Dublin Business School

excellence through learning

Data Analytics and Visualisation Assignment

Data Analytics for IMDB Movie and Salary Prediction datasets

Module Title: Enterprise Information System (B9IS107)

Module Leader: Dr. Shazia A Afzal

Student Name: Manik Mahashabde (10518579)

ACKNOWLEDGMENT

I would like to express my sincere gratitude to our module leader Dr. Shazia Afzal for providing her invaluable guidance, comments, and suggestions throughout the entire project. I would also like to thank Mr. Trevor Haugh for giving a lecture on report writing and referencing.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF FIGURES

PART I: DATA WAREHOUSING

1: INTRODUCTION

1.1 ABOUT DATASET

In part 1, the **IMDB** dataset is used. This dataset tells about different movies released within the period 1895-2004. Each movie consists of different genres such as comedy, horror, drama, romance, documentary, thriller, etc. Each movie has one or more directors and more than one actor. There are mainly 6 tables in this dataset which are actors, director, director_movies, movies_genres, movies, role. The complete IMDB dataset is included in the folder structure of CA.

1.2 REASON FOR SELECTING THE SUBJECT AREA AND DATASET

The reason for selecting this dataset is because data is well versed with many attributes such as movie **year, movie rating, genre, actor_name, gender, director**, etc. It gave a good level of analysis, reporting, and visualizations as described further in the report.

1.3 VISION AND GOALS

The vision and goals of this assignment are developing a data warehouse consisting of fact and dimensions table using SSIS ETL tool of the IMDB dataset. The report answers questions for example: what are the top 10 movies, the top 10 directors, the highest watched genre, etc. Then creating reports using SSRS and performing visualization in Tableau.

1.4 KEY STAKEHOLDERS

There are two important stakeholders in this dataset: **actor** and **director.** An actor is a one who acts in a movie of a particular genre, release on a given year along with ratings. A director is one who directs a movie.

1.5 BUSINESS REQUIREMENTS

The business requirements consist of:

- What are the different genres categories of given movies?
- Who are the different directors and actors in a movie?
- What is the top-rated genre of all time?
- Who are actors and directors of the highest-rated movie of each year of a particular genre?
- How many male and female actors are there in a movie.
- Top actors and directors of all time.
- Genre preference change over time.
- Using visualization for data analysis.
- Generating tabular and matrix reports of different business scenarios.

2: SCHEMA/DIMENSIONAL MODEL

The following is the star schema of the IMDB data warehouse as shown in figure 1 below

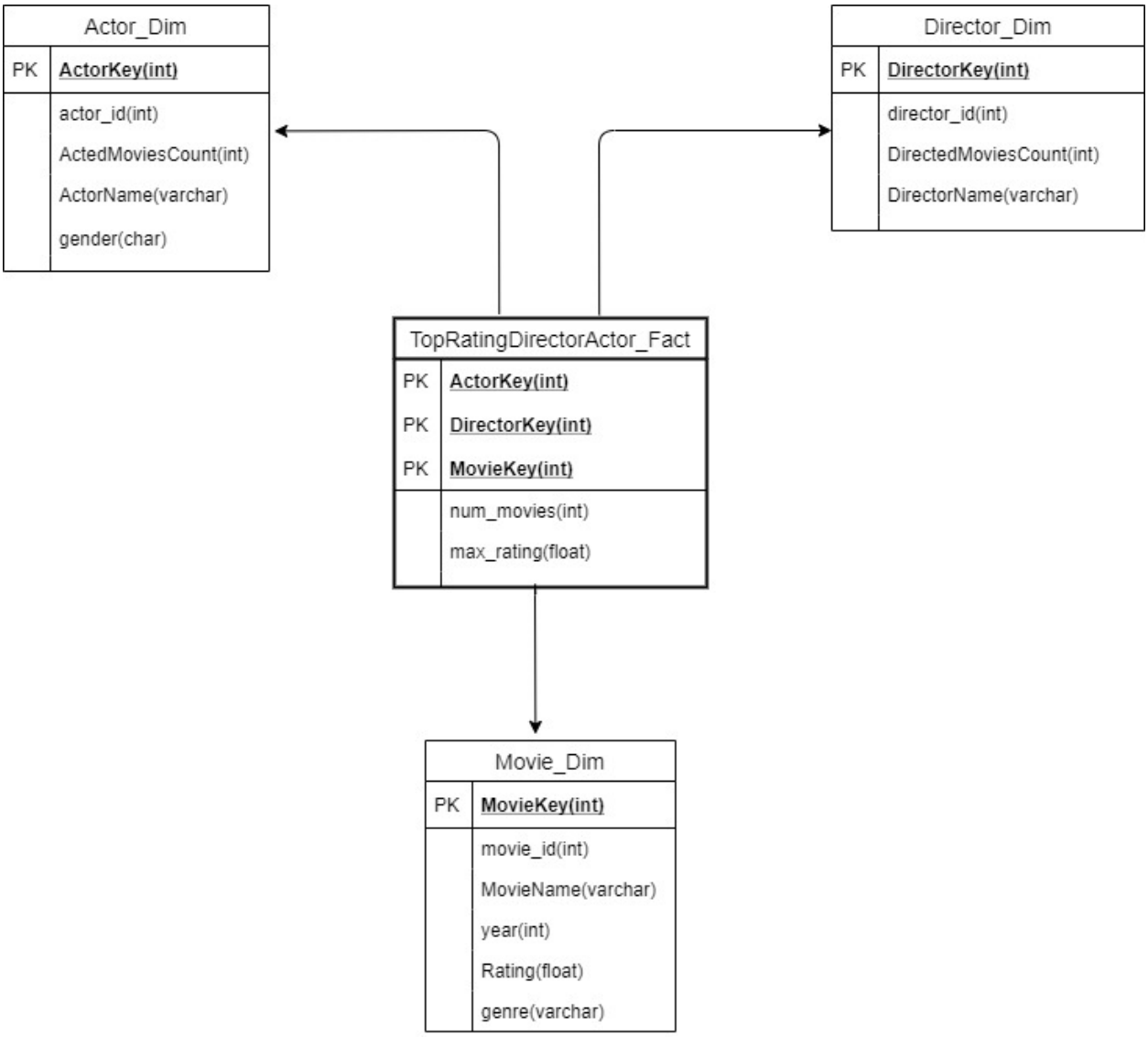


Figure 1: Star Schema of IMDB_DW

The database consists of 3 main entities **Movie, Actor,** and **Director.** Hence, star schema consists of 3 separate dimensions **Movie_Dim, Actor_Dim,** and **Director_Dim**. The fact table consists of the total number of movies in a particular genre of every year given as **num_movies** and rating of the highest-rated movie among the group num_movies is given by **max_rating**. The **Moviekey** in the fact table consists of the ID of the highest-rated movie with max_rating. It is also the primary key in Movie_Dim. Similarly, the **DirectorKey** in the fact table consists of the ID of the director of the highest-rated movie with max_rating and it is the primary key in Director_Dim. Finally, an **ActorKey** corresponds to the actor

of the highest-rated movie and it is the primary key of the Actor_Dim table. All the three attributes i.e Moviekey, ActorKey, and DirectorKey makes a primary key of the TopRatingDirectorActor_Fact.

3: IMPLEMENTATION OF THE DATA WAREHOUSE

Create table queries for all the tables in IMDB_DW are present in the appendix section A.

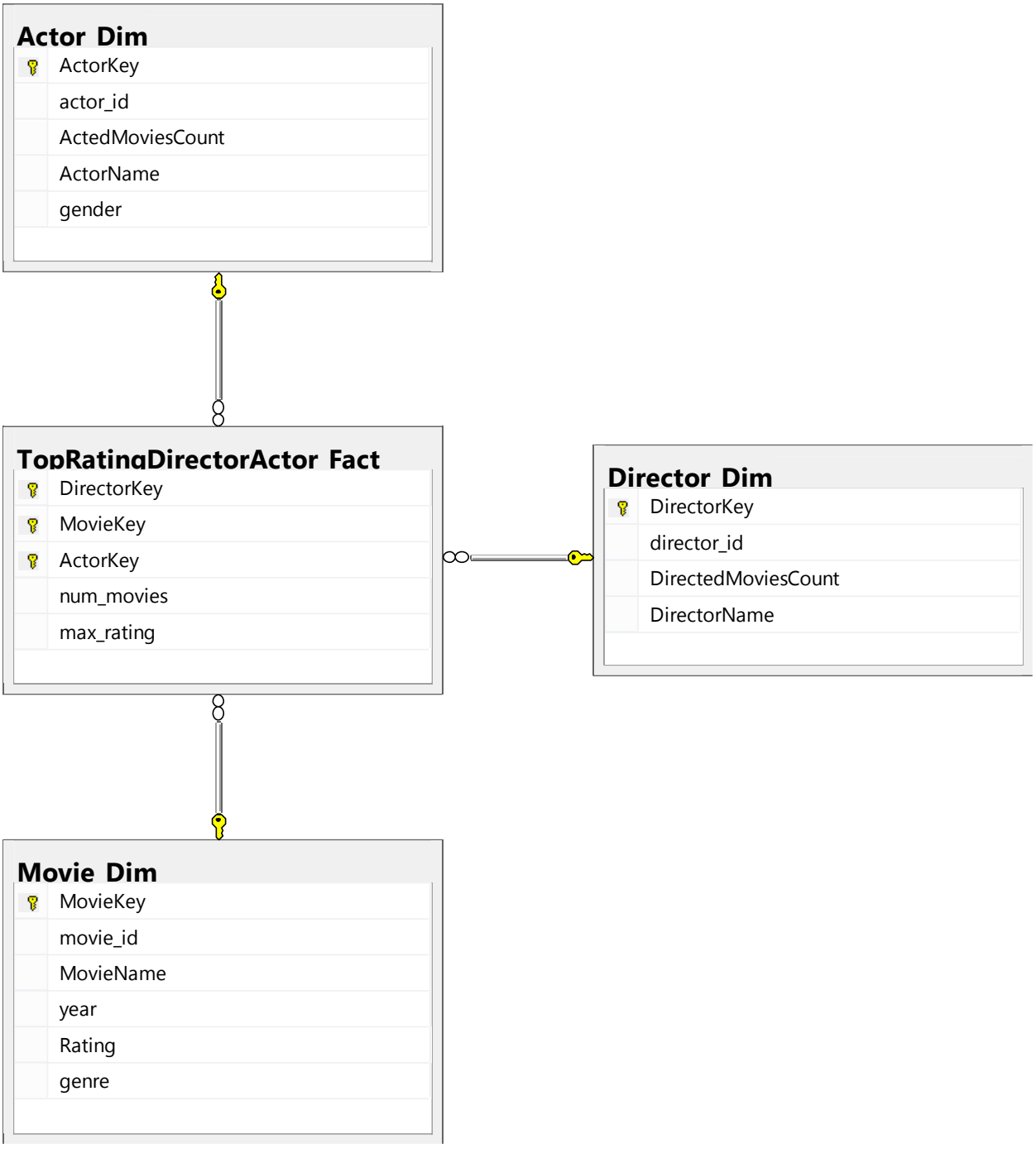


Figure 2: Data diagram of IMDB_DW

4: ETL FOR POPULATING THE DATA WAREHOUSE

The query for Director_Dim is given in appendix B the output of the query is added into an excel file and surrogate key DirectorKey is added into a sheet then it is used in ETL for populating Director_Dim table through SSIS as shown below.

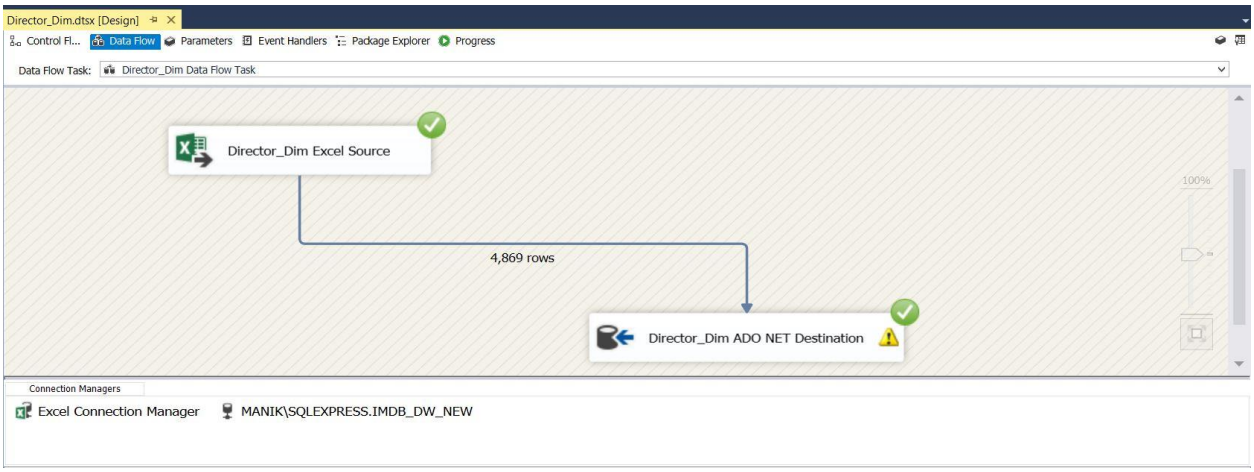


Figure 3: SSIS for Director_Dim

The query for Actor_Dim is given in appendix B the output of the query is added into an excel file and surrogate key ActorKey is added into a sheet then it is used in ETL for populating Actor_Dim table through SSIS as shown below.

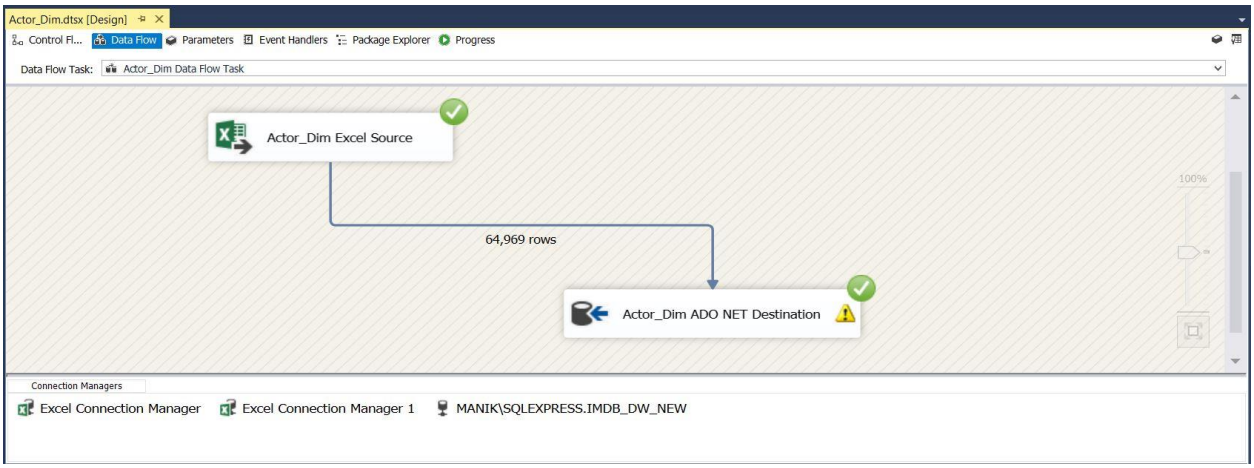


Figure 4: SSIS for Actor_Dim

The query for Movie_Dim is given in appendix B the output of the query is added into an excel file and surrogate key MovieKey is added into a sheet then it is used in ETL for populating Movie_Dim table through SSIS as shown below.

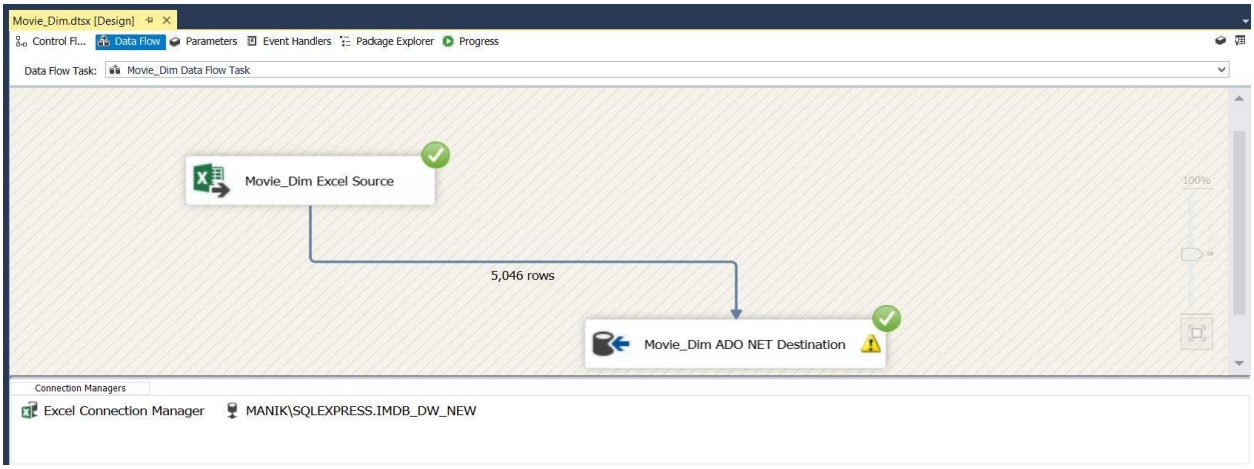


Figure 5: SSIS for Movie_Dim

The query for calculating different parameters for the fact table is given in Appendix B. After this, the query is added in IMDB OLE DB Source in ETL, and lookups for all the dimensions are added and finally, OLE DB Destination for the TopRatingDirectorActor_Fact table is selected and data is populated.

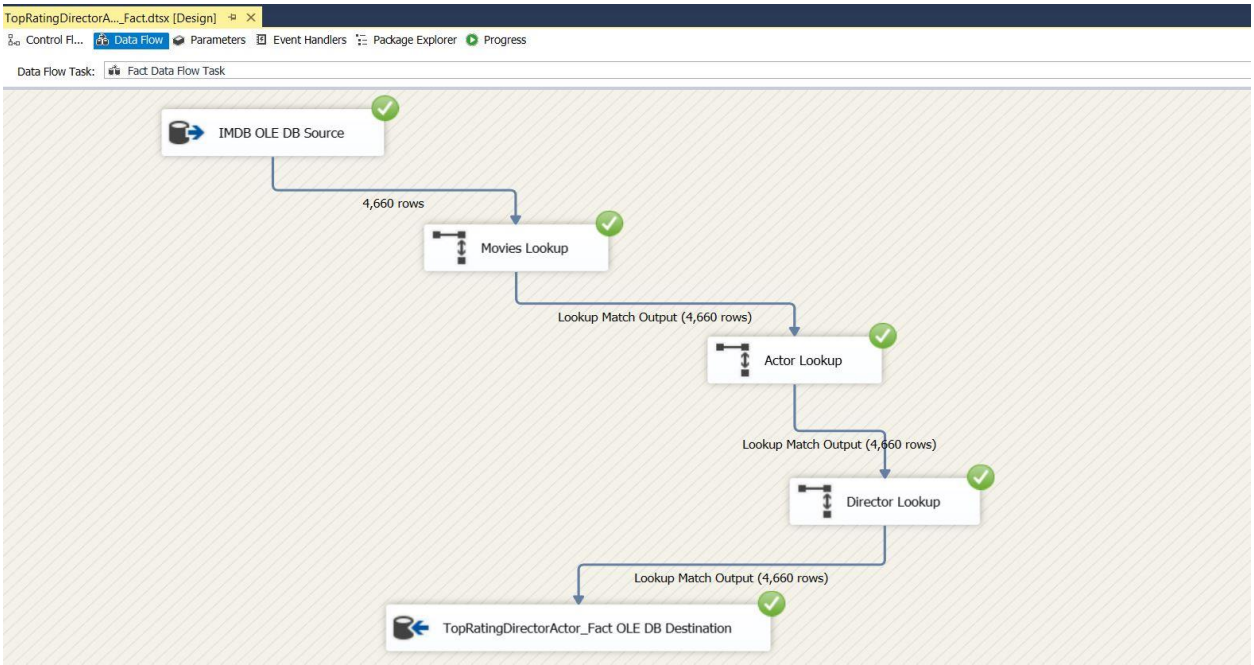


Figure 6: SSIS for `TopRatingDirectorActor_Fact`

5: VISUALISATION AND REPORTING

5.1 TABLEAU VISUALISATION

Business Scenario 1: Comparison of Top 3 movie genres which are Action, Comedy, and Drama.

This visualization is a motion chart over the period 1994-2004 on the X-axis vs the count of the number of movies on the Y-axis. It shows the variations in these 3 genres over time in figure 7 below.

Figure 7: Action-Comedy-Drame Genres Motion Chart in Tableau

Business Scenario 2: Top 10 directors.

It represents the top 10 directors with director names(X-axis) plotted against the movies with the maximum rating(Y-axis) along with the genre and the year in which the movie was released. A set of the top 10 directors is created and the rest of the directors are shown after that. The figure 8 on the tableau is shown on the next page.

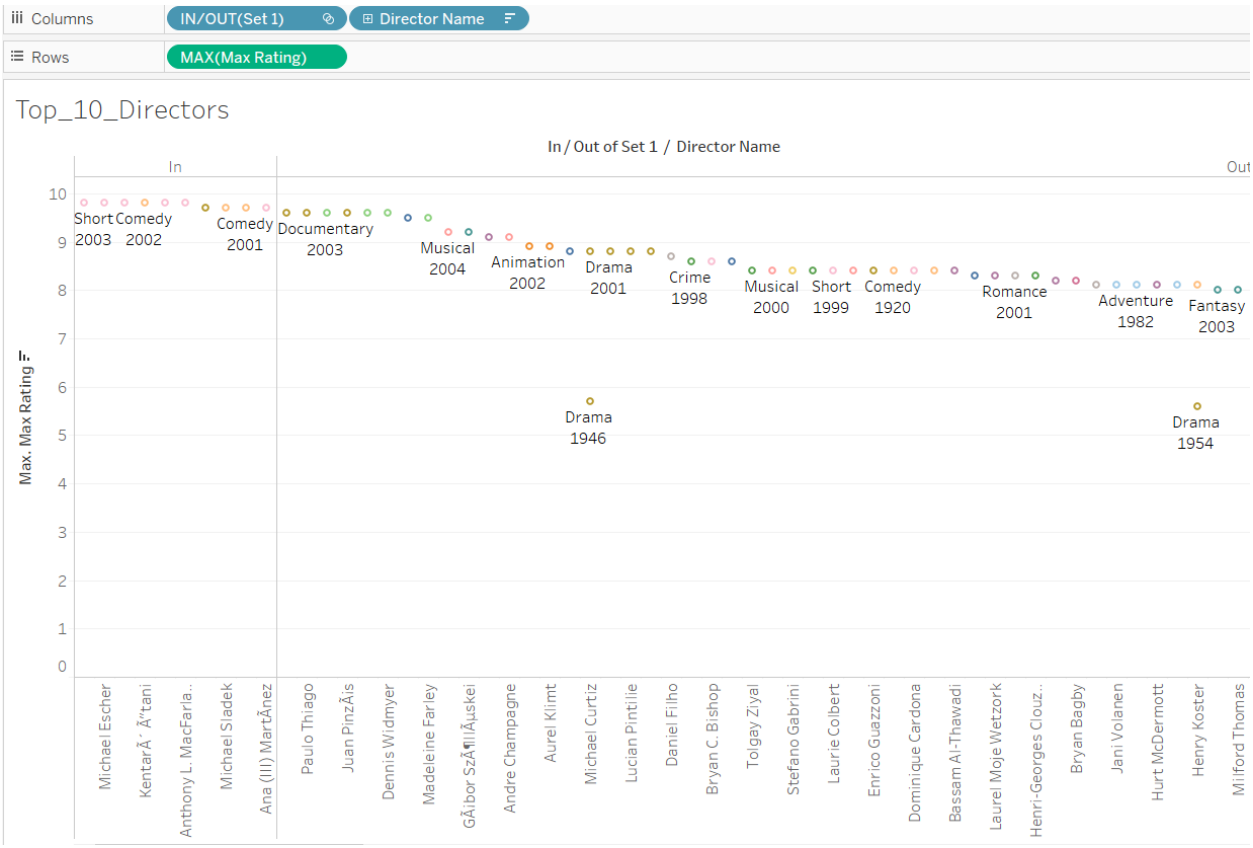


Figure 8: Top_10_Directors in Tableau

Business Scenario 3: Count of female and male actors in each movie

The movie name is plotted on the X-axis and count of Male, Female is plotted on Y-axis. Figure 9 can be seen on the next page.

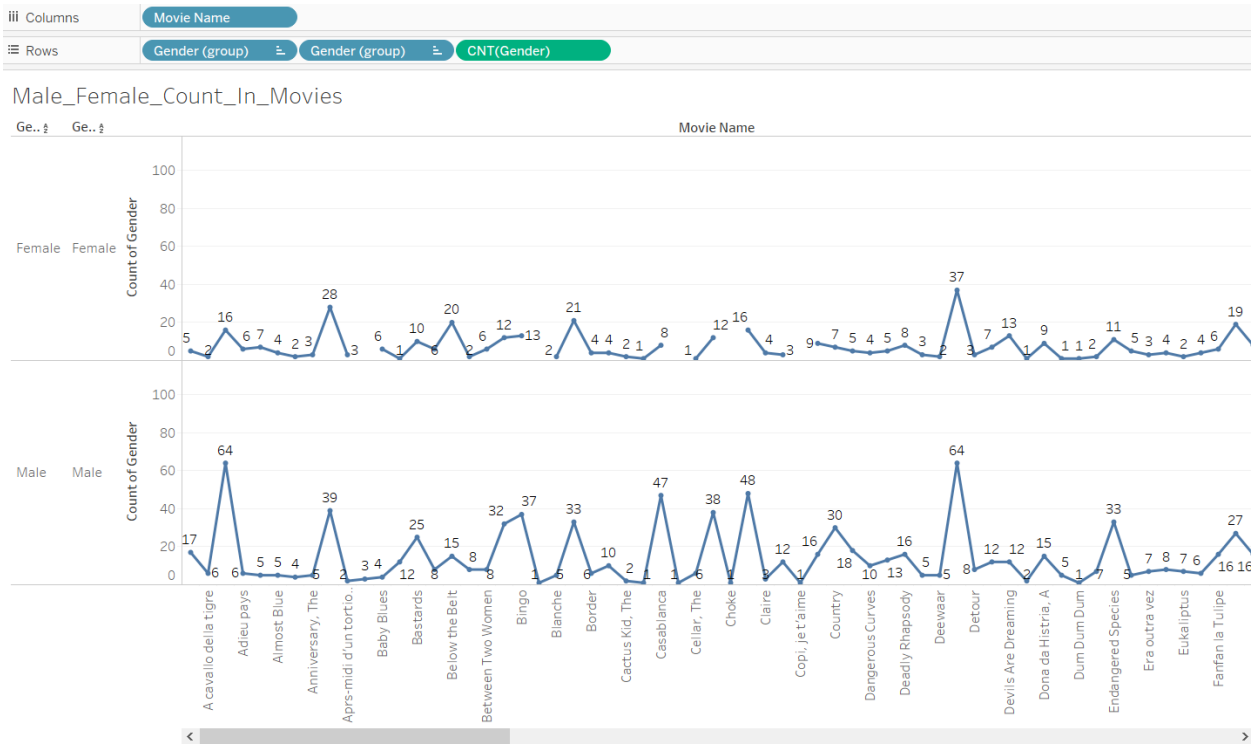


Figure 9: Male_Female_Count_In_Movies in Tableau

Business Scenario 4: It represents the list of highest-rated action movies of each year.

Among the list, the visualization also displays the name of the movie along with the rating details sorted with shades of blue. The movie with the lowest rating among the list is shown with light blue shade and the highest-rated movie among the list is shown with dark blue shade.

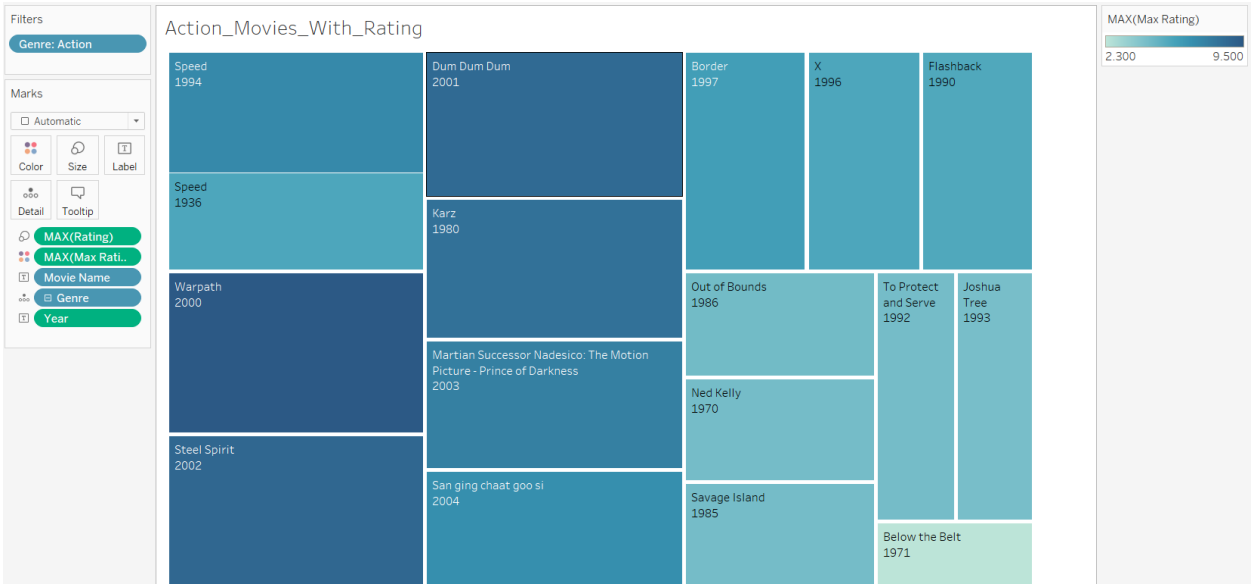


Figure 10: Top rated action movies list in Tableau

5.2 REPORT SSRS

Business Scenario 1: Movie genres categories.
The scenario represents different movie genres categories report in SSRS.
The query for this can be found in appendix section C. The tabular report consist of 3 attributes genre, Total_Movies, and Highest_Rating. The matrix report consists of plotting of the highest rating on genre attribute(X-axis) vs no. of movies attribute(Y-axis). The tabular and matrix report for this can be found in figures 10 and 11 respectively

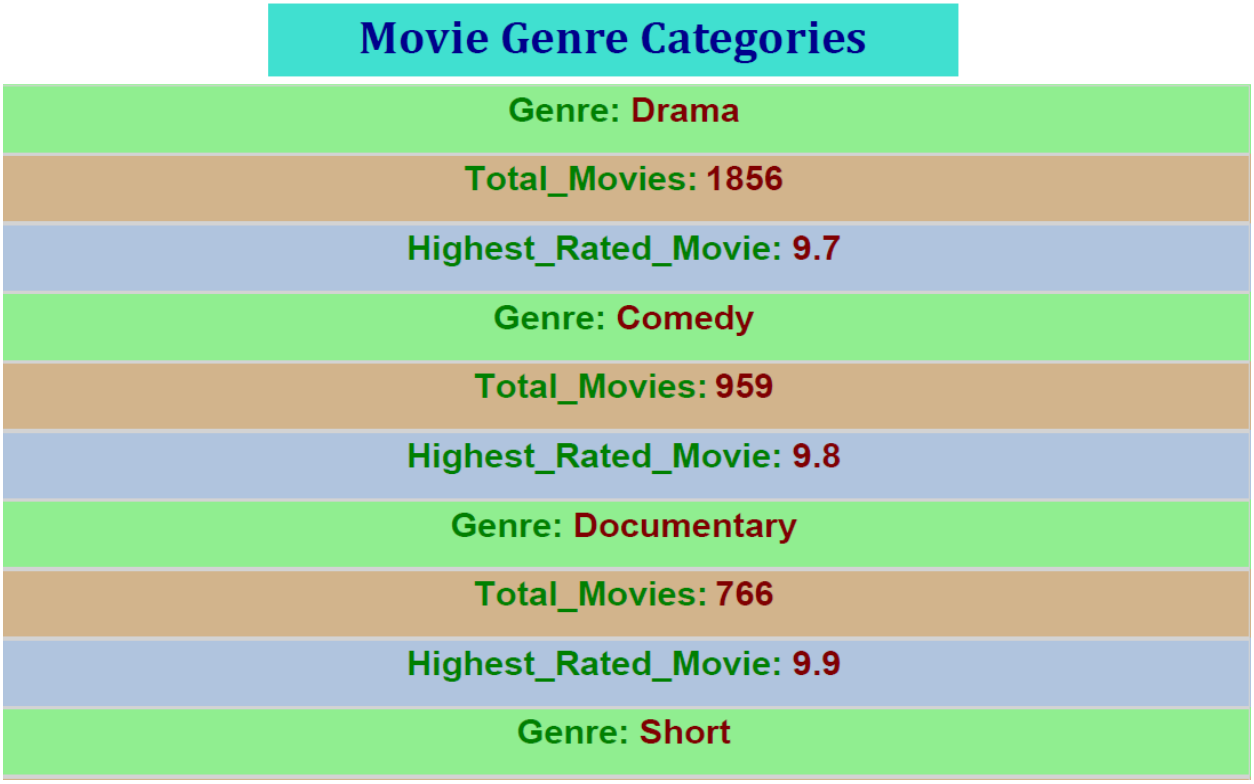


Figure 11: Tabular report of movie genre categories

Highest rated movie with respect to genre(X-axis) and total number of movies(Y-axis)				
No. of Movies	Action	Adult	Adventure	Animation
2		6.2		
3				
9				
11				
117				
19				
79			8.4	
44				
249				
37				
84				
100	9.5			

Figure 12: Matrix report of movie genre categories

Business Scenario 2: Highest rated actors and respective movies

The scenario represents the highest rated movies with their actors and respective ratings of the movie. The query for this can be found in appendix section C. The report consist of 3 attributes actor name, movie name, and max-rating. The tabular report for this can be found in figure 13.

Highest Rated Actors and Respective Movies		
Actor Name	Movie Name	Max Ratings
Asaka Seto	Travail	9.8
Christopher Behrens	Tea with Tiffany	9.8
David De Leon	Milagro of Boyle Heights, The	9.8
Hope Arden	Dimensia Minds Trilogy: The Reds	9.8
Jean Barr	Milagro of Boyle Heights, The	9.8
John Acosta	Milagro of Boyle Heights, The	9.8
Jun Murakami	Travail	9.8
Kevin (II) Small	Milagro of Boyle Heights, The	9.8
MÃ¡rcio Copetii	Copi, je t'aime	9.8
Martin Riddel	Dimensia Minds Trilogy: The Reds	9.8

Figure 13: Top rated actors, movies, and ratings

Business Scenario 3: Females acting in Drama genre over period 2000-2004
The scenario represents the **lead female actors in the drama genre over the period 2000-2004**. The query for this is available in appendix section C. The report consist of 3 attributes actor name, movie name, and year. The tabular report for this can be found in figure 14.

Females Acting in Drama Genre over the period 2000-2004

Actor Name	Movie Name	Year
Coca Bloos	Aprs-midi d'un tortionnaire, L'	2001
Dorina Chiriac	Aprs-midi d'un tortionnaire, L'	2001
Ioana Ana Macaria	Aprs-midi d'un tortionnaire, L'	2001
Isabel Vallejo	Era outra vez	2000
Mara SÃ¡nchez	Era outra vez	2000
Pilar Saavedra	Era outra vez	2000
Jennifer Joan Sullivan	Hunger (2001/I)	2001
Jude Narita	Hunger (2001/I)	2001
Kathleen Luong	Hunger (2001/I)	2001

Figure 14: Female actors in the drama genre

Business Scenario 4: Top rated movies, actors, and directors.
The scenario represents different genre categories, the total number of movies in that category. Then displays top ratings for a movie with actor name, director name, and movie name for every year in the SSRS report. The report consists of 6 attributes genres, num-movies, max rating, actor name, movie name, and its director name. The tabular report for this can be found in figure 15.

Top Rated Movies, Actors & Directors					
genre	num movies	max rating	Movie Name	Actor Name	Director Name
Short	29	9.8	Dimensia Minds Trilogy: The Reds	Hope Arden	Anthony L. MacFarland
Short	29	9.8	Dimensia Minds Trilogy: The Reds	Martin Riddel	Anthony L. MacFarland
Short	29	9.8	Dimensia Minds Trilogy: The Reds	Sam Schifrin	Anthony L. MacFarland
Thriller	20	8.4	Za'er	Ahmed Aqlan	Bassam Al-Thawadi
Thriller	20	8.4	Za'er	Ahmed Awad Ali	Bassam Al-Thawadi
Thriller	20	8.4	Za'er	Ahmed Mubarak	Bassam Al-Thawadi
Thriller	20	8.4	Za'er	Ali Al-Gurair	Bassam Al-Thawadi

Figure 15: Top genre, movie, actors, directors for each year

6: CONCLUSION

To conclude, this report successfully demonstrated the selection of the IMDB dataset along with the business requirements such as the highest watched genre for each year, the total number of males and females in each movie. The different directors and actors in each movie. Ratings of different movie genres, top directors, top actors, etc. The data warehouse for IMDB was created in the SQL server along with dimensions and fact table with queries present in Appendix A. After this data is populated with SSIS ETL as well as queries present in Appendix B. Output of the queries was added in an excel sheet with the addition of surrogate key column in them. Finally, data is populated using Excel source through ETL into dimensional tables. For adding data into the fact table OLE data source was used and lookup of each dimension table was added. Finally, the OLE DB destination was selected with a fact table connection and the data was populated successfully.

Once the data warehouse was completed, SSRS was used for generating different kinds of tabular and matrix reports. Queries for these reports are present in appendix C. These reports demonstrated different business scenarios such as different movie genre categories, highest rated actors and movies, the highest-rated genre of each year with actors and directors, female actors acting in drama genre movies over period 2000-2004, etc.

Finally, data visualization was achieved using Tableau for the different business scenarios such as the top 10 directors, count of the male and the female actors in different movies, top-rated action movies of each year, and motion chart representation of the top 3 genres over period 1995-2004.

PART 2: DATA MINING

1: CHOOSING AND EXPLORING DATA

1.1 DATA SOURCE

The source of the dataset is Kaggle. The objective of this dataset is to create a model for salary prediction of a person based on various attributes such as **SalaryKey, year, gender, age, country, population, occupation, college degree, height, hair color, using glasses**, and **salary**. Once a model is created using a suitable data mining algorithm then the salary of any new person can be predicted by the model if he or she has a similar parameter as per the dataset. The reason for selecting this dataset was that it has a good amount of data of about 1 lakh 11 thousand rows. Hence the model will be trained better because of huge data. There are fewer chances for biasing of the data model towards prediction as it has a good amount of data to learn.

1.2 DATA EXPLORATION

The code for this section can be found in Appendix D. The salary prediction dataset has the following rows and columns (*10 minutes to pandas — pandas 1.0.3 documentation, 2020*)

```
In [5]: #number of rows and columns in the dataset
        print(input_df.shape)

        (111993, 12)
```

Figure 16: Rows and columns information

Correlation of different attributes with the salary can be seen below

```
# checking coorelation
corr = input_df_Outlier_Removed.corr()
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
input_df_Outlier_Removed.corr()
```

	SalaryKey	Year	Age	Population	Using Glasses	Height	Salary
SalaryKey	1.000000	-0.001709	0.000552	-0.003722	0.000958	-0.001455	0.002447
Year	-0.001709	1.000000	-0.002137	0.003650	-0.002256	0.002500	0.166064
Age	0.000552	-0.002137	1.000000	0.004841	-0.000500	-0.001869	0.186496
Population	-0.003722	0.003650	0.004841	1.000000	0.000675	-0.002164	0.014875
Using Glasses	0.000958	-0.002256	-0.000500	0.000675	1.000000	0.007068	0.005774
Height	-0.001455	0.002500	-0.001869	-0.002164	0.007068	1.000000	0.072200
Salary	0.002447	0.166064	0.186496	0.014875	0.005774	0.072200	1.000000

Figure 17: Checking correlations of attributes with salary

It can be observed that **year** and **age** attributes has the highest correlation with salary whereas the population and using glasses have the lowest(*What is a Heat Map, How to Generate One, Example and Case Studies, 2020*). It can also be seen in the below heatmap in figure 19.

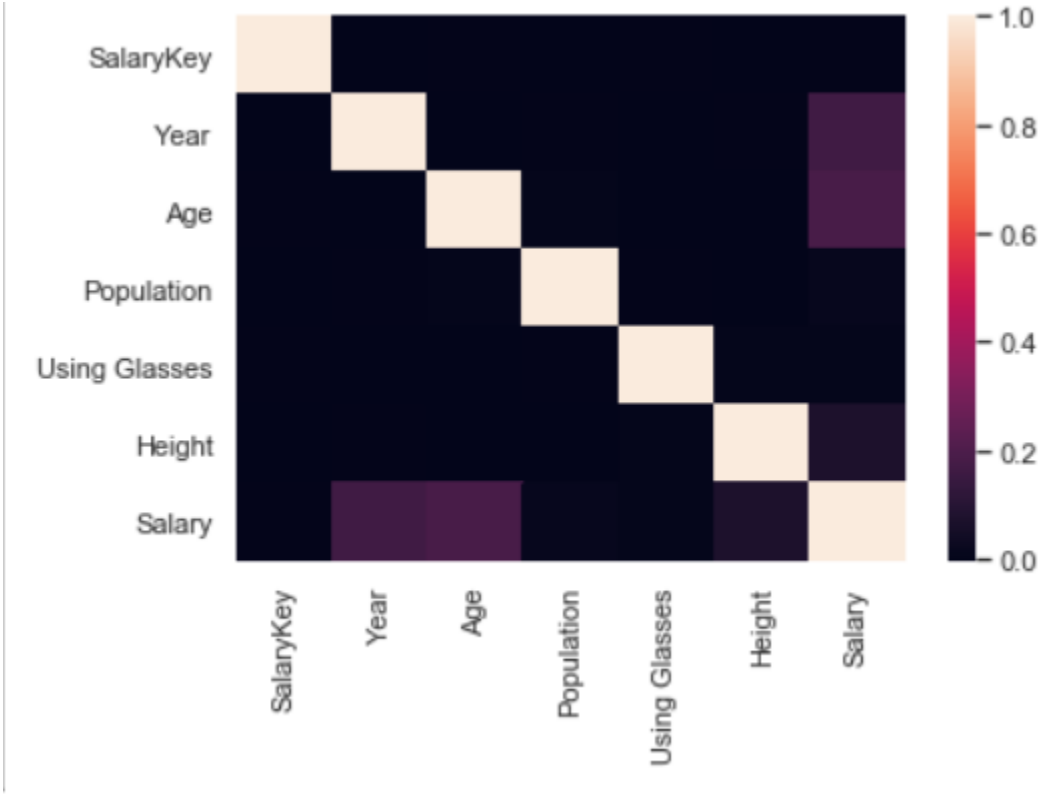


Figure 18: Heatmap of correlations of attributes with salary

A function is also created for plotting regression plot between any two variables as shown below(*Matplotlib: Python plotting — Matplotlib 3.2.1 documentation, 2020*)

```
# function for plotting regression function between any two variables
def regressionplt(width, height, dataset, xcol, ycol):
    plt.figure(figsize=(width, height))
    sns.regplot(x = xcol, y = ycol, data = dataset, line_kws={'color':'red'})
    plt.ylim(0,)
```

Figure 19: Regression plot function

Using the above function regression plot between salary and year can be observed.

```
regressionplt(8, 6, input_df, 'Year', 'Salary')
```

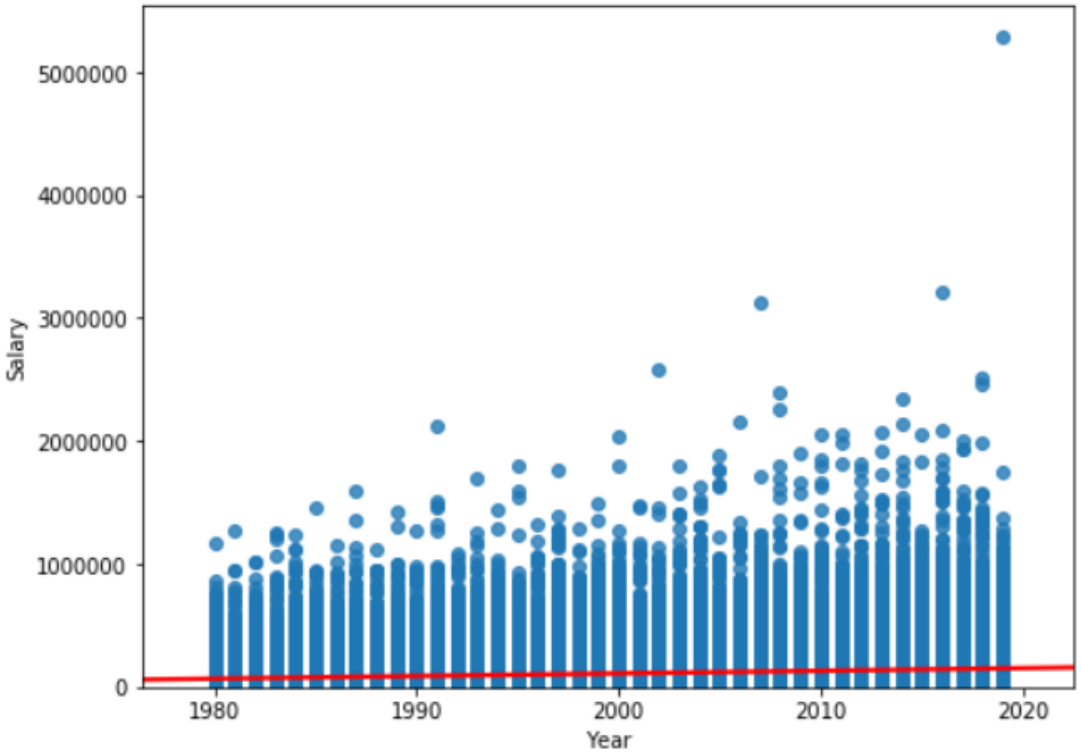


Figure 20: Regression plot of Salary vs Year

Regression plot between age and salary can also be shown below

```
regressionplt(8, 6, input_df, 'Age', 'Salary')
```

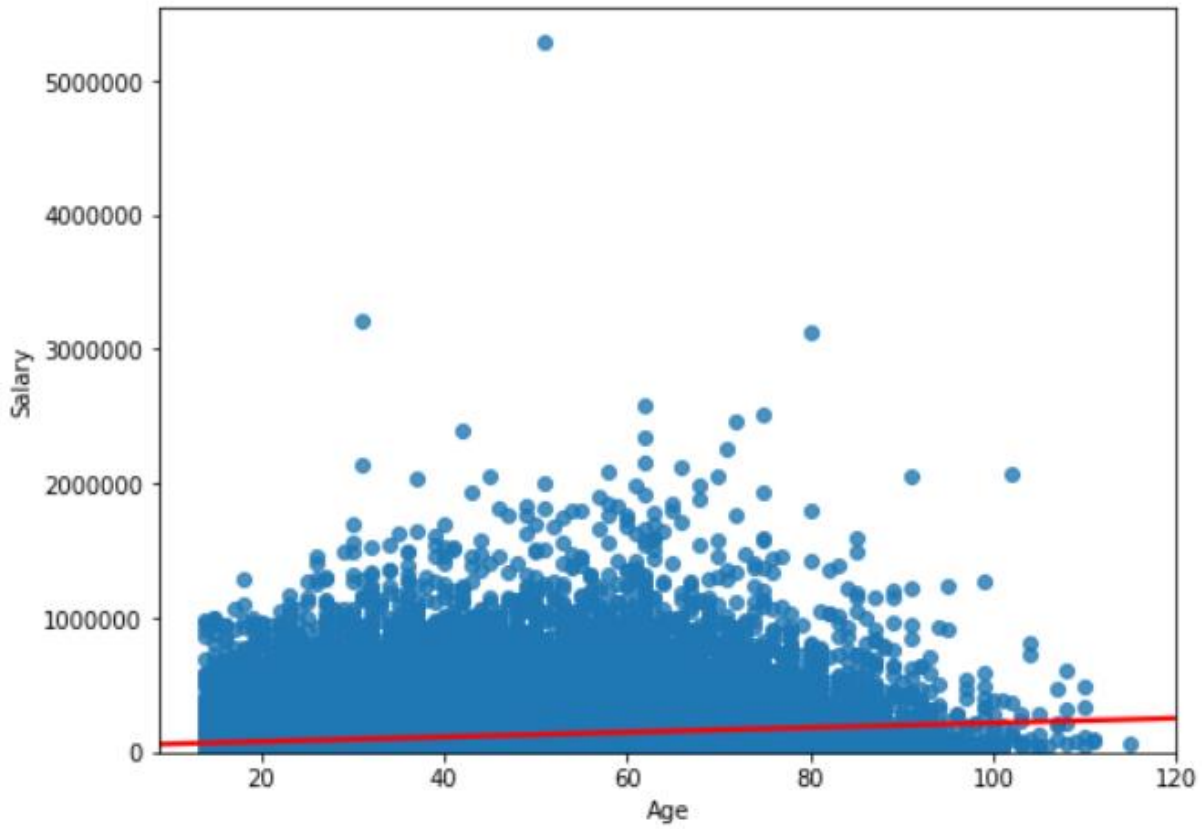


Figure 21: Regression plot of Salary vs Age

As age and year have the highest correlation with salary as compared to other attributes. Outliers can also be seen by plotting low correlation attributes such as height with salary as shown below.

```
regressionplt(8, 6, input_df, 'Height', 'Salary')
```

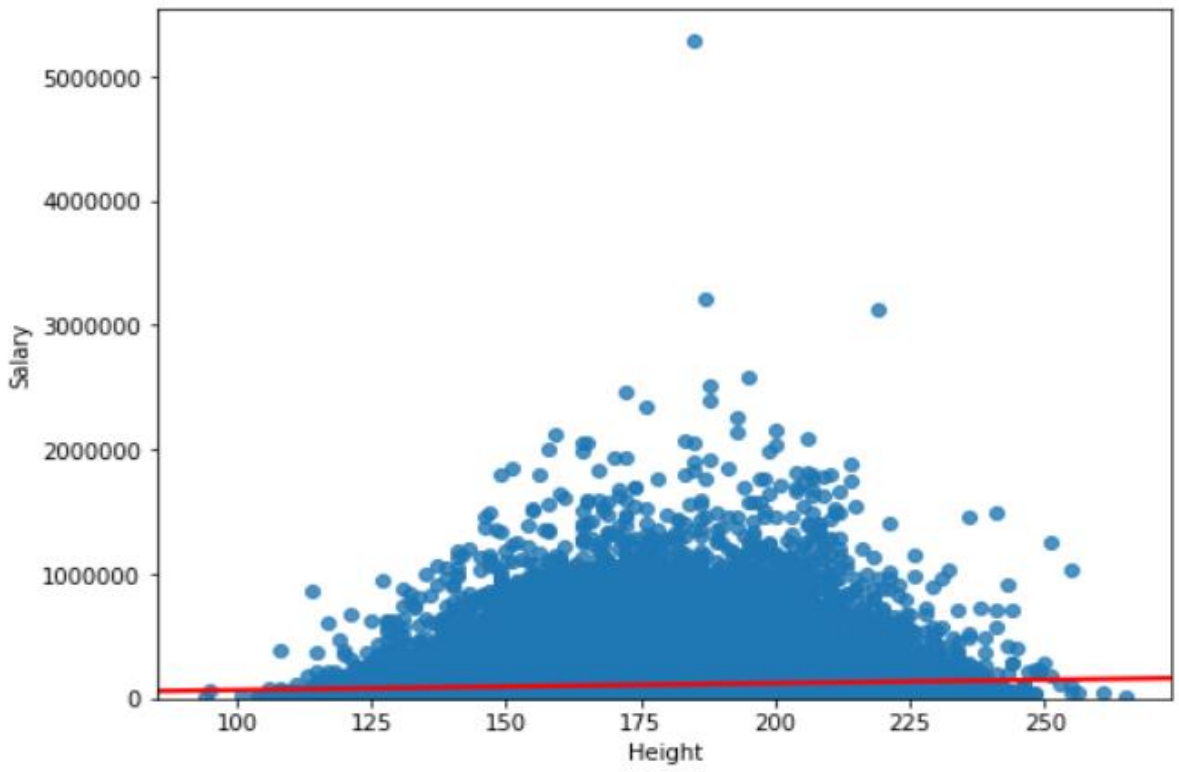


Figure 22: Regression plot of Salary vs Height

2: ANALYSING, CLEANING AND PREPARING DATA

Correlations between different attributes were explained in the previous section. The code for this section can be found in Appendix D. For data cleaning checking null values in each attribute is shown below

```
In [8]: # to check for nulls in each attribute
        missing_instance = input_df.isnull().sum()
        missing_instance
```

```
Out[8]: SalaryKey            0
        Year               441
        Gender            7432
        Age                494
        Country              0
        Population           0
        Occupation         322
        College Degree    7370
        Using Glasses        0
        Color of Hairs    7242
        Height               0
        Salary               0
        dtype: int64
```

Figure 23: Checking nulls

General statistics of the input data for finding parameters such as min, max, mean, median, count for better understanding of data can be found below

```
# general statistics of input data
input_df.describe(include=['object', 'int64', 'float64', 'bool'])
```

	SalaryKey	Year	Gender	Age	Country	Population	Occupation	College Degree	Using Glasses	Color of Hairs	Height	S
count	111993.000000	111552.000000	104561	111499.000000	111993	1.119930e+05	111671	104623	111993.000000	104751	111993.000000	1.11993
unique	NaN	NaN	5	NaN	160	NaN	1340	5	NaN	6	NaN	
top	NaN	NaN	male	NaN	Switzerland	NaN	pipefitter	Bachelor	NaN	Black	NaN	
freq	NaN	NaN	42758	NaN	2327	NaN	261	43172	NaN	42778	NaN	
mean	55997.000000	1999.421274	NaN	37.345304	NaN	8.388538e+05	NaN	NaN	0.500531	NaN	175.220192	1.09213
std	32329.738686	11.576382	NaN	16.036694	NaN	2.196879e+06	NaN	NaN	0.500002	NaN	19.913889	1.49802
min	1.000000	1980.000000	NaN	14.000000	NaN	7.700000e+01	NaN	NaN	0.000000	NaN	94.000000	-5.69690
25%	27999.000000	1989.000000	NaN	24.000000	NaN	7.273400e+04	NaN	NaN	0.000000	NaN	160.000000	3.07716
50%	55997.000000	1999.000000	NaN	35.000000	NaN	5.060920e+05	NaN	NaN	1.000000	NaN	174.000000	5.73391
75%	83995.000000	2009.000000	NaN	48.000000	NaN	1.184501e+06	NaN	NaN	1.000000	NaN	190.000000	1.26093
max	111993.000000	2019.000000	NaN	115.000000	NaN	4.999251e+07	NaN	NaN	1.000000	NaN	265.000000	5.28525

Figure 24: Statistical analysis of the dataset

The next step was finding categorical features for identifying the distribution of values. The categorical distribution feature is used to encode the value of non-numeric attributes for assigning them a numeric label so that they can be identified and plotted. This is done for non-numeric attributes fields as shown below(*Tutorial) Handling Categorical Data in Python, 2020*)

```
#to identify distribution of values in categorical features
print(input_df.groupby('Gender').size().reset_index(name='counts'))
print(input_df.groupby('College Degree').size().reset_index(name='counts'))
print(input_df.groupby('Color of Hairs').size().reset_index(name='counts'))
print(input_df.groupby('Occupation').size().reset_index(name='counts'))
print(input_df.groupby('Country').size().reset_index(name='counts'))
```

```
    Gender  counts
0        0     724
1   female   27170
2     male   42758
3    other   27060
4  unknown    6849
  College Degree  counts
0              0     697
1       Bachelor   43172
2         Master   26941
3             No   26992
4            PhD    6821
  Color of Hairs  counts
0              0      29
1          Black   42778
```

Figure 25: Categorical features of non-numeric fields

The residue plot of age vs salary can also be seen below in fig. 26

```
# residue plot for checking data spread
width = 12
height = 10
plt.figure(figsize=(width, height))
sns.residplot(input_df['Age'], input_df['Salary'])
#plt.show()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x80991e9320>
```

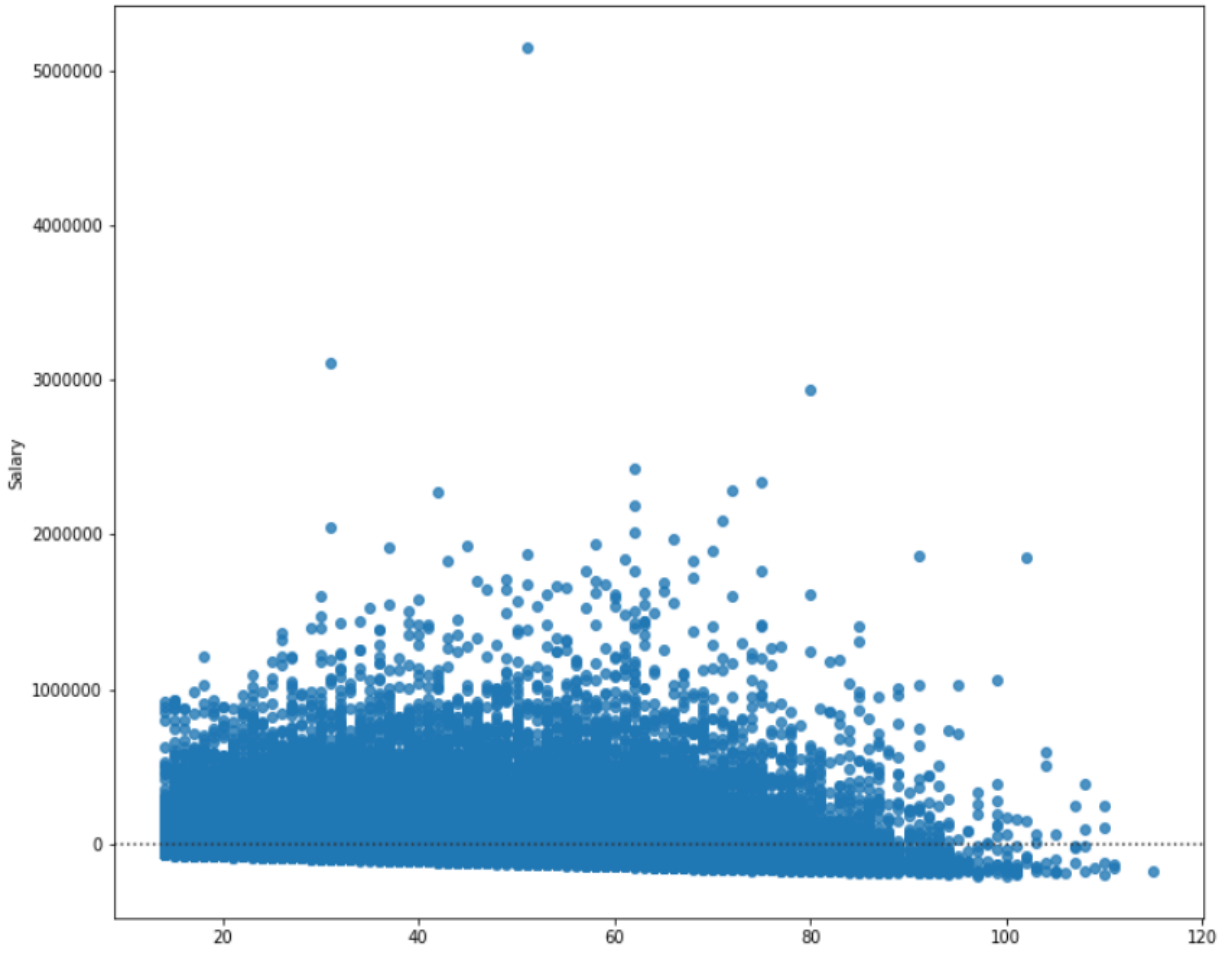


Figure 26: Residue plot of Age vs Salary

The next step was the removal of outliers such as salary cannot be less than 0 and salary to be less than 3000000 as shown below.

```
#removal of outliers[salary cannot be negative]
input_df_Outlier_Removed = input_df[input_df['Salary'] > 0]
#input_df_Outlier_Removed = input_df_Outlier_Removed[input_df_Outlier_Removed['Age'] < 90]
input_df_Outlier_Removed = input_df_Outlier_Removed[input_df_Outlier_Removed['Salary'] < 3000000]
```

Figure 27: Removal of outliers

After the removal of outliers, the next step was determining the correlations of different non-dependent attributes with the dependent attribute salary. This has already been shown in figures 18,20,21 and 22. Once correlations were plotted the next step was data cleaning. The first step in data cleaning was the removal of the population attribute from the dataset as it has a very low correlation with salary as shown below.

```
#ignoring population as it is has low corelation.
selected_training_columns = ['Year',
                             'Age',
                             'Gender',
                             'Country',
                             #'Population',
                             'College Degree',
                             'Using Glasses',
                             'Color of Hairs',
                             'Occupation',
                             'Height',
                             'Salary'
                             ]
```

```
dataset = input_df_Outlier_Removed[selected_training_columns]
```

Figure 28: Removal of population attribute

For the cleaning of gender attribute, all the gender fields apart from male and female which had values like 0, empty, other, unknown have been categorized and they are set to Unknown. The same operation is performed for the color of the hair attribute.

```
# Inputing Gender column
dataset['Gender'].replace('0', 'Unknown', inplace=True)
dataset['Gender'].replace('other', 'Unknown', inplace=True)
dataset['Gender'].replace('unknown', 'Unknown', inplace=True)
dataset['Gender'].replace(np.nan, 'Unknown', inplace=True)


dataset['Color of Hairs'].replace('0', 'Unknown', inplace=True)
dataset['Color of Hairs'].replace('Unknown', 'Unknown', inplace=True)
dataset['Color of Hairs'].replace(np.nan, 'Unknown', inplace=True)


print(dataset.groupby('Gender').size().reset_index(name='counts'))
print(dataset.groupby('Color of Hairs').size().reset_index(name='counts'))
```

Figure 29: Cleaning for gender and hair color attribute

The output can be seen below

```
    Gender  counts
0  Unknown   42010
1   female   27078
2     male   42736
  Color of Hairs  counts
0          Black   42717
1          Blond   27207
2          Brown   27164
3            Red    6823
4        Unknown    7913
```

Figure 30: Output after cleaning

As we know from figure 23 that there are 494 null values in the age field. Hence, the next step was removing these null values rows from the age attribute. The same was done for the year field as well as shown below

```
# command to remove 494 nulls value from age
dataset = dataset[dataset['Age'].notna()]
dataset = dataset[dataset['Year'].notna()]
dataset.isnull().sum()
```

```
Year                0
Age                 0
Gender              0
Country             0
College Degree    7284
Using Glasses        0
Color of Hairs       0
Occupation         321
Height               0
Salary               0
dtype: int64
```

Figure 31: Null removed from age and year

Note:- As age has the highest correlation with salary attribute, so instead of removing null fields the median of age attribute is also calculated and value is used in the models. The steps for this are explained in detail in the linear regression topic.

The Unknown value was set in place of empty fields in the Occupation and College degree field. And after this, no null field was left in the dataset as shown below

```
In [11]: dataset['Occupation'].replace(np.nan, 'Unknown', inplace=True)
         dataset['College Degree'].replace(np.nan, 'Unknown', inplace=True)

         #check for null
         dataset.isnull().sum()
```

```
Out[11]: Year              0
         Age               0
         Gender            0
         Country           0
         College Degree    0
         Using Glasses     0
         Color of Hairs    0
         Occupation        0
         Height            0
         Salary            0
         dtype: int64
```

Figure 32: Unknown set for occupation and college degree

For data preparation dependent and independent variables were separated as shown below

```
# seperating dependant and independent parts of the dataset
y = dataset['Salary']
dataset.drop('Salary', axis=1, inplace=True)
x_dataset = dataset.copy()
```

Figure 33: Dependent and independent variables split

The next step was encoding of categorical features using One Hot Encoding technique(*Tutorial) Handling Categorical Data in Python, 2020*)

```
# Encoding categorical features
# Ref: https://www.datacamp.com/community/tutorials/categorical-data
# Ref: http://thedataist.com/when-categorical-data-goes-wrong/

from feature_engine.categorical_encoders import OneHotCategoricalEncoder

columns_to_encode = ['Gender', 'Country', 'Color of Hairs', 'College Degree', 'Occupation']

encoder = OneHotCategoricalEncoder(top_categories=None,
variables=columns_to_encode,  # we can select which variables to encode
drop_last=True)

encoder.fit(x_dataset)
encoded_dataet = encoder.transform(x_dataset)
```

Figure 34: One Hot encoding on categorical features

Finally, the dataset was split into training and test sets in 80-20 proportion.

```
# Splitting data into training and testing subset
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(encoded_dataet, y, test_size=0.2, random_state=0)
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(89459, 1512)
(89459,)
(22365, 1512)
(22365,)
```

Figure 35: Training and testing split of the dataset

3: MODEL BUILDING, TESTING, AND EVALUATION

The code for this section can be found in Appendix D. The models used in this project are Linear Regression, Random Forest, and Cat Boost. Let us see them one by one.

3.1 LINEAR REGRESSION MODEL

For linear regression, **sklearn.linear_mode** was imported and the model was run using **model.predict** and model performance wad calculated as shown below(*Linear Regression, 2020*)

```
# Linear regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
# Running pediction
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
# Evaluating Model performance
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 42887.0648449759
Mean Squared Error: 6563257811.378272
Root Mean Squared Error: 81013.93590844894
```

Figure 36: Linear Regression metrics

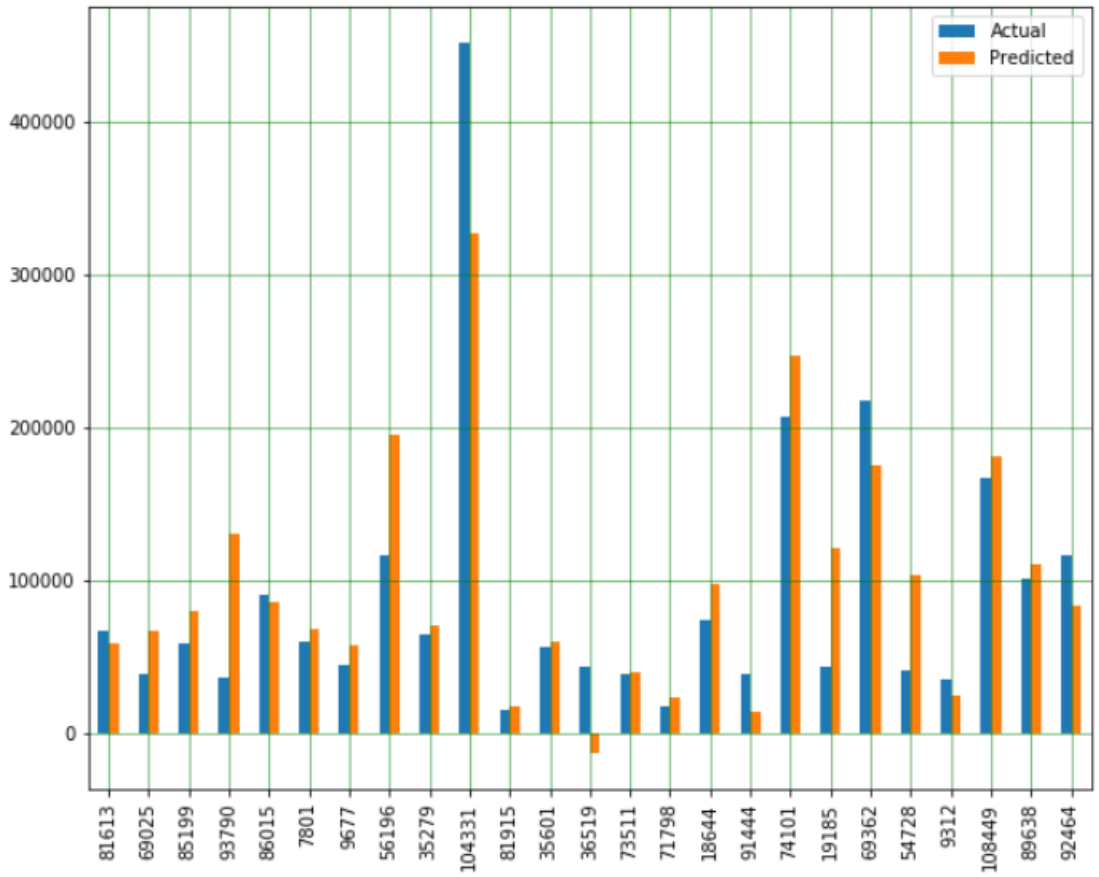
The graphical representation of the model is shown below(*scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation, 2020*)

```
#graphical representation of the model performance
from sklearn import metrics
def plot_prediction(y_test, y_pred):
    df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    df1 = df.head(25)
    df1.plot(kind='bar', figsize=(10, 8))
    plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
    plt.show()
    print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
    print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
    print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
    # Calculate mean absolute percentage error (MAPE)
    errors = abs(y_pred - y_test)
    mape = 100 * (errors / y_test)
    # Calculate and display accuracy
    accuracy = 100 - np.mean(mape)
    print('Accuracy:', round(accuracy, 2), '%.')
```

Figure 37: Graphical representation of linear regression

The plot prediction method was used and parameters like Mean Absolute Error, Mean Squared Error, Root Mean Square Error and Accuracy was calculated as shown below

```
plot_prediction(y_test, y_pred)
```




```
Mean Absolute Error: 42887.0648449759
Mean Squared Error: 6563257811.378272
Root Mean Squared Error: 81013.93590844894
Accuracy: 9.52 %.
```

Figure 38: Accuracy metrics of the linear model(1st attempt)

It can be seen from the 1st try that RMSE and accuracy values of the current model are not good. Hence, the age attribute for which rows with null fields were included. Instead, the median value of the age was inserted into those null fields.

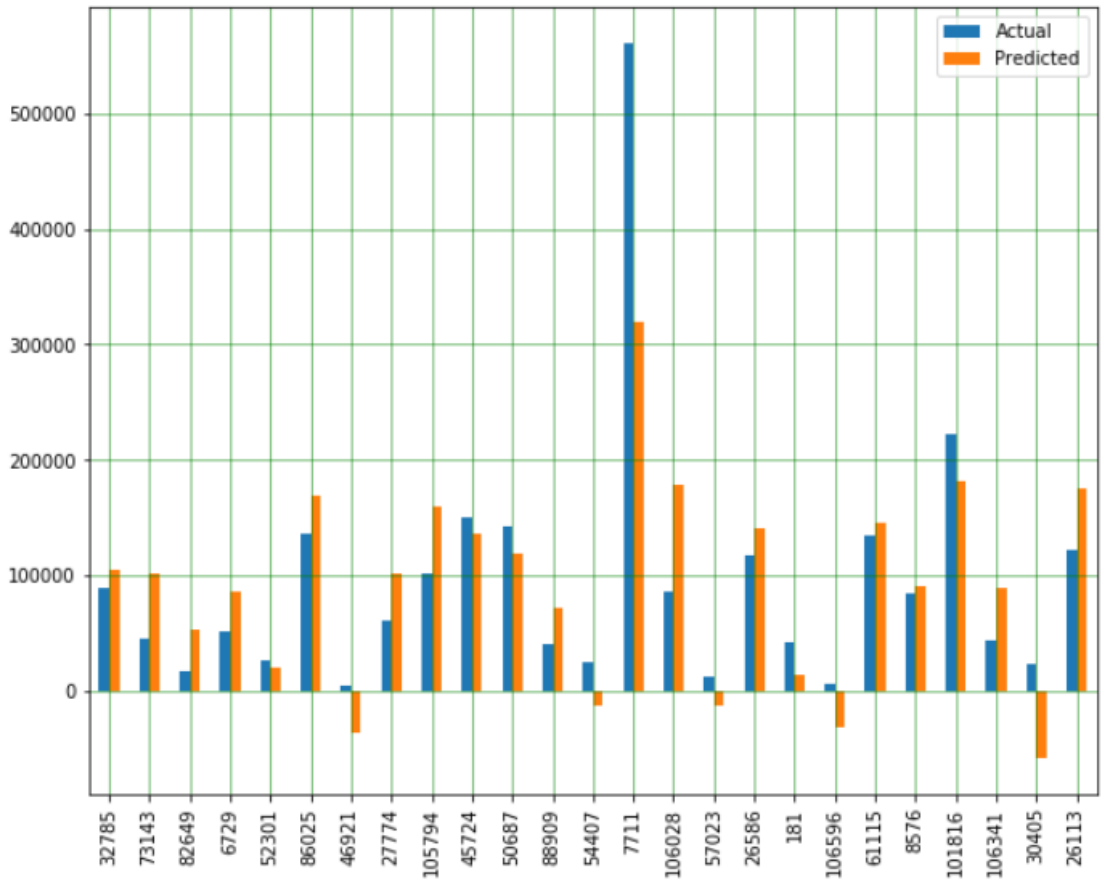
```
# Alternate data imputation strategy
age_median = dataset['Age'].median()
dataset['Age'].replace(np.nan, age_median, inplace=True)
dataset['Age'] = (dataset['Age'] * dataset['Age']) ** (0.5)

year_median = dataset['Year'].median()
dataset['Year'].replace(np.nan, year_median, inplace=True)
```

Figure 39: Median value used for age attribute

For the 2nd try once again the model was trained and run

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
plot_prediction(y_test, y_pred)
```




```
Mean Absolute Error: 42473.668677019195
Mean Squared Error: 6148159676.557089
Root Mean Squared Error: 78410.20135516225
Accuracy: -13.02 %.
```

Figure 40: Accuracy metrics of the linear model(2nd attempt)

The accuracy is less than the first try because there are many skewed values in the model as per the graph. However, it can be seen that MAE, MSE, and RMSE values are less as compared to 1st try. Now for the 3rd try, the next step was checking for the skewed values. The skewed value plot is shown below

```
# Plotting density curve to see if the values are skewed
df = pd.DataFrame({'Salary': y})
sns.distplot(df['Salary'], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xe72ca2e940>
```

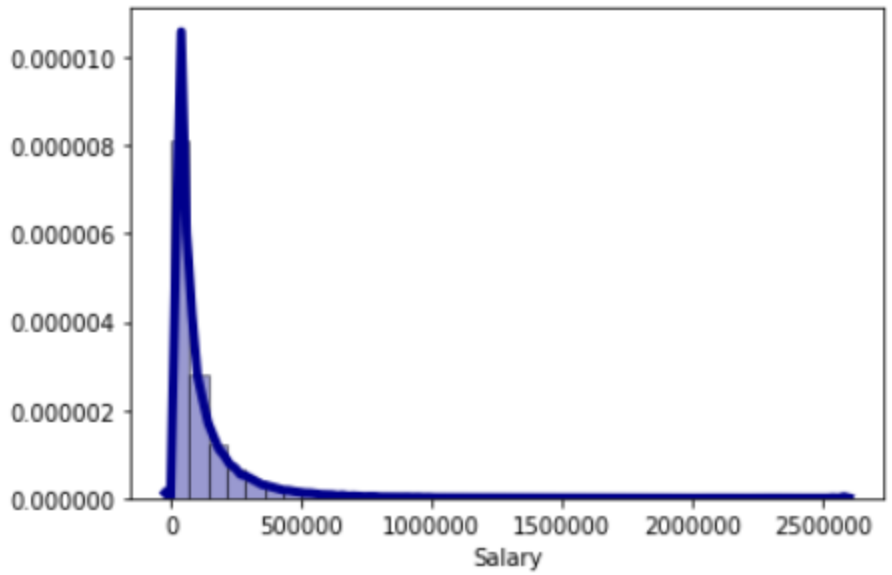


Figure 41: Skewed value check

To proper handle, skewed values log scale handling is done for salary attribute and for plotting it an inverse log of the predicted value is taken as shown below(*Day 8: Data transformation — Skewness, normalization and much more, 2020*)

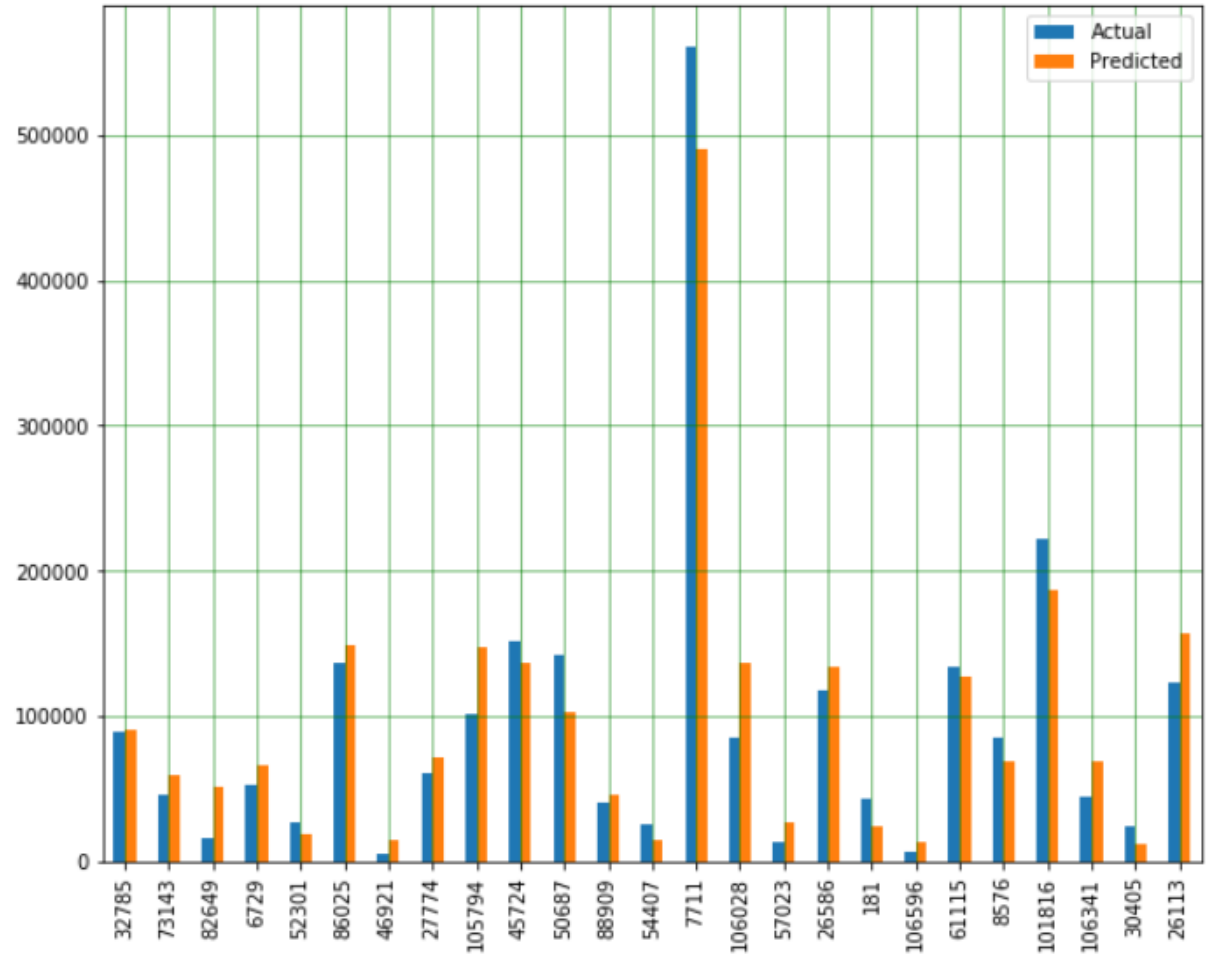
```
# Map y to log scale handle skewed value in training
y_train_log = np.log(y_train)
```

```
# retraining the regressor model
model.fit(X_train, y_train_log)
y_pred = np.exp(model.predict(X_test)) #Taking inverse log of the predicted values
```

Figure 42: Log scaling and inverse log of salary

Once again the plot was predicted in the 3rd try with much better results as shown below

```
plot_prediction(y_test, y_pred)
```




```
Mean Absolute Error: 26026.16864507491
Mean Squared Error: 3058501958.6966205
Root Mean Squared Error: 55303.72463674233
Accuracy: 55.73 %.
```

Figure 43: Accuracy metrics of the linear model(3rd attempt)

Hence much better results can be seen from the linear regression model in the 3rd try. MSE and RMSE are much better for this attempt. This could have been improved further by removing more outlier values from the attributes but then the model's prediction performance would have been compromised and results will be biased. The reason for this is there will be more data points of the similar value range and if in future any field comes with the outliers that are not present in the dataset then the model won't be able to predict them accurately.

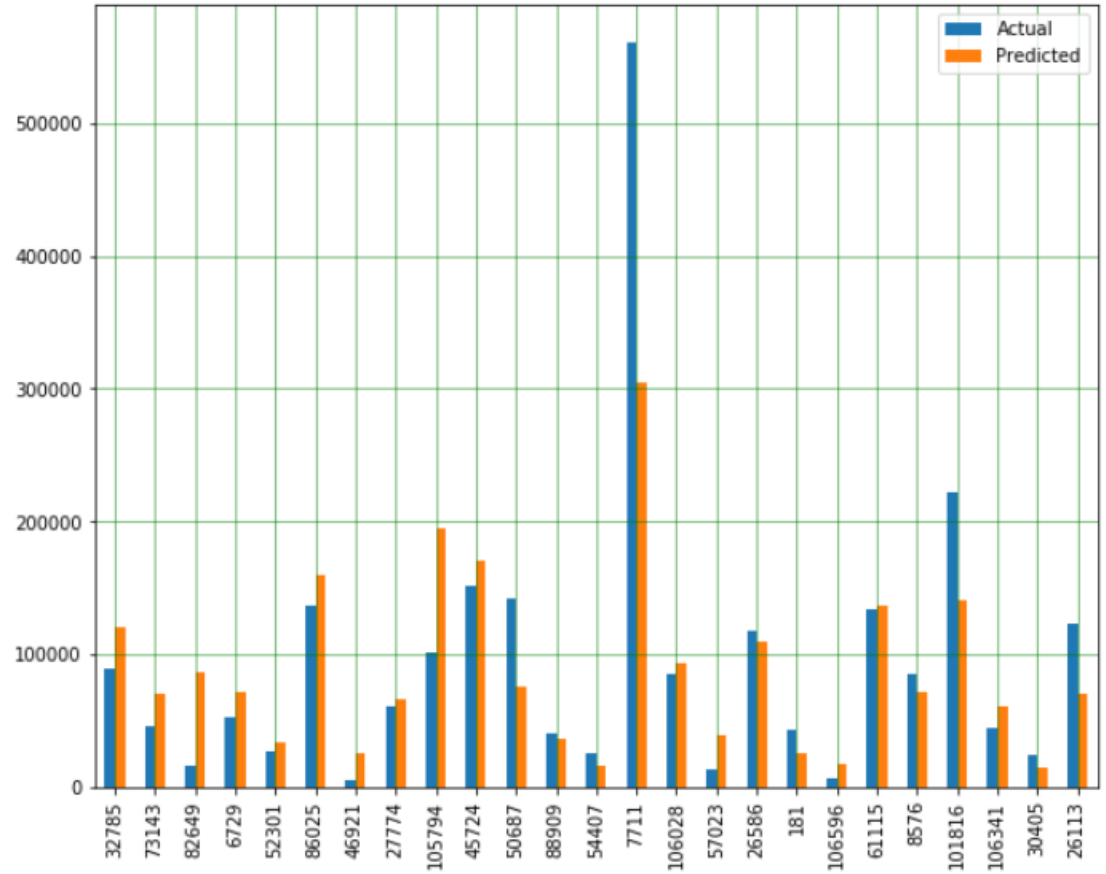
3.2 RANDOM FOREST REGRESSOR

This is the 2nd data mining algorithm used for modeling. The first step was importing the below library.

from **sklearn.ensemble** import **RandomForestRegressor**

The data transformation for this is already done in the linear regression model i.e median value of age is used in place of null fields [as shown in figure 38]. After this using **RandomForestRegressor** for creating a prediction model and plotting the result as shown below(*Random Forest and its Implementation, 2020*)

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators = 5, random_state = 42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot_prediction(y_test, y_pred)
```

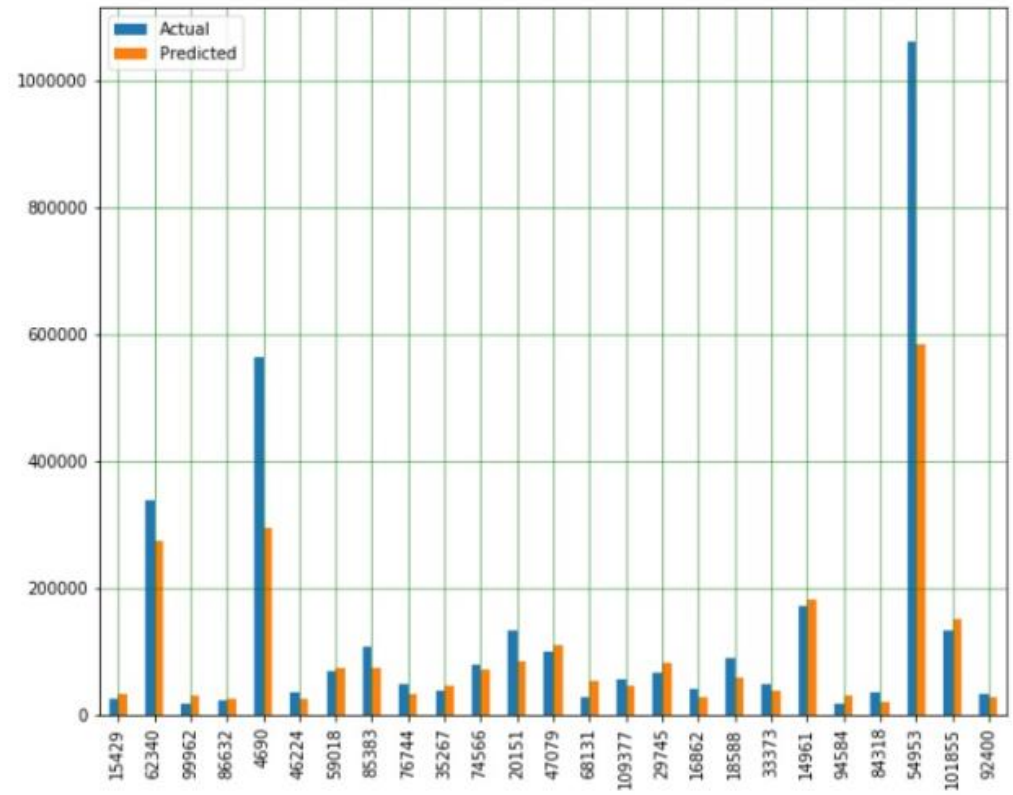



```
Mean Absolute Error: 36754.26070846787
Mean Squared Error: 6569201671.137251
Root Mean Squared Error: 81050.61178755686
Accuracy: 34.94 %.
```

Figure 44: Accuracy metrics of the random forest model(1st attempt)

It can be observed that the metrics values are not that bad as per the dataset. For fine-tuning the Random Forest, the **n_estimators** value can be changed. The n_estimators determines no. of decision trees to be used in the process. When the n_estimators value was set to 10 the results can be seen below(*Hyperparameter Tuning the Random Forest in Python, 2020*).

```
In [92]: # TODO Try other encoding methods and complex models
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators = 10, random_state = 42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot_prediction(y_test, y_pred)
```




```
Mean Absolute Error: 36916.786320033716
Mean Squared Error: 6824447655.439781
Root Mean Squared Error: 82610.21520029943
Accuracy: 37.26 %.
```

Figure 45: Accuracy metrics of the random forest model(2nd attempt)

The n_ estimators value was also increased to 50% but the results were not that good because the model was overfitting.

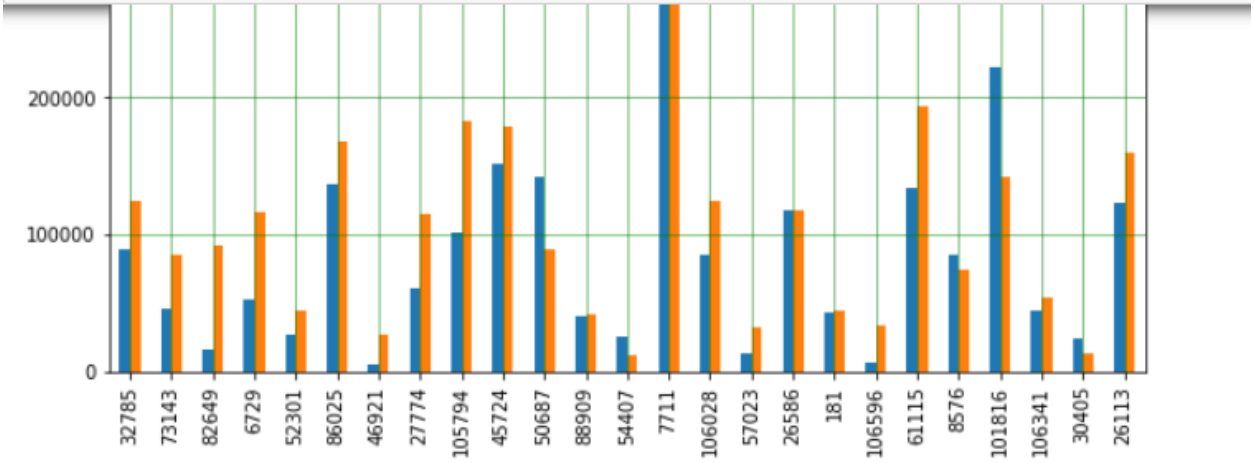
3.3 CAT BOOST MODEL

The next algorithm used was cat boost. The first step was adding the library for it using below command

from **catboost** import **CatBoostRegressor**

The data transformation for this is already done in the linear regression model i.e median value of age is used in place of null fields for model transformation [as shown in figure 38]. Then the model is created using the CatBoostRegressor library. After predicting the model with test set and plotting the result were as shown

```
from catboost import CatBoostRegressor
model = CatBoostRegressor()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot_prediction(y_test, y_pred)
```

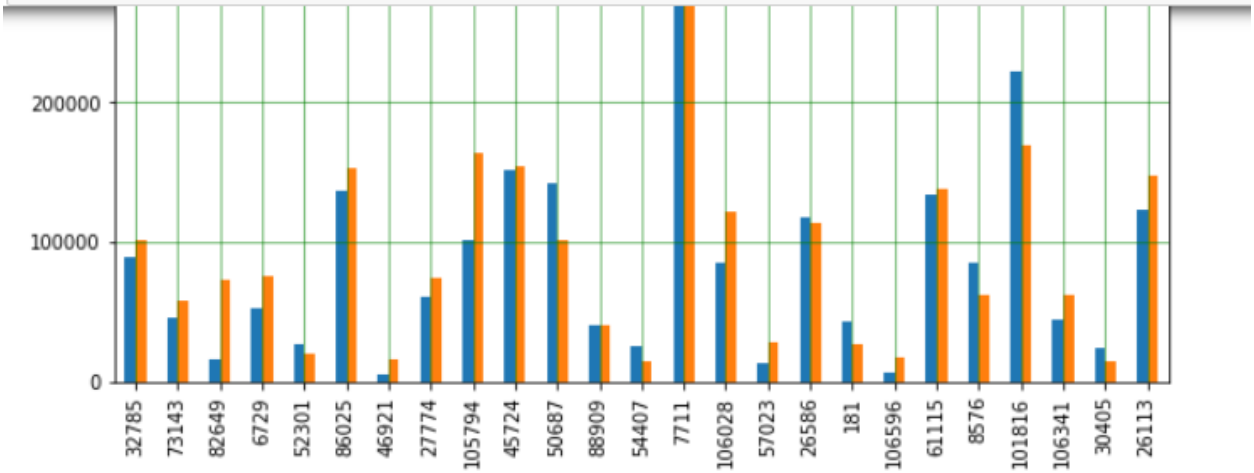



```
Mean Absolute Error: 36488.30160903748
Mean Squared Error: 5527423002.764146
Root Mean Squared Error: 74346.64083039762
Accuracy: 32.51 %.
```

Figure 46: Accuracy metrics of the cat boost model(1st attempt)

For the 2nd try log scaling and the inverse of a log is taken for model transformation as shown in figure 41 in the linear regression model. Apart from this parameter such as **learning_rate=0.03** is set for slow and better learning of model. A parameter **depth=6** is used for deep analysis by model and finally, **iteration=10000** is used(*Usage examples - CatBoost. Documentation, 2020*). The result of 2nd try can be seen below.

```
model = CatBoostRegressor(iterations=10000,
                          learning_rate=0.03,
                          depth=6)
y_train_log = np.log(y_train)
model.fit(X_train, y_train_log)
y_pred = np.exp(model.predict(X_test))
plot_prediction(y_test, y_pred)
```




```
Mean Absolute Error: 28164.715334946268
Mean Squared Error: 4187554377.991738
Root Mean Squared Error: 64711.31568737989
Accuracy: 53.97 %.
```

Figure 47: Accuracy metrics of the cat boost model(2nd attempt)

It can be observed that the performance of the model has increased after the changes.

3.4 EVALUATION OF LINEAR REGRESSION, RANDOM FOREST, AND CAT BOOST

It can be observed that among the 3 models, linear regression performs better as MAE, MSE, and RMSE values are better in this model as compared to random forest and cat boost algorithms.

It can be seen in the table below

	Linear Regression	**Random Forest**	**Cat boost**
Mean Absolute Error	26026.16864507491	36916.786320033717	28164.715334946268
Mean Squared Error	3058501958.6966205	6824447655.439781	4187554377.991738
Root Mean Squared Error	55303.72463674233	82610.21520029943	64711.31568737989

Accuracy	55.73 %	37.26%	53.97 %

Table 1 – Comparison of Data Mining algorithm

Although if target encoding and grid searching would be used for cat boosting. The results could have been better but the grid search is very time-consuming.

4. CONCLUSION

It can be concluded that as per the diversity of the dataset, the results given by all the 3 algorithms Linear Regression, Random Forest, and Cat Boost are good. The data cleaning and data transformation are done according to the suitability of the models. For linear regression, 3 different approaches were used such as removing null fields of age attribute, using the median value of the age, and log scaling and inverse log for skewed values. For Random Forest, 2 different approaches were used by setting up values of n_estimators = 5 and then n_estimators = 10. Finally, in Cat boost as well two different approaches are used for better predictions.

In terms of Linear regression, this could have been improved further by removing more outlier values from the attributes but then the model's prediction performance would have been compromised and results will be biased. The reason for this is there will be more data points of the similar value range and if in future any field comes with the outliers that are not present in the dataset then the model won't be able to predict them accurately. However, if we consider the large data set we can conclude that the model prediction is realistic.

In terms of Cat Boosting if target encoding and grid searching would be used the results could have been better but the grid search is very time-consuming.

REFERENCING

DataCamp Community. 2020. *(Tutorial) Handling Categorical Data In Python*. [online] Available at: <https://www.datacamp.com/community/tutorials/categorical-data> [Accessed 24 April 2020].

Pandas.pydata.org. 2020. *10 Minutes To Pandas — Pandas 1.0.3 Documentation*. [online] Available at: <https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html> [Accessed 24 April 2020].

Medium. 2020. *Day 8: Data Transformation — Skewness, Normalization And Much More*. [online] Available at: <https://medium.com/@TheDataGyan/day-8-data-transformation-skewness-normalization-and-much-more-4c144d370e55> [Accessed 24 April 2020].

Medium. 2020. *Hyperparameter Tuning The Random Forest In Python*. [online] Available at: <https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74> [Accessed 24 April 2020].

Stat.yale.edu. 2020. *Linear Regression*. [online] Available at: <http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm> [Accessed 24 April 2020].

Matplotlib.org. 2020. *Matplotlib: Python Plotting — Matplotlib 3.2.1 Documentation*. [online] Available at: <https://matplotlib.org/> [Accessed 24 April 2020].

Medium. 2020. *Random Forest And Its Implementation*. [online] Available at: <https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f> [Accessed 24 April 2020].

Scikit-learn.org. 2020. *Scikit-Learn: Machine Learning In Python — Scikit-Learn 0.16.1 Documentation*. [online] Available at: <https://scikit-learn.org/> [Accessed 24 April 2020].

Catboost.ai. 2020. *Usage Examples - Catboost. Documentation*. [online] Available at: <https://catboost.ai/docs/concepts/python-usages-examples.html> [Accessed 24 April 2020].

The Daily Egg. 2020. *What Is A Heat Map, How To Generate One, Example And Case Studies*. [online] Available at: <https://www.crazyegg.com/blog/understanding-using-heatmaps-studies/> [Accessed 24 April 2020].

The Dataist. 2020. *When Categorical Data Goes Wrong*. [online] Available at: <http://thedataist.com/when-categorical-data-goes-wrong/> [Accessed 24 April 2020].

APPENDIX A

(Data warehouse create table queries)

Create **Director_Dim** table query is given below.

```
CREATE TABLE Director_Dim (
  DirectorKey int PRIMARY KEY NOT NULL IDENTITY,
  director_id int,
  DirectedMoviesCount int,
  DirectorName nvarchar(100)
  )
```

Create **Actor_Dim** table query is given below.

```
CREATE TABLE Actor_Dim (
  ActorKey int PRIMARY KEY NOT NULL IDENTITY,
  actor_id int,
  ActedMoviesCount int,
  ActorName nvarchar(100),
  gender char(1))
```

Create **Movie_Dim** table query is given below.

```
CREATE TABLE Movie_Dim (
  MovieKey int PRIMARY KEY NOT NULL IDENTITY,
  movie_id int,
  MovieName nvarchar(255) DEFAULT NULL,
  year int DEFAULT NULL,
  Rating float DEFAULT NULL,
  genre nvarchar(255) NOT NULL)
```

Create **TopRatingDirectorActor_Fact** table query is given below.

```
CREATE TABLE TopRatingDirectorActor_Fact (
 DirectorKey int,
 MovieKey int,
 ActorKey int,
 num_movies int,
 max_rating float DEFAULT NULL,

CONSTRAINT fk1_eDirectorKey FOREIGN KEY (DirectorKey)

REFERENCES Director_Dim(DirectorKey),

CONSTRAINT fk2_eMovieKey FOREIGN KEY (MovieKey)

REFERENCES Movie_Dim(MovieKey),

CONSTRAINT fk4_eActorKey FOREIGN KEY (ActorKey)

REFERENCES Actor_Dim(ActorKey),

PRIMARY KEY (DirectorKey, MovieKey, ActorKey))
```

APPENDIX B

(Data warehouse tables data populate queries)

Director_Dim data populate query is given below.

```
select md.director_id,count(md.director_id) as movies,concat(d.first_name,'
',d.last_name) as director_name from [IMDB_New].[dbo].[movies_directors] md
left join [IMDB_New].[dbo].[directors] d on md.director_id = d.id where md.movie_id in(

select output.x from (select a.id as x,a.name,a.rank,a.year,gn.genre from
[IMDB_New].[dbo].[movies] a
left join [IMDB_New].[dbo].[movies_genres] gn on a.id = gn.movie_id
 where a.name in(select m.name from [IMDB_New].[dbo].[movies] m left join
[IMDB_New].[dbo].[movies_genres] mg
on m.id = mg.movie_id where m.rank is NOT NULL and mg.movie_id is NOT NULL and mg.genre
is NOT NULL and m.year between 2000 and 2004
group by m.name having COUNT(m.name)=1) and a.rank is NOT NULL and gn.movie_id is NOT
NULL and gn.genre is NOT NULL
and name NOT LIKE '$%' AND name NOT LIKE '?%' AND name NOT LIKE '.%' AND name NOT LIKE
'1%'
AND name NOT LIKE '4%' AND name NOT LIKE '3%' AND name NOT LIKE '6%')output
)
group by md.director_id,concat(d.first_name,' ',d.last_name)
```

Actor_Dim data populate query is given below.

```
select r.actor_id,count(r.actor_id) as movies,concat(ac.first_name,' ',ac.last_name) as
actor_name from [IMDB_New].[dbo].[roles] r
left join [IMDB_New].[dbo].[actors] ac on r.actor_id = ac.id where r.movie_id in(
select output.x from (select a.id as x,a.name,a.rank,a.year,gn.genre from
[IMDB_New].[dbo].[movies] a
left join [IMDB_New].[dbo].[movies_genres] gn on a.id = gn.movie_id
 where a.name in(select m.name from [IMDB_New].[dbo].[movies] m left join
[IMDB_New].[dbo].[movies_genres] mg
on m.id = mg.movie_id where m.rank is NOT NULL and mg.movie_id is NOT NULL and mg.genre
is NOT NULL and m.year between 2000 and 2004
group by m.name having COUNT(m.name)=1) and a.rank is NOT NULL and gn.movie_id is NOT
NULL and gn.genre is NOT NULL
and name NOT LIKE '$%' AND name NOT LIKE '?%' AND name NOT LIKE '.%' AND name NOT LIKE
'1%'
AND name NOT LIKE '4%' AND name NOT LIKE '3%' AND name NOT LIKE '6%')output
) group by r.actor_id,concat(ac.first_name,' ',ac.last_name),ac.gender
```

Movie_Dim data populate query is given below.

```
select a.id ,a.name,a.rank,a.year,gn.genre from [IMDB_New].[dbo].[movies] a left join
[IMDB_New].[dbo].[movies_genres] gn on a.id = gn.movie_id
 where a.name in(select m.name from [IMDB_New].[dbo].[movies] m left join
[IMDB_New].[dbo].[movies_genres] mg
```

```
on m.id = mg.movie_id where m.rank is NOT NULL and mg.movie_id is NOT NULL and mg.genre
is NOT NULL
and m.year between 2000 and 2004
group by m.name having COUNT(m.name)=1) and a.rank is NOT NULL and gn.movie_id is NOT
NULL and gn.genre is NOT NULL
and name NOT LIKE '$%' AND name NOT LIKE '?%' AND name NOT LIKE '.%' AND name NOT LIKE
'1%'
AND name NOT LIKE '4%' AND name NOT LIKE '3%' AND name NOT LIKE '6%'
```

TopRatingDirectorActor_Fact data populate query is given below.

```
select b.DirectorID,b.MovieID,b.ActorID,a.num_movies,a.max_rating from

(select m.year, mg.genre,count(mg.movie_id) as num_movies,max(m.rank) as max_rating
from [IMDB_New].[dbo].[movies_genres] mg inner join [IMDB_New].[dbo].[movies] m
on mg.movie_id=m.id group by m.year,mg.genre)a

inner join

(select d.id as DirectorID,m.id as MovieID,ac.id as
ActorID,m.year,m.rank,mg.genre,concat(d.first_name,' ',d.last_name) as director,
concat(ac.first_name,' ',ac.last_name) as actor, ac.gender as gender
from [IMDB_New].[dbo].[movies] m inner join [IMDB_New].[dbo].[movies_genres] mg
on mg.movie_id = m.id
inner join [IMDB_New].[dbo].[movies_directors] md
on mg.movie_id = md.movie_id
inner join [IMDB_New].[dbo].[directors] d
on md.director_id = d.id
inner join [IMDB_New].[dbo].[roles] ro
on m.id = ro.movie_id
inner join [IMDB_New].[dbo].[actors] ac
on ro.actor_id=ac.id where ro.role IS NOT NULL)b

on a.year = b.year and a.max_rating = b.rank and a.genre = b.genre
where b.rank!='' and b.director is NOT NULL and b.actor is NOT NULL order by a.num_movies
```

APPENDIX C

(SSRS Report Queries)

The query for **genre category of movies** with the **highest rated movie** is given below

```
SELECT genre, COUNT(genre) AS Total_Movies, MAX(Rating) AS Highest_Rating
FROM     [IMDB_DW_NEW].[dbo].[Movie_Dim]
GROUP BY genre
ORDER BY Total_Movies DESC
```

The query for the **top-rated actor** and **movies** list report is given below.

```
SELECT a.ActorName, m.MovieName, MAX(f.max_rating) AS Max_Ratings
FROM     [IMDB_DW_NEW].[dbo].[TopRatingDirectorActor_Fact] AS f INNER JOIN
                  [IMDB_DW_NEW].[dbo].[Actor_Dim] AS a ON f.ActorKey = a.ActorKey INNER
JOIN
                  [IMDB_DW_NEW].[dbo].[Movie_Dim] AS m ON f.MovieKey = m.MovieKey
WHERE  (f.max_rating > 9)
GROUP BY m.MovieName, a.ActorName
ORDER BY Max_Ratings DESC
```

The query for the **Females Acting** in **Drama Genre** over the period 2000-2004 is given below.

```
select a.ActorName,m.year,m.MovieName from [IMDB_DW_NEW].[dbo].[Actor_Dim] a
inner join [IMDB_DW_NEW].[dbo].[TopRatingDirectorActor_Fact] f on a.ActorKey=f.ActorKey
inner join [IMDB_DW_NEW].[dbo].[Movie_Dim] m on f.MovieKey=m.MovieKey
where a.gender='F' and m.year between 2000 and 2004 and m.genre='Drama'
group by m.MovieName,m.year,a.ActorName
```

The query for the **top-rated Movie Actor Director** is given below.

```
  select m.genre,f.num_movies,f.max_rating,m.MovieName,a.ActorName,d.DirectorName from
[IMDB_DW_NEW].[dbo].[TopRartingDirectorActor_Fact] f
  inner join [IMDB_DW_NEW].[dbo].[Director_Dim] d on f.DirectorKey=d.DirectorKey inner
join
  [IMDB_DW_NEW].[dbo].[Actor_Dim] a on f.ActorKey=a.ActorKey inner join
  [IMDB_DW_NEW].[dbo].[Movie_Dim] m on f.MovieKey=m.MovieKey where m.year=2004
  group by d.DirectorName,a.ActorName,m.MovieName,m.genre,f.num_movies,f.max_rating
```

APPENDIX D

(Data Mining Python code)

```
# importing libraries
#!/usr/bin/env python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# function for plotting regression function between any two variables
def regressionplt(width, height, dataset, xcol, ycol):
   plt.figure(figsize=(width, height))
   sns.regplot(x = xcol, y = ycol, data = dataset, line_kws={'color':'red'})
   plt.ylim(0,)

# Reading of dataset into a dataframe
inputFile = r'F:\Download1\Salary_Prediction.csv'
input_df = pd.read_csv(inputFile)

#number of rows and columns in the dataset
print(input_df.shape)

# describe data
input_df.describe

# general statistics of input data
input_df.describe(include=['object', 'int64', 'float64', 'bool'])
```

```
# to check for nulls in each attribute
missing_instance = input_df.isnull().sum()
missing_instance

#to identify distribution of values in categorical features
print(input_df.groupby('Gender').size().reset_index(name='counts'))
print(input_df.groupby('College Degree').size().reset_index(name='counts'))
print(input_df.groupby('Color of Hairs').size().reset_index(name='counts'))
print(input_df.groupby('Occupation').size().reset_index(name='counts'))
print(input_df.groupby('Country').size().reset_index(name='counts'))

# checking for outliers by plotting a regression curve of salary with one low cardinal feature
regressionplt(8, 6, input_df, 'Height', 'Salary')
regressionplt(8, 6, input_df, 'Year', 'Salary')
regressionplt(8, 6, input_df, 'Age', 'Salary')

# residue plot for checking data spread
width = 12
height = 10
plt.figure(figsize=(width, height))
sns.residplot(input_df['Age'], input_df['Salary'])
#plt.show()

#removal of outliers[salary cannot be negative]
input_df_Outlier_Removed = input_df[input_df['Salary'] > 0]
#input_df_Outlier_Removed = input_df_Outlier_Removed[input_df_Outlier_Removed['Age'] < 90]
input_df_Outlier_Removed = input_df_Outlier_Removed[input_df_Outlier_Removed['Salary'] <
3000000]
```

```
regressionplt(8, 6, input_df_Outlier_Removed, 'Height', 'Salary')


# checking coorelation
corr = input_df_Outlier_Removed.corr()
sns.heatmap(corr,
      xticklabels=corr.columns,
      yticklabels=corr.columns)
input_df_Outlier_Removed.corr()


#ignoring population as it is has low corelation.
selected_training_columns = ['Year',
                    'Age',
                    'Gender',
                    'Country',
                    #'Population',
                    'College Degree',
                    'Using Glasses',
                    'Color of Hairs',
                    'Occupation',
                    'Height',
                    'Salary'
                    ]
dataset = input_df_Outlier_Removed[selected_training_columns]


# Data Cleaning
# Inputing Gender column
dataset['Gender'].replace('0', 'Unknown', inplace=True)
dataset['Gender'].replace('other', 'Unknown', inplace=True)
dataset['Gender'].replace('unknown', 'Unknown', inplace=True)
```

```
dataset['Gender'].replace(np.nan, 'Unknown', inplace=True)
dataset['Color of Hairs'].replace('0', 'Unknown', inplace=True)
dataset['Color of Hairs'].replace('Unknown', 'Unknown', inplace=True)
dataset['Color of Hairs'].replace(np.nan, 'Unknown', inplace=True)
print(dataset.groupby('Gender').size().reset_index(name='counts'))
print(dataset.groupby('Color of Hairs').size().reset_index(name='counts'))

# command to remove 494 nulls value from age
dataset = dataset[dataset['Age'].notna()]
dataset = dataset[dataset['Year'].notna()]
dataset.isnull().sum()
dataset['Occupation'].replace(np.nan, 'Unknown', inplace=True)
dataset['College Degree'].replace(np.nan, 'Unknown', inplace=True)

#check for null
dataset.isnull().sum()

# seperating dependant and independent parts of the dataset
y = dataset['Salary']
dataset.drop('Salary', axis=1, inplace=True)
x_dataset = dataset.copy()

# Encoding categorical features
from feature_engine.categorical_encoders import OneHotCategoricalEncoder
columns_to_encode = ['Gender', 'Country', 'Color of Hairs', 'College Degree', 'Occupation']
encoder = OneHotCategoricalEncoder(top_categories=None,
variables=columns_to_encode,  # we can select which variables to encode
drop_last=True)
encoder.fit(x_dataset)
```

```
encoded_dataet = encoder.transform(x_dataset)

# Splitting data into training and testing subset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(encoded_dataet, y, test_size=0.2, random_state=0)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

# Linear regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()

# Running pediction
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluating Model performance
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

#graphical representation of the model performance
from sklearn import metrics
def plot_prediction(y_test, y_pred):
    df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    df1 = df.head(25)
```

```
df1.plot(kind='bar', figsize=(10, 8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

# Calculate mean absolute percentage error (MAPE)
errors = abs(y_pred - y_test)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
plot_prediction(y_test, y_pred)

# Alternate data imputation strategy
age_median = dataset['Age'].median()
dataset['Age'].replace(np.nan, age_median, inplace=True)
dataset['Age'] = (dataset['Age'] * dataset['Age']) ** (0.5)
year_median = dataset['Year'].median()
dataset['Year'].replace(np.nan, year_median, inplace=True)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot_prediction(y_test, y_pred)

# Plotting density curve to see if the values are skewed
df = pd.DataFrame({'Salary': y})
```

```
sns.distplot(df['Salary'], hist=True, kde=True,
        bins=int(180/5), color = 'darkblue',
        hist_kws={'edgecolor':'black'},
        kde_kws={'linewidth': 4})

# Map y to log scale handle skewed value in training
y_train_log = np.log(y_train)

# retraining the regressor model
model.fit(X_train, y_train_log)
y_pred = np.exp(model.predict(X_test)) #Taking inverse log of the predicted values
plot_prediction(y_test, y_pred)

# TODO Try other encoding methods and complex models
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators = 5, random_state = 42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot_prediction(y_test, y_pred)

# TODO Try other encoding methods and complex models
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators = 10, random_state = 42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot_prediction(y_test, y_pred)
# cat boost modeling
from catboost import CatBoostRegressor
model = CatBoostRegressor()
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot_prediction(y_test, y_pred)
model = CatBoostRegressor(iterations=10000,
                          learning_rate=0.03,
                          depth=6)
y_train_log = np.log(y_train)
model.fit(X_train, y_train_log)
y_pred = np.exp(model.predict(X_test))
plot_prediction(y_test, y_pred)
```