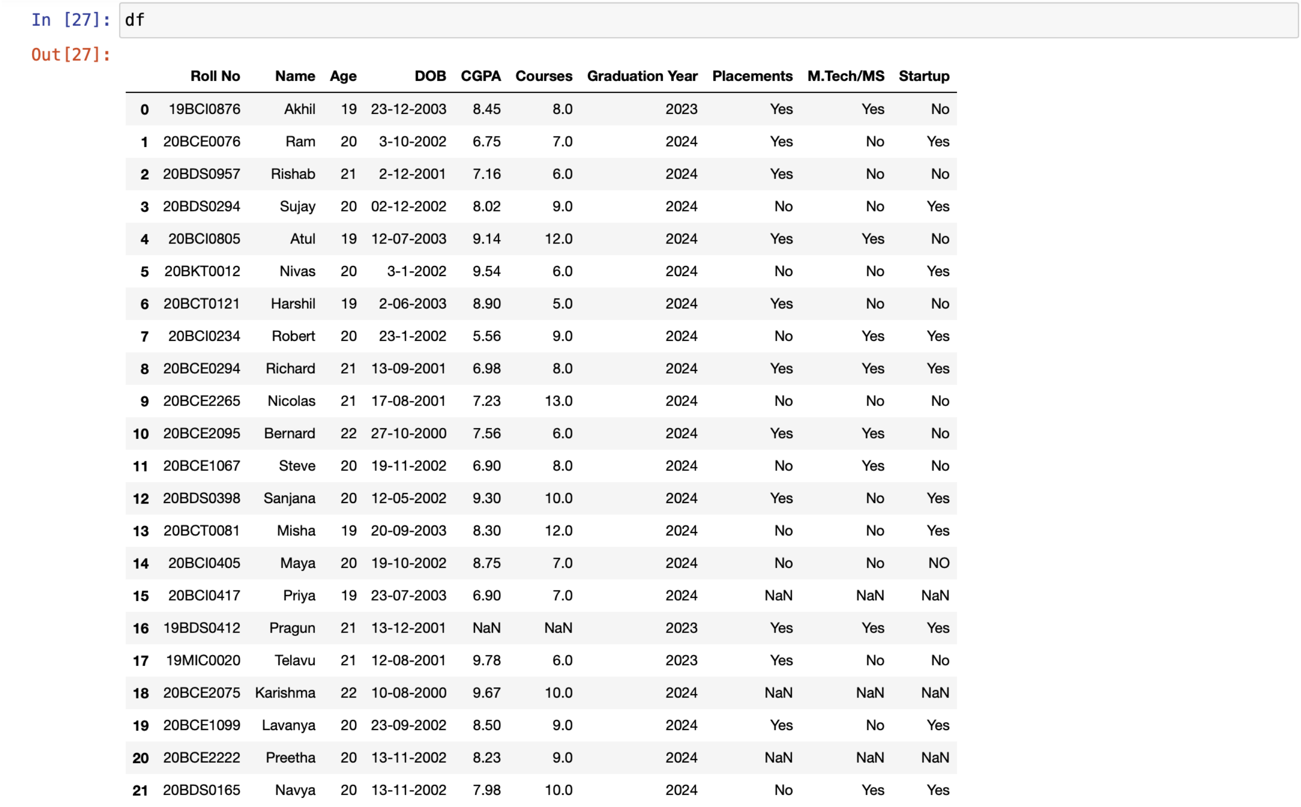
CSE 4020 Machine Learning
Lab Assessment - 5

Sujay Kumar M 20BDS0294
Computer Science Engineering with Specialization with DataScience
sujaykumarreddy.m2020@vitstudent.ac.in
https://github.com/sujaykumarmag/CSE4020

April 3, 2023

1 Dataset

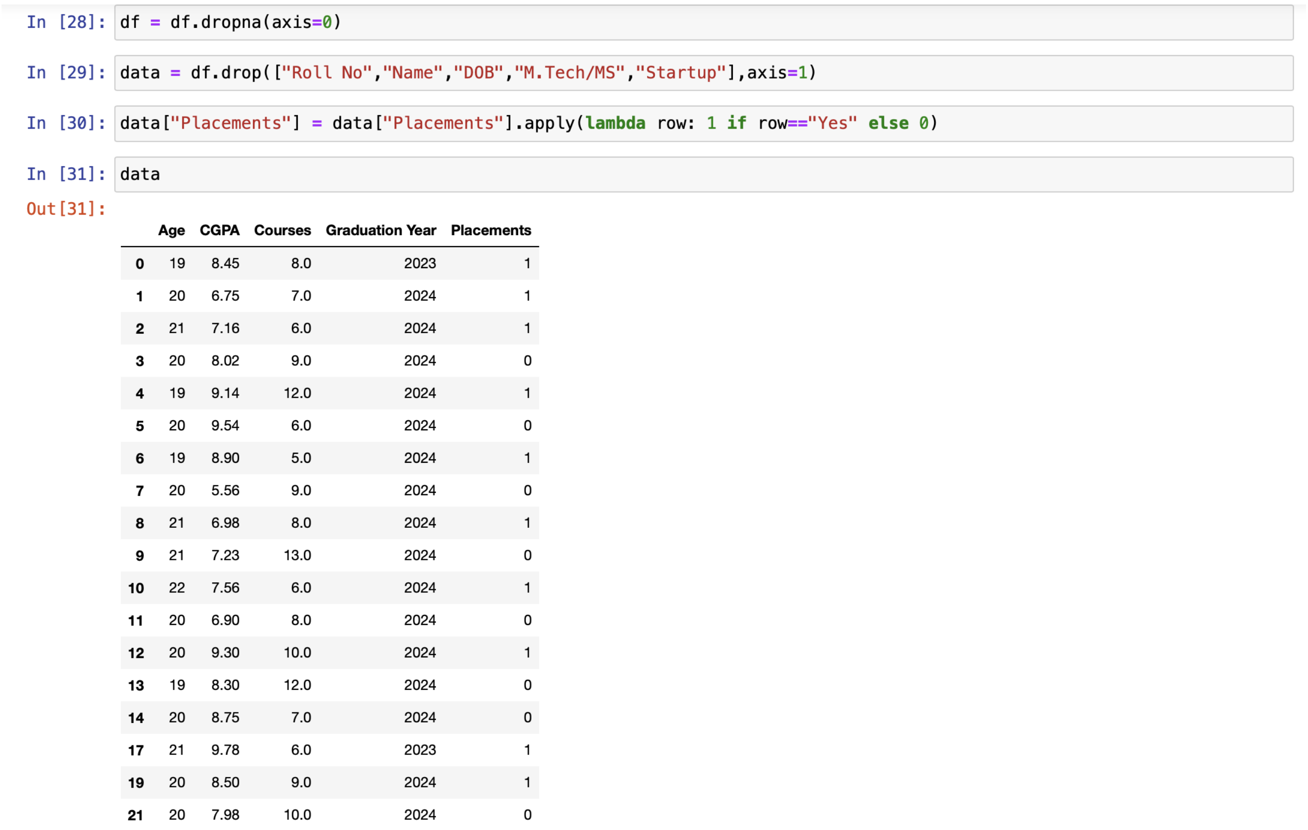


```
In [27]: df
Out[27]:
```

	Roll No	Name	Age	DOB	CGPA	Courses	Graduation Year	Placements	M.Tech/MS	Startup
0	19BCI0876	Akhil	19	23-12-2003	8.45	8.0	2023	Yes	Yes	No
1	20BCE0076	Ram	20	3-10-2002	6.75	7.0	2024	Yes	No	Yes
2	20BDS0957	Rishab	21	2-12-2001	7.16	6.0	2024	Yes	No	No
3	20BDS0294	Sujay	20	02-12-2002	8.02	9.0	2024	No	No	Yes
4	20BCI0805	Atul	19	12-07-2003	9.14	12.0	2024	Yes	Yes	No
5	20BKT0012	Nivas	20	3-1-2002	9.54	6.0	2024	No	No	Yes
6	20BCT0121	Harshil	19	2-06-2003	8.90	5.0	2024	Yes	No	No
7	20BCI0234	Robert	20	23-1-2002	5.56	9.0	2024	No	Yes	Yes
8	20BCE0294	Richard	21	13-09-2001	6.98	8.0	2024	Yes	Yes	Yes
9	20BCE2265	Nicolas	21	17-08-2001	7.23	13.0	2024	No	No	No
10	20BCE2095	Bernard	22	27-10-2000	7.56	6.0	2024	Yes	Yes	No
11	20BCE1067	Steve	20	19-11-2002	6.90	8.0	2024	No	Yes	No
12	20BDS0398	Sanjana	20	12-05-2002	9.30	10.0	2024	Yes	No	Yes
13	20BCT0081	Misha	19	20-09-2003	8.30	12.0	2024	No	No	Yes
14	20BCI0405	Maya	20	19-10-2002	8.75	7.0	2024	No	No	NO
15	20BCI0417	Priya	19	23-07-2003	6.90	7.0	2024	NaN	NaN	NaN
16	19BDS0412	Pragun	21	13-12-2001	NaN	NaN	2023	Yes	Yes	Yes
17	19MIC0020	Telavu	21	12-08-2001	9.78	6.0	2023	Yes	No	No
18	20BCE2075	Karishma	22	10-08-2000	9.67	10.0	2024	NaN	NaN	NaN
19	20BCE1099	Lavanya	20	23-09-2002	8.50	9.0	2024	Yes	No	Yes
20	20BCE2222	Preetha	20	13-11-2002	8.23	9.0	2024	NaN	NaN	NaN
21	20BDS0165	Navya	20	13-11-2002	7.98	10.0	2024	No	Yes	Yes

2 PreProcessing

```

df = df.dropna(axis=0)
data = df.drop(["Roll No","Name","DOB","M.Tech/MS","Startup"],axis=1)
data["Placements"] = data["Placements"].apply(lambda row: 1 if row=="Yes" else 0)
```



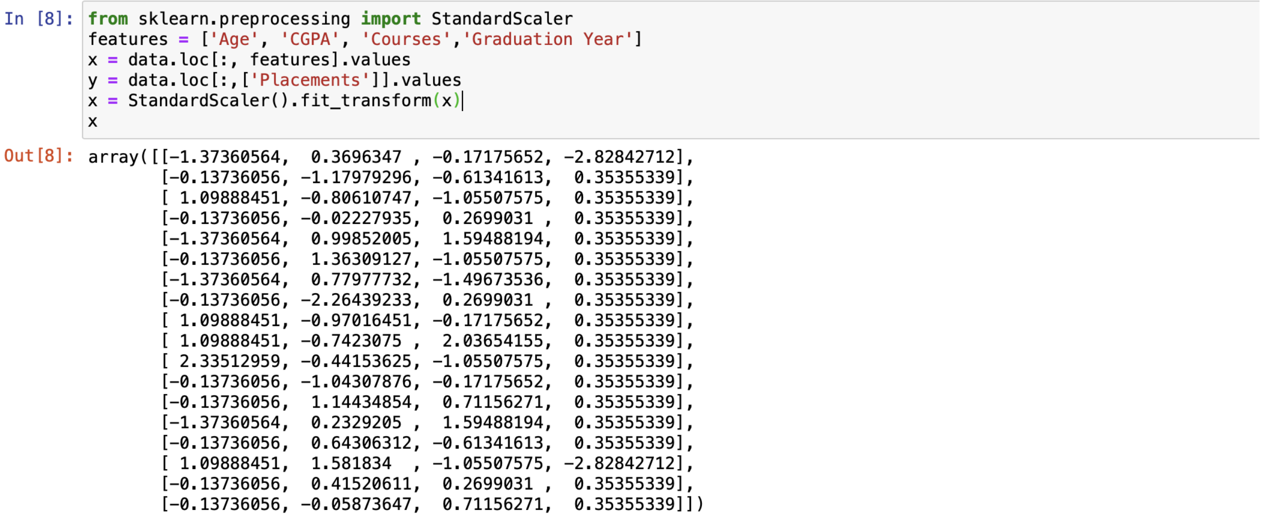
```
In [28]: df = df.dropna(axis=0)

In [29]: data = df.drop(["Roll No","Name","DOB","M.Tech/MS","Startup"],axis=1)

In [30]: data["Placements"] = data["Placements"].apply(lambda row: 1 if row=="Yes" else 0)

In [31]: data
```

Out[31]:

	Age	CGPA	Courses	Graduation Year	Placements
0	19	8.45	8.0	2023	1
1	20	6.75	7.0	2024	1
2	21	7.16	6.0	2024	1
3	20	8.02	9.0	2024	0
4	19	9.14	12.0	2024	1
5	20	9.54	6.0	2024	0
6	19	8.90	5.0	2024	1
7	20	5.56	9.0	2024	0
8	21	6.98	8.0	2024	1
9	21	7.23	13.0	2024	0
10	22	7.56	6.0	2024	1
11	20	6.90	8.0	2024	0
12	20	9.30	10.0	2024	1
13	19	8.30	12.0	2024	0
14	20	8.75	7.0	2024	0
17	21	9.78	6.0	2023	1
19	20	8.50	9.0	2024	1
21	20	7.98	10.0	2024	0

3 Applying Standard Scalar for PCA

```
from sklearn.preprocessing import StandardScaler
features = ['Age', 'CGPA', 'Courses','Graduation Year']
x = data.loc[:, features].values
y = data.loc[:,['Placements']].values
x = StandardScaler().fit_transform(x)
x
```

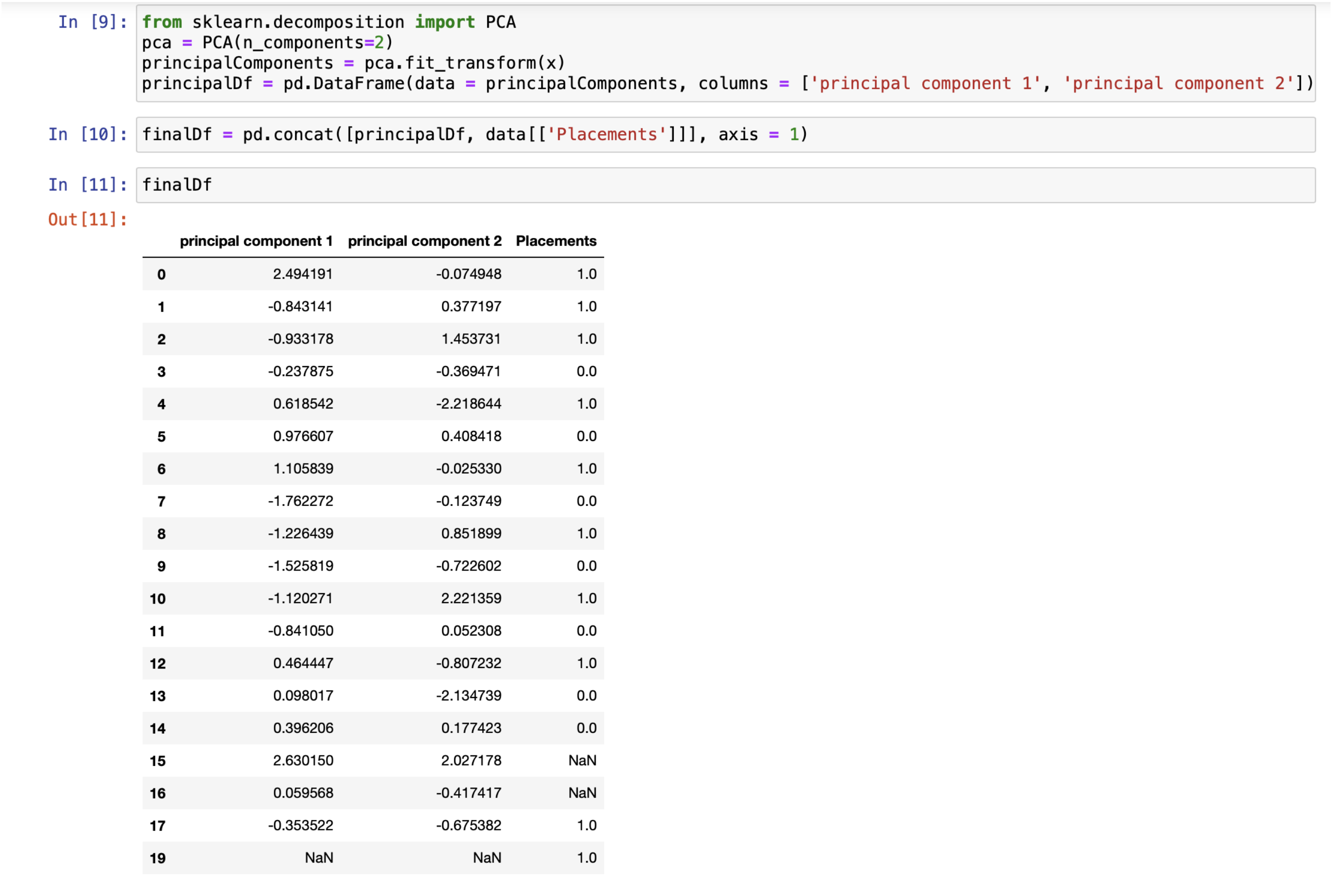


```
In [8]: from sklearn.preprocessing import StandardScaler
        features = ['Age', 'CGPA', 'Courses','Graduation Year']
        x = data.loc[:, features].values
        y = data.loc[:,['Placements']].values
        x = StandardScaler().fit_transform(x)
        x

Out[8]: array([[-1.37360564,  0.3696347 , -0.17175652, -2.82842712],
               [-0.13736056, -1.17979296, -0.61341613,  0.35355339],
               [ 1.09888451, -0.80610747, -1.05507575,  0.35355339],
               [-0.13736056, -0.02227935,  0.2699031 ,  0.35355339],
               [-1.37360564,  0.99852005,  1.59488194,  0.35355339],
               [-0.13736056,  1.36309127, -1.05507575,  0.35355339],
               [-1.37360564,  0.77977732, -1.49673536,  0.35355339],
               [-0.13736056, -2.26439233,  0.2699031 ,  0.35355339],
               [ 1.09888451, -0.97016451, -0.17175652,  0.35355339],
               [ 1.09888451, -0.7423075 ,  2.03654155,  0.35355339],
               [ 2.33512959, -0.44153625, -1.05507575,  0.35355339],
               [-0.13736056, -1.04307876, -0.17175652,  0.35355339],
               [-0.13736056,  1.14434854,  0.71156271,  0.35355339],
               [-1.37360564,  0.2329205 ,  1.59488194,  0.35355339],
               [-0.13736056,  0.64306312, -0.61341613,  0.35355339],
               [ 1.09888451,  1.581834  , -1.05507575, -2.82842712],
               [-0.13736056,  0.41520611,  0.2699031 ,  0.35355339],
               [-0.13736056, -0.05873647,  0.71156271,  0.35355339]])
```

4 Applying PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
```

```
principalDf = pd.DataFrame(data = principalComponents , columns = ['principal component
  1', 'principal component 2'])
finalDf = pd.concat([principalDf , data[['Placements']]], axis = 1)
finalDf = finalDf.dropna(axis=0)
```



In [9]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])
```

In [10]:
```
finalDf = pd.concat([principalDf, data[['Placements']]], axis = 1)
```

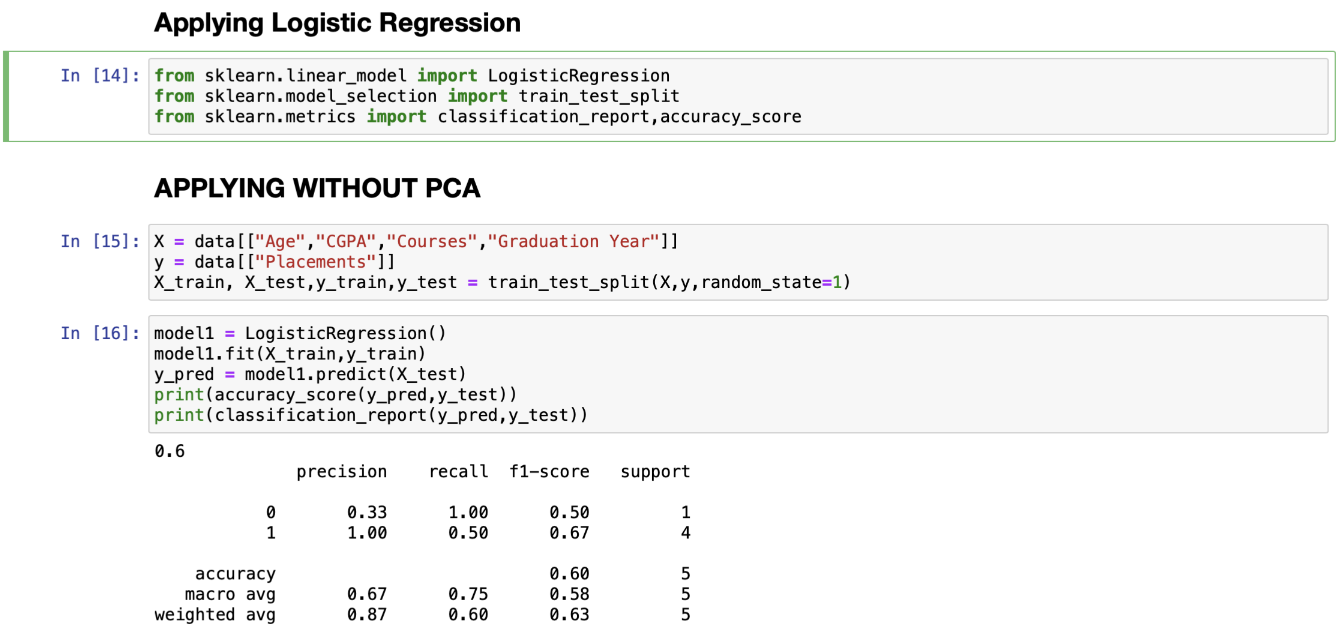
In [11]:
```
finalDf
```

Out[11]:

	principal component 1	principal component 2	Placements
0	2.494191	-0.074948	1.0
1	-0.843141	0.377197	1.0
2	-0.933178	1.453731	1.0
3	-0.237875	-0.369471	0.0
4	0.618542	-2.218644	1.0
5	0.976607	0.408418	0.0
6	1.105839	-0.025330	1.0
7	-1.762272	-0.123749	0.0
8	-1.226439	0.851899	1.0
9	-1.525819	-0.722602	0.0
10	-1.120271	2.221359	1.0
11	-0.841050	0.052308	0.0
12	0.464447	-0.807232	1.0
13	0.098017	-2.134739	0.0
14	0.396206	0.177423	0.0
15	2.630150	2.027178	NaN
16	0.059568	-0.417417	NaN
17	-0.353522	-0.675382	1.0
19	NaN	NaN	1.0

5 Applying Logistic Regression Without PCA

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,accuracy_score


X = data[["Age","CGPA","Courses","Graduation Year"]]
y = data[["Placements"]]
X_train, X_test,y_train,y_test = train_test_split(X,y,random_state=1)

model1 = LogisticRegression()
model1.fit(X_train,y_train)
y_pred = model1.predict(X_test)
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))


```



Applying Logistic Regression

```
In [14]: from sklearn.linear_model import LogisticRegression
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import classification_report,accuracy_score
```

APPLYING WITHOUT PCA

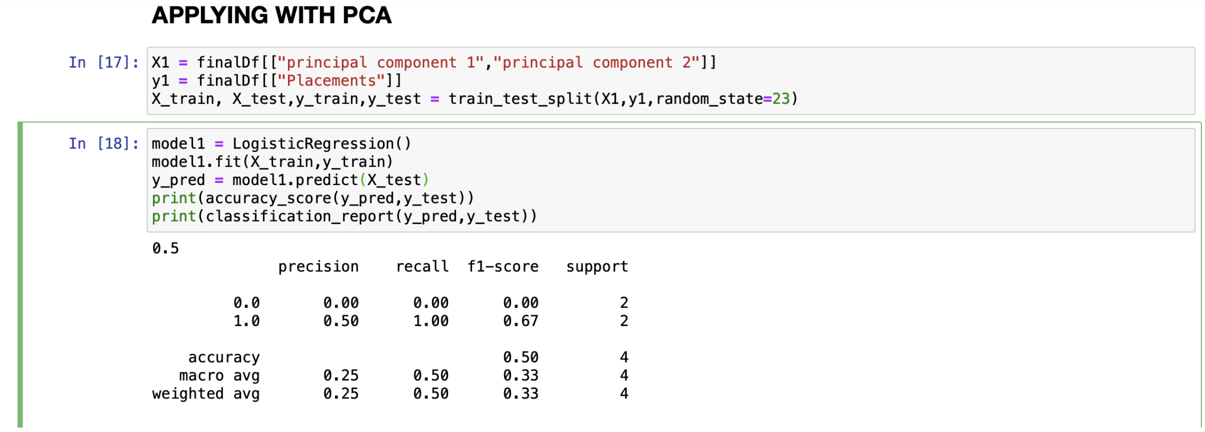
```
In [15]: X = data[["Age","CGPA","Courses","Graduation Year"]]
         y = data[["Placements"]]
         X_train, X_test,y_train,y_test = train_test_split(X,y,random_state=1)
```

```
In [16]: model1 = LogisticRegression()
         model1.fit(X_train,y_train)
         y_pred = model1.predict(X_test)
         print(accuracy_score(y_pred,y_test))
         print(classification_report(y_pred,y_test))
```

```
0.6
              precision    recall  f1-score   support

           0       0.33      1.00      0.50         1
           1       1.00      0.50      0.67         4

    accuracy                           0.60         5
   macro avg       0.67      0.75      0.58         5
weighted avg       0.87      0.60      0.63         5
```

6 Applying Logistic Regression with PCA

```
X1 = finalDf[["principal component 1","principal component 2"]]
y1 = finalDf[["Placements"]]
X_train, X_test,y_train,y_test = train_test_split(X1,y1,random_state=23)

model1 = LogisticRegression()
model1.fit(X_train,y_train)
y_pred = model1.predict(X_test)
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))

```



APPLYING WITH PCA

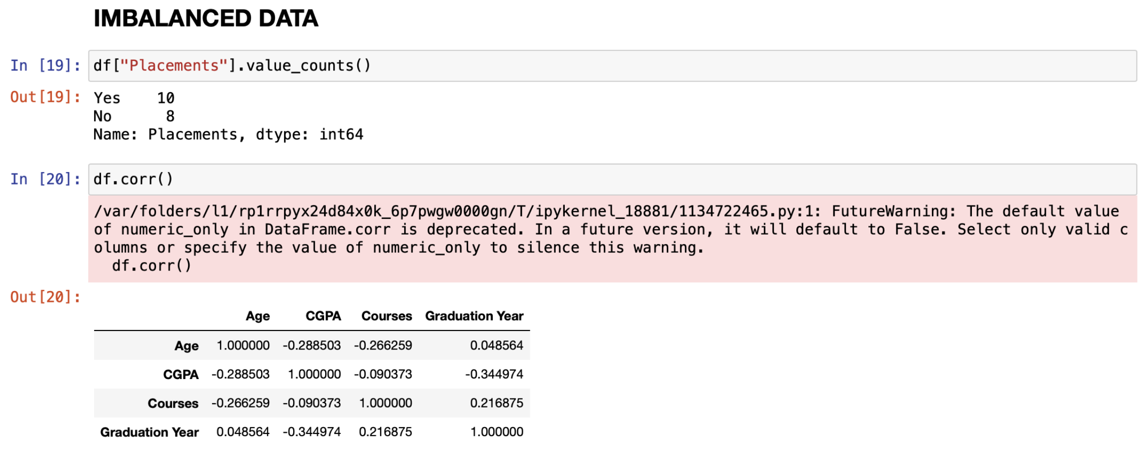
```
In [17]: X1 = finalDf[["principal component 1","principal component 2"]]
         y1 = finalDf[["Placements"]]
         X_train, X_test,y_train,y_test = train_test_split(X1,y1,random_state=23)
```

```
In [18]: model1 = LogisticRegression()
         model1.fit(X_train,y_train)
         y_pred = model1.predict(X_test)
         print(accuracy_score(y_pred,y_test))
         print(classification_report(y_pred,y_test))
```

```
0.5
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00         2
         1.0       0.50      1.00      0.67         2

    accuracy                           0.50         4
   macro avg       0.25      0.50      0.33         4
weighted avg       0.25      0.50      0.33         4
```

7 Taking Care of Imbalanced data

```
df["Placements"].value_counts()
```



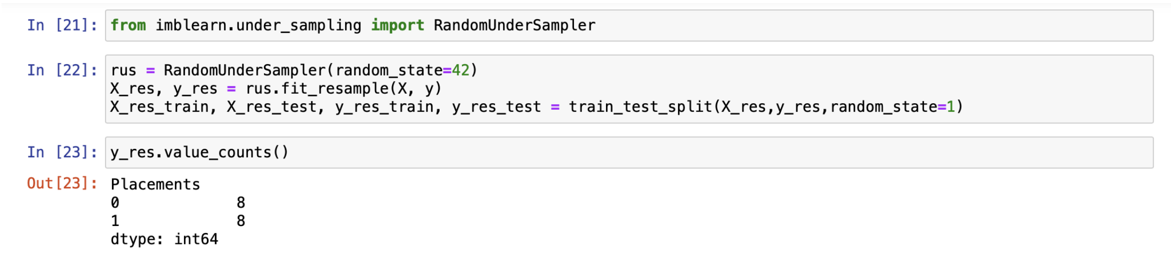
IMBALANCED DATA

```
In [19]: df["Placements"].value_counts()

Out[19]: Yes    10
         No      8
         Name: Placements, dtype: int64

In [20]: df.corr()

/var/folders/l1/rp1rrpyx24d84x0k_6p7pwgw0000gn/T/ipykernel_18881/1134722465.py:1: FutureWarning: The default value
of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid c
olumns or specify the value of numeric_only to silence this warning.
  df.corr()

Out[20]:
```

	Age	CGPA	Courses	Graduation Year
Age	1.000000	-0.288503	-0.266259	0.048564
CGPA	-0.288503	1.000000	-0.090373	-0.344974
Courses	-0.266259	-0.090373	1.000000	0.216875
Graduation Year	0.048564	-0.344974	0.216875	1.000000

8 Undersampling

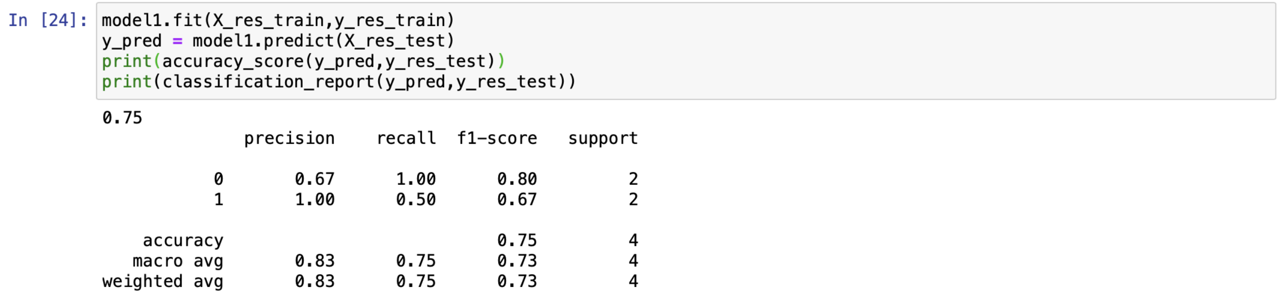
```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42)

X_res, y_res = rus.fit_resample(X, y)
X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res, random_state=1)


y_res.value_counts()
```



```
In [21]: from imblearn.under_sampling import RandomUnderSampler

In [22]: rus = RandomUnderSampler(random_state=42)
         X_res, y_res = rus.fit_resample(X, y)
         X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res,random_state=1)

In [23]: y_res.value_counts()

Out[23]: Placements
         0             8
         1             8
         dtype: int64
```

9 Undersampling Without PCA

```
model1.fit(X_res_train,y_res_train)
y_pred = model1.predict(X_res_test)
print(accuracy_score(y_pred,y_res_test))
print(classification_report(y_pred,y_res_test))
```

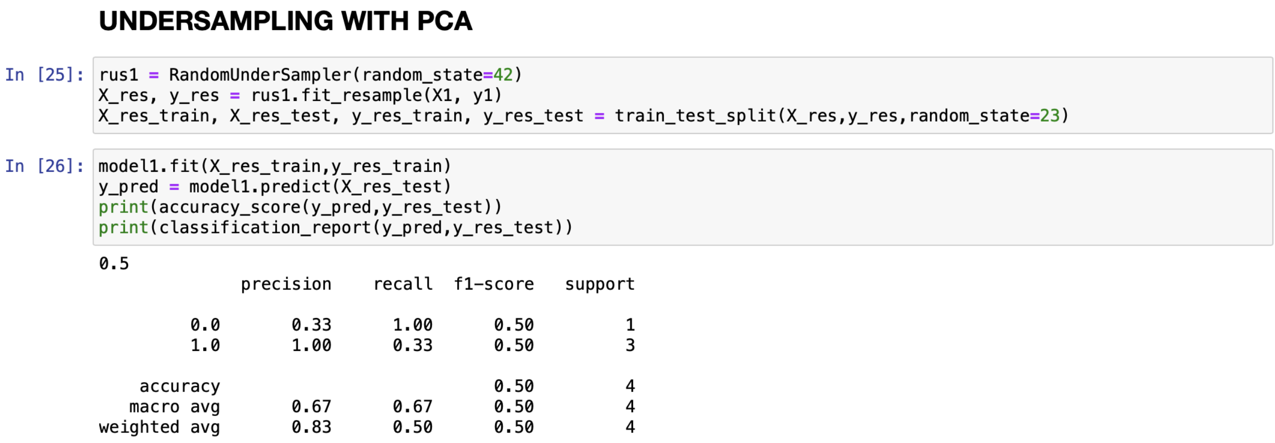


```
In [24]: model1.fit(X_res_train,y_res_train)
         y_pred = model1.predict(X_res_test)
         print(accuracy_score(y_pred,y_res_test))
         print(classification_report(y_pred,y_res_test))

         0.75
                       precision    recall  f1-score   support

                    0       0.67      1.00      0.80         2
                    1       1.00      0.50      0.67         2

             accuracy                           0.75         4
            macro avg       0.83      0.75      0.73         4
         weighted avg       0.83      0.75      0.73         4
```

10 Undersampling With PCA

```
rus1 = RandomUnderSampler(random_state=42)
X_res, y_res = rus1.fit_resample(X1, y1)
```

```python
X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res, random_state=23)


model1.fit(X_res_train,y_res_train)
y_pred = model1.predict(X_res_test)
print(accuracy_score(y_pred,y_res_test))
print(classification_report(y_pred,y_res_test))

```

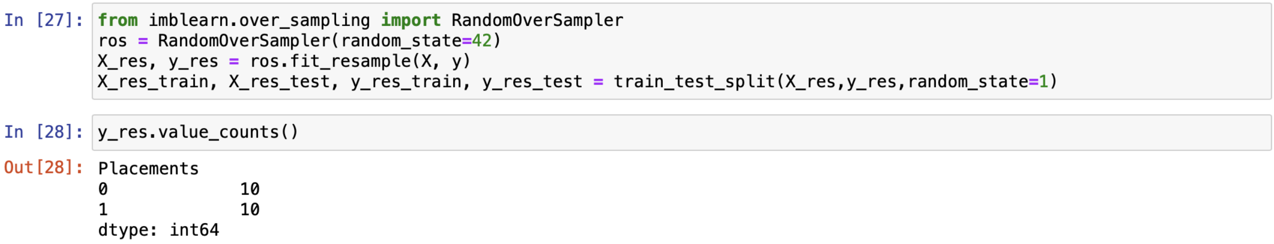


UNDERSAMPLING WITH PCA

```
In [25]: rus1 = RandomUnderSampler(random_state=42)
         X_res, y_res = rus1.fit_resample(X1, y1)
         X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res,random_state=23)

In [26]: model1.fit(X_res_train,y_res_train)
         y_pred = model1.predict(X_res_test)
         print(accuracy_score(y_pred,y_res_test))
         print(classification_report(y_pred,y_res_test))

         0.5
                       precision    recall  f1-score   support

                  0.0       0.33      1.00      0.50         1
                  1.0       1.00      0.33      0.50         3

             accuracy                           0.50         4
            macro avg       0.67      0.67      0.50         4
         weighted avg       0.83      0.50      0.50         4
```

11 Oversampling

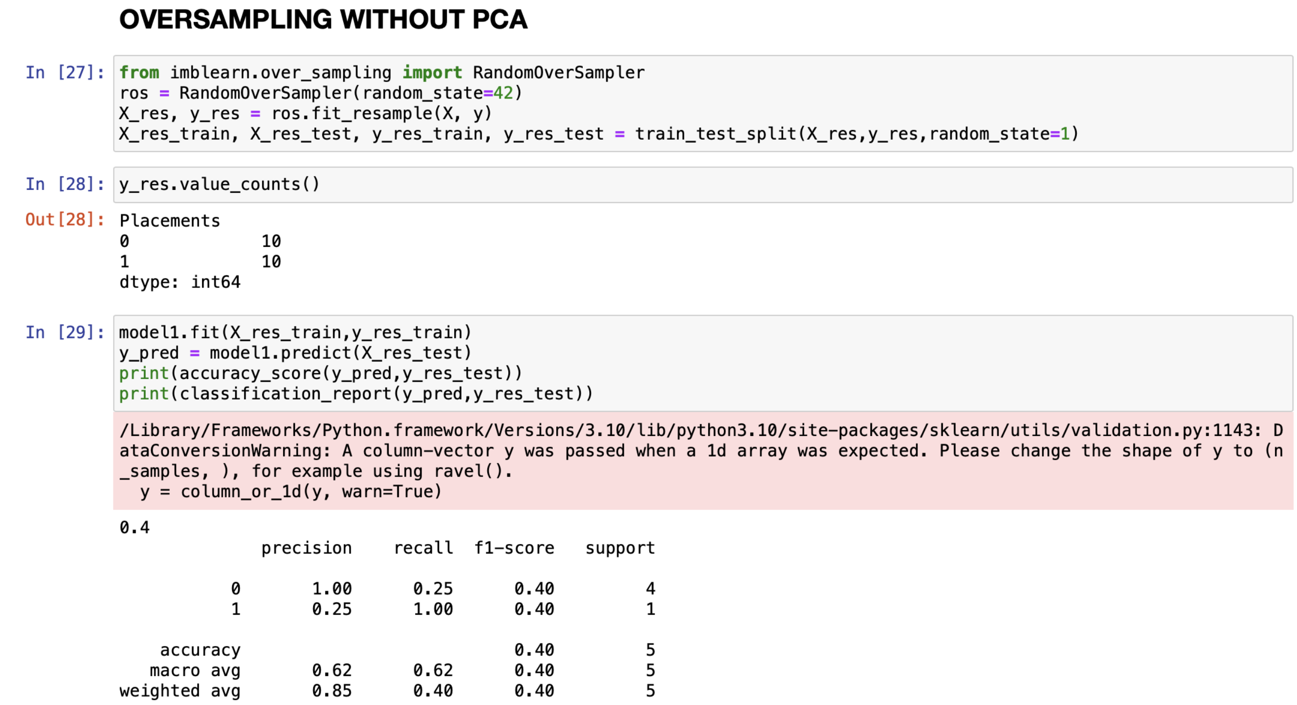
```python
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_res, y_res = ros.fit_resample(X, y)
X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res, random_state=1)

y_res.value_counts()

```



```
In [27]: from imblearn.over_sampling import RandomOverSampler
         ros = RandomOverSampler(random_state=42)
         X_res, y_res = ros.fit_resample(X, y)
         X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res,random_state=1)

In [28]: y_res.value_counts()

Out[28]: Placements
         0             10
         1             10
         dtype: int64
```

12 Oversampling Without PCA

```python
model1.fit(X_res_train,y_res_train)
y_pred = model1.predict(X_res_test)
print(accuracy_score(y_pred,y_res_test))
print(classification_report(y_pred,y_res_test))
```



OVERSAMPLING WITHOUT PCA

```
In [27]: from imblearn.over_sampling import RandomOverSampler
         ros = RandomOverSampler(random_state=42)
         X_res, y_res = ros.fit_resample(X, y)
         X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res,random_state=1)
```

```
In [28]: y_res.value_counts()
```

```
Out[28]: Placements
         0             10
         1             10
         dtype: int64
```

```
In [29]: model1.fit(X_res_train,y_res_train)
         y_pred = model1.predict(X_res_test)
         print(accuracy_score(y_pred,y_res_test))
         print(classification_report(y_pred,y_res_test))
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n
_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
0.4
              precision    recall  f1-score   support

           0       1.00      0.25      0.40         4
           1       0.25      1.00      0.40         1

    accuracy                           0.40         5
   macro avg       0.62      0.62      0.40         5
weighted avg       0.85      0.40      0.40         5
```

13 Oversampling With PCA

```
X_res, y_res = ros.fit_resample(X1, y1)
X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res,
  random_state=1)

model1.fit(X_res_train,y_res_train)
y_pred = model1.predict(X_res_test)
print(accuracy_score(y_pred,y_res_test))
print(classification_report(y_pred,y_res_test))

```

OVERSAMPLING WITH PCA

In [30]:
```
X_res, y_res = ros.fit_resample(X1, y1)
X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res,y_res,random_state=1)
```

In [31]:
```
model1.fit(X_res_train,y_res_train)
y_pred = model1.predict(X_res_test)
print(accuracy_score(y_pred,y_res_test))
print(classification_report(y_pred,y_res_test))
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n
_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
0.4
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00         0
         1.0       1.00      0.40      0.57         5

    accuracy                           0.40         5
   macro avg       0.50      0.20      0.29         5
weighted avg       1.00      0.40      0.57         5
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:
1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:
1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:
1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```