Exercise 1

1.1 Problem Statement:

Implement CART algorithm for decision tree learning. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

1.2 Description of the Algorithm:

Decision Trees are an important type of algorithm for predictive modeling machine learning. Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

CART is a decision tree algorithm and it stands for Classification and Regression Trees. CART can be used for classification or regression predictive modeling problems. The representation for the CART model is a binary tree. It builds decision trees based on the attribute selection measure "Gini's Impurity Index". CART is an umbrella word that refers to the following types of decision trees:

Classification Trees: These are used to forecast the value of a categorical target variable.

Regression trees: These are used to forecast the value of a continuous target variable.

Advantages of Decision Trees:

- Decision trees are simple to understand, interpret, visualize.
- Decision trees implicitly perform variable screening or feature selection.
- Decision trees require little data preparation.
- Decision trees can handle both numerical and categorical data.
- Decision trees can also handle single-class and multi-class classification problems.
- Nonlinear relationships between parameters do not affect tree performance.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

Disadvantages of Decision Trees:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging, boosting etc.
- Predictions of decision trees are neither smooth nor continuous, but are piecewise constant approximations. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

Advantages of CART algorithm:

- CART can handle missing values in the data.
- CART is not significantly impacted by outliers in the input variables.
- CART is non-parametric; Thus it does not depend on information from a certain sort of distribution.
- CART supports growing a full decision tree and further post-pruning the decision tree so that the probability that important structure in the data set will be overlooked by stopping too soon is minimized.
- CART combines both testing with a test data set and cross-validation to more precisely measure the goodness of fit.
- CART allows to utilize the same attributes many times in various regions of the tree. This skill is capable of revealing intricate interdependencies between groups of variables.
- CART can be used in conjunction with other prediction methods to select the input set of variables.

1.3 Description of the Dataset:

Title of the dataset: Iris Plants Database

The Iris Dataset contains information of three species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor). The data set contains 3 classes of 50 instances each, where each class

refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Data Set Characteristics: Multivariate

Area: Life Sciences

Number of samples (or instances) in the Dataset: 150

Number of attributes (or features) in the Dataset: 05

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica
6. Number of samples of each species of iris flowers:
   Class Distribution: 33.3% for each of 3 classes.
   50 (Setosa), 50 (Versicolor), 50 (Virginica)

Predicted Attribute: class of iris plant

Missing Attribute Values: None

Source of Dataset: sklearn.datasets

Feature Name	Units of measurement	Datatype	Description
sepal length	Centimeters	Real (Numerical)	Length of Iris flower's sepal
sepal width length	Centimeters	Real (Numerical)	Width of Iris flower's sepal
petal length	Centimeters	Real (Numerical)	Length of Iris flower's petal
petal width length	Centimeters	Real (Numerical)	Width of Iris flower's petal
variety	Variety of species [Setosa, Virginica, Versicolor]	Object (Categorical)	Variety of the species of Iris flower

1.4 Description of the Machine Learning Library Classes and Methods used:

The **scikit-learn** (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It provides simple and efficient tools for predictive data analysis. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

sklearn.model_selection.train_test_split()

The **sklearn.model_selection.train_test_split()** method splits arrays or matrices into random train and test subsets. The parameters for the method are as follows:

sklearn.model_selection.train_test_split(**arrays*, *test_size=None*, *train_size=None*, *random_state=None*, *shuffle=True*, *stratify=None*)

It returns lists containing train-test split of inputs.

sklearn.tree.DecisionTreeClassifier()

The **sklearn.tree.DecisionTreeClassifier** is a class capable of performing multi-class classification on a dataset. It takes as input two arrays: an array X, holding the training samples, and an array Y holding the class labels for the training samples. In case that there are multiple classes with the same and highest probability, the classifier will predict the class with the lowest index amongst those classes. As an alternative to outputting a specific class, the probability of each class can be predicted, which is the fraction of training samples of the class in a leaf. DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification. The parameters of the DecisionTreeClassifier class are as follows:

class sklearn.tree.DecisionTreeClassifier(***, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*)

The default values for the parameters controlling the size of the trees (e.g. max_depth, min_samples_leaf, etc.) lead to fully grown and unpruned trees which can potentially be very large on some data sets.

Limitations of sklearn.tree.DecisionTreeClassifier class:

- scikit-learn implementation does not support categorical variables for now.
- scikit-learn implementation can build only CART trees for now.

The **DecisionTreeClassifier.fit()** method builds a decision tree classifier from the training set (X, y). The parameters of the fit() method are as follows:

fit(*X*, *y*, *sample_weight=None*, *check_input=True*, *X_idx_sorted='deprecated'*)

It returns an fitted Decision Tree estimator.

The **DecisionTreeClassifier.predict()** method predicts class or regression value for X. The parameters of the predict() method are as follows:

predict(*X*, *check_input=True*)

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

sklearn.tree.plot_tree()

The **sklearn.tree** class supports visualization of decision trees. Once a Decision Tree Classifier is built it can be visualized in test representation using **sklearn.tree.export_text()** method. The parameters for the method are as follows:

sklearn.tree.export_text(*decision_tree*, *, *feature_names=None*, *max_depth=10*, *spacing=3*, *decimals=2*, *show_weights=Fal*

se)

Alternatively, the decision tree can be plotted **sklearn.tree.plot_tree()** method. The parameters for the method are as follows:

sklearn.tree.plot_tree(*decision_tree*, *, *max_depth=None*, *feature_names=None*, *class_names=None*, *label='all'*, *filled=False*, *impurity=True*, *node_ids=False*, *proportion=False*, *rounded=False*, *precision=3*, *ax=None*, *fontsize=None*)

sklearn.metrics.accuracy_score()

Once a Decision Tree Classifier is built, it can be evaluated for performance. The **sklearn.metrics** module implements several loss, score, and utility functions to measure classification performance. The **sklearn.metrics.accuracy_score()** method computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true. The parameters for the method are as follows:

sklearn.metrics.accuracy_score(*y_true*, *y_pred*, *, *normalize=True*, *sample_weight=None*)

1.5 Implementation

```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np

iris = pd.read_csv("iris.csv")
iris
```

o/p:

```
     sepal.length  sepal.width  petal.length  petal.width    variety
0             5.1          3.5           1.4          0.2     Setosa
1             4.9          3.0           1.4          0.2     Setosa
2             4.7          3.2           1.3          0.2     Setosa
3             4.6          3.1           1.5          0.2     Setosa
4             5.0          3.6           1.4          0.2     Setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]
```

```
# explore the shape of the dataset
iris.shape
```

```
(150, 5)
```

```
# explore the information of the dataset

iris.info()
```

o/p:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

The decision tree model is to be constructed in order to "predict the variety of the iris flowers given their sepal & petal length & width". Hence, "variety" attribute is Target variable. The attributes "sepal.length" , "sepal.width" , "petal.length", "petal.width" are the independent variables.

Extracting Independent variables "sepal.length" , "sepal.width" , "petal.length", "petal.width"

X is a dataframe comprising of Independent variables data.

```
X = iris.iloc[ : , 0:4]

# display X dataframe
X
```

o/p:

```
     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 x 4 columns]
```

Extracting Dependent variables "variety"

Y is a dataframe comprising of Dependent variable's data.

```
Y = iris.iloc[ : , 4: ]

# display Y dataframe
Y.variety.unique()
```

o/p:

```
array(['Setosa', 'Versicolor', 'Virginica'], dtype=object)
```

Implementing CART algorithm for decision tree learning.

```
# construct the decision tree model
# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X ,
                                                    Y,
                                                    test_size = 0.25,
                                                    random_state = 5)
X_train
```

o/p:

```
     sepal.length  sepal.width  petal.length  petal.width
40            5.0          3.5           1.3          0.3
115           6.4          3.2           5.3          2.3
142           5.8          2.7           5.1          1.9
69            5.6          2.5           3.9          1.1
17            5.1          3.5           1.4          0.3
..            ...          ...           ...          ...
8             4.4          2.9           1.4          0.2
73            6.1          2.8           4.7          1.2
144           6.7          3.3           5.7          2.5
118           7.7          2.6           6.9          2.3
99            5.7          2.8           4.1          1.3

[112 rows x 4 columns]
```

```
# construct the decision tree model

# create an instance "DecisionTreeClassifier" class
# set the criterion to be "entropy"


from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state = 1234, criterion = 'entropy')


# fit() method will construct the decision tree
# by fitting the given training dataset.

clf.fit(X_train , Y_train)


DecisionTreeClassifier(criterion='entropy', random_state=1234)

# Visualize the decision tree as text representation

from sklearn import tree

text_representation = tree.export_text(clf)

print(text_representation)
```

o/p:

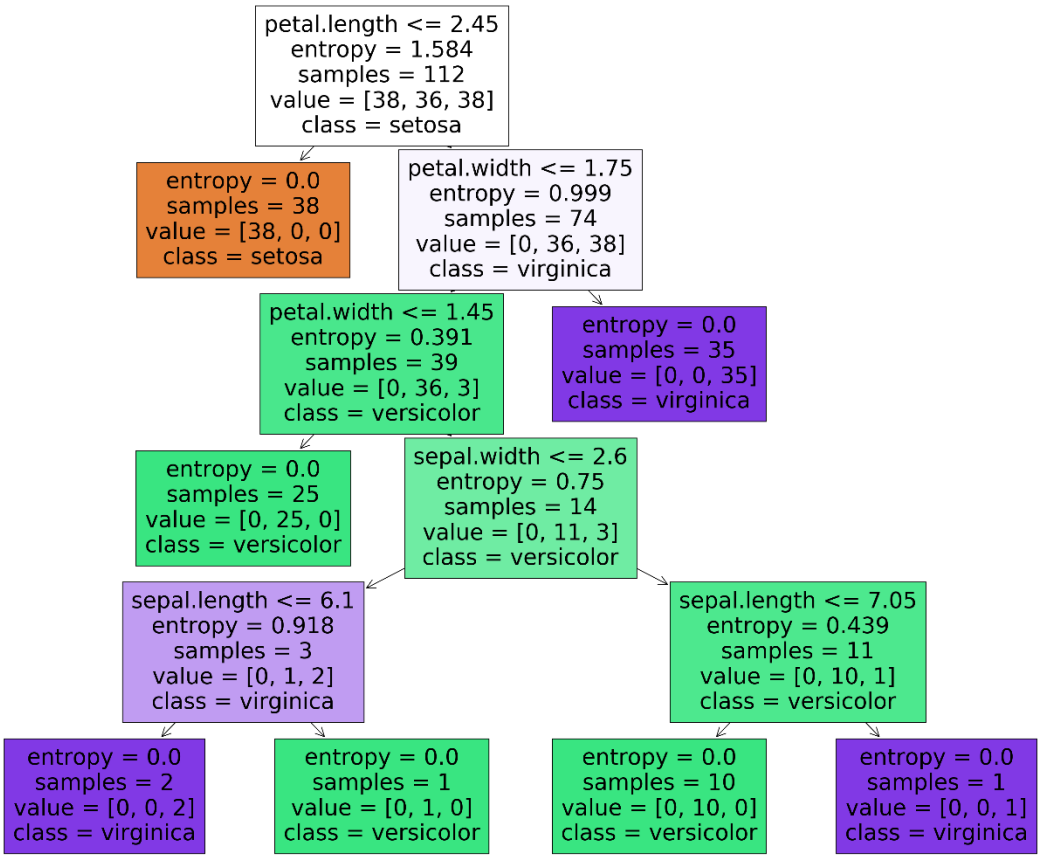
```
|--- feature_2 <= 2.45
|   |--- class: Setosa
```

```
|--- feature_2 >  2.45
|   |--- feature_3 <= 1.75
|   |   |--- feature_3 <= 1.45
|   |   |   |--- class: Versicolor
|   |   |--- feature_3 >  1.45
|   |   |   |--- feature_1 <= 2.60
|   |   |   |   |--- feature_0 <= 6.10
|   |   |   |   |   |--- class: Virginica
|   |   |   |   |--- feature_0 >  6.10
|   |   |   |   |   |--- class: Versicolor
|   |   |   |--- feature_1 >  2.60
|   |   |   |   |--- feature_0 <= 7.05
|   |   |   |   |   |--- class: Versicolor
|   |   |   |   |--- feature_0 >  7.05
|   |   |   |   |   |--- class: Virginica
|   |--- feature_3 >  1.75
|   |   |--- class: Virginica
```

plot_tree() will work only from sklearn version 0.21 and above.

```
# visualizing the decision tree pictorially

fig = plt.figure(figsize = (25 , 20) , dpi = 200.0)

_ = tree.plot_tree(clf,
                   feature_names = ['sepal.length' ,'sepal.width',
'petal.length', 'petal.width'],
                   class_names = ['setosa', 'versicolor', 'virginica'],
                   filled = True)
```

o/p:




```
# Now, Let us test the accuracy of the decision tree model
# on the training data and on test data.
```

```
# predict() method predicts classes for test dataset
# accuracy_score() will take original class labels,  predicted class labels
and
#  computes accuracy of the model.


# Let us first test the accuracy of the model on the training data itself.


from sklearn.metrics import accuracy_score

pred_train = clf.predict(X_train)

accuracy_train = accuracy_score(Y_train, pred_train)

print('% of Accuracy on training data: ', accuracy_train * 100 )



# Let us test the accuracy of the model on the test data (or new data or
unseen data).

pred_test = clf.predict(X_test)

accuracy_test = accuracy_score(Y_test, pred_test)

print('% of Accuracy on test data: ', accuracy_test * 100 )
```

o/p:

```
% of Accuracy on training data:  100.0
% of Accuracy on test data:  92.10526315789474
```

It can be observed that the model performs almost perfect on the training dataset. But its ability to make predictions on new data is less.

Applying the Decision Tree model to classify a new sample.

```
# Use the decision tree model to predict class label of new sample i.e.,
classify new sample.

# creating new iris flower's data and loading into Dataframe

new_data = {'sepal.length' : [3.7],
            'sepal.width' :  [3.0],
            'petal.length' : [2.2],
            'petal.width' : [1.3] }

new_df = pd.DataFrame(new_data)

new_df.head()
```

o/p:

```
   sepal.length  sepal.width  petal.length  petal.width
0           3.7          3.0           2.2          1.3

new_pred = clf.predict(new_df)

print('Prediction: The variety of the iris flower is ' , new_pred)
```

o/p:

```
Prediction: The variety of the iris flower is  ['Setosa']
```

1..6 Results and Discussion

- ✓ CART algorithm has been implemented for decision tree learning using scikit-learn machine learning library.
- ✓ Iris dataset has been used and decision tree machine learning model has been constructedwith a height of 5 and number of nodes in the tree were 13.
- ✓ The accuracy score of the decision tree was 92.1%.

Hence, a decision tree model has been trained using the iris flowers data that predicts the species of the new iris flower

Exercise 2

2.1 Problem Statement:

Explore the problem of overfitting in decision tree and develop solution using pruning technique.

2.2 Description of the Algorithm:

Decision Trees are an important type of algorithm for predictive modeling machine learning. Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

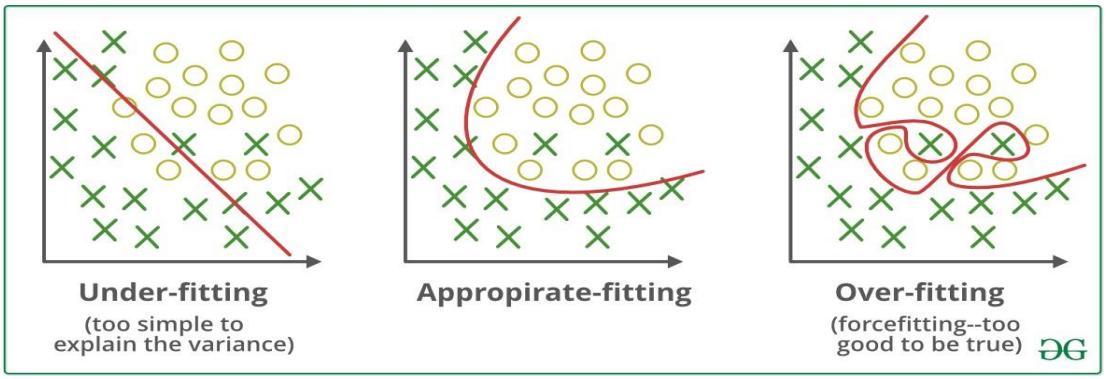
CART is a decision tree algorithm and it stands for Classification and Regression Trees. CART can be used for classification or regression predictive modeling problems. The representation for the CART model is a binary tree. It builds decision trees based on the attribute selection measure "Gini's Impurity Index". CART is an umbrella word that refers to the following types of decision trees:

Classification Trees: These are used to forecast the value of a categorical target variable.

Regression trees: These are used to forecast the value of a continuous target variable.

2.3 Description of the Overfitting Problem:

In statistics, a fit refers to how well the model approximates a target function. The terms underfitting and overfitting refer to how the model fails to match the data. The fitting of a model directly correlates to whether it will return accurate predictions from a given data set. Overfitting refers to a model that models too well (too tight) on the training data that it fails to generalize on new data. A statistical model is said to be overfitted when we train it with a lot of data *(just like fitting ourselves in oversized pants!)*. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too many details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore, they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.



2.4 Description of Dataset:

Title of the data set: Advertising Dataset

A company is maintaining its advertising data containing information about the users who have purchased their product and users who have not purchased their product. There are 400 samples in the dataset.

Data Set characteristics: Multivariable

Area: Advertising/Marketing

Number of Samples (or Instances) in the Dataset: 400

Number of Attributes (or Features) in the Dataset: 5 Attributes

Attribute Information:

1. User ID
2. Gender
3. Age
4. Estimated Salary
5. Purchased:
   ---Yes
   ---No
6. Number of Samples of Purchased = Yes Class: 143
7. Number of Samples of Purchased = No Class: 257

Predicted Attribute: Purchased

Missing Attribute Values: None

Source of Dataset: https://github.com/sudarshan-koirala/Logistic-Regression-Social-Network-Ads

Feature Name	Datatype	Description
User ID	Integer (Numerical)	Unique user ID given to each Customer
Gender	Object (Categorical)	Gender of Customer, Unique Values = Male, Female
Age	Integer (Numerical)	Age of Customer
Estimated Salary	Integer (Numerical)	Estimated Salary of Customer
Purchased	Object (Categorical)	It specifies whether the Customer has purchased the product or not. Unique Values = 0, 1 0 represents that the customer has not purchased the product; 1 represents that the customer has purchased the product;

2.5 Description of the Machine Learning Library Classes and Methods used:

The **scikit-learn** (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It provides simple and

efficient tools for predictive data analysis. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

sklearn.model_selection.train_test_split()

The **sklearn.model_selection.train_test_split()** method splits arrays or matrices into random train and test subsets. The parameters for the method are as follows:

sklearn.model_selection.train_test_split(**arrays*, *test_size=None*, *train_size=None*, *random_state=None*, *shuffle=True*, *stratify=None*)

It returns lists containing train-test split of inputs.

sklearn.tree.DecisionTreeClassifier()

The **sklearn.tree.DecisionTreeClassifier** is a class capable of performing multi-class classification on a dataset. It takes as input two arrays: an array X, holding the training samples, and an array Y holding the class labels for the training samples. In case that there are multiple classes with the same and highest probability, the classifier will predict the class with the lowest index amongst those classes. As an alternative to outputting a specific class, the probability of each class can be predicted, which is the fraction of training samples of the class in a leaf. DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification. The parameters of the DecisionTreeClassifier class are as follows:

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=N one*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*)

The default values for the parameters controlling the size of the trees (e.g. max_depth, min_samples_leaf, etc.) lead to fully grown and unpruned trees which can potentially be very large on some data sets.

Limitations of sklearn.tree.DecisionTreeClassifier class:

- scikit-learn implementation does not support categorical variables for now.
- scikit-learn implementation can build only CART trees for now.

The **DecisionTreeClassifier.fit()** method builds a decision tree classifier from the training set (X, y). The parameters of the fit() method are as follows:

fit(X, y, sample_weight=None, check_input=True, X_idx_sorted='deprecated')

It returns an fitted Decision Tree estimator.

The **DecisionTreeClassifier.predict()** method predicts class or regression value for X. The parameters of the predict() method are as follows:

predict(X, check_input=True)

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

sklearn.tree.export_text()

The **sklearn.tree** class supports visualization of decision trees. Once a Decision Tree Classifier is built it can be visualized in test representation using **sklearn.tree.export_text()** method. The parameters for the method are as follows:

*sklearn.tree.export_text(decision_tree, *, feature_names=None, max_depth=10, spacing =3, decimals=2, show_weights=False)*

Alternatively, the decision tree can be plotted **sklearn.tree.plot_tree()** method. The parameters for the method are as follows:

*sklearn.tree.plot_tree(decision_tree, *, max_depth=None, feature_names=None, class_n ames=None, label='all', filled=False, impurity=True, node_ids=False, proportion=False, rounded=False, precision=3, ax=None, fontsize=None)*

sklearn.metrics.accuracy_score()

Once a Decision Tree Classifier is built, it can be evaluated for performance. The **sklearn.metrics** module implements several loss, score, and utility functions to measure classification performance. The **sklearn.metrics.accuracy_score()** method computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true. The parameters for the method are as follows:

*sklearn.metrics.accuracy_score(y_true, y_pred, *, normalize=True, sample_weight=None)*

2.6 Using Grid Search CV to prune the decision tree model to handle the problem of over fitting:

What is Grid Search CV?

sklearn.model_selection.GridSearchCV()

class sklearn.model_selection.GridSearchCV(*estimator*, *param_grid*, *, *scoring=None*, *n_jobs=None*,*refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score = nan*, *return_train_score=False*)

GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. It performs an exhaustive search over specified parameter values for an estimator. So, in the end, you can select the best parameters from the listed hyperparameters. In addition to that, you can specify the number of times for the cross-validation for each set of hyperparameters. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Defining hyperparameters:

1. **estimator**: estimator object you created

2. **params_grid**: the dictionary object that holds the hyperparameters you want to try
3. **scoring**: evaluation metric that you want to use, you can simply pass a valid string/object of evaluation metric
4. **cv**: number of cross-validation you have to try for each selected set of hyperparameters
5. **verbose**: you can set it to 1 to get the detailed print out while you fit the data to GridSearchCV
6. **n_jobs**: number of processes you wish to run in parallel for this task if it -1 it will use all available processors.

return_train_score: ***bool, default=False***

> If False, the cv_results_ attribute will not include training scores. Computing training scores is used to get insights on how different parameter settings impact the overfitting/underfitting trade-off. However computing the scores on the training set can be computationally expensive and is not strictly required to select the parameters that yield the best generalization performance.

Important members are fit, predict. GridSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used

2.7 Implementation

using advertising dataset

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Advertising_data.csv')



# check no. of rows and columns in dataset

dataset.shape
```

o/p:

```
(400, 5)
```

```
# display the first 5 rows of dataset
dataset.head()
```

o/p:

```
    User ID  Gender   Age  EstimatedSalary  Purchased
0  15624510    Male  19.0          19000.0          0
1  15810944    Male  35.0          20000.0          0
2  15668575  Female  26.0          43000.0          0
3  15603246  Female  27.0          57000.0          0
4  15804002    Male  19.0          76000.0          0
```

```
# sklearn DecisionTreeClassifier cannot handle categorical data.
# Hence, categorical attributes must be transformed into numerical
attributes.


dataset['Gender'].replace(['Male', 'Female'],
                          [0, 1], inplace = True)

Dataset
```

o/p:

```
      User ID  Gender   Age  EstimatedSalary  Purchased
0    15624510       0  19.0          19000.0          0
1    15810944       0  35.0          20000.0          0
2    15668575       1  26.0          43000.0          0
3    15603246       1  27.0          57000.0          0
4    15804002       0  19.0          76000.0          0
..        ...     ...   ...              ...        ...
395  15691863       1  46.0          41000.0          1
396  15706071       0  51.0          23000.0          1
397  15654296       1  50.0          20000.0          1
398  15755018       0  36.0          33000.0          0
399  15594041       1  49.0          36000.0          1

[400 rows x 5 columns]
```

```
# check the unique values of 'Purchased' attribute

dataset.Purchased.unique()

array([0, 1], dtype=int64)
```

Let the knowledge to be gathered be "What are the characteristics of Potentital Customers??"

Hence, Target variable would be "Purchased" attribute. Remaining atrributes shall act as Independent variables.

The User ID does not affect the purchasal of products. Hence, ignored.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
# extracting the Independent variables into a dataframe

X = dataset.loc[:, ['Gender' , 'Age', 'EstimatedSalary']]
X
```

o/p:

```
     Gender   Age  EstimatedSalary
0         0  19.0          19000.0
1         0  35.0          20000.0
2         1  26.0          43000.0
3         1  27.0          57000.0
4         0  19.0          76000.0
..      ...   ...              ...
395       1  46.0          41000.0
396       0  51.0          23000.0
397       1  50.0          20000.0
398       0  36.0          33000.0
399       1  49.0          36000.0
```

```
[400 rows x 3 columns]
```

```
# extracting the Target variables into a dataframe

y = dataset.loc[:, 'Purchased']

y
```

o/p:

```
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
```

Implementing CART algorithm for decision tree learning

```
# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X ,
                                                    y,
                                                    test_size = 0.25,
                                                    random_state = 5)
```

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

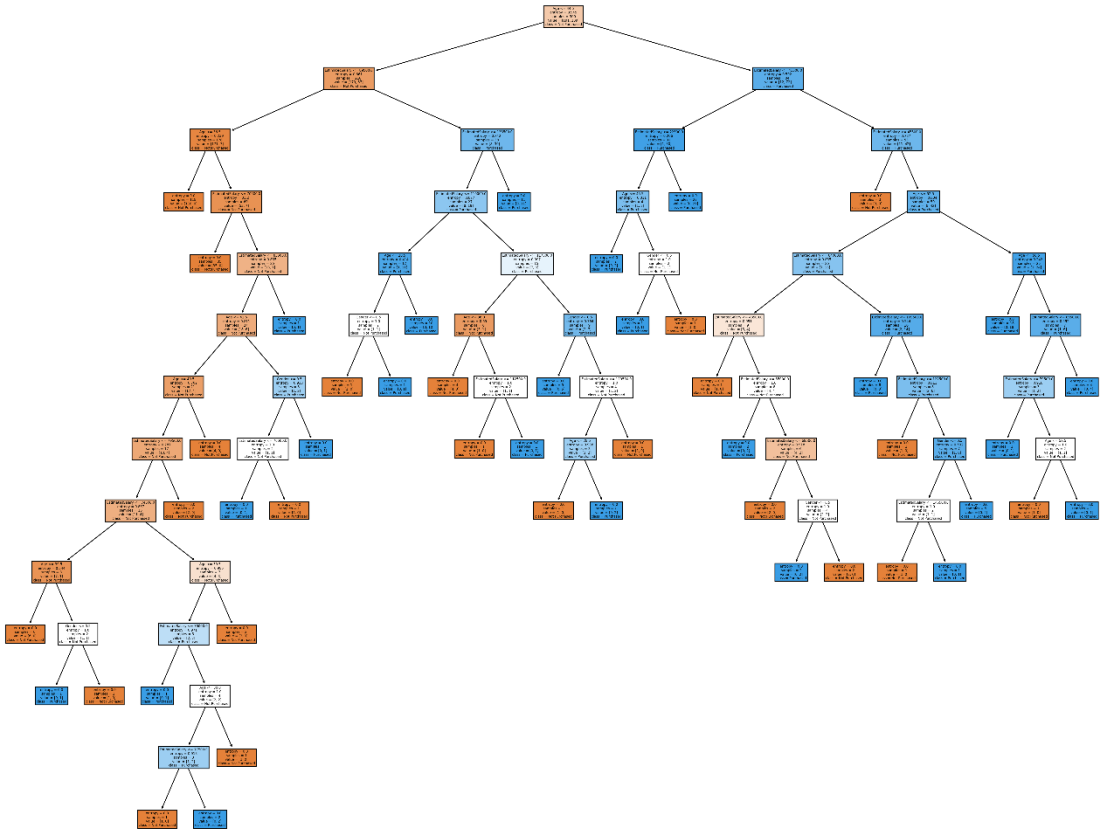
```
# construct the decision tree model

from sklearn.tree import DecisionTreeClassifier

# create an instance "DecisionTreeClassifier" class
# set the criterion to be "entropy"

dtc = DecisionTreeClassifier(criterion = 'entropy' ,
                             random_state = 0)

# fit() method will construct the decision tree
# by fitting the giventraining dataset.


dtc.fit(X_train , y_train)


DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Exploring the complexity of the decision tree model

```
# visualizing the decision tree.

from sklearn import tree

plt.figure(figsize = ( 25 , 20) , dpi = 300.0)
```

```
_ = tree.plot_tree(dtc,
                   feature_names = ['Gender' , 'Age' , 'EstimatedSalary'],
                   class_names = ['Not Purchased', 'Purchased'] ,
                    filled = True )
```

o/p:



```
# check the details of the tree

print ( 'Depth of the tree: ' , dtc.get_depth() )
print ( 'No. of leaves in the tree:' , dtc.get_n_leaves() )
```

o/p:

```
Depth of the tree:  13
No. of leaves in the tree: 47
```

```
# measure the accuracy of the constructed decision tree

from sklearn.metrics import accuracy_score

# predict() method predicts classes for given dataset
# accuracy_score() will compute accuracy of model given
#                  original class labels and predicted class labels.


# Let us first test the accuracy of the model on the training data itself.

pred_train = dtc.predict(X_train)

accuracy_train = accuracy_score(y_train, pred_train)

print('% of Accuracy on training data: ', accuracy_train * 100)
```

```
# Let us test the accuracy of the model on the test data.
# Test data (or new data) is previously unseen by the model.
# Hence, its performance may reduce.

pred_test = dtc.predict(X_test)

accuracy_test = accuracy_score(y_test, pred_test)

print('% of Accuracy on test data: ', accuracy_test * 100)
```

o/p:

```
% of Accuracy on training data:  100.0
% of Accuracy on test data:  80.0
```

It can be observed that the model performs almost perfect on the training dataset. But its ability to make predictions on new data is less.

using "GridSearchCV" for finding the optimal attribute measure and optimal depth of the decision tree.

set the parameters for gridsearch. Here, two params are set = criterion, max_depth. GridSearchCV will repeat the process of constructing decision tree by iterating through different parameters values and determines (1) whether 'entropy' or 'gini' is best for given training dataset?? (2) What is the best depth of the decision tree for the given training dataset.

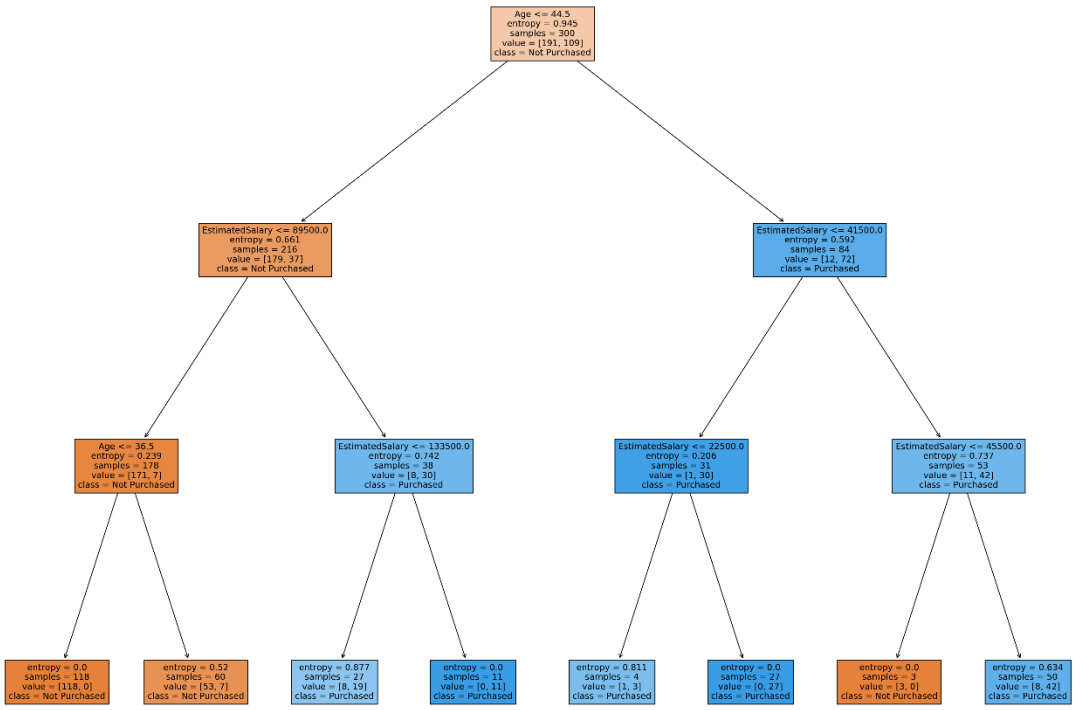
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

using GridSearchCV to prune the decision tree model to handle overfitting and hyperparameter tuning.

```
from sklearn.model_selection import GridSearchCV

# set the required parameters
params = {'criterion': ['entropy', 'gini'],
          'max_depth': range(1, 13)}

# create an instance of "GridSearchCV"  class

grid_search = GridSearchCV(estimator = dtc,
                           param_grid = params,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)

# Apply fit() method to construct different decision trees with given
parameters

grid_search = grid_search.fit(X_train, y_train)


# check the parameters of the best model

grid_search.best_estimator_

DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)


# create an insance of "DecisionTreeClassifier" class with best parameters
```

```
best_dtc = DecisionTreeClassifier(criterion = 'entropy' , max_depth=3,
random_state = 0)


# reconstruct the decision tree with best estimator parameters

best_dtc.fit(X_train , y_train)

DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)


plt.figure(figsize = ( 25 , 20) , dpi = 200.0)
_ = tree.plot_tree( best_dtc,
                    feature_names = ['Gender' , 'Age' , 'EstimatedSalary'],
                    class_names = ['Not Purchased', 'Purchased'] ,
                    filled = True)
```

o/p:




```
# check the details of the tree

print ( 'Depth of the tree after pruning: ' ,  best_dtc.get_depth() )

print ( 'No. of leaves in the tree after pruning: ' ,
best_dtc.get_n_leaves() )
```

o/p:

```
Depth of the tree after pruning:  3
No. of leaves in the tree after pruning:  8


# test the accuracy of the model by using test dataset
```

```
pred_test = best_dtc.predict(X_test)

accuracy_test = accuracy_score(y_test, pred_test)

print('% of Accuracy of the tree on test data after pruning : ',
accuracy_test * 100)
```

o/p:

```
% of Accuracy of the tree on test data after pruning :  90.0
```

2.8 Results and Discussion

- ✓ to construct decision tree.
- ✓ Upon analyzing the decision tree, it is noted that the decision tree has become too complex which has depth of 13 and the number of leaves in the tree is 47, this causes the problem of overfitting.
- ✓ It can be observed that the model performs almost perfect on the training dataset (100%). But its ability to make predictions on new data is less (80%).
- ✓ using "GridSearchCV" for finding the optimal attribute measure and optimal depth of thedecision tree.
- ✓ The maximum height of the tree is set to "3" with criterion as 'entropy'.
- ✓ The decision is plotted after the pruning and it is observed that the number of leaves in
  tree has decreased to "8".
- ✓ Accuracy of the tree on test data after pruning is "90%".

Hence, the problem of overfitting in the decision tree is explored and a solution is developedusing pruning technique.

Exercise 3

3.1 Problem Statement:

Perform Exploratory Data Analysis (EDA) on the given dataset. Implement CART algorithm for decision tree learning. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

3.2 Description of Exploratory Data Analysis (EDA):

In statistics, exploratory data analysis is an approach of analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods. Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling and thereby contrasts traditional hypothesis testing. Exploratory data analysis encourages statisticians to explore the data, and possibly formulate hypotheses that could lead to new data collection and experiments. Typical graphical techniques used in EDA are: Box plot, Histogram, Multi-vari chart, Run chart, Pareto chart, Scatter plot (2D/3D), Stem-and-leaf plot, Parallel coordinates, Odds ratio, Targeted projection pursuit, Heat map, Bar chart, Horizon graph, Glyph-based visualization methods, Projection methods etc. There are a number of tools that are useful for EDA, but EDA is characterized more by the attitude taken than by particular techniques.

3.3 Description of Data Preprocessing

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model. It involves steps: Handling Missing Data, Handling noise in data, Data Cleaning, Data Scaling, Encoding Categorical Data, Attribute subset selection etc.

3.4 Description of Dataset:

Title of the data set: Titanic Dataset

It is one of the most popular datasets used for understanding machine learning basics. It contains information of all the passengers aboard the RMS Titanic, which unfortunately was shipwrecked. This dataset can be used to predict whether a given passenger survived or not.

Data Set characteristics: Multivariable

Area: Tourism and Travel

Number of Samples (or Instances) in the Dataset: 400

Number of Attributes (or Features) in the Dataset: 5

Attribute Information:

1. Passenger
2. Survived
3. Pclass
4. Name
5. Sex --- Male, Female
6. Age
7. SibSp
8. Parch
9. Ticket
10. Fare
11. Cabin
12. Embarked

Predicted Attribute: Survived

Missing Attribute Values: Age, Embarked

Variable	Definition
survival	Survival {0 = No, 1 = Yes}
pclass	Ticket class {1 = Class A, 2 = Class B, 3 = Class C}
sex	Sex
Age	Age in years
sibsp	# of siblings / spouses aboard the Titanic
parch	# of parents / children aboard the Titanic
ticket	Ticket number
fare	Passenger fare
cabin	Cabin number
embarked	Port of Embarkation {C = Cherbourg, Q = Queenstown, S = Southampton, O= Others}

Source of Dataset: https://www.kaggle.com/c/titanic

3.5 Description of the Machine Learning Library Classes and Methods used:

plotly.express.histogram()

plotly.express Python library is used to perform **Exploratory Data Analysis**. In statistics, a **histogram** is representation of the distribution of numerical data, where the data are binned and the count for each bin is represented. More generally, in Plotly a histogram is an aggregated bar chart, with several possible aggregation functions (ex: sum, average, count...) which can be used to visualize data on categorical and date axes as well as linear axes.

plotly.express.histogram(data_frame=None, x=None, y=None, color=None, pattern_shape=None, facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None, hover_name=None, hover_data=None, animation_frame=None, animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None, color_discrete_map=None, pattern_shape_sequence=None, pattern_shape_map=None, marginal=None, opacity=None, orientation=None, barmode='relative', barnorm=None, histnorm=None, log_x=False, log_y=False, range_x=None, range_y=None, histfunc=None, cumulative=None, nbins=None, text_auto=False, title=None, template=None, width=None, height=None)

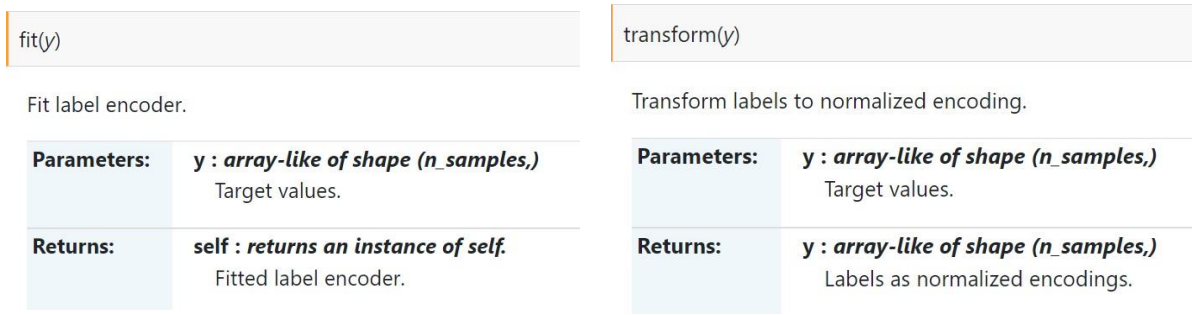
Pandas.DataFrame.fillna()

Missing values in Age attribute are filled by the method "Fill with Mean". Missing values in Embarked attribute are filled by the method "Fill with Mode". For filling missing values in the dataset the fillna() method is used from Python Pandas library.

Pandas.DataFrame.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None)

sklearn.preprocessing.LabelEncoder()

Categorical attributes are transformed into numerical attributes using Label Encoder in sklearn.preprocessing. The fit() method fits the labels to the given attribute and the transform() method transforms the data of the attribute.



fit(*y*)

Fit label encoder.

Parameters:	**y : *array-like of shape (n_samples,)*** Target values.
Returns:	**self : *returns an instance of self.*** Fitted label encoder.

transform(*y*)

Transform labels to normalized encoding.

Parameters:	**y : *array-like of shape (n_samples,)*** Target values.
Returns:	**y : *array-like of shape (n_samples,)*** Labels as normalized encodings.

3.6 Implementation:

importing required python libraries and reading dataset into a dataframe.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

reading dataset from the .csv file and loading into Dataframe.

```
titanic = pd.read_csv('D:/Python Datasets/titanic.csv')
```

Print the number of rows (records) and columns(attributes) in the X dataset.

```
titanic.shape
```

```
(891, 12)
```

head() function can be used to get a feel of the dataset. head(no. of rows) is an inbuilt function. It returns the first n rows.

```
titanic.head(5)
```

o/p:

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp
\
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

sample(no. of rows) function can also be used to get a feel of the dataset. sample(n) is an inbuilt function of random module in Python. It returns a "n" randomly chosen rows from the sequence. It uses random sampling without replacement.

```
titanic.sample(5)
```

o/p:

```
     PassengerId  Survived  Pclass                          Name   Sex   Age
\
21            22         1       2       Beesley, Mr. Lawrence  male  34.0
62            63         0       1  Harris, Mr. Henry Birkhardt  male  45.0
616          617         0       3    Danbom, Mr. Ernst Gilbert  male  34.0
844          845         0       3          Culumovic, Mr. Jeso  male  17.0
373          374         0       1          Ringhini, Mr. Sante  male  22.0

     SibSp  Parch    Ticket      Fare Cabin Embarked
21       0      0    248698   13.0000   D56        S
62       1      0     36973   83.4750   C83        S
616      1      1    347080   14.4000   NaN        S
844      0      0    315090    8.6625   NaN        S
373      0      0  PC 17760  135.6333   NaN        C
```

Extracting Independent variables.
"PassengerId", "Name", "Ticket", "Cabin" are ignored as it does not affect the survival.
X is a dataframe comprising of Independent variables data.

```
X = pd.DataFrame(titanic.loc[ : , \
            ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
'Embarked']])
```

Visualizing X data frame

```
X
```

o/p:

```
     Pclass     Sex   Age  SibSp  Parch     Fare Embarked
0         3    male  22.0      1      0   7.2500        S
1         1  female  38.0      1      0  71.2833        C
2         3  female  26.0      0      0   7.9250        S
3         1  female  35.0      1      0  53.1000        S
4         3    male  35.0      0      0   8.0500        S
..      ...     ...   ...    ...    ...      ...      ...
886       2    male  27.0      0      0  13.0000        S
887       1  female  19.0      0      0  30.0000        S
888       3  female   NaN      1      2  23.4500        S
889       1    male  26.0      0      0  30.0000        C
890       3    male  32.0      0      0   7.7500        Q

[891 rows x 7 columns]
```

Extracting and target (or dependent) variable -- "Survived".
Y is a dataframe comprising of Dependent variable's data.

```
Y = pd.DataFrame(titanic.loc[:, ['Survived']])
```

Visualizing Y data frame

```
Y
```

o/p:

```
     Survived
0           0
1           1
2           1
3           1
4           0
..        ...
886         0
887         1
888         0
889         1
890         0

[891 rows x 1 columns]
```

Check, the unique values of target variable. Here, target variable is "Survived".

```
Y.Survived.unique()
```

o/p:

```
array([0, 1], dtype=int64)
```

The result shows that there are two unique values for
"Survived" attribute represented as

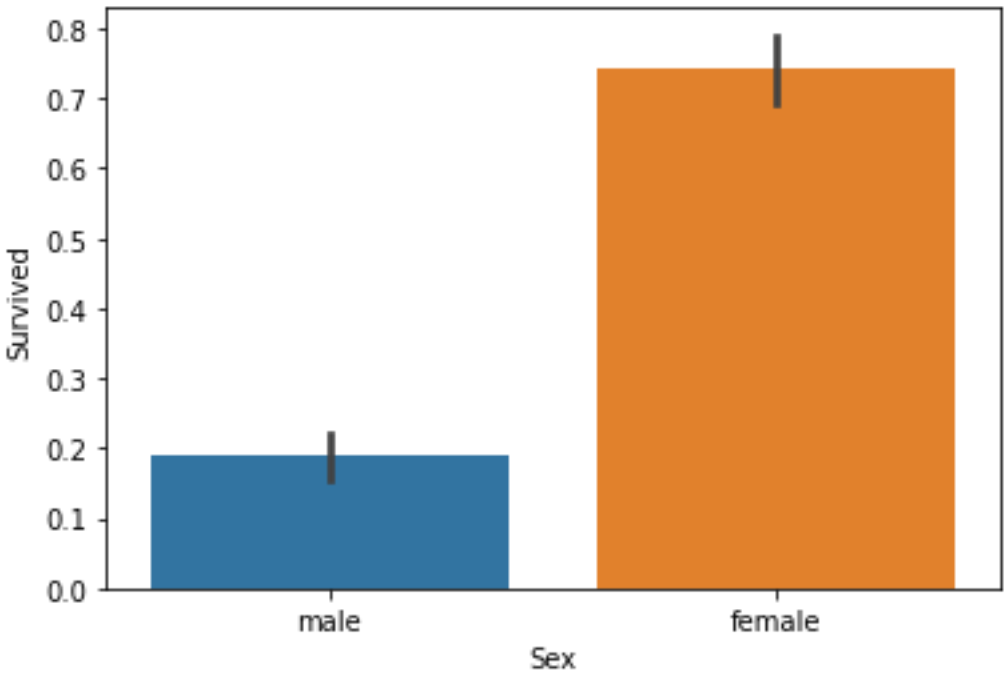
```
    "Survived = 0", if passenger has not survived Titanic crash
    "Survived = 1", if passenger has survived Titanic crash
```

Hence, the given dataset leads to single class classification problem.
Exploratory Data Analysis (EDA) is a method used to analyze and summarize datasets. Majority of the EDA techniques involve the use of graphs.

```
# Countplot of "Sex" vs. "Survived"
sns.barplot(x = X.Sex, y = Y.Survived)
```

```
<AxesSubplot:xlabel='Sex', ylabel='Survived'>
```

o/p:

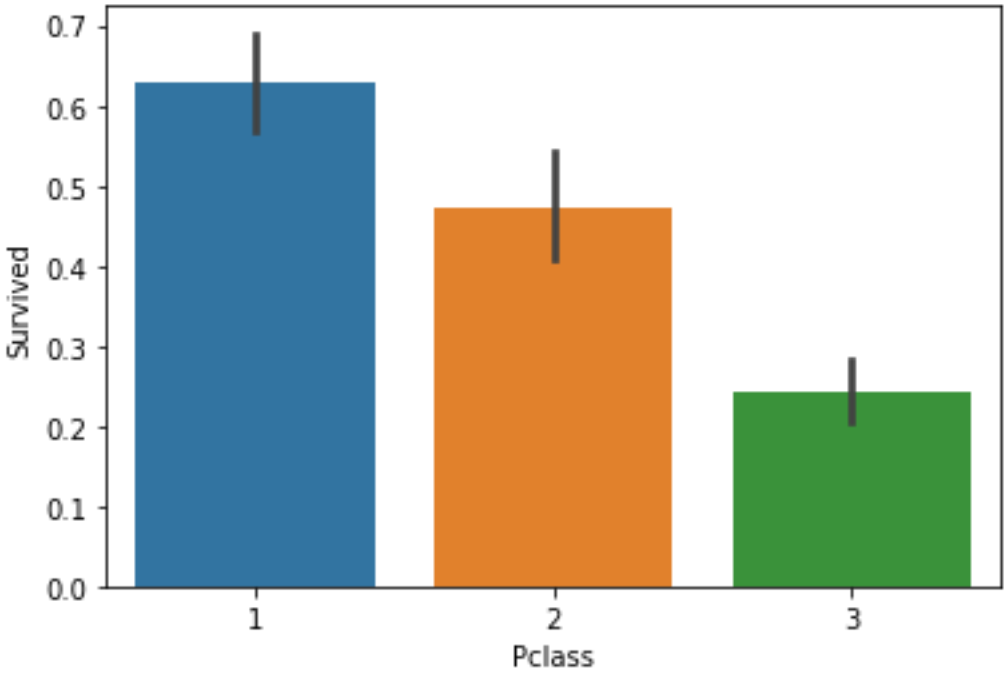


By observing the graph, it can be approximated that the survival rate of men is around 20% and that of women is around 75%. Therefore, whether a passenger is a male or a female plays an important role in determining if one is going to survive.

```
# Countplot of "Pclass" vs. "Survived"
sns.barplot(x = X.Pclass, y = Y.Survived)
```

```
<AxesSubplot:xlabel='Pclass', ylabel='Survived'>
```

o/p:

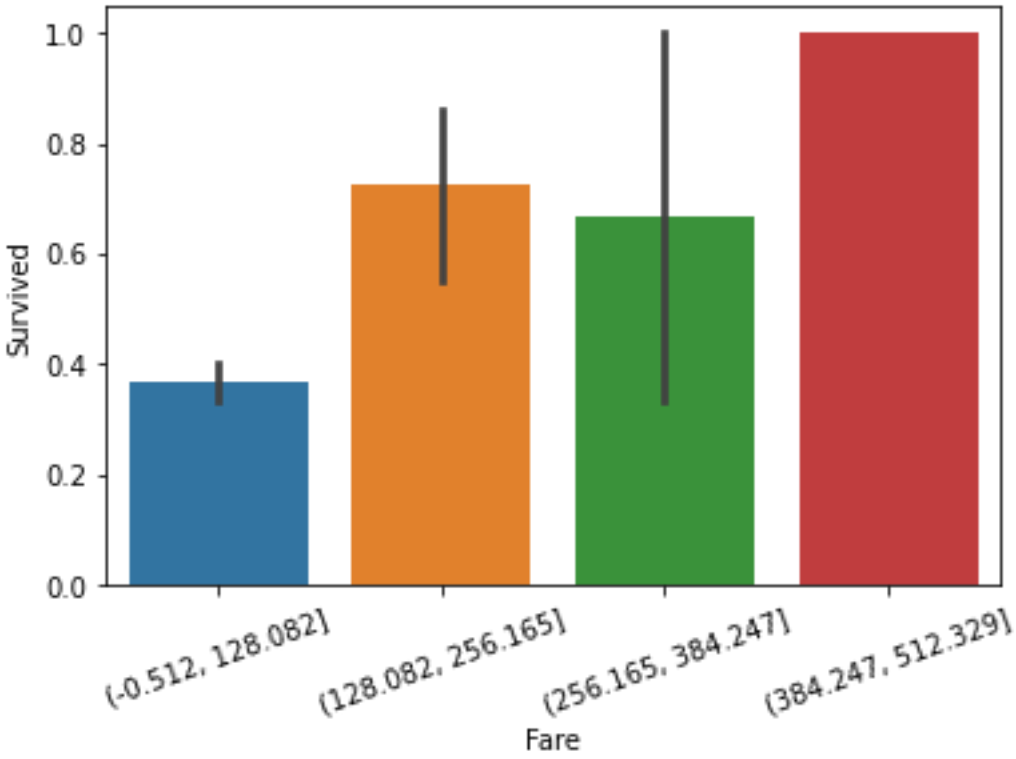


Class 1 passengers have a higher survival chance compared to classes 2 and 3. It implies that Pclass contributes a lot to a passenger's survival rate.

```
# Divide Fare into 4 bins
Fare_Range = pd.cut(X['Fare'], 4)

# Barplot - Shows approximate values based
# on the height of bars.
sns.barplot(x = Fare_Range, y = Y.Survived )

plt.xticks(rotation = 20)
```

o/p:

```
(array([0, 1, 2, 3]),
 [Text(0, 0, '(-0.512, 128.082]'),
  Text(1, 0, '(128.082, 256.165]'),
  Text(2, 0, '(256.165, 384.247]'),
  Text(3, 0, '(384.247, 512.329]')])
```

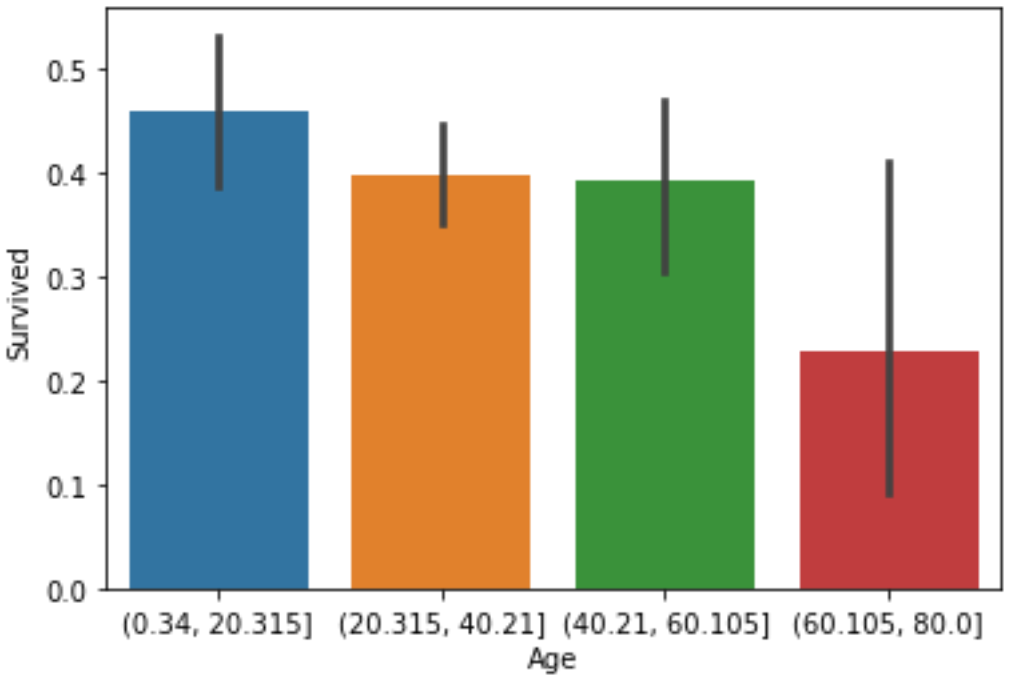


It can be observed that passengers who paid more fare for the ticket, survived more. That is people of higher class survived more.

```
# Divide Age into 4 bins
Age_Range = pd.cut(X['Age'], 4)

# Barplot - Shows approximate values based
# on the height of bars.
sns.barplot(x = Age_Range, y =Y.Survived)
```

```
<AxesSubplot:xlabel='Age', ylabel='Survived'>
```

o/p:

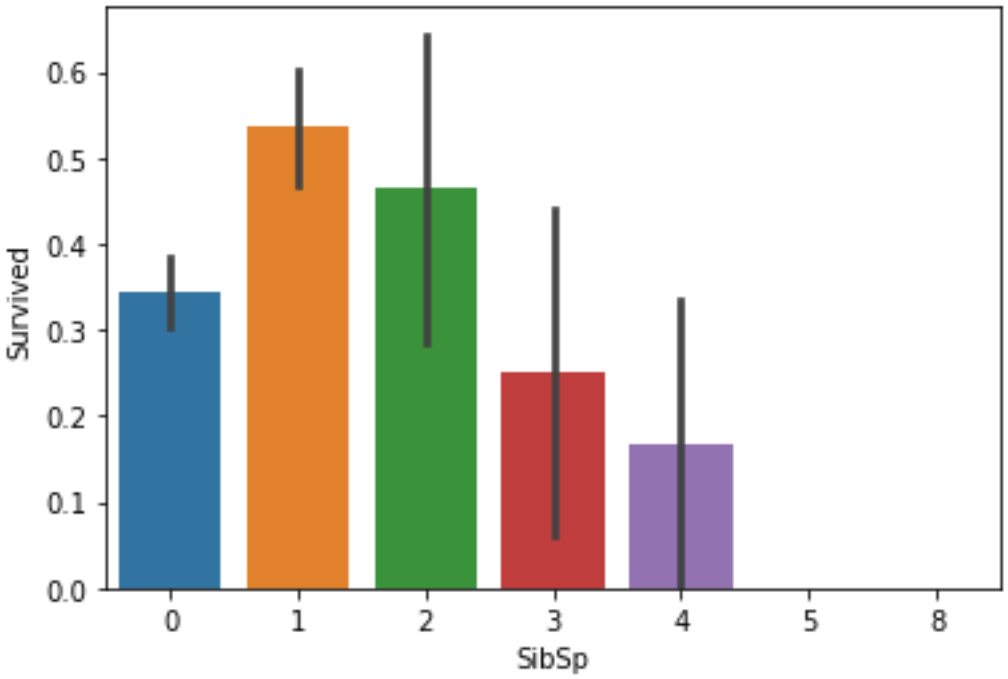


it can be observed children survived more and old-aged people survived less.

```
# Countplot of "SibSp" vs. "Survived"
sns.barplot(x = X.SibSp, y = Y.Survived)
```

```
<AxesSubplot:xlabel='SibSp', ylabel='Survived'>
```

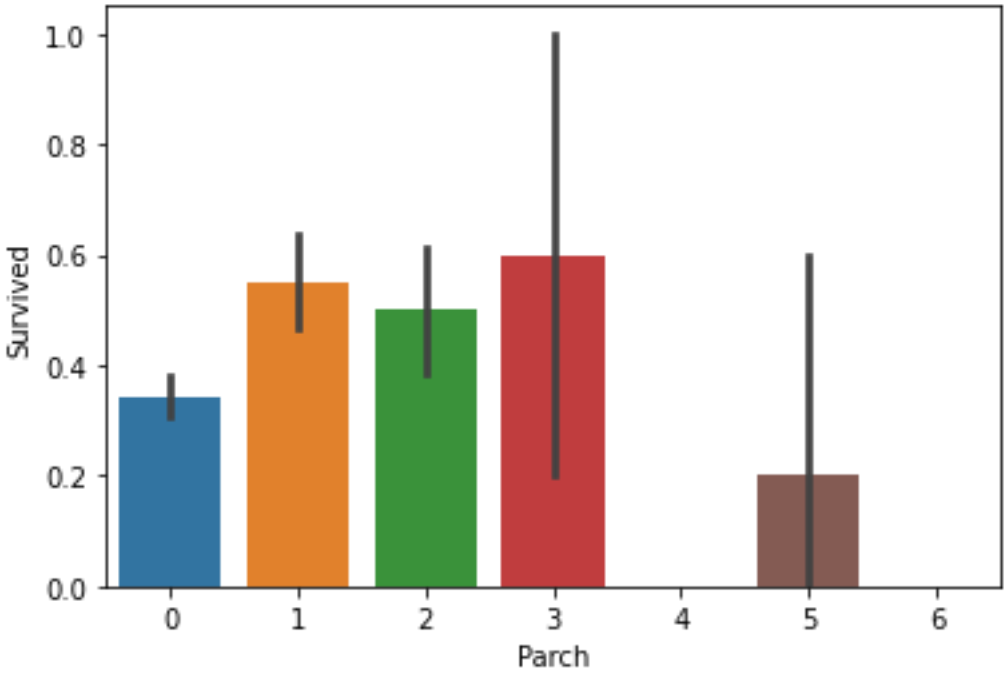
o/p:




```
# Countplot of "Parch" vs. "Survived"
sns.barplot(x = X.Parch, y = Y.Survived )
```

```
<AxesSubplot:xlabel='Parch', ylabel='Survived'>
```

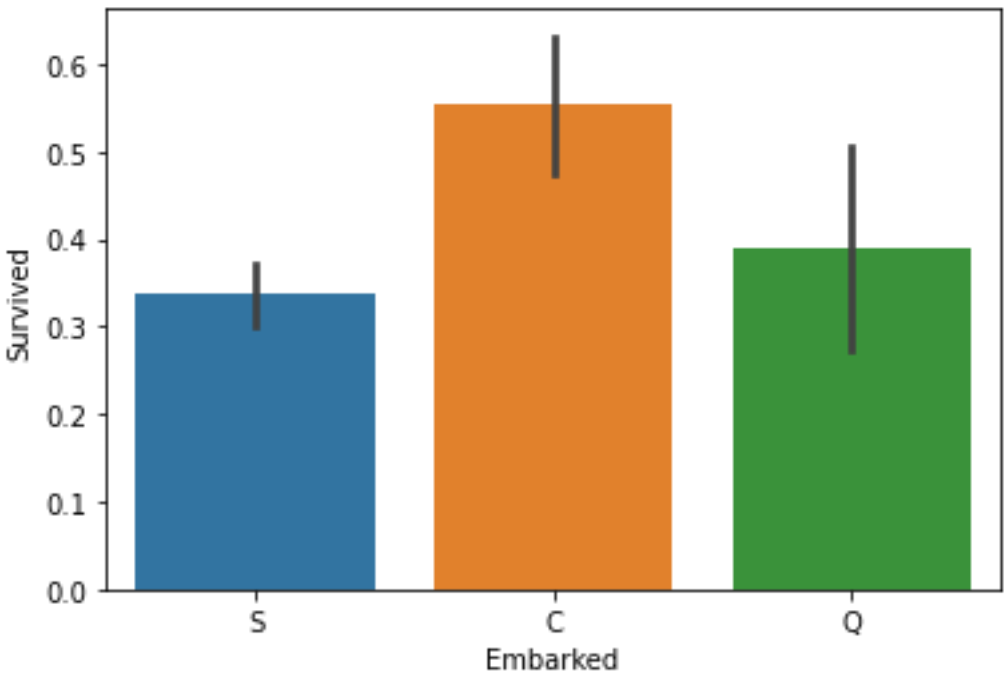
o/p:




```
# Countplot of "Embarked" vs. "Survived"
sns.barplot(x = X.Embarked, y = Y.Survived)
```

```
<AxesSubplot:xlabel='Embarked', ylabel='Survived'>
```

o/p:



The survival rate graph showed that passengers who came from Cherbourg were more likely to survive.
Print the information of dataset.
Displays informaton of each attribute - column name, non-null count, datatype

```
X.info()
```

o/p:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
```

```
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    891 non-null    int64
 1   Sex       891 non-null    object
 2   Age       714 non-null    float64
 3   SibSp     891 non-null    int64
 4   Parch     891 non-null    int64
 5   Fare      891 non-null    float64
 6   Embarked  889 non-null    object
dtypes: float64(2), int64(3), object(2)
memory usage: 48.9+ KB
```

it can be observed that there are missing values in the Age, Embarked attributes

isnull() function can also be used to check if there are any null values in dataset. mean() function will return the average number of fields with null values.

```
print('% of missing values')
X.isnull().mean()*100
```

o/p:

```
% of missing values

Pclass       0.000000
Sex          0.000000
Age         19.865320
SibSp        0.000000
Parch        0.000000
Fare         0.000000
Embarked     0.224467
dtype: float64
```

Missing values must be handled as part of preprocessing. The missing values in the 'Age', 'Embarked' attributes must be filled.

fillna() method is used to fill missing values in the dataset.

When inplace = True , the data is modified in place, which means it will return nothing and the dataframe is now updated.

When inplace = False , which is the default, then the operation is performed and it returns a copy of the object.

```
# Handling missing values in the “Age” attribute with the mean of "Age"

# Finding the mean of "Age"

meanAge = X['Age'].mean()
meanAge
```

o/p:

```
29.69911764705882
```

```
# Filling missing "Age" fields with mean of "Age"

X['Age'].fillna( meanAge , inplace = True)


# check that all the missing "Age" values are filled
X.isna().sum()
```

o/p:

```
Pclass      0
```

```
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    2
dtype: int64
```

Handling missing values in “Embarked” atrribute using mode of "Embarked".

Finding the mode value of the “Embarked” attribute. Mode is the value that appears most frequently in a set of values of an attribute. An attribute may have one mode, more than one mode (multi-modal), or no mode at all.

```
modeEmbarked = X['Embarked'].mode()[0]

modeEmbarked
```

o/p:

```
'S'
```

```
# Replacing the missing values in the “Embarked” column with mode of
"Embarked"

X['Embarked'].fillna( modeEmbarked , inplace = True)

# check that all the missing "Embarked" values are filled
X.isna().sum()
```

o/p:

```
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

```
# Check if there are any categorical attributes in the dataset
X.info()
```

o/p:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    891 non-null    int64
 1   Sex       891 non-null    object
 2   Age       891 non-null    float64
 3   SibSp     891 non-null    int64
 4   Parch     891 non-null    int64
 5   Fare      891 non-null    float64
 6   Embarked  891 non-null    object
dtypes: float64(2), int64(3), object(2)
memory usage: 48.9+ KB
```

```
# You may check the unique values of the 'Sex' Attribute
X.Sex.unique()
```

o/p:

```
array(['male', 'female'], dtype=object)
```

```
# You may check the unique values of the 'Embarked' Attribute
X.Embarked.unique()
```

o/p:

```
array(['S', 'C', 'Q'], dtype=object)
```

Handling categorical attributes.
Categorical attributes have to be transformed into numerical attributes.
sklearn.preprocessing.LabelEncoder -- Encodes target labels with value between 0 and n_classes-1.
Label encoding converts the data in machine-readable form, but it assigns a unique number(starting from 0) to each class of data. This may lead to generation of priority issues in the training datasets. A label with a high value may be considered to have high priority than a label having a lower value.

```
from sklearn import preprocessing

# fit the 'Sex' attribute for label encoding

label_encoder_Sex = preprocessing.LabelEncoder().fit(X['Sex'])


# Encode labels for 'Sex' attribute

X['Sex'] = label_encoder_Sex.transform(X['Sex'])
X.head(5)
```

o/p:

```
   Pclass  Sex   Age  SibSp  Parch     Fare Embarked
0       3    1  22.0      1      0   7.2500        S
1       1    0  38.0      1      0  71.2833        C
2       3    0  26.0      0      0   7.9250        S
3       1    0  35.0      1      0  53.1000        S
4       3    1  35.0      0      0   8.0500        S
```

```
# fit the 'Embarked' attribute for label encoding

label_encoder_Embarked = preprocessing.LabelEncoder().fit(X['Embarked'])

# Encode labels for 'Embarked' attribute

X['Embarked'] = label_encoder_Embarked.transform(X['Embarked'])
X.head(5)
```

o/p:

```
   Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
0       3    1  22.0      1      0   7.2500         2
1       1    0  38.0      1      0  71.2833         0
2       3    0  26.0      0      0   7.9250         2
3       1    0  35.0      1      0  53.1000         2
4       3    1  35.0      0      0   8.0500         2
```

```
# Go for attribute selection as part of preprocessing
```

```
# Check correlation of attributes with target variable

X.corrwith(Y.Survived, method='pearson')
```

o/p:

```
Pclass     -0.338481
Sex        -0.543351
Age        -0.069809
SibSp      -0.035322
Parch       0.081629
Fare        0.257307
Embarked   -0.167675
dtype: float64
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.25,
                                                   random_state = 35)
X_train
```

o/p:

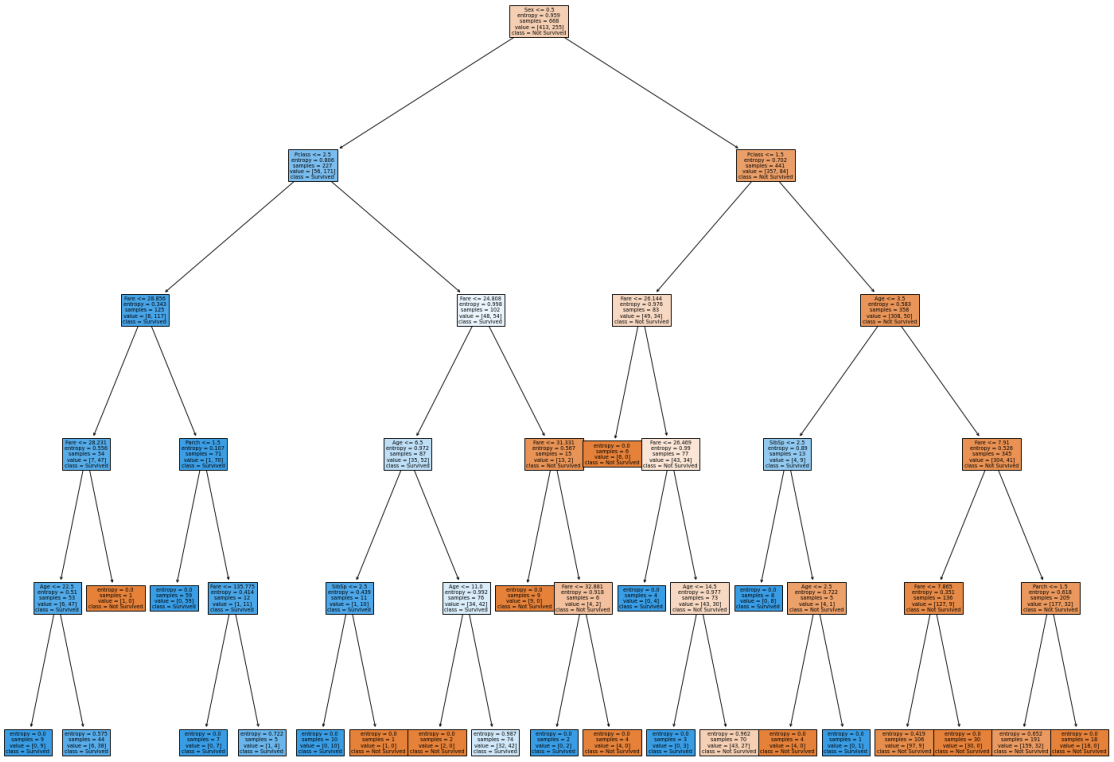
```
     Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
786       3    0  18.0      0      0   7.4958         2
636       3    1  32.0      0      0   7.9250         2
401       3    1  26.0      0      0   8.0500         2
811       3    1  39.0      0      0  24.1500         2
780       3    0  13.0      0      0   7.2292         0
..      ...  ...   ...    ...    ...      ...       ...
249       2    1  54.0      1      0  26.0000         2
448       3    0   5.0      2      1  19.2583         0
33        2    1  66.0      0      0  10.5000         2
271       3    1  25.0      0      0   0.0000         2
713       3    1  29.0      0      0   9.4833         2

[668 rows x 7 columns]
```

training the model

```
from sklearn.tree import DecisionTreeClassifier

# creating an object of KNeighborsClassifier class
dtc = DecisionTreeClassifier( criterion = 'entropy' , random_state = 1 )

# train the  model
dtc.fit(X_train, Y_train)

DecisionTreeClassifier(criterion='entropy', random_state=1)

from sklearn import tree

fig = plt.figure( figsize = (25 , 20) )

_ = tree.plot_tree( dtc,
                   feature_names = ['Pclass' ,'Sex', 'Age', 'SibSp',
'Parch', 'Fare', 'Embarked'],
```

```
                    class_names = ['Not Survived' , 'Survived'] ,
                    filled = True)
```

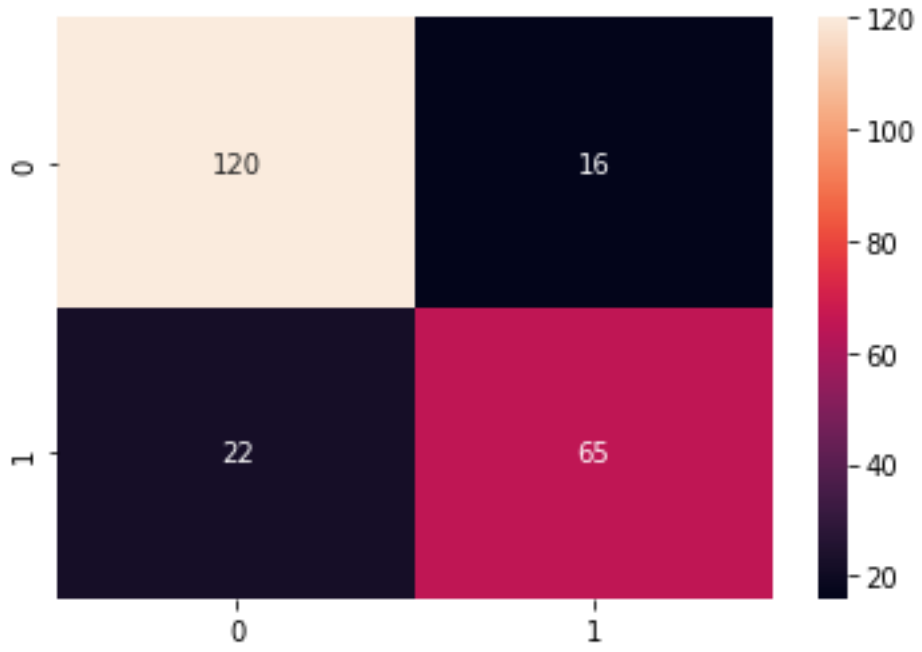
o/p:



```
# testing the model for accuracy of its predictions

# predict() function will make predictions on test dataset

Y_pred = dtc.predict(X_test)



from sklearn.metrics import confusion_matrix , accuracy_score

# make a confusion matrix
cm = confusion_matrix(Y_test, Y_pred)

# display confusion matrix as a heatmap
sns.heatmap(cm, annot = True , fmt = 'd')

# compute and display the the accuracy of the KNN model
print('The % of Accuracy is :', accuracy_score(Y_test , Y_pred)*100)
```

o/p:

```
The % of Accuracy is : 82.95964125560538
```




```
# use Decision Tree model to make predictions

# creating new passenger's data and loading into Dataframe

# Passenger is female(0) aged 25 with 2 siblings and travelled with 2
parents;
# She has 1st class ticket with fare = 300.0; She embarked ship at
Southampton(2).

new_data = {'Pclass': [1], 'Sex': [0], 'Age': [25], 'SibSp': [2],
'Parch':[2],
            'Fare': [300.0], 'Embarked': [2]}

new_df = pd.DataFrame(new_data)

new_df.head()
```

o/p:

```
   Pclass  Sex  Age  SibSp  Parch   Fare  Embarked
0       1    0   25      2      2  300.0         2
```

```
new_pred = dtc.predict(new_df)

if new_pred == 0:
    print('Prediction: The passenger will not survive')
else:
    print('Prediction: The passenger will survive')
```

o/p:

```
Prediction: The passenger will survive
```

3.7 Results and Discussion:

- ✓ Exploratory Data Analaysis (EDA) method is used to analyse and summarize dataset. Histograms were plotted and inferences were drawn to get insights into the characteristics and relationships of the dataset.
- ✓ By observing the histograms, the following inferences were drawn:
    - It can be approximated that the survival rate of men is around 19% and that of women is around 75%. Therefore, whether a passenger is a Male or a Female plays an important role in determining if one is going to survive.
    - It can be observed that passengers who paid more fare for the ticket, survived more.That is people of higher class survived more.
    - It can be observed children survived more and old-aged people survived less.
    - The survival rate graph showed that passengers who embarked from "Cherbourg" port were more likely to survive.
- ✓ Missing values were handled as part of preprocessing. "fillna()" method is used to fill missing values in the dataset.
- ✓ CART algorithm is implemented for decision tree learning.
- ✓ The decision tree constructed was with height = 4 and number of nodes were 15.
- ✓ Accuracy of the decision tree model is observed to be 81.17%.
- ✓ The decision tree model is used to predict the survival of the new passenger = {'Pclass': [1], 'Sex': [0], 'Age': [25], 'SibSp': [2], 'Parch':[2], 'Fare': [300.0], 'Embarked': [2]} and the prediction made by the decision tree model was "The Passenger will Survive".

Hence, Exploratory Data Analysis, Data Preprocessing were successfully performed on the Titanic dataset. CART algorithm was implemented for decision tree learning. The decision tree model was used to classify a new sample.

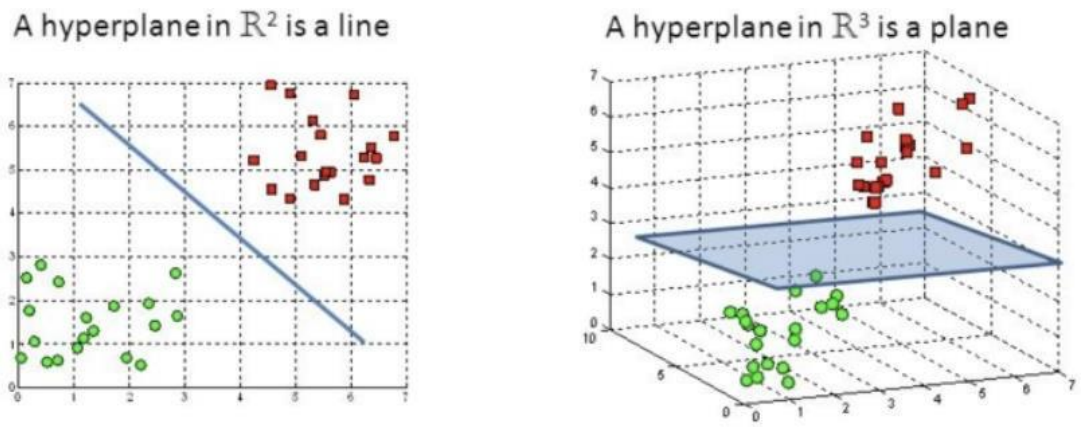
Exercise 4

4.1 Problem Statement:

Train an SVM based classifier to predict whether the cancer is malignant or benign.

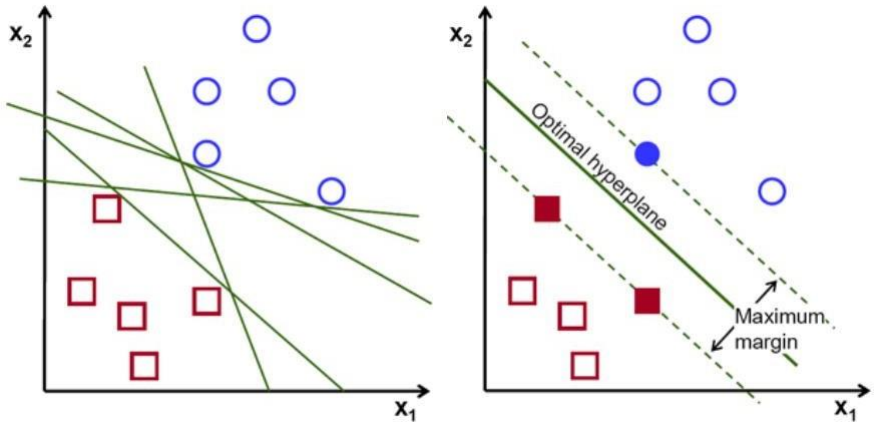
4.2 Description of the Algorithm:

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers' detection. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.



Hyperplanes in 2D and 3D feature space

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e., the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.



Possible hyperplanes

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then

the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 1. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

4.3 Description of the Dataset:

Title of the Data Set: Cancer Data

This data set has information about classification of tumors into malignant (cancerous) or benign (non-cancerous) using features obtained from several cell images.

Data Set characteristics: Multivariable

Area: Health and Medical

Number of Sample (or Instances) in the Dataset: 569

Number of Attributes (or Features) in the Dataset: 32

Attribute Information:

1) ID number
2) Diagnosis (M = malignant, B = benign)

Ten real-valued features are computed for each cell nucleus:
a) radius (mean of distances from centre to points on the perimeter)
b) texture (standard deviation of Gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)

I) symmetry
j) fractal dimension ("coastline approximation" - 1)

Predicted Attribute: Diagnosis

Missing values: No

Source of Dataset: http://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28diagnostic%29

4.4 Description of the Machine Learning Library Classes and Methods used:

sklearn.svm.SVC()

class sklearn.svm.SVC(, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None)*

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using LinearSVC or SGDClassifier instead, possibly after a Nystroem transformer. The multiclass support is handled according to a one-vs-one scheme.

Parameters:

C: *float, default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel: *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*

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).

degree: *int, default=3*

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma: *{'scale', 'auto'} or float, default='scale'*

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

coef0: *float, default=0.0*

Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

tol: *float, default=1e-3*

Tolerance for stopping criterion.

random_state : *int, RandomState instance or None, default=None*

Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False. Pass an int for reproducible output across multiple function calls.

Attributes:

class_weight: ndarray of shape (n_classes,)
Multipliers of parameter C for each class. Computed based on the class_ parameter.

classes: ndarray of shape (n_classes,)
The classes labels.

shape_fit: tuple of int of shape (n_dimensions_of_X,)
Array dimensions of training vector X.

Feature_names_in: ndarray of shape (n_features_in_,)
Names of features seen during fit. Defined only when X has feature names that are all strings.

intercept: ndarray of shape (n_classes * (n_classes - 1) / 2,)
Constants in decision function.

4.5 Implementation:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

#Import Cancer data from the Sklearn library
# Dataset can also be found here
(http://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28diagnostic%29)

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
cancer
```

o/p:

{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01, 1.189e-01],
[2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01, 8.902e-02], [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01, 8.758e-02], ..., [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01, 7.820e-02], [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01, 1.240e-01], [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01, 7.039e-02]]),

'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,

0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
...
'worst perimeter', 'worst area', 'worst smoothness', 'worst compactness', 'worst concavity',
'worst concave points', 'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
'filename': 'breast_cancer.csv', 'data_module': 'sklearn.datasets.data'}

```
df_cancer = pd.DataFrame(np.c_[cancer['data'], cancer['target']], columns =
np.append(cancer['feature_names'], ['target']))
df_cancer.head()
```

o/p:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0.0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0.0

5 rows × 31 columns

```
df_cancer.shape
```

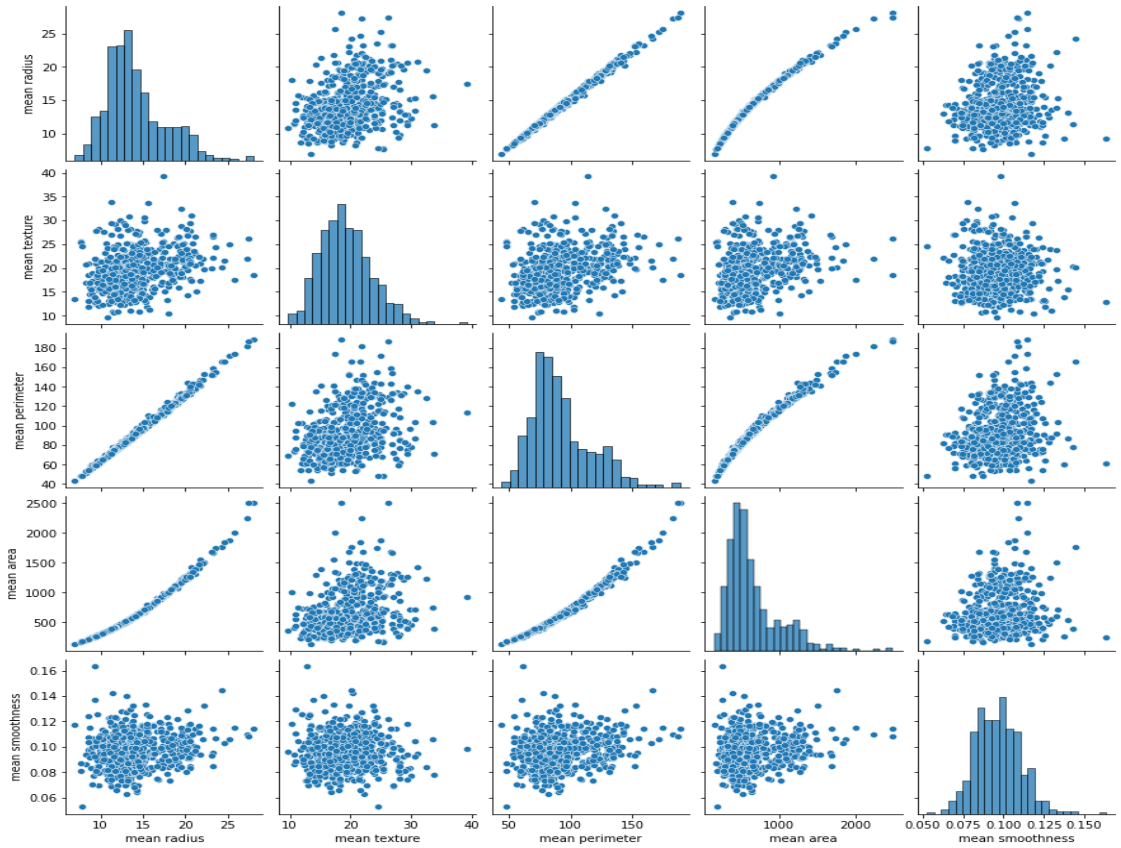
o/p: (569, 31)

```
df_cancer.columns
```

o/p:

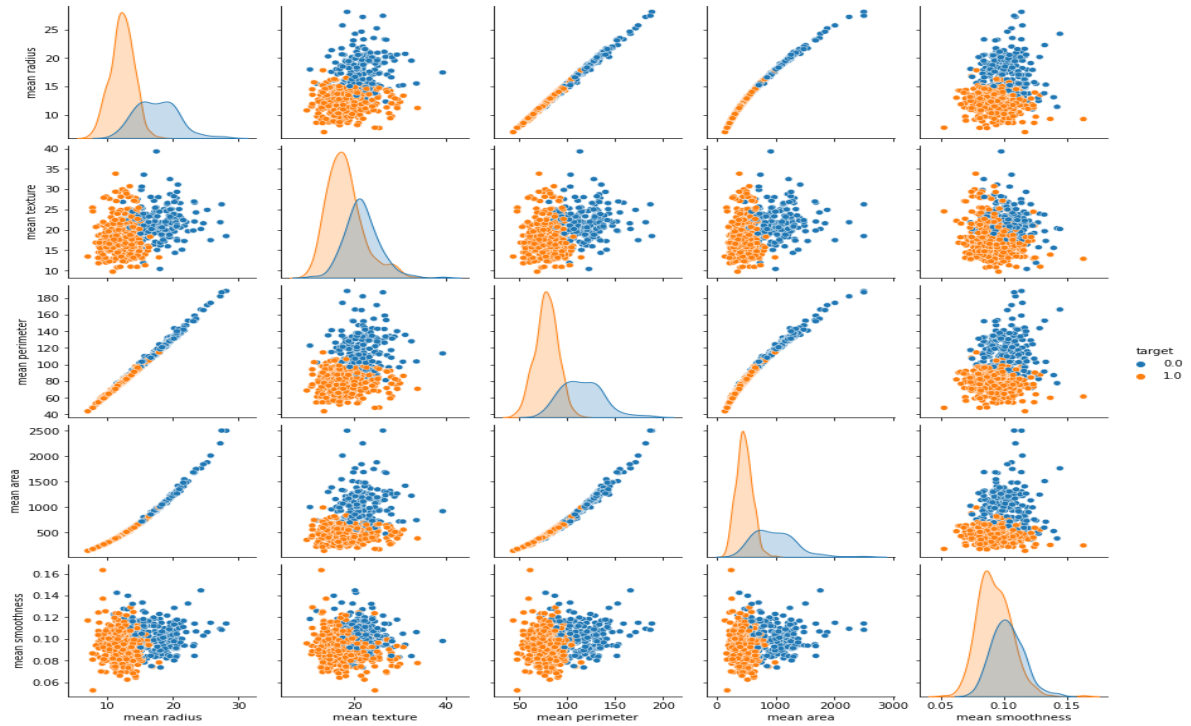
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smoothness', 'mean compactness', 'mean concavity', 'mean concave points', 'mean symmetry', 'mean fractal dimension', 'radius error', 'texture error', 'perimeter error', 'area error', 'smoothness error', 'compactness error', 'concavity error', 'concave points error', 'symmetry error', 'fractal dimension error', 'worst radius', 'worst texture', 'worst perimeter', 'worst area', 'worst smoothness', 'worst compactness', 'worst concavity', 'worst concave points', 'worst symmetry', 'worst fractal dimension', 'target'], dtype='object')

```
# Let's plot out just the first 5 variables (features)
sns.pairplot(df_cancer, vars = ['mean radius', 'mean texture', 'mean perimeter', 'mean
area', 'mean smoothness'] )
```
o/p:



# Let's plot out just the first 5 variables (features)
sns.pairplot(df_cancer, hue = 'target', vars = ['mean radius', 'mean texture', 'mean perimeter','mean area','mean smoothness'])

o/p:



df_cancer['target'].value_counts()

o/p:

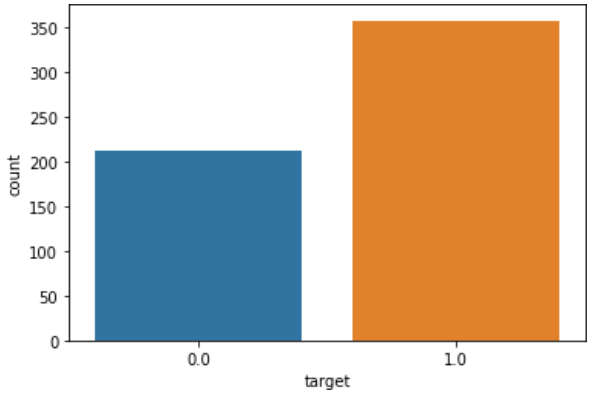
1.0 357

0.0 212

Name: target, dtype: int64

sns.countplot(df_cancer['target'], label = "Count")

o/p:



X = df_cancer.drop(['target'], axis = 1) # We drop our "target" feature and use all the remaining features in our dataframe to train the model.
y = df_cancer['target']
y.head()

o/p:

0 0.0

1 0.0

2 0.0

3 0.0

4 0.0

Name: target, dtype: float64

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 20)
print ('The size of our training "X" (input features) is', X_train.shape)
print ('\n')
print ('The size of our testing "X" (input features) is', X_test.shape)
print ('\n')
print ('The size of our training "y" (output feature) is', y_train.shape)
print ('\n')

print ('The size of our testing "y" (output features) is', y_test.shape)

o/p:

The size of our training "X" (input features) is (455, 30)

The size of our testing "X" (input features) is (114, 30)

The size of our training "y" (output feature) is (455,)

The size of our testing "y" (output features) is (114,)

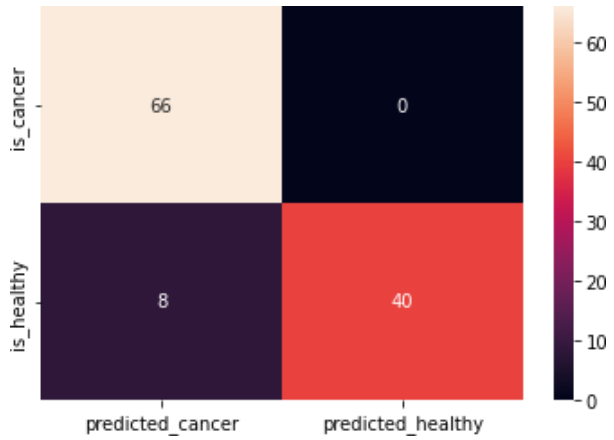
```
from sklearn.svm import SVC
svc_model = SVC()
svc_model.fit(X_train, y_train)
y_predict = svc_model.predict(X_test)
# Import metric libraries

from sklearn.metrics import classification_report, confusion_matrix
cm = np.array(confusion_matrix(y_test, y_predict, labels=[1,0]))
confusion = pd.DataFrame(cm, index=['is_cancer', 'is_healthy'],
                         columns=['predicted_cancer','predicted_healthy'])
confusion
```

o/p:

	predicted_cancer	predicted_healthy
is_cancer	66	0
is_healthy	8	40

```
sns.heatmap(confusion, annot=True)
```

o/p:



print(classification_report(y_test, y_predict))

o/p:

	precision	recall	f1-score	support
0.0	1.00	0.83	0.91	48
1.0	0.89	1.00	0.94	66
accuracy			0.93	114
macro avg	0.95	0.92	0.93	114
weighted avg	0.94	0.93	0.93	114

4.6 Results and discussion:

- ✓ Required python libraries are imported.
- ✓ Data is imported.
- ✓ After exploring the data, it is observed that dataset has 596 rows or instances and 31 columns or features.
- ✓ Data frame is visualized (first 5 features)
- ✓ Plots shows the relationship between the dataset features. But the only problem with them is that they do not show us which of the "dots" is Malignant and which is Benign.
- ✓ This issue will be addressed below by using "target" variable as the "hue" for the plots.
- ✓ It is observed that there are total of 357 Benign (NO cancer and 212 malignant (Cancer) tumours.
- ✓ Total counts are target is visualized for better understanding.
- ✓ Data is spilt into in to y and X, where “y” is the target and “X” is the predictors.
- ✓ Further, data is split for training and testing the SVM.
- ✓ Training data is fit into Support Vector Classifier.
- ✓ Prediction is made for above SVC.
- ✓ Confusion matrix is plotted for better understanding the accuracy of the SVM.
- ✓ Upon analysing the heat map, its sis observed that accuracy of the SVM is 62%.

Hence, an SVM based classifier is trained to predict whether the cancer is malignant or benign.