Help the HR

Problem Statement:

Help the HR division of a company by developing a model which provides the company with the data of how well the candidate will perform once inducted into the company.

Group Information:

1) Soham Bhave
2) Atharva Purohit
3) Swaraj Puri

Project GitHub Link:

https://github.com/RoboSpark-2021/robospark-2021-FT-Help-HR/tree/final_task

Project Algorithm:

1. Mount the drive on google colab.
2. Import packages like pandas, numpy etc.
3. Create the dataframe using Pandas .
4. Now start the data preprocessing.
5. Find the null values and remove the column with the maximum number of null values around 200 and greater than 200.
6. Those columns which have less number of null values say for example 10, replace them by median or mean values.
7. Then drop the columns which we find unnecessary.
8. Normalize the required columns by using the formula (x-xmin)/(xmax-xmin).
9. Now create the 2 copies one for model training and one for visualization.
10. Now import matplotlib for data visualization and use scatter plot 2D and 3D for Predictions score vs rest of the columns.
11. Also implement the Box plot.

12. After that import sklearn library and split the data in training and testing data.
13. Scale data using standard scaler.
14. Now Import LinearRegression from Sklearn.preprocessing and train the model and check the accuracy for training and testing dataset.
15. Also check whether the model is overfitting or not.
16. Do the same for Polynomial Regression.
17. from sklearn.ensemble import RandomForestRegressor and from sklearn.linear_model import LinearRegression and fit the model and predict.
18. Find the average of the predict and at last find the mean squared error.
19. After that import tensorflow and keras libraries for deep learning models.
20. Create a ANN.
21. Train the ANN model and check the accuracy.
22. Now test the model and check the accuracy.
23. Use Naïve bayes Algorithm and train the model.
24. Use a confusion matrix to understand how many correct & wrong values the model predicted.
25. Finally compare the accuracy of all the different models used.

Problems Faced:

1. Confusion whether to use regression or classification.
2. Didn't understand how many columns to drop
3. The dataset was imbalanced.
4. Very less columns had good correlation with the target variable so it was hard to train the model
5. Problem implementing confusion matrix for regression.
6. Too much RAM consumption for Polynomial Regression.
7. Google colab crashed while implementing Polynomial Regression.
8. Thought of Multiple Regression but Algorithm was too complex to understand.
9. Problem in deciding what should be the batch size, units in a dense network.
10. Couldn't implement CNN.

Alternative Solutions found for problems mentioned above:

1. First problem was solved by Developing ANN for classification and ML models for Regression.
2. Number of columns to be dropped was decided on the basis of Data visualization and by reading the dataset and trying to correlate the target variable with the other columns.
3. Through research we learned that the confusion matrix couldn't be implemented for regression.
4. For the Polynomial Regression problem I came up with an Ensemble model which helped to determine the mean squared error between the two predicted results of the two models.
5. Through research and watching videos I had to set the standard batch size, units for ANN dense networks.
6. CNN can't be implemented for dataframes, It is implemented for Images.
7. KNN can also be used

Code snippet ss:

DRIVE ACCESS AUTHENTICATION

[2]
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

IMPORTING PACKAGES

[3]
```
import numpy as np
import pandas as pd
```

READING THE DATASET

[4]
```
df = pd.read_csv('/content/drive/MyDrive/TRF/HRDataset_v14.csv')
```

DATA PREPROCESSING

[5]
```
df
```

[5]

1	Ait Sidi, Karthikeyan	10084	1	1	1	5	3	3	0	104437	1	27	Sr. DBA	MA	2148	05/05/75	M
2	Akinkuolie, Sarah	10196	1	1	0	5	5	3	0	64955	1	20	Production Technician II	MA	1810	09/19/88	F
3	Alagbe,Trina	10088	1	1	0	1	5	3	0	64991	0	19	Production Technician I	MA	1886	09/27/88	F
4	Anderson, Carol	10069	0	2	0	5	5	3	0	50825	1	19	Production Technician I	MA	2169	09/08/89	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
306	Woodson, Jason	10135	0	0	1	1	5	3	0	65893	0	20	Production Technician II	MA	1810	05/11/85	M
307	Ybarra, Catherine	10301	0	0	0	5	5	1	0	48513	1	19	Production Technician I	MA	2458	05/04/82	F
308	Zamora, Jennifer	10010	0	0	0	1	3	4	0	220450	0	6	CIO	MA	2067	08/30/79	F
309	Zhou, Julia	10043	0	0	0	1	3	3	0	89292	0	9	Data Analyst	MA	2148	02/24/79	F
310	Zima, Colleen	10271	0	4	0	1	5	3	0	45046	0	19	Production Technician I	MA	1730	08/17/78	F

311 rows × 36 columns

DATA CLEANING AND REMOVING THE NULL VALUES

```
[6] df.isnull().sum()
```

```
Employee_Name               0
EmpID                       0
MarriedID                   0
MaritalStatusID             0
GenderID                    0
EmpStatusID                 0
DeptID                      0
PerfScoreID                 0
FromDiversityJobFairID      0
Salary                      0
Termd                       0
PositionID                  0
Position                    0
State                       0
Zip                         0
DOB                         0
Sex                         0
MaritalDesc                 0
CitizenDesc                 0
HispanicLatino              0
RaceDesc                    0
DateofHire                  0
DateofTermination         207
TermReason                  0
EmploymentStatus            0
Department                  0
ManagerName                 0
ManagerID                   8
RecruitmentSource           0
PerformanceScore            0
EngagementSurvey            0
```

```
[6] EmploymentStatus            0
    Department                  0
    ManagerName                 0
    ManagerID                   8
    RecruitmentSource           0
    PerformanceScore            0
    EngagementSurvey            0
    EmpSatisfaction             0
    SpecialProjectsCount        0
    LastPerformanceReview_Date  0
    DaysLateLast30              0
    Absences                    0
    dtype: int64
```

```
[7] df.columns
```

```
Index(['Employee_Name', 'EmpID', 'MarriedID', 'MaritalStatusID', 'GenderID',
       'EmpStatusID', 'DeptID', 'PerfScoreID', 'FromDiversityJobFairID',
       'Salary', 'Termd', 'PositionID', 'Position', 'State', 'Zip', 'DOB',
       'Sex', 'MaritalDesc', 'CitizenDesc', 'HispanicLatino', 'RaceDesc',
       'DateofHire', 'DateofTermination', 'TermReason', 'EmploymentStatus',
       'Department', 'ManagerName', 'ManagerID', 'RecruitmentSource',
       'PerformanceScore', 'EngagementSurvey', 'EmpSatisfaction',
       'SpecialProjectsCount', 'LastPerformanceReview_Date', 'DaysLateLast30',
       'Absences'],
      dtype='object')
```

```
[8] len(df.index)
```

```
311
```

```
[9] df['DateofTermination'].count()
```

```
[8] len(df.index)
```

```
311
```

```
[9] df['DateofTermination'].count()
```

```
104
```

```
[10] df['ManagerID'].count()
```

```
303
```

```
[11] df.drop(labels = {'Employee_Name','EmpID','Position','State','DOB','DateofHire','Department','ManagerName','ManagerID','RecruitmentSource','LastPerformanceReview_Date'} , axis = 1 , i
```

```
[12] df.drop(labels = {'DateofTermination'} , axis = 1 , inplace = True)
```

```
[13] df
```

	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID	PerfScoreID	FromDiversityJobFairID	Salary	Termd	PositionID	Zip	Sex	MaritalDesc	CitizenDesc	HispanicLatino	RaceI
0	0	0	1	1	5	4	0	62506	0	19	1960	M	Single	US Citizen	No	W
1	1	1	1	5	3	3	0	104437	1	27	2148	M	Married	US Citizen	No	W
2	1	1	0	5	5	3	0	64955	1	20	1810	F	Married	US Citizen	No	W
3	1	1	0	1	5	3	0	64991	0	19	1886	F	Married	US Citizen	No	W

[13]

4	0	2	0	5	5	3	0	50825	1	19	2169	F	Divorced	US Citizen	No	W
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
306	0	0	1	1	5	3	0	65893	0	20	1810	M	Single	US Citizen	No	W
307	0	0	0	5	5	1	0	48513	1	19	2458	F	Single	US Citizen	No	A
308	0	0	0	1	3	4	0	220450	0	6	2067	F	Single	US Citizen	No	W
309	0	0	0	1	3	3	0	89292	0	9	2148	F	Single	US Citizen	No	W
310	0	4	0	1	5	3	0	45046	0	19	1730	F	Widowed	US Citizen	No	A

311 rows × 24 columns

[14]
```
df.isnull().sum()
```

```
MarriedID                 0
MaritalStatusID           0
GenderID                  0
EmpStatusID               0
DeptID                    0
PerfScoreID               0
FromDiversityJobFairID    0
Salary                    0
Termd                     0
PositionID                0
```

[14]
```
MaritalDesc               0
CitizenDesc               0
HispanicLatino            0
RaceDesc                  0
TermReason                0
EmploymentStatus          0
PerformanceScore          0
EngagementSurvey          0
EmpSatisfaction           0
SpecialProjectsCount      0
DaysLateLast30            0
Absences                  0
dtype: int64
```

[15]
```
df.isnull().sum()
```

```
MarriedID                 0
MaritalStatusID           0
GenderID                  0
EmpStatusID               0
DeptID                    0
PerfScoreID               0
FromDiversityJobFairID    0
Salary                    0
Termd                     0
PositionID                0
Zip                       0
Sex                       0
MaritalDesc               0
CitizenDesc               0
HispanicLatino            0
RaceDesc                  0
TermReason                0
EmploymentStatus          0
PerformanceScore          0
EngagementSurvey          0
```

```
[16] # copy for visualization
     df_visualization = df.copy()
```

NORMAILZATION OF THE REQUIRED COLUMNS

```
[17] df
```

	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID	PerfScoreID	FromDiversityJobFairID	Salary	Termd	PositionID	Zip	Sex	MaritalDesc	CitizenDesc	HispanicLatino	RaceD
0	0	0	1	1	5	4	0	62506	0	19	1960	M	Single	US Citizen	No	W
1	1	1	1	5	3	3	0	104437	1	27	2148	M	Married	US Citizen	No	W
2	1	1	0	5	5	3	0	64955	1	20	1810	F	Married	US Citizen	No	W
3	1	1	0	1	5	3	0	64991	0	19	1886	F	Married	US Citizen	No	W
4	0	2	0	5	5	3	0	50825	1	19	2169	F	Divorced	US Citizen	No	W
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
306	0	0	1	1	5	3	0	65893	0	20	1810	M	Single	US Citizen	No	W
307	0	0	0	5	5	1	0	48513	1	19	2458	F	Single	US Citizen	No	A
308	0	0	0	1	3	4	0	220450	0	6	2067	F	Single	US Citizen	No	W

```
[18] cols_to_normalize = ['Salary','MaritalStatusID','EmpStatusID','DeptID','PerfScoreID','PositionID','Zip','EngagementSurvey','EmpSatisfaction','SpecialProjectsCount','DaysLateLast30','A

     # Find the maximum and minimum values
     max = []
     min = []

     # loop for getting the max and min of each column
     for i in cols_to_normalize :
       max.append(df[i].max())

     for i in cols_to_normalize:
       min.append(df[i].min())

     print("max = ",max,"min = ",min)

     k = 0

     # loop for normalizing the column values
     for i in cols_to_normalize :
       df[i] = (df[i] - min[k])/(max[k] - min[k])
       k = k + 1

max =  [250000, 4, 5, 6, 4, 30, 98052, 5.0, 5, 8, 6, 20] min =  [45046, 0, 1, 1, 1, 1, 1013, 1.12, 1, 0, 0, 1]
```

```
[19] df.head()
```

	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID	PerfScoreID	FromDiversityJobFairID	Salary	Termd	PositionID	Zip	Sex	MaritalDesc	CitizenDesc	HispanicLatino	Ra
0	0	0.00	1	0.0	0.8	1.000000	0	0.085190	0	0.620690	0.009759	M	Single	US Citizen	No	
1	1	0.25	1	1.0	0.4	0.666667	0	0.289777	1	0.896552	0.011696	M	Married	US Citizen	No	
3	1	0.25	0	0.0	0.8	0.666667	0	0.097315	0	0.620690	0.008996	F	Married	US Citizen	No	
4	0	0.50	0	1.0	0.8	0.666667	0	0.028197	1	0.620690	0.011913	F	Divorced	US Citizen	No	

SPLITING OF THE DATA

```
[20] # the copy of the dataframe
     df_model = df.copy()
```

ENCODING

```
[21] # Implement Sklearn label encoding on df
     from sklearn import preprocessing

[22] Lable_Encoder = preprocessing.LabelEncoder()

[23] df_model['Sex'] = Lable_Encoder.fit_transform(df_model['Sex'])

[24] df_model['MaritalDesc'] = Lable_Encoder.fit_transform(df_model['MaritalDesc'])

[25] df_model['CitizenDesc'] = Lable_Encoder.fit_transform(df_model['CitizenDesc'])

[26] df_model['HispanicLatino'] = Lable_Encoder.fit_transform(df_model['HispanicLatino'])
```

```
[24] df_model['MaritalDesc'] = Lable_Encoder.fit_transform(df_model['MaritalDesc'])
```

```
[25] df_model['CitizenDesc'] = Lable_Encoder.fit_transform(df_model['CitizenDesc'])
```

```
[26] df_model['HispanicLatino'] = Lable_Encoder.fit_transform(df_model['HispanicLatino'])
```

```
[27] df_model['RaceDesc'] = Lable_Encoder.fit_transform(df_model['RaceDesc'])
```

```
[28] df_model['TermReason'] = Lable_Encoder.fit_transform(df_model['TermReason'])
```

```
[29] df_model['EmploymentStatus'] = Lable_Encoder.fit_transform(df_model['EmploymentStatus'])
```

```
[30] df_model['PerformanceScore'] = Lable_Encoder.fit_transform(df_model['PerformanceScore'])
```

```
[31] df_model
```

	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID	PerfScoreID	FromDiversityJobFairID	Salary	Termd	PositionID	Zip	Sex	MaritalDesc	CitizenDesc	HispanicLatino
0	0	0.00	1	0.0	0.8	1.000000	0	0.085190	0	0.620690	0.009759	1	3	2	0
1	1	0.25	1	1.0	0.4	0.666667	0	0.289777	1	0.896552	0.011696	1	1	2	0
2	1	0.25	0	1.0	0.8	0.666667	0	0.097139	1	0.655172	0.008213	0	1	2	0
3	1	0.25	0	0.0	0.8	0.666667	0	0.097315	0	0.620690	0.008996	0	1	2	0
4	0	0.50	0	1.0	0.8	0.666667	0	0.028197	1	0.620690	0.011913	0	0	2	0

✓ 0s completed at 12:35 PM

```
[32] df_model.corr().head()
```

	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID	PerfScoreID	FromDiversityJobFairID	Salary	Termd	PositionID	Zip	Sex	MaritalDesc	Citiz
MarriedID	1.000000	0.164044	-0.024199	0.085619	-0.119932	-0.058362	-0.012708	0.026165	0.077028	-0.027334	-0.041147	-0.024199	-0.638592	-0.0
MaritalStatusID	0.164044	1.000000	-0.030236	0.114630	0.012768	0.044693	0.041117	-0.070291	0.099367	0.021923	0.010620	-0.030236	-0.460629	0.0
GenderID	-0.024199	-0.030236	1.000000	-0.032440	-0.038838	-0.054915	0.031493	0.056097	-0.015741	-0.081612	0.048539	1.000000	0.019533	-0.0
EmpStatusID	0.085619	0.114630	-0.032440	1.000000	0.088711	-0.071208	0.189025	-0.110912	0.948058	0.221221	-0.150527	-0.032440	-0.159212	-0.0
DeptID	-0.119932	0.012768	-0.038838	0.088711	1.000000	-0.084811	-0.129998	-0.448132	0.065922	0.030294	0.290023	-0.038838	0.090461	0.0

DATA VISUALIZATION

```
[33] import matplotlib.pyplot as plt
```

```
[34] df1 = df_visualization['Salary'].describe()
     fig = plt.figure(figsize =(10,7))
```

```
<Figure size 720x504 with 0 Axes>
```

```
[35] plt.boxplot(df1)
```

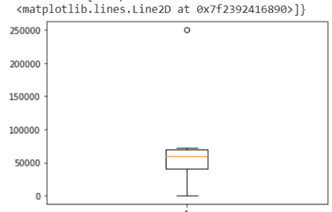
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f2392438990>],
 'caps': [<matplotlib.lines.Line2D at 0x7f2392416dd0>,
  <matplotlib.lines.Line2D at 0x7f239239e350>],
```


```
[34] fig = plt.figure(figsize =(10,7))
```

```
<Figure size 720x504 with 0 Axes>
```

```
[35] plt.boxplot(df1)
```


```
{'boxes': [<matplotlib.lines.Line2D at 0x7f2392438990>],
 'caps': [<matplotlib.lines.Line2D at 0x7f2392416dd0>,
  <matplotlib.lines.Line2D at 0x7f239239e350>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f239239ee10>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f239239e8d0>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f2392416350>,
  <matplotlib.lines.Line2D at 0x7f2392416890>]}
```

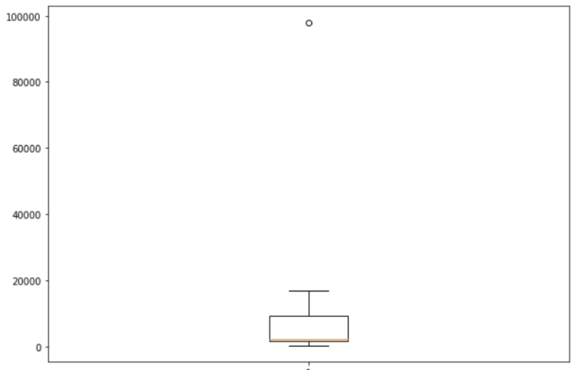



```
[36] df2 = df_visualization['Zip'].describe()
     fig = plt.figure(figsize =(10,7))
     plt.boxplot(df2)
```

```
{'boxes': [<matplotlib.lines.Line2D at 0x7f2391ee3690>]
```

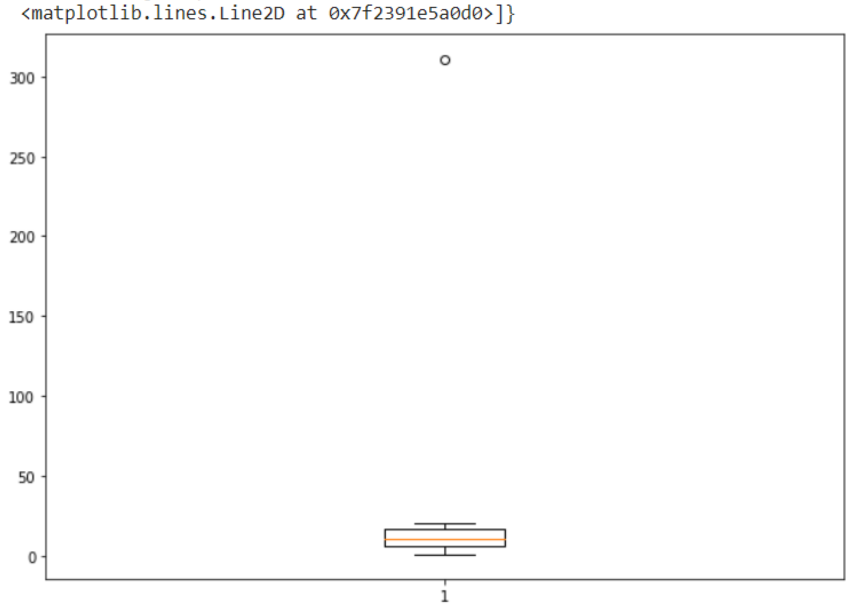
```
[36] df2 = df_visualization['Zip'].describe()
     fig = plt.figure(figsize =(10,7))
     plt.boxplot(df2)
```

{'boxes': [<matplotlib.lines.Line2D at 0x7f2391ee3690>],
 'caps': [<matplotlib.lines.Line2D at 0x7f2391ee8710>,
  <matplotlib.lines.Line2D at 0x7f2391ee8c10>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f2391ef0750>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f2391ef0210>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f2391ee3c50>,
  <matplotlib.lines.Line2D at 0x7f2391ee81d0>]}

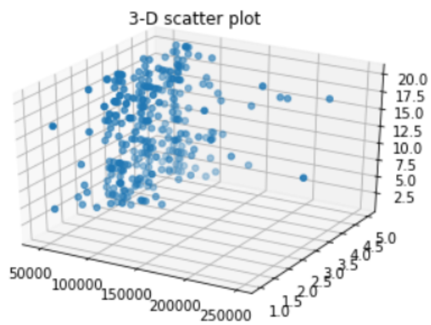



```
df3 = df_visualization['Absences'].describe()
fig = plt.figure(figsize =(10,7))
plt.boxplot(df3)
```

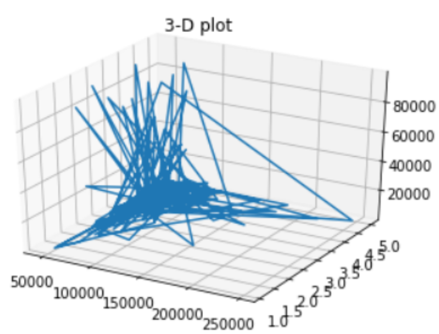
{'boxes': [<matplotlib.lines.Line2D at 0x7f2391e55590>],
 'caps': [<matplotlib.lines.Line2D at 0x7f2391e5a610>,
  <matplotlib.lines.Line2D at 0x7f2391e5ab50>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f2391e62650>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f2391e62110>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f2391e55b50>,
  <matplotlib.lines.Line2D at 0x7f2391e5a0d0>]}



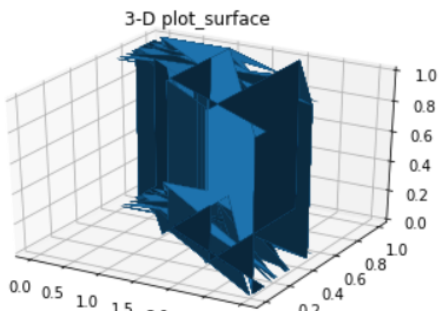
```
[38] from mpl_toolkits import mplot3d
     import numpy as np
     import matplotlib.pyplot as plt

     fig = plt.figure()

     # syntax for 3-D projection
     ax = plt.axes(projection ='3d')
     x = df_visualization['Salary']
     y = df_visualization['EmpSatisfaction']
     z = df_visualization['Absences']

     ax.scatter(x,y,z)
     ax.set_title('3-D scatter plot')
     plt.show()
```



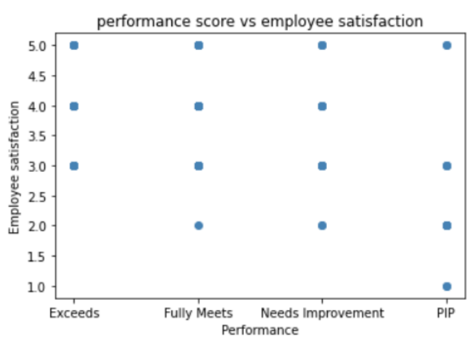

```
[39] from mpl_toolkits import mplot3d
     import numpy as np
     import matplotlib.pyplot as plt

     fig = plt.figure()

     # syntax for 3-D projection
     ax = plt.axes(projection ='3d')
     x = df_visualization['Salary']
     y = df_visualization['EngagementSurvey']
     z = df_visualization['Zip']

     ax.plot3D(x,y,z)
     ax.set_title('3-D plot')
     plt.show()
```



```
[40] from mpl_toolkits import mplot3d
     import numpy as np
     import matplotlib.pyplot as plt

     fig = plt.figure()

     # syntax for 3-D projection
     ax = plt.axes(projection ='3d')
     x = df_model['PerformanceScore']
     y = df_model['EngagementSurvey']
     z = df_model[['Sex']]

     ax.plot_surface(x,y,z)
     ax.set_title('3-D plot_surface')
     plt.show()
```



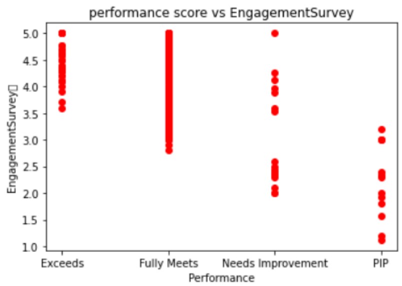

```
[41] # scatterplot between performance score and  EmpSatisfaction
     plt.scatter((df_visualization['PerformanceScore']) , df_visualization['EmpSatisfaction'], c = ['steelblue'])
     plt.xlabel("Performance")
     plt.ylabel("Employee satisfaction")
     plt.title("performance score vs employee satisfaction")
     plt.show()
```




```
[42] # scatterplot between performance score and  EngagementSurvey
     plt.scatter((df_visualization['PerformanceScore']) , df_visualization['EngagementSurvey'], c = ['Red'])
     plt.xlabel("Performance")
     plt.ylabel("EngagementSurvey  ")
```

```
[42] # scatterplot between performance score and  EngagementSurvey
plt.scatter((df_visualization['PerformanceScore']) , df_visualization['EngagementSurvey'], c = ['Red'])
plt.xlabel("Performance")
plt.ylabel("EngagementSurvey  ")
plt.title("performance score vs EngagementSurvey")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarning: Glyph 9 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarning: Glyph 9 missing from current font.
  font.set_text(s, 0, flags=flags)
```




```
[43] # scatterplot between performance score and  Absences
plt.scatter((df_visualization['PerformanceScore']) , df_visualization['Absences'], c = ['Green'])
plt.xlabel("Performance")
plt.ylabel("Absences")
plt.title("performance score vs Absences")
```

```
[43] # scatterplot between performance score and  Absences
plt.scatter((df_visualization['PerformanceScore']) , df_visualization['Absences'], c = ['Green'])
plt.xlabel("Performance")
plt.ylabel("Absences")
plt.title("performance score vs Absences")
plt.show()
```


```
[44] # scatterplot between performance score and  SpecialProjectsCount
plt.scatter((df_visualization['PerformanceScore']) , df_visualization['SpecialProjectsCount'], c = ['Maroon'])
plt.xlabel("Performance")
plt.ylabel("SpecialProjectsCount")
plt.title("performance score vs SpecialProjectsCount")
plt.show()
```

[44]

[45]

```
# scatterplot between performance score and   Salary
plt.scatter((df_visualization['PerformanceScore']) , df_visualization['Salary'], c = ['Orange'])
plt.xlabel("Performance")
plt.ylabel("Salary")
plt.title("performance score vs Salary")
plt.show()
```

MODEL IMPLEMENTATION

TRAIN TEST SPLIT

[46]
```python
from sklearn import model_selection
```

[47]
```python
# separates target variable from the rest of the features
X = df_model.drop('PerformanceScore' , axis = 1)
Y = df_model['PerformanceScore']
```

[48]
```python
X
```

	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID	PerfScoreID	FromDiversityJobFairID	Salary	Termd	PositionID	Zip	Sex	MaritalDesc	CitizenDesc	HispanicLatino
0	0	0.00	1	0.0	0.8	1.000000	0	0.085190	0	0.620690	0.009759	1	3	2	0
1	1	0.25	1	1.0	0.4	0.666667	0	0.289777	1	0.896552	0.011696	1	1	2	0
2	1	0.25	0	1.0	0.8	0.666667	0	0.097139	1	0.655172	0.008213	0	1	2	0
3	1	0.25	0	0.0	0.8	0.666667	0	0.097315	0	0.620690	0.008996	0	1	2	0
4	0	0.50	0	1.0	0.8	0.666667	0	0.028197	1	0.620690	0.011913	0	0	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
306	0	0.00	1	0.0	0.8	0.666667	0	0.101716	0	0.655172	0.008213	1	3	2	0
307	0	0.00	0	1.0	0.8	0.000000	0	0.016916	1	0.620690	0.014891	0	3	2	0
308	0	0.00	0	0.0	0.4	1.000000	0	0.855821	0	0.172414	0.010862	0	3	2	0
309	0	0.00	0	0.0	0.4	0.666667	0	0.215883	0	0.275862	0.011696	0	3	2	0
310	0	1.00	0	0.0	0.8	0.666667	0	0.000000	0	0.620690	0.007389	0	4	2	0

311 rows × 23 columns

[49]
```python
Y
```

```
0      0
1      1
2      1
3      1
4      1
      ..
306    1
307    3
308    0
309    1
310    1
Name: PerformanceScore, Length: 311, dtype: int64
```

[50]
```python
# For training the model we divide the data into training data and testing data
# for that we use train_test_split function

X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size = 0.3, random_state = 0, stratify=Y )
```

[51]
```python
X_train.shape
```

```
(217, 23)
```

[52]
```python
Y_train.shape
```

```
(217,)
```

[53]
```python
X_test.shape
```

```
(94, 23)
```

Feature scaling

```
[55] from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

USING THE LINEAR REGRESSION TO TRAIN THE MODEL

```
[56] from sklearn.linear_model import LinearRegression
     from sklearn.preprocessing import PolynomialFeatures
```

```
[57] df_model_regression = LinearRegression(fit_intercept = True).fit(X_train, Y_train)
```

```
[58] df_model_regression.coef_
```

```
array([ 1.66902404e-02,  7.50449402e-03,  3.54431332e-03, -1.71028711e-01,
        1.33136169e-02, -5.47293085e-01,  1.88991253e-03, -4.84253710e-03,
        1.83516586e-01, -2.04967494e-02, -1.49861040e-02,  3.54431332e-03,
       -5.85545350e-04,  4.99270623e-03, -8.02452234e-04, -1.22113159e-03,
       -1.42260322e-02, -8.50210103e-05, -2.09664762e-02,  1.72402835e-02,
        1.57876604e-03,  1.93518852e-02, -1.41651467e-02])
```

```
[59] # for training dataset
     print("Score of the training dataset is ----->",df_model_regression.score(X_train, Y_train))
```

```
Score of the training dataset is -----> 0.9639817018710427
```

```
[60] # for testing dataset
     print("Score of the testing dataset is ----->",df_model_regression.score(X_test, Y_test))
```

```
Score of the testing dataset is -----> 0.854992242934876
```

```
[66] from sklearn.ensemble import RandomForestRegressor
     from sklearn.linear_model import LinearRegression
```

```
[65] model_1 = LinearRegression()
     model_2 = RandomForestRegressor()
```

```
[67] model_1.fit(X_train, Y_train)
     model_2.fit(X_train, Y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

```
[68] pred_1 = model_1.predict(X_test)
     pred_2 = model_2.predict(X_test)
```

```
[69] pred_final = (pred_1+pred_2)/2.0
```

```
pred_1 = model_1.predict(X_test)
```

```
[68] pred_2 = model_2.predict(X_test)
```

```
[69] pred_final = (pred_1+pred_2)/2.0
```

```
[73] from sklearn.metrics import mean_squared_error, accuracy_score
```

```
[71] print(mean_squared_error(Y_test, pred_final))

0.04088476927290222
```

MAKING THE CONFUSION MATRIX

```
[75] from sklearn.metrics import confusion_matrix, accuracy_score
```

DEEP LEARNING MODEL

ARTIFICIAL NEURAL NETWORK

```
[76] import keras
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
```

```
[77] df_model_dense = Sequential()
```

```
[78] df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu', input_dim = 23))
```

```
[79] df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
     df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
     df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
     df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
     df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
     df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
     df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
     df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
     df_model_dense.add(Dense(units = 12, kernel_initializer= 'uniform', activation = 'relu'))
```

```
[80] df_model_dense.add(Dense(units = 1, kernel_initializer= 'uniform', activation = 'softmax'))
```

```
[81] df_model_dense.compile(optimizer = 'adam', loss = "MSE", metrics = ['accuracy'])
```

```
[82] print("For Training ---------------------------->")
     df_model_dense.fit(X_train, Y_train, batch_size = 5, epochs = 20)

For Training ---------------------------->
Epoch 1/20
44/44 [==============================] - 1s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 2/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 3/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 4/20
```

[82]

```
For Training ---------------------------->
Epoch 1/20
44/44 [==============================] - 1s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 2/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 3/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 4/20
44/44 [==============================] - 0s 3ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 5/20
44/44 [==============================] - 0s 3ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 6/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 7/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 8/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 9/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 10/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 11/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 12/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 13/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 14/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 15/20
44/44 [==============================] - 0s 3ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 16/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 17/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 18/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
```

```
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
```

[82]
```
Epoch 14/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 15/20
44/44 [==============================] - 0s 3ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 16/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 17/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 18/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 19/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
Epoch 20/20
44/44 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.7788
<keras.callbacks.History at 0x7f233bf7ac10>
```

[83]
```python
print("For Testing ----------------------------->")
df_model_dense.fit(X_test, Y_test, batch_size = 5, epochs = 20)
```

```
For Testing ----------------------------->
Epoch 1/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 2/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 3/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 4/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 5/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 6/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 7/20
19/19 [==============================] - 0s 3ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 8/20
```

```
[83] 19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 7/20
19/19 [==============================] - 0s 3ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 8/20
19/19 [==============================] - 0s 4ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 9/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 10/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 11/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 12/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 13/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 14/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 15/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 16/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 17/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 18/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 19/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
Epoch 20/20
19/19 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.7872
<keras.callbacks.History at 0x7f233a42ca50>
```

```
[84] Y_pred = df_model_dense.predict(X_test)
```

```
[85] print(Y_pred)
```

```
print(Y_pred)
```

```
[86] df_model_dense_confuse = confusion_matrix(Y_test,Y_pred)
     print(df_model_dense_confuse)
     accuracy_score(Y_test,Y_pred)
```

```
[[ 0 11  0  0]
 [ 0 74  0  0]
 [ 0  5  0  0]
 [ 0  4  0  0]]
0.7872340425531915
```