LNMIIT
The LNM Institute of
Information Technology

INTRODUCTION TO DATA SCIENCE CSE327

“Applying ML Classification Algorithm & Data Pre-Processing ,Data Analysing On Placement Dataset”

Project Report By
Arin Khandelwal (21UCS027)
Kaushal Singh Rao (21UCC058)
Piyush Bharadwaj (21UCC075)
Yash Gupta (21UCC119)

Course Instructors
Dr. Subrat Dash
Dr. Aloke Datta
Dr. Lal Upendra Singh

Department of Computer Science & Engineering
The LNM Institute of Information Technology

Table of Contents

PROBLEM STATEMENT

We need to collect a dataset from the given website and perform the following steps:

1. Data pre-processing .
2. Analysing the data and doing its visualization
3. Explain all the inferences we got from our data.
4. Explain why and which ML Classification Algorithms are being used.
5. Implementing those algorithms and representing them.
6. Output the result of the testing set and its visualization using plots.

Objective

Placement of a student depends on the various attributes of his/her performance in the past. We used some of those attributes to create a classification model which predicts if a student will get placed.

The code analyses a placement dataset, visualizing and exploring various factors influencing job placement. It pre-processes data, applies label encoding, and uses logistic regression, k-nearest neighbours, and support vector machine classifiers to predict placement status, evaluating model performance with metrics such as accuracy, precision, and recall.

About Dataset

This data set consists of Placement data of students on the basis of secondary and higher secondary school percentage, specialization and gender. It also includes degree specialization, type and Work experience and salary offered to the placed students.

Data Attributes:

- Serial Number
- Gender
- Secondary Education Percentage
- Board Of Secondary Education
- Higher Secondary Education Percentage
- Board of Higher Secondary Education
- Specialization in Higher Secondary Education
- Degree Percentage
- Under Graduation Field of Degree

Code: https://github.com/technology1520/IDS_Project

Source of DataSet : https://www.kaggle.com/benroshan/factors-affecting-campus-placement

(All the above information about dataset is taken from this link as well)

DATA ANALYSIS

- First, we import all the necessary ***libraries*** which will be required in our code.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score
```

- Next, we read the dataset in a variable and output its first 5 rows, using **pandas.read_csv( ).**

```
data_set = pd.read_csv('Placement_Data_Full_Class.csv')
data_set.head()
```

	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

- Using the **data_set.drop( ),** drops 'sl_no' and 'salary' columns from 'data_set', displaying information about the modified dataframe 'pl_df'.

```
pl_df = data_set.drop(['sl_no', 'salary'], axis=1)
pl_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   gender          215 non-null    object
 1   ssc_p           215 non-null    float64
 2   ssc_b           215 non-null    object
 3   hsc_p           215 non-null    float64
 4   hsc_b           215 non-null    object
 5   hsc_s           215 non-null    object
 6   degree_p        215 non-null    float64
 7   degree_t        215 non-null    object
 8   workex          215 non-null    object
 9   etest_p         215 non-null    float64
 10  specialisation  215 non-null    object
 11  mba_p           215 non-null    float64
 12  status          215 non-null    object
dtypes: float64(5), object(8)
memory usage: 22.0+ KB
```

data_set.columns ,retrieves and displays the list of column names present in the 'data_set', showing the headers or features available.

```
[4] data_set.columns
```

```
Index(['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'],
      dtype='object')
```

- We also use **DataFrame.describe( ).transpose(),** generates descriptive statistics (mean, min, max, etc.) for numerical columns in 'pl_df', transposing rows and columns for readability.

```
pl_df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
ssc_p	215.0	67.303395	10.827205	40.89	60.600	67.0	75.700	89.40
hsc_p	215.0	66.333163	10.897509	37.00	60.900	65.0	73.000	97.70
degree_p	215.0	66.370186	7.358743	50.00	61.000	66.0	72.000	91.00
etest_p	215.0	72.100558	13.275956	50.00	60.000	71.0	83.500	98.00
mba_p	215.0	62.278186	5.833385	51.21	57.945	62.0	66.255	77.89

data_set['degree_t'].value_counts(normalize=True), counts occurrences of unique values in the 'degree_t' column within 'data_set', presenting the normalized proportion of each distinct value.

```
[ ] data_set['degree_t'].value_counts(normalize=True)
```

```
Comm&Mgmt    0.674419
Sci&Tech     0.274419
Others       0.051163
Name: degree_t, dtype: float64
```

- We use **data_set.isna().sum()**checks 'data_set' for missing values, summing the count of NaN (null) values across each column, indicating their respective quantities.

```
data_set.isna().sum()
```

```
sl_no             0
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
```

INFERENCE:

In our dataset, we can observe the following things:

- The code checks for missing values in the dataset using data_set.isna().sum().
- No missing values are observed in the dataset, indicating a complete dataset with no null entries for any variable.
- This ensures a robust foundation for analysis and modeling without the need for imputation or handling missing data.

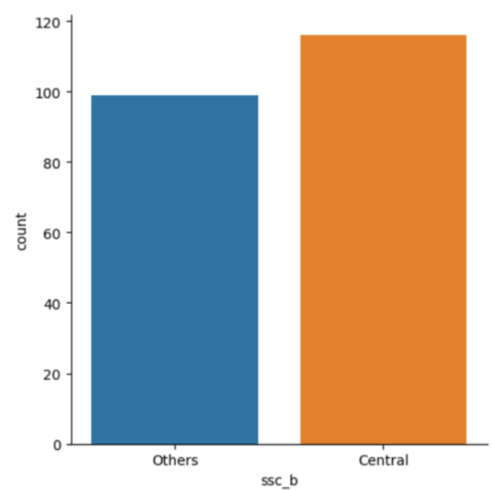
So we can begin with our next part, **Data Visualization,** to get inferences from the dataset.

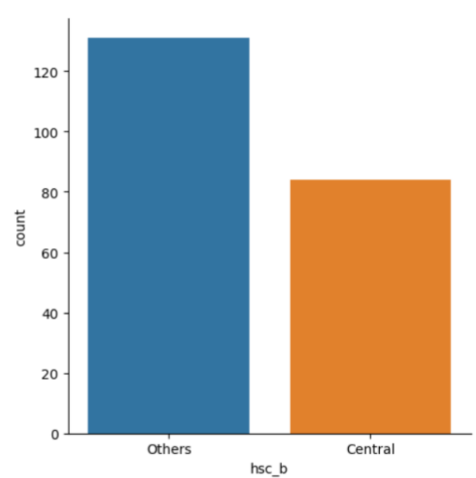
DATA VISUALIZATION

- First, we use **DataFrame.catplot(),** creates categorical plots, allowing visualization of relationships between variables using categories, often used with seaborn for creating categorical plots like boxplots, count plots, etc.

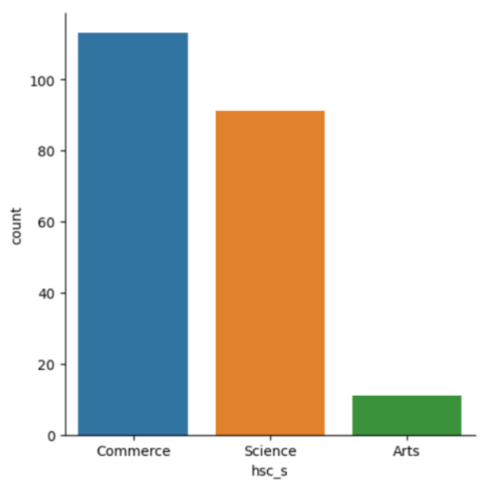
```
sb.catplot(x='ssc_b', kind='count', data=data_set)
```

```
[ ] sb.catplot(x='hsc_b', kind='count', data=data_set)

<seaborn.axisgrid.FacetGrid at 0x78b67ac35ae0>
```

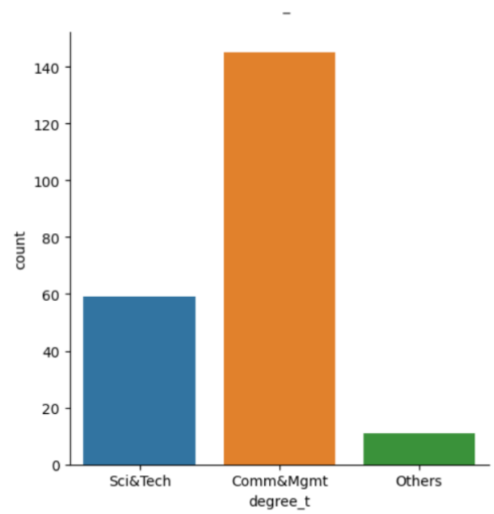




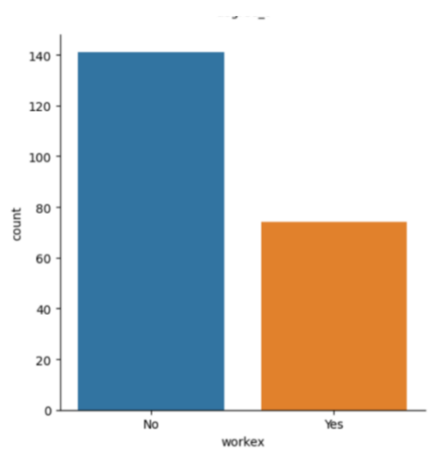
```
[ ] sb.catplot(x='hsc_s', kind='count', data=data_set)
```



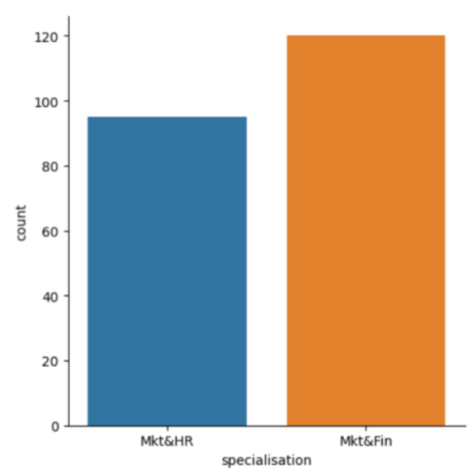

```
[ ] sb.catplot(x='degree_t', kind='count', data=data_set)
```



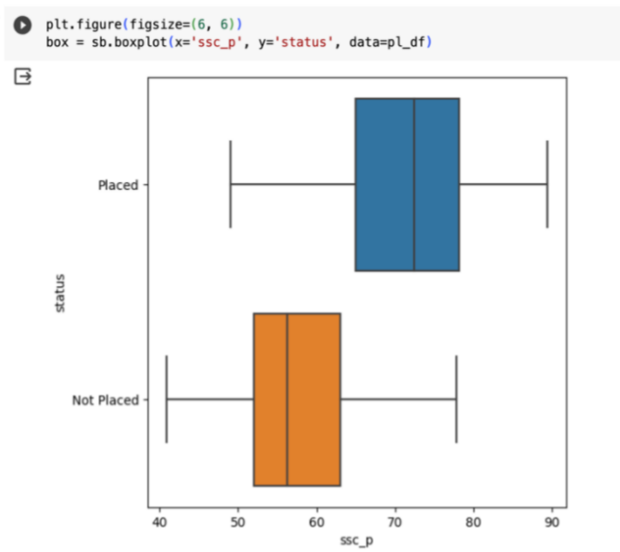

```
[ ] sb.catplot(x='workex', kind='count', data=data_set)
```

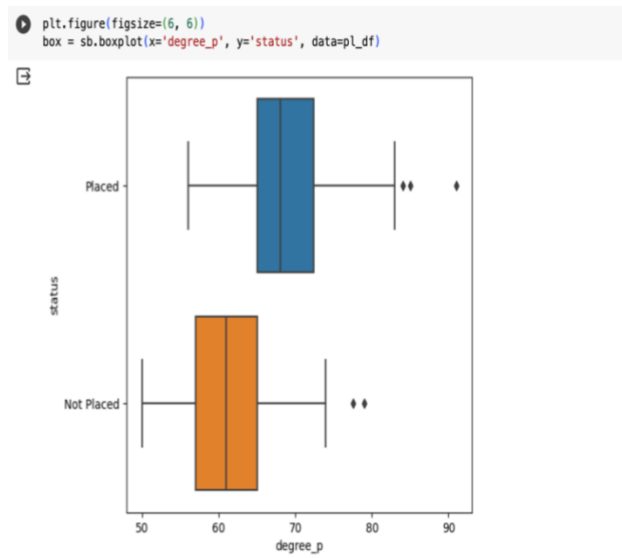


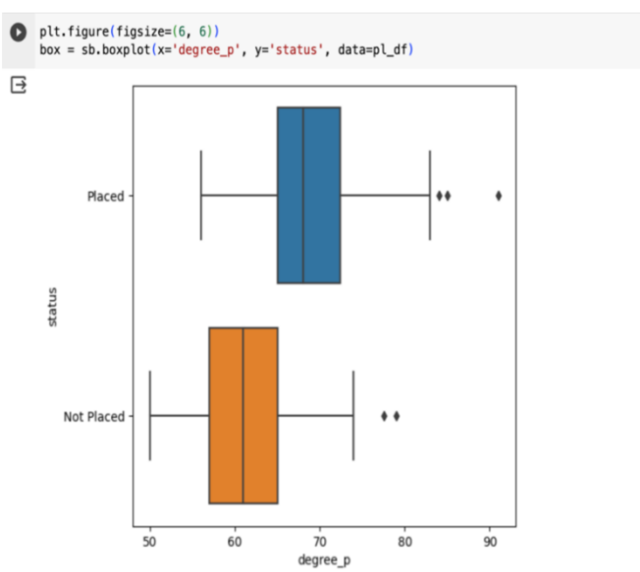

```
[ ] sb.catplot(x='specialisation', kind='count', data=data_set)
```

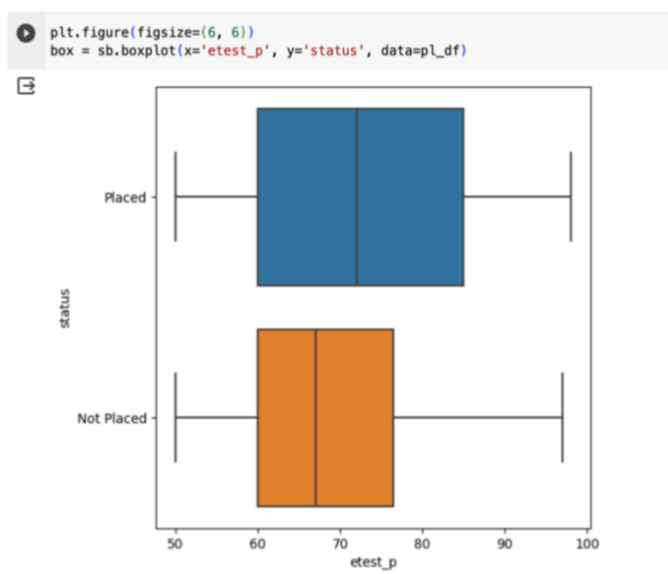


dataframe.boxplot() , creates a box plot for the specified x and y variables using the provided data in the dataframe, displaying statistical information and identifying outliers.

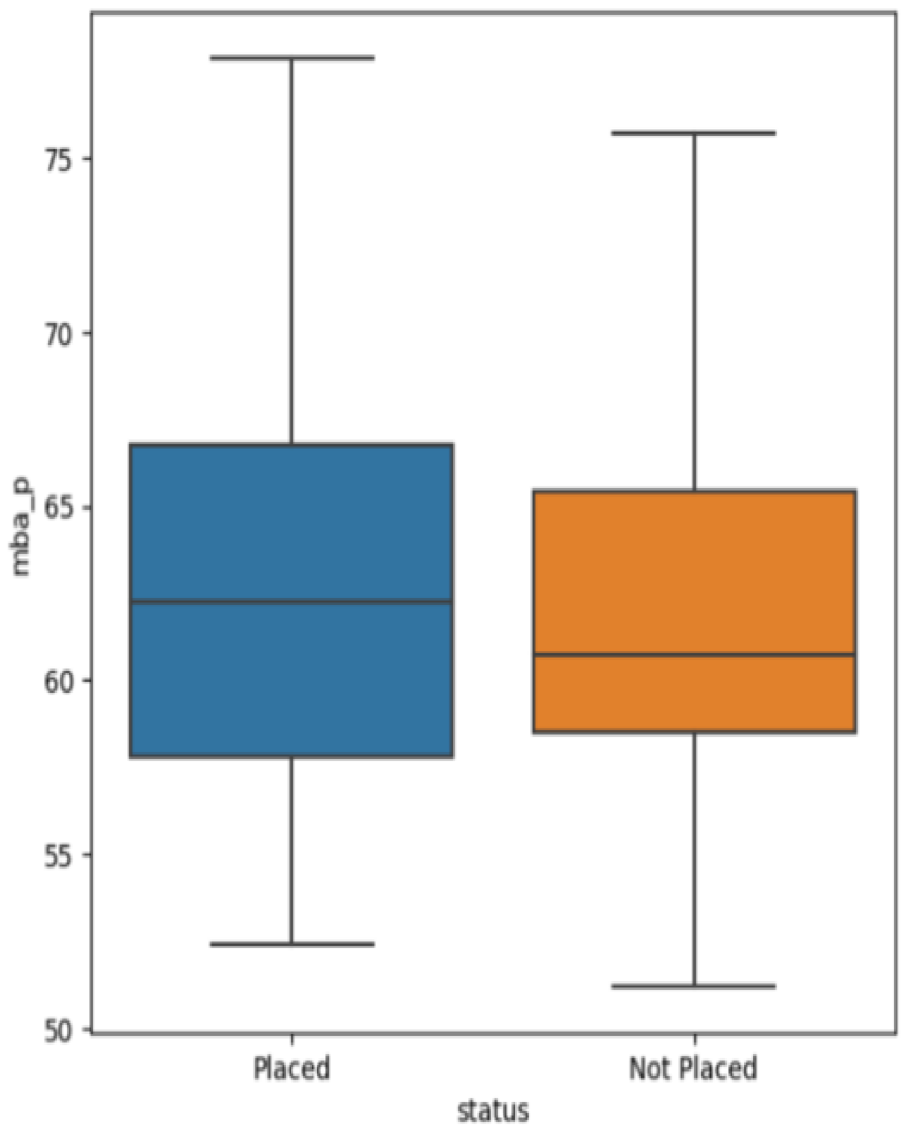






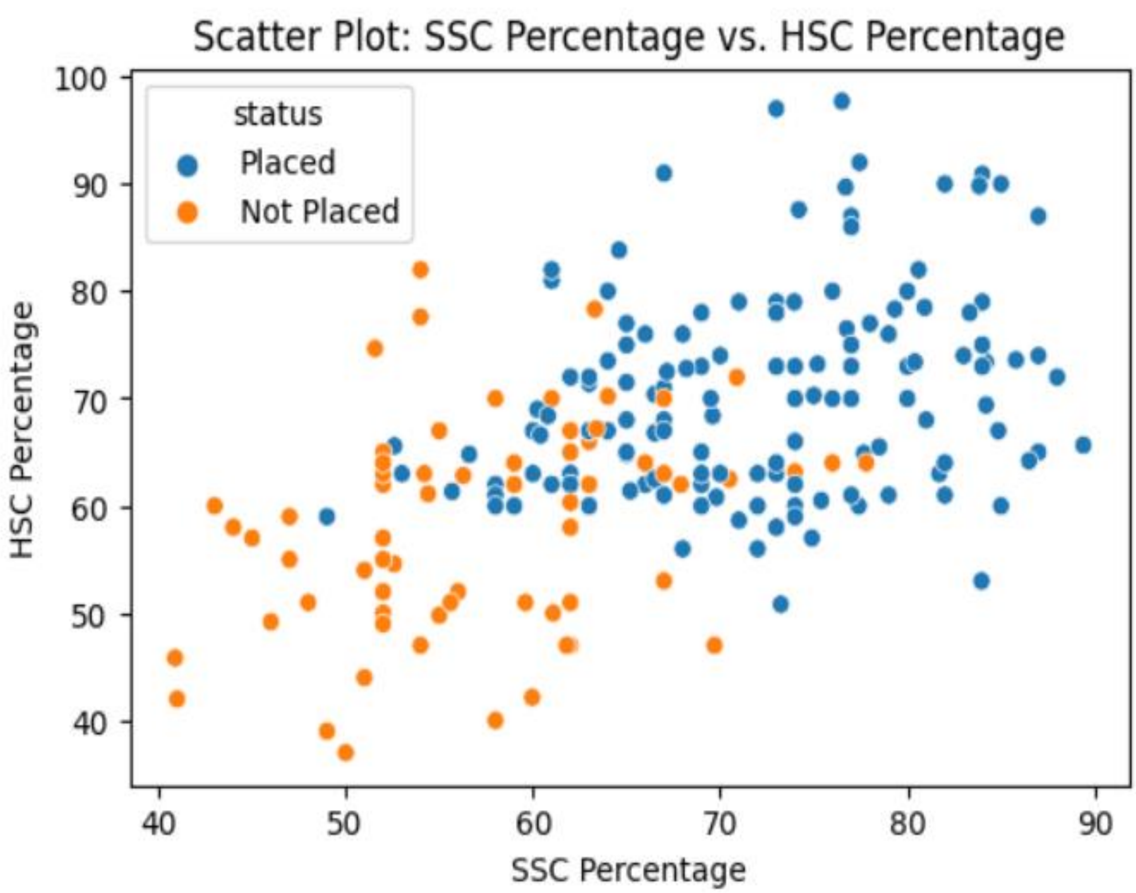


```
plt.figure(figsize=(6, 6))
box = sb.boxplot(y='mba_p', x='status', data=pl_df)
```

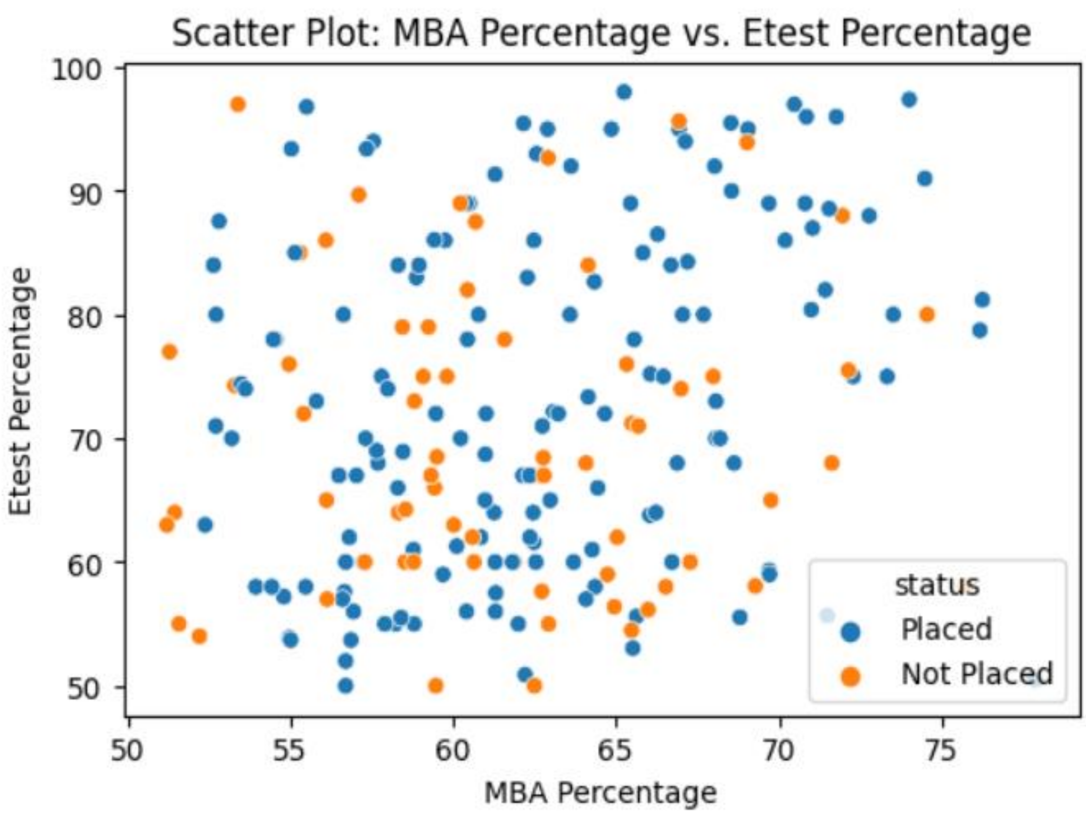


Dataframe.scatterplot() , generates a scatter plot using data from a dataframe, visualizing the relationship between two variables through their respective data points.

```
[ ] # Scatter plot for 'mba_p' vs 'etest_p'
    plt.figure(figsize=(12, 8))
    sb.scatterplot(x='mba_p', y='etest_p', hue='status', data=pl_df)
    plt.title('Scatter Plot: MBA Percentage vs. Etest Percentage')
    plt.xlabel('MBA Percentage')
    plt.ylabel('Etest Percentage')
    plt.show()
```

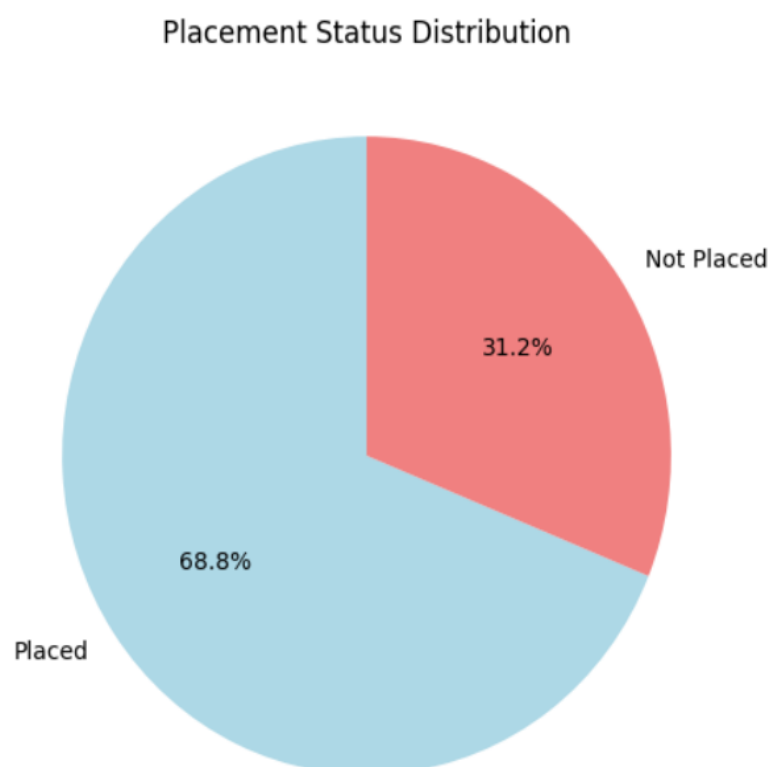


```
[ ] # Scatter plot for 'mba_p' vs 'etest_p'
    plt.figure(figsize=(12, 8))
    sb.scatterplot(x='mba_p', y='etest_p', hue='status', data=pl_df)
    plt.title('Scatter Plot: MBA Percentage vs. Etest Percentage')
    plt.xlabel('MBA Percentage')
    plt.ylabel('Etest Percentage')
    plt.show()
```



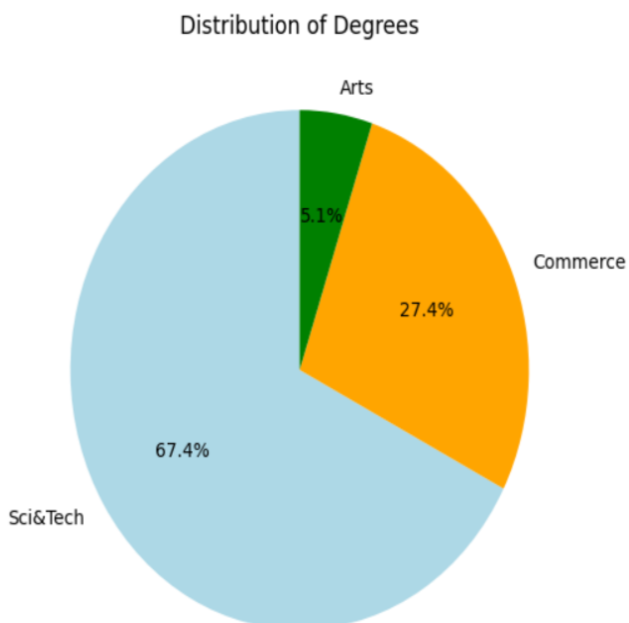
Dataframe.pie(), creates a pie chart using 'status' data in 'pl_df', illustrating the distribution of 'Placed' and 'Not Placed' categories with percentage labels and distinct colors.

```
[ ] plt.figure(figsize=(10, 6))
    status_counts = pl_df['status'].value_counts()
    labels = ['Placed', 'Not Placed']
    colors = ['lightblue', 'lightcoral']
    explode = (0.1, 0)  # explode the 1st slice (i.e., 'Placed')

    plt.pie(status_counts, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
    plt.title('Placement Status Distribution')
    plt.show()
```



Generating a pie chart depicting the distribution of **'degree_t'** categories ('Sci&Tech', 'Commerce', 'Arts') in **'pl_df'**, showcasing percentages with distinct colors.

```
# Plotting a pie chart
plt.figure(figsize=(10, 6))
degree_counts = pl_df['degree_t'].value_counts()
labels = ['Sci&Tech', 'Commerce', 'Arts']
colors = ['lightblue', 'orange', 'green']

plt.pie(degree_counts, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors)
plt.title('Distribution of Degrees')
plt.show()
```



INFERENCE:

Box Plots:

- Higher SSC, HSC, and degree percentages tend to positively influence placement status, with placed candidates having higher median scores.
- Etest scores show variability, but successful placements exhibit slightly higher median values.
- MBA percentages vary, with placements having a wider distribution.

Scatter Plots:

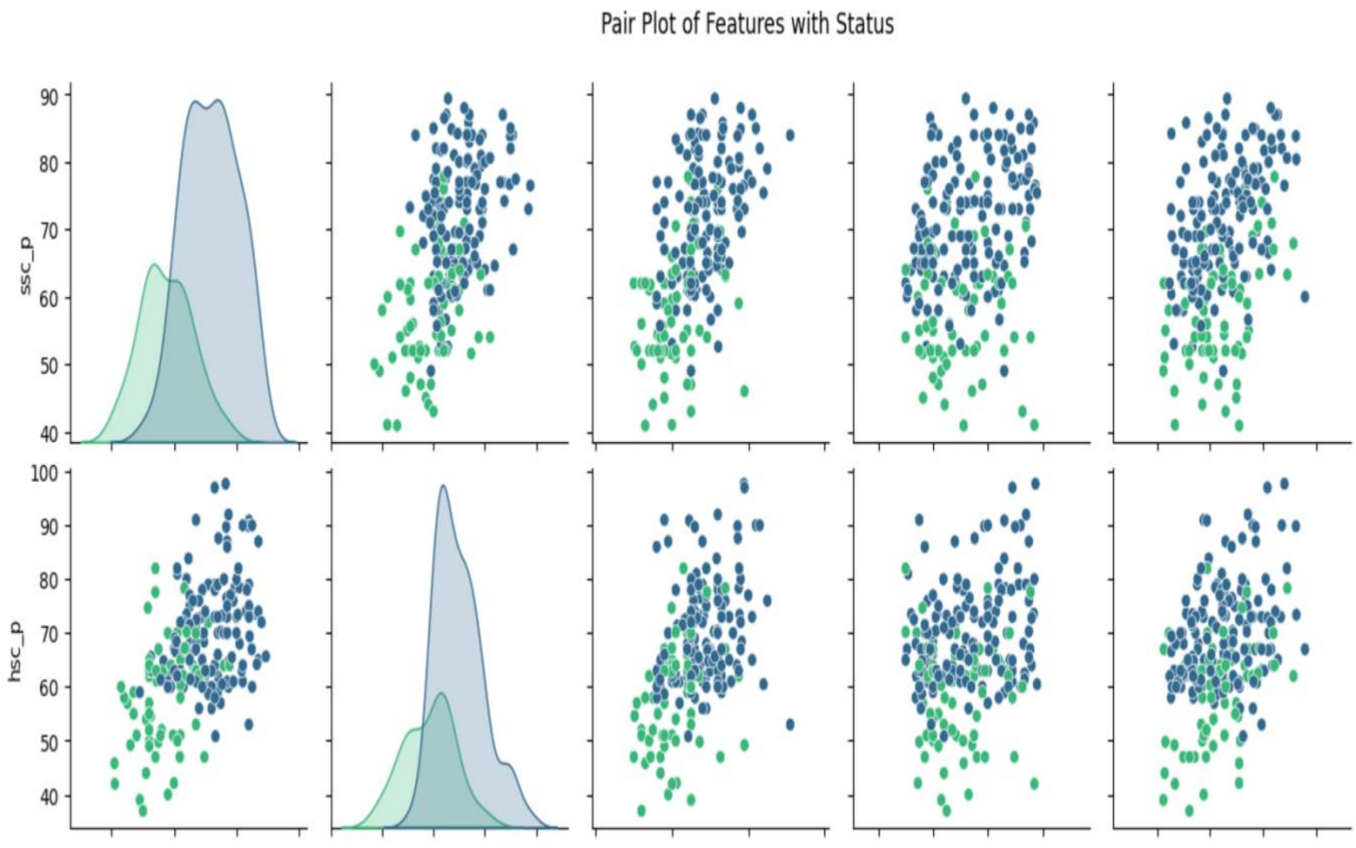
- A positive correlation between SSC and HSC percentages is evident for both placed and non-placed candidates.
- MBA percentages and Etest scores do not distinctly separate placed and non-placed candidates.

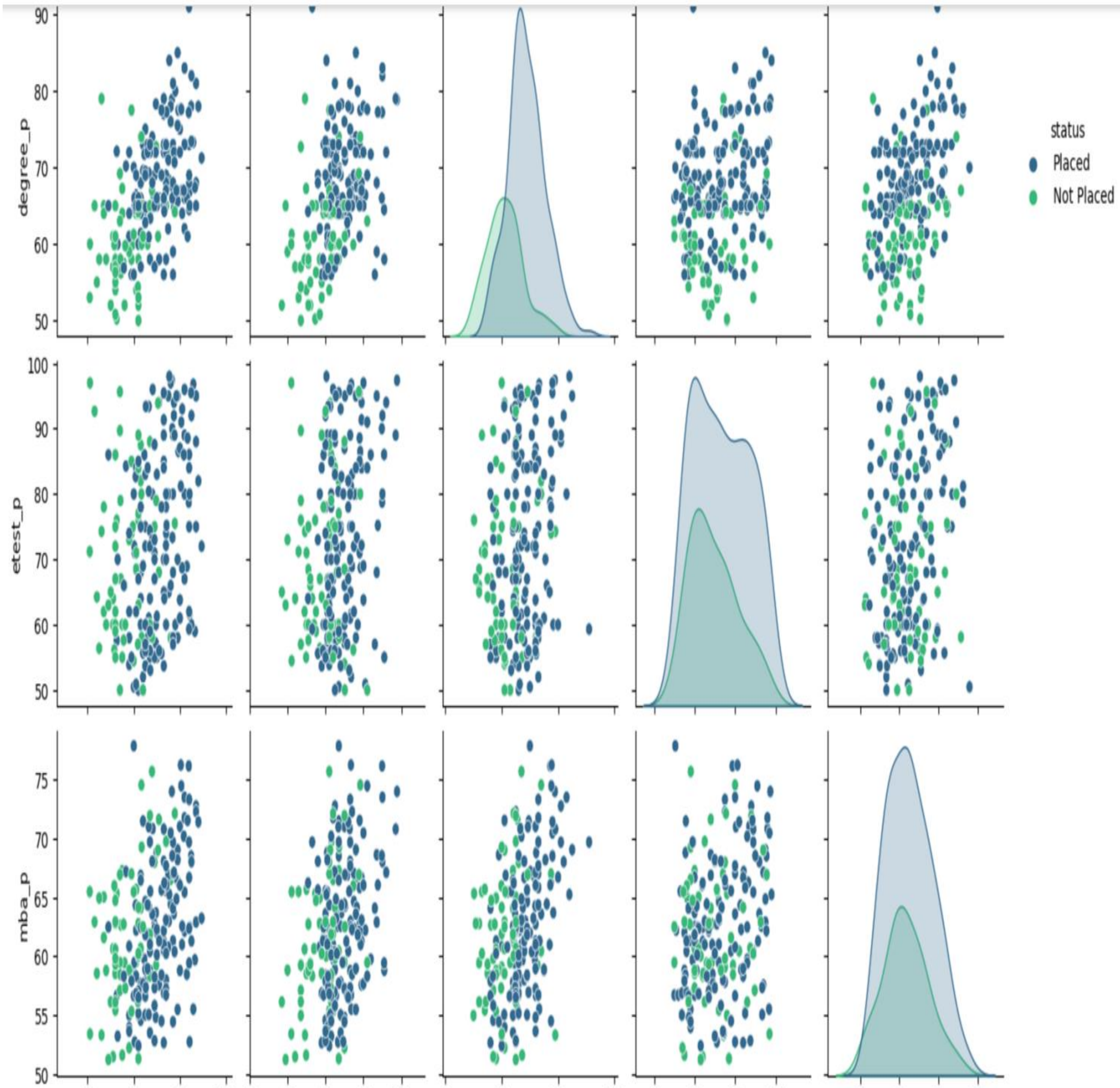
Pie Charts:

- In the placement status distribution, around 68.1% are placed, indicating a generally successful outcome.
- Degree distribution shows a prevalence of Science and Commerce degrees, with Science and Technology being the most common.

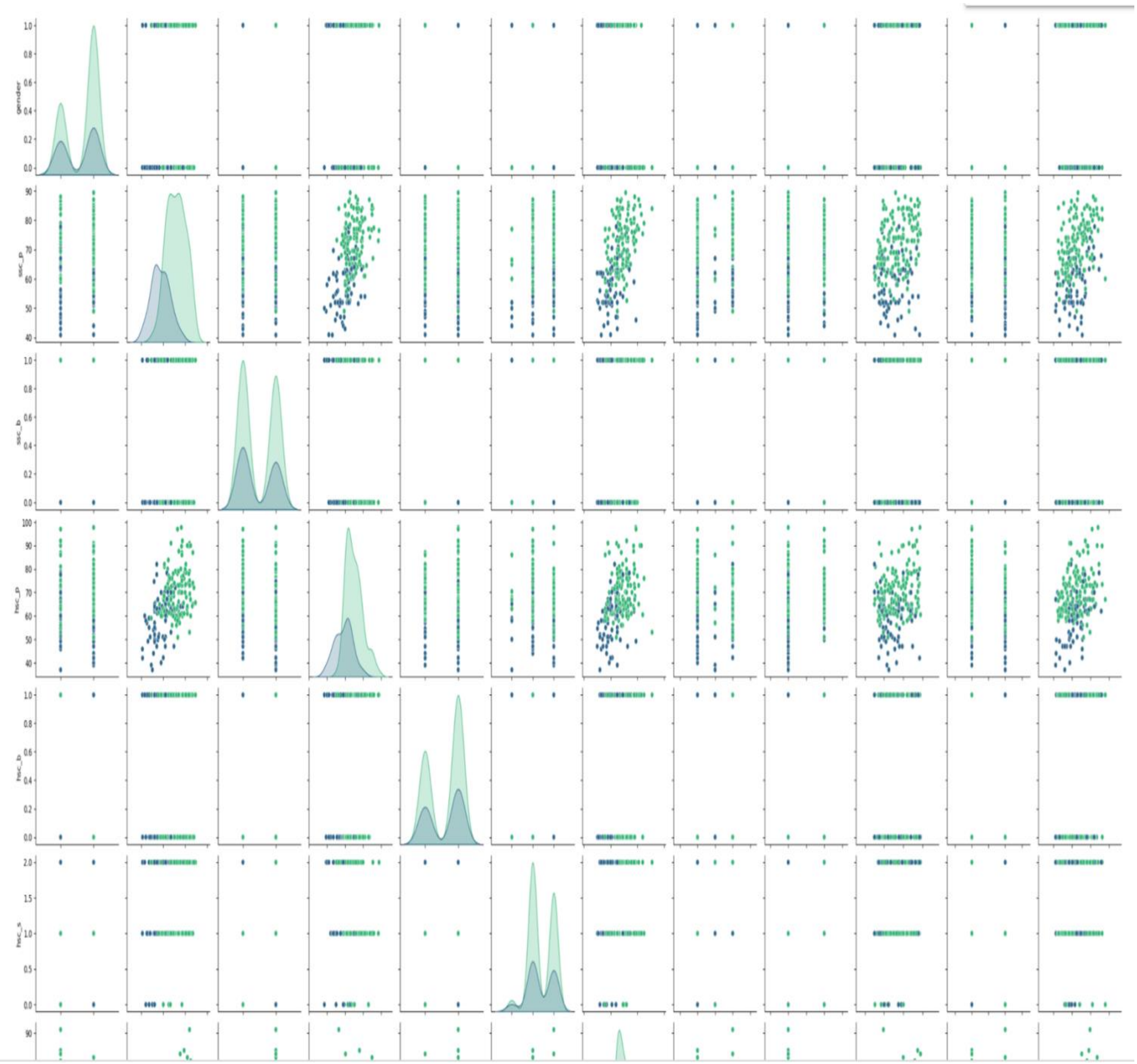
```
[26] # Pair plot
     plt.figure(figsize=(15, 15))
     sb.pairplot(pl_df, hue='status', palette='viridis')
     plt.suptitle('Pair Plot of Features with Status', y=1.02)
     plt.show()
```

```
<Figure size 1500x100 with 0 Axes>
```





degree_p
etest_p
mba_p
status
Placed
Not Placed



gender
ssc_p
ssc_b
hsc_p
hsc_b
hsc_s

INFERENCE:

ssc_p vs. hsc_p:

Higher percentages in SSC and HSC are associated with a higher likelihood of placement.

degree_p vs. etest_p:

Positive correlation suggests candidates with higher degree percentages and E-test scores have better placement outcomes.

mba_p vs. etest_p:

No clear pattern, indicating MBA and E-test percentages may not strongly influence placement status.

ssc_p vs. etest_p:

No distinct pattern, suggesting a lack of strong correlation between SSC percentages and E-test scores in relation to placement.

hsc_p vs. mba_p:

Placement outcomes vary across different combinations of HSC and MBA percentages, with no clear linear trend.

ssc_p vs. mba_p:

Similar to HSC vs. MBA, placement outcomes show variability with different combinations of SSC and MBA percentages.

hsc_p vs. etest_p:

No clear trend, indicating a lack of strong correlation between HSC percentages and E-test scores in relation to placement.

These visualizations help identify potential relationships between feature pairs and the placement status of candidates.

```
pl_df
```

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

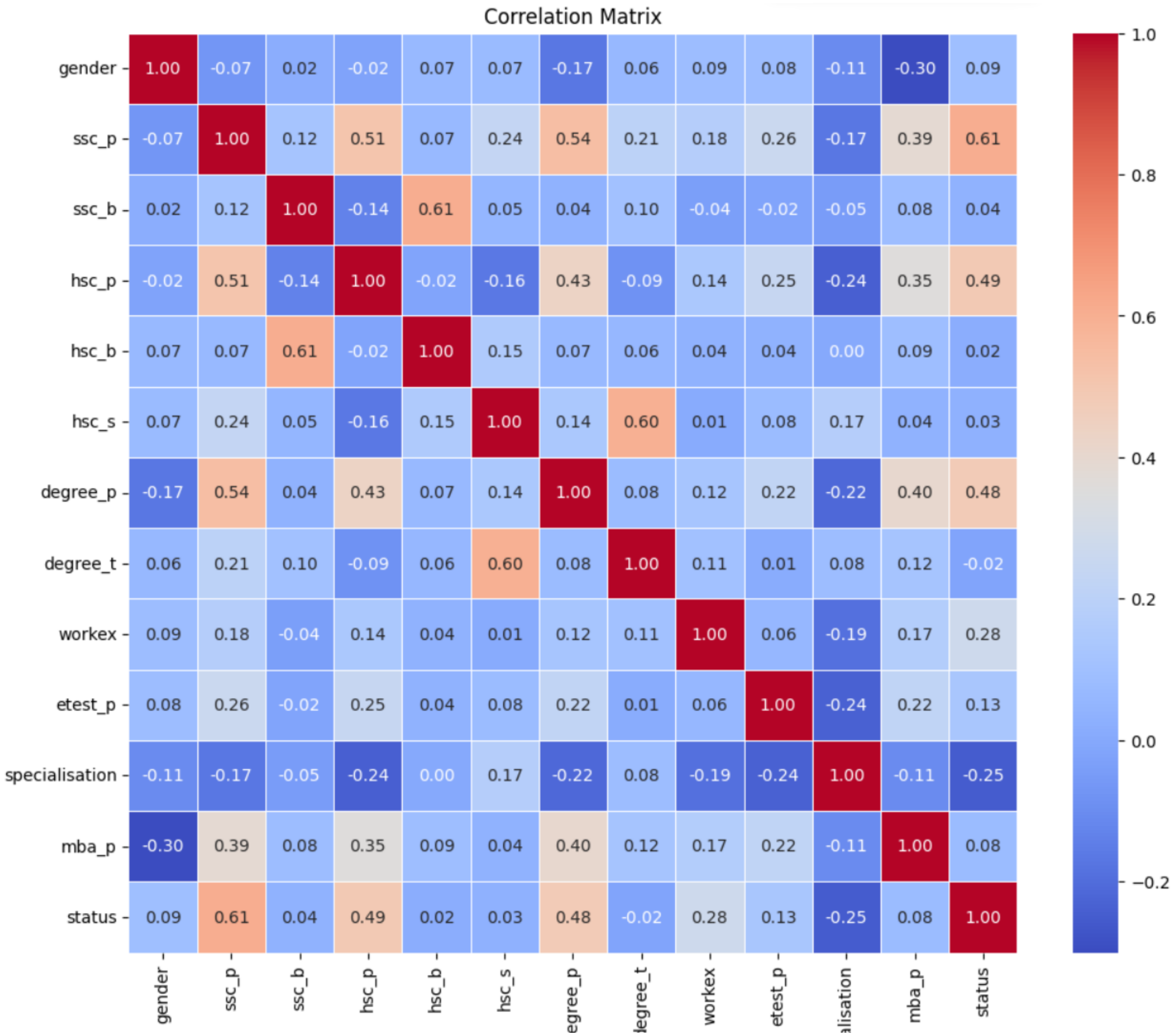
215 rows × 13 columns

```
[ ] pl_df.duplicated().sum()
```

```
0
```

- After inferencing the dataset from our graphs, we move towards the mathematical values of correlation between the attributes.
- For that, we plot the **Correlation Matrix,** using **DataFrame.corr()** in the **seaborn.heatmap()** function.

```python
# Calculate the correlation matrix
correlation_matrix = pl_df.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 10))

# Draw the heatmap using seaborn
sb.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

# Set the title of the plot
plt.title('Correlation Matrix')

# Show the plot
plt.show()
```



INFERENCE:

- The correlation matrix heatmap reveals moderate positive correlations between academic percentages (ssc_p, hsc_p, degree_p), suggesting a potential link between academic performance. Employment status shows weak correlations, indicating independence from academic scores.

DATA PRE-PROCESSING

- After thoroughly analyzing the dataset through visuals(graph plots), we process our dataset a bit.
- We display the counts of our class variables.
- We apply **LabelEncoder()** to convert our labels(“good”, “bad”) into numeric form (as ‘0’ and ‘1’).
- This is used to encode the output variable “y”.

```
[ ] # Encode categorical variables
    labels = LabelEncoder()
```

```
pl_df['gender'] = labels.fit_transform(pl_df['gender'])
pl_df['ssc_b'] = labels.fit_transform(pl_df['ssc_b'])
pl_df['hsc_b'] = labels.fit_transform(pl_df['hsc_b'])
pl_df['hsc_s'] = labels.fit_transform(pl_df['hsc_s'])
pl_df['degree_t'] = labels.fit_transform(pl_df['degree_t'])
pl_df['workex'] = labels.fit_transform(pl_df['workex'])
pl_df['specialisation'] = labels.fit_transform(pl_df['specialisation'])
pl_df['status'] = labels.fit_transform(pl_df['status'])
```

```
[ ] pl_df
```

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

215 rows × 13 columns

ML CLASSIFICATION ALGORITHMS

- After getting a cleaned dataset, we can now apply our prediction algorithms, to predict the quality of our wine.
- In our project, instead of applying just one, we use 5 different classification algorithms to predict the results.
- The motivation to apply all these algorithms was that we wanted to compare their accuracy results to see which algorithm works better on our dataset.

We applied the algorithms given below.

  - Logistic regression
  - Decision Tree Classifier
  - Support Vector Machine

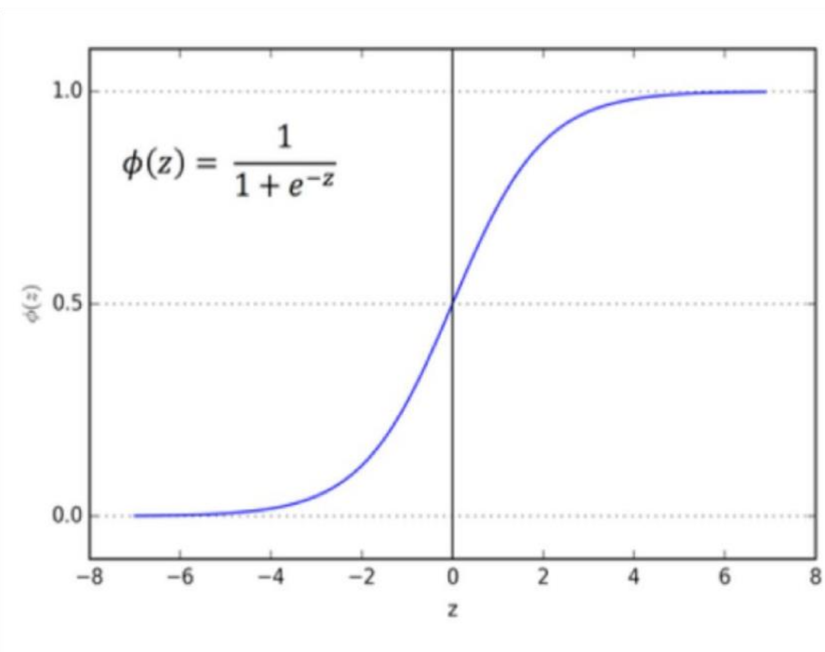
- For each case, we've visualized the Confusion Matrix along with it.
- We've also displayed the accuracy percentage for each case too.
- We have also represented accuracy in the form of pie chart.

Logistic Regression

In its fundamental structure, Logistic Regression is a model based on statistics. It is used when we have a categorical dependent variable(y), instead of continuous

This model is used to calculate the probability of a certain class. It can also be used for multiclass attribute values as well.

It basically follows the linear regression model, but the continuous output value is passed through a function called as **"Sigmoid Function",** which is used to scale the value between 0 and 1. A threshold value selected. For our problem which is a 2- Class, if the sigmoid function gives a value greater than threshold, then class 1 is selected, otherwise class 0.



This sigmoid function helps to scale the values between 0-1.

```
[ ] # Train-test split
    X = pl_df.drop(['status'], axis=1)
    Y = pl_df['status']
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=.25, random_state=42)
```

```
[ ] # Logistic Regression
    logreg = LogisticRegression(solver='liblinear')
    logreg.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(solver='liblinear')
```

```
pred_test = logreg.predict(X_test)
pred_test
```

```
array([1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0])
```

```
[ ] print("Logistic Regression Results:")
    print("Test confusion matrix:\n", confusion_matrix(pred_test, y_test))
    print("Accuracy:", accuracy_score(y_test, pred_test) * 100)
    print("Precision:", precision_score(y_test, pred_test) * 100)
    print("Recall:", recall_score(y_test, pred_test) * 100)
```

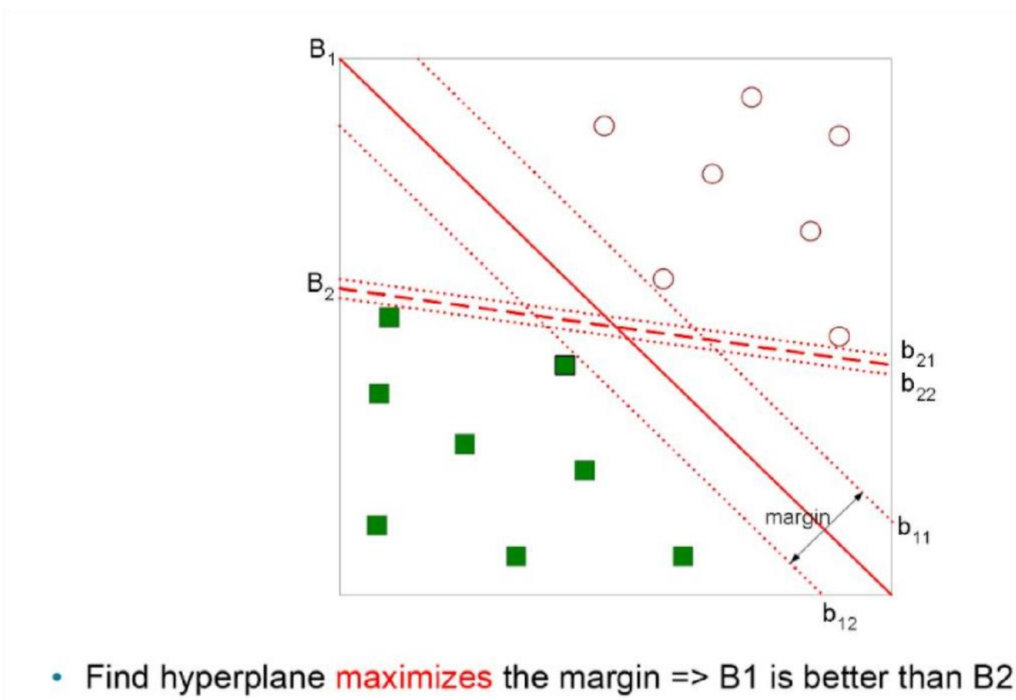
```
Logistic Regression Results:
Test confusion matrix:
 [[11  1]
 [ 3 39]]
Accuracy: 92.5925925925926
Precision: 92.85714285714286
Recall: 97.5
```

Support Vector Machine

Support Vector Machine uses a supervised learning algorithm. It is used to find a hyperplane that will separate the data classes.

Now, there may be many hyperplanes which can separate the data, but SVM tries to find the best fit line for this separation.

This classifier generally works well when there is a clear line of separation between the classes, and the dataset is not large enough.



Structure of SVM

```python
# SVM Classifier
svc = SVC()
svc.fit(X_train, y_train)
```

```
▾ SVC
SVC()
```

```python
svc_pred = svc.predict(X_test)
svc_pred
```

```
array([1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 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])
```

```python
print("\nSVM Classifier Results:")
print("Test confusion matrix:\n", confusion_matrix(svc_pred, y_test))
print("Accuracy:", accuracy_score(y_test, svc_pred) * 100)
print("Precision:", precision_score(y_test, svc_pred) * 100)
print("Recall:", recall_score(y_test, svc_pred) * 100)
```

```

SVM Classifier Results:
Test confusion matrix:
 [[ 5  2]
 [ 9 38]]
Accuracy: 79.62962962962963
Precision: 80.85106382978722
Recall: 95.0
```

KNN CLASSIFIER

```
[ ] # KNN Classifier
    classifier = KNeighborsClassifier(n_neighbors=8)
    classifier.fit(X_train, y_train)
```

```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=8)
```

```
[ ] y_pred = classifier.predict(X_test)
    y_pred
```

```
array([1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 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])
```

```
print("\nKNN Classifier Results:")
print("Test confusion matrix:\n", confusion_matrix(y_pred, y_test))
print("Accuracy:", accuracy_score(y_test, y_pred) * 100)
print("Precision:", precision_score(y_test, y_pred) * 100)
print("Recall:", recall_score(y_test, y_pred) * 100)
```

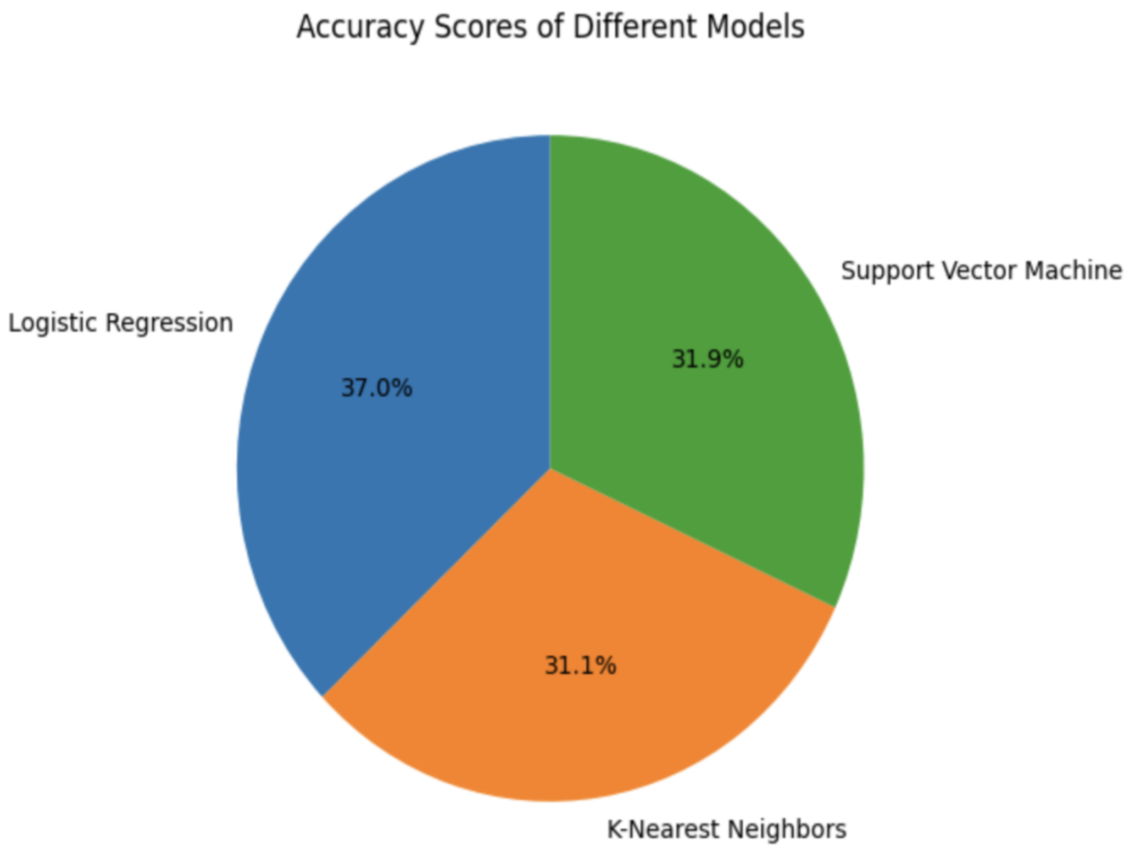
```
KNN Classifier Results:
Test confusion matrix:
 [[ 5  3]
 [ 9 37]]
Accuracy: 77.77777777777779
Precision: 80.43478260869566
Recall: 92.5
```

```
[ ] classifiers = {
        'Logistic Regression': LogisticRegression(solver='liblinear'),
        'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=8),
        'Support Vector Machine': SVC()
    }
    results = {'Model': [], 'Accuracy': [], 'Precision': [], 'Recall': []}
```

```
for name, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred) * 100
    precision = precision_score(y_test, y_pred) * 100
    recall = recall_score(y_test, y_pred) * 100

    results['Model'].append(name)
    results['Accuracy'].append(accuracy)
    results['Precision'].append(precision)
    results['Recall'].append(recall)

    print(f"{name} - Accuracy: {accuracy:.2f}%, Precision: {precision:.2f}%, Recall: {recall:.2f}%")
```

```
Logistic Regression - Accuracy: 92.59%, Precision: 92.86%, Recall: 97.50%
K-Nearest Neighbors - Accuracy: 77.78%, Precision: 80.43%, Recall: 92.50%
Support Vector Machine - Accuracy: 79.63%, Precision: 80.85%, Recall: 95.00%
```

```python
# Pie chart for accuracy scores
plt.figure(figsize=(10, 6))
plt.pie(results['Accuracy'], labels=results['Model'], autopct='%1.1f%%', startangle=90)
plt.title('Accuracy Scores of Different Models')
plt.show()
```



CONCLUSION

To summarize our results from the implementations, we get the following table:

Algorithm	Accuracy
Logistic Regression	92.59%
KNN Classifier	77.78%
Support Vector Machine	79.63%

From the results, it’s clear that **Logistic Regression** gives the highest accuracy of **92.59 %** on our dataset.