PLACEMENT PREDICTOR

A mini-project submitted for

Business Intelligence Lab (Semester VI)

by

SAP ID	Name
60003170039	PARTH KANSARA
60003170047	RACHIT PATANI
60003170052	ROHIT JAIN
60003170054	SAURAV KABRA

PROBLEM DEFINITION

Data Analysis is all about finding some interesting insights in the data. A lot of questions such as 'which board of education should one choose', 'Male vs Female in placements', 'Reason of unemployment' etc. have always been a concern to students as well as faculties. The aim of this project is to give insights on the factors that play a role in placements and also answers many recruitments concerned questions including the above mentioned. The model, based on user input classifies if a person will be recruited or no. Furthermore, after the recruitment, what will be the person's salary is also predicted by the model.

DATASET

The dataset consists of Placement data of students in our campus. It includes secondary and higher secondary school percentage and specialization. It also includes degree specialization, type and Work experience and salary offers to the placed students.

Kaggle Link- Campus Recruitment Dataset

DATA EXPLORATION

LIBRARIES:

```
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, plot_confusion_matrix
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LinearRegression

import statsmodels.api as sm
```

LOADING THE DATASET:

```
In [2]: df = pd.read_csv('Placement_Data_Full_Class.csv')
        df
```

Out[2]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	NaN
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0

DATA ANALYSIS:

```
In [3]: # removing sl_no column
        df.drop(columns=['sl_no'], inplace=True)
```

```
In [4]: # checking distribution range
        df.describe()
```

Out[4]:

	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
count	215.000000	215.000000	215.000000	215.000000	215.000000	148.000000
mean	67.303395	66.333163	66.370186	72.100558	62.278186	288655.405405
std	10.827205	10.897509	7.358743	13.275956	5.833385	93457.452420
min	40.890000	37.000000	50.000000	50.000000	51.210000	200000.000000
25%	60.600000	60.900000	61.000000	60.000000	57.945000	240000.000000
50%	67.000000	65.000000	66.000000	71.000000	62.000000	265000.000000
75%	75.700000	73.000000	72.000000	83.500000	66.255000	300000.000000
max	89.400000	97.700000	91.000000	98.000000	77.890000	940000.000000

In [5]:
```python
# checking for null values
df.isnull().sum()
```

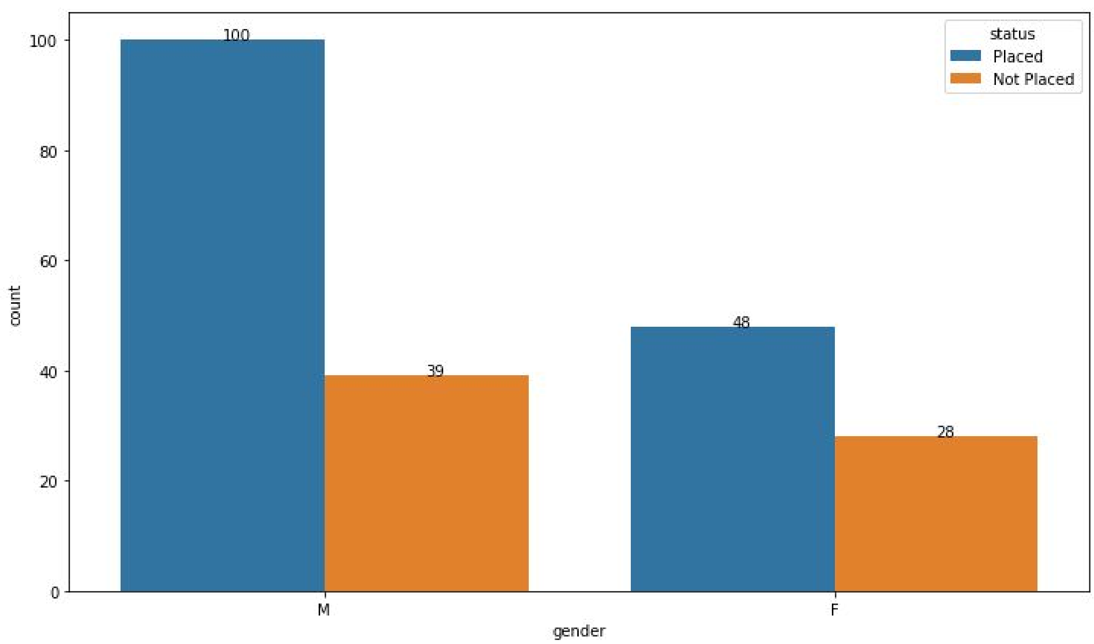
Out[5]:
```
gender            0
ssc_p             0
ssc_b             0
hsc_p             0
hsc_b             0
hsc_s             0
degree_p          0
degree_t          0
workex            0
etest_p           0
specialisation    0
mba_p             0
status            0
salary           67
dtype: int64
```

EXPLORING DATA BY FEATURES

1) GENDER:

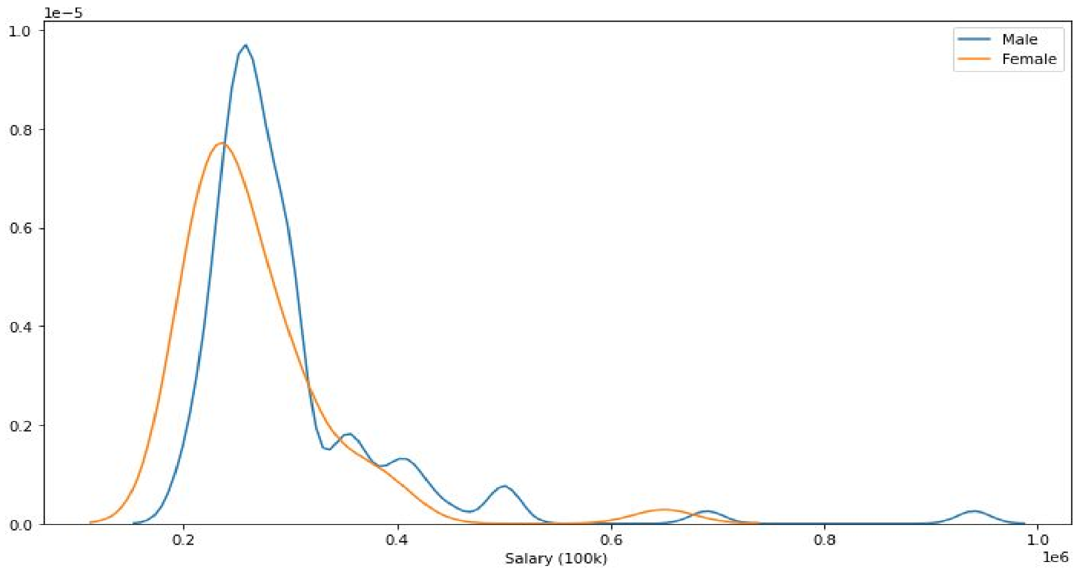
A) PLACEMENT

In [9]:
```python
plt.figure(figsize=(12, 7))
ax = sns.countplot("gender", hue="status", data=df)
for i in ax.patches:
    ax.annotate(i.get_height(), (i.get_x() + i.get_width()/2, i.get_height()))
```



B) SALARY

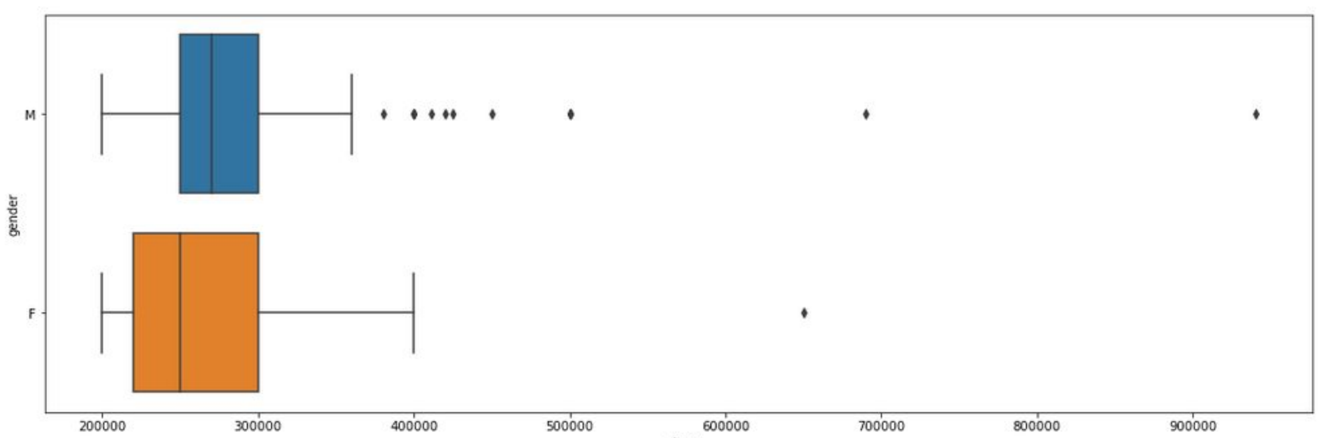
```
In [10]: # This plot ignores NaN values for salary, igoring students who are not placed
         plt.figure(figsize=(12, 7))
         sns.kdeplot(df.salary[df.gender=="M"])
         sns.kdeplot(df.salary[df.gender=="F"])
         plt.legend(["Male", "Female"])
         plt.xlabel("Salary (100k)")

Out[10]: Text(0.5, 0, 'Salary (100k)')
```



C) SALARY VS GENDER

```
plt.figure(figsize =(18, 6))
sns.boxplot("salary", "gender", data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4a1e13748>
```



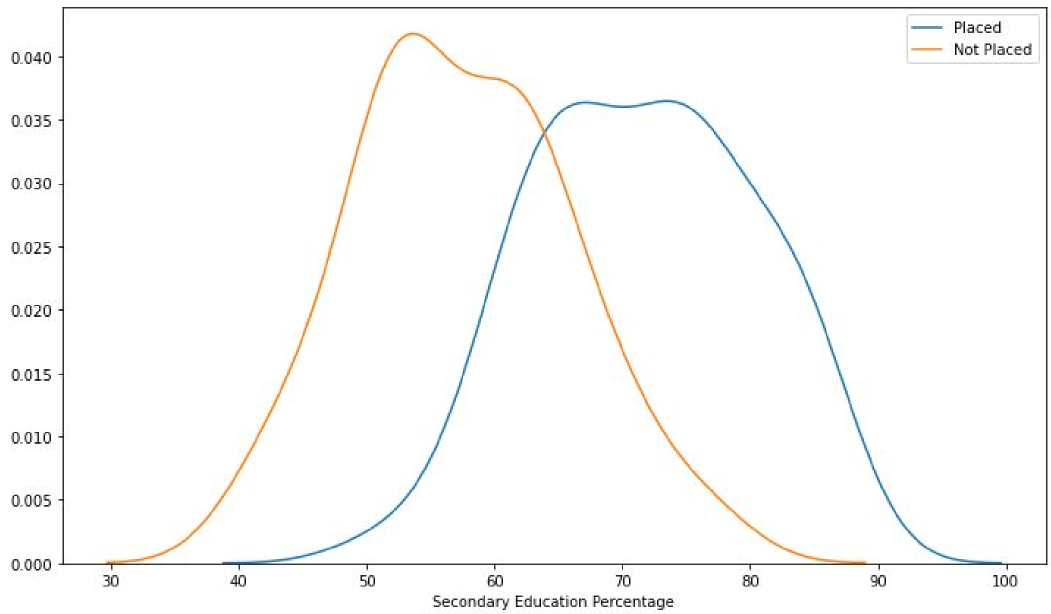
- **Insights**
  - We have samples of 139 Male students and 76 Female students.
  - 30 Female and 40 Male students are not placed. Male students have comparatively higher placemets.
  - Male students are offered slightly greater salary than female on an average.

2) SECONDARY EDUCATION PERCENTAGE

A) PLACED AND NOT PLACED

In [12]:
```
# Kernel-Density Plot
plt.figure(figsize=(12, 7))
sns.kdeplot(df.ssc_p[df.status=="Placed"])
sns.kdeplot(df.ssc_p[df.status=="Not Placed"])
plt.legend(["Placed", "Not Placed"])
plt.xlabel("Secondary Education Percentage")
```

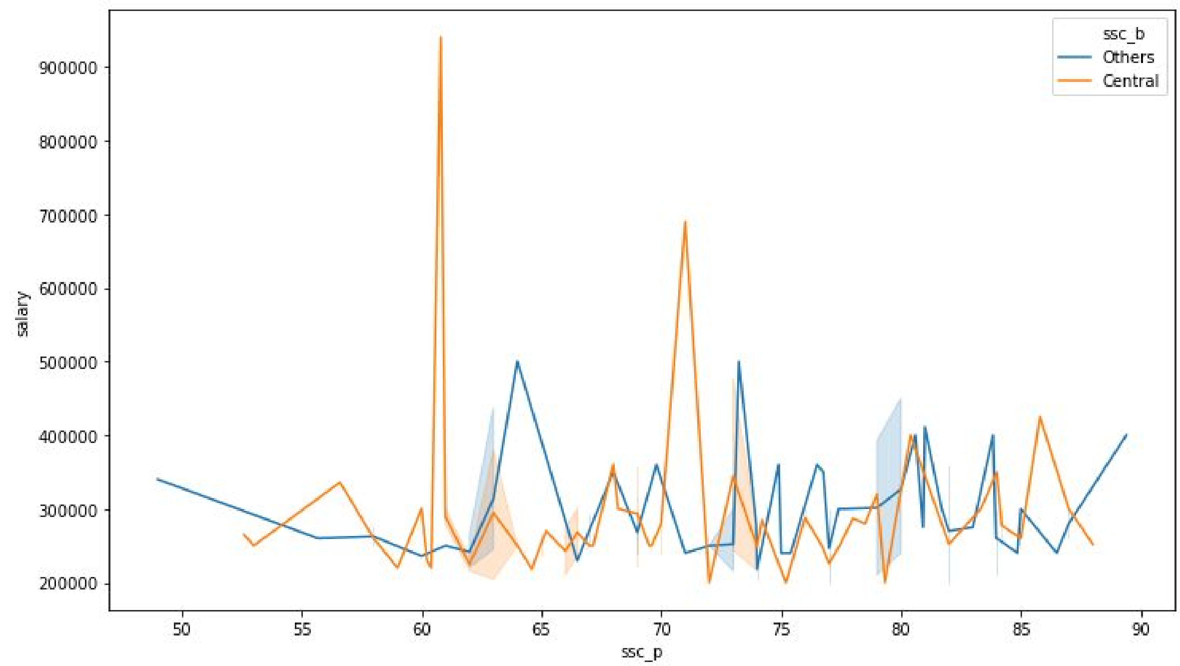
Out[12]:
```
Text(0.5, 0, 'Secondary Education Percentage')
```



B) SALARY

```
In [15]: plt.figure(figsize=(12, 7))
         sns.lineplot("ssc_p", "salary", hue="ssc_b", data=df)
```

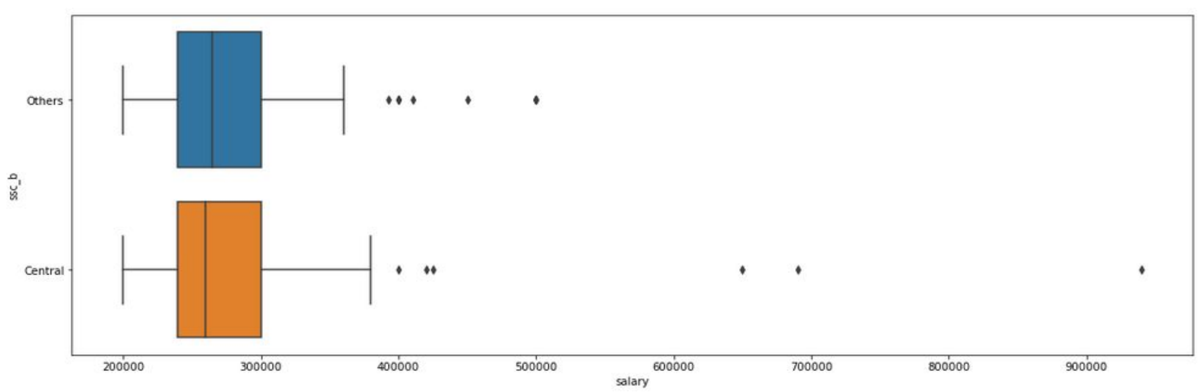
```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1b4a360e108>
```



C) BOARD OF EDUCATION VS SALARY

```
In [16]: plt.figure(figsize =(18, 6))
         sns.boxplot("salary", "ssc_b", data=df)
```

```
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1b4a3453f88>
```



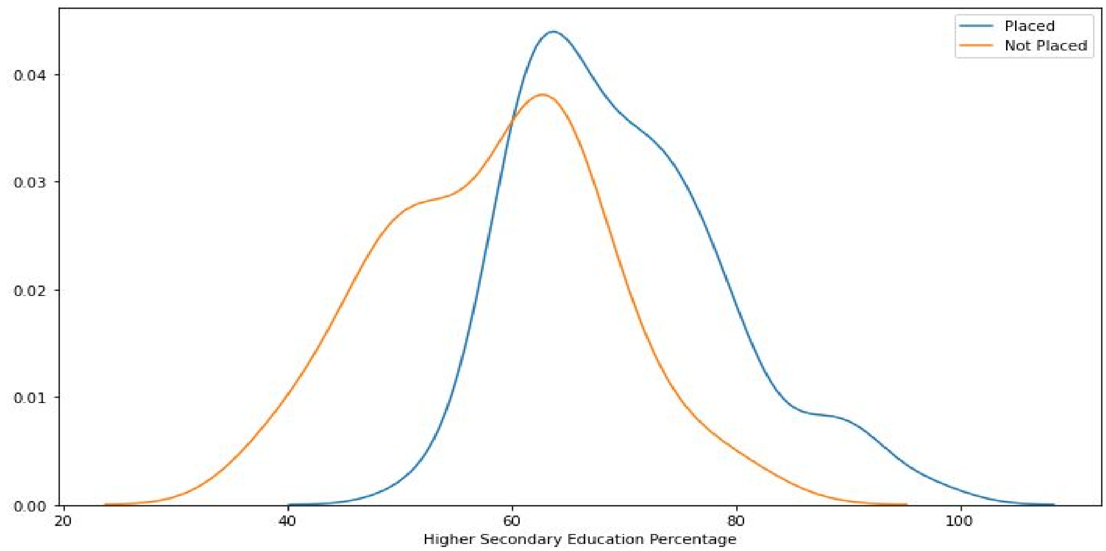
- **Insights**
  - We have samples of 116 central students and 99 others students.
  - No specific pattern (correlation) between Secondary Education Percentage and Salary.
  - Board of Education is Not Affecting Salary

3) HIGHER SECONDARY EDUCATION

A) PLACED AND NOT PLACED

In [17]:
```
# Kernel-Density Plot
plt.figure(figsize=(12, 7))
sns.kdeplot(df.hsc_p[df.status=="Placed"])
sns.kdeplot(df.hsc_p[df.status=="Not Placed"])
plt.legend(["Placed", "Not Placed"])
plt.xlabel("Higher Secondary Education Percentage")
```

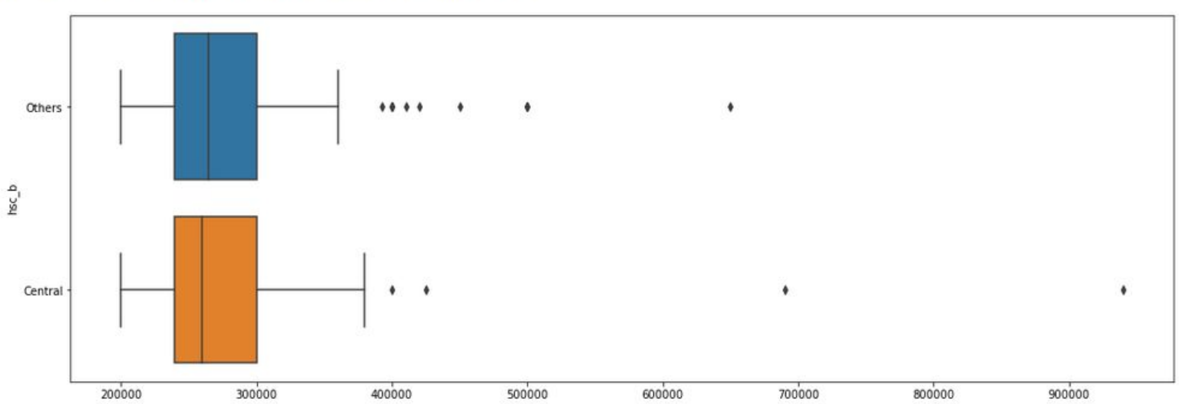
Out[17]:
```
Text(0.5, 0, 'Higher Secondary Education Percentage')
```



B) EDUCATION BOARD VS SALARY

In [19]:
```
plt.figure(figsize =(18, 6))
sns.boxplot("salary", "hsc_b", data=df)
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4a36053c8>
```

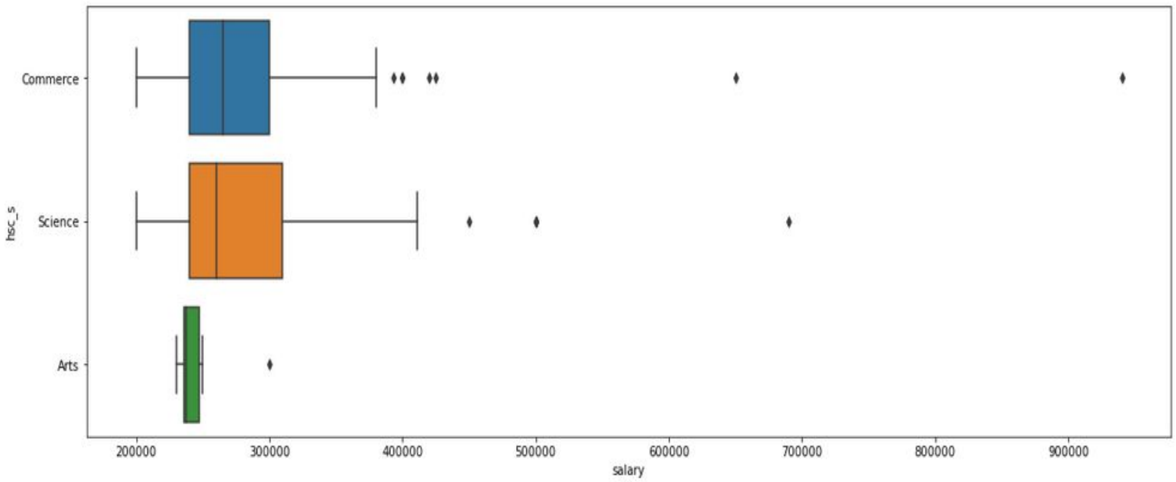


- Outliers on both, board doesn't affect getting highly paid jobs. Highest paid job was obtailed by student from Central Board though.

C) EDUCATION STREAM VS SALARY

In [22]:
```
plt.figure(figsize =(18, 6))
sns.boxplot("salary", "hsc_s", data=df)
```

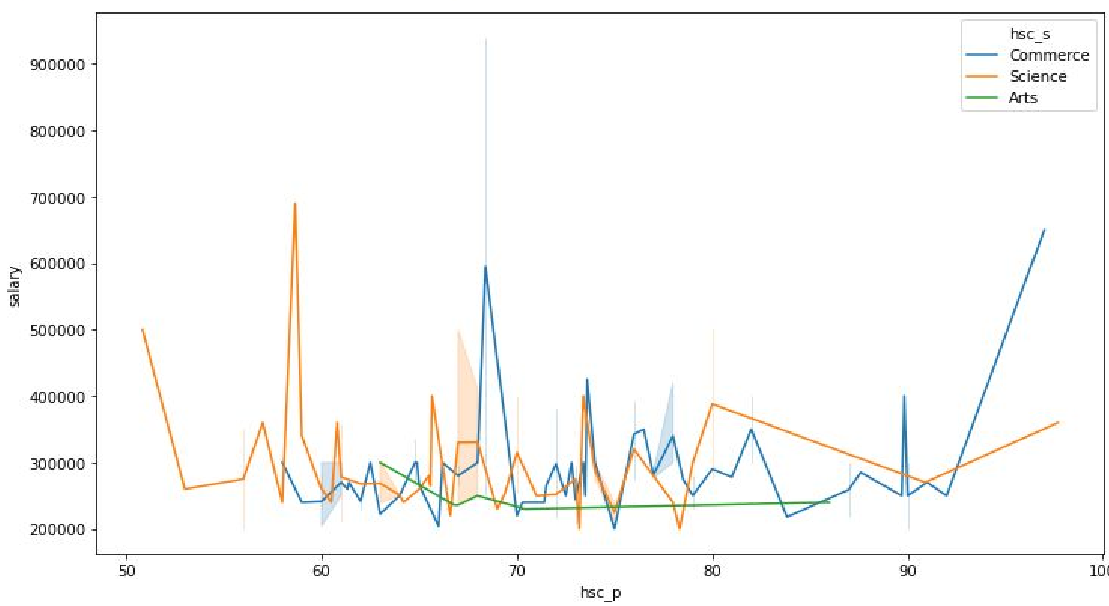
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1b4a4243f08>



- We can't really say for sure due to only few samples of students with Arts Major, but they aren't getting good salaries.
- Commerse students have slightly better placement status.

In [24]:
```
plt.figure(figsize=(12, 7))
sns.lineplot("hsc_p", "salary", hue="hsc_s", data=df)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1b4a469d208>



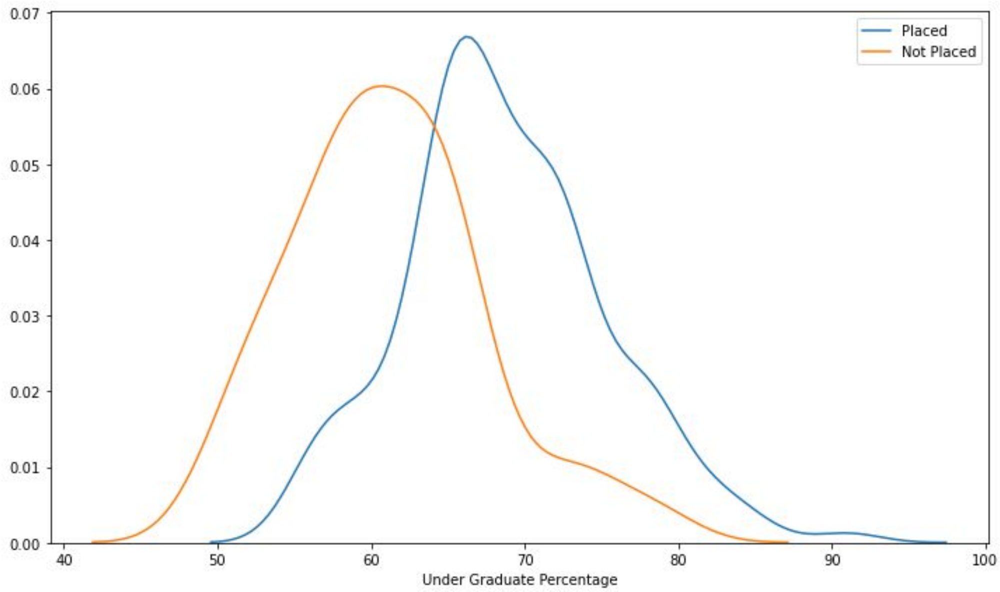
- Student with Art Specialization surprisingly have comparatively low salary

4) UNDERGRADUATE DEGREE PERCENTAGE:

A) PLACED AND NOT PLACED

In [25]:
```
# Kernel-Density Plot
plt.figure(figsize=(12, 7))
sns.kdeplot(df.degree_p[df.status=="Placed"])
sns.kdeplot(df.degree_p[df.status=="Not Placed"])
plt.legend(["Placed", "Not Placed"])
plt.xlabel("Under Graduate Percentage")
```

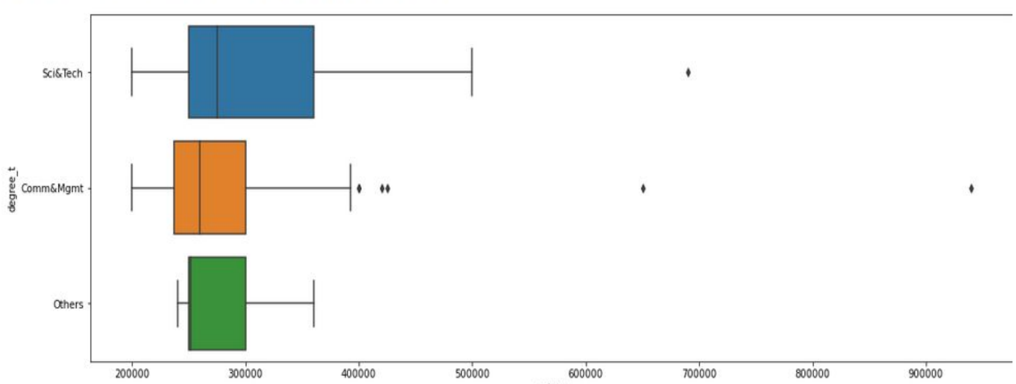
Out[25]: `Text(0.5, 0, 'Under Graduate Percentage')`



B) STREAM VS SALARY

In [28]:
```
plt.figure(figsize =(18, 6))
sns.boxplot("salary", "degree_t", data=df)
```

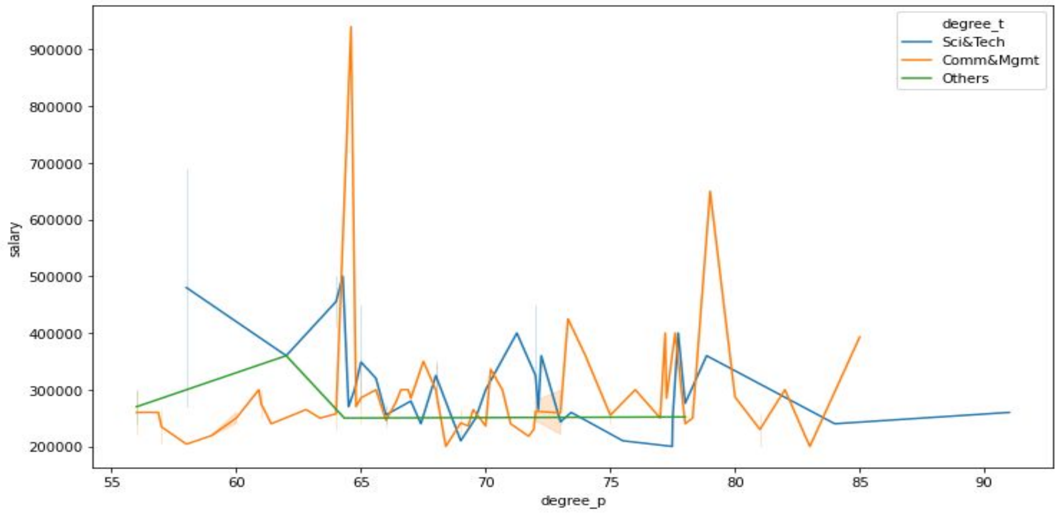
Out[28]: `<matplotlib.axes._subplots.AxesSubplot at 0x1b4a2301f08>`



- Science&Tech students getting more salary on average
- Management students are getting more highly paid dream jobs.

```
In [29]: plt.figure(figsize=(12, 7))
         sns.lineplot("degree_p", "salary", hue="degree_t", data=df)
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x1b4a229d488>
```

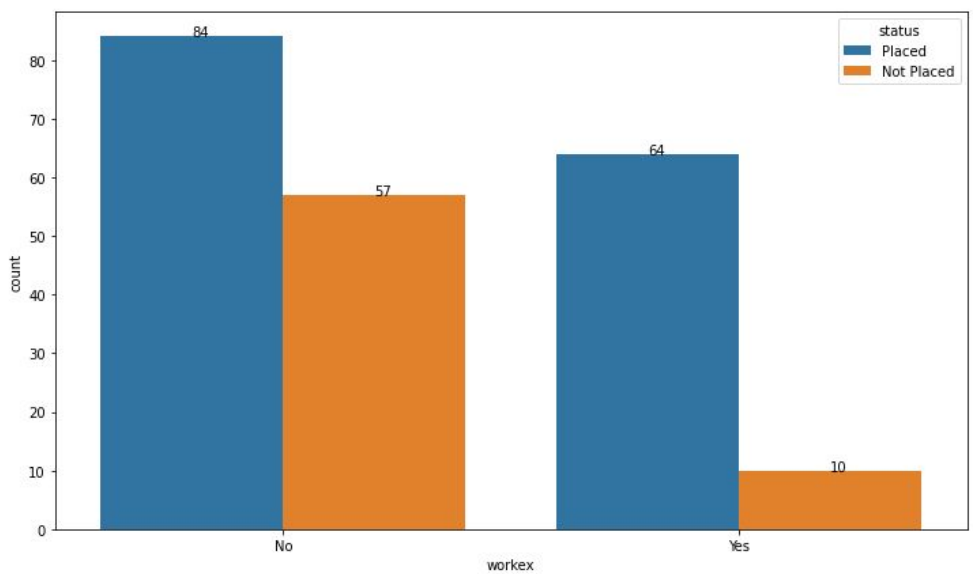


- Percentage does not seem to affect salary.
- Commerce&Mgmt students occasionally get dream placements with high salary

5) WORK EXPERIENCE

A) PLACED AND NOT PLACED

```
plt.figure(figsize=(12, 7))
ax = sns.countplot("workex", hue="status", data=df)
for i in ax.patches:
    ax.annotate(i.get_height(), (i.get_x() + i.get_width()/2, i.get_height()))
```

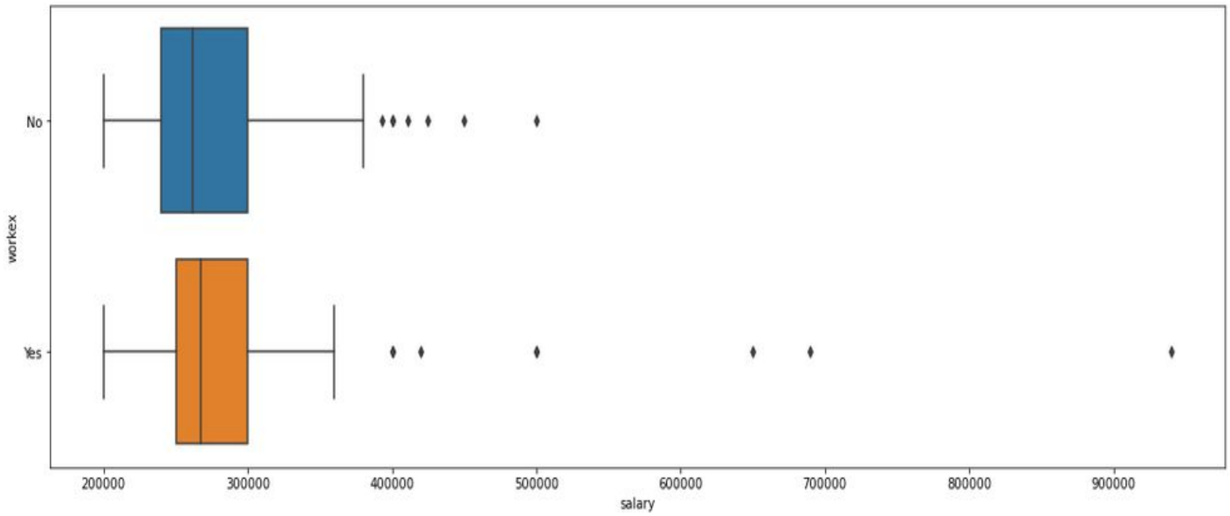


- This affects Placement. Very few students with work experience not getting placed!

B) WORK EXPERIENCE VS SALARY

```
plt.figure(figsize =(18, 6))
sns.boxplot("salary", "workex", data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4a204fd88>
```

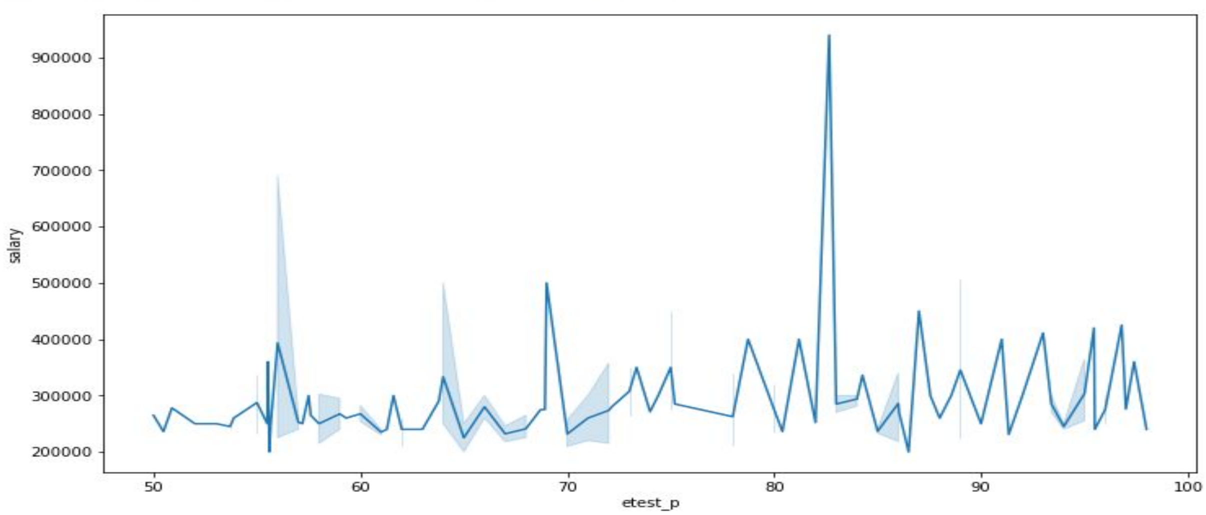


- Outliers (High salary than average) on bith end but students with experience getting dream jobs
- Average salary as well as base salary high for students with work experience.

6) EMPLOYABILITY TEST PERCENTAGE

```
plt.figure(figsize=(12, 7))
sns.lineplot("etest_p", "salary", data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4a3952788>
```

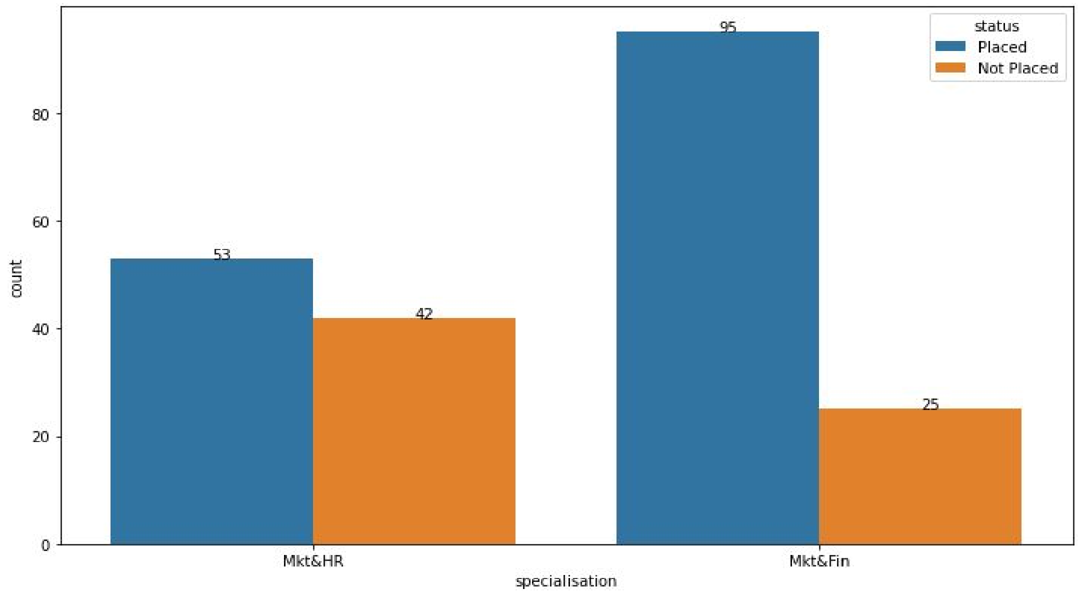


- This feature surprisingly does not affect placements and salary much

7) POST GRADUATE SPECIALIZATION

A) PLACED AND NOT PLACED (COUNT VS SPECIALIZATION STREAM)

```
plt.figure(figsize=(12, 7))
ax = sns.countplot("specialisation", hue="status", data=df)
for i in ax.patches:
    ax.annotate(i.get_height(), (i.get_x() + i.get_width()/2, i.get_height()))
```

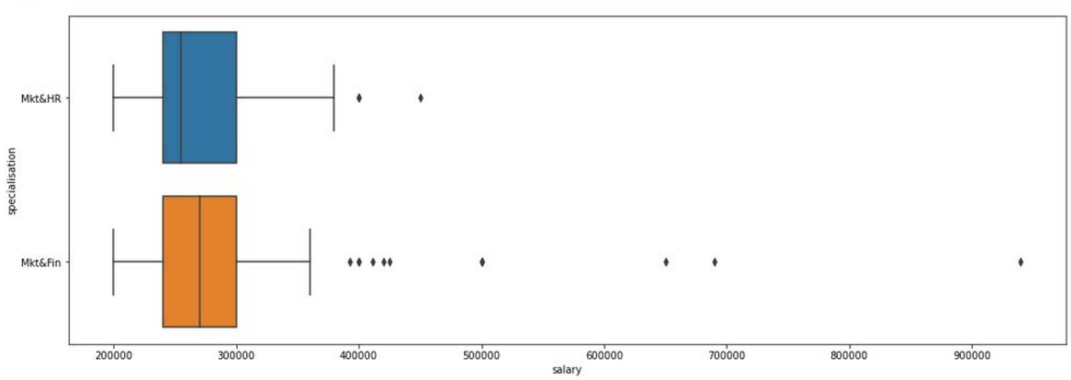


- This feature affects Placement status.
- Comparitively very low not-placed students in Mkt&Fin Section

B) SPECIALIZATION VS SALARY

```
plt.figure(figsize =(18, 6))
sns.boxplot("salary", "specialisation", data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4a3a5b408>
```



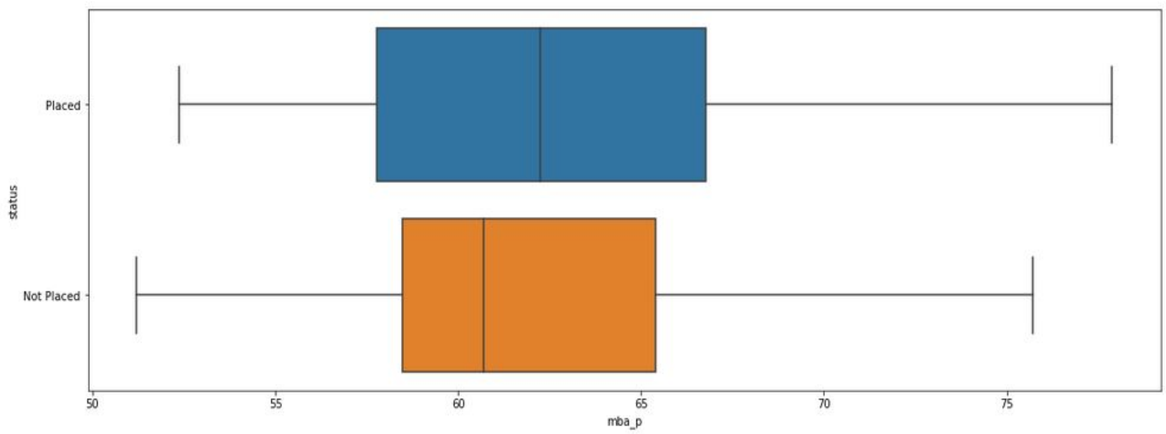
- More Highly Paid Jobs for Mkt&Fin students

8) MBA PERCENTAGE:

A) PLACED AND NOT PLACED

```
plt.figure(figsize =(18, 6))
sns.boxplot("mba_p", "status", data=df)
```

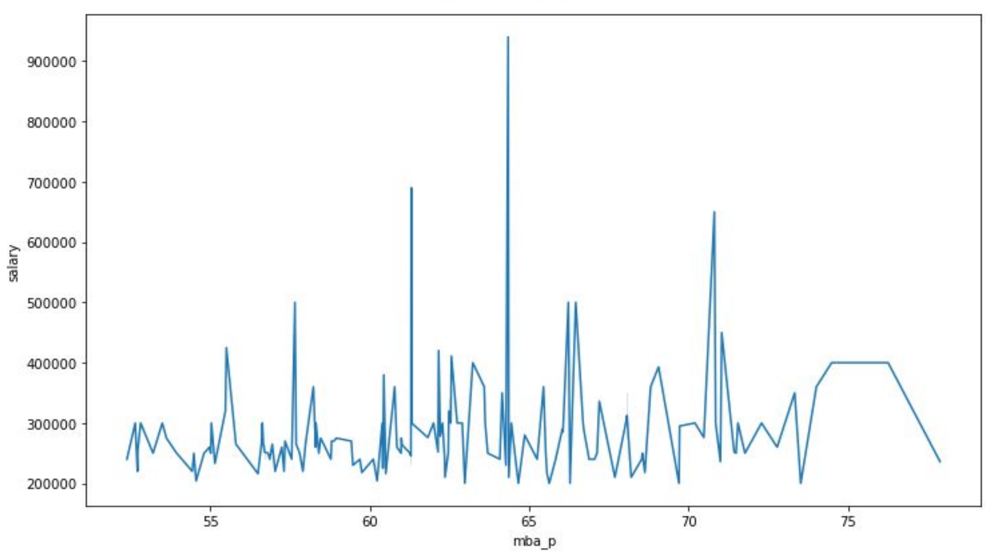
<matplotlib.axes._subplots.AxesSubplot at 0x1b4a3aba548>



B) SALARY

```
plt.figure(figsize=(12, 7))
sns.lineplot("mba_p", "salary", data=df)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1b4a5d1c0c8>



- MBA Percentage also does not affect salary much

DATA PREPROCESSING

Feature Selection

Using Only following features (Ignoring Board of Education -> they didnt seem to have much effect)

- Gender
- Secondary Education percentage
- Higher Secondary Education Percentsge
- Specialization in Higher Secondary Education
- Under Graduate Dergree Percentage
- Under Graduation Degree Field
- Work Experience
- Employability test percentage
- Specialization
- MBA Percentage

In [40]:
```
df.drop(columns=['ssc_b', 'hsc_b'], inplace=True)
```

Feature Encoding

In [41]:
```
df["gender"] = df.gender.map({"M":0, "F":1})
df["workex"] = df.workex.map({"No":0, "Yes":1})
df["status"] = df.status.map({"Not Placed":0, "Placed":1})
df["specialisation"] = df.specialisation.map({"Mkt&HR":0, "Mkt&Fin":1})
```

In [42]:
```
df
```

Out[42]:

	gender	ssc_p	hsc_p	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	0	67.00	91.00	Commerce	58.00	Sci&Tech	0	55.0	0	58.80	1	270000.0
1	0	79.33	78.33	Science	77.48	Sci&Tech	1	86.5	1	66.28	1	200000.0
2	0	65.00	68.00	Arts	64.00	Comm&Mgmt	0	75.0	1	57.80	1	250000.0
3	0	56.00	52.00	Science	52.00	Sci&Tech	0	66.0	0	59.43	0	NaN
4	0	85.80	73.60	Commerce	73.30	Comm&Mgmt	0	96.8	1	55.50	1	425000.0
...	...	...	...	...	...	...	...	...	...	...	...	...
210	0	80.60	82.00	Commerce	77.60	Comm&Mgmt	0	91.0	1	74.49	1	400000.0
211	0	58.00	60.00	Science	72.00	Sci&Tech	0	74.0	1	53.62	1	275000.0
212	0	67.00	67.00	Commerce	73.00	Comm&Mgmt	1	59.0	1	69.72	1	295000.0
213	1	74.00	66.00	Commerce	58.00	Comm&Mgmt	0	70.0	0	60.23	1	204000.0
214	0	62.00	58.00	Science	53.00	Comm&Mgmt	0	89.0	0	60.22	0	NaN

215 rows × 12 columns

In [43]:

```
hsc_s = pd.get_dummies(df['hsc_s'])
hsc_s
```

Out[43]:

	Arts	Commerce	Science
0	0	1	0
1	0	0	1
2	1	0	0
3	0	0	1
4	0	1	0
...	...	...	...
210	0	1	0
211	0	0	1
212	0	1	0
213	0	1	0
214	0	0	1

215 rows × 3 columns

In [44]:

```
degree_t = pd.get_dummies(df['degree_t'])
degree_t
```

Out[44]:

	Comm&Mgmt	Others	Sci&Tech
0	0	0	1
1	0	0	1
2	1	0	0
3	0	0	1
4	1	0	0
...	...	...	...
210	1	0	0
211	0	0	1
212	1	0	0
213	1	0	0
214	1	0	0

215 rows × 3 columns

In [45]:

```
# ignoring 'Arts' from hsc_s & 'others' from degree_t
df.drop(columns=['hsc_s', 'degree_t'], inplace=True)
df = pd.concat([hsc_s.iloc[:, 1:], degree_t.iloc[:, [0, 2]], df], axis=1)
df
```

Out[45]:

	Commerce	Science	Comm&Mgmt	Sci&Tech	gender	ssc_p	hsc_p	degree_p	workex	etest_p	specialisation	mba_p	status	salary
0	1	0	0	1	0	67.00	91.00	58.00	0	55.0	0	58.80	1	270000.0
1	0	1	0	1	0	79.33	78.33	77.48	1	86.5	1	66.28	1	200000.0
2	0	0	1	0	0	65.00	68.00	64.00	0	75.0	1	57.80	1	250000.0
3	0	1	0	1	0	56.00	52.00	52.00	0	66.0	0	59.43	0	NaN
4	1	0	1	0	0	85.80	73.60	73.30	0	96.8	1	55.50	1	425000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	1	0	1	0	0	80.60	82.00	77.60	0	91.0	1	74.49	1	400000.0
211	0	1	0	1	0	58.00	60.00	72.00	0	74.0	1	53.62	1	275000.0
212	1	0	1	0	0	67.00	67.00	73.00	1	59.0	1	69.72	1	295000.0
213	1	0	1	0	1	74.00	66.00	58.00	0	70.0	0	60.23	1	204000.0
214	0	1	1	0	0	62.00	58.00	53.00	0	89.0	0	60.22	0	NaN

215 rows × 14 columns

Data Preprocessing for Predicting if students gets placed or not (Binary Classification Problem)

Dropping salary column

In [46]:
```
data1 = df.copy()
data1.drop(columns=['salary'], inplace=True)
data1
```

Out[46]:

	Commerce	Science	Comm&Mgmt	Sci&Tech	gender	ssc_p	hsc_p	degree_p	workex	etest_p	specialisation	mba_p	status
0	1	0	0	1	0	67.00	91.00	58.00	0	55.0	0	58.80	1
1	0	1	0	1	0	79.33	78.33	77.48	1	86.5	1	66.28	1
2	0	0	1	0	0	65.00	68.00	64.00	0	75.0	1	57.80	1
3	0	1	0	1	0	56.00	52.00	52.00	0	66.0	0	59.43	0
4	1	0	1	0	0	85.80	73.60	73.30	0	96.8	1	55.50	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	1	0	1	0	0	80.60	82.00	77.60	0	91.0	1	74.49	1
211	0	1	0	1	0	58.00	60.00	72.00	0	74.0	1	53.62	1
212	1	0	1	0	0	67.00	67.00	73.00	1	59.0	1	69.72	1
213	1	0	1	0	1	74.00	66.00	58.00	0	70.0	0	60.23	1
214	0	1	1	0	0	62.00	58.00	53.00	0	89.0	0	60.22	0

215 rows × 13 columns

Splitting dataframe into dependent & independent variables

In [47]:
```
X1 = data1.iloc[:, :-1].values
Y1 = data1.iloc[:, -1:].values
```

Splitting into training & test sets

In [48]:
```
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, test_size=0.1, shuffle=True)
```

Normailization

In [49]:
```
scaler_X1 = MinMaxScaler()
X1_train = scaler_X1.fit_transform(X1_train)
X1_test = scaler_X1.transform(X1_test)
```

In [50]:
```
pd.DataFrame(pd.concat([pd.DataFrame(data=X1_train, columns=data1.columns[:-1]), pd.DataFrame(data=Y1_train, columns=data1.colum
ns[-1:])], axis=1))
```

Out[50]:

	Commerce	Science	Comm&Mgmt	Sci&Tech	gender	ssc_p	hsc_p	degree_p	workex	etest_p	specialisation	mba_p	status
0	0.0	1.0	0.0	1.0	0.0	0.454545	0.391823	0.428571	0.0	0.791139	0.0	0.355322	0
1	1.0	0.0	1.0	0.0	0.0	0.537190	0.477002	0.657143	1.0	0.189873	1.0	0.693778	1
2	1.0	0.0	1.0	0.0	0.0	0.925620	0.589438	0.665714	0.0	0.987342	1.0	0.160795	1
3	1.0	0.0	1.0	0.0	0.0	0.227273	0.391823	0.114286	0.0	0.464135	0.0	0.157421	0
4	0.0	1.0	1.0	0.0	0.0	0.272727	0.408859	0.228571	0.0	0.611814	0.0	0.270990	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
188	1.0	0.0	1.0	0.0	0.0	0.661157	0.323680	0.171429	0.0	0.717300	0.0	0.053598	1
189	1.0	0.0	1.0	0.0	0.0	0.757645	0.441056	0.590571	0.0	0.822785	1.0	0.344078	1
190	0.0	1.0	0.0	1.0	0.0	0.555785	0.391823	0.485714	1.0	0.170886	1.0	0.918291	0
191	0.0	1.0	1.0	0.0	0.0	0.681818	0.391823	0.514286	0.0	0.506329	1.0	0.254123	1
192	0.0	1.0	0.0	1.0	1.0	0.940083	0.429302	0.497143	0.0	0.189873	1.0	0.317841	1

193 rows × 13 columns

Data Preprocessing for Predicting salary of student (Regression Problem)

Handling status column

In [63]:
```
# making a copy of dataset
data2 = df.copy()

# dropping unemployed status
data2 = data2[data2['status']==1]
data2
```

Out[63]:

	Commerce	Science	Comm&Mgmt	Sci&Tech	gender	ssc_p	hsc_p	degree_p	workex	etest_p	specialisation	mba_p	status	salary
0	1	0	0	1	0	67.00	91.00	58.00	0	55.0	0	58.80	1	270000.0
1	0	1	0	1	0	79.33	78.33	77.48	1	86.5	1	66.28	1	200000.0
2	0	0	1	0	0	65.00	68.00	64.00	0	75.0	1	57.80	1	250000.0
4	1	0	1	0	0	85.80	73.60	73.30	0	96.8	1	55.50	1	425000.0
7	0	1	0	1	0	82.00	64.00	66.00	1	67.0	1	62.14	1	252000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209	1	0	1	0	0	62.00	72.00	65.00	0	67.0	1	56.49	1	216000.0
210	1	0	1	0	0	80.60	82.00	77.60	0	91.0	1	74.49	1	400000.0
211	0	1	0	1	0	58.00	60.00	72.00	0	74.0	1	53.62	1	275000.0
212	1	0	1	0	0	67.00	67.00	73.00	1	59.0	1	69.72	1	295000.0
213	1	0	1	0	1	74.00	66.00	58.00	0	70.0	0	60.23	1	204000.0

148 rows × 14 columns

In [64]:
```
data2.drop(columns=['status'], inplace=True)
```

Handling missing values

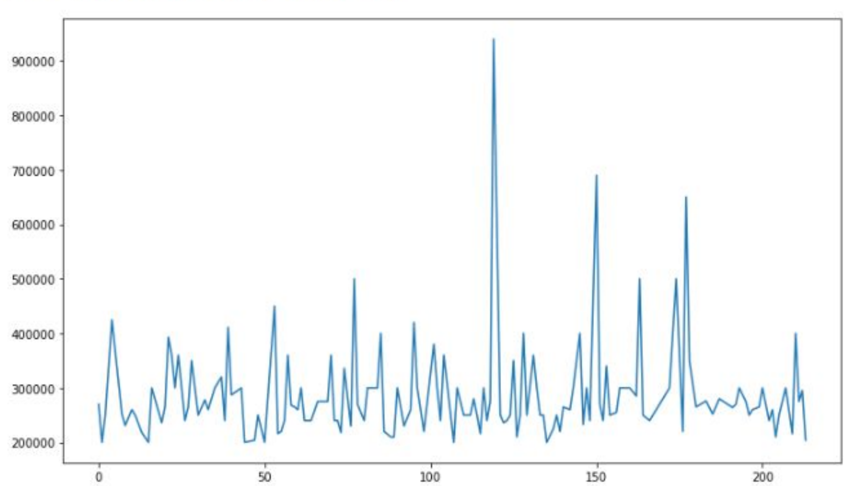
In [65]:
```
# no need to handle as there are none
data2['salary'].isnull().sum()
```

Out[65]: 0

Removing Outliers

In [66]:
```
plt.figure(figsize=(12, 7))
plt.plot(data2['salary'])
```

Out[66]: [<matplotlib.lines.Line2D at 0x29d4b722e08>]



In [67]:
```python
# removing them
data2['salary'][data2['salary']>500000]
```

Out[67]:
```
119    940000.0
150    690000.0
177    650000.0
Name: salary, dtype: float64
```

In [68]:
```python
data2 = data2[data2['salary']<500000]
data2
```

Out[68]:

	Commerce	Science	Comm&Mgmt	Sci&Tech	gender	ssc_p	hsc_p	degree_p	workex	etest_p	specialisation	mba_p	salary
0	1	0	0	1	0	67.00	91.00	58.00	0	55.0	0	58.80	270000.0
1	0	1	0	1	0	79.33	78.33	77.48	1	86.5	1	66.28	200000.0
2	0	0	1	0	0	65.00	68.00	64.00	0	75.0	1	57.80	250000.0
4	1	0	1	0	0	85.80	73.60	73.30	0	96.8	1	55.50	425000.0
7	0	1	0	1	0	82.00	64.00	66.00	1	67.0	1	62.14	252000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
209	1	0	1	0	0	62.00	72.00	65.00	0	67.0	1	56.49	216000.0
210	1	0	1	0	0	80.60	82.00	77.60	0	91.0	1	74.49	400000.0
211	0	1	0	1	0	58.00	60.00	72.00	0	74.0	1	53.62	275000.0
212	1	0	1	0	0	67.00	67.00	73.00	1	59.0	1	69.72	295000.0
213	1	0	1	0	1	74.00	66.00	58.00	0	70.0	0	60.23	204000.0

142 rows × 13 columns

Splitting dataframe into dependent & independent variables

In [69]:
```python
X2 = data2.iloc[:, :-1].values
Y2 = data2.iloc[:, -1:].values
```

Splitting into training & test sets

In [70]:
```python
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, Y2, test_size=0.1, shuffle=True)
```

Normailization

In [71]:
```python
scaler_X2 = MinMaxScaler()
X2_train = scaler_X2.fit_transform(X2_train)
X2_test = scaler_X2.transform(X2_test)

scaler_Y2 = MinMaxScaler()
Y2_train = scaler_Y2.fit_transform(Y2_train)
Y2_test = scaler_Y2.transform(Y2_test)
```

In [72]:
```python
pd.DataFrame(pd.concat([pd.DataFrame(data=X2_train, columns=data2.columns[:-1]), pd.DataFrame(data=Y2_train, columns=data2.colum
ns[-1:])], axis=1))
```

Out[72]:

	Commerce	Science	Comm&Mgmt	Sci&Tech	gender	ssc_p	hsc_p	degree_p	workex	etest_p	specialisation	mba_p	salary
0	1.0	0.0	1.0	0.0	1.0	0.297030	0.626398	0.297143	0.0	0.018542	0.0	0.385339	0.312
1	0.0	1.0	0.0	1.0	0.0	0.792079	0.335570	0.228571	0.0	0.895833	1.0	0.399059	0.844
2	0.0	1.0	1.0	0.0	0.0	0.396040	0.335570	0.371429	0.0	0.077083	0.0	0.103097	0.200
3	1.0	0.0	1.0	0.0	0.0	0.222772	0.201342	0.228571	0.0	0.080833	1.0	0.101529	0.240
4	1.0	0.0	1.0	0.0	0.0	0.495050	0.268456	0.028571	0.0	0.479167	0.0	0.134065	0.260
...	...	...	...	...	...	...	...	...	...	...	...	...	...
122	1.0	0.0	1.0	0.0	0.0	0.321782	0.268456	0.285714	0.0	0.000000	0.0	0.169345	0.260
123	1.0	0.0	1.0	0.0	0.0	0.709653	0.265996	0.419143	0.0	0.812500	1.0	0.313995	0.400
124	1.0	0.0	1.0	0.0	1.0	0.693069	0.178971	0.342857	1.0	0.156250	1.0	0.350059	0.400
125	0.0	1.0	0.0	1.0	0.0	0.891089	0.156600	0.498000	1.0	0.208333	1.0	0.349275	0.240
126	0.0	1.0	1.0	0.0	0.0	0.445545	0.178971	0.457143	0.0	0.458333	1.0	0.338299	0.256

127 rows × 13 columns

JUSTIFICATION

Supervised Learning is implemented to train the model as the dataset is provided with the features and labels. In order to determine whether a person will be recruited yes (1) or no (0), classification is needed as the result of this is discrete and binary (0 or 1). Simple Linear Regression is used to predict the salary of a recruited person. Since the salary is a continuous domain, regression is best suited for salary prediction.

We have tried Decision-Tree Model and Random-Forest Model for classification as there are a number of features and they can be handled more efficiently by these models. We have used the Multivariate Linear Regression Model for regression tasks.

CODE OF THE SELECTED ALGORITHM

1. Decision Tree Model

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
classifier1 = DecisionTreeClassifier(criterion='gini', splitter='best')
classifier1.fit(X=X1_train, y=Y1_train)
Y1_pred1 = classifier1.predict(X=X1_test)
```

2. Random Forest Model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
classifier2 = RandomForestClassifier(n_estimators=200, criterion='gini', n_jobs=-1)
classifier2.fit(X1_train, np.ravel(Y1_train))
Y1_pred2 = classifier2.predict(X1_test)
```

3. Multivariate Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
```

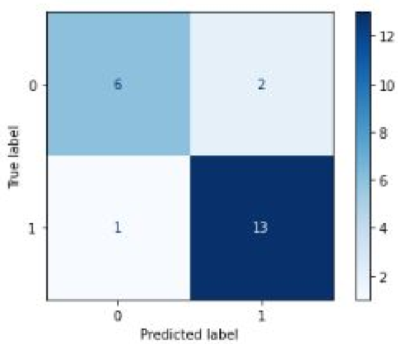
DETAILED OUTPUT OF THE SELECTED ALGORITHM

1. Decision Tree Model

   Output of the Decision Tree Model

```
plot_confusion_matrix(estimator=classifier1, X=X1_test, y_true=Y1_test, include_values=True, cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29d4b38ecc8>
```




```
accuracy_score(y_true=Y1_test, y_pred=Y1_pred1, normalize=True)
```

```
0.8636363636363636
```

```
print(classification_report(y_true=Y1_test, y_pred=Y1_pred1))
```

```
              precision    recall  f1-score   support

           0       0.86      0.75      0.80         8
           1       0.87      0.93      0.90        14

    accuracy                           0.86        22
   macro avg       0.86      0.84      0.85        22
weighted avg       0.86      0.86      0.86        22
```

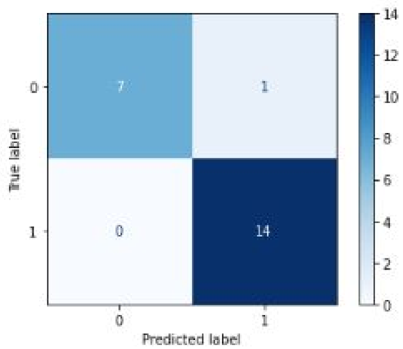
2. Random Forest Model

Output of the Random Forest Model

```
classifier2 = RandomForestClassifier(n_estimators=200, criterion='gini', n_jobs=-1)
classifier2.fit(X1_train, np.ravel(Y1_train))
Y1_pred2 = classifier2.predict(X1_test)
```

```
plot_confusion_matrix(estimator=classifier2, X=X1_test, y_true=Y1_test, include_values=True, cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29d44b51a88>
```




```
accuracy_score(Y1_test, Y1_pred2)
```

```
0.9545454545454546
```

```
print(classification_report(Y1_test, Y1_pred2))
```

```
              precision    recall  f1-score   support

           0       1.00      0.88      0.93         8
           1       0.93      1.00      0.97        14

    accuracy                           0.95        22
   macro avg       0.97      0.94      0.95        22
weighted avg       0.96      0.95      0.95        22
```

3. Multivariate Linear Regression Model

Output of the Multivariate Linear Regression Model

9.7.2 Multivariate Linear Regression

In [74]:
```
regressor = LinearRegression(n_jobs=-1)
regressor.fit(Xp2_train, Y2_train)
Y2_pred = regressor.predict(Xp2_test)
```

In [75]:
```
# coefficient of determination
print('R2_Score: ', regressor.score(Xp2_test, Y2_test))   #  Best possible score is 1.0
```

```
R2_Score:  -0.4658057929086359
```

In [76]:
```
regressor_OLS = sm.OLS(endog=Y2_train, exog=Xp2_train).fit()
print(regressor_OLS.summary())
```

```
                                 OLS Regression Results                                
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.002
Model:                            OLS   Adj. R-squared (uncentered):             -0.006
Method:                 Least Squares   F-statistic:                             0.2114
Date:                Sat, 06 Jun 2020   Prob (F-statistic):                       0.646
Time:                        21:25:48   Log-Likelihood:                         -56.197
No. Observations:                 127   AIC:                                      114.4
Df Residuals:                     126   BIC:                                      117.2
Df Model:                           1                                                  
Covariance Type:            nonrobust                                                  
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.0180      0.039      0.460      0.646      -0.059       0.095
==============================================================================
Omnibus:                       21.292   Durbin-Watson:                   0.563
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               25.927
Skew:                           1.050   Prob(JB):                     2.34e-06
Kurtosis:                       3.698   Cond. No.                         1.00
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

VISUALIZATION OF THE RESULTS

Classification

Dimension Reduction

```
In [51]:
# dimension reduction: useful for plotting
pca1 = PCA(n_components=2)
Xp2_train = pca1.fit_transform(X1_train)
Xp2_test = pca1.transform(X1_test)
print(pca1.explained_variance_ratio_)
```

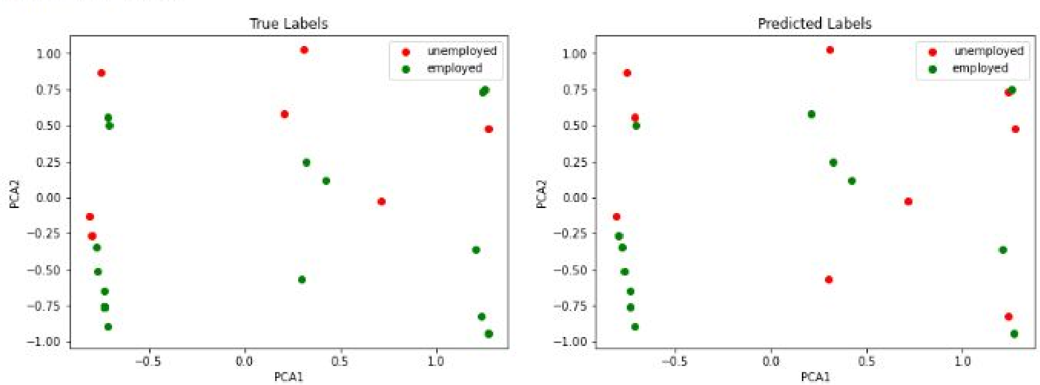
```
[0.39968699 0.16426418]
```

1. Decision Tree Model

Scatter Plot visualization for Decision Tree Model

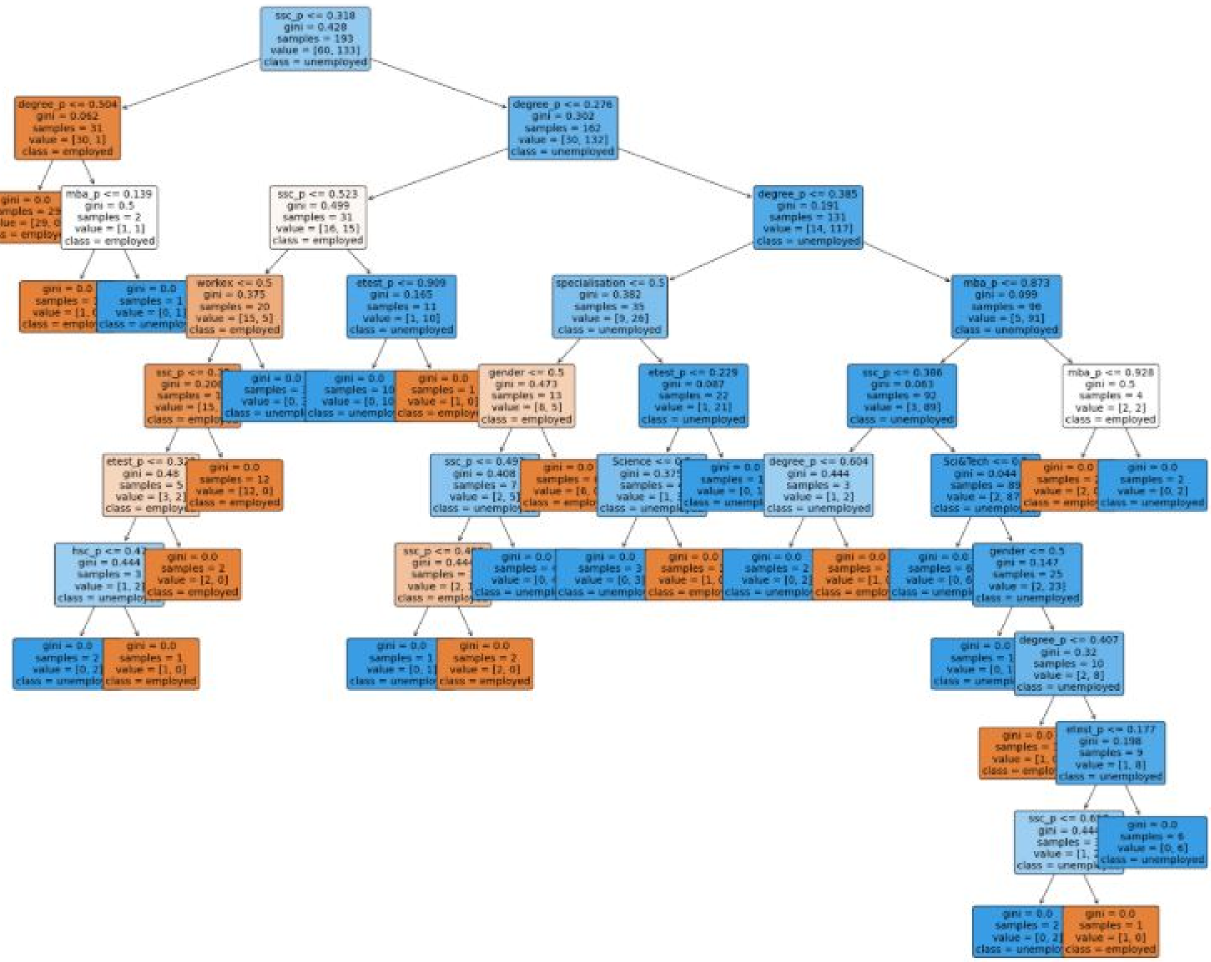
```
In [57]:
plt.figure(figsize=(15, 5))
cdict = {0:'unemployed', 1:'employed'}
color = {0:'red', 1:'green'}


ax = plt.subplot(1, 2, 1)
for g in np.unique(list(cdict.keys())):
    ix = np.where(Y1_test==g)
    ax.scatter(x=Xp2_test[ix, 0], y=Xp2_test[ix, 1], c=color[g], label=cdict[g])
ax.legend()
plt.title('True Labels')
plt.xlabel('PCA1')
plt.ylabel('PCA2')


ax = plt.subplot(1, 2, 2)
for g in np.unique(list(cdict.keys())):
    ix = np.where(Y1_pred1==g)
    ax.scatter(x=Xp2_test[ix, 0], y=Xp2_test[ix, 1], c=color[g], label=cdict[g])
ax.legend()
plt.title('Predicted Labels')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
```

```
Out[57]: Text(0, 0.5, 'PCA2')
```



Decision Tree generated by the Model

In [56]:
```
plt.figure(figsize=(30, 25))
tree1 = plot_tree(decision_tree=classifier1, feature_names=data1.columns[:-1], class_names=['employed', 'unemployed'], filled=True, rounded=True, fontsize=14)
```

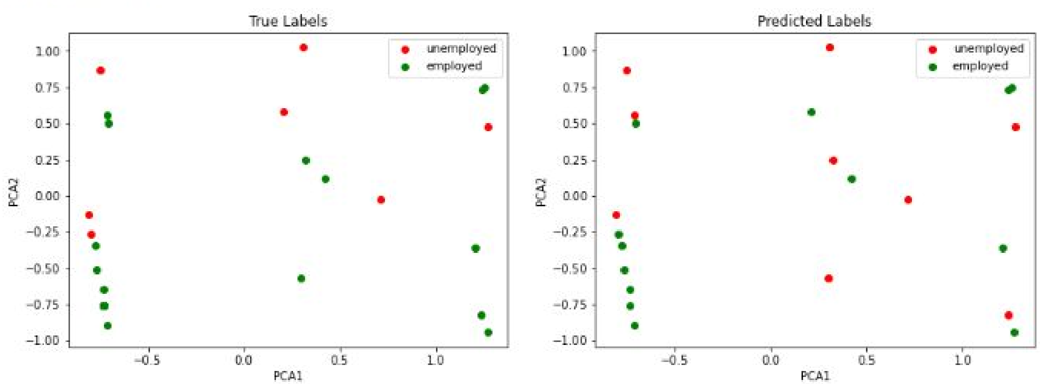


2. Random Forest Model

Scatter Plot visualization for Random Forest Model

In [62]:
```python
plt.figure(figsize=(15, 5))
cdict = {0:'unemployed', 1:'employed'}
color = {0:'red', 1:'green'}


ax = plt.subplot(1, 2, 1)
for g in np.unique(list(cdict.keys())):
    ix = np.where(Y1_test==g)
    ax.scatter(x=Xp2_test[ix, 0], y=Xp2_test[ix, 1], c=color[g], label=cdict[g])
ax.legend()
plt.title('True Labels')
plt.xlabel('PCA1')
plt.ylabel('PCA2')


ax = plt.subplot(1, 2, 2)
for g in np.unique(list(cdict.keys())):
    ix = np.where(Y1_pred2==g)
    ax.scatter(x=Xp2_test[ix, 0], y=Xp2_test[ix, 1], c=color[g], label=cdict[g])
ax.legend()
plt.title('Predicted Labels')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
```

Out[62]: Text(0, 0.5, 'PCA2')



3. Multivariate Linear Regression Model

9.7.1 Dimension Reduction

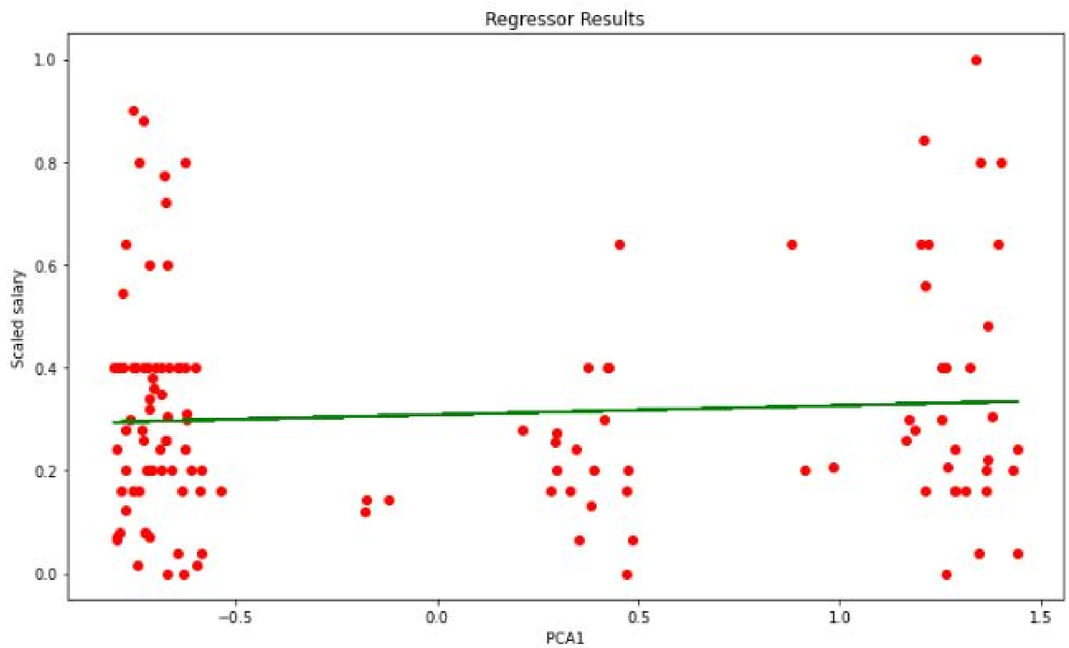
In [73]:
```python
pca2 = PCA(n_components=1)
Xp2_train = pca2.fit_transform(X2_train)
Xp2_test = pca2.transform(X2_test)
print(pca2.explained_variance_ratio_)
```

```
[0.40242528]
```

Scatter Plot visualization for Multivariate Linear Regression Model

In [77]:
```
plt.figure(figsize=(12, 7))
plt.scatter(x=Xp2_train, y=Y2_train, color="red")
plt.plot(Xp2_train, regressor.predict(Xp2_train), color="green")
plt.title("Regressor Results")
plt.xlabel("PCA1")
plt.ylabel("Scaled salary")
```

Out[77]: Text(0, 0.5, 'Scaled salary')



Interpret the results obtained

- Classification Task :-
  - Random-Forest Classifier :
    - Mean Accuracy of Random-Forest Classifier is 0.95.
  - Decision-Tree Classifier :
    - Mean Accuracy of Decision-Tree Classifier is 0.8

Hence it can be stated as the accuracy of Random-Forest Classifier is greater than Decision-Tree Classifier & provides with good results.

- Regression Task :-
  - Adjusted R square:
    - Value is -0.006
  - Multivariate Linear Regression :
    - R2 score is -0.466.

Hence, the Regressor Model fits the best line possible to the dataset after dimension reduction of features.

BI Decision that can be taken based on the results obtained

- The model helps the students to know where they stand and based on their grades and projects.
- The model solves the problems for companies as the selection of ideal candidates becomes faster and easier.
- Requirement specific decisions can be made and candidates are chosen accordingly.
- The model also is used to predict the salary of the candidates in the companies based on his grades and projects.
- The result of this application proposes that the model can be used in various colleges to aid the students.