumar Suresh	2	3	3	0	0	666192	YES	0	
117	8.62	0.56	0	299	3.5	(12 rows)									
4	102	3	YES	4	0	cqlsh:group17>									
144	9.92	0.97	0	340	4										
4.5	120	4	NO	3	0										
120	8.84	0.71	0	327	3.5										
3	104	5	NO	5	0										
219	8.97	0.84	0	324	3.5										
3	110	4	NO	3	0										
140	9.12	0.78	0	318	3.5										
3.5	109	1	NO	0	0										
308	9	0.8	0	325	4										
4	112	4	NO	2	0										
55	8	0.7	0	322	3.5										
3	110	3	NO	3	0										
255	9.12	0.85	0	321	5										
4	114	4	YES	0	0										
331	8.66	0.8	0	327	3										
3.5	113	3	NO	5	0										
129	9.1	0.84	0	326	3										
3.5	112	3	NO	0	0										
324	8.18	0.62	0	305	2.5										
2	102	2	NO	5	0										

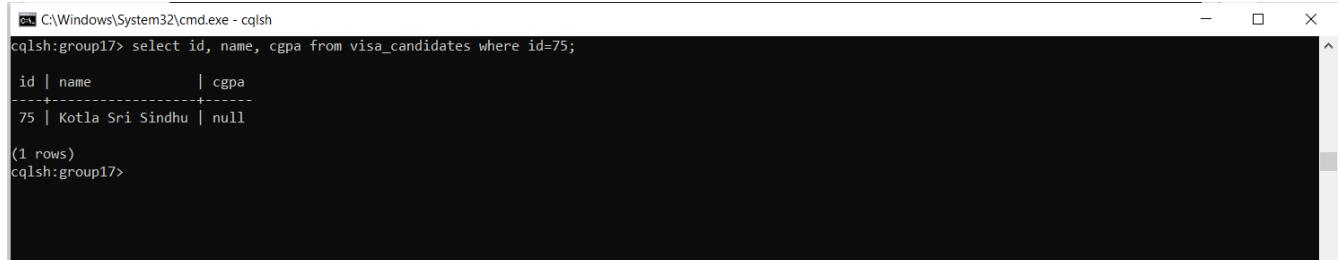
Using select \* we can see all the data in the table so that we can check whether all data is imported or not into the table.

Finding the highest approval chances based on some conditions

visa_candidates		chance_of_visa_approval
id	name	chance_of_visa_approval
214	Kata Akhil	0.96
373	M Himanshu	0.95
213	Kadiyala Jayasimha	0.95
204	Donkada Pratyusha	0.97
121	D G Anjan Kumar	0.94
203	Divya Pragya	0.97
149	Dantuluri Meghana	0.96
71	Rugved Jayavarapu	0.94
386	Veluru Aneesh Kumar Chowdary	0.96
26	Sree Keerthana Kanchi	0.94
24	Sravya Tangirala	0.95
362	Akhil Dadhwal	0.93
(12 rows)		
cqlsh:group17>		

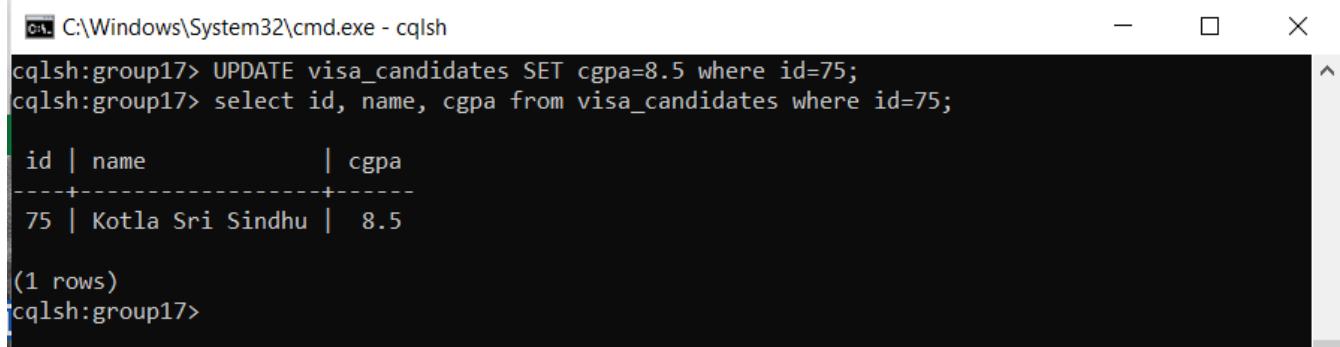
For this, we have taken some considerations such as gre\_score, toefl\_score, chance\_of\_visa\_approval, cgpa and their parents\_annual\_income.

## Null value insertion



```
c:\Windows\System32\cmd.exe - cqlsh
cqlsh:group17> select id, name, cgpa from visa_candidates where id=75;
id | name      | cgpa
---+---+-----+
75 | Kotla Sri Sindhu | null
(1 rows)
cqlsh:group17>
```

Here by checking the value at id =75 we found that the cgpa value is null.



```
c:\Windows\System32\cmd.exe - cqlsh
cqlsh:group17> UPDATE visa_candidates SET cgpa=8.5 where id=75;
cqlsh:group17> select id, name, cgpa from visa_candidates where id=75;
id | name      | cgpa
---+---+-----+
75 | Kotla Sri Sindhu | 8.5
(1 rows)
cqlsh:group17>
```

- Using the update and set command we have inserted the cgpa value for the id=75.
- Also, check whether the value is reflected or not by using the select command

## Data Pre-Processing:

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** jupyter Data\_preprocessing Last Checkpoint: an hour ago (autosaved), Logout
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- Cell 1 [13]:**

```
#import required library packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```
- Cell 2 [14]:**

```
#create dataframe with dataset
visa_candidates = pd.read_csv(r'C:\project\visa_candidates.csv')
```
- Cell 3 [15]:**

```
#print the first few rows of dataframe
visa_candidates.head()
```
- Output [15]:** A data frame with 6 rows and 14 columns. The columns are: id, name, gre\_score, toefl\_score, university\_rating, sop, lor, cgpa, research, no\_of\_backlogs, relevant\_study, year\_gaps, parents\_annual\_income, and no\_of\_1.

	id	name	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	relevant_study	year_gaps	parents_annual_income	no_of_1
0	1	Alluri Udgith Reddy	337	118	4	4.5	4.5	9.65	1	0	YES	0	1690680.0	
1	2	Annen Naga Sujitha	324	107	4	4.0	4.5	8.87	1	0	YES	0	1207915.0	
2	3	Borra Kiran Kumar	316	104	3	3.0	3.5	8.00	1	0	YES	0	1761149.0	
3	4	Dadisetti V. S. Rajkumar	322	110	3	3.5	2.5	8.67	1	0	YES	0	1983861.0	
4	5	Dasari Sai Rishitha	314	103	2	2.0	3.0	8.21	0	3	YES	0	588408.0	

Here we are trying to use libraries like pandas, NumPy, and matplot lib by importing them first. Then we

create a data frame by using the `read_csv(file path)` method. Next, we check the data frame by viewing the head and tail of the data frame.

`Dataframe.info()` gives details of the dataset like the columns and number of entries and their datatypes as shown above.



File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel) ○

[+]

Run

Code

Out[16]: visa_candidates.tail()															
	id	name	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	relevant_study	year_gaps	parents_annual_income	in	
404	127	Namrata Sira Rao	323	113		3	4.0	3.0	9.32	1	0	YES	0	2792053.0	
405	128	Oruganti Chakradhar Prabal Kumar		319	112		3	2.5	2.0	8.71	1	1	YES	0	1947922.0
406	129	Parmen Rahul Reddy	326	112		3	3.5	3.0	9.10	1	0	YES	0	1239769.0	
407	130	Peddi Sai Pankaj	333	118		5	5.0	5.0	9.35	1	0	YES	0	2563093.0	
408	131	Potluri Sai Cherish	339	114		5	4.0	4.5	9.76	1	0	YES	0	1352725.0	

In [17]: #print the basic information of the dataframe  
visa\_candidates.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 409 entries, 0 to 408
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   id               409 non-null    int64  
 1   name              409 non-null    object 
 2   gre_score         409 non-null    int64  
 3   toefl_score       409 non-null    object 
 4   university_rating 409 non-null    int64  
 5   sop               409 non-null    float64 
 6   lor               409 non-null    float64 
 7   cgpa              407 non-null    float64 
 8   research          409 non-null    int64  
 9   no_of_backlogs    409 non-null    int64  
 10  relevant_study    406 non-null    object 
 11  year_gaps         409 non-null    int64  
 12  parents annual income 407 non-null    float64
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter Data\_preprocessing Last Checkpoint: an hour ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Not Trusted, Python 3 (ipykernel), Logout
- In [18]:** `#output basic stats of the dataframe  
visa_candidates.describe()`
- Out[18]:** A table showing basic statistics for each column of the `visa_candidates` DataFrame. The columns are: id, gre\_score, university\_rating, sop, lor, cgpa, research, no\_of\_backlogs, year\_gaps, parents\_annual\_income, and no\_of\_tir.

	id	gre_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	year_gaps	parents_annual_income	no_of_tir
count	409.000000	409.000000	409.000000	409.000000	409.000000	407.000000	409.000000	409.000000	409.000000	4.070000e+02	
mean	199.454768	317.031785	3.110024	3.409535	3.455990	8.610737	0.552567	1.838831	0.141809	1.888082e+06	
std	115.137209	11.514546	1.146231	1.008120	0.903028	0.601022	0.497838	2.005721	0.442184	7.745468e+05	
min	1.000000	290.000000	1.000000	1.000000	1.000000	6.800000	0.000000	0.000000	0.000000	4.000340e+05	
25%	101.000000	309.000000	2.000000	2.500000	3.000000	8.175000	0.000000	0.000000	0.000000	9.683195e+05	
50%	198.000000	317.000000	3.000000	3.500000	3.600000	8.640000	1.000000	2.000000	0.000000	1.709228e+06	
75%	299.000000	325.000000	4.000000	4.000000	4.000000	9.090000	1.000000	3.000000	0.000000	2.376814e+06	
max	400.000000	340.000000	5.000000	5.000000	5.000000	9.920000	1.000000	9.000000	2.000000	2.975122e+06	

- In [19]:** `#check for null values in dataframe  
visa_candidates.isna().sum()`
- Out[19]:** A Series showing the count of null values for each column. The columns are: id, name, gre\_score, toefl\_score, university\_rating, sop, lor, cgpa, research, no\_of\_backlogs, relevant\_study, year\_gaps, parents\_annual\_income, no\_of\_times\_rejected, us\_relatives, work\_experience, criminal\_records, and chance\_of\_visa\_approval. The dtype is int64.

Dataframe.describe() provides the basic statistics of the dataset like count, range, maximum, minimum, average/mean, median, and standard deviation as shown.

Next, we check if there are null values in our data using ISNA().sum().



File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3 (ipykernel)

```
In [20]: #remove the rows containing null values  
visa_candidates.dropna(inplace=True)
```

```
In [21]: #check for null values again  
visa_candidates.isna().sum()
```

```
Out[21]: id          0  
name         0  
gre_score    0  
toefl_score  0  
university_rating 0  
sop          0  
lor          0  
cgpa         0  
research     0  
no_of_backlogs 0  
relevant_study 0  
year_gaps    0  
parents_annual_income 0  
no_of_times_rejected 0  
us_relatives 0  
work_experience 0  
criminal_records 0  
chance_of_visa_approval 0  
dtype: int64
```

Here we try to drop these null values using drop() and check for null values again using ISNA().sum().

jupyter Data\_preprocessing Last Checkpoint: an hour ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

In [22]: `#check for duplicated rows  
visa_candidates[visa_candidates.duplicated('name')]`

Out[22]:

		id	name	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	relevant_study	year_gaps	parents_annual_income	no_of_rejected
400	84	Jaswanth Talasila	322	115		5	4.0	4.5	9.36	1	0	YES	0	1709228.0	0
401	327	Vedula Sri Bala Sai Deepthi	299	100		3	2.0	2.0	8.02	0	3	NO	0	1644473.0	0
402	254	P Anusha	335	115		4	4.5	4.5	9.68	1	0	YES	0	2663024.0	0
404	127	Namrata Sira Rao	323	113		3	4.0	3.0	9.32	1	0	YES	0	2792053.0	0
405	128	Oruganti Chakradhar Prabab Kumar	319	112		3	2.5	2.0	8.71	1	1	YES	0	1947922.0	0
406	129	Parmen Rahul Reddy	326	112		3	3.5	3.0	9.10	1	0	YES	0	1239769.0	0
407	130	Peddi Sai Pankaj	333	118		5	5.0	5.0	9.35	1	0	YES	0	2563093.0	0
408	131	Potluri Sai Cherish	339	114		5	4.0	4.5	9.76	1	0	YES	0	1352725.0	0

In [23]: `#drop the duplicate rows  
visa_candidates.drop_duplicates(keep='first',inplace=True)`

In [24]: `#check for the duplicate rows again  
visa_candidates[visa_candidates.duplicated('name')]`

Out[24]:

		id	name	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	relevant_study	year_gaps	parents_annual_income	no_of_rejected
--	--	----	------	-----------	-------------	-------------------	-----	-----	------	----------	----------------	----------------	-----------	-----------------------	----------------

Next, we check for duplicate rows and then remove these duplicates using drop\_duplicates() and check again for any duplicates.

In [25]: `#find correlation between the columns  
visa_candidates.corr()`

Out[25]:

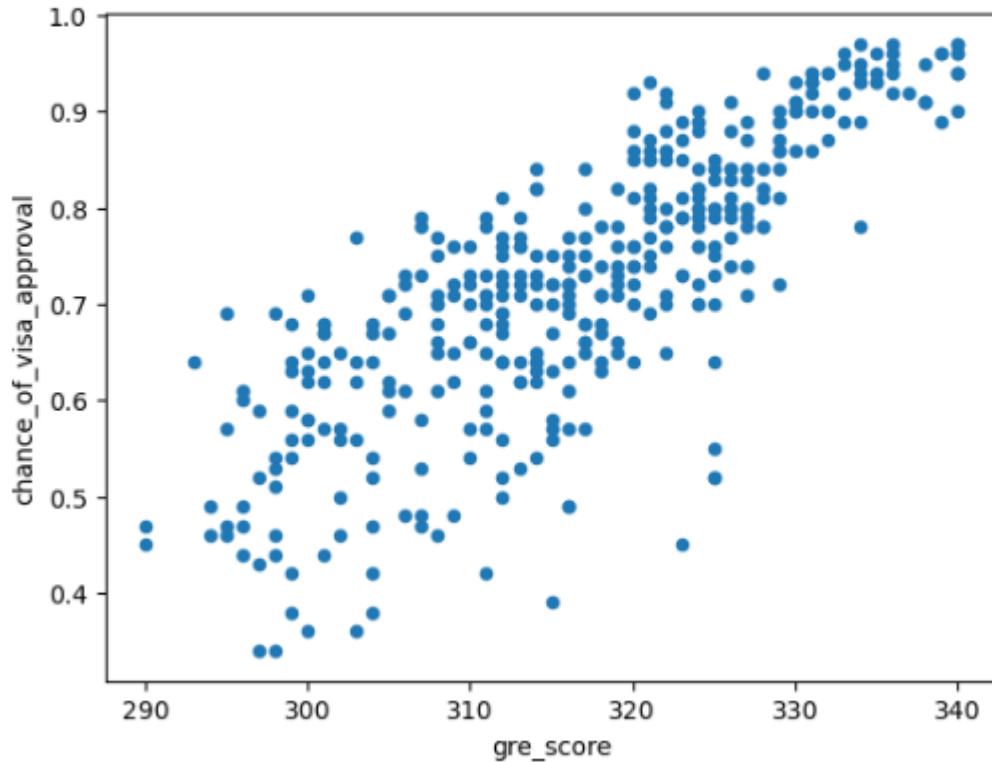
	id	gre_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	year_gaps	parents_annual_income	no_of_rejected
id	1.000000	-0.098286	-0.168640	-0.169451	-0.094936	-0.046355	-0.053978	0.028724	-0.294708	0.002783	0.002783
gre_score	-0.098286	1.000000	0.668404	0.613669	0.564859	0.832868	0.586148	-0.628810	-0.303685	-0.000362	-0.000362
university_rating	-0.168640	0.668404	1.000000	0.737968	0.667689	0.741798	0.438488	-0.548995	-0.280961	0.058966	0.058966
sop	-0.169451	0.613669	0.737968	1.000000	0.735642	0.718538	0.441445	-0.513921	-0.191298	0.026969	0.026969
lor	-0.094936	0.564859	0.667689	0.735642	1.000000	0.670178	0.412030	-0.475076	-0.248319	-0.012421	-0.012421
cgpa	-0.046355	0.832868	0.741798	0.718538	0.670178	1.000000	0.532531	-0.673097	-0.384862	0.039072	0.039072
research	-0.053978	0.586148	0.438488	0.441445	0.412030	0.532531	1.000000	-0.508188	-0.148423	-0.059527	-0.059527
no_of_backlogs	0.028724	-0.628810	-0.546995	-0.513921	-0.475076	-0.673097	-0.508188	1.000000	0.515076	-0.006656	-0.006656
year_gaps	-0.294708	-0.303685	-0.260961	-0.191298	-0.248319	-0.384862	-0.148423	0.515076	1.000000	-0.009062	-0.009062
parents_annual_income	0.002783	-0.000362	0.058966	0.026969	-0.012421	0.039072	-0.059527	-0.008656	-0.009062	1.000000	1.000000
no_of_times_rejected	0.303685	-0.336154	-0.368355	-0.456494	-0.388800	-0.393622	-0.290834	0.318997	0.070598	0.080230	0.080230
work_experience	0.113001	0.049793	0.007088	0.030321	-0.063600	0.052939	0.089619	-0.047744	-0.084975	0.003466	0.003466
criminal_records	-0.031114	-0.208823	-0.132884	-0.120250	-0.182403	-0.250473	-0.127275	0.418635	0.407290	0.080757	0.080757
ance_of_visa_approval	0.041048	0.802802	0.705293	0.670841	0.670068	0.876633	0.558113	-0.774150	-0.550647	0.015606	0.015606

Here we try to find out the correlation between the columns in the dataset, more specifically we are trying to get the relation between each column and visa\_chances prediction.

The next few figures are visual representations of these relations between a few important columns and visa\_approval chances like gre\_score, cgpa,no\_of\_backlogs, etc using a plot. scatter(x,y).

```
In [26]: #scatter plot of gre_score vs chance_of_visa-approval  
visa_candidates.plot.scatter(x="gre_score",y="chance_of_visa_approval")
```

```
Out[26]: <AxesSubplot:xlabel='gre_score', ylabel='chance_of_visa_approval'>
```

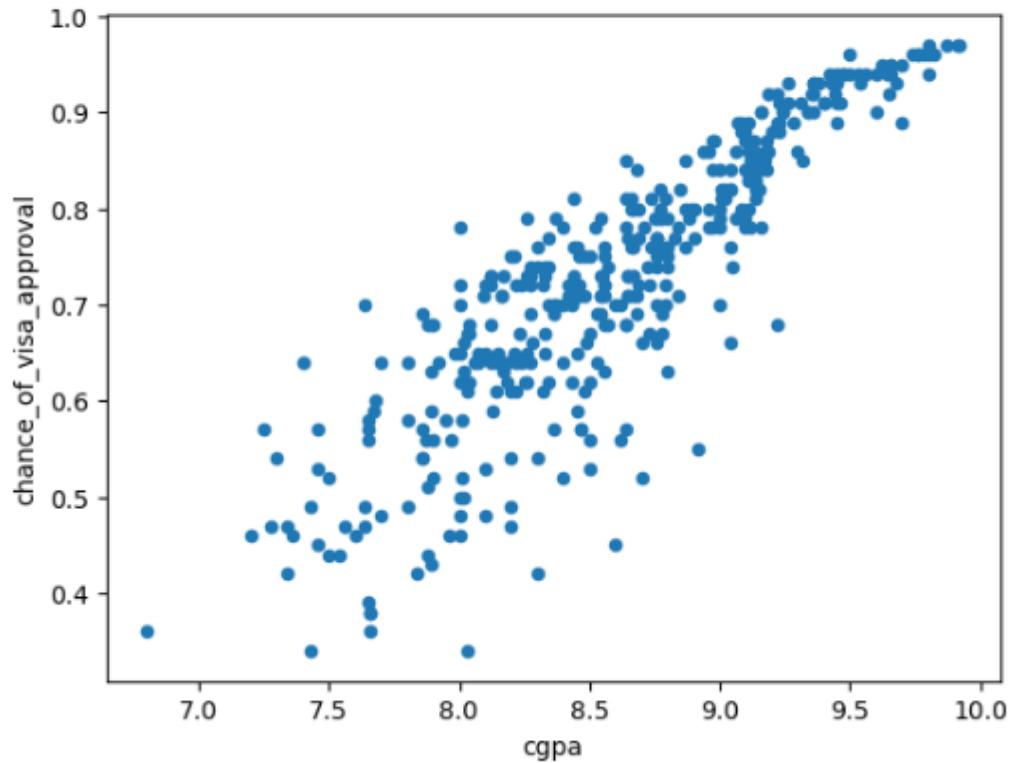


- The students test scores and chance of visa approval are directly proportional to each other. If the students have good test scores then the probability of getting visa is increased greatly.

\*

```
In [27]: #scatter plot of cgpa vs chance_of_visa-approval  
visa_candidates.plot.scatter(x="cgpa",y="chance_of_visa_approval")
```

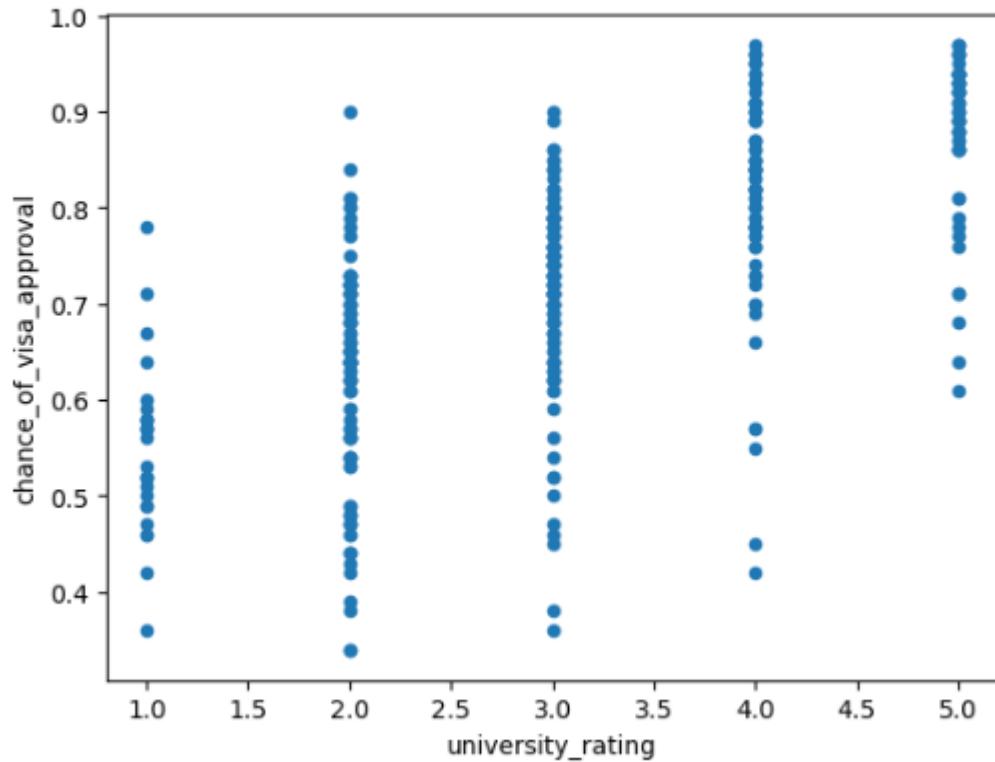
```
Out[27]: <AxesSubplot:xlabel='cgpa', ylabel='chance_of_visa_approval'>
```



- The higher CGPA will greatly improve the chance of visa approval.

```
In [28]: #scatter plot of university_rating vs chance_of_visa-approval  
visa_candidates.plot.scatter(x="university_rating",y="chance_of_visa_approval")
```

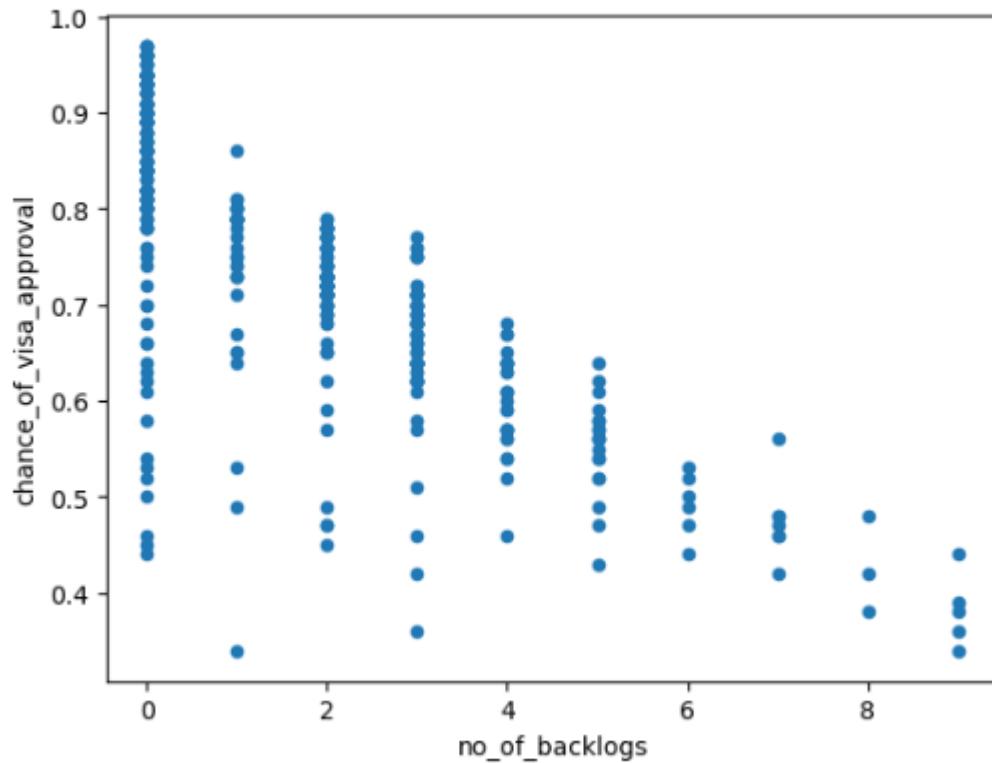
```
Out[28]: <AxesSubplot:xlabel='university_rating', ylabel='chance_of_visa_approval'>
```



- Similar to that of test scores and cgpa. The relation between university\_rating and chance of visa approval is directly proportional.

```
In [30]: #scatter plot of no_of_backlogs vs chance_of_visa-approval  
visa_candidates.plot.scatter(x="no_of_backlogs",y="chance_of_visa_approval")
```

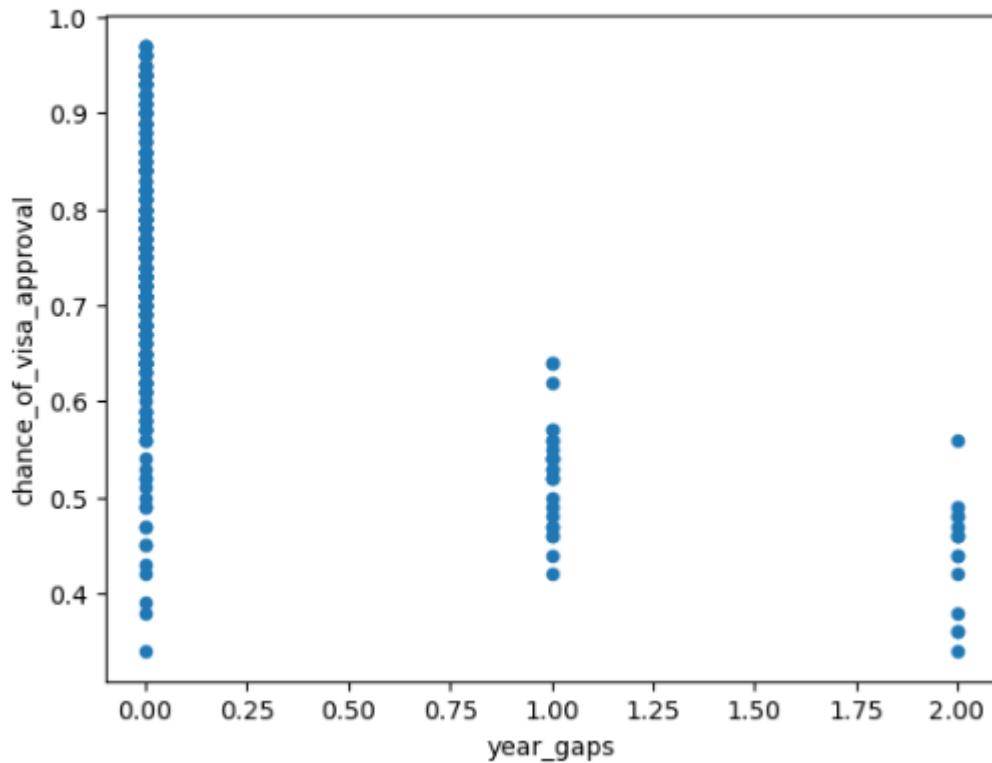
```
Out[30]: <AxesSubplot:xlabel='no_of_backlogs', ylabel='chance_of_visa_approval'>
```



- The no of backlogs is inversely proportional to the chance of visa approval. If the no of backlogs are high then the visa approval chances are greatly reduced.

```
In [31]: #scatter plot of year_gaps vs chance_of_visa-approval  
visa_candidates.plot.scatter(x="year_gaps",y="chance_of_visa_approval")
```

```
Out[31]: <AxesSubplot:xlabel='year_gaps', ylabel='chance_of_visa_approval'>
```



```
In [32]: #extract the yes/no values as 1/0 values respectively  
mymap={'NO':0,'YES':1}  
visa_candidates=visa_candidates.applymap(lambda s:mymap.get(s) if s in mymap else s)
```

```
In [33]: #check if yes/no values are mapped to 1/0  
visa_candidates.head()
```

```
Out[33]:
```

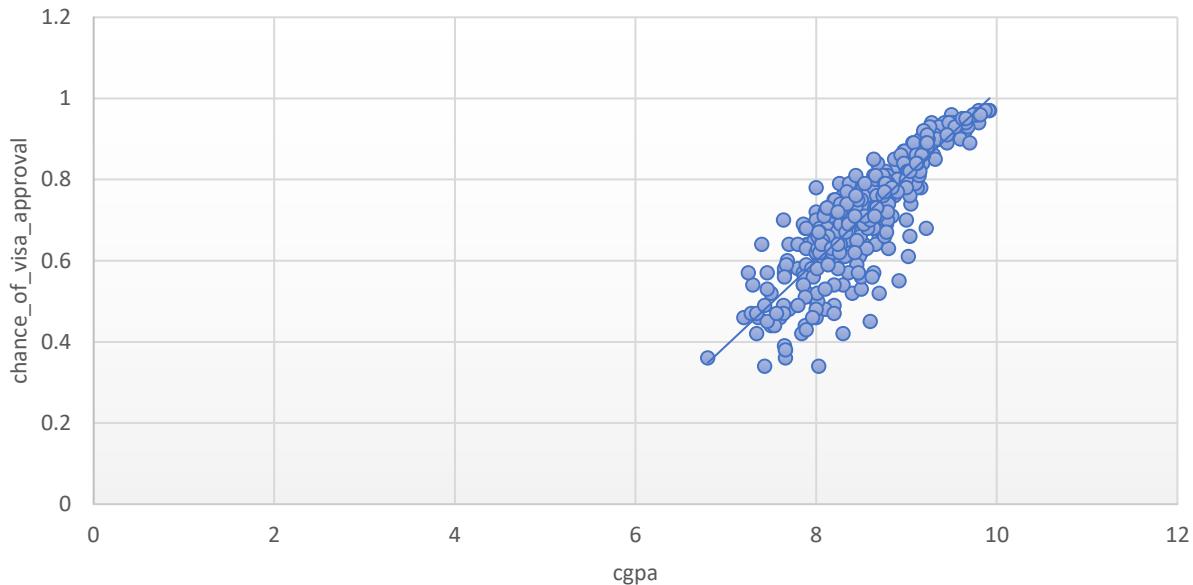
	id	name	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	no_of_backlogs	relevant_study	year_gaps	parents_annual_income	no_of_
0	1	Alluri Udgith Reddy	337	118	4	4.5	4.5	9.65	1	0	1	0	1690680.0	
1	2	Annen Naga Sujitha	324	107	4	4.0	4.5	8.87	1	0	1	0	1207915.0	
2	3	Borra Kiran Kumar	316	104	3	3.0	3.5	8.00	1	0	1	0	1761149.0	
3	4	Dadisetti V. S. Rajkumar	322	110	3	3.5	2.5	8.67	1	0	1	0	1983861.0	
4	5	Dasari Sai Rishitha	314	103	2	2.0	3.0	8.21	0	3	1	0	588408.0	

```
In [ ]:
```

As a final step in data preprocessing, we convert any string-indicated values to integers like yes/no values here and are mapped to 1/0 values for further processing.

## Analysis and Preliminary Results:

Field: cgpa and Field: chance\_of\_visa\_approval appear highly correlated.

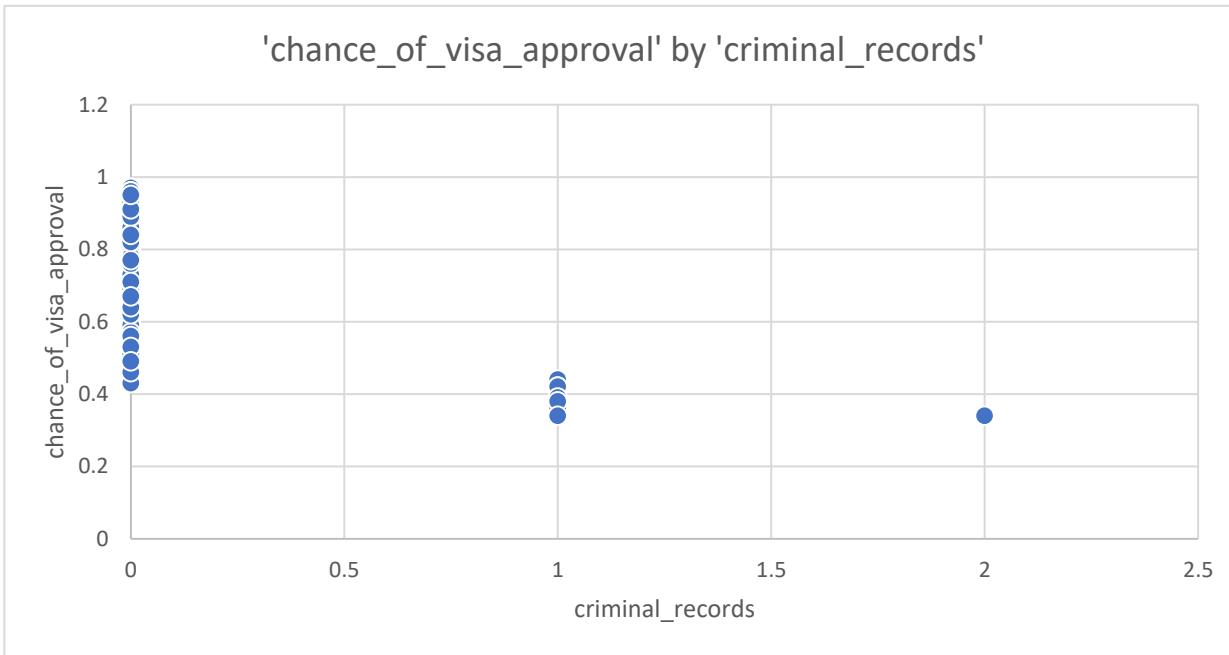


- The above plot graph depicts the correlation between CGPA and the Chance of visa approval

'parents\_annual\_income' and 'chance\_of\_visa\_approval' appear to form a cluster with 4 outliers.



- The parent's annual income contributes only a portion to the criteria but high annual income can bolster the student's profile and can help in improving approval chances



- The chance of visa approval and criminal records are inversely proportional to each other. Any prior criminal records in the student's profile will tremendously affect the visa approval rate.

## **Machine learning:**

- To implement the machine learning part, First, we split the data into two different sets in one set we used 75% of the data for training the machine learning models and the remaining 25% for testing purposes. In the second set, we used 80% for training purposes and the remaining 20% for testing purposes.
- For each dataset we have used five machine-learning algorithms namely Multiple linear regression, KNN regression, Decision tree regression, Random Forest regression, and SVM regression. We fit models using these algorithms and the data we processed earlier and find the accuracy of the predicted output.

## **MACHINE LEARNING**

```
In [33]: from sklearn.model_selection import train_test_split
```

### **80 - 20 split of Dataset**

```
In [34]: X= visa_candidates.loc[:, "gre_score": "criminal_records"]
y = visa_candidates['chance_of_visa_approval']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

### **75 - 25 split of DataSet**

```
In [35]: X2= visa_candidates.loc[:, "gre_score": "criminal_records"]
y2 = visa_candidates['chance_of_visa_approval']

X2_train,X2_test,y2_train,y2_test = train_test_split(X2,y2,test_size=0.25,random_state=1)
```

- The above screenshot represents the splitting of pre-processed data into two different sets. One set uses 80% of data for training and 20% for testing. Another dataset uses 75% for training and 25% for testing purposes.

## Multiple Linear Regression:

```
In [78]: from sklearn import linear_model  
from sklearn.metrics import accuracy_score
```

with 80-20 split data

```
In [79]: regr = linear_model.LinearRegression()  
regr.fit(X_train,y_train)  
  
Approval_chances = regr.predict(X_test)  
MLR_Score = regr.score(X_test,y_test)  
print('Accuracy/R^2 score with multiple Linear regression is = ', regr.score(X_test,y_test))  
  
Accuracy/R^2 score with multiple Linear regression is =  0.8720378702746149
```

with 75-25 split data

```
In [80]: regr2 = linear_model.LinearRegression()  
regr2.fit(X2_train,y2_train)  
mlr_score2 = regr2.score(X2_test,y2_test)  
#Approval_chances = regr.predict(X_test)  
print('Accuracy/R^2 score with multiple Linear regression is = ', regr2.score(X2_test,y2_test))  
  
Accuracy/R^2 score with multiple Linear regression is =  0.8771625615748793
```

- We have used multiple linear algorithms for regression to fit in our models and executed the model on both datasets and the accuracy on these datasets is depicted in the above screenshot.
- Multiple linear regression is a statical technique commonly known as multiple regression. It is a statistical method for predicting the result of a response variable by using two or more explanatory factors. Unlike the linear regression which uses the single explanatory variable, Multiple linear regression which is extension to linear regression uses two or more explanatory variables.
- Formula to calculate multiple linear regression

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} + \epsilon$$

$y_i$  = dependent variable

$x_i$  = explanatory variables

$\beta_0$  = y-intercept

$\beta_p$  = slope coefficients

$\epsilon$  = the model's error term

## K- Nearest Neighbor:

### 3.K- NEAREST NEIGHBOUR

```
In [84]: from sklearn.neighbors import KNeighborsRegressor
```

with 80-20 split data

```
In [85]: for i in range(1,10,2):
    knn=KNeighborsRegressor(n_neighbors=i)
    KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
        metric_params=None, n_jobs=1, n_neighbors=8, p=2,
        weights='uniform')
    knn.fit(X_train,y_train)
    knn_score = knn.score(X_test,y_test)
    #print ('Train','K=',i,'|t->\t',knn.score(X_train,y_train))
    print ('Test','K=',i,'|t->\t',knn.score(X_test,y_test))

Test K= 1      ->      0.7605047164032831
Test K= 3      ->      0.8346536540215335
Test K= 5      ->      0.8229174323165503
Test K= 7      ->      0.8328242542882643
Test K= 9      ->      0.81233543253736
```

with 75-25 split data

```
In [86]: for i in range(1,10,2):
    knn2=KNeighborsRegressor(n_neighbors=i)
    KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
        metric_params=None, n_jobs=1, n_neighbors=8, p=2,
        weights='uniform')
    knn2.fit(X2_train,y2_train)
    knn_score2 = knn2.score(X2_test,y2_test)
    #print ('Train','K=',i,'|t->\t',knn2.score(X2_train,y2_train))
    print ('Test','K=',i,'|t->\t',knn2.score(X2_test,y2_test))

Test K= 1      ->      0.7298333247317614
Test K= 3      ->      0.8192510137773986
Test K= 5      ->      0.8197729069041241
Test K= 7      ->      0.8272450910359719
Test K= 9      ->      0.8154603085565323
```

- Now we have used the K- nearest neighbor algorithm on both datasets and the accuracy of the predicted output for different K values is depicted in the above screenshot.
- KNN is a machine learning algorithm which can be used for both regression and classification. By calculating the distance between all training points and test data the KNN predicts suitable class for the test data. The KNN algorithm determines the possibility that each of the "K" training data classes will contain the test data, and then it selects the class with the highest likelihood. In a regression scenario, the value is the mean of the 'K' selected training points. KNN is a non-parametric technique. Which estimates the relation between independent variables by considering the mean of the data that belongs similar neighborhood and the continuous result.

## Decision Tree Regression:

```
In [90]: from sklearn.tree import DecisionTreeRegressor
```

with 80-20 split data

```
In [91]: #DecisionTreeRegressor class has many parameters. Input only #random_state=0 or 42.  
dec_tree_model = DecisionTreeRegressor(random_state=42)  
#Fit the regressor object to the dataset.  
dec_tree_model.fit(X_train,y_train)  
DT_score = dec_tree_model.score(X_test,y_test)
```

```
In [92]: print('Accuracy/R^2 score with multiple Linear regression is = ', dec_tree_model.score(X_test,y_test))  
Accuracy/R^2 score with multiple Linear regression is =  0.7915135366899424
```

with 75-25 split data

```
In [93]: #DecisionTreeRegressor class has many parameters. Input only #random_state=0 or 42.  
dec_tree_model2 = DecisionTreeRegressor(random_state=42)  
#Fit the regressor object to the dataset.  
dec_tree_model2.fit(X2_train,y2_train)  
dt_score2 = dec_tree_model2.score(X2_test,y2_test)  
print('Accuracy/R^2 score with multiple Linear regression is = ', dec_tree_model2.score(X2_test,y2_test))
```

```
Accuracy/R^2 score with multiple Linear regression is =  0.7573196960726383
```

- Here we have used the Decision tree algorithm to find the accuracy of the predicted output. We have executed the Decision tree algorithm on 2 datasets and the outcomes of this algorithm are shown in the above picture.
- In order to forecast data in the future and provide useful continuous output, decision tree regression trains a model using the characteristics of an object's features. Continuous output denotes that the output or result isn't discrete, i.e., it's not exactly a discrete, well-known collection of values and numbers.

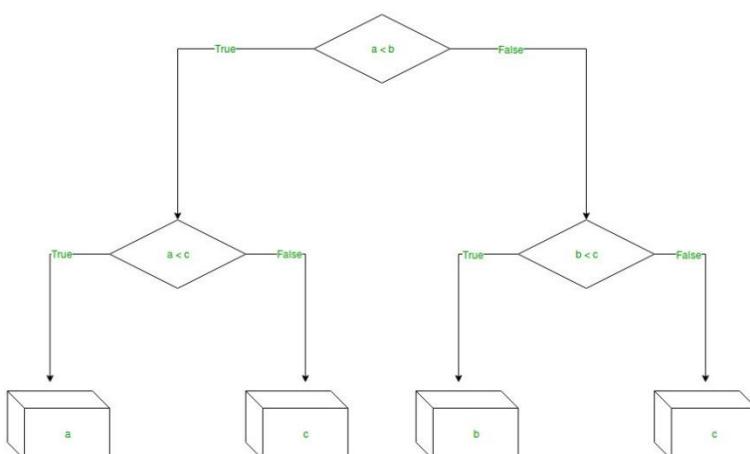


Fig 3: Decision Tree

## Random Forest Regression:

```
# import the regressor
from sklearn.ensemble import RandomForestRegressor

In [98]: # create regressor model
rand_forest_model = RandomForestRegressor(n_estimators = 100, random_state = 42)

# fit the regressor with x and y data
rand_forest_model.fit(X_train,y_train)
rf_score = rand_forest_model.score(X_test,y_test)

In [99]: print('Accuracy/R^2 score with multiple Linear regression is = ', rand_forest_model.score(X_test,y_test))

Accuracy/R^2 score with multiple Linear regression is =  0.8672064651476172
```

with 75-25 split data

```
In [100]: # create regressor model
rand_forest_model2 = RandomForestRegressor(n_estimators = 100, random_state = 42)

# fit the regressor with x and y data
rand_forest_model2.fit(X2_train,y2_train)
rf_score2 = rand_forest_model2.score(X2_test,y2_test)
print('Accuracy/R^2 score with multiple Linear regression is = ', rand_forest_model2.score(X2_test,y2_test))

Accuracy/R^2 score with multiple Linear regression is =  0.8860247458525444
```

- Here we have created a regression model using the random forest algorithm and fitted the model with a random forest regressor to evaluate the accuracy of the predicted output. The accuracy values are shown in the above screenshot.
- A supervised learning technique called Random Forest Regression leverages the ensemble learning approach for regression. The ensemble learning approach combines predictions from many machine learning algorithms to provide more accurate predictions than those from a single model.

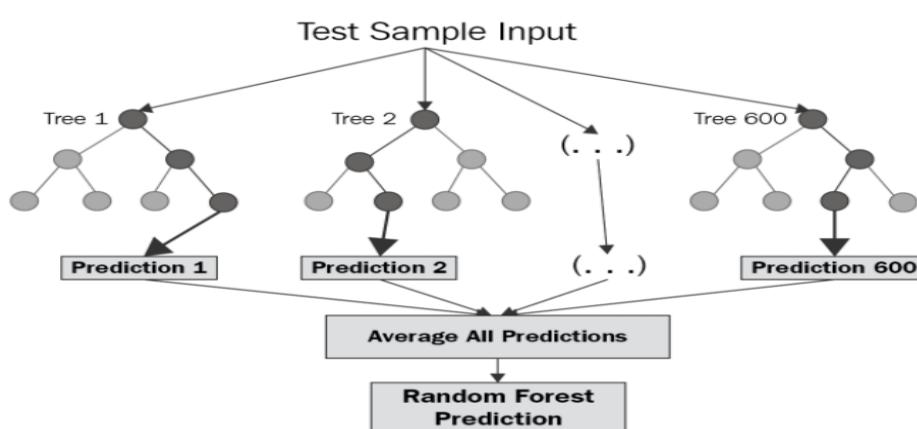


Fig 4: Random Forest

## SVM Regression:

```
In [72]: # Create support vector regressor here
from sklearn.svm import SVR
```

with 80-20 split data

```
In [73]: # most important SVR parameter is Kernel type. It can be #linear,polynomial or gaussian SVR. We have a non-linear condition #so we will use rbf
SVM_model = SVR(kernel='rbf')
SVM_model.fit(X_train,y_train)
svm_score = SVM_model.score(X_test,y_test)
print('Accuracy/R^2 score with multiple Linear regression is = ', SVM_model.score(X_test,y_test))
```

Accuracy/R^2 score with multiple Linear regression is = 0.6150084393014983

with 75-25 split data

```
In [74]: # most important SVR parameter is Kernel type. It can be #linear,polynomial or gaussian SVR. We have a non-linear condition #so we will use rbf
SVM_model2 = SVR(kernel='rbf')
SVM_model2.fit(X2_train,y2_train)
svm_score2 = SVM_model2.score(X2_test,y2_test)
print('Accuracy/R^2 score with multiple Linear regression is = ', SVM_model2.score(X2_test,y2_test))
```

Accuracy/R^2 score with multiple Linear regression is = 0.628076415178771

- Here we have created a regressor using the Support Vector Machine algorithm and fitted the model with this regressor. We have executed this model on both datasets and the accuracy of the model is displayed in the above snippet.
- The Support vector regression is a common ML algorithm widely used in regression and classification problems. Since SVM regression is highly dependent on kernels for processing, this algorithm is considered as a non-parametric algorithm. SVM algorithm works by mapping the data in datasets to a high-dimensional feature space so that data points can be easily categorized even when the data is not linearly separable.

## **Results:**

### **Multiple Linear Regression:**

```
MLR performance
```

In [81]:

```
# Predictions
preds_MLR = regr2.predict(X2_test)

# Performance
performance_MLR = pd.DataFrame({ 'True Value': y2_test,
                                    'Prediction': preds_MLR,
                                    'Error': y2_test - preds_MLR})

# View
performance_MLR
```

Out[81]:

	True Value	Prediction	Error
180	0.71	0.676357	0.033643
369	0.67	0.621006	0.048994
258	0.77	0.782186	-0.012186
282	0.81	0.777702	0.032298
291	0.56	0.552477	0.007523
...	...	...	...
111	0.69	0.770350	-0.080350
340	0.75	0.742502	0.007498
178	0.72	0.683257	0.036743
86	0.72	0.712230	0.007770
376	0.34	0.453798	-0.113798

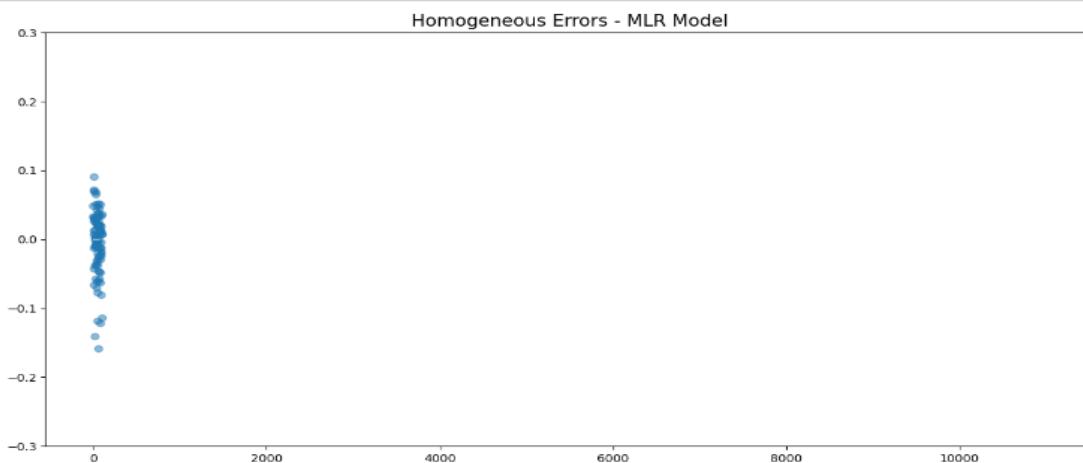
100 rows × 3 columns

- In the above screenshot, we are calculating the performance of Multiple linear regression and displaying the true value that is stored in y2\_test, Displaying the predicted values that we calculated using the model and the error that is obtained by subtracting the predicted from the true value
- In the error column, we could observe some are positive errors and some are negative errors. If the predicted value is greater than the actual value, then the resulting error is a negative error.
- If the predicted value is less than the actual value, then the resulting error is a positive error

### MLR Error plot

```
In [82]: plt.figure(figsize=(15,7))
# Errors
ax_x= performance_MLR['True Value']
ax_y= performance_MLR['Prediction']
yerr= performance_MLR['Error']

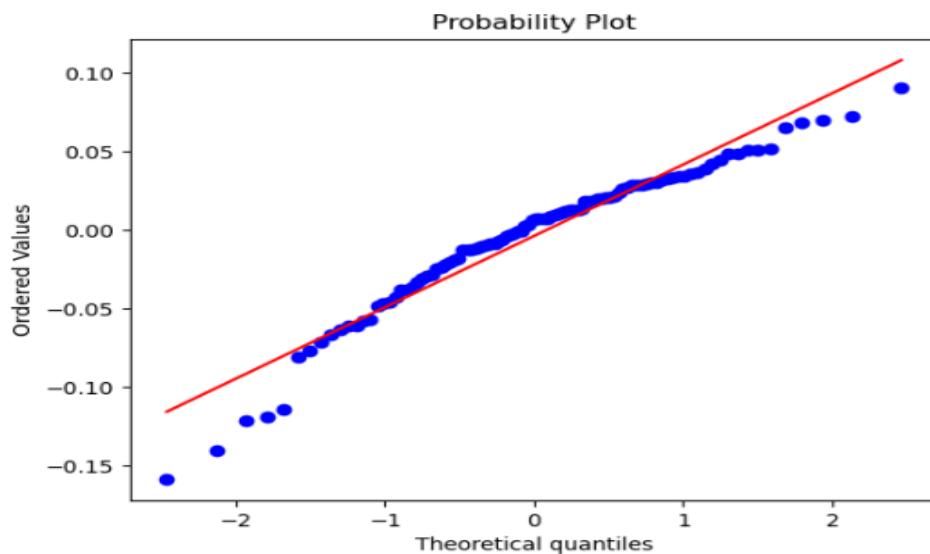
plt.scatter(range(len(yerr)), yerr, alpha=.5)
plt.title('Homogeneous Errors - MLR Model', size=15)
plt.hlines(y=0, xmin=0, xmax=11000, linestyle='--', color='white', alpha=.5)
plt.ylim(-.3, .3)
plt.show()
```



- The above snippet represents a scatter plot graph plotted by error value we obtained by subtracting the predicted value from the actual value.
- Most of the values are close to zero.

### MLR Probability Plot

```
In [83]: #QQ Plot
probplot(yerr, dist='norm', plot=plt);
plt.show()
```



- This probability graph displayed in the above snippet shows how close are the predicted values to the actual values. The red solid line represents the actual values, while blue dots represent the predicted values, and we could see using the MLR algorithm the predicted values are close to the actual values.
- The theoretical values are on the x axis and the ordered values are on y axis.



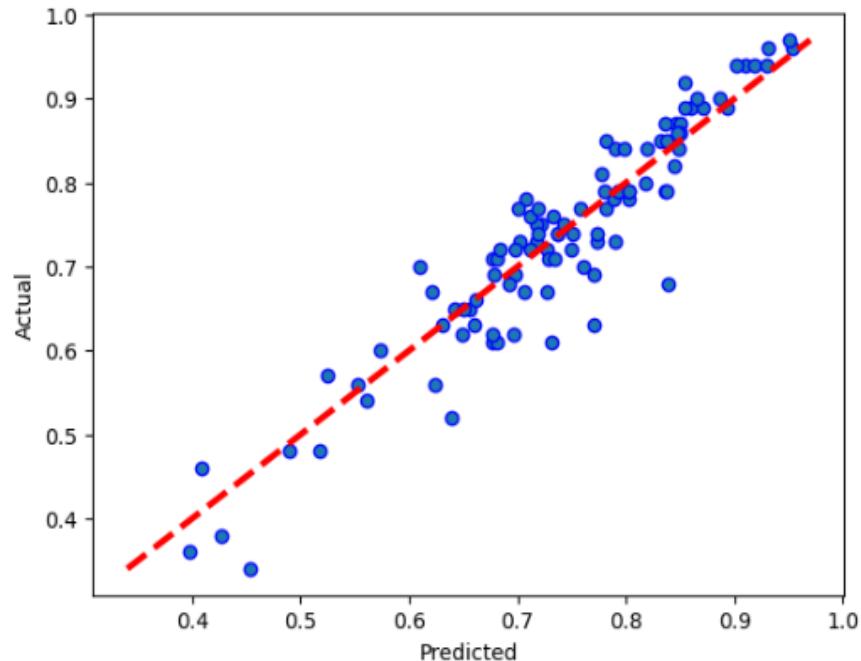
```
In [295]: # model evaluation for testing set
mae = metrics.mean_absolute_error(y2_test, preds_MLR)
mse = metrics.mean_squared_error(y2_test, preds_MLR)
r2 = metrics.r2_score(y2_test, preds_MLR)

print("The model performance for testing set")
print("-----")
print('MAE is {}'.format(mae))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))

The model performance for testing set
-----
MAE is 0.03454629836715684
MSE is 0.0021183205702717477
R2 score is 0.8771625615748793
```

- Here we are evaluating the model based on mean absolute error (MAE) and mean squared error (MSE) and also based on the R2 score.
- We have used metrics package for finding the values of MAE, MSE and R2.
- Based upon the R2 score we will evaluate the model percentage.

```
In [294]: fig, ax = plt.subplots()
ax.scatter(preds_MLR, y2_test, edgecolors=(0, 0, 1))
ax.plot([y2_test.min(), y2_test.max()], [y2_test.min(), y2_test.max()], 'r--', lw=3)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
```



- Here we created a graph that is plotted based on predicted and actual values.
- On the x axis predicted values are there and on y axis actual values are present.
- Moreover the graph values are in a straight line.

## KNN Regression:

### KNN Performance

```
In [87]: # Predictions
preds_knn = knn.predict(X_test)

# Performance
performance_knn = pd.DataFrame({ 'True Value': y_test,
                                  'Prediction': preds_knn,
                                  'Error': y_test - preds_knn})

# View
performance_knn
```

Out[87]:

	True Value	Prediction	Error
180	0.71	0.675556	0.034444
369	0.67	0.625556	0.044444
258	0.77	0.716667	0.053333
282	0.81	0.725556	0.084444
291	0.56	0.558889	0.001111
...	...	...	...
152	0.86	0.867778	-0.007778
336	0.72	0.710000	0.010000
231	0.74	0.697778	0.042222
165	0.78	0.791111	-0.011111
25	0.94	0.943333	-0.003333

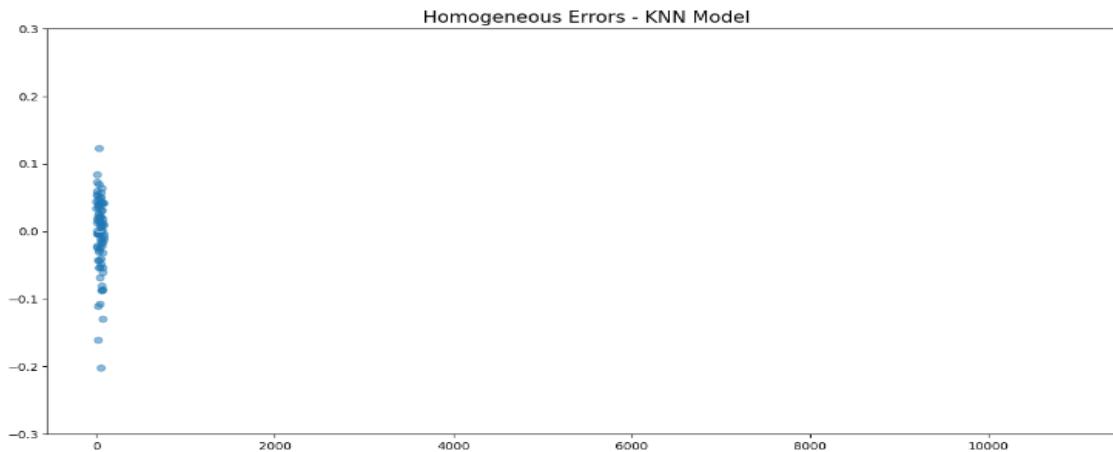
80 rows × 3 columns

- In KNN regression, the model stores all the values and it then predicts the target based upon similarity measure.
- In the image, we are calculating the performance of KNN and the true value is stored in `y_test`, the predicted values derived using the model which we got by the `knn.predict` from `x_test`, and the error achieved by subtracting the predicted from the true value.
- The errors can be positive or negative errors.

### KNN Error plot

```
In [88]: plt.figure(figsize=(15,7))
# Errors
ax_x= performance_knn['True Value']
ax_y= performance_knn['Prediction']
yerr= performance_knn['Error']

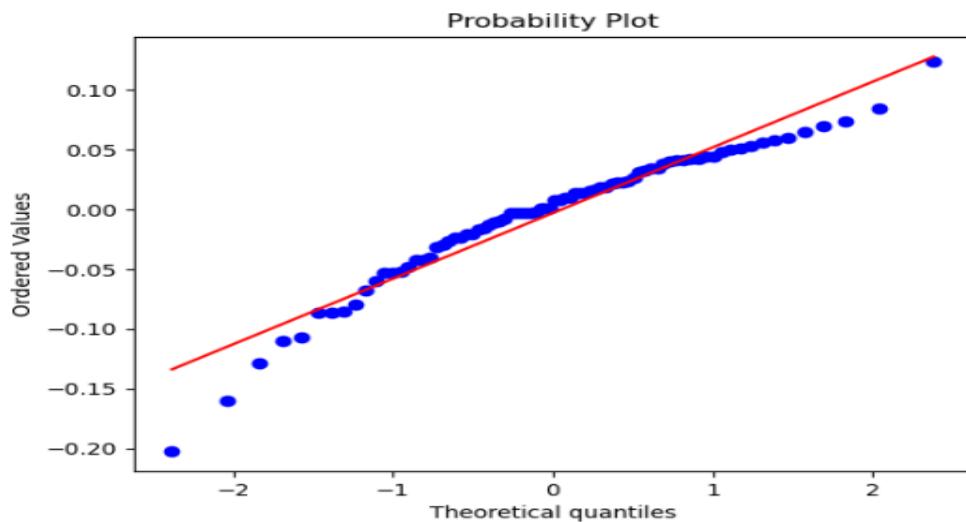
plt.scatter(range(len(yerr)), yerr, alpha=.5)
plt.title('Homogeneous Errors - KNN Model', size=15)
plt.hlines(y=0, xmin=0, xmax=11000, linestyle='--', color='white', alpha=.5)
plt.ylim(-.3, .3)
plt.show()
```



- In the snippet above, a scatter plot graph with error values that were derived by subtracting the expected value from the actual value is displayed.
- As we observe the graph, we can see that the graph is similar to the above model graph and the values are close to the zero.

### KNN Probability Plot

```
In [89]: #QQ Plot
probplot(yerr, dist='norm', plot=plt);
plt.show()
```



- This probability graph displayed in the above snippet shows how close are the predicted values to the actual values.
- The red solid line represents the actual values, while blue dots represent the predicted

values, and we could see using the KNN algorithm the predicted values are close to the actual values.

- Suppose take the theoretical value of -0.19 so it matches to the ordered value of -0.14.

Predicted

```
In [301]: # model evaluation for testing set
mae = metrics.mean_absolute_error(y_test, preds_knn)
mse = metrics.mean_squared_error(y_test, preds_knn)
r2 = metrics.r2_score(y_test, preds_knn)

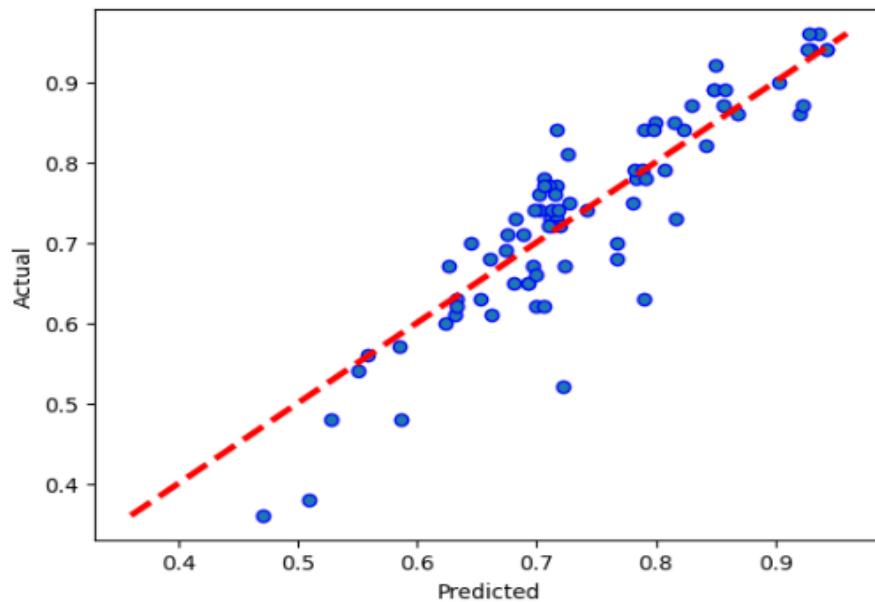
print("The model performance for testing set")
print("-----")
print('MAE is {}'.format(mae))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))

The model performance for testing set
-----
MAE is 0.04120833333333333
MSE is 0.0030638117283950618
R2 score is 0.81233543253736
```

- Here, we assess the model using its mean absolute error (MAE), mean squared error (MSE), and R2 score.
- The main thing that we consider is the R2 score that is accuracy which helps to find out which is the best model.
- The mean absolute value for the above model is 0.041208 and the mean square error is .003063 and the R2 value is 0.812335.

## Graph of predicted values to actual values

```
In [299]: fig, ax = plt.subplots()
ax.scatter(preds_knn, y_test, edgecolors=(0, 0, 1))
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=3)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
```



- Here we created a graph that is plotted based on predicted and actual values.
- As we can see most of the predicted values are close to the actual values.
- Take the values from 0.6 to 0.7 we can see all the values are close to each other.

## Decision Tree Regression:

### DT Performance

```
In [94]: # Predictions  
preds_dt = dec_tree_model.predict(X_test)  
  
# Performance  
performance_dt = pd.DataFrame({ 'True Value': y_test,  
                                'Prediction': preds_dt,  
                                'Error': y_test - preds_dt})  
  
# View  
performance_dt
```

Out[94]:

	True Value	Prediction	Error
180	0.71	0.73	-0.02
369	0.67	0.63	0.04
258	0.77	0.75	0.02
282	0.81	0.78	0.03
291	0.56	0.58	-0.02
...	...	...	...
152	0.86	0.88	-0.02
336	0.72	0.71	0.01
231	0.74	0.79	-0.05
165	0.78	0.71	0.07
25	0.94	0.95	-0.01

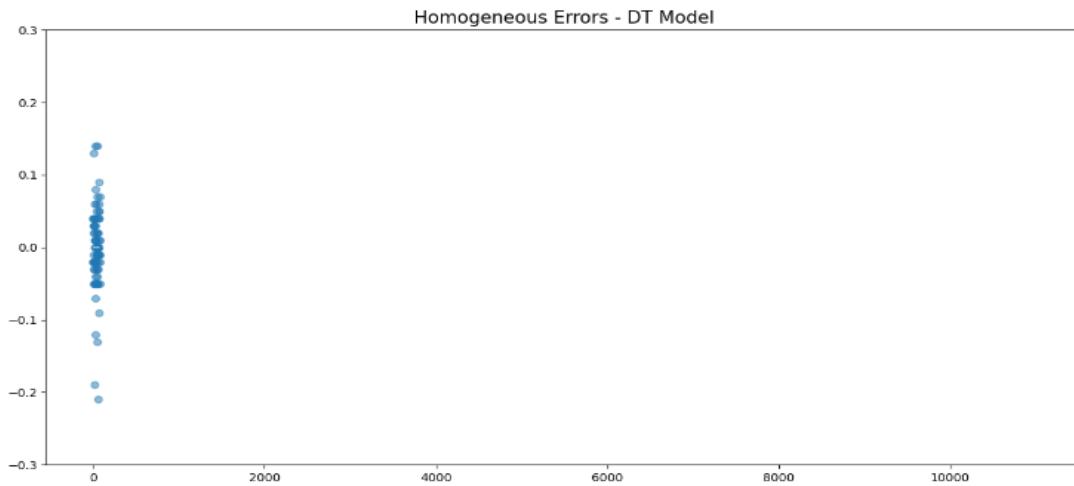
80 rows × 3 columns

- Here, we are calculating the performance of Decision Tree regression and displaying the true value that is stored in `y2_test`, Displaying the predicted values that we calculated using the model and the error that is obtained by subtracting the predicted from the true value.
- In the error column, we could observe some are positive errors and some are negative errors. If the predicted value is greater than the actual value then the resulting error is a negative error, else it is a positive error.
- The error values are near to zero in most of the cases.

### DT Error plot

```
In [95]: plt.figure(figsize=(15,7))
# Errors
ax_x= performance_dt['True Value']
ax_y= performance_dt['Prediction']
yerr= performance_dt['Error']

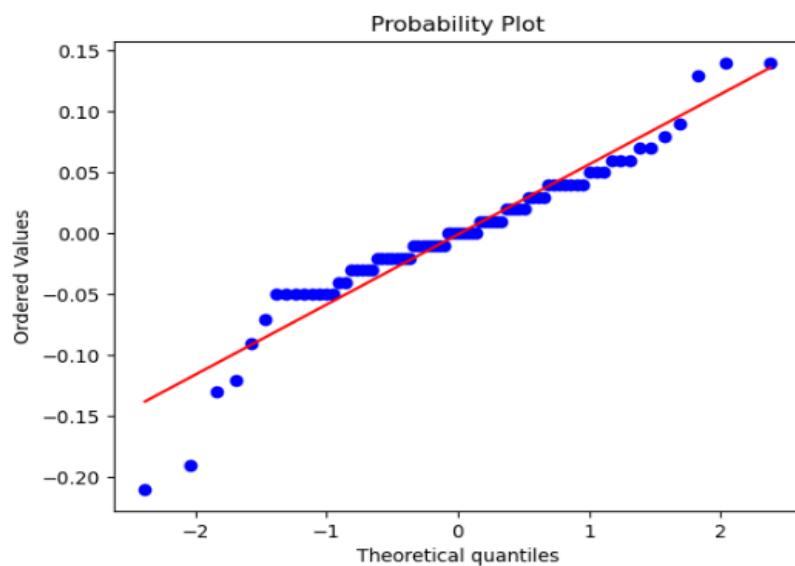
plt.scatter(range(len(yerr)), yerr, alpha=.5)
plt.title('Homogeneous Errors - DT Model', size=15)
plt.hlines(y=0, xmin=0, xmax=11000, linestyle='--', color='white', alpha=.5)
plt.ylim(-.3, .3)
plt.show()
```



- In the above image, we have plotted a graph with error values that were obtained by subtracting the expected value from the actual value is displayed.
- This is almost same as the above graph the values are close to zero.

### DT Probability Plot

```
In [96]: #QQ Plot
probplot(yerr, dist='norm', plot=plt);
plt.show()
```



- Here we are plotting the graph between the Theoretical quantities and the ordered values

and the resultant graph is probability graph, here red line is the actual value and all the blue dots are the predicted values we obtained by Decision tree algorithm.

- The probability plot of decision tree is different when compared to the above two graphs.

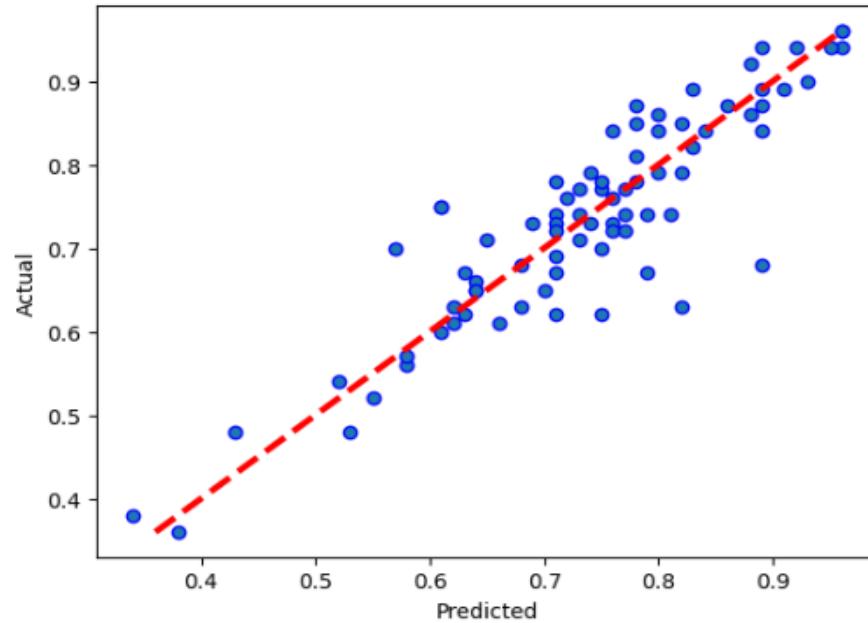
```
03]: # model evaluation for testing set
mae = metrics.mean_absolute_error(y_test, preds_dt)
mse = metrics.mean_squared_error(y_test, preds_dt)
r2 = metrics.r2_score(y_test, preds_dt)

print("The model performance for testing set")
print("-----")
print('MAE is {}'.format(mae))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))

The model performance for testing set
-----
MAE is 0.04112500000000001
MSE is 0.003403749999999997
R2 score is 0.7915135366899424
```

- Here, we evaluate the model using its R2 score, mean squared error (MSE), and mean absolute error (MAE).
- These are the main features which will show how good a model is.
- The MAE value is 0.04112, MSE value is 0.003403 and the R2 value is 0.79151.
- The accuracy value is low compared to above models.

```
In [302]: fig, ax = plt.subplots()
ax.scatter(preds_dt, y_test, edgecolors=(0, 0, 1))
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=3)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
```



- Here, a graph based on expected and actual values has been constructed.
- For instance, take the 0.6 value of predicted which matches to the 0.6 value of the actual value.
- And the last value of predicted is 1 is which is same to 1 in predicted.

## Random Forest Regression:

```
RF performance
```

---

In [101]:

```
# Predictions
preds_rf2 = rand_forest_model2.predict(X2_test)

# Performance
performance_rf = pd.DataFrame({ 'True Value': y2_test,
                                  'Prediction': preds_rf2,
                                  'Error': y2_test - preds_rf2})
# View
performance_rf
```

Out[101]:

	True Value	Prediction	Error
180	0.71	0.6749	0.0351
369	0.67	0.6345	0.0355
258	0.77	0.7178	0.0522
282	0.81	0.7433	0.0667
291	0.56	0.5725	-0.0125
...	...	...	...
111	0.69	0.7206	-0.0306
340	0.75	0.7221	0.0279
178	0.72	0.7059	0.0141
86	0.72	0.7261	-0.0061
376	0.34	0.4474	-0.1074

100 rows × 3 columns

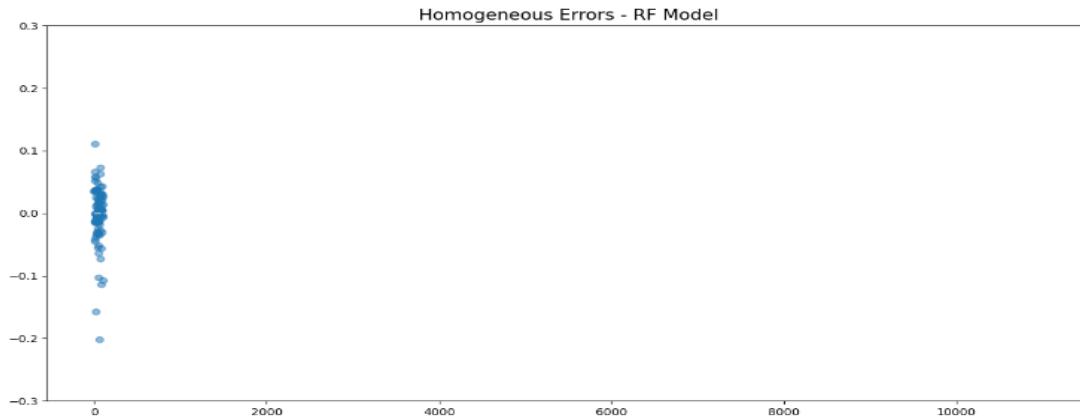
---

- In this section, we calculate the effectiveness of Random Forest Regression and display the real value that is stored in `y2_test`, the predicted values that we computed using the model, and the error that is derived by deducting the predicted from the true value.
- We can see that some of the errors in the error column are positive and some are negative. The resulting error is either positive or negative depending on whether the anticipated value is bigger than the actual value.

#### RF Error plot

```
In [102]: plt.figure(figsize=(15,7))
# Errors
ax_x= performance_rf['True Value']
ax_y= performance_rf['Prediction']
yerr= performance_rf['Error']

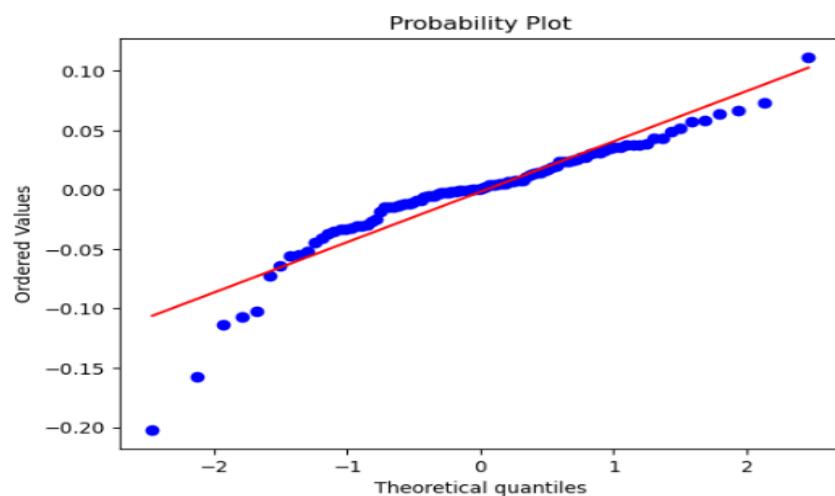
plt.scatter(range(len(yerr)), yerr, alpha=.5)
plt.title('Homogeneous Errors - RF Model', size=15)
plt.hlines(y=0, xmin=0, xmax=11000, linestyle='--', color='white', alpha=.5)
plt.ylim(-.3, .3)
plt.show()
```



- The graph showing error values that were obtained by deducting the expected value from the actual value is shown in the above image.
- We can see the above result is similar to all the above graphs.

#### RF Probability Plot

```
In [103]: #QQ Plot
probplot(yerr, dist='norm', plot=plt);
plt.show()
```



- The red line represents the actual value, and all the blue dots represent the predicted values we received using the Random Forest algorithm. In this graph, we are charting the relationship between theoretical quantities and ordered values, and the resultant graph is a probability graph.

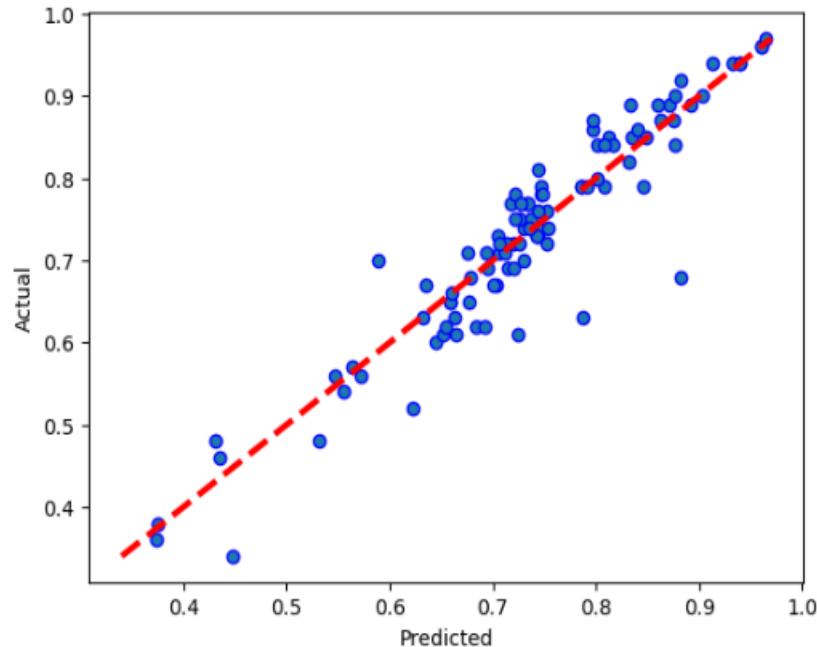
```
In [304]: # model evaluation for testing set
from sklearn import metrics
mae = metrics.mean_absolute_error(y2_test, preds_rf2)
mse = metrics.mean_squared_error(y2_test, preds_rf2)
r2 = metrics.r2_score(y2_test, preds_rf2)

print("The model performance for testing set")
print("-----")
print('MAE is {}'.format(mae))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))

The model performance for testing set
-----
MAE is 0.02942199999999934
MSE is 0.001965492999999974
R2 score is 0.8860247458525444
```

- Here, we are doing the model evaluation by mean absolute error and mean square error and also base on the r2 score above image displays the values of these respectively.
- The R2 value is 0.8860 and MAE is 0.294 and the MSE is 0.0019.
-

```
In [305]: fig, ax = plt.subplots()
ax.scatter(preds_rf2, y2_test, edgecolors=(0, 0, 1))
ax.plot([y2_test.min(), y2_test.max()], [y2_test.min(), y2_test.max()], 'r--', lw=3)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
```



- Here, an expected and actual value graph has been created for Random Forest regression model.
- Most of the values are common in predicted and actual value between the 0.7 to 0.8 values.

## SVM Regression:

### SVM performance

```
In [75]: # Predictions
preds_svm = SVM_model2.predict(X2_test)

# Performance
performance_svm = pd.DataFrame({ 'True Value': y2_test,
                                  'Prediction': preds_svm,
                                  'Error': y2_test - preds_svm})
# View
performance_svm
```

Out[75]:

	True Value	Prediction	Error
180	0.71	0.609032	0.100968
369	0.67	0.582166	0.087834
258	0.77	0.747058	0.022942
282	0.81	0.685840	0.124160
291	0.56	0.584578	-0.024578
...	...	...	...
111	0.69	0.745169	-0.055169
340	0.75	0.689705	0.060295
178	0.72	0.672630	0.047370
86	0.72	0.698135	0.021865
376	0.34	0.562550	-0.222550

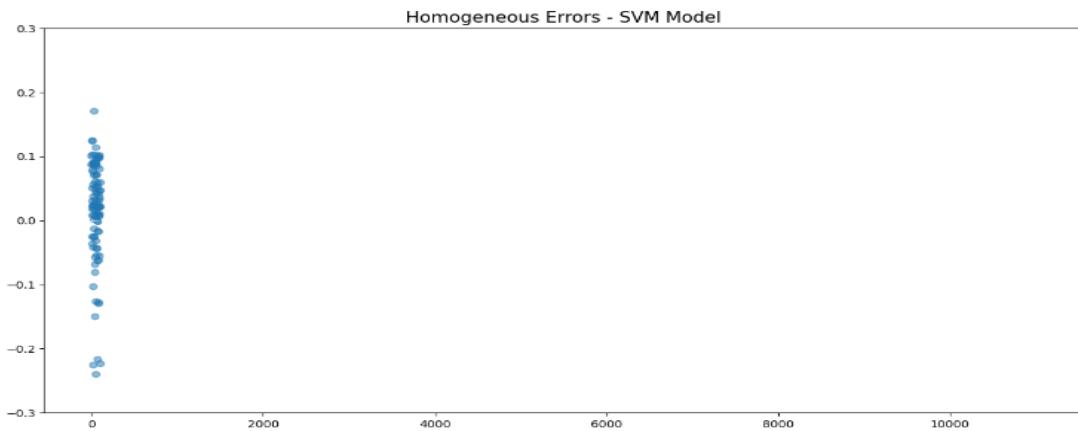
100 rows × 3 columns

- The efficacy of SVM Regression is determined in this part, and the true value that is stored in `y2_test`, the predicted values that we computed using the model, and the error that is produced by subtracting the predicted from the true value are all displayed.
- We can see that there are both positive and negative mistakes in the error column. Depending on whether the anticipated value is greater than the actual value, the resulting error can be either positive or negative.

### SVM Error plot

```
In [76]: plt.figure(figsize=(15,7))
# Errors
ax_x= performance_svm['True Value']
ax_y= performance_svm['Prediction']
yerr= performance_svm['Error']

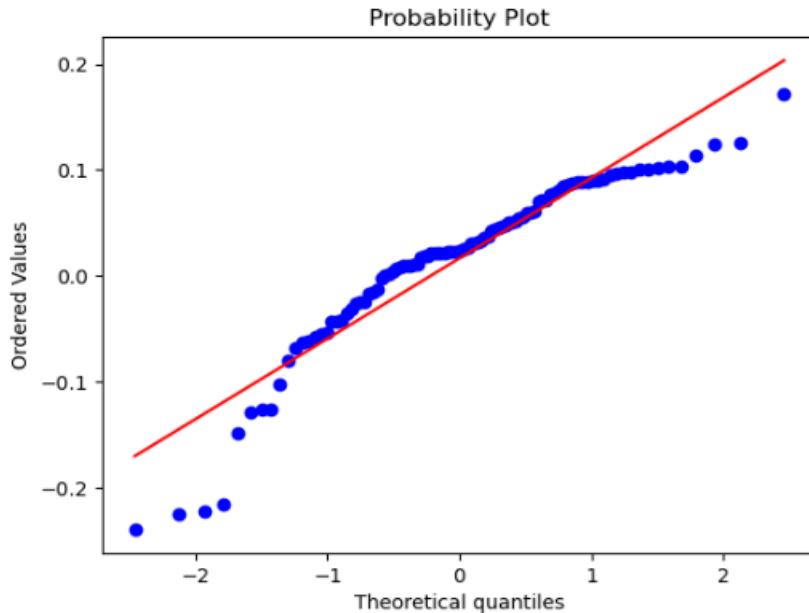
plt.scatter(range(len(yerr)), yerr, alpha=.5)
plt.title('Homogeneous Errors - SVM Model', size=15)
plt.hlines(y=0, xmin=0, xmax=11000, linestyle='--', color='white', alpha=.5)
plt.ylim(-.3, .3)
plt.show()
```



- The figure above displays a graph of the error values that were calculated by subtracting the expected value from the actual value.
- The values are close to zero in many of the cases.
- And this graph is similar in all the regressions.

### SVM Probability plot

```
In [77]: from scipy.stats import probplot
#QQ Plot
probplot(yerr, dist='norm', plot=plt);
plt.show()
```



- The blue dots reflect the anticipated values we received from the SVM method, and the red line represents the actual value.
- This graph is a probability graph because it shows the relationship between theoretical quantities and ordered values.

```
In [290]: # model evaluation for testing set
mae = metrics.mean_absolute_error(y2_test, svm_prediction)
mse = metrics.mean_squared_error(y2_test, svm_prediction)
r2 = metrics.r2_score(y2_test, svm_prediction)

print("The model performance for testing set")
print("-----")
print('MAE is {}'.format(mae))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
```

```
The model performance for testing set
-----
MAE is 0.06247063507802804
MSE is 0.006413788747119458
R2 score is 0.628076415178771
```

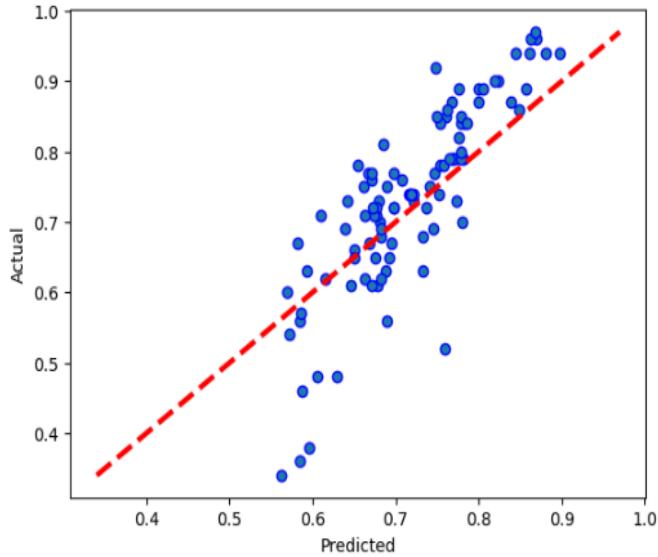
- In this case, we are evaluating the model based on its mean absolute error = 0.06 and mean square error=0.006, as well as the r2 score = 0.62, which is displayed in the above

image.

- The R2 value is more important compared to all the values.

```
In [288]: svm_prediction=SVM_model2.predict(X2_test)
```

```
In [289]: fig, ax = plt.subplots()
ax.scatter(svm_prediction, y2_test, edgecolors=(0, 0, 1))
ax.plot([y2_test.min(), y2_test.max()], [y2_test.min(), y2_test.max()], 'r--', lw=3)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
```



- Above graph depicts the linear relation between expected and actual value graph as shown here for SVM model.
- Most of the values in predicted and the actual are common around 0.7 value.

## Final Comparison and Prediction check:

compare model scores with 80-20 split data

```
In [106]: pd.DataFrame({'Multiple Linear Regression':[MLR_Score],
                      'Support Vector machine': [svm_score],
                      'KNN Regression': [knn_score],'Decision Tree Regression':[DT_score], 'Random Forest Regression':[rf_score]})
```

	Multiple Linear Regression	Support Vector machine	KNN Regression	Decision Tree Regression	Random Forest Regression
0	0.872038	0.615008	0.812335	0.791514	0.867206

compare model scores with 75-25 split data

```
In [105]: pd.DataFrame({'Multiple Linear Regression':[mlr_score2],
                      'Support Vector machine': [svm_score2],
                      'KNN Regression': [knn_score2],'Decision Tree Regression':[dt_score2], 'Random Forest Regression':[rf_score2]})
```

	Multiple Linear Regression	Support Vector machine	KNN Regression	Decision Tree Regression	Random Forest Regression
0	0.877163	0.628076	0.81546	0.75732	0.886025

We choose Random Forest Regression with 75% training data and 25% testing data

As shown above , after comparing the R2 score or accuracy obtained from all 5 algorithm models with both 80-20 split data and 75-25 split data , It is evident that Random forest regression and multiple linear regression model performed way better than other 3 models. Hence We choose random forest model with 75-25 split data as it has the highest accuracy of 88.6 score.

	A	B	C	D	E	H	I	J	K	L	N	O	P	Q	R
93	92	Brungi Manish Kun	299	97	3	7.66	0	5 NO	2	4 YES	0	1	0.38		
94	93	Kalla Girish Pavan	298	98	2	8.03	0	4 YES	2	4 YES	0	1	0.34		
95	94	Karururi Sri Harika	301	97	2	7.88	1	4 YES	2	1 NO	0	1	0.44		
96	95	Katukuri Pavankun	303	99	3	7.66	0	5 NO	2	4 YES	0	1	0.36		
97	96	Kintali Amrutha	304	100	4	7.84	0	5 YES	1	1 NO	0	1	0.42		
98	97	Konapala Hemanth	306	100	2	8	0	6 YES	1	1 NO	0	0	0.48		
99	98	Mandalapu Purnac	331	120	3	8.96	1	1 YES	0	0 NO	6	0	0.86		
100	99	Mathsa Vikash	332	119	4	9.24	1	0 YES	0	0 NO	3	0	0.9		
101	100	Motakatla Sai Pava	323	113	3	8.88	1	1 YES	0	0 NO	3	0	0.79		
102	101	Nithya Raju Uppala	322	107	3	8.46	1	2 YES	0	0 NO	2	0	0.71		
103	102	Pandava Swetha	312	105	2	8.12	0	4 YES	0	0 NO	4	0	0.64		
104	103	Pandava Swetha	314	106	2	8.25	0	5 YES	1	0 NO	2	0	0.62		

We choose Random Forest Regression with 75% training data and 25% testing data

```
In [114]: Mathsa_Vikash= [[332,119,4,9.24,1,0,1,0,0,0,3,0]]
print(rand_forest_model2.predict(Mathsa_Vikash))

[0.8986]
```

With the chosen model , we try to check the prediction for any particular row. For example , we tried to input the feature values of Mr. mathsa Vikash who got 0.90 i.e., 90% chances of visa approval as original prediction. When checked with Random Forest model , we got 0.8986 as prediction which is almost equal to 90%.

## **Project Management:**

### **Work Completed:**

- ***Description***

We aimed at finishing the data analysis part of the project which we finished . We used HDFS to store our dataset in Hadoop by making a directory and putting our dataset into it. Then we used hive for easy query execution instead of implementing direct map-reduce code. We created hive tables and ran different queries analyzing the data. Then we implemented the same queries in Solr-Lucene and Cassandra too.

Next , we split the data into training and testing sets and implemented 5 Machine Learning regression algorithms namely Multiple Linear regression, K Nearest Neighbor , Support Vector Machine , Decision Tree regression , Random Forest Regression. Then we analyzed the performance of these 5 models and picked the best possible model which in our case is Random Forest regression model when data is split into 75% training data and 25% testing data.

- ***Responsibility***

Suhas Siddarajgari Tellatakula	Data analysis features, Data Preprocessing, Machine Learning features, Data visualization and making report.
Sai Tejesh Gonemadatala	Data analysis features, Data Preprocessing, Machine Learning features, Data visualization and making report.
Sai Rohith Varma Kantem	Data analysis features, Data Preprocessing, Machine Learning features, Data visualization and making report.
Sai Praneeth Reddy Avula	Data analysis features, Data Preprocessing, Machine Learning features, Data visualization and making report.

- *Contributions*

Suhas Siddarajgari Tellatakula	25% in implementing Data analysis features, Data Preprocessing, Machine Learning features, Data visualization and making report.
Sai Tejesh Gonemadatala	25% in implementing Data analysis features, Data Preprocessing, Machine Learning features, Data visualization and making report.
Sai Rohith Varma Kantem	25% in implementing Data analysis features, Data Preprocessing, Machine Learning features, Data visualization and making report.
Sai Praneeth Reddy Avula	25% in implementing Data analysis features, Data Preprocessing, Machine Learning features, Data visualization and making report.

- *Issues*

- Finding the appropriate dataset for our problem statement was difficult.
- Cleaning the dataset and renaming the column names meaningfully.
- Implementing the data analysis on Hadoop ecosystem on a single laptop was difficult as a group.
- Implementing 5 models for both 80-20 and 75-25 split datasets every time and picking the right datasets every time was a confusing task as we are used to fit for only one dataset.
- Sometimes , the data saved in csv format used to give NaN values unknowingly. We used to pick up the dataset from backup most of the times.

## **References / Bibliography:**

5. <https://www.kaggle.com/code/aryantiwari123/graduate-admission-prediction>
6. <https://www.kaggle.com/datasets/saddamazyazy/go-to-college-dataset>
7. <https://www.jetir.org/papers/JETIR2204278.pdf>
8. <https://www.kaggle.com/code/akhilkasare/h-1b-visa-prediction-using-machine-learning>
9. <https://www.kaggle.com/code/campusx/gre-admission-prediction/data>
10. <https://ieeexplore.ieee.org/document/8933628>
11. <https://www.wallstreetmojo.com/multiple-regression-formula/>
12. <https://towardsdatascience.com/machine-learning-basics-multiple-linear-regression-9c70f796e5e3>
13. <https://bookdown.org/f100441618/bookdown-regresion/ml-tools.html>
14. <https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/>
15. <https://levelup.gitconnected.com/random-forest-regression-209c0f354c84>
16. <https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345>