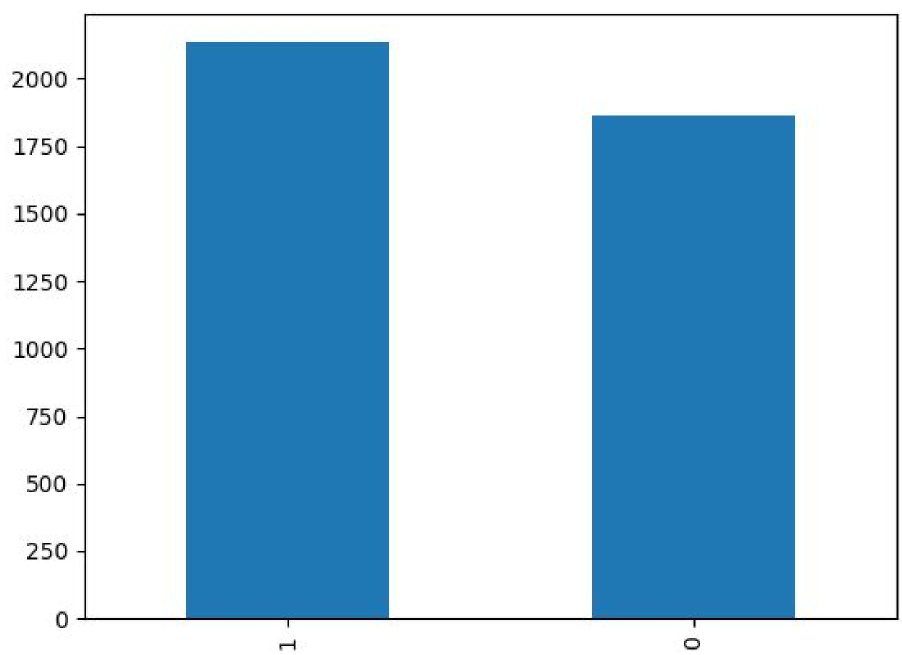
PROJECT REPORT

Rupanshu Yadav
2019475

Steps taken in analyzing and preprocessing data :

1. Checking for any significant imbalance in the type of student's with high salary vs type of students without high-salary

   CountPlot for no. of students with high-salary (1) *vs without high-salary* (0)



Since there is no significant difference no change is made to the data

2. Judging the relevance of ID in the data

```
In [1552]: dataset['ID'].value_counts()

Out[1552]: 337919     1
           831000     1
           775902     1
           330544     1
           700535     1
                     ..
           331131     1
           230778     1
           124278     1
           963956     1
           1245184    1
           Name: ID, Length: 3998, dtype: int64
```

Since all the IDs are distinct ID doesn't play a role in a student getting a high salary or not, so this column is eliminated from the dataset

```
dataset.drop('ID', axis = 1)
```

3. Mapping gender male: 1 and female : 0

To convert gender into numbers we convert all the 'm' in data to 1 and all 'f' to 0

```
dataset['Gender'] = dataset['Gender'].map(dict(m=1, f=0))
```

4. Convert all Object data type into int64 or float64 using LabelEncoder:

```
In [3458]: dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3998 entries, 0 to 3997
Data columns (total 34 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID                  3998 non-null   int64
 1   Gender              3998 non-null   object
 2   DOB                 3998 non-null   object
 3   10percentage        3998 non-null   float64
 4   10board             3998 non-null   object
 5   12graduation        3998 non-null   int64
 6   12percentage        3998 non-null   float64
 7   12board             3998 non-null   object
 8   CollegeID           3998 non-null   int64
 9   CollegeTier         3998 non-null   int64
 10  Degree              3998 non-null   object
 11  Specialization      3998 non-null   object
 12  collegeGPA          3998 non-null   float64
 13  CollegeCityID       3998 non-null   int64
```

Code :

```
label1 = LabelEncoder()
for (columnName, columnData) in dataset.iteritems():
    if dataset[columnName].dtype == object:
        dataset[columnName] = label1.fit_transform(dataset[columnName])
```

After Execution :

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3998 entries, 0 to 3997
Data columns (total 34 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               3998 non-null   int64
 1   Gender           3998 non-null   int64
 2   DOB              3998 non-null   int64
 3   10percentage     3998 non-null   float64
 4   10board          3998 non-null   int64
 5   12graduation     3998 non-null   int64
 6   12percentage     3998 non-null   float64
 7   12board          3998 non-null   int64
 8   CollegeID        3998 non-null   int64
 9   CollegeTier      3998 non-null   int64
 10  Degree           3998 non-null   int64
```

5. Converting all the -1's in data to 0

```
for (columnName, columnData) in dataset.iteritems():
        if dataset[columnName].min() == -1 and columnName != 'Domain':
            dataset[columnName].replace([-1], [0], inplace=True)
```

Doing this increases the accuracy by 0.0040(approx, where accuracy is of form 0.xxxx)

6. When analyzing the '10Board' we found the there are many students with distinct boards, grouping of data is performed on this column due to which accuracy increases from 0.7277 to 0.7331

```
s = dataset['10board'].value_counts()
dataset['10board'] = np.where(dataset['10board'].isin(s.index[s > 1]), dataset['10board'], 0)
```

7. Dividing data into a test set and training set. Since the classes are almost balanced already we don't need to balance classes.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.6105, random_state=43)
```

After Binary Searching the ratio 0.6105 is found as the optimal choice

8. Scaling of data is performed it is important to scale the features to a range that is centered around zero. This is done so that the variance of the features are in the same range. If a feature's variance is orders of magnitude more than the variance of other features, that particular feature might dominate other features in the dataset, which is not something we want happening in our model.

   To do this StandardScaler() from Sklearn is used:

```
sc = StandardScaler();
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

9. Now to select the most optimal columns in the data a technique called Recursive Feature Elimination is used. The goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features.

   Now to choose how many columns to select in the final regression Binary Search is again performed to find the optimal number of columns to choose out of all the choices and the optimal choice is 20.

Some of the results are :

19 -> accuracy of 0.7309

20 -> accuracy of 0.7331

21 -> accuracy of 0.7250

So we can see 20 is the optimal choice

Code:

```
from sklearn.feature_selection import RFE
logreg = LogisticRegression()
rfe = RFE(logreg, 20)
rfe = rfe.fit(X_train, y_train.values.ravel())
print(rfe.support_)
cols = list((rfe.support_))
col = []
before = []
for (columnName, columnData) in dataset.iteritems():
    before.append(columnName)

for i in range(len(cols)):
    if (cols[i]):
        col.append(before[i])
```

10. Then again after choosing the columns data is divided into test sets and training sets and scaled using a standard scaler.

    This time tho the optimal ratio for splitting into test and training might be different so again Binary Search is performed to get the optimal ratio which turns out to be 0.325 (this optimal ratio might depend upon the number of columns choosen in RFE, this result is for 20 columns)

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.325, random_state=43)

sc = StandardScaler();
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
logreg.fit(X_train, y_train)
```

Now After preprocessing and regressing the date is done it’s time to determine how good the data is using various techniques discussed below.

1. **Accuracy**

   code:

```
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set: {:.4f}'.format(logreg.score(X_test, y_test)))
```

   Result:

```
Accuracy of logistic regression classifier on test set: 0.7331
```

2. **Confusion Matrix**

   Definition :

Predicted Values		Actual Values	
		Positive (1)	Negative (0)
	Positive (1)	TP	FP
	Negative (0)	FN	TN

   Result:

```
In [911]: from sklearn.metrics import confusion_matrix
          confusion_matrix(y_test, y_pred)

Out[911]: array([[437, 182],
                 [165, 516]])
```

3. **Class-wise-accuracy**

Print the classification report of the result

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.71      0.72       619
           1       0.74      0.76      0.75       681

    accuracy                           0.73      1300
   macro avg       0.73      0.73      0.73      1300
weighted avg       0.73      0.73      0.73      1300
```

4. **Receiver operating characteristic curve**

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible (toward the top-left corner).