37. Decision Tree (Regression)

Decision Tree

- Decision Tree is a **Supervised Learning** technique that can be used for both classification and regression problems, but mostly it is preferred for solving **classification problems**
- In order to build a tree, we can use the **CART algorithm**, which stands for Classification and Regression Tree algorithm
- it splits your data (Binary splitting)
- It works on non-linear splitting data
- It works as a conditional statement

Important Terminology related to Decision Tree

- **Root Node:** It represents the entire population or sample and this further gets divided into two or more homogenous sets
- **Splitting:** It is a process of dividing a node into two or more sub-nodes
- **Decision Node:** When a sub-node splits into furhter sub-nodes
- **Leaf/Terminal Node:** Nodes do not split further
- **Pruning:** When we remove sub-nodes of a decision node, this is an opposite process of splitting. Some time tree bcome too big, so the chances of over-fitting. So to avoid over-ftting, we use pruning
- **Branch/Sub-Tree:** A subsection of the entire tree
- **Parent and child node:** A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node

No description has been provided for this image

In below example, we can split the tree from:

1. company
2. Job
3. Degree

- However, we will consider the factors (which are explained below) to decide from which node, we should start splitting

No description has been provided for this image

Absolute Selection Measures

This measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

1. Information Gain
2. Entropy / Gini Index

Entropy: Entropy is a metric to measure the impurity in a given attribute. it specifies randomness in data.

$$\text{Entropy(s) = - P(yes) log2 P(yes) - P(no) Log2 P(no)}$$

Where:

- S = Total number of samples
- P(yes) = Probability of yes
- P(no) = Probability of no

Information Gain: Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class.

$$\text{Information Gain = Entropy(S) - [(Weighted Avg) * Entropy(each feature)]}$$

Gini Index: Gin index is a measure of impurity or purity used while creating a decision tree in the CART (Classification and Regression Tree) algorithm. An attribute with the low Gini index should be preferred as compared to the high Gini Index

$$Gini(D) = 1 - \sum_{i=1}^{n} p_i^2$$

Where:

- (D) is the dataset
- (p_i) is the proportion of class (i) in the dataset
- (n) is the number of classes

In [ ]:

```

```

- It means that your lowest impure data will become decision node
- We prefer less impure data for next splitting
- We will make root node (for example company or Degree) in below example which will have:
- **Low entropy**
- **High information gain**

No description has been provided for this image

- In the above example, the node that contains distinct number of either 1 or 0, it is **less impure**
- So we will choose company as a root/parent node becuase it has high number of 1 and low number of 0
- In other case i.e. Degree, number of 1 and 0 are equal, so we are not sure which value is true, so it is **more impure**
- In above example we have low entropy in case of splitting through company, and
- high entropy in case of splitting through Degree
- So we will choose company as parent/root node for further splitting

So we will calculate entropies of:

1. company
2. Job
3. Degree

- And then decide to start splitting from node which should have **Lowest entropy** (impurity).
- Low entropy means, **high information gain**.
- And vice versa

Algo for doing above task:

- **1st step:** We will calculate entropies of company, job, and degree. In this example, company has lowest entropy, so it will be first decision node, and we will split company into Amazon, Boat, Flipcard
- **2nd step:** We will again calculate entropies of job and degree, and choose the node for furhter splitting which have lowest entropy, which is degree in this case
- **3rd step:** Only one node left i.e., Job. This would be terminal/leaf node

In [ ]:

```

```

38. Decision Tree (Classification) (Practical)

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

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

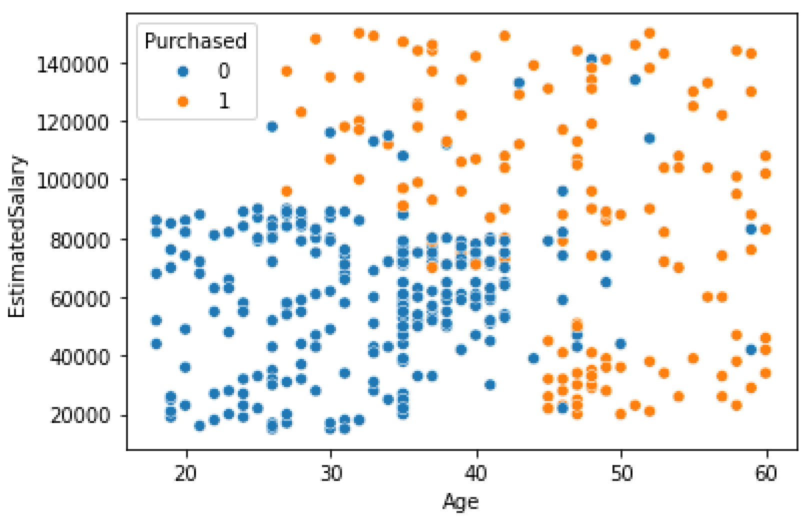
Out[3]:

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

To see how splitting is taking place through graph

- Decision tree is non-linear algorithm

In [33]:
```
sns.scatterplot(x="Age", y="EstimatedSalary", data=dataset, hue="Purchased")
plt.show()
```



- So this is non-linear graph

Step 1: Check for missing data

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

Step 2: Split the data into dependent and independent variables

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

Out[5]:

	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 [6]:
```
y = dataset['Purchased']
y
```

Out[6]:
```
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
```

Step 3: Do scaling of data

In [7]:
```
dataset.head(3)
```

Out[7]:

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

Scaling is needed b/c there is huge difference between values of Age and EstimatedSalary. So there is need to do scaling of data before model building

In [8]:
```
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
sc = StandardScaler()
sc.fit(x)
# Next step will transform (sc.transform(x)) the data and will convert into datafra
x = pd.DataFrame(sc.transform(x), columns=x.columns)
```

In [13]:
```
x
```

Out[13]:

	Age	EstimatedSalary
0	-1.781797	-1.490046
1	-0.253587	-1.460681
2	-1.113206	-0.785290
3	-1.017692	-0.374182
4	-1.781797	0.183751
...	...	...
395	0.797057	-0.844019
396	1.274623	-1.372587
397	1.179110	-1.460681
398	-0.158074	-1.078938
399	1.083596	-0.990844

400 rows × 2 columns

Now our has been scalled

Step 3: Split the data into train and test dataset

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
```

Step 4: Build Model through Decision Tree

- Decision tree can work for both classification through **DecisionTreeClassifier** or for regression through **DecisionTreeRegressor**
- As our output (dataset['Purchased']) consists of 0 and 1 form, so DecisionTreeClassifier will be used

In [16]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [18]:
```
# default: DecisionTreeClassifier(criterion='gini')
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
```

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

Step 5: Check Accuracy of Built Model

In [20]:
```
dt.score(x_test, y_test)*100
```

Out[20]:
```
83.75
```

Step 6: Perform Predictions on Built Model

In [22]:
```
dataset.head(3)
```

Out[22]:

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

In [23]:
```
dt.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 DecisionTreeClassifi
er was fitted with feature names
  warnings.warn(
```

Out[23]:
```
array([1], dtype=int64)
```

It gave wrong prediction

In [24]:
```
dt.predict([[35,20000]])
```

```
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 DecisionTreeClassifi
er was fitted with feature names
  warnings.warn(
```

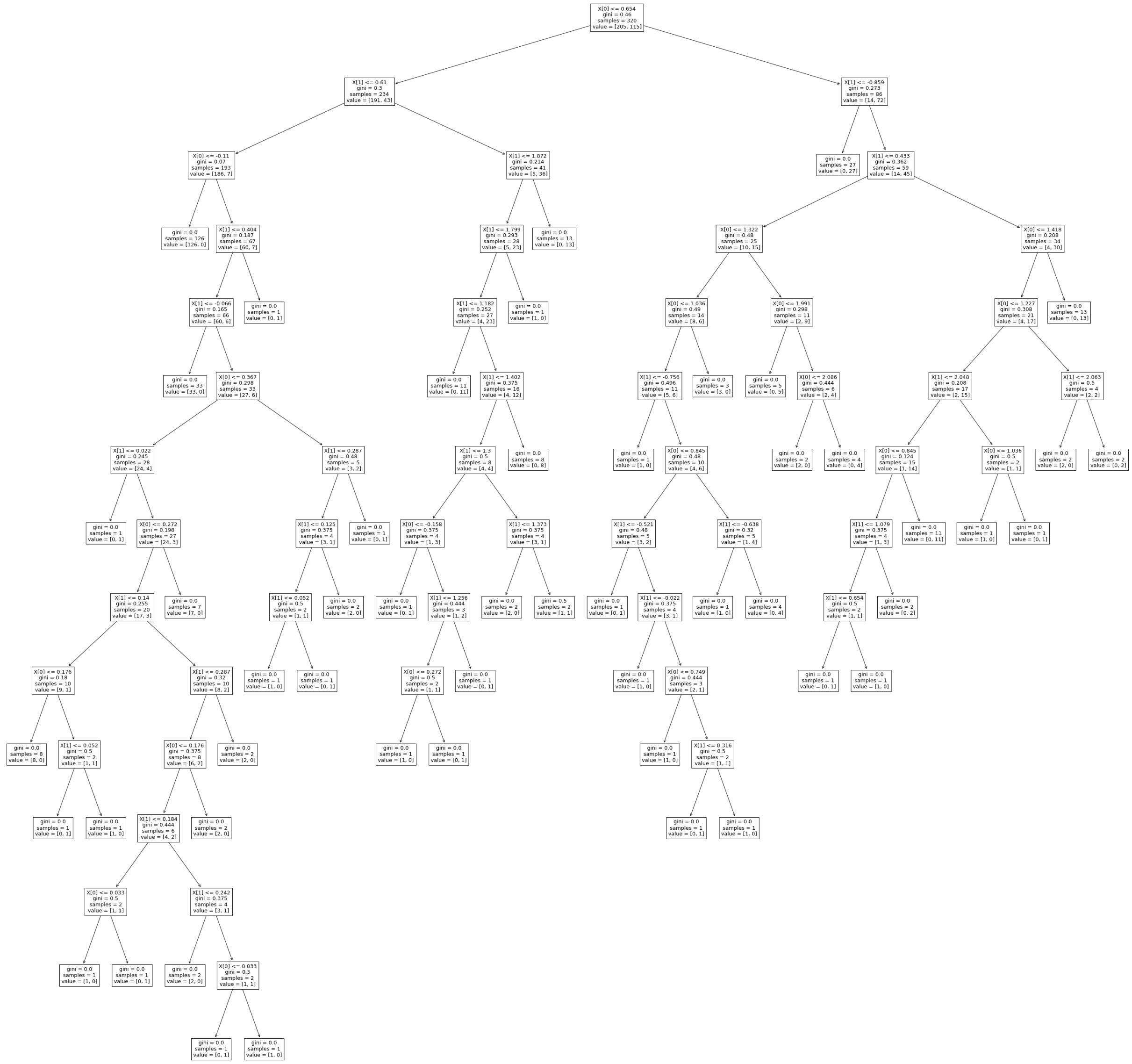
Out[24]:
```
array([1], dtype=int64)
```

again wrong prediction

Step 7: Analysis of Model through Graph

In [25]:
```
from sklearn.tree import plot_tree
```

In [29]:
```
# plot_tree(decision_tree)
plt.figure(figsize=(50,50))
plot_tree(dt)
plt.savefig(r'Generated_images/decision-tree-demo.jpg')
plt.show()
```



Step 8: Visualize Decision Tree Boundaries (How decision tree was split)

- We used CART algorithm, which will split the data in binary
-

Make Model through Entropy

In [30]:
```
# default: DecisionTreeClassifier(criterion='gini')
dt1 = DecisionTreeClassifier(criterion='gini')
dt1.fit(x_train, y_train)
```

Out[30]: ▾ DecisionTreeClassifier

DecisionTreeClassifier()

In [32]:
```
dt1.score(x_test, y_test)*100
```

Out[32]: 83.75

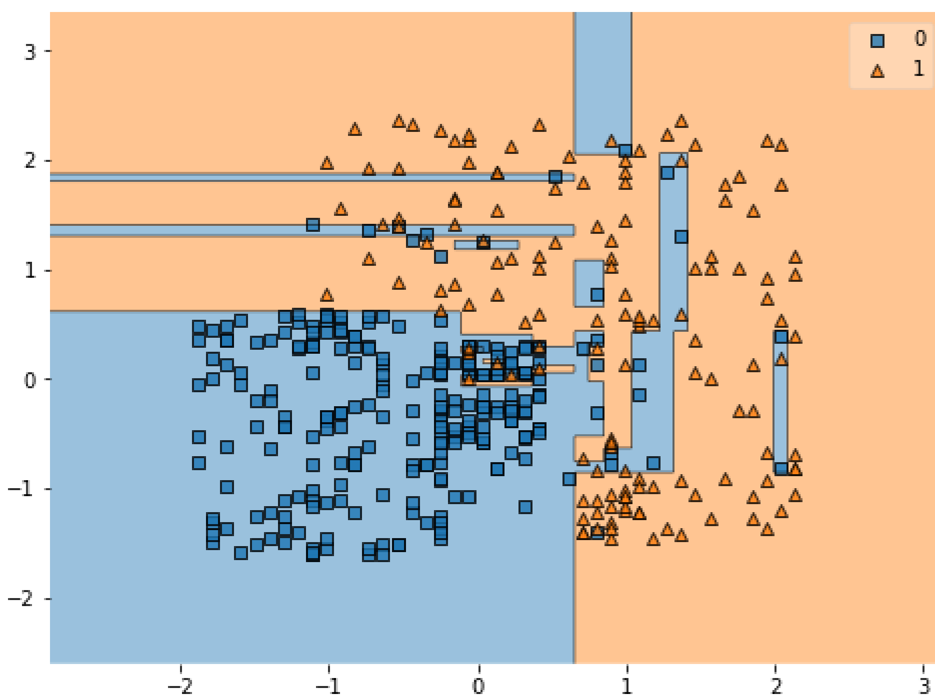
- No difference was found between model built by gini and entropy

To see Non-linear line splitting

In [35]:
```
from mlxtend.plotting import plot_decision_regions
```

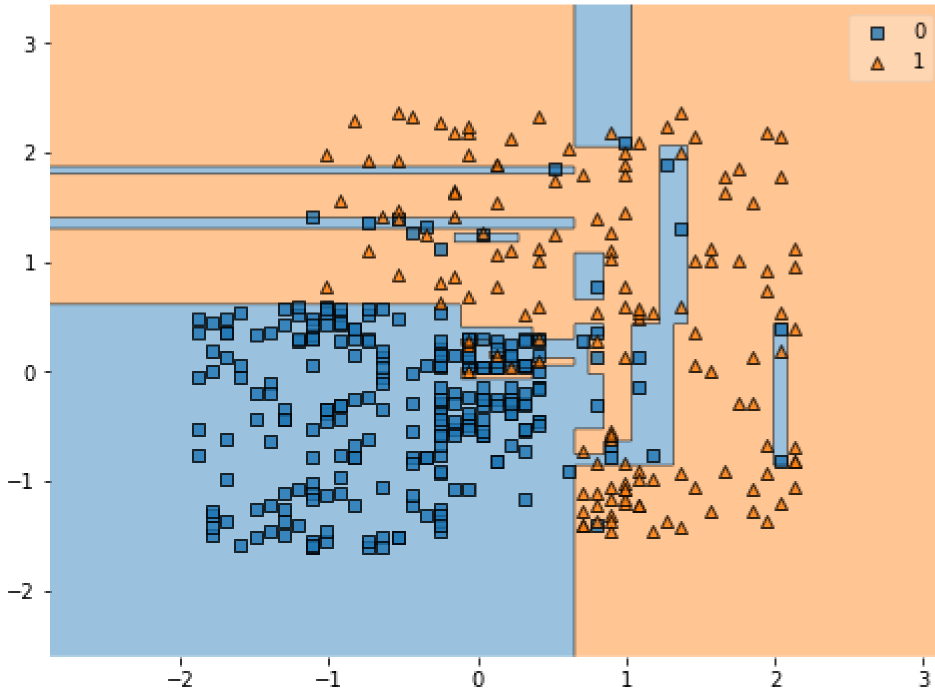
In [40]:
```
plt.figure(figsize=(8,6))
plot_decision_regions(x.to_numpy(),y.to_numpy(),clf=dt)
plt.show()
```

```
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 DecisionTreeClassifi
er was fitted with feature names
  warnings.warn(
```



In [41]:
```
plt.figure(figsize=(8,6))
plot_decision_regions(x.to_numpy(),y.to_numpy(),clf=dt1)
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 DecisionTreeClassifi
er was fitted with feature names
  warnings.warn(
```



In [ ]:

39. Pre and Post Pruning in a Decision Tree

- Pruning is performed to avoid your model from over-fitting
- **Pre-Pruning:** You perfrom pruning before making model
- **Post-Pruning:** You perfrom pruning after making model

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

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

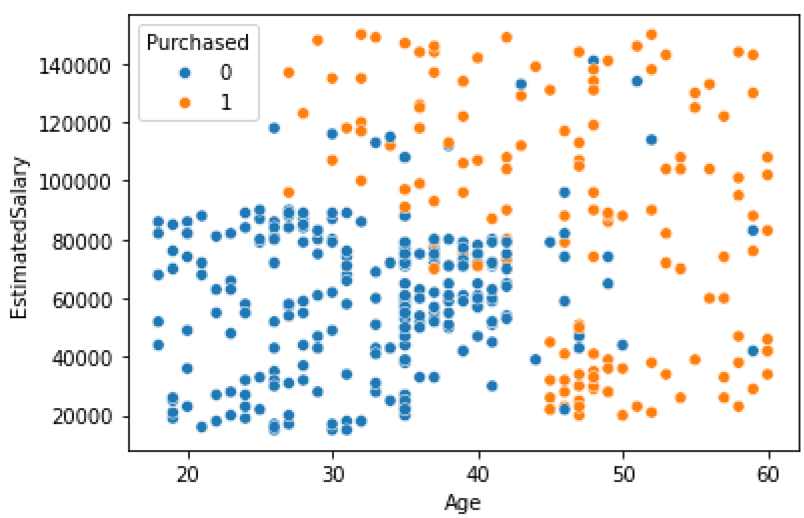
Out[3]:

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

To see how splitting is taking place through graph

- Decision tree is non-linear algorithm

In [4]:
```
sns.scatterplot(x="Age", y="EstimatedSalary", data=dataset, hue="Purchased")
plt.show()
```



- So this is non-linear graph

Step 1: Check for missing data

In [5]:
```
dataset.isnull().sum()
```

Out[5]:
```
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

Step 2: Split the data into dependent and independent variables

In [6]:
```
x = dataset.iloc[:,:-1]
x
```

Out[6]:

	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 [7]:
```
y = dataset['Purchased']
y
```

Out[7]:
```
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
```

Step 3: Do scaling of data

In [8]:
```
dataset.head(3)
```

Out[8]:

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

Scaling is needed b/c there is huge difference between values of Age and EstimatedSalary.

So there is need to do scaling of data before model building

In [9]:
```
from sklearn.preprocessing import StandardScaler
```

In [10]:
```
sc = StandardScaler()
sc.fit(x)
# Next step will transform (sc.transform(x)) the data and will convert into datafra
x = pd.DataFrame(sc.transform(x), columns=x.columns)
```

In [11]:
```
x
```

Out[11]:

	Age	EstimatedSalary
0	-1.781797	-1.490046
1	-0.253587	-1.460681
2	-1.113206	-0.785290
3	-1.017692	-0.374182
4	-1.781797	0.183751
...	...	...
395	0.797057	-0.844019
396	1.274623	-1.372587
397	1.179110	-1.460681
398	-0.158074	-1.078938
399	1.083596	-0.990844

400 rows × 2 columns

Now our has been scalled

Step 3: Split the data into train and test dataset

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_st
```

Step 4: Build Model through Decision Tree

- Decision tree can work for both classification through **DecisionTreeClassifier** or for regression through **DecisionTreeRegressor**
- As our output (dataset['Purchased']) consists of 0 and 1 form, so DecisionTreeClassifier will be used

In [14]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [15]:
```
# default: DecisionTreeClassifier(criterion='gini')
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
```

Out[15]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Step 5: Check Accuracy of Built Model

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

Out[16]:
```
83.75
```

Step 6: Perform Predictions on Built Model

In [17]:
```
dataset.head(3)
```

Out[17]:

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

In [18]:
```
dt.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 DecisionTreeClassifi
er was fitted with feature names
  warnings.warn(
```

Out[18]:
```
array([1], dtype=int64)
```

It gave wrong prediction

dt.predict([[35,20000]])

again wrong prediction

39.1 Perform Pruning

- First check whether your model is over-fit, the model will be over-fit, if accuracy of the training model is high and testing model accuracy is significantly low.

In [20]:
```
dt.score(x_test, y_test)*100
```

Out[20]:
```
83.75
```

In [28]:
```
dt.score(x_train, y_train)*100
```

Out[28]:
```
99.6875
```

- See hug difference b/w accuracies of training and testing data, so the model is over-fit

39.2 Perform Pre-Pruning

In [31]:
```
# default: DecisionTreeClassifier(criterion='gini')
dtpre = DecisionTreeClassifier(max_depth=5)
dtpre.fit(x_train, y_train)
```

Out[31]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

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

Out[32]:
```
90.0
```

In [34]:
```
dtpre.score(x_train, y_train)*100
```

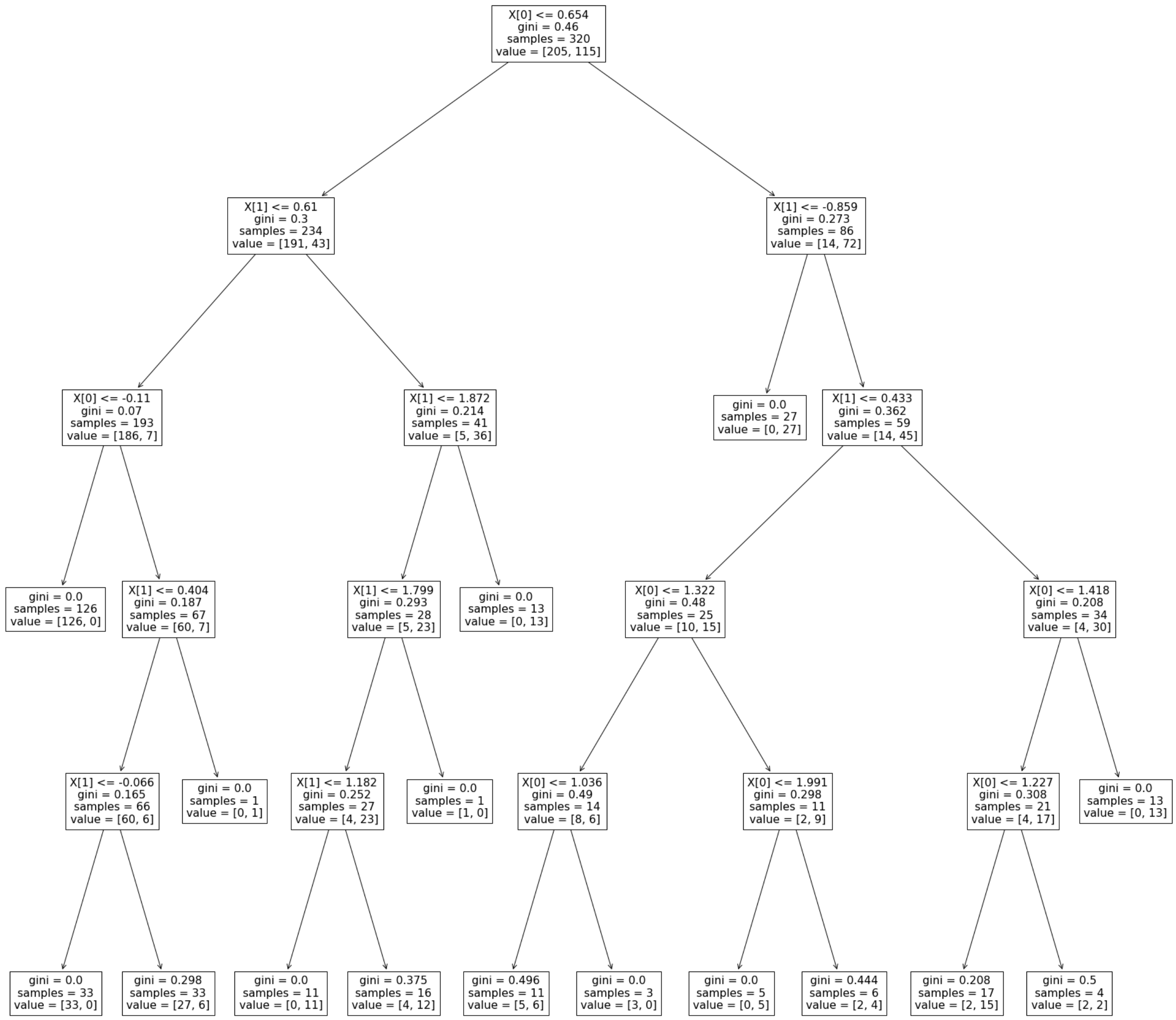
Out[34]:
```
93.4375
```

- See the difference between training data and testing data accuracy has been reduced, and hence over-fitting is reduced

39.2.1 Analysis of Model through Graph

In [37]:
```
from sklearn.tree import plot_tree
```

In [39]:
```
# plot_tree(decision_tree)
plt.figure(figsize=(30,30))
plot_tree(dtpre)
plt.savefig(r'Generated_images/decision-tree-demo-pre-prunning.jpg')
plt.show()
```

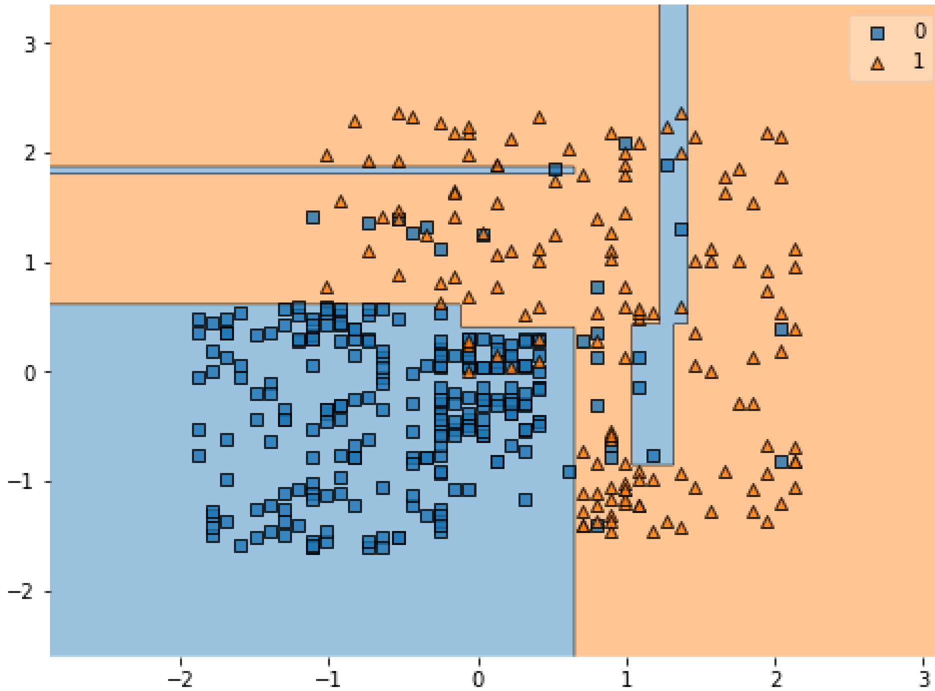


39.2.2 To see Non-linear line splitting

In [25]:
```
from mlxtend.plotting import plot_decision_regions
```

In [35]:
```
plt.figure(figsize=(8,6))
plot_decision_regions(x.to_numpy(),y.to_numpy(),clf=dtpre)
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 DecisionTreeClassifi
er was fitted with feature names
  warnings.warn(
```



In [ ]:

39.3 Perform Post-Pruning

In [42]:
```
for i in range(1, 19):
    dtpost = DecisionTreeClassifier(max_depth=i)
    dtpost.fit(x_train, y_train)
    print(dtpost.score(x_train, y_train)*100, dtpost.score(x_test, y_test)*100, i)
```

```
82.1875 90.0 1
91.875 91.25 2
91.875 91.25 3
93.125 91.25 4
93.4375 90.0 5
95.0 86.25 6
96.875 85.0 7
97.5 85.0 8
98.125 85.0 9
98.4375 85.0 10
99.0625 83.75 11
99.375 83.75 12
99.375 83.75 13
99.6875 83.75 14
99.6875 83.75 15
99.6875 83.75 16
99.6875 83.75 17
99.6875 83.75 18
```

- the difference in training and testing accuracy is negligible for model 2 and 3, **so it means that max_depth value should be 2 or 3**

- We can take max_depth till 5, as there is no major difference b/w accuracies of training and testing models

In [48]:
```
dtpost1 = DecisionTreeClassifier(max_depth=3)
dtpost1.fit(x_train, y_train)
```

Out[48]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

In [50]:
```
dtpost1.score(x_test, y_test)*100
```

Out[50]:
```
91.25
```

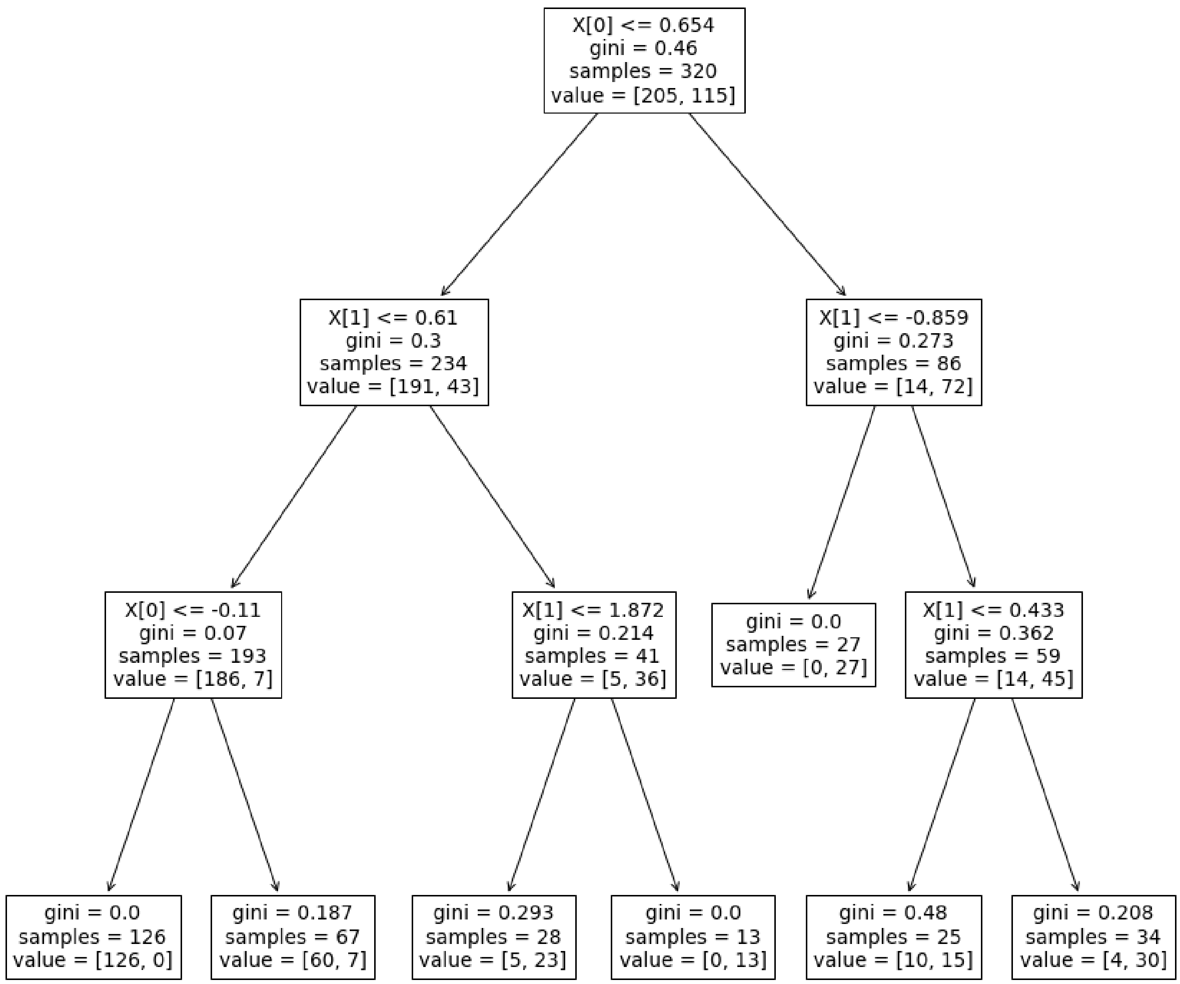
In [51]:
```
dtpost1.score(x_train, y_train)*100
```

Out[51]:
```
91.875
```

So Our model is no more over-fit

39.3.1 Analysis of Model through Graph

In [54]:
```
# plot_tree(decision_tree)
plt.figure(figsize=(15,15))
plot_tree(dtpost1)
plt.savefig(r'Generated_images/decision-tree-demo-post-prunning.jpg')
plt.show()
```

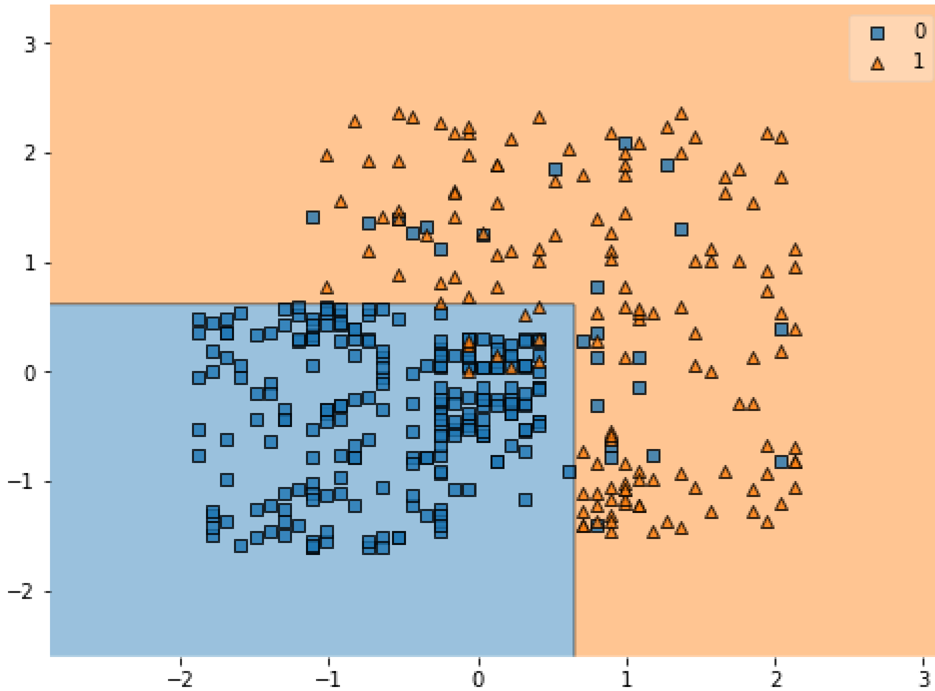


39.3.2 To see Non-linear line splitting

In [52]:

```
plt.figure(figsize=(8,6))
plot_decision_regions(x.to_numpy(),y.to_numpy(),clf=dtpost1)
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 DecisionTreeClassifi
er was fitted with feature names
  warnings.warn(
```



In [ ]:

40. Decision Tree (Regression)

- When data is non-linear and cannot separated through straight line.
- In left side of figure (A), data can be separated through simple linear regression
- but in right side of figure (B), data cannot be separated through simple linear regression, so we apply decision tree regression
-

No description has been provided for this image

In [ ]:

No description has been provided for this image

In [ ]:

No description has been provided for this image

In [ ]:

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

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

Out[3]:

	Age	Experience	Salary
0	53	21	274930.685866
1	39	19	217753.696272
2	32	19	166660.977435

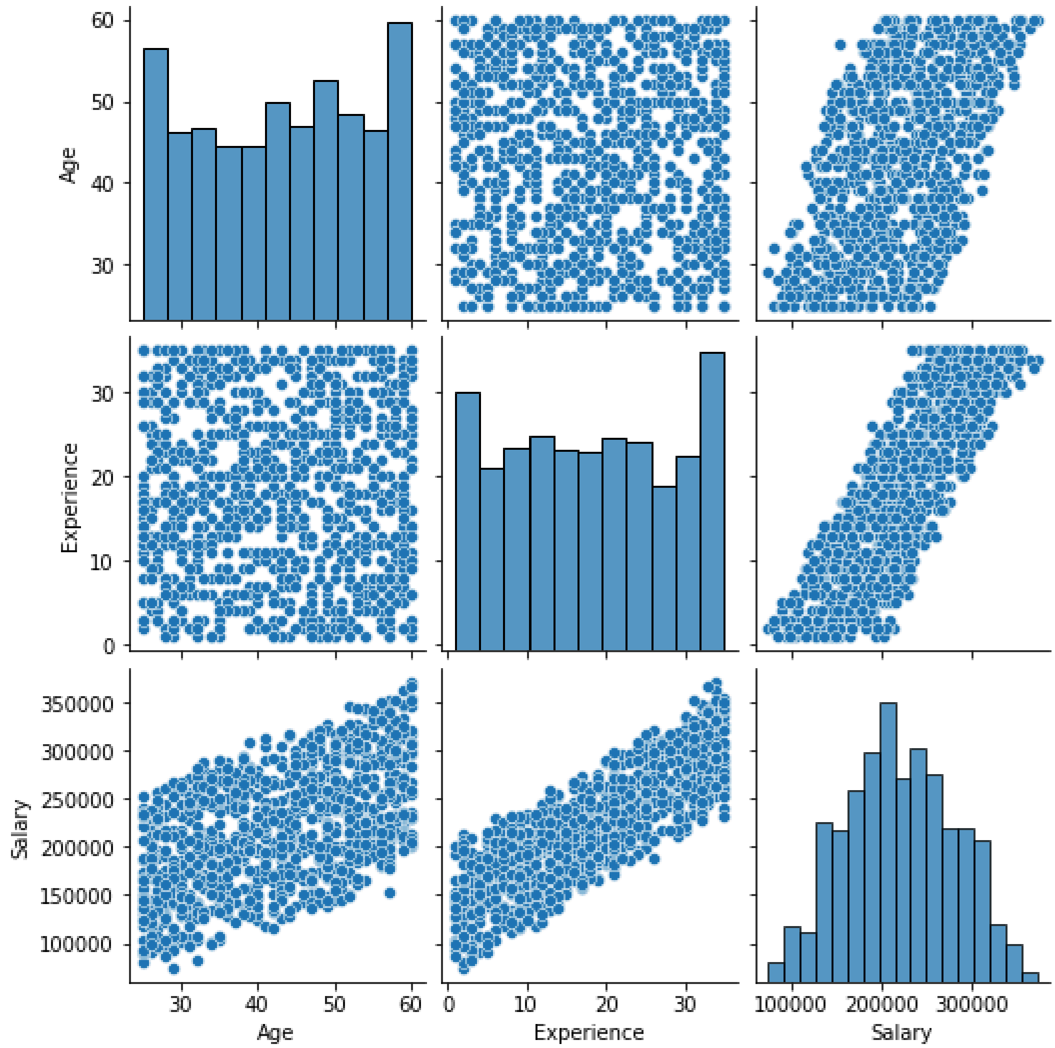
In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
Age           0
Experience    0
Salary        0
dtype: int64
```

Check the data if it is linear or non-linear through graph

In [5]:
```
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)
```



Split the data into depdent and independent variables

- The data is linear and we can apply simple linear regression
- but to demonstrate linear regression through decision tree, we will apply decision tree regression

In [7]:
```
x = dataset.iloc[:,:-1]
x
```

Out[7]:

	Age	Experience
0	53	21
1	39	19
2	32	19
3	45	29
4	43	18
...	...	...
995	31	32
996	34	1
997	31	23
998	57	8
999	47	13

1000 rows × 2 columns

In [8]:
```
y = dataset['Salary']
y
```

Out[8]:
```
0      274930.685866
1      217753.696272
2      166660.977435
3      281857.674921
4      221357.621324
           ...
995    246721.167856
996     98140.456867
997    207088.257665
998    231458.172881
999    213710.389200
Name: Salary, Length: 1000, dtype: float64
```

Split the data into train and test dataset

In [9]:
```
from sklearn.model_selection import train_test_split
```

In [10]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

Build model through decision tree regressor

In [12]:
```
from sklearn.tree import DecisionTreeRegressor, plot_tree
```

In [13]:
```
dt = DecisionTreeRegressor()
dt.fit(x_train, y_train)
```

Out[13]:

▾ DecisionTreeRegressor

DecisionTreeRegressor()

Check accuracy of built model

In [15]:
```
dt.score(x_test, y_test)*100
```

Out[15]: 94.73975868182897

Check if model is over-fit

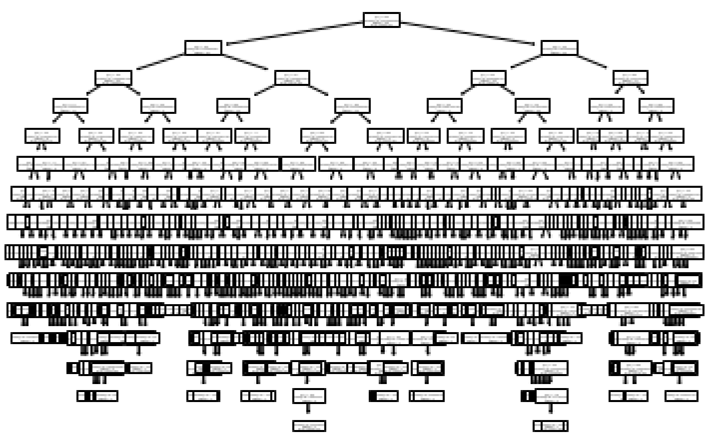
In [17]:
```
dt.score(x_train, y_train)*100
```

Out[17]: 99.20845616821404

- It is slightly over-fit

Plot tree

In [16]:
```
plot_tree(dt)
plt.show()
```



In [ ]:

41. K_Nearest Neighbours (Classification)

- Used for non-linear data
- K-NN can be used for regression as well as for classification but mostly it is used for the classification problems
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data
- It is also called a **lazy learner algorithm**

41.1 How does K-NN Work?

No description has been provided for this image

- It calculate distance from all neighboring data point and find the nearest data point
- It will calculate the nature of that data point
- The distance is calculated based on nature of projects based on two methods:

1. Manhattan distance (L1)
2. Euclidean distance (L2)

- K-value varies according to dataset. It is a hypo-parameter which needs tuning

No description has been provided for this image

In [ ]:

42. K-Nearest Neighbour (Classification) (Practical)

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

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

Out[3]:

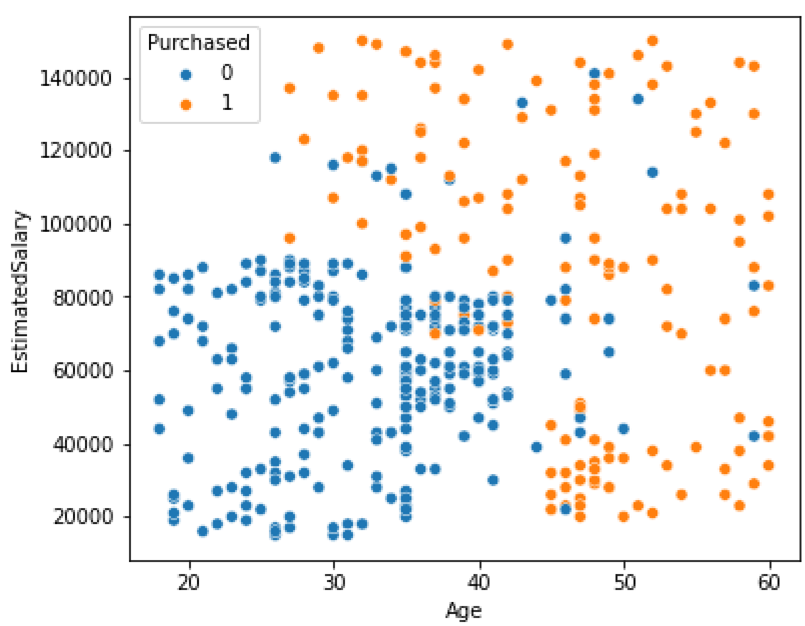
	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

Step 1: Check how the data is distributed through graph

In [7]:
```
plt.figure(figsize=(6,5))
sns.scatterplot(x="Age", y="EstimatedSalary", data=dataset, hue="Purchased")
plt.show()
```



Step 2: Split the data into dependent and independent variables

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

Step 3: Perform scaling of the data

In [8]:
```
from sklearn.preprocessing import StandardScaler
```

In [11]:
```
sc = StandardScaler()
sc.fit(x)
# after transforming the data through 'sc.transform(x)' convert it to dataframe
x = pd.DataFrame(sc.transform(x), columns=x.columns)
x
```

Out[11]:

	Age	EstimatedSalary
0	-1.781797	-1.490046
1	-0.253587	-1.460681
2	-1.113206	-0.785290
3	-1.017692	-0.374182
4	-1.781797	0.183751
...	...	...
395	0.797057	-0.844019
396	1.274623	-1.372587
397	1.179110	-1.460681
398	-0.158074	-1.078938
399	1.083596	-0.990844

400 rows × 2 columns

Step 4: Split the data into train 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_st
```

Step 5: Build the Model through K-NN

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
```

- We are using 'KNeighborsClassifier' b/c the output in this example is in classification nature (0 and 1)

In [16]:
```
# default: n_neighbors=5
''' p : int, default=2
    Power parameter for the Minkowski metric. When p = 1, this is
    equivalent to using manhattan_distance (l1), and euclidean_distance
    (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.'''

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train, y_train)
```

Out[16]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Step 6: Check the accuracy of built K-NN model

In [17]:
```
knn.score(x_test, y_test)*100
```

Out[17]:
```
92.5
```

Change the value of neighbor to adjust the accuracy of model

In [19]:
```
knn1 = KNeighborsClassifier(n_neighbors=7)
knn1.fit(x_train, y_train)
```

Out[19]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

In [20]:
```
knn1.score(x_test, y_test)*100
```

Out[20]:
```
93.75
```

Step 7: To check whether the built K-NN model is over-fit

In [21]:
```
knn1.score(x_test, y_test)*100
```

Out[21]:
```
93.75
```

In [22]:
```
knn1.score(x_train, y_train)*100
```

Out[22]:
```
91.875
```

The built KNN Model is not well trained as there is difference b/w training and testing score difference. So keep on changing value of n_neighor to train the model well and to avoid over-fitting

Step 8: Apply loop to find the optimum n-neighbor value for avoiding over-fitting

- To find right value of n-neighbor, we will run loop to see at which value there is no major difference b/w training and testing data accuracies.

In [28]:
```
for i in range(1,30):
    knn2 = KNeighborsClassifier(n_neighbors=i)
    knn2.fit(x_train, y_train)
    #print("Testing Data Score:", knn2.score(x_test, y_test)*100, "Training Data Sc
    print(i, knn2.score(x_test, y_test)*100, knn2.score(x_train, y_train)*100)
```

```
1 86.25 99.6875
2 86.25 91.5625
3 91.25 92.5
4 92.5 91.875
5 92.5 90.9375
6 90.0 90.9375
7 93.75 91.875
8 92.5 90.625
9 93.75 91.25
10 92.5 90.625
11 92.5 90.9375
12 92.5 91.25
13 92.5 91.5625
14 92.5 90.625
15 92.5 90.625
16 92.5 90.0
17 92.5 90.625
18 92.5 90.3125
19 92.5 90.9375
20 93.75 90.0
21 92.5 90.3125
22 93.75 90.0
23 93.75 90.3125
24 93.75 89.375
25 93.75 90.0
26 93.75 89.375
27 92.5 89.375
28 93.75 88.75
29 93.75 88.75
```

- **Over-fitting:**: When accuracy of traning data set is greater than testing data set
- **Under-fitting:**: When accuracy of traning data set is less than testing data set
- 1 86.25 99.6875 = overfitting
- 2 86.25 91.5625 = overfitting
- 3 91.25 92.5 = almost good model, as no major difference b/w accuracies of training and testing data set
- 4 92.5 91.875 = underfitting

- 5 92.5 90.9375 = underfitting
- **6 90.0 90.9375 = Best fit**

In [32]:
```
# So will choose 6
knn3 = KNeighborsClassifier(n_neighbors=6)
knn3.fit(x_train, y_train)
```

Out[32]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=6)
```

In [33]:
```
knn3.score(x_test, y_test)*100
```

Out[33]:
```
90.0
```

In [34]:
```
knn3.score(x_train, y_train)*100
```

Out[34]:
```
90.9375
```

Step 9: Peform prediction on tunned model i.e., knn3

It is very important to remember that give scalling data for prediction instead of original data, as the model is trained on scalling data

In [40]:
```
# This is original data
dataset.head(3)
```

Out[40]:

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

In [41]:
```
# This is scalled data
x.head(3)
```

Out[41]:

	Age	EstimatedSalary
0	-1.781797	-1.490046
1	-0.253587	-1.460681
2	-1.113206	-0.785290

In [42]:
```
# So we will give scalled data as input to the model for prediction
knn3.predict([[-1.781797,-1.490046]])
```

```
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 KNeighborsClassifier
was fitted with feature names
  warnings.warn(
```

Out[42]:
```
array([0], dtype=int64)
```

Accurate Prediction!!!

In [43]:
```
dataset.tail(3)
```

Out[43]:

	Age	EstimatedSalary	Purchased
397	50	20000	1
398	36	33000	0
399	49	36000	1

In [44]:
```
x.tail(3)
```

Out[44]:

	Age	EstimatedSalary
397	1.179110	-1.460681
398	-0.158074	-1.078938
399	1.083596	-0.990844

In [45]:
```
knn3.predict([[1.083596,-0.990844]])
```

```
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 KNeighborsClassifier
was fitted with feature names
  warnings.warn(
```

Out[45]:
```
array([1], dtype=int64)
```

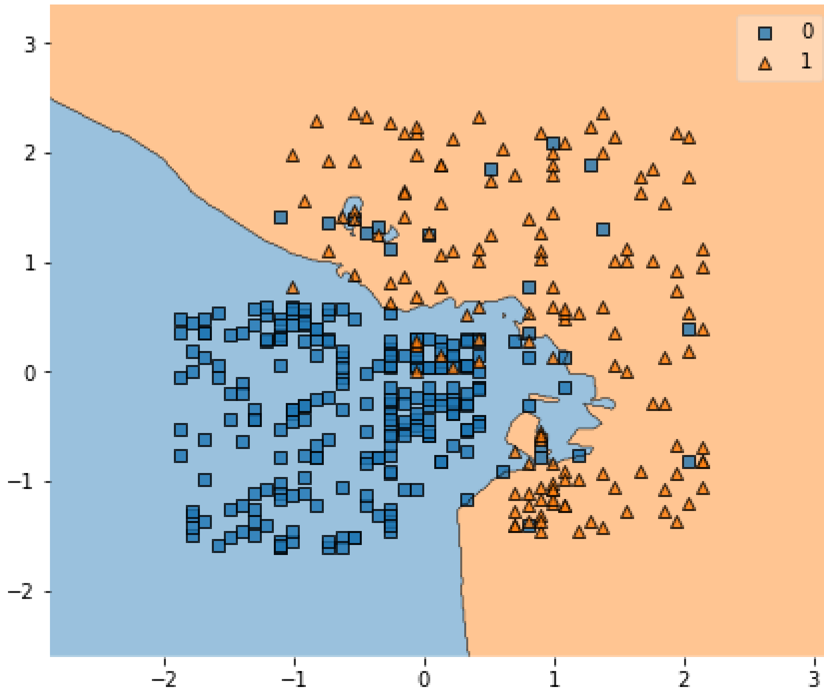
Accurate Prediction!!!

Step 10: Check Decision Boundaries through graph

In [46]:
```
from mlxtend.plotting import plot_decision_regions
```

In [48]:
```
plt.figure(figsize=(7,6))
plot_decision_regions(x.to_numpy(), y.to_numpy(), clf=knn3)
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 KNeighborsClassifier
was fitted with feature names
  warnings.warn(
```



In [ ]:

43. K-Nearest Neighbors (Regression)

No description has been provided for this image

In [ ]:

```

```

No description has been provided for this image

In [6]:

```
import pandas as pd
```

In [7]:

```
dataset = pd.read_csv(r'Data/salary_data_2.csv')
dataset.head(3)
```

Out[7]:

	Age	Salary	Experience
0	53	274930.6859	21
1	39	217753.6963	19
2	32	166660.9774	19

In [8]:

```
dataset.isnull().sum()
```

Out[8]:

```
Age           0
Salary        0
Experience    0
dtype: int64
```

Step 1: Split the data into dependent and independent variables

In [9]:

```
x = dataset.drop(columns='Salary')
y = dataset['Salary']
```

In [10]:

```
x
```

Out[10]:

	Age	Experience
0	53	21
1	39	19
2	32	19
3	45	29
4	43	18
...	...	...
995	31	32
996	34	1
997	31	23
998	57	8
999	47	13

1000 rows × 2 columns

In [11]:
```
y
```

Out[11]:
```
0      274930.68590
1      217753.69630
2      166660.97740
3      281857.67490
4      221357.62130
           ...
995    246721.16790
996     98140.45687
997    207088.25770
998    231458.17290
999    213710.38920
Name: Salary, Length: 1000, dtype: float64
```

Step 2: Split the data into train and test variables

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [16]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

Step 3: Apply KNN Regression Model

In [17]:
```
from sklearn.neighbors import KNeighborsRegressor
```

In [22]:
```
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train, y_train)
```

Out[22]:

```
▾ KNeighborsRegressor
KNeighborsRegressor()
```

Step 4: Check Accuracy of Model

In [23]:

```python
knn.score(x_test, y_test)*100
```

Out[23]:

```
96.56477286387577
```

In [ ]:

44. Support Vector Machines (SVM) - Classification

-- SVM is one of the most popular **Supervised Learning** algorithm, which is used fro classification as well as Regression problems

- With the help of SVM, you can handle both linear and non-linear data
- Its working is similar to logistic regression algo
- **Algo of SVM**:

1. It finds two points (support vectors) in the data (shown in red color in below figure)
2. It passes marginal plan/line (hyperplane) from these support vectors (dotted red lines)
3. It measures the distance b/w these two lines
4. Take avearge of the distance (divided by 2)
5. This average is denoted by a separable line (Maximum margin) (solid red line)
6. Prediction is done through this line and decided the new data would go to which category (Splitting of data)
7. The distance (d) b/w 2 vectors should be maximum

No description has been provided for this image

- **Hard Margin:** The algorithm aims to find a hyperplane that perfectly separates the data into two classes w/o any misclassifications.
- **Soft Margin:** The algorithm allows for some misclassifications to find a hyperplane that generalizes better to unseen data and is more robust to outliers

No description has been provided for this image

No description has been provided for this image

In [ ]:

Types of SVM: There are two different types of SVMs, each used for different things:

1. **Simple SVM:** Typically used for linear regression and classifications problems.
2. **Kernel SVM:** Has more flexibility for non-linear data b/c you can add more features to fit a hyperplane instead of a two-dimensional space.

- Kernel SVM is used when our data is not linearly separable.
- It modifies our data

Kernel Functions:

- Kernel functions play a crucial role in transforming input into a higher-dimensional space.
- The primary purpose of kernel functions is to allow SVMs to handle non-linearly separable data by implicitly mapping the input data into a higher-dimensional feature space where linear separation may be more feasible.
- This transformation is done w/o explicitly calculating the coordinate points in that higher-dimensional space.

Kernel Functions in SVM

1. **Linear Kernel**:

$$K(x_i, x_j) = x_i^T x_j$$

2. **Polynomial Kernel**:

$$K(x_i, x_j) = (\gamma \cdot x_i^T x_j + r)^d$$

3. **Gaussian Radial Basis Function (RBF) Kernel**:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$$

4. **Sigmoid Kernel**:

$$K(x_i, x_j) = \tanh(\gamma \cdot x_i^T x_j + r)$$

Description of Symbols

- (x_i, x_j): Input feature vectors.
- (x_i^T x_j): Dot product between vectors (x_i) and (x_j).
- (\gamma): Scaling factor (often related to (\sigma) in the Gaussian RBF kernel as (\gamma = \frac{1}{2\sigma^2})).
- (r): Constant term (bias).
- (d): Degree of the polynomial in the Polynomial Kernel.
- (|x_i - x_j|): Euclidean distance between (x_i) and (x_j).
- (\exp): Exponential function.
- (\tanh): Hyperbolic tangent function.

```
In [ ]:
```

45. Support Vector Machines (SVM) - Classification (Practical)

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions
```

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

Out[4]:

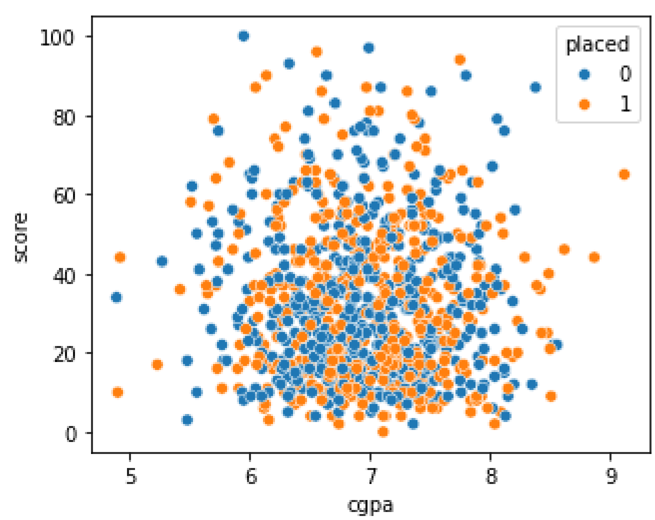
	cgpa	score	placed
0	7.19	26	1
1	7.46	38	1
2	7.54	40	1

In [5]:
```
dataset.isnull().sum()
```

Out[5]:
```
cgpa      0
score     0
placed    0
dtype: int64
```

Step 1: To check if the data is linearly/non-linearly separable data

In [11]:
```
plt.figure(figsize=(5,4))
sns.scatterplot(x='cgpa', y='score', data=dataset, hue='placed')
plt.show()
```



Step 2: Separate dependent and independent variables

In [13]:
```
x = dataset.iloc[:,:-1]
y = dataset['placed']
```

Step 3: Split data into train and test data

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

In [51]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

Step 4: Train data through SVM Model

SVC: Support Vector Classifier. As our data output consist of 0 and 1, so we will apply classifier, SVC

In [52]:
```
from sklearn.svm import SVC
```

In [66]:
```
'''kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable,
    Specifies the kernel type to be used in the algorithm.
    If none is given, 'rbf' will be used. If a callable is given it is
    used to pre-compute the kernel matrix from data matrices; that matrix
    should be an array of shape ``(n_samples, n_samples)``.'''
# precomputed used for data that is one-encoded (in the form of 0 and 1)

sv = SVC(kernel='linear')
sv.fit(x_train, y_train)
```

Out[66]:
```
▾ SVC
SVC(kernel='linear')
```

Step 5: Check accuracy of SVM Model

In [63]:
```
sv.score(x_test, y_test)*100
```

Out[63]:
```
52.0
```

Step 6: Check whether SVM Model is over/under-fit

In [64]:
```
sv.score(x_train, y_train)*100
```

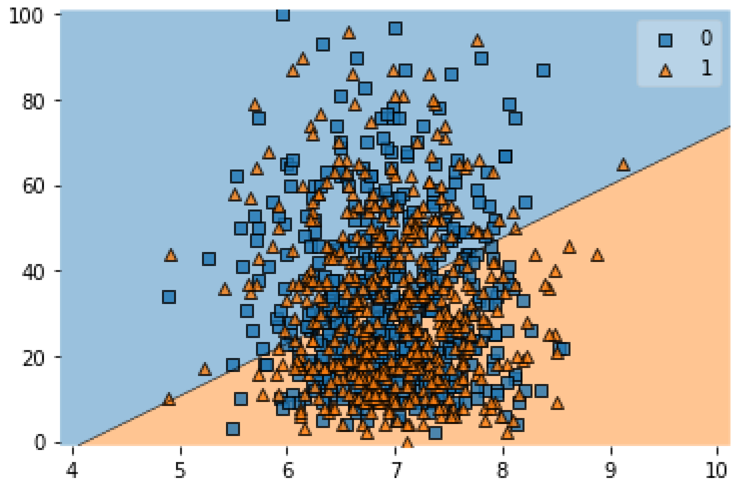
Out[64]:
```
49.875
```

- Model is underfit

Step 7: Check boundaries of separation

In [65]:
```
plot_decision_regions(x.to_numpy(), y.to_numpy(), clf=sv)
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 SVC was fitted with
feature names
  warnings.warn(
```



- Lot of misclassifications

In [ ]:

46. Support Vector Machines (SVM) - Regression

Support Vector Regression (SVR) is a regression technique that uses SVM for modellling and predicting continuous outcomes.

- Opposite of SVC
- Here distance between decision b/w support vectors should be minimum

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

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

Out[2]:

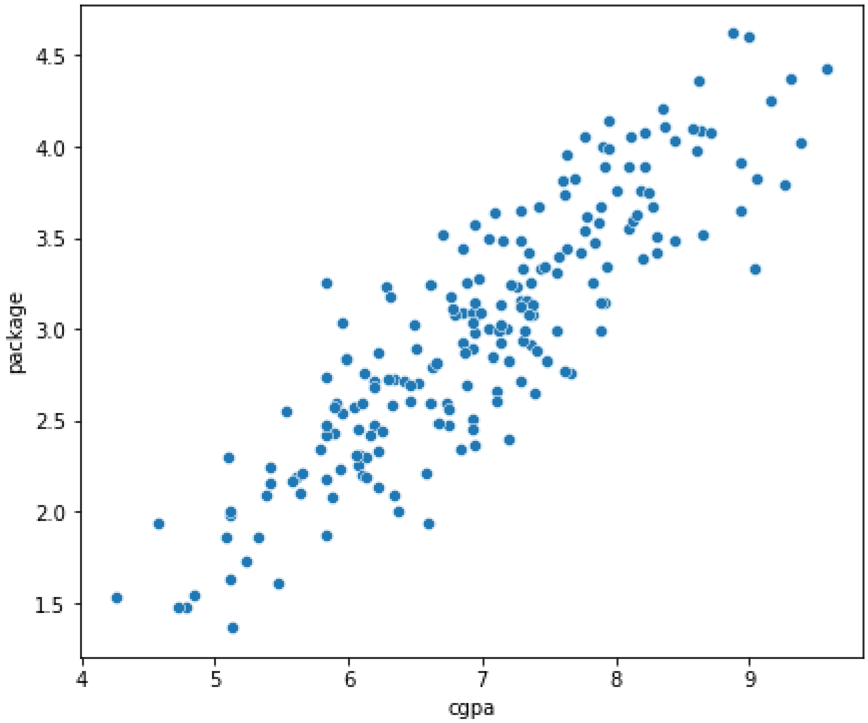
	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25

In [3]:
```
dataset.isnull().sum()
```

Out[3]:
```
cgpa       0
package    0
dtype: int64
```

Step 1: To check if the data is linearly/non-linearly separable data

In [27]:
```
plt.figure(figsize=(7,6))
sns.scatterplot(x='cgpa', y='package', data=dataset)
plt.show()
```



This graph represents that our data is linearly separable

Step 2: Separate dependent and independent variables

In [16]:
```
x = dataset[['cgpa']]
y =dataset['package']
```

Step 3: Split data into train and test data

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

Step 4: Train data through SVR Model

In [19]:
```
from sklearn.svm import SVR
```

In [22]:
```
'''kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable,
     Specifies the kernel type to be used in the algorithm.
     If none is given, 'rbf' will be used. If a callable is given it is
     used to precompute the kernel matrix.'''
sv = SVR(kernel='linear')
sv.fit(x_train, y_train)
```

Out[22]:

▾ SVR

```
SVR(kernel='linear')
```

Step 5: Check accuracy of SVM Model

In [23]:
```
sv.score(x_test, y_test)*100
```

Out[23]: 77.06668029575103

Step 6: Check whether SVM Model is over/under-fit

In [24]:
```
sv.score(x_test, y_test)*100
```

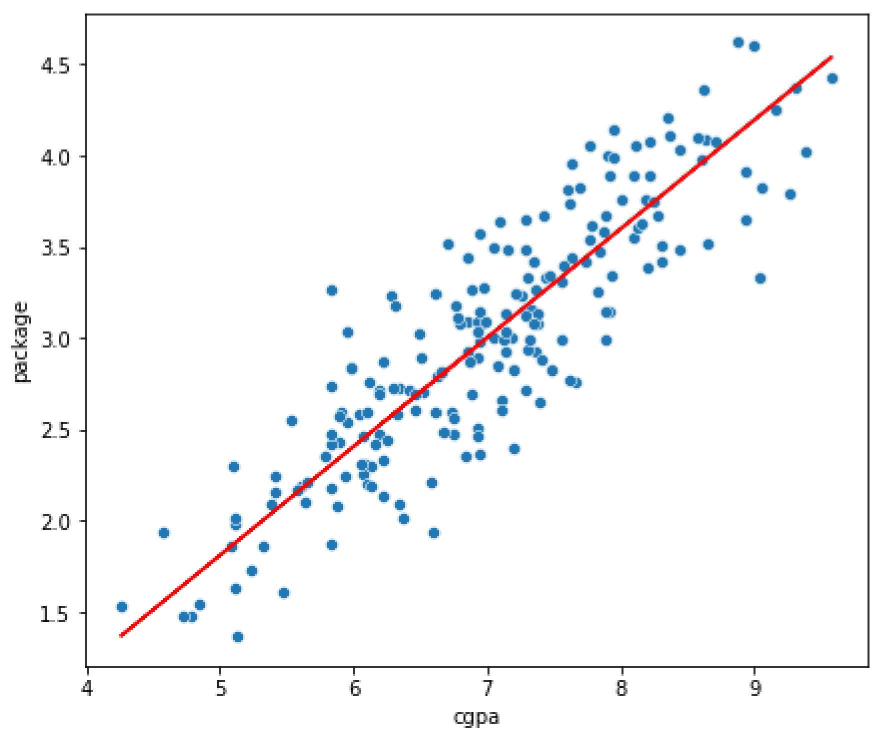
Out[24]: 77.06668029575103

In [25]:
```
sv.score(x_train, y_train)*100
```

Out[25]: 77.45351616879739

Step 7: Draw Prediction Line

In [29]:
```
plt.figure(figsize=(7,6))
sns.scatterplot(x='cgpa', y='package', data=dataset)
plt.plot(dataset['cgpa'], sv.predict(x), color='red')
plt.show()
```



Train data through SVR Model - Kernel: poly

In [33]:
```
sv1 = SVR(kernel='poly', degree=3)
sv1.fit(x_train, y_train)
```

Out[33]:

```
▾ SVR
SVR(kernel='poly')
```

In [ ]:

In [34]:

```python
sv.score(x_test, y_test)*100
```

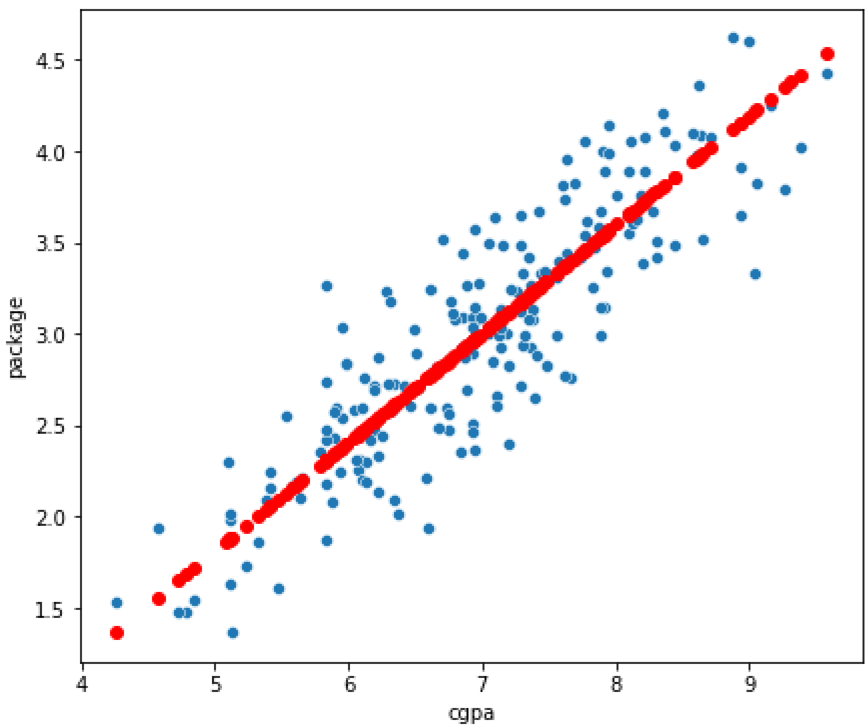
Out[34]:

```
77.06668029575103
```

In [ ]:

In [35]:

```python
plt.figure(figsize=(7,6))
sns.scatterplot(x='cgpa', y='package', data=dataset)
plt.scatter(dataset['cgpa'], sv.predict(x), color='red')
plt.show()
```



In [ ]:

47. HyperParameter Tuning, Model Parameter

47.1 What is a model parameter

Model Parameter are configuration varaibles that are internal to the model, and a model learns them on its own.

In following equation:

$$\mathrm{y = mx + c}$$

m and **c** are model parameters

47.2 Hyperparameter

- **Hyperparameters** are those parameters that are explicitly defined by the user to control the learning process.
- The best value can be determined either by the rule of thumb or by trial and error.
- These are usually default parameters.

No description has been provided for this image

Hyperparameter Tuning: Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are:

1. GridSearchCV
2. RandomizedSearchCV

47.2.1 GridSearchCV

- GridSearchCV is a technique to search through the best parameter values from the given set of the grid of parameters
- slower processing in case of large data

No description has been provided for this image

47.2.2 RandomizedSearchCV

- It goes through only a fixed number of hyperparameter settings
- It moves within the grid in a random fashion to find the best set of hyperparameters

In [ ]:
```

```

48. Hyperparameter Tuning (Practical)

In [10]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [11]:
```
dataset = pd.read_csv(r'Data/level_salaries.csv')
dataset.head(3)
```

Out[11]:

	Level	Salaries
0	1.000000	55167.141530
1	1.019019	48825.036941
2	1.038038	56692.389975

In [ ]:

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

In [ ]:

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [14]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

In [ ]:

In [15]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [16]:
```
dt = DecisionTreeRegressor()
dt.fit(x_train, y_train)
```

Out[16]:

▾ DecisionTreeRegressor

DecisionTreeRegressor()

In [ ]:

In [17]:
```
dt.score(x_test, y_test)*100
```

Out[17]: 73.22053360458676

In [18]:
```
dt.score(x_train, y_train)*100
```

Out[18]: 100.0

- Model is over-fitting

48.1 Perform Hyperparameters Tuning to reduce over-fitting

48.1.1 Tuning by GridSearchCV

In [19]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [24]:
```
df = {
    "criterion":["squared_error", "friedman_mse", "absolute_error","poisson"],
    "splitter":["best", "random"],
    "max_depth":[i for i in range(2,20)]
     }
```

In [28]:
```
gd = GridSearchCV(DecisionTreeRegressor(), param_grid=df)
gd.fit(x_train, y_train)
```

Out[28]:
```
GridSearchCV
estimator: DecisionTreeRegressor
DecisionTreeRegressor
```

In [29]:
```
gd.best_params_
```

Out[29]: {'criterion': 'squared_error', 'max_depth': 4, 'splitter': 'best'}

In [33]:
```
gd.best_score_
```

Out[33]: 0.8393136355736118

In [ ]:

In [30]:
```
dt2 = DecisionTreeRegressor(criterion='squared_error', max_depth=4, splitter='best'
dt2.fit(x_train, y_train)
```

Out[30]:
```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=4)
```

In [32]:
```
dt.score(x_test, y_test)*100, dt.score(x_train, y_train)*100
```

Out[32]: (73.22053360458676, 100.0)

In [ ]:

48.1.2 Tuning by RandomizedSearchCV

In [35]:
```
rd = RandomizedSearchCV(DecisionTreeRegressor(), param_distributions=df, n_iter=20)
rd.fit(x_train, y_train)
```

Out[35]:
```
▸ RandomizedSearchCV
▸ estimator: DecisionTreeRegressor
    ▸ DecisionTreeRegressor
```

In [ ]:

In [37]:
```
rd.score(x_test, y_test)*100, rd.score(x_train, y_train)*100
```

Out[37]:
```
(85.14998219015995, 86.78684301893401)
```

Over-Fitting is reduced significantly in this case

In [38]:
```
rd.best_params_
```

Out[38]:
```
{'splitter': 'best', 'max_depth': 4, 'criterion': 'squared_error'}
```

In [39]:
```
rd.best_score_
```

Out[39]:
```
0.8393136355736118
```

In [ ]:

49. Cross-Validation in Machine Learning

- It gives the information about how long your model can give you highest accuracy on particular data
- Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data
- It will give the range which will tell that your data has how much min accuracy and max accuracy it can attain

No description has been provided for this image

49.1 Methods used for cross-validation:

- Leave p out cross-validation
- Leave one out cross-validation
- Holdout cross-validation
- Repeated random subsampling validation
- k-fold cross-validation
- Stratified k-fold cross-validation
- Time Series cross-validation
- Nested cross-validation

49.1.1 K-Fold Cross-Validation:

- The original dataset is equally partitioned into k subparts or folds.
- Out of the k-folds or groups, for each iteration, one group is selected as validation data,
- and the remaining (k-1) groups are selected as training data.
- Not suitable for an imbalanced data

49.1.2 Stratified Cross-Validation:

- It works when the data is in classification nature
- It works on unbalanced data
- The original dataset is equally partitioned into k subparts or folds.
- Out of the k-folds or groups, for each iteration, one group is selected as validation data,
- and the remaining (k-1) groups are selected as training data.
- Stratified k-fold cross-validation solved the problem of imbalanced data

49.1.3 Leave-One-Out Cross-Validation:

- It gets trained on whole data
- It is an exhaustive cross-validation technique
- it is a category of Lp OCV with the case of p=1.
- It is slower in case of large data b/c of this issue it is used less
- The model trained by this method is very accurate

49.1.4 Leave-P-Out Cross-Validation:

- It is an exhaustive cross-validation technique, that involves using p-obervation as validation data
- It is slower in case of large data b/c of this issue it is used less

In [ ]:

```

```

50. Cross-Validation in Machine Learning (Practical)

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

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

Out[2]:

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25

In [ ]:

In [3]:
```
x = dataset.iloc[:,:-1]
y = dataset['package']
```

50.1 Check how much accuracy this data can have

We will use cross-validation

In [14]:
```
# Below model will ask for estimator (on which model you want to train it on)
from sklearn.linear_model import LinearRegression
```

In [16]:
```
from sklearn.model_selection import cross_val_score
```

In [18]:
```
# cv: cross-validation: number or LeaveOneOut, LeavePOut, KFold, Stratified, KFold
p = cross_val_score(LinearRegression(), x,y,cv=5)
p
```

Out[18]:
```
array([0.75398043, 0.79051763, 0.75683837, 0.78086775, 0.70887127])
```

In [20]:
```
p.sort()
p*100
```

Out[20]:
```
array([70.88712673, 75.39804264, 75.68383749, 78.0867752 , 79.05176315])
```

min_accuracy: 70% and max_accuracy: 79%

In [ ]:

In [24]:
```
# cv: cross-validation: number or LeaveOneOut, LeavePOut, KFold, StratifiedKFold
p1 = cross_val_score(LinearRegression(), x,y,cv=KFold(n_splits=10))
p1.sort()
p1*100
```

Out[24]:
```
array([60.48000765, 65.67540106, 67.20523867, 69.890411  , 73.50599138,
       74.37616704, 80.3181025 , 82.0986355 , 82.64799643, 83.96333567])
```

min_accuracy: 60% and max_accuracy: 83%

In [ ]:

50.2 Cross-Validation Methods

In [4]:
```
new_data = dataset.head(10)
```

In [9]:
```
x_new = new_data.iloc[:,:-1]
y_new = new_data['package']
```

In [6]:
```
from sklearn.model_selection import LeaveOneOut, LeavePOut, KFold, StratifiedKFold
```

In [10]:
```
lo = LeaveOneOut()

for train, test in lo.split(x_new,y_new):
    print(train, test)
```

```
[1 2 3 4 5 6 7 8 9] [0]
[0 2 3 4 5 6 7 8 9] [1]
[0 1 3 4 5 6 7 8 9] [2]
[0 1 2 4 5 6 7 8 9] [3]
[0 1 2 3 5 6 7 8 9] [4]
[0 1 2 3 4 6 7 8 9] [5]
[0 1 2 3 4 5 7 8 9] [6]
[0 1 2 3 4 5 6 8 9] [7]
[0 1 2 3 4 5 6 7 9] [8]
[0 1 2 3 4 5 6 7 8] [9]
```

In [12]:
```
lp = LeavePOut(p=2)

for train, test in lp.split(x_new,y_new):
    print(train, test)
```

```
[2 3 4 5 6 7 8 9] [0 1]
[1 3 4 5 6 7 8 9] [0 2]
[1 2 4 5 6 7 8 9] [0 3]
[1 2 3 5 6 7 8 9] [0 4]
[1 2 3 4 6 7 8 9] [0 5]
[1 2 3 4 5 7 8 9] [0 6]
[1 2 3 4 5 6 8 9] [0 7]
[1 2 3 4 5 6 7 9] [0 8]
[1 2 3 4 5 6 7 8] [0 9]
[0 3 4 5 6 7 8 9] [1 2]
[0 2 4 5 6 7 8 9] [1 3]
[0 2 3 5 6 7 8 9] [1 4]
[0 2 3 4 6 7 8 9] [1 5]
[0 2 3 4 5 7 8 9] [1 6]
[0 2 3 4 5 6 8 9] [1 7]
[0 2 3 4 5 6 7 9] [1 8]
[0 2 3 4 5 6 7 8] [1 9]
[0 1 4 5 6 7 8 9] [2 3]
[0 1 3 5 6 7 8 9] [2 4]
[0 1 3 4 6 7 8 9] [2 5]
[0 1 3 4 5 7 8 9] [2 6]
[0 1 3 4 5 6 8 9] [2 7]
[0 1 3 4 5 6 7 9] [2 8]
[0 1 3 4 5 6 7 8] [2 9]
[0 1 2 5 6 7 8 9] [3 4]
[0 1 2 4 6 7 8 9] [3 5]
[0 1 2 4 5 7 8 9] [3 6]
[0 1 2 4 5 6 8 9] [3 7]
[0 1 2 4 5 6 7 9] [3 8]
[0 1 2 4 5 6 7 8] [3 9]
[0 1 2 3 6 7 8 9] [4 5]
[0 1 2 3 5 7 8 9] [4 6]
[0 1 2 3 5 6 8 9] [4 7]
[0 1 2 3 5 6 7 9] [4 8]
[0 1 2 3 5 6 7 8] [4 9]
[0 1 2 3 4 7 8 9] [5 6]
[0 1 2 3 4 6 8 9] [5 7]
[0 1 2 3 4 6 7 9] [5 8]
[0 1 2 3 4 6 7 8] [5 9]
[0 1 2 3 4 5 8 9] [6 7]
[0 1 2 3 4 5 7 9] [6 8]
[0 1 2 3 4 5 7 8] [6 9]
[0 1 2 3 4 5 6 9] [7 8]
[0 1 2 3 4 5 6 8] [7 9]
[0 1 2 3 4 5 6 7] [8 9]
```

In [13]:

```
kf = KFold(n_splits=5)

for train, test in kf.split(x_new,y_new):
    print(train, test)
```

```
[2 3 4 5 6 7 8 9] [0 1]
[0 1 4 5 6 7 8 9] [2 3]
[0 1 2 3 6 7 8 9] [4 5]
[0 1 2 3 4 5 8 9] [6 7]
[0 1 2 3 4 5 6 7] [8 9]
```

In [ ]:

```
sf = StratifiedFold(n_splits=5)

for train, test in kf.split(x_new,y_new):
    print(train, test)

# It will generate error, b/c it works only in classification analysis, and don't w
```

In [ ]:

```

```

51. Unsupervised Learning

- Unsupervised learning is a type of machine learning that learns from **unlabelled data**.
- In labelled data, we know input and output, but in unlabelled data, we don't know about output
- This means that the data does not have any pre-existing labels or categories
- The goal of unsupervised learning is to discover patterns and relationships in the data w/o any explicit guidance.
- In Unsupervised learning, we do categorization and clusttering of the data

No description has been provided for this image

No description has been provided for this image

In [ ]: