PREDICTIVE ANALYTICS

HR Analytics

By: Shubham Gupta
DM243080

Q1) Write a brief corporate summary of business case.

The business scenario centers around ScaleneWorks, a startup headquartered in Bangalore, founded in 2010 by Sanjay Shelvankar, Ashish Tiwari, and Sharon George. This company specializes in delivering innovative solutions for talent acquisition, aiming to transform the way businesses approach hiring in today's dynamic environment. Their primary expertise lies in optimizing business processes by enhancing people's skills, improving process efficiency, and embracing new technologies.

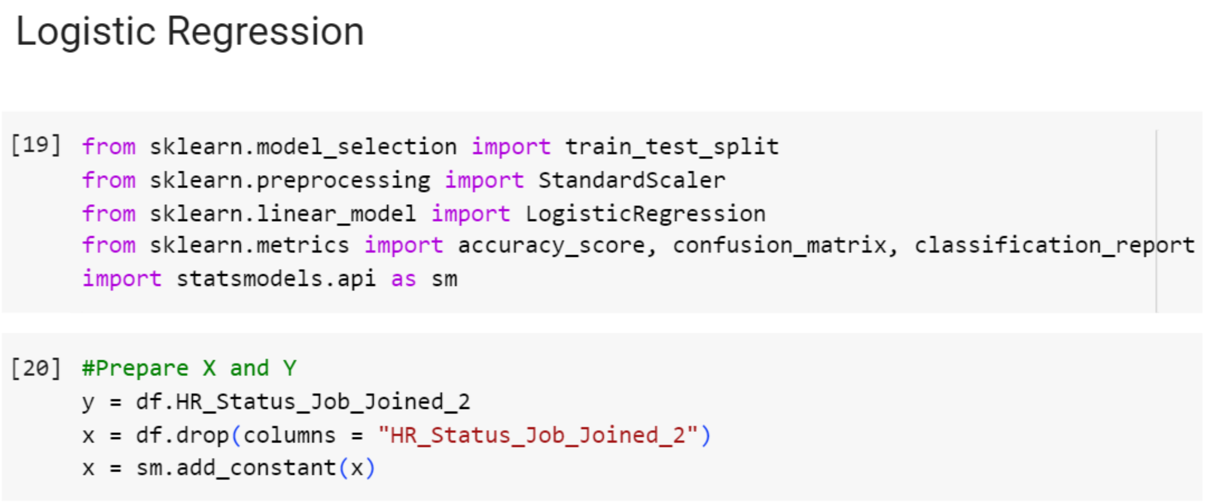
ScaleneWorks and its clients encounter a significant challenge in managing a high-quality workforce in an industry notorious for its high employee turnover. They have identified a problem known as "renege," where candidates accept job offers but fail to follow through by joining the company. Sanjay Shelvankar emphasizes the critical importance of identifying these reneges to prevent the wastage of valuable resources.

Sanjay underscores the impact of renege cases on clients who issue thousands of job offers each year. The associated costs, including the time and money invested in recruitment efforts and payments to agencies, increase substantially due to reneges. Ashish highlights the difficulties and potential compromises in resource quality that arise when reneges occur after commitments have been made to clients.

The primary goal of this case study is to investigate the feasibility of utilizing analytics to develop a predictive model that can anticipate whether a candidate will both accept a job offer and join the company.

This approach offers several advantages, including a reduction in the time and financial resources expended on finding and hiring individuals, better allocation of resources, and increased client satisfaction. This initiative aligns with ScaleneWorks' objective of becoming a highly respected provider of talent acquisition solutions.

Q2) Develop a logistic regression model which could be used by Company to predict candidates who are likely to join after accepting job offer? Support your answer with independent and dependent variables chosen with respect to dataset provided.



Logistic Regression

```
[19] from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
     import statsmodels.api as sm
```

```
[20] #Prepare X and Y
     y = df.HR_Status_Job_Joined_2
     x = df.drop(columns = "HR_Status_Job_Joined_2")
     x = sm.add_constant(x)
```

So we arrived at following independent variable after removing some variables which were not having any significant impact on dependent variable like Candidate_Reference_Number and Date_Of_Joining and there are some independent variables which are having high correlation with other variable which leads to false increase or decrease in R^2, to nullify that impact we have removed Salary_Pay_Band_Offered ,Age , Years_With_Current_Company.

Here is the list of independent variables

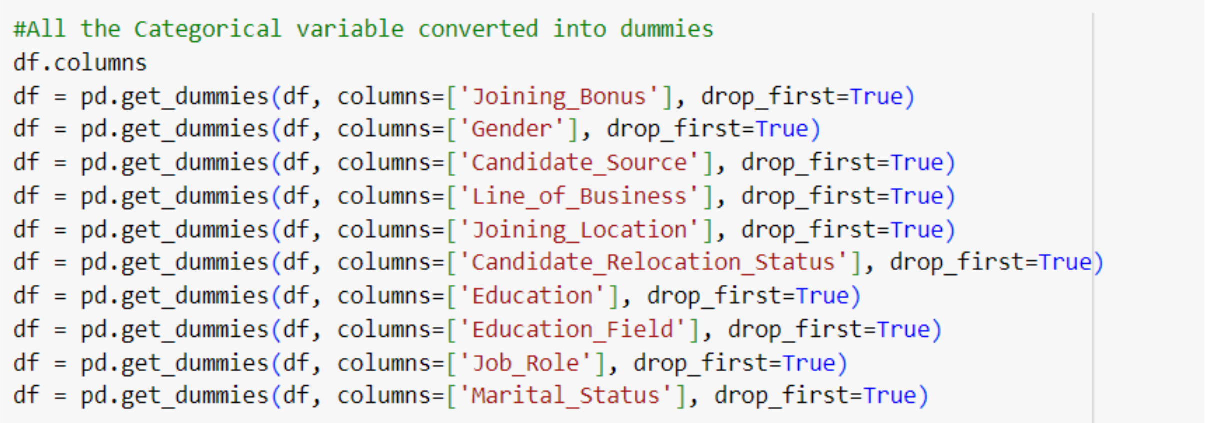
- Duration_to_Accept_Offer
- Notice_Period
- Percentage_Hike_Salary_Expected
- Percentage_Hike_Salary_Offered
- Joining_Bonus
- Gender
- Candidate_Source
- Relevant_Years_of_Experience
- Line_of_Business
- Joining_Location
- Candidate_Relocation_Status0
- Distance_from_Home
- Education
- Education_Field
- Job_Role
- Marital_Status
- Number_of_Companies_Worked
- Total_Number_of_Working_Years
- Work_Life_Balance
- Years_In_Current_Role
- Years_Since_Last_Promotion
- HR_Status_Job_Acceptance

The dependent variable in this case is “HR_Status_Job_Joined” because in this case we have to judge the impact of independent variables on the number of people finally joining the company therefore the HR_Status_Job_Joined is dependent variable.

3) Devise a predictive algorithm in order to calculate probability of acceptance of job offer and joining Company after offer acceptance? [Hint: Use odds ratio in logistic regression]

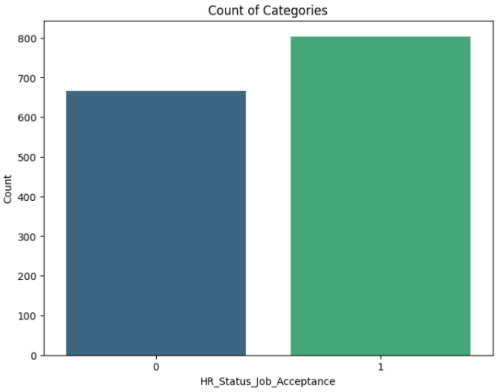
In order to build a predictive algorithm, we will use logistic regression to calculate the probability of acceptance of job offer and joining the company after offer acceptance.

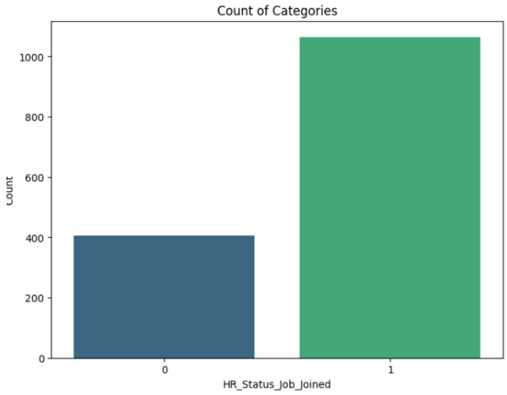
After EDA we dummy coded the variables



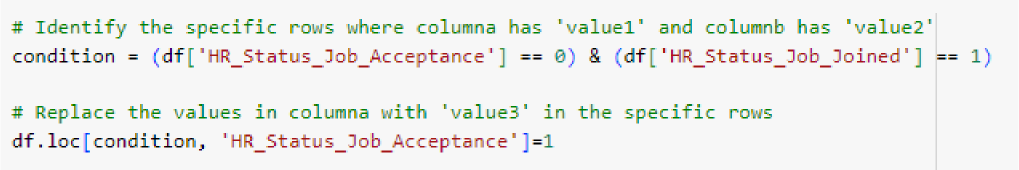
```
#All the Categorical variable converted into dummies
df.columns
df = pd.get_dummies(df, columns=['Joining_Bonus'], drop_first=True)
df = pd.get_dummies(df, columns=['Gender'], drop_first=True)
df = pd.get_dummies(df, columns=['Candidate_Source'], drop_first=True)
df = pd.get_dummies(df, columns=['Line_of_Business'], drop_first=True)
df = pd.get_dummies(df, columns=['Joining_Location'], drop_first=True)
df = pd.get_dummies(df, columns=['Candidate_Relocation_Status'], drop_first=True)
df = pd.get_dummies(df, columns=['Education'], drop_first=True)
df = pd.get_dummies(df, columns=['Education_Field'], drop_first=True)
df = pd.get_dummies(df, columns=['Job_Role'], drop_first=True)
df = pd.get_dummies(df, columns=['Marital_Status'], drop_first=True)
```

After this we found that there is a class imbalance in Job Joined and Job Acceptance



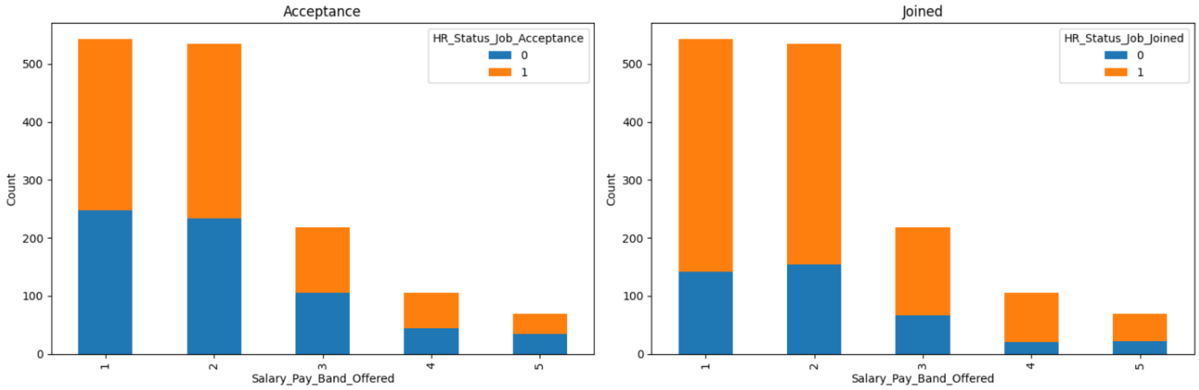


We need to balance it with the help of this code

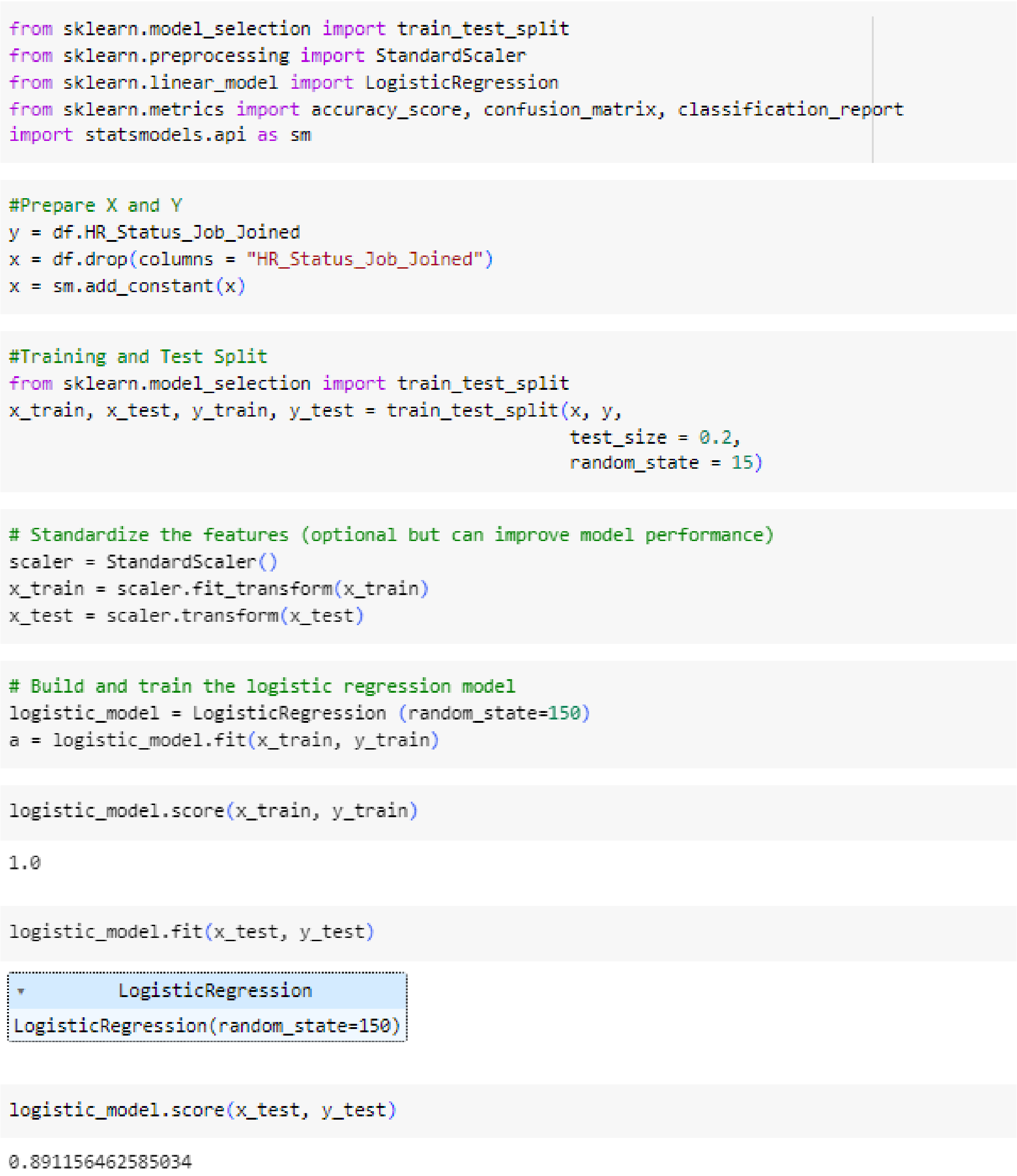


```
# Identify the specific rows where columna has 'value1' and columnb has 'value2'
condition = (df['HR_Status_Job_Acceptance'] == 0) & (df['HR_Status_Job_Joined'] == 1)

# Replace the values in columna with 'value3' in the specific rows
df.loc[condition, 'HR_Status_Job_Acceptance']=1
```

In this we can see that even if we create a new column which combined both HR_Status_Job_Acceptance and HR_Status_Job_Joined then the intersection of both of them would be similar to HR_Status_Job_Acceptance therefore we can proceed with the regression of only with HR_Status_Job_Joined.



Therefore we can go with logistic regression with only HR_Status_Job_Joined which is as follow



```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import statsmodels.api as sm
```

```
#Prepare X and Y
y = df.HR_Status_Job_Joined
x = df.drop(columns = "HR_Status_Job_Joined")
x = sm.add_constant(x)
```

```
#Training and Test Split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.2,
                                                    random_state = 15)
```

```
# Standardize the features (optional but can improve model performance)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
# Build and train the logistic regression model
logistic_model = LogisticRegression (random_state=150)
a = logistic_model.fit(x_train, y_train)
```

```
logistic_model.score(x_train, y_train)
```

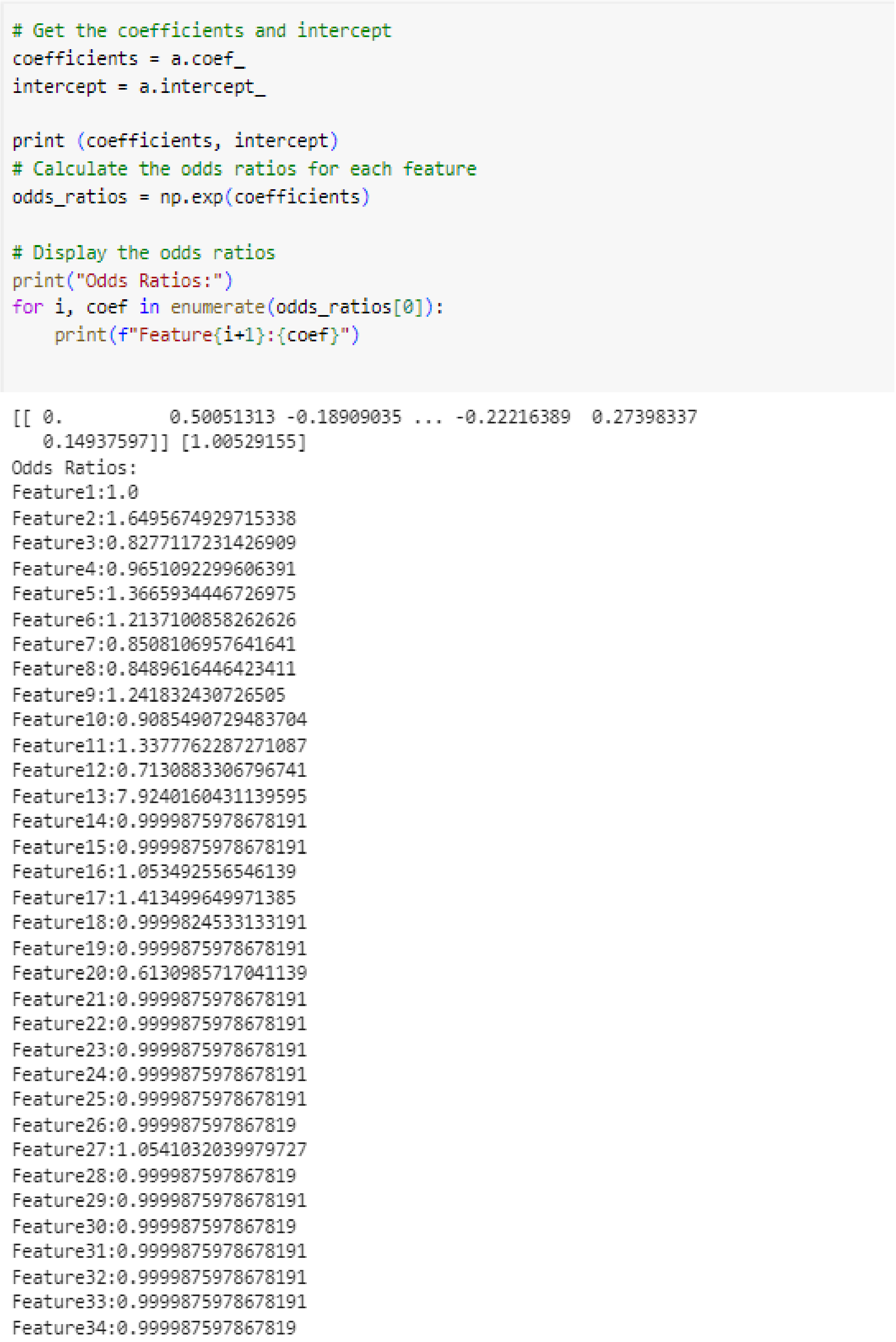
```
1.0
```

```
logistic_model.fit(x_test, y_test)
```

```
▾         LogisticRegression
LogisticRegression(random_state=150)
```

```
logistic_model.score(x_test, y_test)
```

```
0.891156462585034
```



```
# Get the coefficients and intercept
coefficients = a.coef_
intercept = a.intercept_

print (coefficients, intercept)
# Calculate the odds ratios for each feature
odds_ratios = np.exp(coefficients)

# Display the odds ratios
print("Odds Ratios:")
for i, coef in enumerate(odds_ratios[0]):
    print(f"Feature{i+1}:{coef}")
```

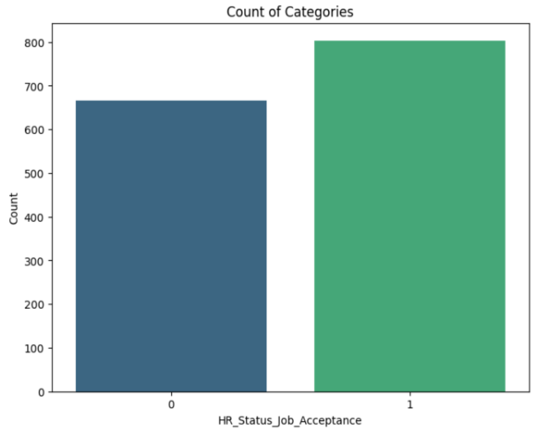
```
[[ 0.          0.50051313 -0.18909035 ... -0.22216389  0.27398337
   0.14937597]] [1.00529155]
Odds Ratios:
Feature1:1.0
Feature2:1.6495674929715338
Feature3:0.8277117231426909
Feature4:0.9651092299606391
Feature5:1.3665934446726975
Feature6:1.2137100858262626
Feature7:0.8508106957641641
Feature8:0.8489616446423411
Feature9:1.241832430726505
Feature10:0.9085490729483704
Feature11:1.3377762287271087
Feature12:0.7130883306796741
Feature13:7.9240160431139595
Feature14:0.9999875978678191
Feature15:0.9999875978678191
Feature16:1.053492556546139
Feature17:1.413499649971385
Feature18:0.9999824533133191
Feature19:0.9999875978678191
Feature20:0.6130985717041139
Feature21:0.9999875978678191
Feature22:0.9999875978678191
Feature23:0.9999875978678191
Feature24:0.9999875978678191
Feature25:0.9999875978678191
Feature26:0.999987597867819
Feature27:1.0541032039979727
Feature28:0.999987597867819
Feature29:0.9999875978678191
Feature30:0.999987597867819
Feature31:0.9999875978678191
Feature32:0.9999875978678191
Feature33:0.9999875978678191
Feature34:0.999987597867819
```

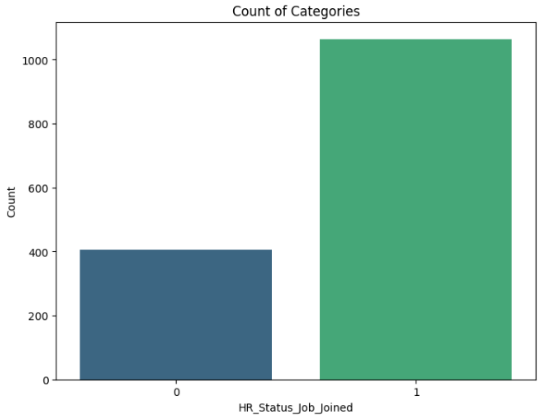
Q4) Using Python perform stated activities:

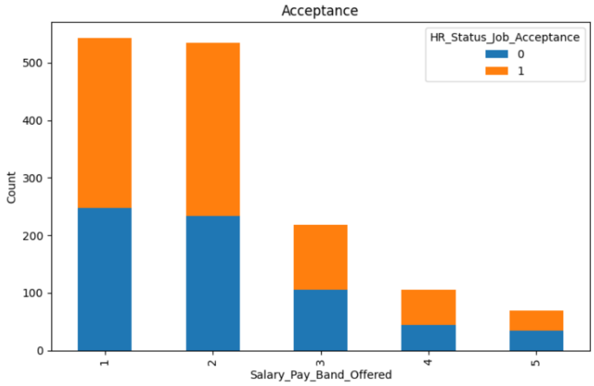
(a) Develop basic EDA on data provided using Python

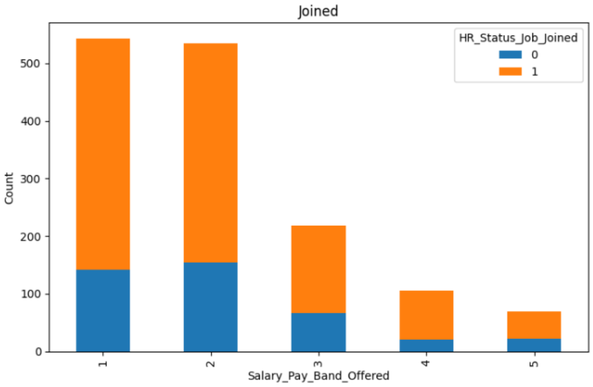
During our EDA Analysis we analysed the data initially using describe, shape and info function for general outlay of what type of data is present. Then we moved towards finding null values if there are any and we found that the data doesn’t contain any null values.

We moved towards graphs after the basic EDA

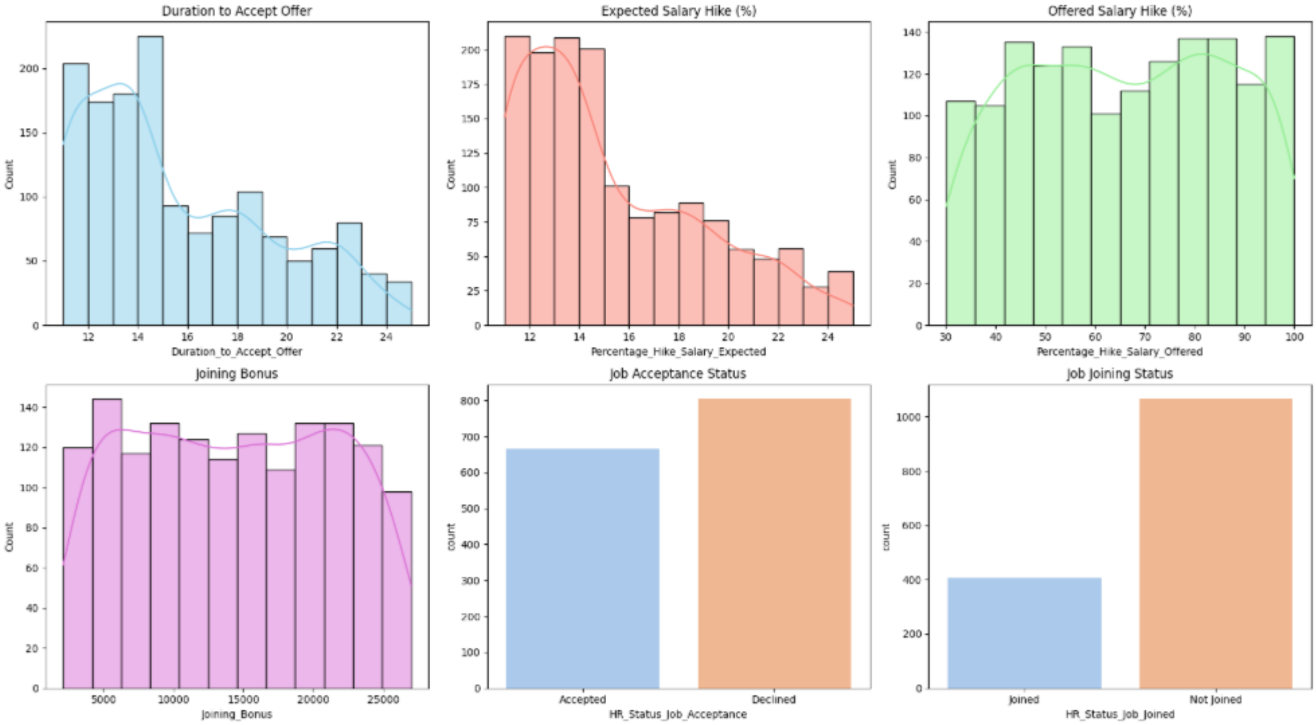


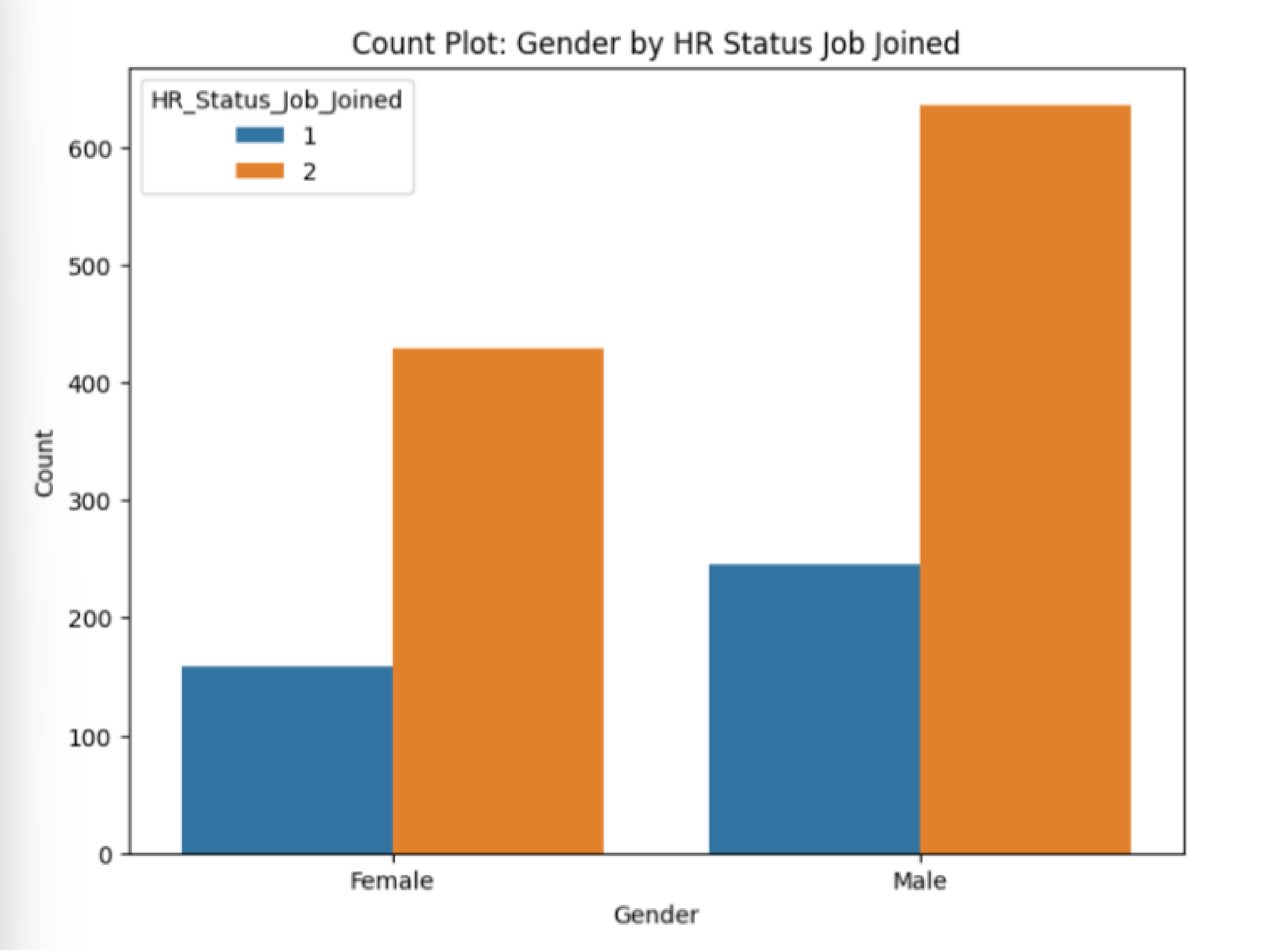


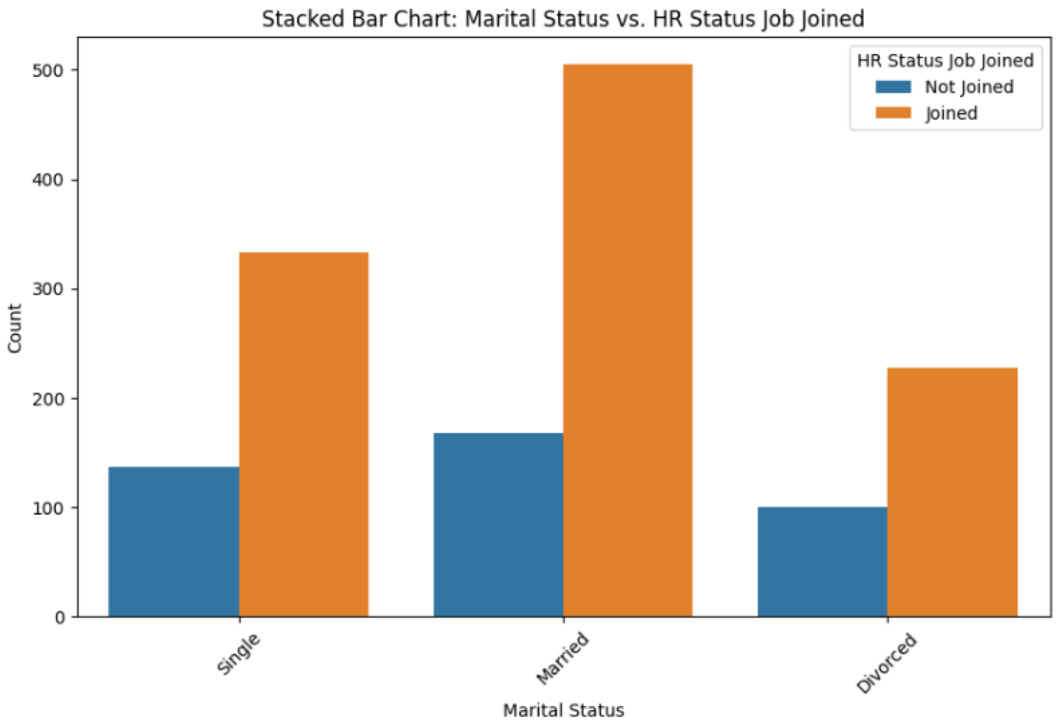




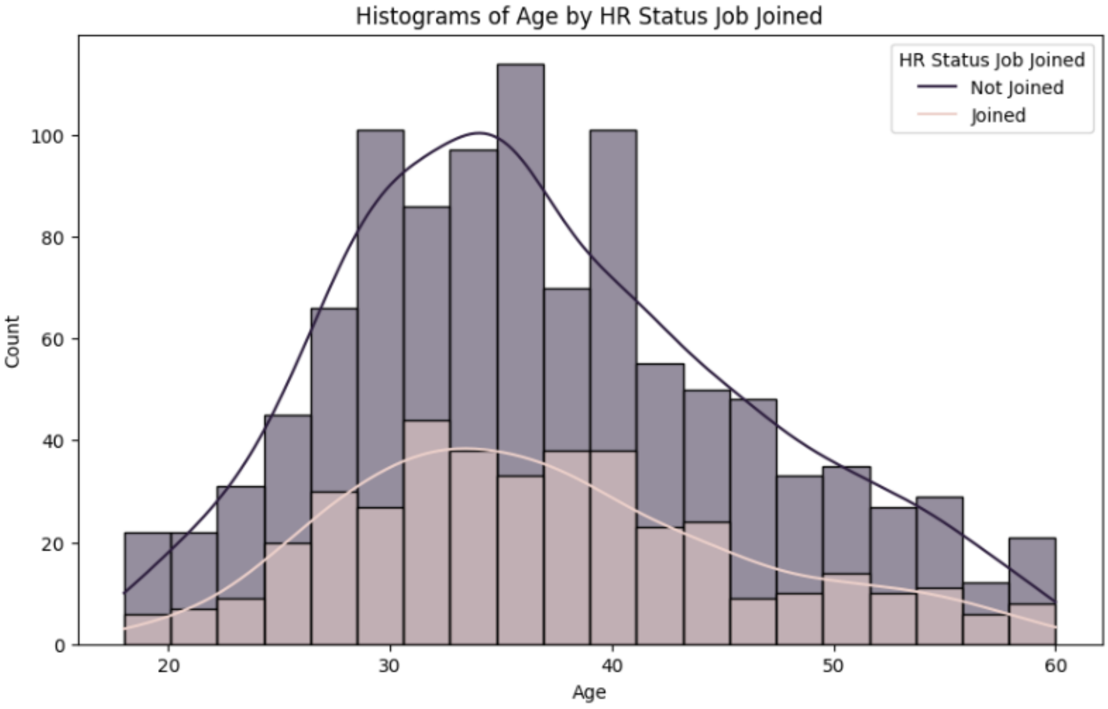
As in the case of acceptance and joining of job we are not given with whether 0 or 1 reflect “Yes” towards acceptance of Job offer or “No” towards acceptance of Job offer. To detect this, we created a cross tab where we can easily see that as the Salary Band will increase the joining rate of employees will boost therefore 1 should be Acceptance and 0 should be Non Acceptance of offer.

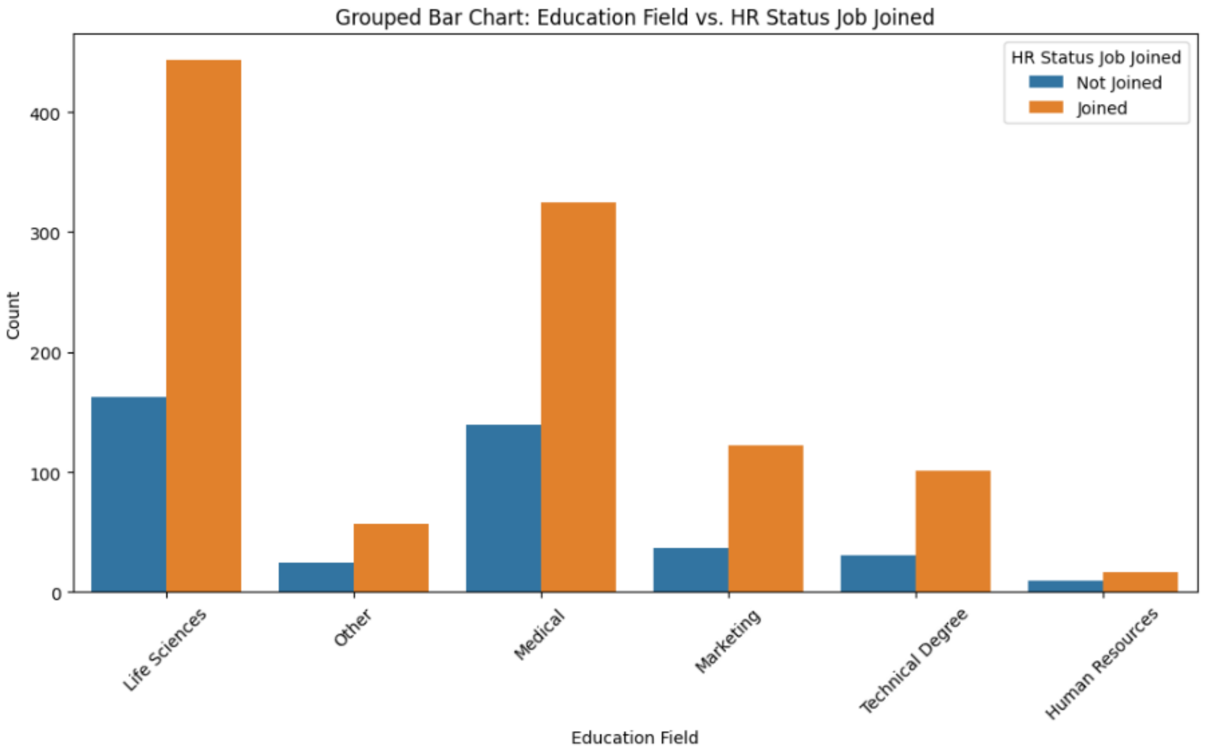

Duration to Accept Offer
Count
0
50
100
150
200
12
14
16
18
20
22
24
Duration_to_Accept_Offer
Expected Salary Hike (%)
Count
0
25
50
75
100
125
150
175
200
12
14
16
18
20
22
24
Percentage_Hike_Salary_Expected
Offered Salary Hike (%)
Count
0
20
40
60
80
100
120
140
30
40
50
60
70
80
90
100
Percentage_Hike_Salary_Offered
Joining Bonus
Count
0
20
40
60
80
100
120
140
5000
10000
15000
20000
25000
Joining_Bonus
Job Acceptance Status
count
0
100
200
300
400
500
600
700
800
Accepted
Declined
HR_Status_Job_Acceptance
Job Joining Status
count
0
200
400
600
800
1000
Joined
Not Joined
HR_Status_Job_Joined


Count Plot: Gender by HR Status Job Joined
HR_Status_Job_Joined
1
2
Count
0
100
200
300
400
500
600
Female
Male
Gender



As we can see from the graph where the percentage of people joining on the basis of marital status is given
Single - 140/320 = 43.75%
Married - 160/500 =32%
Divorced - 100/220 = 45.45%
This clearly depicts that Single and Divorced people have more chance of joining the Job rather than Married people.

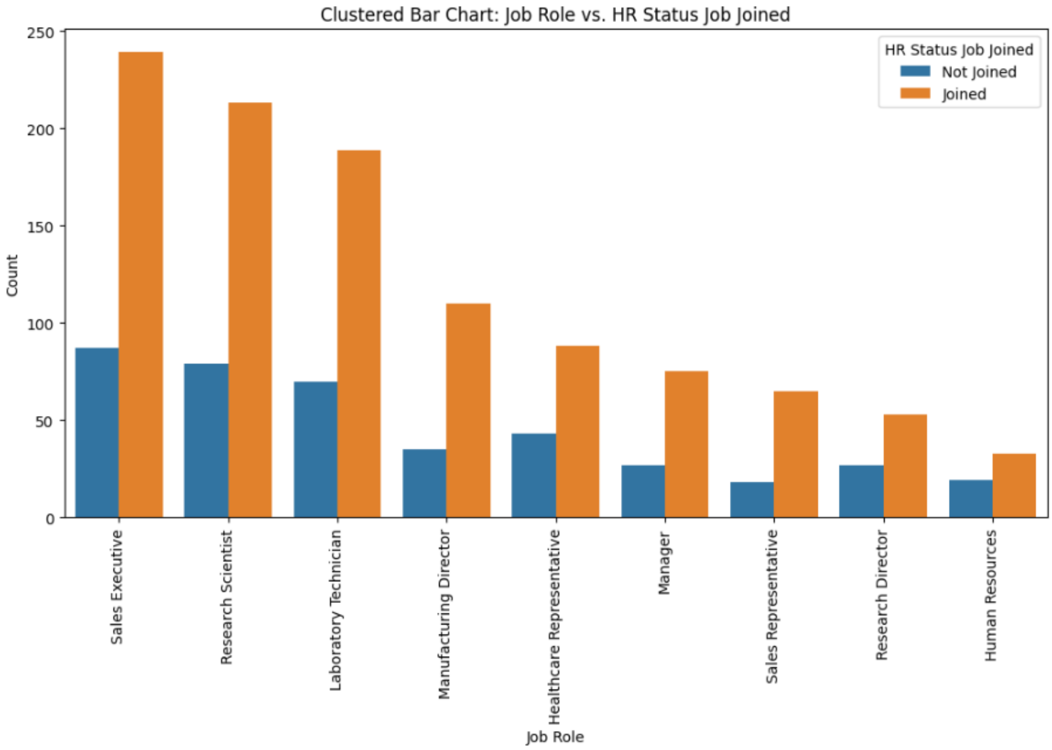




Life Science and Medical have highest join rate as we can see from the graph so promoting people joining from those field would improve the joining rate.

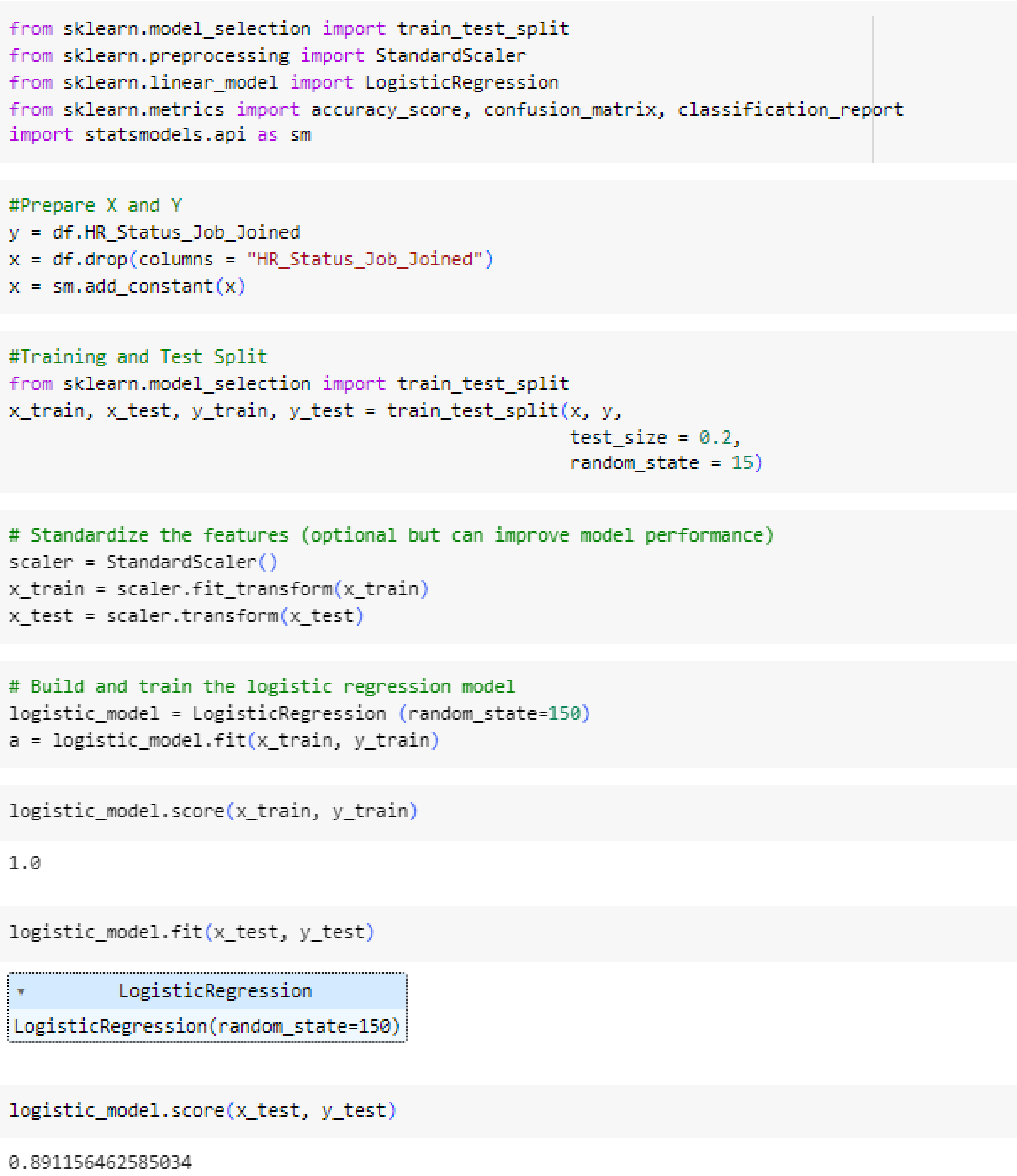
Life Science - 160/450 = 35.5%

Medical - 140/320 = 43.75%



Here we can see that the Joining rate of Sales Executive, Research Scientist and Laboratory Technician is significantly as compared to others.

b) Consider any of the above attributes as outcome variable and develop a regression model



```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import statsmodels.api as sm
```

```
#Prepare X and Y
y = df.HR_Status_Job_Joined
x = df.drop(columns = "HR_Status_Job_Joined")
x = sm.add_constant(x)
```

```
#Training and Test Split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.2,
                                                    random_state = 15)
```

```
# Standardize the features (optional but can improve model performance)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
# Build and train the logistic regression model
logistic_model = LogisticRegression (random_state=150)
a = logistic_model.fit(x_train, y_train)
```

```
logistic_model.score(x_train, y_train)
```

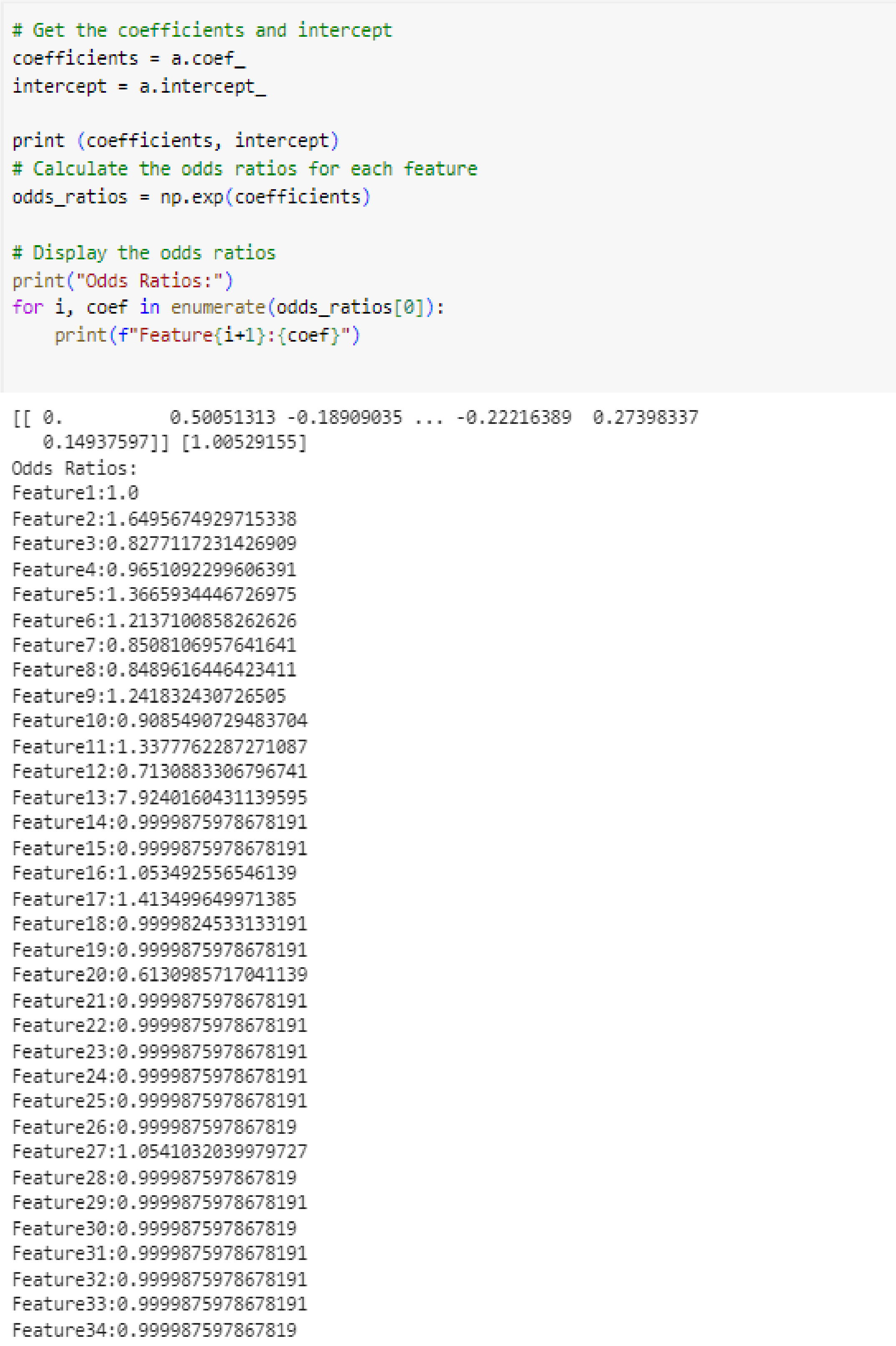
```
1.0
```

```
logistic_model.fit(x_test, y_test)
```

```
▾          LogisticRegression
LogisticRegression(random_state=150)
```

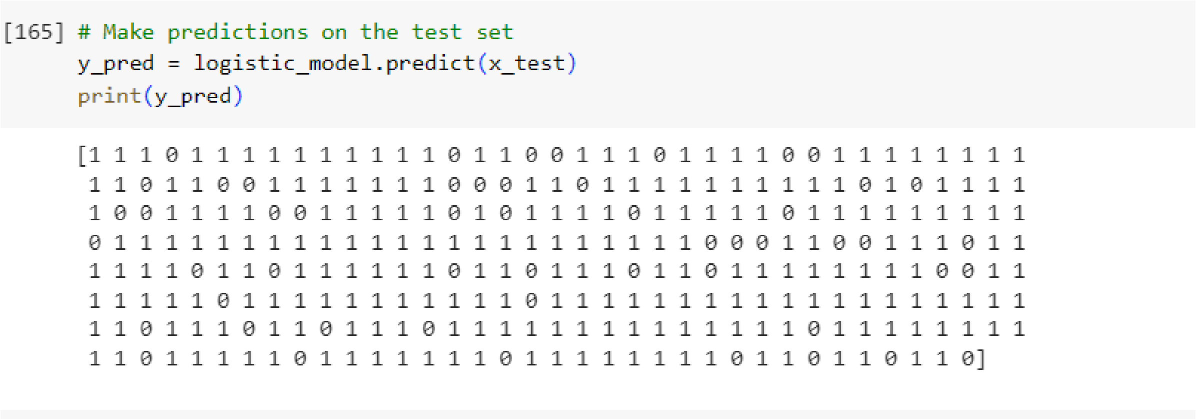
```
logistic_model.score(x_test, y_test)
```

```
0.891156462585034
```



```python
# Get the coefficients and intercept
coefficients = a.coef_
intercept = a.intercept_

print (coefficients, intercept)
# Calculate the odds ratios for each feature
odds_ratios = np.exp(coefficients)

# Display the odds ratios
print("Odds Ratios:")
for i, coef in enumerate(odds_ratios[0]):
    print(f"Feature{i+1}:{coef}")
```

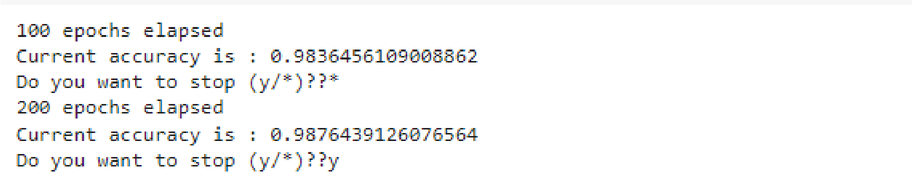
```
[[ 0.          0.50051313 -0.18909035 ... -0.22216389  0.27398337
   0.14937597]] [1.00529155]
Odds Ratios:
Feature1:1.0
Feature2:1.6495674929715338
Feature3:0.8277117231426909
Feature4:0.9651092299606391
Feature5:1.3665934446726975
Feature6:1.2137100858262626
Feature7:0.8508106957641641
Feature8:0.8489616446423411
Feature9:1.241832430726505
Feature10:0.9085490729483704
Feature11:1.3377762287271087
Feature12:0.7130883306796741
Feature13:7.9240160431139595
Feature14:0.9999875978678191
Feature15:0.9999875978678191
Feature16:1.053492556546139
Feature17:1.413499649971385
Feature18:0.9999824533133191
Feature19:0.9999875978678191
Feature20:0.6130985717041139
Feature21:0.9999875978678191
Feature22:0.9999875978678191
Feature23:0.9999875978678191
Feature24:0.9999875978678191
Feature25:0.9999875978678191
Feature26:0.999987597867819
Feature27:1.0541032039979727
Feature28:0.999987597867819
Feature29:0.9999875978678191
Feature30:0.999987597867819
Feature31:0.9999875978678191
Feature32:0.9999875978678191
Feature33:0.9999875978678191
Feature34:0.999987597867819
```



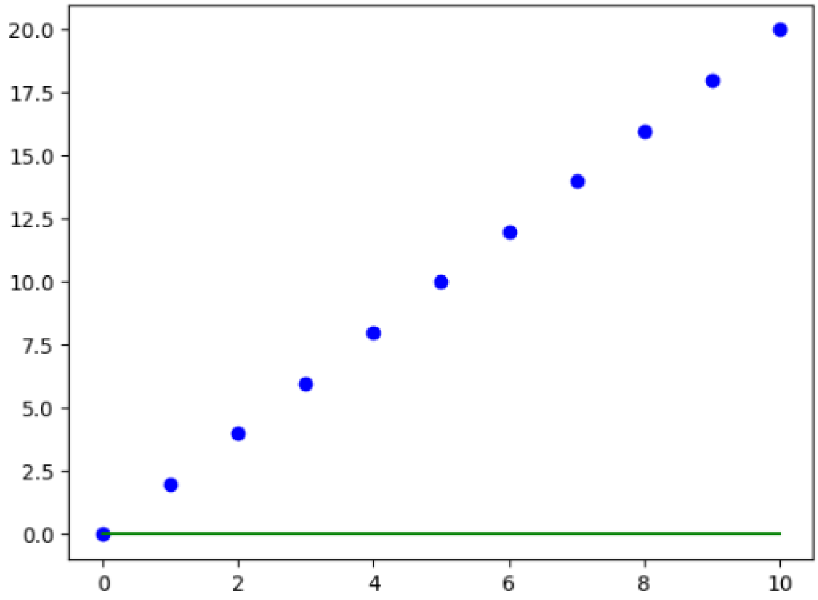
```
[165] # Make predictions on the test set
      y_pred = logistic_model.predict(x_test)
      print(y_pred)
```

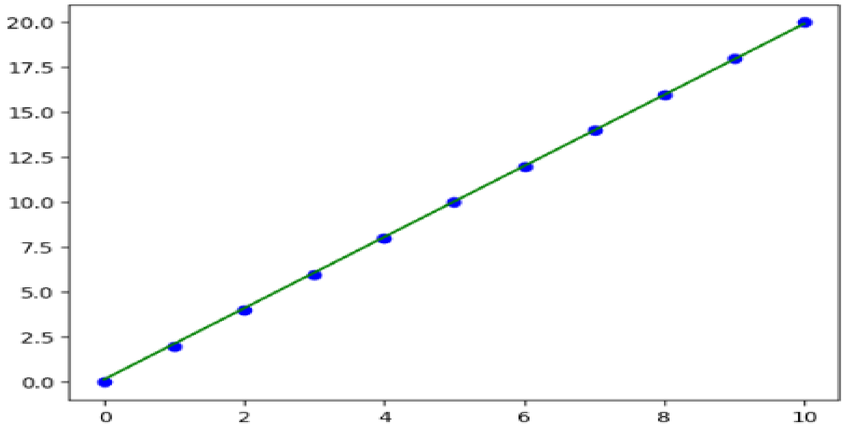
```
[1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1
 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1
 1 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 0 1 1
 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1
 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 0]
```

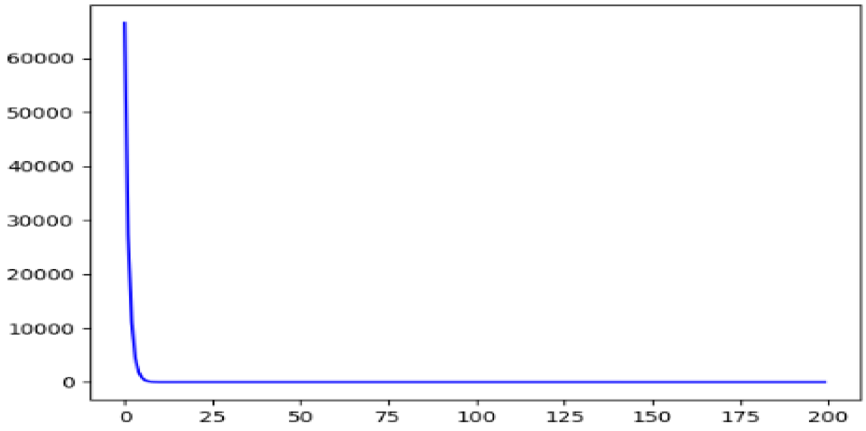
c) Calculate gradient descent of regression model developed in (b)



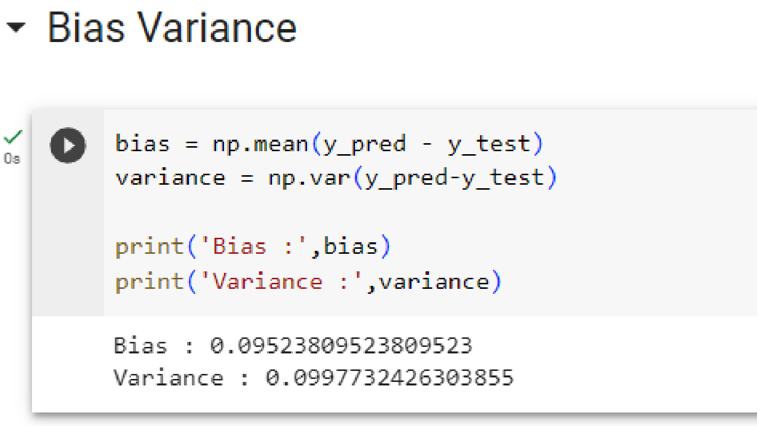
```
100 epochs elapsed
Current accuracy is : 0.9836456109008862
Do you want to stop (y/*)??*
200 epochs elapsed
Current accuracy is : 0.9876439126076564
Do you want to stop (y/*)??y
```







c) Calculate bias variance of regression model developed in (b)

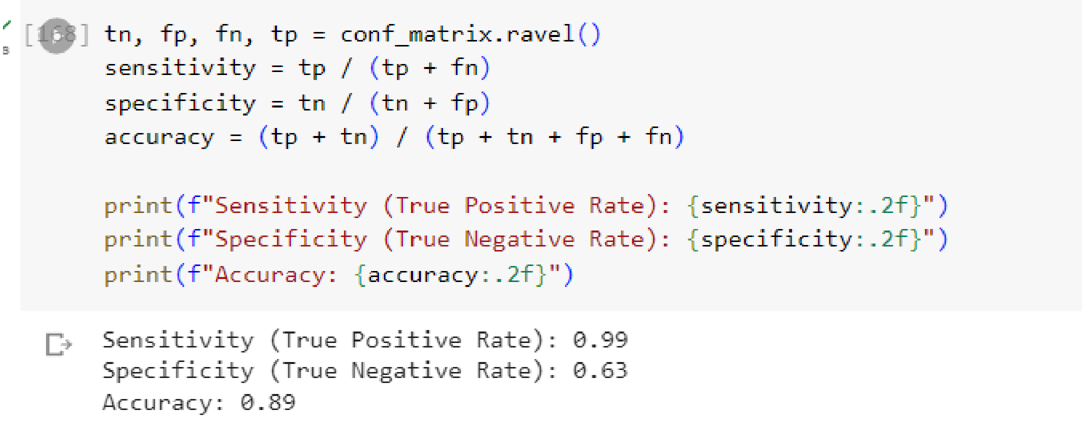


Bias Variance

```
bias = np.mean(y_pred - y_test)
variance = np.var(y_pred-y_test)

print('Bias :',bias)
print('Variance :',variance)
```

```
Bias : 0.09523809523809523
Variance : 0.0997732426303855
```

Q5) How do you interpret sensitivity, specificity and accuracy in model development



```
tn, fp, fn, tp = conf_matrix.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
accuracy = (tp + tn) / (tp + tn + fp + fn)

print(f"Sensitivity (True Positive Rate): {sensitivity:.2f}")
print(f"Specificity (True Negative Rate): {specificity:.2f}")
print(f"Accuracy: {accuracy:.2f}")
```

```
Sensitivity (True Positive Rate): 0.99
Specificity (True Negative Rate): 0.63
Accuracy: 0.89
```

- The **high sensitivity value** (0.99) suggests that the model is excellent at correctly identifying candidates who accepted the job. This is a desirable trait, especially if correctly identifying positive cases is a priority.

- The relatively **low specificity value** (0.63) suggests that the model has a high rate of false positives, meaning it incorrectly predicts that some candidates accepted the job when they did not. This may lead to some inefficiencies or additional follow-up for candidates who were wrongly classified as positive cases.

- The **high accuracy** (0.89) may be driven by the high sensitivity but can be misleading, especially if the cost of false positives is significant in your application. High sensitivity can sometimes come at the cost of lower specificity.

```
DDfrom sklearn import datasets
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import seaborn as sns
```

```
df = pd.read_csv("/content/HR Analytics Talent Acquisition Datasets.csv")
```

EDA

```
df.head()
```

	Candidate_Reference_Number	Date_of_Joining	Duration_to_Accept_C
0	1	07-05-2011	
1	2	3/30/2015	
2	4	07-05-2011	
3	5	01-07-2008	
4	7	07-11-2011	

```
 #We get the basic summary statistics
df.describe()
```

	Candidate_Reference_Number	Duration_to_Accept_Offer	Notice_
count	1470.000000	1470.000000	1470.
mean	1024.865306	15.485034	2.
std	602.024335	3.755202	1.
min	1.000000	11.000000	1.
25%	491.250000	12.000000	1.
50%	1020.500000	14.000000	2.
75%	1555.750000	18.000000	3.

```
df.shape
```

```
(1470, 28)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 28 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Candidate_Reference_Number     1470 non-null   int64
 1   Date_of_Joining                1470 non-null   object
 2   Duration_to_Accept_Offer       1470 non-null   int64
 3   Notice_Period                  1470 non-null   int64
 4   Salary_Pay_Band_Offered        1470 non-null   int64
 5   Percentage_Hike_Salary_Expected 1470 non-null  int64
 6   Percentage_Hike_Salary_Offered 1470 non-null   int64
 7   Joining_Bonus                  1470 non-null   int64
 8   Gender                         1470 non-null   object
 9   Candidate_Source               1470 non-null   int64
 10  Relevant_Years_of_Experience   1470 non-null   int64
 11  Line_of_Business               1470 non-null   object
 12  Age                            1470 non-null   int64
 13  Joining_Location               1470 non-null   int64
 14  Candidate_Relocation_Status      1470 non-null   int64
 15  Distance_from_Home               1470 non-null   int64
 16  Education                        1470 non-null   int64
 17  Education_Field                  1470 non-null   object
 18  Job_Role                         1470 non-null   object
 19  Marital_Status                   1470 non-null   object
 20  Number_of_Companies_Worked       1470 non-null   int64
 21  Total_Number_of_Working_Years    1470 non-null   int64
 22  Work_Life_Balance                1470 non-null   int64
 23  Years_In_Current_Role            1470 non-null   int64
 24  Years_Since_Last_Promotion       1470 non-null   int64
 25  Years_With_Current_Company       1470 non-null   int64
 26  HR_Status_Job_Acceptance         1470 non-null   int64
 27  HR_Status_Job_Joined             1470 non-null   int64
dtypes: int64(22), object(6)
memory usage: 321.7+ KB
```

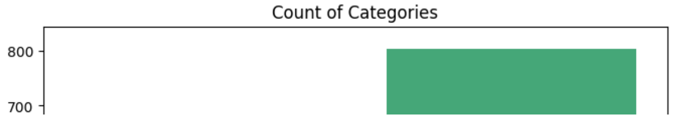
```
 #Here, we check for missing values
df.isnull().sum().any()
```

```
False
```

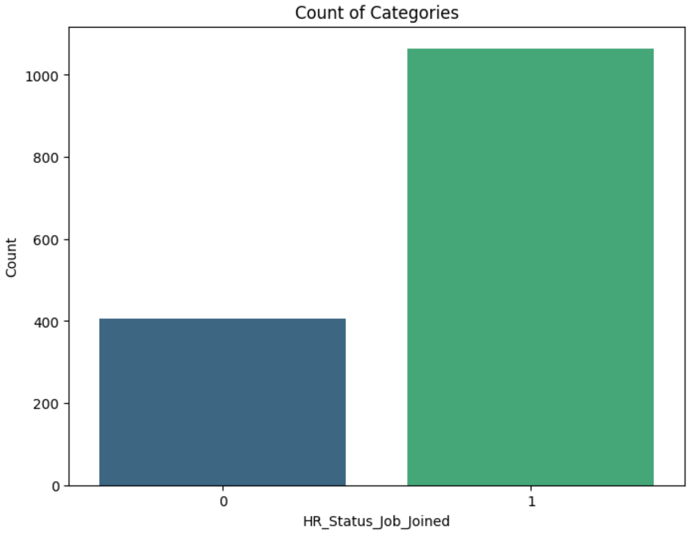
```
df.describe()
```

	Candidate_Reference_Number	Duration_to_Accept_Offer	Notice_
count	1470.000000	1470.000000	1470.
mean	1024.865306	15.485034	2.
std	602.024335	3.755202	1.
min	1.000000	11.000000	1.
25%	491.250000	12.000000	1.
50%	1020.500000	14.000000	2.
75%	1555.750000	18.000000	3.

```
# Count the occurrences of each category
category_counts = df['HR_Status_Job_Acceptance'].value_counts()
# Create a bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x=category_counts.index, y=category_counts.values, palette='viridis')
plt.xlabel('HR_Status_Job_Acceptance')
plt.ylabel('Count')
plt.title('Count of Categories')
plt.show()

category_counts
```

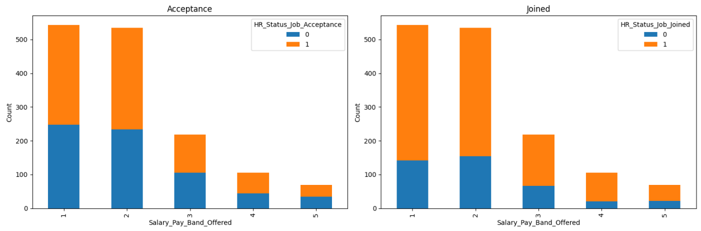


```
# Count the occurrences of each category
category_counts = df['HR_Status_Job_Joined'].value_counts()
# Create a bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x=category_counts.index, y=category_counts.values, palette='viridis')
plt.xlabel('HR_Status_Job_Joined')
plt.ylabel('Count')
plt.title('Count of Categories')
plt.show()

category_counts
```

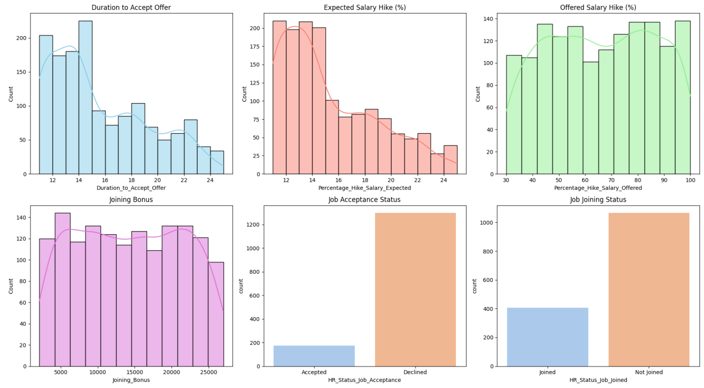


```
1    1065
0     405
Name: HR_Status_Job_Joined, dtype: int64
```

```
# Create a figure with three subplots (one for each categorical variable)
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

cross_tab1 = pd.crosstab(df['Salary_Pay_Band_Offered'], df['HR_Status_Job_Acceptance'])
cross_tab1.plot(kind='bar', stacked=True, ax=axes[0])
axes[0].set_xlabel('Salary_Pay_Band_Offered')
axes[0].set_ylabel('Count')
axes[0].set_title('Acceptance')

cross_tab2 = pd.crosstab(df['Salary_Pay_Band_Offered'], df['HR_Status_Job_Joined'])
cross_tab2.plot(kind='bar', stacked=True, ax=axes[1])
axes[1].set_xlabel('Salary_Pay_Band_Offered')
axes[1].set_ylabel('Count')
axes[1].set_title('Joined')

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```

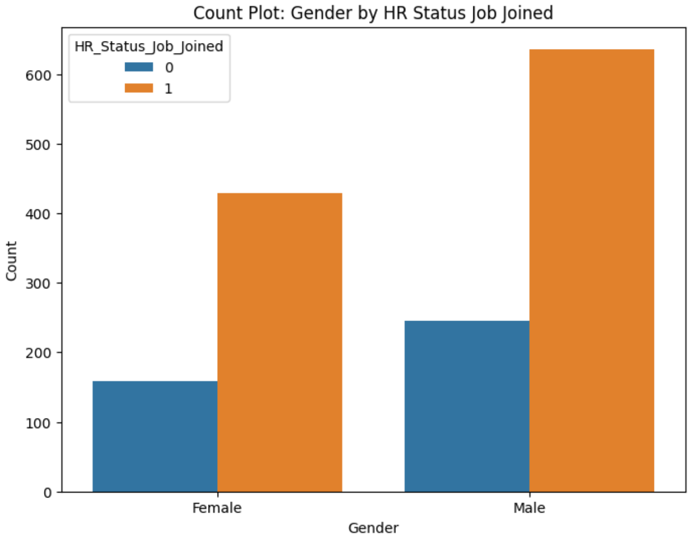


```
# Identify the specific rows where columna has 'value1' and columnb has 'value2'
condition = (df['HR_Status_Job_Acceptance'] == 0) & (df['HR_Status_Job_Joined'] == 1)

# Replace the values in columna with 'value3' in the specific rows
df.loc[condition, 'HR_Status_Job_Acceptance']=1


# Set up the figure and axes
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))

# Plot distribution for key features
sns.histplot(df['Duration_to_Accept_Offer'], ax=axes[0, 0], kde=True, color='skyblue')
axes[0, 0].set_title('Duration to Accept Offer')

sns.histplot(df['Percentage_Hike_Salary_Expected'], ax=axes[0, 1], kde=True, color='salmon')
axes[0, 1].set_title('Expected Salary Hike (%)')

sns.histplot(df['Percentage_Hike_Salary_Offered'], ax=axes[0, 2], kde=True, color='lightgreen')
axes[0, 2].set_title('Offered Salary Hike (%)')

sns.histplot(df['Joining_Bonus'], ax=axes[1, 0], kde=True, color='orchid')
axes[1, 0].set_title('Joining Bonus')

sns.countplot(data=df, x='HR_Status_Job_Acceptance', ax=axes[1, 1], palette="pastel")
axes[1, 1].set_title('Job Acceptance Status')
axes[1, 1].set_xticklabels(['Accepted', 'Declined'])

sns.countplot(data=df, x='HR_Status_Job_Joined', ax=axes[1, 2], palette="pastel")
axes[1, 2].set_title('Job Joining Status')
axes[1, 2].set_xticklabels(['Joined', 'Not Joined'])

plt.tight_layout()
plt.show()
```

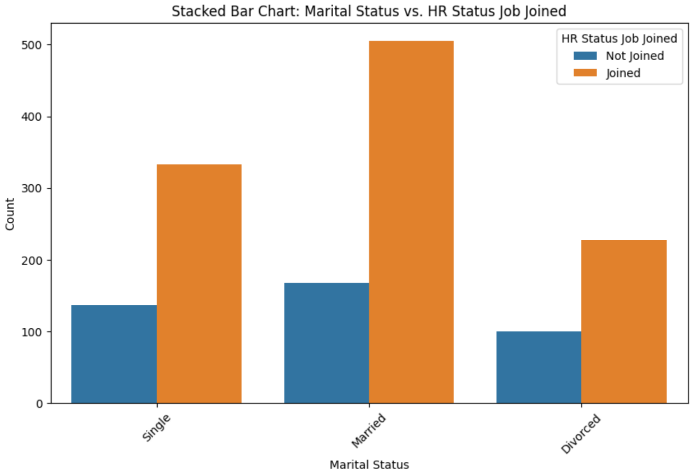


Bivariate Analysis

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Gender', hue='HR_Status_Job_Joined', data=df)
plt.title('Count Plot: Gender by HR Status Job Joined')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```



```
# Example: Stacked bar chart between 'Marital_Status' and 'HR_Status_Job_Joined'
plt.figure(figsize=(10, 6))
sns.countplot(x='Marital_Status', hue='HR_Status_Job_Joined', data=df)
plt.title('Stacked Bar Chart: Marital Status vs. HR Status Job Joined')
plt.xlabel('Marital Status')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='HR Status Job Joined', loc='upper right', labels=['Not Joined', 'Joined'])
plt.show()
```



As we can see from the graph where the percentage of people joining on the basis of marital status is given

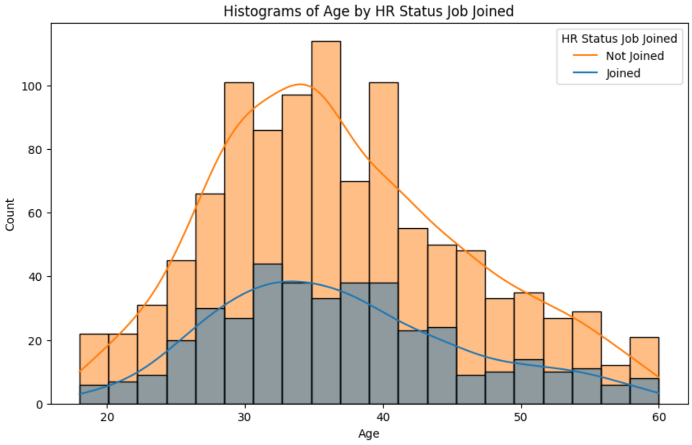
Single - 140/320 = 43.75%

Married - 160/500 =32%

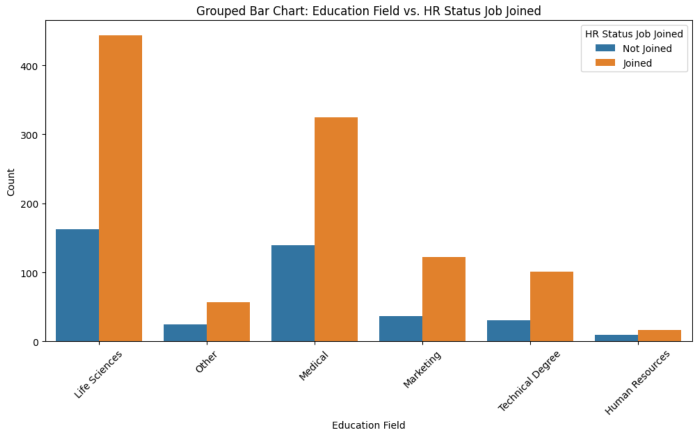
Divorced - 100/220 = 45.45%

This clearly depicts that Single and Divorced people have more chance of joining the Job rather than Married people.

```
#Histograms of 'Age' by HR Status Job Joined
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Age', hue='HR_Status_Job_Joined', kde=True, bins=20)
plt.title('Histograms of Age by HR Status Job Joined')
plt.xlabel('Age')
plt.ylabel('Count')
plt.legend(title='HR Status Job Joined', loc='upper right', labels=['Not Joined', 'Joined'])
plt.show()
```



```
#Grouped bar chart between 'Education_Field' and 'HR_Status_Job_Joined'
plt.figure(figsize=(12, 6))
sns.countplot(x='Education_Field', hue='HR_Status_Job_Joined', data=df)
plt.title('Grouped Bar Chart: Education Field vs. HR Status Job Joined')
plt.xlabel('Education Field')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='HR Status Job Joined', loc='upper right', labels=['Not Joined', 'Joined'])
plt.show()
```

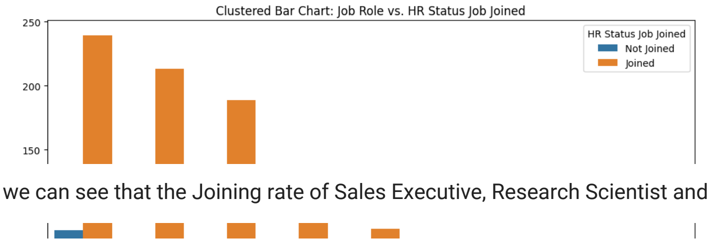


Life Science and Medical have highest join rate as we can see from the graph so promoting people joining from those field would improve the joining rate.

Life Science - 160/450 = 35.5%

Medical - 140/320 = 43.75%

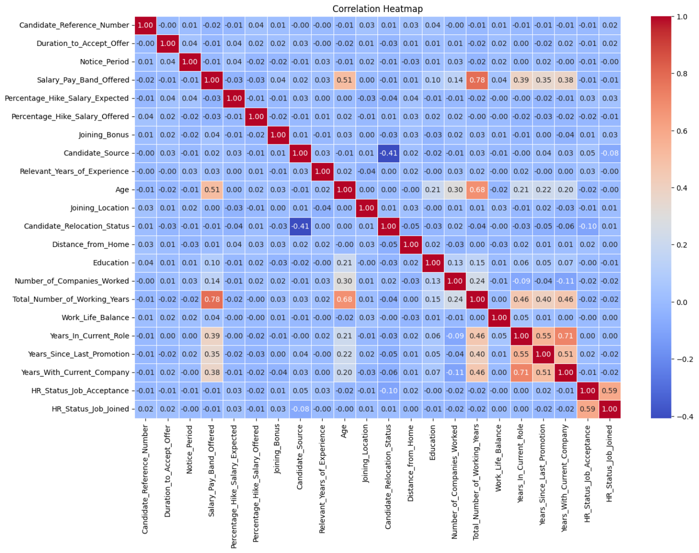
```
#Clustered bar chart between 'Job_Role' and 'HR_Status_Job_Joined'
plt.figure(figsize=(12, 6))
sns.countplot(x='Job_Role', hue='HR_Status_Job_Joined', data=df)
plt.title('Clustered Bar Chart: Job Role vs. HR Status Job Joined')
plt.xlabel('Job Role')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.legend(title='HR Status Job Joined', loc='upper right', labels=['Not Joined', 'Joined'])
plt.show()
```



Here we can see that the Joining rate of Sales Executive, Research Scientist and Laboratory Technician is significantly as compared to others.

```
# Now we compute the correlation matrix
correlation_matrix = df.corr()

# HEATMAP to display the correlations
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5, fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

```
    <ipython-input-151-ea1a571137a2>:2: FutureWarning: The default value
      correlation_matrix = df.corr()
```



As we can see **Total_Number_of_Working_Years** and **Salary_Pay_Band_Offer** have high collinearity possibly because the payscale of employee imcreases as the Number of working year increases. So we will be considering **Total_Number_of Working_Years** as our independent variable and remove **Salary_Pay_Band** ***because number of *working years** influences the **salary_band** and not vice versa.

There is a High Correlation between **Age** and **Total_Number_Of_Working_Years** we will be removing **Age** because **Age** does not have a direct impact on our dependent variable.

There is a High Correlation between **Years_With_Current_Company** and **Years_In_Current_Role** hence as the role has more importance that is why we are removing the **Years_With_Current_Company**

```
#Drop name variable
columns_to_remove = ["Candidate_Reference_Number", "Date_of_Joining", "Salary_Pay_Band_Offered","Age","Years_With_Current_Company"]
df = df.drop(columns=columns_to_remove)
```

```
df.columns
```

```
Index(['Duration_to_Accept_Offer', 'Notice_Period',
       'Percentage_Hike_Salary_Expected', 'Percentage_Hike_Salary_Offered',
       'Joining_Bonus', 'Gender', 'Candidate_Source',
       'Relevant_Years_of_Experience', 'Line_of_Business', 'Joining_Location',
       'Candidate_Relocation_Status', 'Distance_from_Home', 'Education',
       'Education_Field', 'Job_Role', 'Marital_Status',
       'Number_of_Companies_Worked', 'Total_Number_of_Working_Years',
       'Work_Life_Balance', 'Years_In_Current_Role',
       'Years_Since_Last_Promotion', 'HR_Status_Job_Acceptance',
       'HR_Status_Job_Joined'],
      dtype='object')
```

```
#All the Categorical variable converted into dummies
df.columns
df = pd.get_dummies(df, columns=['Joining_Bonus'], drop_first=True)
df = pd.get_dummies(df, columns=['Gender'], drop_first=True)
df = pd.get_dummies(df, columns=['Candidate_Source'], drop_first=True)
df = pd.get_dummies(df, columns=['Line_of_Business'], drop_first=True)
df = pd.get_dummies(df, columns=['Joining_Location'], drop_first=True)
df = pd.get_dummies(df, columns=['Candidate_Relocation_Status'], drop_first=True)
df = pd.get_dummies(df, columns=['Education'], drop_first=True)
df = pd.get_dummies(df, columns=['Education_Field'], drop_first=True)
df = pd.get_dummies(df, columns=['Job_Role'], drop_first=True)
df = pd.get_dummies(df, columns=['Marital_Status'], drop_first=True)
```

```
df.columns
```

```
Index(['Duration_to_Accept_Offer', 'Notice_Period',
       'Percentage_Hike_Salary_Expected', 'Percentage_Hike_Salary_Offered',
       'Relevant_Years_of_Experience', 'Distance_from_Home',
       'Number_of_Companies_Worked', 'Total_Number_of_Working_Years',
       'Work_Life_Balance', 'Years_In_Current_Role',
       ...
       'Job_Role_Human Resources', 'Job_Role_Laboratory Technician',
       'Job_Role_Manager', 'Job_Role_Manufacturing Director',
       'Job_Role_Research Director', 'Job_Role_Research Scientist',
       'Job_Role_Sales Executive', 'Job_Role_Sales Representative',
       'Marital_Status_Married', 'Marital_Status_Single'],
      dtype='object', length=1467)
```

Logistic Regression

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import statsmodels.api as sm
```

```
#Prepare X and Y
y = df.HR_Status_Job_Joined
x = df.drop(columns = "HR_Status_Job_Joined")
x = sm.add_constant(x)
```

```
#Training and Test Split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.2,
                                                    random_state = 15)
```

```
# Standardize the features (optional but can improve model performance)
scaler = StandardScaler()
```

```
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)


# Build and train the logistic regression model
logistic_model = LogisticRegression (random_state=150)
a = logistic_model.fit(x_train, y_train)
```

```
logistic_model.score(x_train, y_train)
```

```
1.0
```

```
logistic_model.fit(x_test, y_test)
```

```
▾         LogisticRegression
LogisticRegression(random_state=150)
```

```
logistic_model.score(x_test, y_test)
```

```
0.891156462585034
```

This shows that the model performs well on the training data with accuracy of 100% but when exposed to the test data set the accuracy decreases to 85% which shows that model qualifies for the condition of overfitting where the model performs well on training data but poorly on test data. Here Class imbalance is one of the factors

```
# Get the coefficients and intercept
coefficients = a.coef_
intercept = a.intercept_

print (coefficients, intercept)
# Calculate the odds ratios for each feature
odds_ratios = np.exp(coefficients)

# Display the odds ratios
print("Odds Ratios:")
for i, coef in enumerate(odds_ratios[0]):
    print(f"Feature{i+1}:{coef}")
```

```
Feature241:0.9290749408884158
Feature242:1.171519497813757
Feature243:0.9999875978678191
Feature244:0.9999875978678191
Feature245:0.9999875978678191
Feature246:1.0392891329263925
Feature247:0.9999875978678191
Feature248:0.9999875978678191
Feature249:0.9999875978678191
Feature250:1.0260154047833703
Feature251:0.8204247107081004
Feature252:1.2341242700567803
Feature253:0.6695971369456329
Feature254:0.9999875978678191
Feature255:0.9999875978678191
Feature256:0.9999875978678191
Feature257:0.9999875978678191
```

```
# Make predictions on the test set
y_pred = logistic_model.predict(x_test)
print(y_pred)
```

```
[1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1
 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1
 1 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 0 1 1
 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1
 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 0]
```

```
# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
tn, fp, fn, tp = conf_matrix.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
accuracy = (tp + tn) / (tp + tn + fp + fn)

print(f"Sensitivity (True Positive Rate): {sensitivity:.2f}")
print(f"Specificity (True Negative Rate): {specificity:.2f}")
print(f"Accuracy: {accuracy:.2f}")
```

```
Sensitivity (True Positive Rate): 0.99
Specificity (True Negative Rate): 0.63
Accuracy: 0.89
```

```
# Display results
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(confusion)
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 0.89
Confusion Matrix:
[[ 51  30]
 [  2 211]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.63      0.76        81
           1       0.88      0.99      0.93       213

    accuracy                           0.89       294
   macro avg       0.92      0.81      0.85       294
weighted avg       0.90      0.89      0.88       294
```

Bias Variance

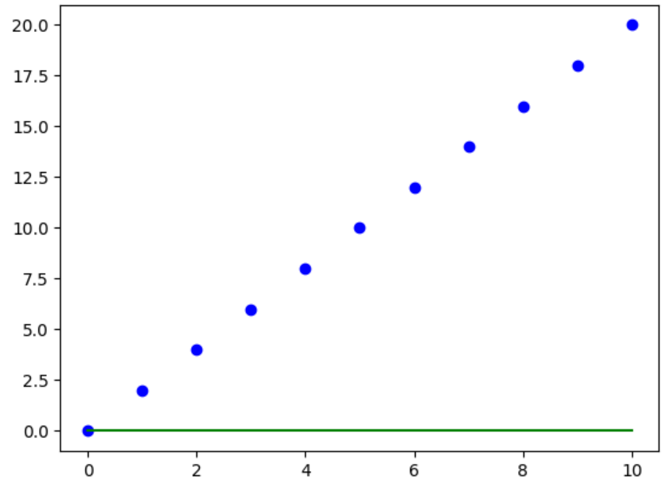
```
bias = np.mean(y_pred - y_test)
variance = np.var(y_pred-y_test)

print('Bias :',bias)
print('Variance :',variance)
```

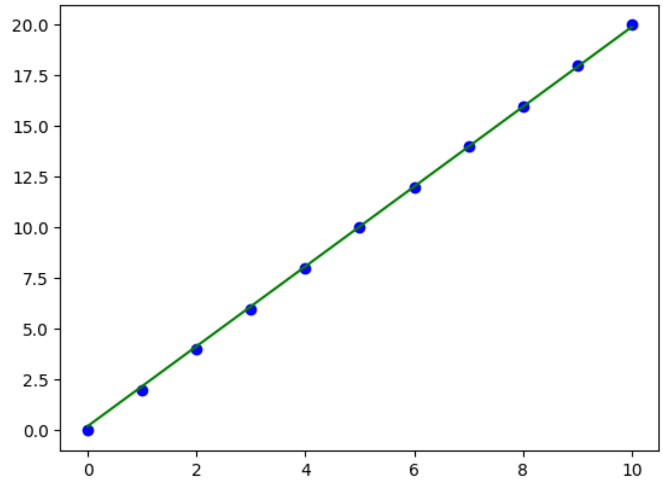
```
Bias : 0.09523809523809523
Variance : 0.0997732426303855
```

Gradient Descent

```
# Implementation of gradient descent in linear regression
import numpy as np
import matplotlib.pyplot as plt
class Linear_Regression:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.b = [0, 0]
    def update_coeffs(self, learning_rate):
        y_pred = self.predict()
        y = self.y
        m = len(y)
        self.b[0] = self.b[0] - (learning_rate * ((1/m) *
                                                  np.sum(y_pred - y)))
        self.b[1] = self.b[1] - (learning_rate * ((1/m) *
                                                  np.sum((y_pred - y) * self.x)))
    def predict(self, x=[]):
        y_pred = np.array([])
        if not x:
            x = self.x
        b = self.b
        for x in x:
            y_pred = np.append(y_pred, b[0] + (b[1] * x))
        return y_pred
    def get_current_accuracy(self, y_pred):
        p, e = y_pred, self.y
        n = len(y_pred)
        return 1-sum(
            [
                abs(p[i]-e[i])/e[i]
                for i in range(n)
                if e[i] != 0]
        )/n
    # def predict(self, b, yi):

    def compute_cost(self, y_pred):
        m = len(self.y)
        J = (1 / 2*m) * (np.sum(y_pred - self.y)**2)
        return J
    def plot_best_fit(self, y_pred, fig):
        f = plt.figure(fig)
        plt.scatter(self.x, self.y, color='b')
        plt.plot(self.x, y_pred, color='g')
        f.show()
def main():
    x = np.array([i for i in range(11)])
    y = np.array([2*i for i in range(11)])
    regressor = Linear_Regression(x, y)
    iterations = 0
    steps = 100
    learning_rate = 0.01
    costs = []
    # original best-fit line
    y_pred = regressor.predict()
    regressor.plot_best_fit(y_pred, 'Initial Best Fit Line')
    while 1:
        y_pred = regressor.predict()
        cost = regressor.compute_cost(y_pred)
        costs.append(cost)
        regressor.update_coeffs(learning_rate)
        iterations += 1
        if iterations % steps == 0:
            print(iterations, "epochs elapsed")
            print("Current accuracy is :",
                  regressor.get_current_accuracy(y_pred))
            stop = input("Do you want to stop (y/*)??")
            if stop == "y":
                break
    # final best-fit line
    regressor.plot_best_fit(y_pred, 'Final Best Fit Line')

    # plot to verify cost function decreases
    h = plt.figure('Verification')
    plt.plot(range(iterations), costs, color='b')
    h.show()
    plt.show()

    # if user wants to predict using the regressor:
    regressor.predict([i for i in range(10)])


if __name__ == '__main__':
    main()
```

```
100 epochs elapsed
Current accuracy is : 0.9836456109008862
Do you want to stop (y/*)??y
```





60000
50000
40000
30000
20000
10000