27. Classification Algorithm

- The classification algorithm is used to idenitfy the category of new observations on the basis of training data
- In classification, a program learns from the given dataset or observations and the classifies new observation into a number of classes or groups
- Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as targets/labels or categories
- The output is in discrete nature

There are two types of classifications:

1. **Binary Classifier:** If the classification problem has only two possible outcomes, then it is called as Binary Classifier.

- Example: Spam or Not spam, cat or dog etc.

2. **Multi-class Classifier:** If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.

- Example: Classification of types of crops, classification of type of music

Types of ML Classification Algorithms

1. **Non-linear Models:**

- K-Nearest Neighbours
- Support Vector Machines (SVM)
- Naive Bayes
- Decision Tree Classification
- Random Forest Classification

2. **Linear Models:**

- Logistic Regression
- Support Vector Machines

Please note that Naive Bayes algo and Logistic Regression can only be used for classification. The rest of algos above can be used for classification as well as for regression

Evaluating a Classification Model

1. **Log Loss or Cross-Entropy Loss:** It calculates losses in the model and give output through gradient descent
2. **Confusion Matrix:** It is important. It gives reason why a model is reject even though it was showing accuracy
3. **AUC-ROC Curve:** It tells how good a particular model is working

In [ ]:
```

```

28. Logistic Regression (Practical) (Binary Classification)

- **Logistic Regression** is one of the most popular Machine Learning algorithms, which comes under the **Supervised Learning Technique**
- It is used for predicting the **categorical dependent variables** using a given set of independent variables
- Therefore, the coutcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or false,etc. but instead of giving the exact value as 0 and 1, **it gives the probablisitic values which lie b/w 0 and 1.**
- The data should be linearly separable

Types of Logistic Regression

On the basis of **categories**, Logistic Regression can be classified into three types:

1. **Binomial:** In binomial logistic regression, there can be two possible types of the dependent variables, such as 0 or 1, Pass or Fail etc.
2. **Multinomial:** In multinomial logistic regression, there can be 3 or more possible **unordered** types of the dependent variables, such as cat, dog or sheep
3. **Ordinal:** In ordinal logistic regression, there can be 5 or more possible **ordered** types of dependent varaibles, such as low, medium or high

- In logistic regression, the prediction is done through **Sigmoid algorithm**

No description has been provided for this image

Logistic Regression Equation

The logistic regression equation can be obtained from the Linear Regress Model. The mathematical steps to get Logistic Regression equation are given below:

$$y = \frac{1}{1 + e^{-x}}$$

where:

- y = dependent variable (Bought Product)
- x = independent variable (Salary) (x= m1x1 + m2x2 + b)
- e = Euler's constant-2.71828

In [ ]:

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
dataset = pd.read_csv(r'Data/Social_Network_Ads.csv')
dataset.head(3)
```

Out[5]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0

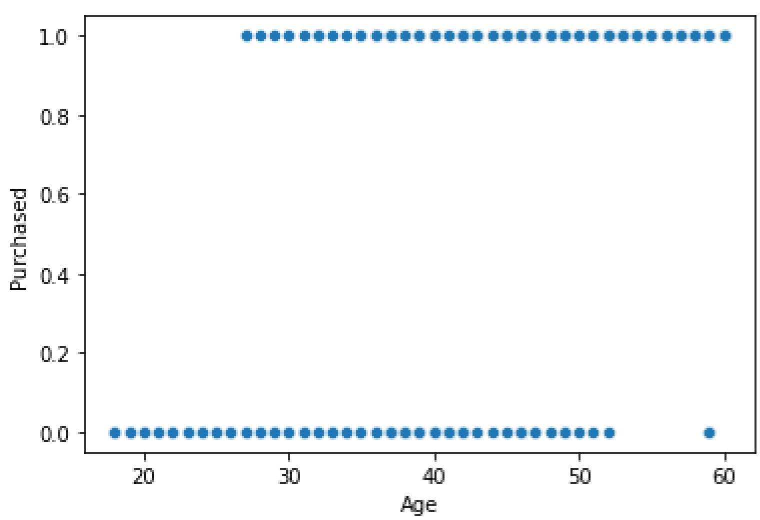
In [9]:
```
# For now, we want to see effect of age on purchase and ignore EstimatedSalary, so
dataset.drop(columns=['EstimatedSalary'], inplace=True)
dataset.head(3)
```

Out[9]:

	Age	Purchased
0	19	0
1	35	0
2	26	0

To see if our data follows Logistic Regression or Not

In [11]:
```
sns.scatterplot(x="Age", y="Purchased", data=dataset)
plt.show()
```



Our data follows logistic regression

1. Next we will split the data into dependent (x) and independent (y) variables

In [13]:
```
# Note that data should be in 2 dimension
x = dataset[['Age']]
y = dataset[['Purchased']]
```

2. Now we will split the data into train and test data

In [14]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.20, random_stat
```

3. Apply Logistic Regression

In [17]:
```
from sklearn.linear_model import LogisticRegression
```

In [18]:
```
lr = LogisticRegression()
lr.fit(x_train, y_train)
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\util
s\validation.py:1111: DataConversionWarning: 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)
```

Out[18]:
```
▾ LogisticRegression
LogisticRegression()
```

5. Check the accuracy of model

In [19]:
```
lr.score(x_test, y_test)*100
```

Out[19]:
```
91.25
```

6. Perform predictions on built model

In [20]:
```
lr.predict([[40]])
```

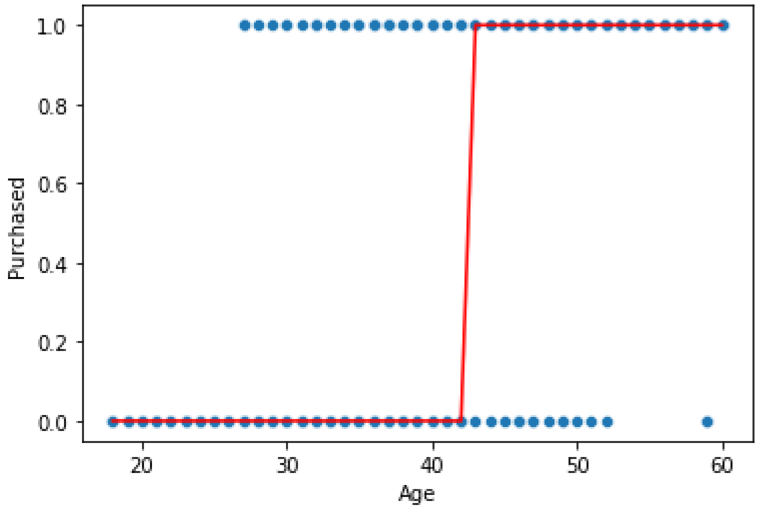
```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[20]:
```
array([0], dtype=int64)
```

In [ ]:

In [22]:

```
sns.scatterplot(x="Age", y="Purchased", data=dataset)
sns.lineplot(x='Age', y=lr.predict(x), data=dataset, color='red')
plt.show()
```



In [ ]:

```

```

29. Logistic Regression (Practical) (Binary Classification) (Multiple Inputs)

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

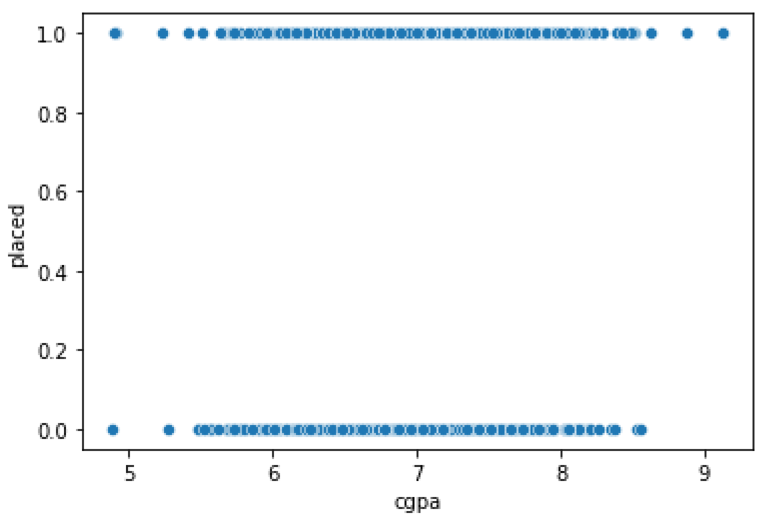
In [2]:
```
dataset = pd.read_csv(r'Data/placement_2.csv')
dataset.head(3)
```

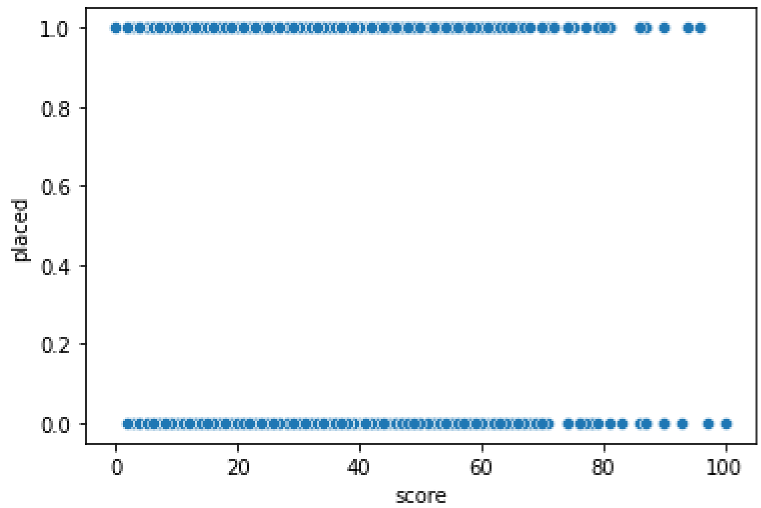
Out[2]:

	cgpa	score	placed
0	7.19	26	1
1	7.46	38	1
2	7.54	40	1

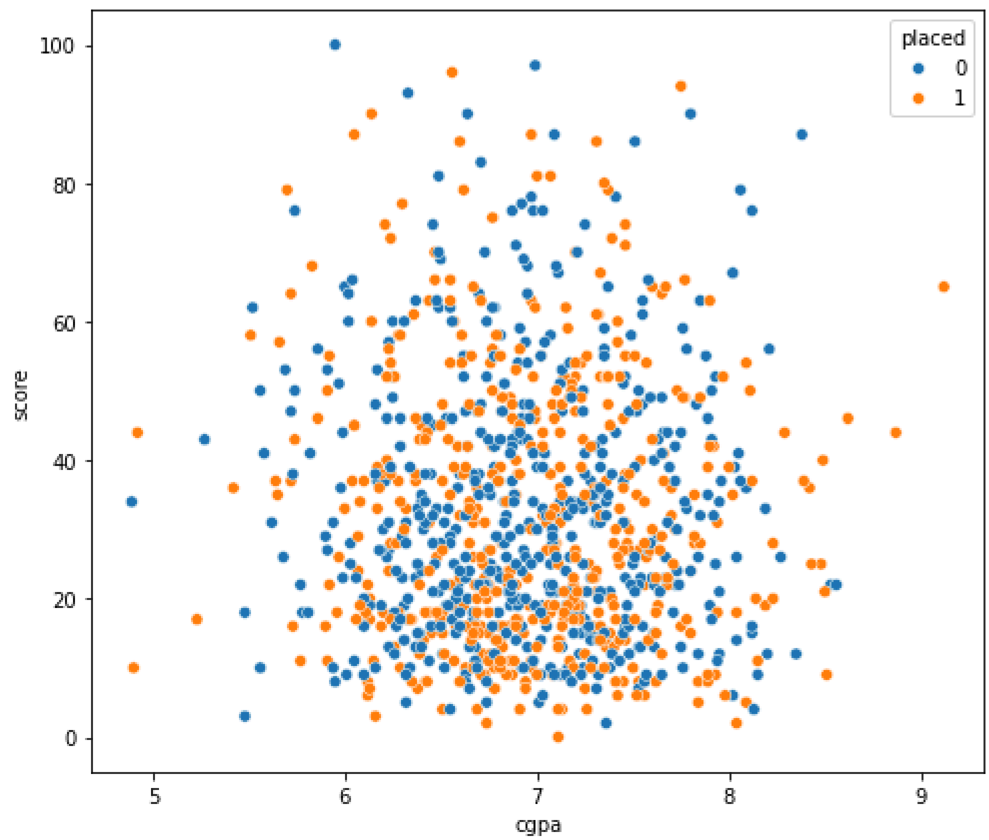
Step 1: Check if the data follows logistic regression

In [9]:
```
sns.scatterplot(x="cgpa", y="placed", data=dataset)
plt.show()
sns.scatterplot(x="score", y="placed", data=dataset)
plt.show()
```





In [24]:
```
plt.figure(figsize=(8,7))
sns.scatterplot(x="cgpa", y="score", data=dataset, hue='placed')
plt.show()
```



Step 2: Split the data into independent/input (x: cgpa, score) and dependent variables/output (y: placed)

In [10]:
```
x = dataset.iloc[:,:-1]
x
```

Out[10]:

	cgpa	score
0	7.19	26
1	7.46	38
2	7.54	40
3	6.42	8
4	7.23	17
...	...	...
995	8.87	44
996	9.12	65
997	4.89	34
998	8.62	46
999	4.90	10

1000 rows × 2 columns

In [11]:

```
y = dataset['placed']
y
```

Out[11]:

```
0      1
1      1
2      1
3      1
4      0
      ..
995    1
996    1
997    0
998    1
999    1
Name: placed, Length: 1000, dtype: int64
```

Step 3: Split the data into training and test data

In [12]:

```
from sklearn.model_selection import train_test_split
```

In [13]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.20, random_stat
```

Step 4: Build Logistic Regression Model

In [14]:

```
from sklearn.linear_model import LogisticRegression
```

In [16]:

```
lr = LogisticRegression()
lr.fit(x_train, y_train)
```

Out[16]:

```
▾ LogisticRegression
LogisticRegression()
```

Step 5: Check the accuracy of Model

In [18]:
```
lr.score(x_test, y_test)*100
```

Out[18]:
```
51.5
```

Step 6: Do prediction based on the built model

In [20]:
```
lr.predict([[6,53]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

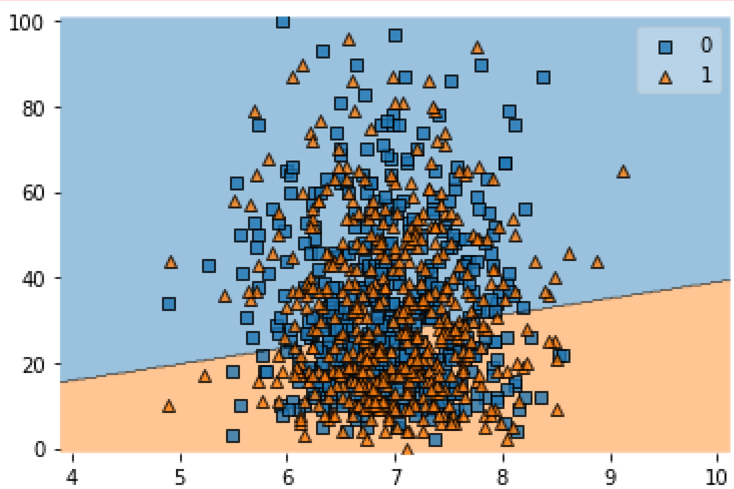
Out[20]:
```
array([0], dtype=int64)
```

Step 7: Create Classifier boundary

In [26]:
```
from mlxtend.plotting import plot_decision_regions
```

In [27]:
```
plot_decision_regions(x.to_numpy(), y.to_numpy(), clf=lr)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```



In [ ]:

30. Logistic Regression (Practical) (Binary Classification) (Polynomial Input))

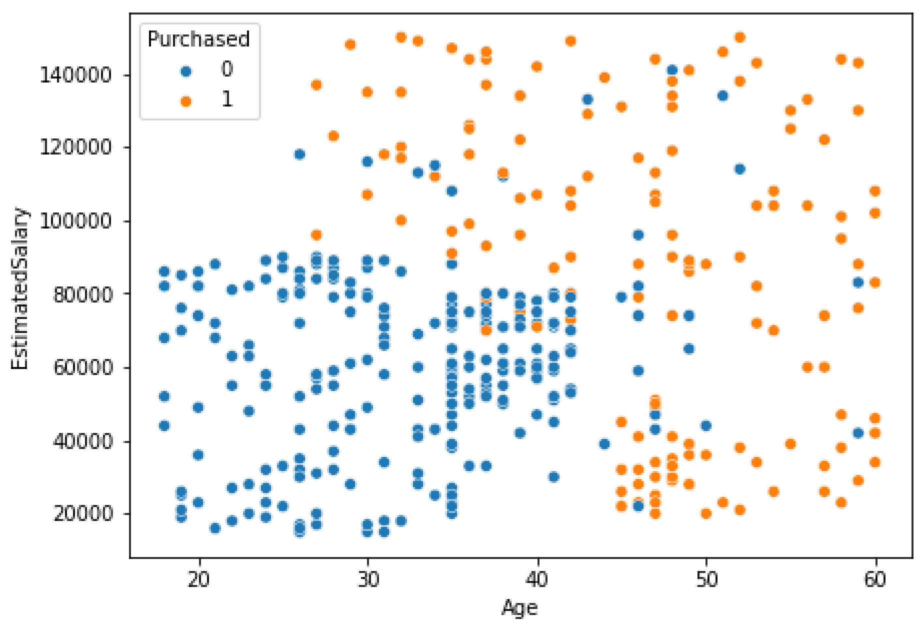
In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
dataset = pd.read_csv(r'Data/Social_Network_Ads_2.csv')
dataset.head(3)
```

Out[5]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0

In [7]:
```
plt.figure(figsize=(7,5))
sns.scatterplot(x="Age", y="EstimatedSalary", data=dataset, hue="Purchased")
plt.show()
```



In [9]:
```
x = dataset.iloc[:,:-1]
x
```

Out[9]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

400 rows × 2 columns

In [10]:

```
y=dataset["Purchased"]
y
```

Out[10]:

```
0      0
1      0
2      0
3      0
4      0
      ..
395    1
396    1
397    1
398    0
399    1
Name: Purchased, Length: 400, dtype: int64
```

In [11]:

```
from sklearn.model_selection import train_test_split
```

In [12]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.20, random_stat
```

In [13]:

```
from sklearn.linear_model import LogisticRegression
```

In [14]:

```
lr = LogisticRegression()
lr.fit(x_train, y_train)
```

Out[14]:

```
▾ LogisticRegression
LogisticRegression()
```

In [16]:
```
lr.score(x_test, y_test)*100
```

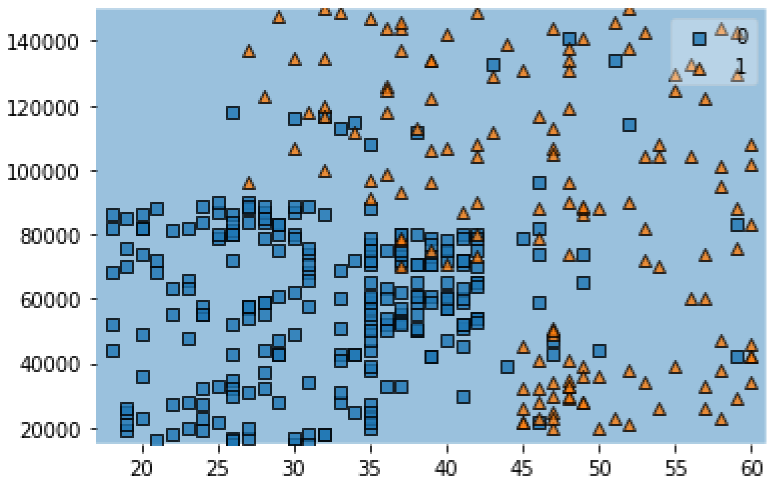
Out[16]: 65.0

In [ ]:

In [19]:
```
from mlxtend.plotting import plot_decision_regions
```

In [20]:
```
plot_decision_regions(x.to_numpy(), y.to_numpy(), clf=lr)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```



You can see the data is not linearly separable. so we will classify this data through **polynomial Feature**

In [21]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [42]:
```
pf = PolynomialFeatures(degree=3)
pf.fit(x)
pf.transform(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12444/2232260164.py in <module>
      1 pf = PolynomialFeatures(degree=3)
      2 pf.fit(x)
----> 3 pf.transform(x)

~\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\preprocessing\_po
lynomial.py in transform(self, X)
    430             # Do as if _min_degree = 0 and cut down array after the
    431             # computation, i.e. use _n_out_full instead of n_output_features
_.
--> 432             XP = np.empty(
    433                 shape=(n_samples, self._n_out_full), dtype=X.dtype, order=se
lf.order
    434             )

ValueError: array is too big; `arr.size * arr.dtype.itemsize` is larger than the max
imum possible size.
```

The output data is in array form so we will convert it into dataset format

In [ ]:
```
x = pd.DataFrame(pf.transform(x))
x.head(5)
```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

In [36]:
```
from sklearn.linear_model import LogisticRegression
```

In [37]:
```
lrg = LogisticRegression()
lrg.fit(x_train, y_train)
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\line
ar_model\_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[37]:
```
▾ LogisticRegression
LogisticRegression()
```

In [38]:
```
lrg.score(x_test, y_test)*100
```

Out[38]:
```
65.0
```

In [ ]:

31. Logistic Regression (Practical) (Multiclass Classification)

- It follows **OVR (One versus Rest) method**.
- for exmaple our data is comprises of cat, dog and cow - so to convert it to one-hot-encoding:
- Cat Dog Cow
- 1 0 0
- 0 1 0
- 0 0 1

In [13]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [14]:
```
dataset = pd.read_csv(r'Data/iris.csv')
dataset.head(3)
```

Out[14]:

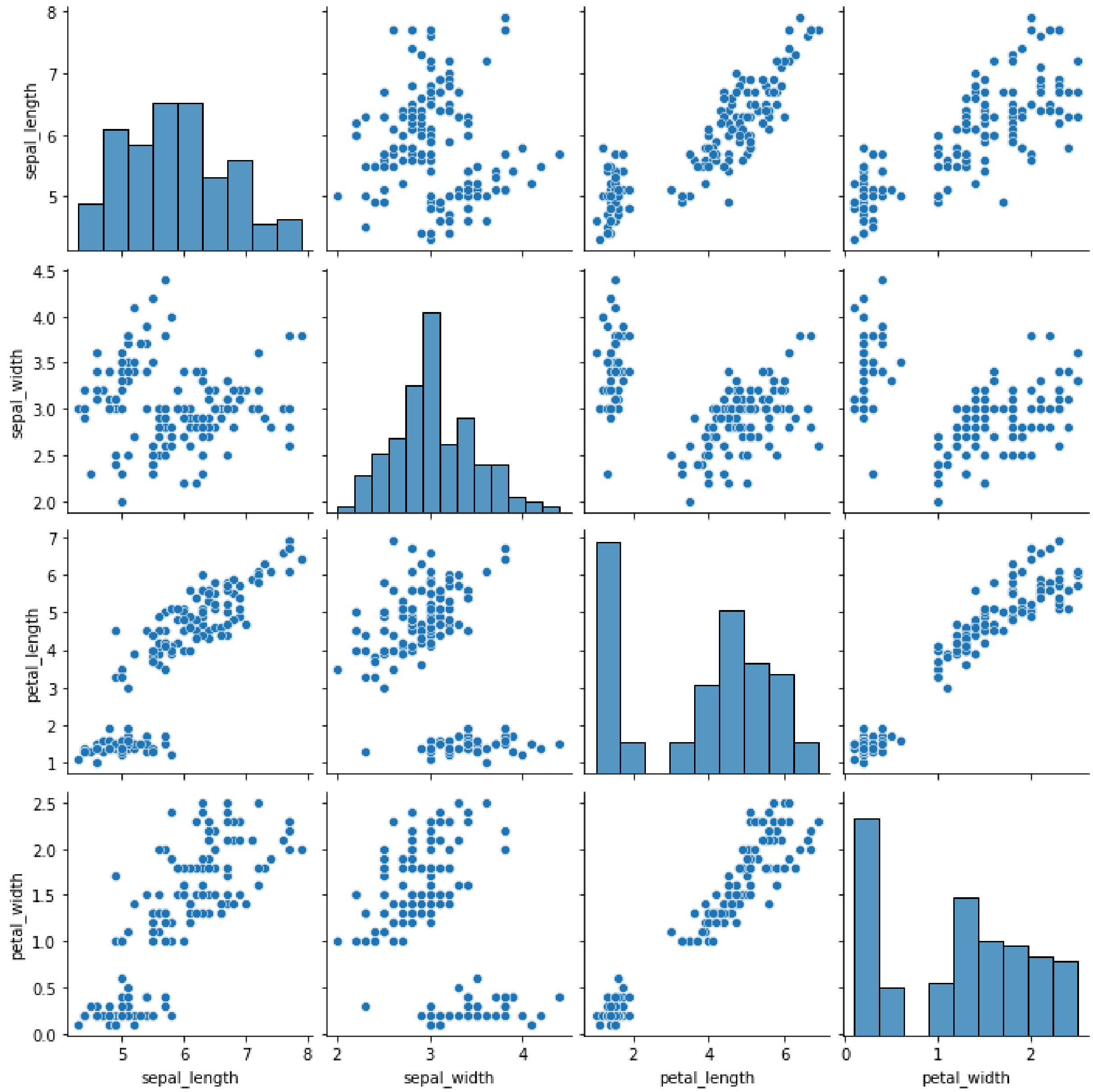
	sepal_length	sepal_width	petal_length	petal_width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

In [15]:
```
dataset["Species"].unique()
```

Out[15]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

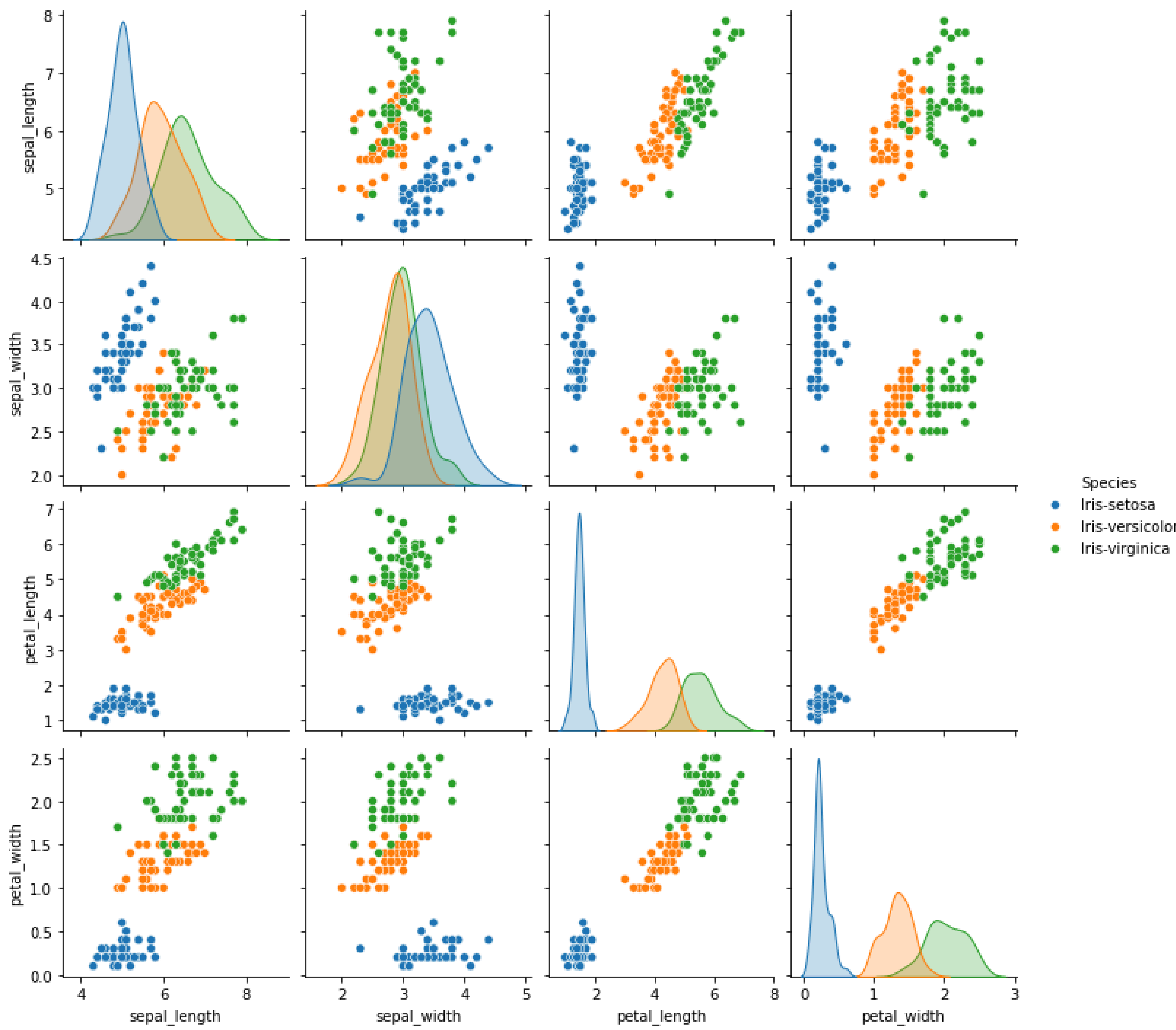
In [16]:
```
sns.pairplot(data=dataset)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [17]:
```
sns.pairplot(data=dataset, hue='Species')
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Interpretation of above graphs

- sepal length - all three curvatures are overlapping with one another, so to find differential is difficult
- sepal width - all 3 curvatures are more overlapping than even those of seapl length
- petal length - all 3 curvatures are distinct and discrete - so we can find differential easily - so we can easily classify them
- petal_width - all 3 curvatures are distinct and discrete - so we can find differential easily - so we can easily classify them
- From these curvatures, we can take information for feature(s) selection, which feature to take and which to drop for building model so that the built model will be accurately trained
- sepal_width is pretty much overlapped, so we cannot use this features for model training - it should be dropped

- but as for this notebook, feature_selection is not the topic so we will keep all the features for now
- the focus of this exercise is logistic regression with multiclass features selection

Separate dependent (ouput, y) and independent variables (input, x)

In [19]:
```
x = dataset.iloc[:,:-1]
x
```

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [20]:
```
y = dataset['Species']
y
```

Out[20]:
```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
              ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

Split the data into train and test data

In [21]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

Apply Logistic Regression through OVR Method

multi_class{'auto', 'ovr', 'multinomial'}, default='auto' If the option chosen is 'ovr', then a binary problem is fit for each label. For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. 'multinomial' is unavailable when solver='liblinear'. 'auto' selects 'ovr' if the data is binary, or if solver='liblinear', and otherwise selects 'multinomial'.

In [24]:
```
from sklearn.linear_model import LogisticRegression
```

In [29]:
```
lg = LogisticRegression(multi_class="ovr")
lg.fit(x_train, y_train)
```

Out[29]:
```
▾ LogisticRegression
LogisticRegression(multi_class='ovr')
```

Check accuracy of the model

In [30]:
```
lg.score(x_test, y_test)*100
```

Out[30]:
```
96.66666666666667
```

Apply Logistic Regression through Multinomial Method

In [31]:
```
lg1 = LogisticRegression(multi_class="multinomial")
lg1.fit(x_train, y_train)
```

Out[31]:
```
▾ LogisticRegression
LogisticRegression(multi_class='multinomial')
```

Check accuracy of the model

In [32]:
```
lg1.score(x_test, y_test)*100
```

Out[32]:
```
100.0
```

Apply Logistic Regression through Direct (Default) Method

In [36]:
```
lg2 = LogisticRegression()
lg2.fit(x_train, y_train)
```

Out[36]:

```
▾ LogisticRegression
LogisticRegression()
```

Check accuracy of the model

In [35]:

```
lg2.score(x_test, y_test)*100
```

Out[35]:

```
100.0
```

In [ ]:

32. Confusion Matrix

- Confusion Matrix is to model the difference b/w the output data generated from testing of a model and the output of the original data. i.e. matrix b/w predicted output and original output.
- Model with 90%, 95% or even 100% can give wrong predictions
- The problem with wrong predictions can be traced through **confusion matrix**
- Confusion matrix gives better analysis of the built model

- A confusion matrix is a simple and useful tool for understanding the performance of a classification model, like one used in machine learning or statistics.
- It helps you evaluate how well your model is doing in categorizing things correctly.
- It is also know as the **error matrix / evaluation matrix**.
- The matrix consists of predictions result in a summarized from, which has number of correct predictions and incorrect predictions.

No description has been provided for this image

Interpretation of graphs

- TN = True Negative = Actual: 0, Predicted: 0 -> **True Negative**
- FN = False Negative = Actual: 1, Predicted: 0 -> **False Negative**
- FP = False Positive = Actual: 0, Predicted: 1 -> **False Positive**
- TP = True Positive = Actual: 1, Predicted: 1 -> **True Positive**

$$\text{Model Accuracy} = \frac{TN + TP}{TN + TP + FN + FP}$$

$$\text{Model Error} = \frac{FN + FP}{TN + TP + FN + FP}$$

False Negative: The model has predicted no (0), but the actual value was yes (1), it is also called as **Type-II error**

False Positive: The model has predicted yes (1), but the actual value was no (0), it is also called as **Type-I error**

- **False Negative is more dangerous**, depends on the situation

- if false negative is upto 5%, then reject this model

In [ ]:

32.1 Confusion Matrix (Sensitivity, Precision, Recall, F1-score)

Precision

Precision: It helps us to measure the ability to classify positive samples in the model.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- To increase the recall, False Positive value should be lower.

Recall

Recall: It helps us to measure how many positive samples were correctly classified by the ML model.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- To increase the recall, False Negative value should be lower.

F1-Score

- when we donot have information because of lack of knowledge in domain to whether improve Precsion or/and recall, then we will use **F1 Score**.

- It is the harmonic mean of precision and recall. It takes false positive and false negative into account.
- Therefore, it performs well on an imbalanced dataset.

$$\text{F1 Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

- Should increase the value of F1-Score

In Confusion matix,

- Precsion should should be high
- Recall should be high
- F-Score should be high

In [ ]:
```

```

33. Confusion Matrix (Practical) (Precision, Recall, F1-score)

In [3]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [4]:
```
dataset = pd.read_csv(r'Data/placement_3.csv')
dataset.head(3)
```

Out[4]:

	cgpa	score	placed
0	7.19	26	1
1	7.46	38	1
2	7.54	40	1

Split data into input and output data

In [5]:
```
x = dataset.iloc[:, :-1]
x
```

Out[5]:

	cgpa	score
0	7.19	26
1	7.46	38
2	7.54	40
3	6.42	8
4	7.23	17
...	...	...
995	8.87	44
996	9.12	65
997	4.89	34
998	8.62	46
999	4.90	10

1000 rows × 2 columns

In [6]:
```
y=dataset['placed']
y
```

Out[6]:
```
0      1
1      1
2      1
3      1
4      0
      ..
995    1
996    1
997    0
998    1
999    1
Name: placed, Length: 1000, dtype: int64
```

Split data into test and training data

In [7]:
```
from sklearn.model_selection import train_test_split
```

In [8]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

Build Model

In [9]:
```
from sklearn.linear_model import LogisticRegression
```

In [10]:
```
lg = LogisticRegression()
lg.fit(x_train, y_train)
```

Out[10]:
```
▾ LogisticRegression
LogisticRegression()
```

Checking model accuracy

In [11]:
```
lg.score(x_test, y_test)*100
```

Out[11]:
```
51.5
```

Confusion Matrix

In [12]:
```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_sco
```

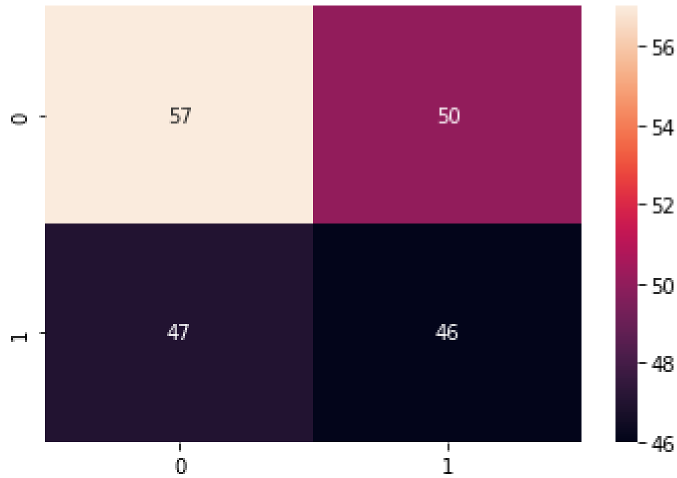
In [14]:
```
# Confusion matrix(y_true (actual), y_prediction)
cf = confusion_matrix(y_test,lg.predict(x_test))
cf
```

Out[14]:
```
array([[57, 50],
       [47, 46]], dtype=int64)
```

Graphically representing above results

In [15]:
```
sns.heatmap(cf, annot=True)
```

```
plt.show()
```



Interpretation of above graph:

- True Negative (TN)= 57
- True Positive (TP) = 46
- False Negative (FN) = 47
- False Positive (FP) = 50

Find Precision Score

In [17]:
```
# precision_score(y_true, y_pred)
precision_score(y_test, lg.predict(x_test))*100
```

Out[17]: 47.91666666666667

Find Recall Score

In [18]:
```
# recall_score(y_true, y_pred)
recall_score(y_test, lg.predict(x_test))*100
```

Out[18]: 49.46236559139785

Find F1-Score

In [20]:
```
# f1_score(y_true, y_pred)
f1_score(y_test, lg.predict(x_test))*100
```

Out[20]: 48.67724867724868

In [ ]:
```

```

34. Imbalanced Dataset

- Imbalanced dataset means that your data consists of multi categories and one category is repetitive in the data
- model is baised due to repitition of one category in the data
- Suppose your data consist of 500 rows:
- 400 rows for cat and 100 rows for dog,
- so the model will be biased towards cat

34.1 Techniques to handle imbalanced data

34.1.1 Random Under Sampling

- we will reduce the majority of the class so that it will have same number of as the minority
- for example out of 500 rows for cats and dogs, we will reduce (randomly) the rows to 100 for cats that is equal to 100 rows of dogs

34.1.2 Random Over Sampling

- We will increase the size of manority is inactive class to the size of majority calss i.e. active
- for example out of 500 rows for cats and dogs, we will repeat/duplicate (randomly) the rows for dogs to make it to 400 that is equal to 400 rows of cats

In [1]:
```
import pandas as pd
```

In [3]:
```
dataset = pd.read_csv(r'Data/Social_Network_Ads.csv')
dataset.head(3)
```

Out[3]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0

To check the data if it is imbalanced or not

In [4]:
```
dataset['Purchased'].value_counts()
```

Out[4]:
```
0    257
1    143
Name: Purchased, dtype: int64
```

So hence the data is **imbalanced** b/c both categories are not equal, so the data will be baised towards 0

In [12]:
```
x = dataset.iloc[:,:-1]
x
```

Out[12]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

400 rows × 2 columns

In [13]:
```
y = dataset['Purchased']
y
```

Out[13]:
```
0      0
1      0
2      0
3      0
4      0
      ..
395    1
396    1
397    1
398    0
399    1
Name: Purchased, Length: 400, dtype: int64
```

In [14]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

In [16]:
```
from sklearn.linear_model import LogisticRegression
```

In [17]:
```
lg = LogisticRegression()
lg.fit(x_train, y_train)
```

Out[17]:
```
▾ LogisticRegression
LogisticRegression()
```

In [22]:
```
lg.score(x_test, y_test)*100
```

Out[22]:
```
65.0
```

In [23]:
```
# y_true is 0
lg.predict([[19, 19000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[23]:
```
array([0], dtype=int64)
```

In [25]:
```
# y_true is 1
lg.predict([[45, 26000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[25]:
```
array([0], dtype=int64)
```

It has given wrong prediction, Reason: B/c the input data is **imbalanced**

In [26]:
```
# y_true is 1
lg.predict([[46, 28000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[26]:
```
array([0], dtype=int64)
```

Again, it has given wrong prediction, Reason: B/c the input data is **imbalanced**

In [ ]:
```

```

So hence we will balance our data by either:

- random under sampling, or
- random over sampling

34.2.1 Balacing the data by Random Under Sampling (Practical)

In [28]:
```
from imblearn.under_sampling import RandomUnderSampler
```

In [31]:
```
ru = RandomUnderSampler()
ru_x, ru_y = ru.fit_resample(x,y)
```

In [32]:
```
ru_x
```

Out[32]:

	Age	EstimatedSalary
224	35	60000
49	31	89000
153	36	50000
132	30	87000
359	42	54000
...	...	...
393	60	42000
395	46	41000
396	51	23000
397	50	20000
399	49	36000

286 rows × 2 columns

In [33]:
```
ru_y
```

Out[33]:
```
224    0
49     0
153    0
132    0
359    0
      ..
393    1
395    1
396    1
397    1
399    1
Name: Purchased, Length: 286, dtype: int64
```

Now after applying under sampling technique we will see, if 0 count is reduced to 143 or not

In [34]:
```
# Remember, out original data has following counts:
dataset['Purchased'].value_counts()
```

Out[34]:
```
0    257
1    143
Name: Purchased, dtype: int64
```

In [35]:
```
ru_y.value_counts()
```

Out[35]:
```
0    143
1    143
Name: Purchased, dtype: int64
```

So you can see 0 has reduced to 143 and now **our data is balanced!**

Now we have new data variables that are **ru_x, ru_y**

We will apply logitic regression on this new dataset that is balanced data, so we first split the data into train and test and then will apply logistic regression model

In [36]:
```
from sklearn.model_selection import train_test_split
```

In [37]:
```
x_train, x_test, y_train, y_test = train_test_split(ru_x, ru_y, test_size=0.20, ran
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
```

In [39]:
```
ru_lg = LogisticRegression()
ru_lg.fit(x_train, y_train)
```

Out[39]:
```
▾ LogisticRegression
LogisticRegression()
```

ru_lg.score(x_test, y_test)*100

To check if the model has improved or not we will supply same value as were predicted wrongly

In [41]:
```
# y_true is 1
ru_lg.predict([[45, 26000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[41]:
```
array([1], dtype=int64)
```

Hurrahh, now it has given accurate prediction, lets try second test..

In [43]:
```
# y_true is 1
ru_lg.predict([[46, 28000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[43]:
```
array([1], dtype=int64)
```

Oh yes, the second prediction is also accurate!!!

In [44]:
```
# y_true is 0
lg.predict([[19, 19000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[44]:
```
array([0], dtype=int64)
```

Great, accurate prediction again!!!

Conclusion is our model is not more biased

34.2.2 Balacing the data by Random Over Sampling (Practical)

In [46]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [47]:
```
ro = RandomOverSampler()
ro_x, ro_y = ro.fit_resample(x,y)
```

In [48]:
```
ro_x
```

Out[48]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
509	42	73000
510	55	39000
511	46	28000
512	37	93000
513	46	79000

514 rows × 2 columns

In [49]:
```
ro_y
```

Out[49]:
```
0      0
1      0
2      0
3      0
4      0
      ..
509    1
510    1
511    1
512    1
513    1
Name: Purchased, Length: 514, dtype: int64
```

In [61]:
```
# Remember, out original data has following counts:
dataset['Purchased'].value_counts()
```

Out[61]:
```
0    257
1    143
Name: Purchased, dtype: int64
```

So 1 should be increased to 257 as well as we have applied random over sampling method

In [60]:
```
ro_y.value_counts()
```

Out[60]:
```
0    257
1    257
Name: Purchased, dtype: int64
```

Now input is ro_x and output is ro_y, we will split the data into test and train and then apply logistic regression model

In [50]:
```
x_train, x_test, y_train, y_test = train_test_split(ro_x, ro_y, test_size=0.20, ran
```

In [52]:
```
ro_lg = LogisticRegression()
ro_lg.fit(x_train, y_train)
```

Out[52]:
```
▾ LogisticRegression
LogisticRegression()
```

In [54]:
```
ro_lg.score(x_test, y_test)*100
```

Out[54]:
```
88.3495145631068
```

Accuracy of the model is increased, impressive!!

In [56]:
```
# y_true is 0
lg.predict([[19, 19000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[56]:
```
array([0], dtype=int64)
```

In [57]:
```
# y_true is 1
ru_lg.predict([[46, 28000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[57]:
```
array([1], dtype=int64)
```

In [58]:
```
# y_true is 1
ru_lg.predict([[45, 26000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[58]:
```
array([1], dtype=int64)
```

Conclusion:

- All predictions are accurate by even Random Over Sampling
- in case of Random Over Sampling, the model accuracy is also increased from 65% (on imbalanced data) to 88% (on balanced data)

- in case of Random Under Sampling, the model accuracy is also decreased from 65% (on imbalanced data) to 58% (on balanced data)
- However, the model is predicting accurately after making the data balanced by both methods, i.e, Random over sampling, Random under sampling

In [ ]:
```

```

35. Naive Bayes

- These are classification algorithm which work on **conditional probabilty basis**
- Naive Bayes is a classification algorithm based on Bayes' theorem.
- which is a probability theory that describes the probability of an event, based on prior knowledge of conditions that might be related to the event
- **Naive**: It is called Naive b/c it assumes that the occurrence of a certain feature is independent of the occurrence of other features.
- **Bayes**: It is called Bayes b/c it depends on the principle of Bayes' Theorem.

Conditional Probability

$$\mathrm{P(E)} = \frac{\text{favourable outcome(s)}}{\text{total outcomes}}$$

0 <= P(E) >= 1

Example: A bag contains 3 red balls and 2 blue balls.

$$\mathrm{P(red\ balls)} = \frac{3}{3+2} = 0.60$$

Conditional Probability has 2 more types:

1. Indepedent Probability
2. Depdent Probability

1. **Independent Probability**

- Rolling a dice can have following events: {1,2,3,4,5,6}
- For single event the probability will be 1/6
- The preceding event and followng event both are not dependents on each other

2. **Dependent Probability**

- It is also called **Conditional Probability**
- In above balls example, the probability of P(red balls) = 3/5.
- But condition is that when you take the ball out, then do not put it back to the bag.
- so if we have taken one red ball from the bag, then there will be 2 red balls left and 4 total balls,
- In such circumstance, probability of blue ball is 2/4
- Hence the probabily of blue ball is depending upon the probabilty of red balls

- so we can express this as:

$$P(\text{Blue Balls}) = P(\frac{\text{Blue Balls}}{\text{Red Balls}}) = P(\frac{B}{R})$$

Also can be written as:

$$P(\text{R or B}) = P(R) * P(\frac{B}{R})$$

OR

$$P(\text{R or B}) = P(B/R) * P(R)$$

Bayes' Theorem:

$$P(\text{A n B}) = P(B/A) * P(A)$$

as

$$P(\text{A n B}) = P(\text{B n A})$$

so

$$P(\text{B n A}) = P(A/B) * P(B)$$

No description has been provided for this image

As

$$P(\text{A n B}) = P(\text{B n A})$$

So

$$P(B/A) * P(A) = P(A/B) * P(B)$$

To find P(A/B)

$$P(A/B) = \frac{P(B/A) * P(A)}{P(B)}$$

The above forumula is called **Bayes' Theorem**

- This formula states that when event B is occured, then what are chances of event A to come

Bayes' Theorem

- **Bayes' Theorem** is also known as Bayes' Rule or Bayes' law.
- which is used to determine the probability of a hypothesis with prior knowledge.

- It depends on the conditional probability.
- It is expressed as:

$$P(A/B) = \frac{P(B/A) * P(A)}{P(B)}$$

Where:
- **P(A/B) is Posterior Probability**: Probability of hypothesis A on the observed event B.
- **P(B/A) is Likelihood Probability**: Probability of hypothesis B when event A is occurring. Probability of the evidence given that the probability of a hypothesis is true.
- **P(A) is Prior Probability**: Probability of hypothesis before obeserving the evidence.
- **P(B) is Marginal Probability**: Probability of evidence.

No description has been provided for this image

No description has been provided for this image

Types of Naive Bayes Model

There are three types of Naive Bayes Model:

1. Gaussian
2. Multinomial
3. Bernoulli

1. **Gaussian Naive Bayes:**

- Assumes that continuous features follow a Gaussian (normal) distribution
- Suitable for features that are continuous and have a normal distribution

2. **Bernoulli Naive Bayes:**

- Assumes that features are binary (Boolean) variables
- Suitable for data that can be represented as binary features, such as document classification problems where each term is either present or absent

3. **Multinomial Naive Bayes:**

- Assumes that features follow a multinomial distribution
- Typically used for discrete data, such as text data, where each features represents the frequency of a term.

In [ ]:

36. Naive Bayes (Practical)

In [4]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions
```

In [6]:
```
dataset = pd.read_csv(r'Data/placement_3.csv')
dataset.head(3)
```

Out[6]:

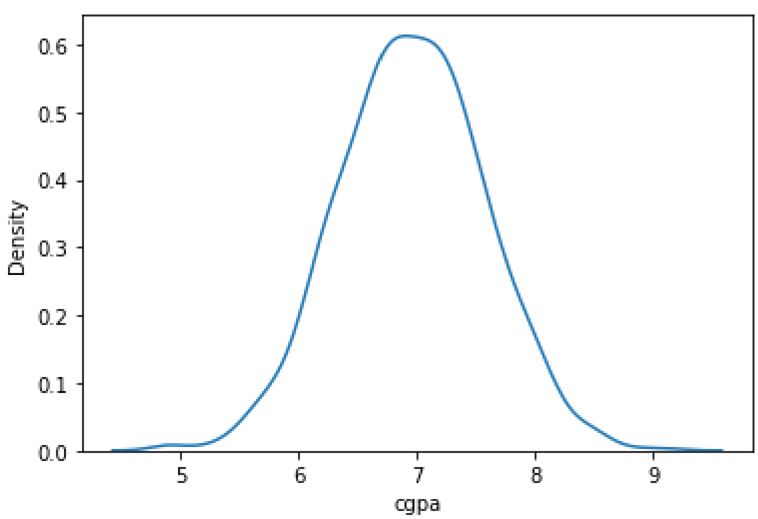
	cgpa	score	placed
0	7.19	26	1
1	7.46	38	1
2	7.54	40	1

In [7]:
```
dataset.isnull().sum()
```

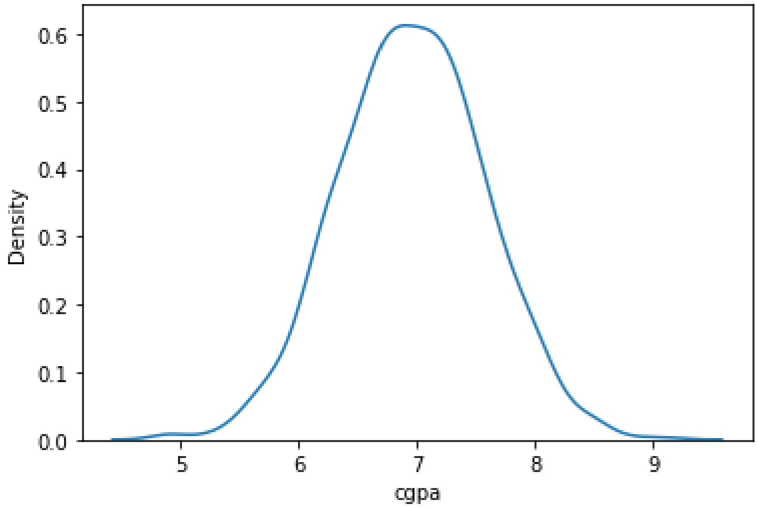
Out[7]:
```
cgpa      0
score     0
placed    0
dtype: int64
```

To check if the data is normally distributed or not, we will use disribution plot to check this

In [17]:
```
sns.kdeplot(data=dataset["cgpa"])
plt.show()
```

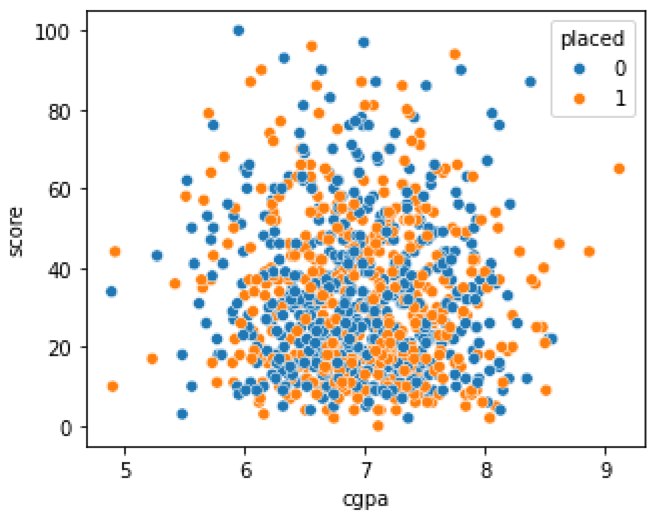


In [18]:
```
sns.kdeplot(data=dataset["cgpa"])
plt.show()
```



So will apply Gaussian Naive Bayes, b/c data is normally distributed.

In [10]:
```
plt.figure(figsize=(5,4))
sns.scatterplot(x="cgpa", y="score", data=dataset, hue="placed")
plt.show()
```



In [11]:
```
x = dataset.iloc[:,:-1]
x
```

Out[11]:

	cgpa	score
0	7.19	26
1	7.46	38
2	7.54	40
3	6.42	8
4	7.23	17
...	...	...
995	8.87	44
996	9.12	65
997	4.89	34
998	8.62	46
999	4.90	10

1000 rows × 2 columns

In [13]:
```
y = dataset["placed"]
y
```

Out[13]:
```
0      1
1      1
2      1
3      1
4      0
      ..
995    1
996    1
997    0
998    1
999    1
Name: placed, Length: 1000, dtype: int64
```

In [14]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

In [ ]:
```
```

In [19]:
```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
```

In [20]:
```
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

Out[20]:

```
▾ GaussianNB
GaussianNB()
```

In [23]:
```
gnb.score(x_test, y_test)*100, gnb.score(x_train, y_train)*100
```

Out[23]:
```
(53.0, 53.5)
```

In [ ]:

In [24]:
```
mnb = MultinomialNB()
mnb.fit(x_train, y_train)
```

Out[24]:

```
▾ MultinomialNB
MultinomialNB()
```

In [25]:
```
gnb.score(x_test, y_test)*100, gnb.score(x_train, y_train)*100
```

Out[25]:
```
(53.0, 53.5)
```

In [ ]:

In [ ]:
```
mnb = MultinomialNB()
mnb.fit(x_train, y_train)
```

In [ ]:

In [26]:
```
bnb = BernoulliNB()
bnb.fit(x_train, y_train)
```

Out[26]:

```
▾ BernoulliNB
BernoulliNB()
```

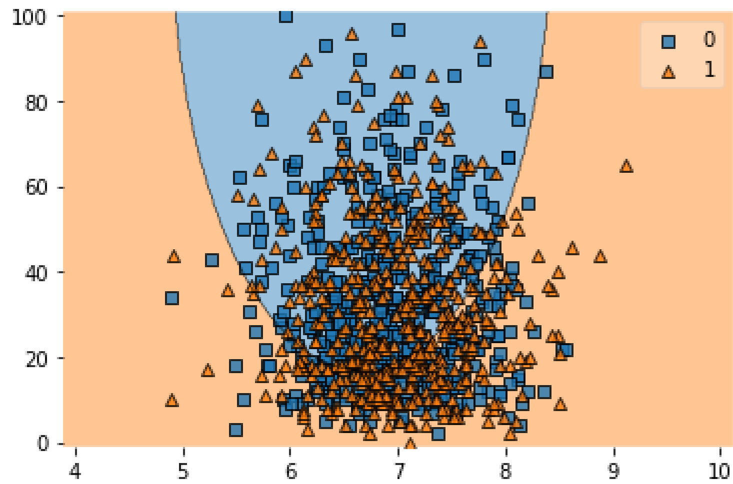
In [27]:
```
gnb.score(x_test, y_test)*100, gnb.score(x_train, y_train)*100
```

Out[27]:
```
(53.0, 53.5)
```

In [ ]:

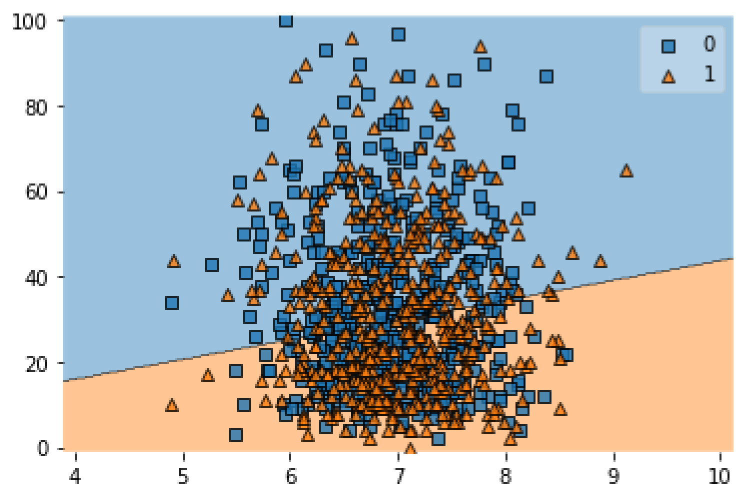
In [28]:
```
plot_decision_regions(x.to_numpy(), y.to_numpy(), clf=gnb)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but GaussianNB was fitte
d with feature names
  warnings.warn(
```



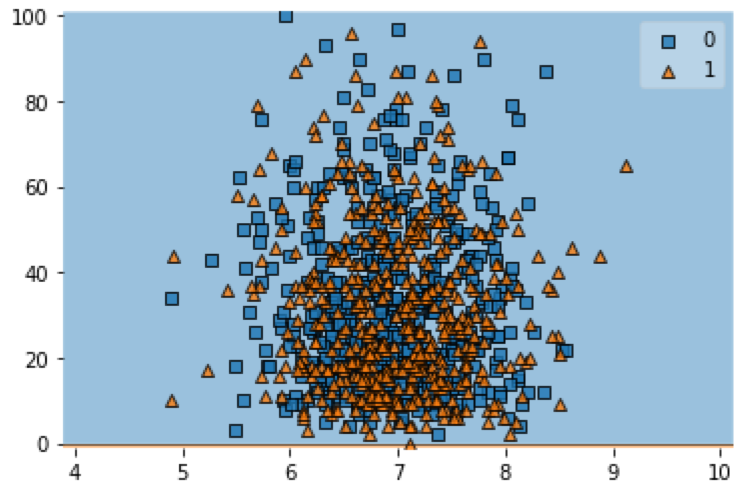
In [29]:
```
plot_decision_regions(x.to_numpy(), y.to_numpy(), clf=mnb)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but MultinomialNB was fi
tted with feature names
  warnings.warn(
```



In [30]:
```
plot_decision_regions(x.to_numpy(), y.to_numpy(), clf=bnb)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but BernoulliNB was fitt
ed with feature names
  warnings.warn(
```



In [ ]:

In [31]:
```
gnb.predict([[6.17, 5.17]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but GaussianNB was fitte
d with feature names
  warnings.warn(
```

Out[31]:
```
array([1], dtype=int64)
```

In [ ]: