ASSIGNMENT - 1

Title:- **Data Wrangling- I**

Problem Statement:-

Data Wrangling-I
Perform the following operations using Python on any open-source dataset (e.g., data.csv)
1. Import all the required Python Libraries.
2. Locate an open source data from the web (e.g., https://www.kaggle.com). Provide a clear

description of the data and its source (i.e., URL of the web site).
3. Load the Dataset into pandas dataframe.
4. Data Pre-processing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.
5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.
6. Turn categorical variables into quantitative variables in Python.

In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.

Objectives: -

Learn How to do data wrangling and pre-processing.

Theory: -

Data Wrangling is the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging.

Importance Of Data Wrangling

Data Wrangling is a very important step. The below example will explain its importance as:

Books selling Website want to show top-selling books of different domains, according to user preference. For example, a new user search for motivational books, then they want to show those motivational books which sell the most or having a high rating, etc.

Data Wrangling in Python

Data Wrangling is a crucial topic for Data Science and Data Analysis. Pandas Framework of Python is used for Data Wrangling. Pandas is an open-source library specifically developed for Data Analysis and Data Science. The process like data sorting or filtration, Data grouping, etc.

Data wrangling in python deals with the below functionalities:

1. **Data exploration:** In this process, the data is studied, analyzed and understood by visualizing representations of data.
2. **Dealing with missing values:** Most of the datasets having a vast amount of data contain missing values of NaN, they are needed to be taken care of by replacing them with mean, mode, the most frequent value of the column or simply by dropping the row having a NaN value.
3. **Reshaping data:** In this process, data is manipulated according to the requirements, where new data can be added, or pre-existing data can be modified.
4. **Filtering data:** Sometimes datasets are comprised of unwanted rows or columns which are required to be removed or filtered
5. **Other:** After dealing with the raw dataset with the above functionalities we get an efficient dataset as per our requirements and then it can be used for a required purpose like data analyzing, machine learning, data visualization, model training etc.

Python Function to import pandas is

```
# Import pandas package
import pandas as pd

#Reading csv to dataframe
import pandas as pd  #importing the module

dataFrame=pd.read_csv('data.csv')
print(dataFrame)
```

1. Finding Missing Values
We can find the missing values using isnull() function.
Example of finding missing values:

```
dataFrame.isnull()
```

2. Removing Missing Values

Example of dropping missing values:

```
dataFrame.dropna()
```

3. Replacing with a value

Example of replacing missing values with a constant value:

```
dataFrame.fillna('')
```

Filling in missing data with the fillna method is a special case of more general value replacement.

Replace NULL values with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)

data.replace(-999, np.nan)
```

Replace Using Mean, Median, or Mode

A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

Pandas uses the mean() median() and mode() methods to calculate the respective values for a specified column:

Calculate the MEAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mean()

df["Calories"].fillna(x, inplace = True)
```

Reshaping Data

Before doing this, first let us add a column named ‘Gender’.
Example of adding a column:

```
dataFrame['Gender']=['M','F','F','M','M','F']
dataFrame
```

Transforming Data Using a Function or Mapping

We can reshape the data by replacing the data by categorizing and numbering.
Example of reshaping data:

```
dataFrame['Gender'] = dataFrame['Gender'].map({'M': 0, 'F': 1, }).astype(float)

dataFrame
```

Discovering Duplicates

```
print(df.duplicated())
```

Wrangling data by removing Duplication

dataFrame.drop_duplicates(inplace=True**)**

The describe () method returns description of the data in the DataFrame.

If the DataFrame contains numerical data, the description contains this information for each column:

count - The number of not-empty values.
mean - The average (mean) value.
std - The standard deviation.
min - the minimum value.
25% - The 25% percentile*.
50% - The 50% percentile*.
75% - The 75% percentile*.
max - the maximum value.

Syntax

dataframe.describe(percentiles, include, exclude, datetime_is_numeric)

data.describe()

To see what all the data types are in a dataframe, use df.dtypes

```
df.dtypes
df.info()
```

Using the astype() function

The simplest way to convert a pandas column of data to a different type is to use astype() . For instance, to convert the Customer Number to an integer we can call it like this:

```
df['Customer Number'].astype('int')
```

To converting the separate month, day and year columns into a datetime

```
pd.to_datetime(df[['Month', 'Day', 'Year']])
```

Pandas .size, .shape **and** .ndim **are used to return size, shape and dimensions of data frames and series.**

# making data frame

```
data = pd.read_csv("nba.csv")
  # dataframe.size
size = data.size
  # dataframe.shape
shape = data.shape
  # dataframe.ndim
df_ndim = data.ndim
```

Scaling And Normalization

The values of every feature in a data point can vary between random values. So, it is important to scale them so that this matches specified rules.

Standard Scaling

Several machine learning algorithms, like linear regression support vector machines (SVMs) assume all the features in a dataset are centered around 0 and have unit variances. It's a common practice to apply standard scaling to your data before training these machine learning algorithms on your dataset.

In standard scaling, a feature is scaled by subtracting the mean from all the data points and dividing the resultant values by the standard deviation of the data.

Mathematically, this is written as:

```
scaled = (x-u)/s
```

Here, `u` refers to the mean value and `s` corresponds to the standard deviation.
To apply standard scaling with Python, you can use the `StandardScaler` class from the **sklearn.preprocessing** module. You need to call the `fit_transform()` method from the `StandardScaler` class and pass it your Pandas Dataframe containing the features you want scaled. Here's an example using the tips_ds_numeric dataset we made earlier.

```
from sklearn.preprocessing import StandardScaler


ss = StandardScaler()
```

```
tips_ds_scaled = ss.fit_transform(tips_ds_numeric)
```

The fit_transform() method returns a NumPy array which you can convert to a Pandas Dataframe by passing the array to the Dataframe class constructor.

Min/Max scaling normalizes the data between 0 and 1 by subtracting the overall minimum value from each data point and dividing the result by the difference between the minimum and maximum values.
The Min/Max scaler is commonly used for data scaling when the maximum and minimum values for data points are known. For instance, you can use the min/max scaler to normalize image pixels having values between 0 and 255.

You can use the following code for scaling –

```
from sklearn.preprocessing import MinMaxScaler
```

```
data_scaler = MinMaxScaler(feature_range = (0, 1))
data_scaled = data_scaler.fit_transform(input_data)
print "\nMin max scaled data = ", data_scaled
```

Maximum Absolute Scaling

Maximum absolute scaling is another commonly used data scaling technique where the difference between the data points and the minimum value is divided by the maximum value. The maximum absolute scaling technique also normalizes the data between 0 and 1. Maximum absolute scaling doesn't shift or center the data so it's commonly used for scaling sparse datasets.

```
from sklearn.preprocessing import MaxAbsScaler


mas = MaxAbsScaler()

tips_ds_mas = mas.fit_transform(tips_ds_numeric)
```

Median and Quantile Scaling

In median and quantile scaling, also known as robust scaling, the first step is to subtract the median value from all the data points. In the next step, the resultant values are divided by the IQR (interquartile range). The IQR is calculated by subtracting the first quartile values in your dataset from the third quartile values.

Median and quantile scaling can be implemented via the RobustScaler class from the **sklearn.preprocessing** module. As with the other data scaling classes from the scikit-learn library, you need to call the fit_transform() method and pass it the input dataset, as shown in the script below:

```
from sklearn.preprocessing import RobustScaler


rs = RobustScaler()

tips_ds_rs = rs.fit_transform(tips_ds_numeric)
```

Normalization

Normalization involves adjusting the values in the feature vector so as to measure them on a common scale. Here, the values of a feature vector are adjusted so that they sum up to 1. We add the following lines to the prefoo.py file −

You can use the following code for normalization −

```
data_normalized = preprocessing.normalize(input_data, norm  = 'l1')
print "\nL1 normalized data = ", data_normalized
```

Normalization is used to ensure that data points do not get boosted due to the nature of their features.

Binarization

Binarization is used to convert a numerical feature vector into a Boolean vector. You can use the following code for binarization −

```
data_binarized = preprocessing.Binarizer(threshold=1.4).transform(input_data)
print "\nBinarized data =", data_binarized
```

This technique is helpful when we have prior knowledge of the data.

Handling Categorical Features

This process of converting categories into numbers is called encoding. Two of the most effective and widely used encoding methods are:

1. Label Encoding   2. One Hot Encoding

1.Label Encoding

Label encoding is the process of assigning numeric label to each category in the feature. If N is the number of categories, all the category values will be assigned a unique number from 0 to N-1.

```
#importing the libraries
import pandas as pd
import numpy as np

#reading the dataset
df=pd.read_csv("Salary.csv")

# Import label encoder
from sklearn import preprocessing
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'Country'.
data['Country']= label_encoder.fit_transform(data['Country'])
print(data.head())
```

```
Category : Label
"red" : 0
"blue" : 1
"green" : 2
"yellow" : 3
```

Note: As we can see here, the labels produced for the categories are not normalized, i.e. not between 0 and 1. Because of this limitation, label encoding should not be used with linear models where magnitude of features plays an important role. Since tree based algorithms do not need feature normalization, label encoding can be easily used with these models such as :

- Decision trees , Random forest etc

2.One Hot Encoding

The limitation of label encoding can be overcome by binarizing the categories, i.e. representing those using only 0's and 1's. Here we represent each category by a vector of size N, where N is the number of categories in that feature. Each vector has one 1 and rest all values are 0. Hence it is called one-hot encoding.

```
# Importing one hot encoder
from sklearn from sklearn.preprocessing import OneHotEncoder
# Creating one hot encoder object
onehotencoder = OneHotEncoder()
#reshape the 1-D country array to 2-D as fit_transform expects 2-D and finally fit the object
X = onehotencoder.fit_transform(data. Country.values.reshape(-1,1)).toarray()
#To add this back into the original dataframe
dfOneHot = pd.DataFrame(X, columns = ["Country_"+str(int(i)) for i in range(data.shape[1])])
df = pd.concat([data, dfOneHot], axis=1)
#Droping the country column
```

```
df= df.drop(['Country'], axis=1)
#Printing to verify
print(df.head())
```

```
Category Encoded vector
Freezing 0 0 0 1
Cold 0 0 1 0
Warm 0 1 0 0
Hot 1 0 0 0
```

Challenges of One-Hot Encoding: Dummy Variable Trap
One-Hot Encoding results in a Dummy Variable Trap as the outcome of one variable can easily be predicted with the help of the remaining variables.

To overcome the problem of multicollinearity, one of the dummy variables has to be dropped.

When to use a Label Encoding vs. One Hot Encoding
This question generally depends on your dataset and the model which you wish to apply. But still, a few points to note before choosing the right encoding technique for your model:

We apply One-Hot Encoding when:

1. The categorical feature is **not ordinal** (like the countries above)
2. The number of categorical features is less so one-hot encoding can be effectively applied

We apply Label Encoding when:

1. The categorical feature is **ordinal** (like Jr. kg, Sr. kg, Primary school, high school)
2. The number of categories is quite large as one-hot encoding can lead to high memory consumption

Outcomes:

After completion of this assignment students are ready with clean and processed dataset .Now they are able to apply any machine learning algorithm to analyze the data

Software Requirements:

Anaconda with Python 3.7

Conclusion:

In this way we learn data wrangling and data pre-processing

ASSIGNMENT - 5

Title: - **Data Analytics II**

Problem Statement: -

Data Analytics II
1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

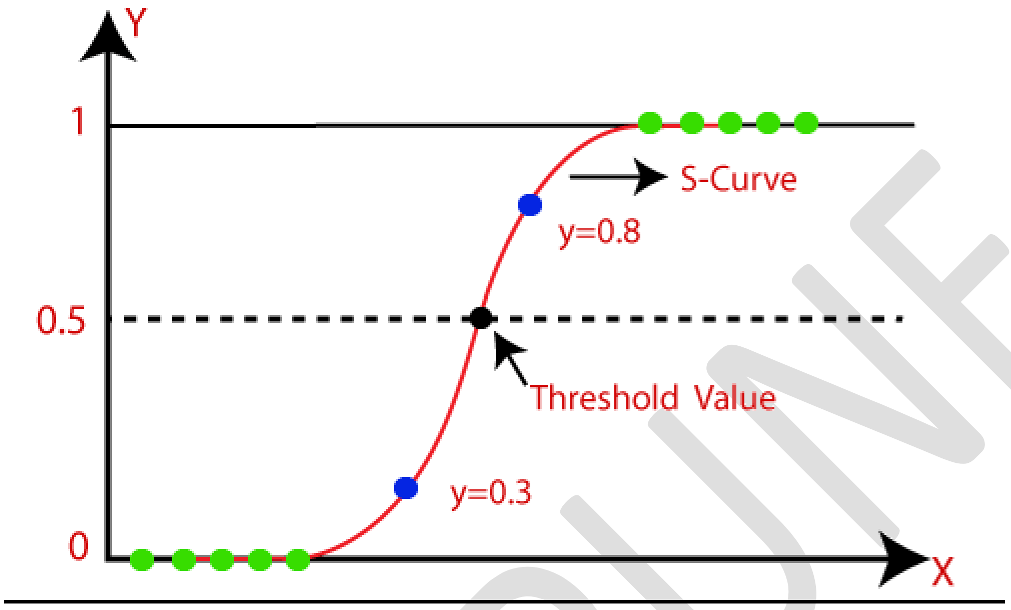
Objectives:-

Learn How to logistic regression on the given data set and predict the result, also to measure the performance of algorithm on given dataset

Theory:-

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

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

- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}\text{; 0 for y= 0, and infinity for y=1}$$

But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

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

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

- Data Pre-processing step
- Fitting Logistic Regression to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

```
#Data Pre-procesing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')
    #Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
# Splitting the dataset into training 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=0)
   #feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
#Predicting the test set result
y_pred= classifier.predict(x_test)
#Creating the Confusion matrix
```

```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix()
```

Applications of Logistic Regression

1. Predicting a probability of a person having a heart attack

2. Predicting a customer's propensity to purchase a product or halt a subscription.

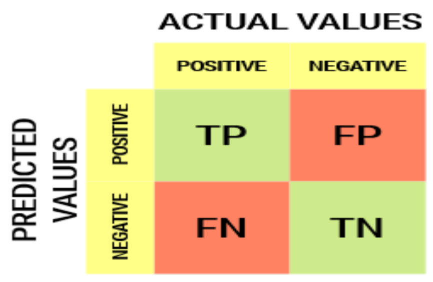
3. Predicting the probability of failure of a given process or product.

Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

What is a Confusion Matrix?

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:



- The target variable has two values: **Positive** or **Negative**
- The **columns** represent the **actual values** of the target variable

- The **rows** represent the **predicted values** of the target variable

True Positive (TP)

- The predicted value matches the actual value
- The actual value was positive and the model predicted a positive value

True Negative (TN)

- The predicted value matches the actual value
- The actual value was negative and the model predicted a negative value

False Positive (FP) – Type 1 error

- The predicted value was falsely predicted
- The actual value was negative but the model predicted a positive value
- Also known as the **Type 1 error**

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted
- The actual value was positive but the model predicted a negative value
- Also known as the **Type 2 error**

Precision vs. Recall

Precision tells us how many of the correctly predicted cases actually turned out to be positive. This would determine whether our model is reliable or not. Here's how to calculate Precision:

Precision is a useful metric in cases where False Positive is a higher concern than False Negatives.

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

Recall tells us how many of the actual positive cases we were able to predict correctly with our model.And here's how we can calculate Recall:

Recall is a useful metric in cases where False Negative trumps False Positive.

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

Confusion Matrix using scikit-learn in Python

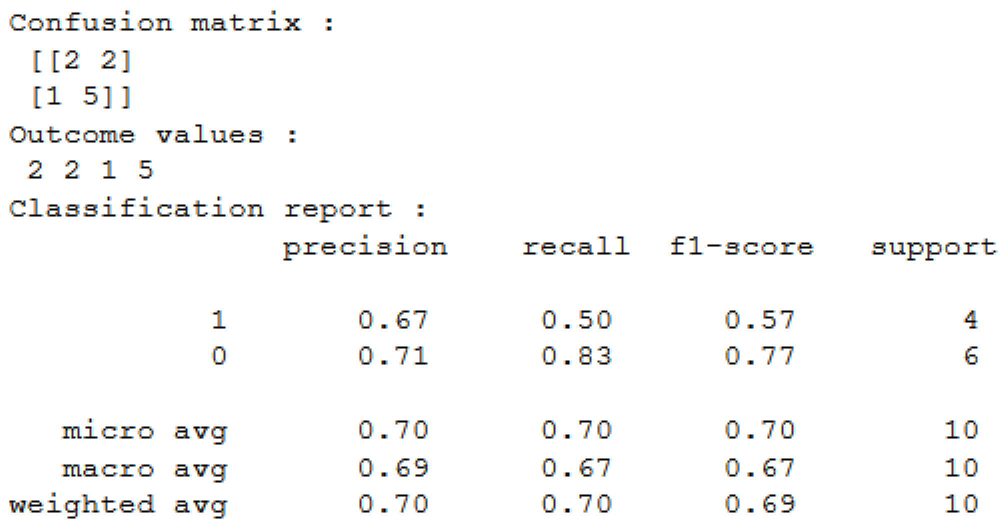
```
# 
confusion
matrix in
sklearn
            from sklearn.metrics import confusion_matrix
            from sklearn.metrics import classification_report

            # actual values
            actual = [1,0,0,1,0,0,1,0,0,1]
            # predicted values
            predicted = [1,0,0,1,0,0,0,1,0,0]

            # confusion matrix
            matrix = confusion_matrix(actual,predicted, labels=[1,0])
            print('Confusion matrix : \n',matrix)

            # outcome values order in sklearn
            tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
            print('Outcome values : \n', tp, fn, fp, tn)

            # classification report for precision, recall f1-score and accuracy
            matrix = classification_report(actual,predicted,labels=[1,0])
            print('Classification report : \n',matrix)
```



```
Confusion matrix :
 [[2 2]
 [1 5]]
Outcome values :
 2 2 1 5
Classification report :
               precision    recall  f1-score   support

           1       0.67      0.50      0.57         4
           0       0.71      0.83      0.77         6

   micro avg       0.70      0.70      0.70        10
   macro avg       0.69      0.67      0.67        10
weighted avg       0.70      0.70      0.69        10
```

<u>Acuuracy in Logistic Regression</u>

Accuracy is the proportion of correct predictions over total predictions. This is how we can find the accuracy with logistic regression:

accuracy = correct_predictions / total_predictions

Using Python

```
from sklearn.metrics import accuracy_score
x= accuracy_score(y_train, y_pred)
```

<u>Software Requirements:</u>

Anaconda with Python 3.7

<u>Conclusion</u>:

After completion of this assignment students are able Implement code for the logistic Regression and evaluate the performance and Accuracy of the model using confusion matrix.

ASSIGNMENT - 6

Title: - **Data Analytics III**

Problem Statement: -

1.Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset.

2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

Objectives: -

Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset and Compute Confusion matrix ,error, Precision and recall on the dataset

Theory:-

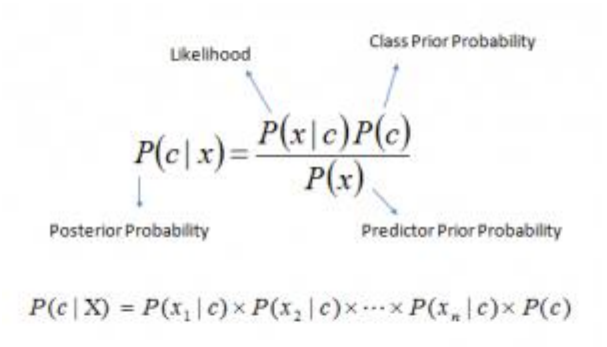
What is Naive Bayes algorithm?

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c\mid x)=\frac{P(x\mid c)P(c)}{P(x)}$$

$$P(c\mid X)=P(x_1\mid c)\times P(x_2\mid c)\times\cdots\times P(x_n\mid c)\times P(c)$$

Above,

- *P*(*c*|*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

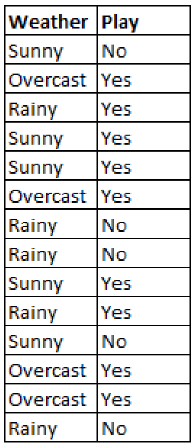
P(B) is Marginal Probability: Probability of Evidence.

How Naive Bayes algorithm works?

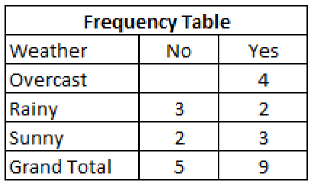
Let's understand it using an example. Consider a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

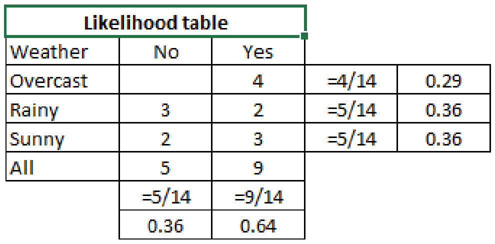
Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.



Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No



Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9



Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Problem: Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

What are the Pros and Cons of Naive Bayes?

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”. To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.
- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Python Implementation of the Naïve Bayes algorithm:

Now we will implement a Naive Bayes Algorithm using Python. So for this, we will use the "**user_data**" **dataset**, which we have used in our other classification model. Therefore we can easily compare the Naive Bayes model with the other models.

Steps to implement:

- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)

1) Data Pre-processing step:

1. Importing the libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd

```

# Importing the dataset
dataset = pd.read_csv('user_data.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# 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 = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

2) Fitting Naive Bayes to the Training Set:

```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
```

3) Prediction of the test set result:

```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

4) Creating Confusion Matrix:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Applications of Naive Bayes Algorithms

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely

used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- **Recommendation System:** Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

Outcomes:

Iris dataset is Classified using Naïve Bayes algorithm and model is used to predict the type of Iris flower based on sepal length, sepal width ,petal length and petal width

Software Requirements:

Anaconda with Python 3.7

Conclusion:

After completion of this assignment students are able Classify, dataset using Naïve Bayes algorithm and evaluate the performance and Accuracy of the model using confusion matrix.

ASSIGNMENT - 8

Title: - **Data Visualization I**

Problem Statement: -

1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data.
2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.

Objectives:-

Learn How to Visualize the given data points(Use Titanic data set)
Use Seaborn library to find any pattern in the data

Theory:-

Exploratory Data Analysis (**EDA**) is an approach to analyzing datasets to summarize their main characteristics. It is used to understand data, get some context regarding it, understand the variables and the relationships between them, and formulate hypotheses that could be useful when building predictive models.

we will learn how to perform EDA using data visualization. we will focus on seaborn, a Python library that is built on top of matplotlib and has support for NumPy and pandas.

seaborn allows us to make attractive and informative statistical graphics. Although matplotlib makes it possible to visualize essentially anything, it is often difficult and tedious to make the plots visually attractive. seaborn is often used to make default matplotlib plots look nicer, and introduces some additional plot types.

We will cover how to visually analyze:

- Numerical variables with histograms,
- Categorical variables with count plots,
- Relationships between numerical variables with scatter plots, joint plots, and pair plots, and
- Relationships between numerical and categorical variables with box-and-whisker plots and complex conditional plots.

By effectively visualizing a dataset's variables and their relationships, a data analyst or data scientist is able to quickly understand trends, outliers, and patterns. This understanding can then be used to tell a story, drive decisions, and create predictive models.

Seaborn: statistical data visualization

Seaborn helps to visualize the statistical relationships, To understand how variables in a dataset are related to one another and how that relationship is dependent on other variables, we perform statistical analysis. This Statistical analysis helps to visualize the trends and identify various patterns in the dataset.

These are the plot will help to visualize:

- Line Plot
- Scatter Plot
- Box plot
- Point plot
- Count plot
- Violin plot
- Swarm plot
- Bar plot
- KDE Plot

Downloading the Seaborn Library

The seaborn library can be downloaded in a couple of ways. If you are using pip installer for Python libraries, you can execute the following command to download the library:

```
pip install seaborn
```

Alternatively, if you are using the Anaconda distribution of Python, you can use execute the following command to download the seaborn library:

```
conda install seaborn
```

The Dataset

The dataset that we are going to use to draw our plots will be the Titanic dataset, which is downloaded by default with the Seaborn library. All you have to do is use the load_dataset function and pass it the name of the dataset.

Let's see what the Titanic dataset looks like. Execute the following script:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

dataset = sns.load_dataset('titanic')

dataset.head()
```

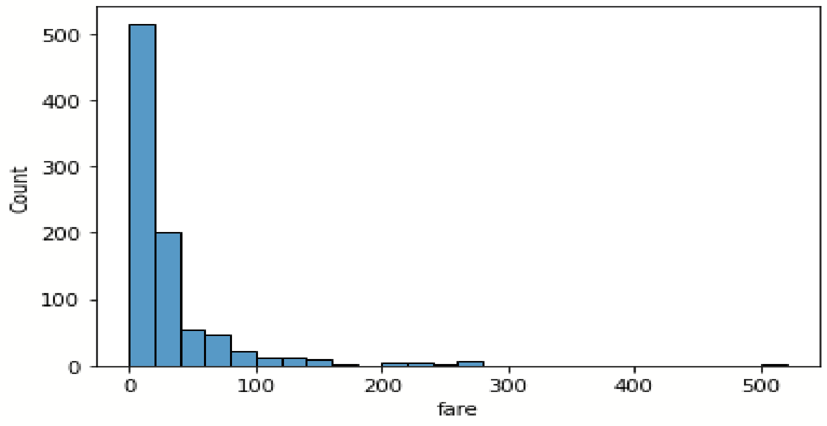
The dataset contains 891 rows and 15 columns and contains information about the passengers who boarded the unfortunate Titanic ship. The original task is to predict whether or not the passenger survived depending upon different features such as their age, ticket, cabin they boarded, the class of the ticket, etc. We will use the Seaborn library to see if we can find any patterns in the data.

Histograms are visualization tools that represent the distribution of a set of continuous data. In a histogram, the data is divided into a set of intervals or bins (usually on the x-axis) and the count of data points that fall into each bin corresponding to the height of the bar above that bin. These bins may or may not be equal in width but are adjacent (with no gaps).

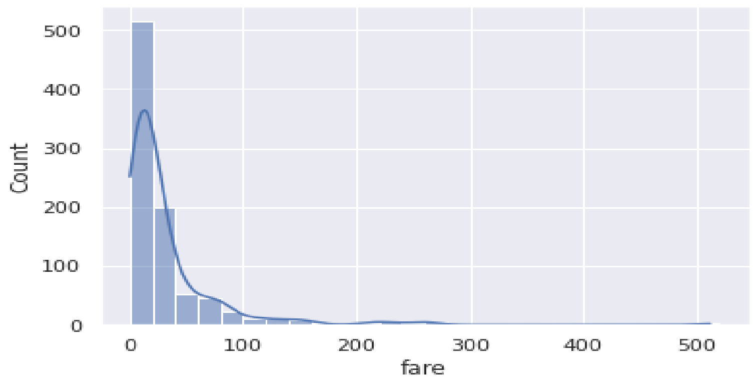
A density plot (also known as kernel density plot) is another visualization tool for evaluating data distributions. It can be considered as a smoothed histogram. The peaks of a density plot help display where values are concentrated over the interval. There are a variety of smoothing techniques. Kernel Density Estimation (KDE) is one of the techniques used to smooth a histogram.

```
seaborn.histplot(data, x, y, hue, stat, bins, binwidth, discrete, kde, log_scale)
```

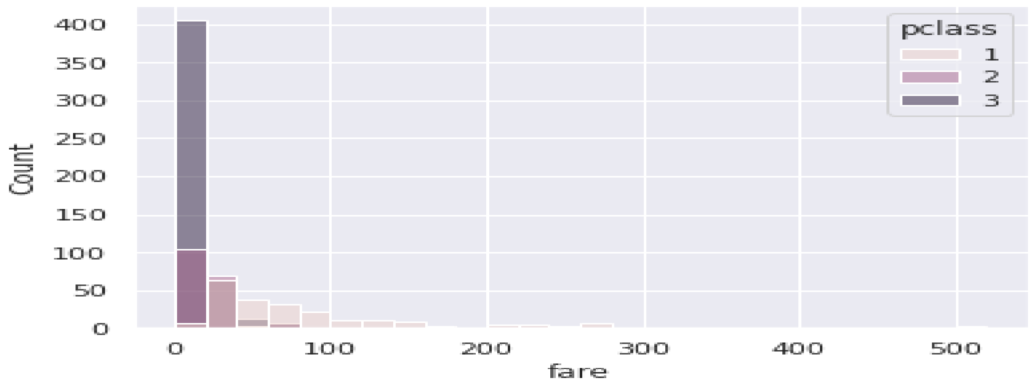
```
sns.histplot(data=df1, x="fare",binwidth=20)
plt.show()
```



```
sns.histplot(data=df1, x="fare",kde=True,binwidth=20)
```



```
sns.histplot(data=df1, x="fare",hue="pclass",binwidth=20)
plt.show()
```



Distributional Plots

Distributional plots, as the name suggests are type of plots that show the statistical distribution of data. In this section we will see some of the most commonly used distribution plots in Seaborn.

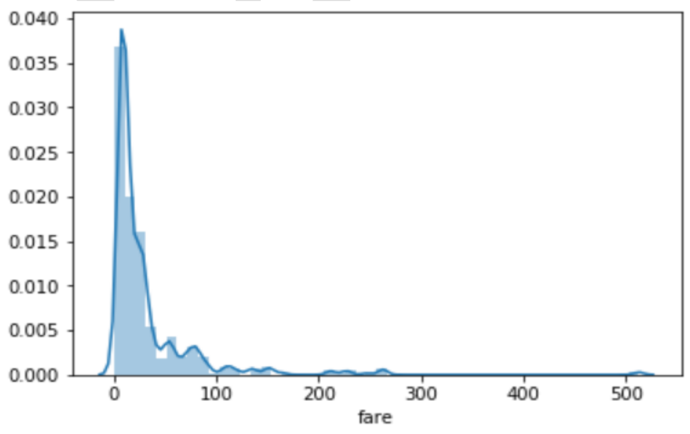
4 types of distribution plots namely:

1. joinplot
2. distplot
3. pairplot
4. rugplot

1.The Dist Plot

The distplot() shows the histogram distribution of data for a single column. The column name is passed as a parameter to the distplot() function. Let's see how the price of the ticket for each passenger is distributed. Execute the following script:

sns.distplot(dataset['fare'])



You can see that most of the tickets have been solved between 0-50 dollars. The line that you see represents the kernel density estimation. You can remove this line by passing False as the parameter for the kde attribute as shown below:

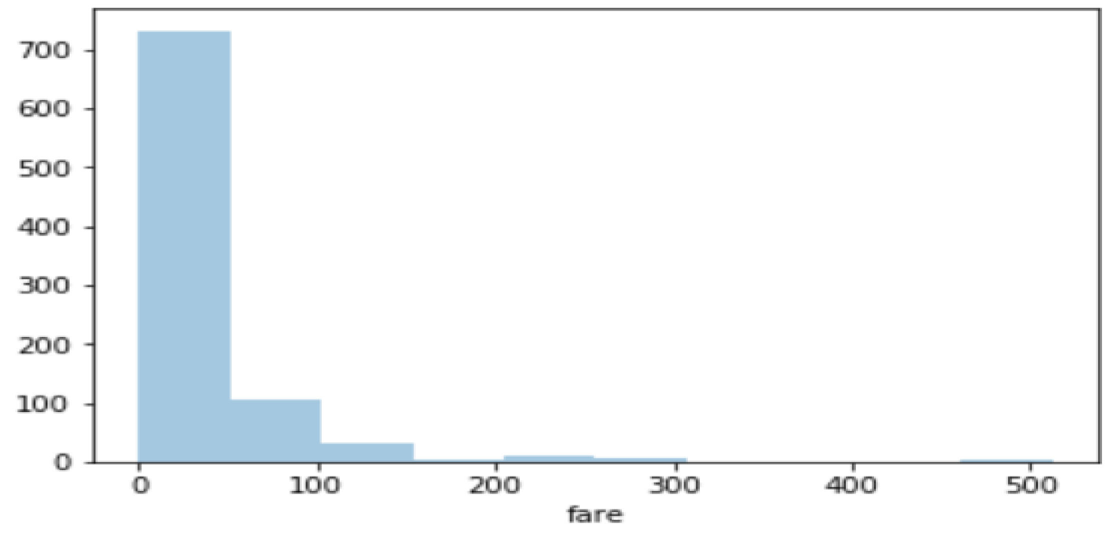
```
sns.distplot(dataset['fare'], kde=False)
```

You can also pass the value for the bins parameter in order to see more or less details in the graph. Take a look at he following script:

```
sns.distplot(dataset['fare'], kde=False, bins=10 )
```

Here we set the number of bins to 10. In the output, you will see data distributed in 10 bins as shown below:

Output:



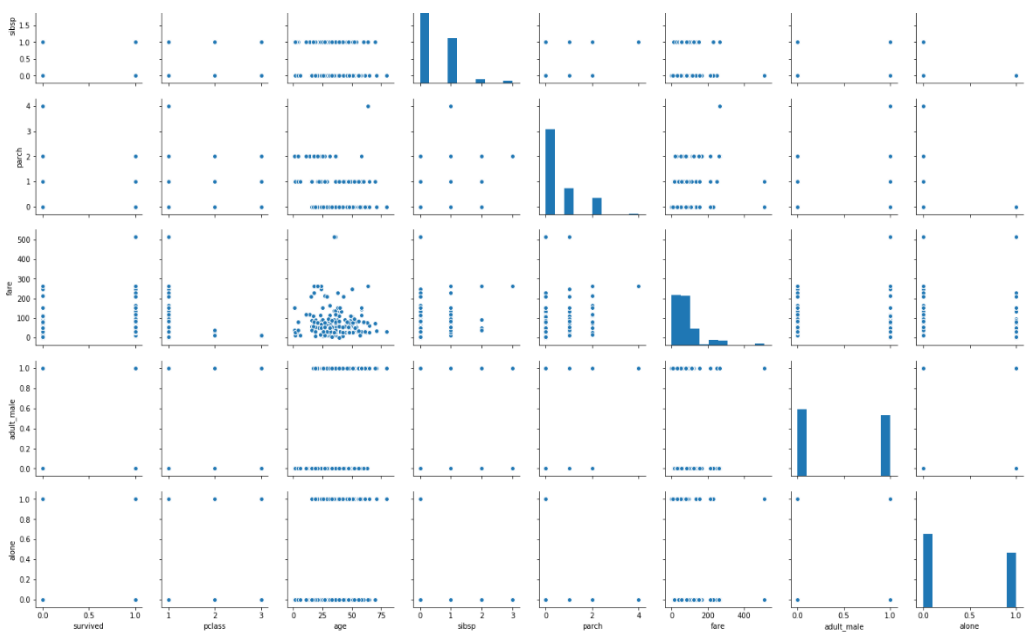
You can clearly see that for more than 700 passengers, the ticket price is between 0 and 50

The Pair Plot

The paitplot() is a type of distribution plot that basically plots a joint plot for all the possible combination of numeric and Boolean columns in your dataset. You only need to pass the name of your dataset as the parameter to the pairplot() function as shown below:

```
sns.pairplot(dataset)
```

A snapshot of the portion of the output is shown below:



Note: Before executing the script above, remove all null values from the dataset using the following command:

dataset = dataset.dropna()

To add information from the categorical column to the pair plot, you can pass the name of the categorical column to the hue parameter. For instance, if we want to plot the gender information on the pair plot, we can execute the following script:

sns.pairplot(dataset, hue='sex')

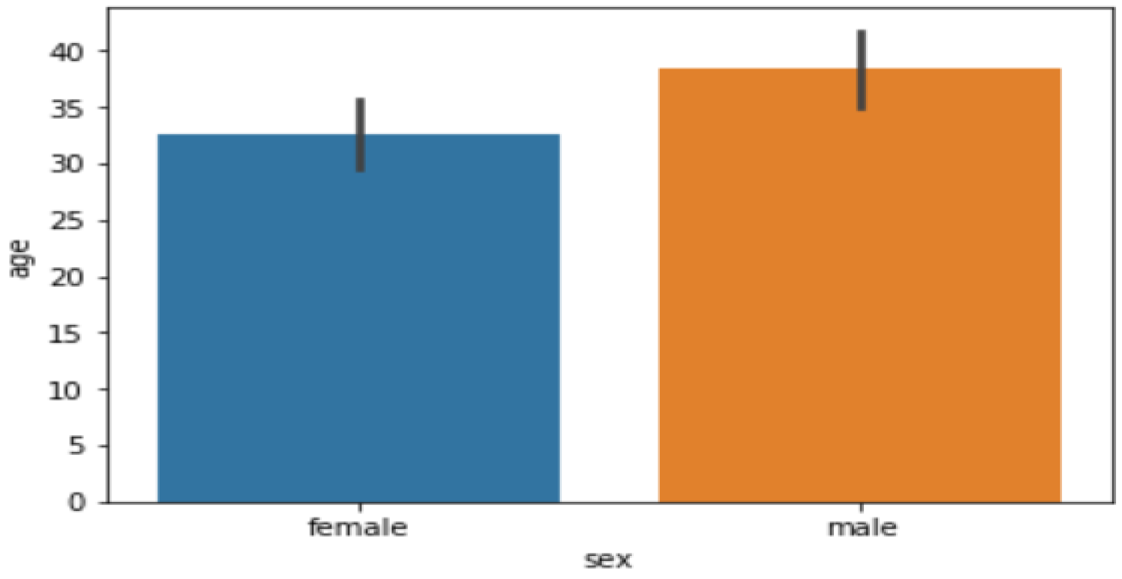
Categorical Plots

Categorical plots, as the name suggests are normally used to plot categorical data. The categorical plots plot the values in the categorical column against another categorical column or a numeric column. Let's see some of the most commonly used categorical data.

The Bar Plot

The barplot() is used to display the mean value for each value in a categorical column, against a numeric column. The first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. For instance, if you want to know the mean value of the age of the male and female passengers, you can use the bar plot as follows.

sns.barplot(x='sex', y='age', data=dataset)



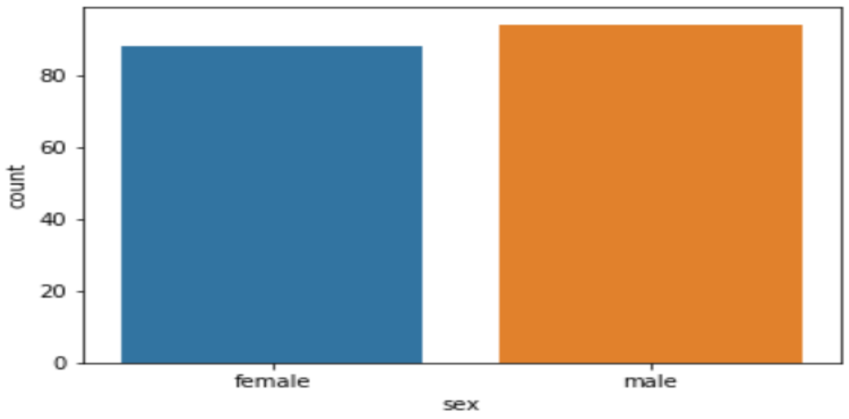
From the output, you can clearly see that the average age of male passengers is just less than 40 while the average age of female passengers is around 33.

The Count Plot

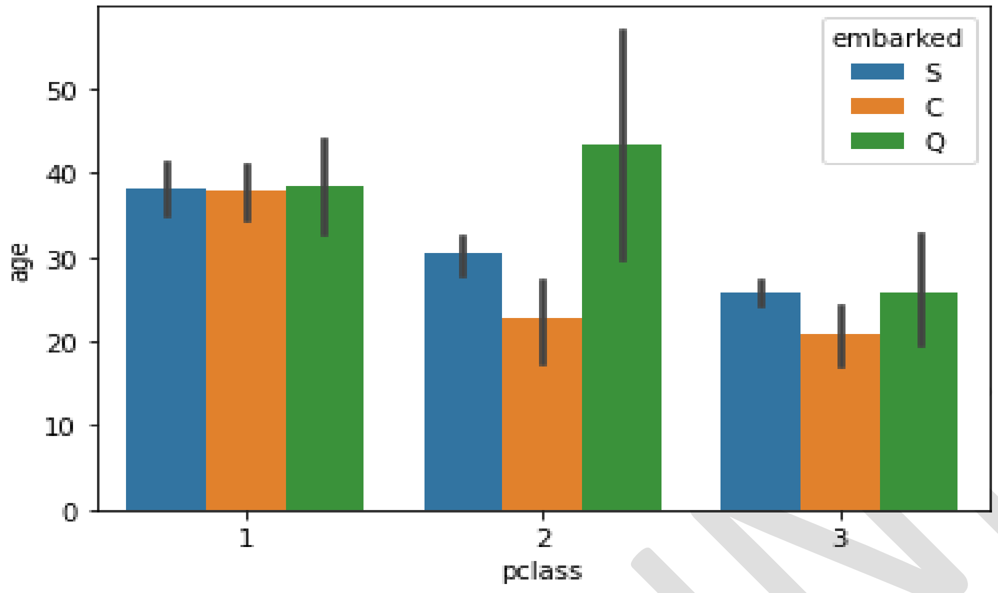
The count plot is similar to the bar plot, however it displays the count of the categories in a specific column. For instance, if we want to count the number of males and women passenger we can do so using count plot as follows:

```
sns.countplot(x='sex', data=dataset)
```

The output shows the count as follows:

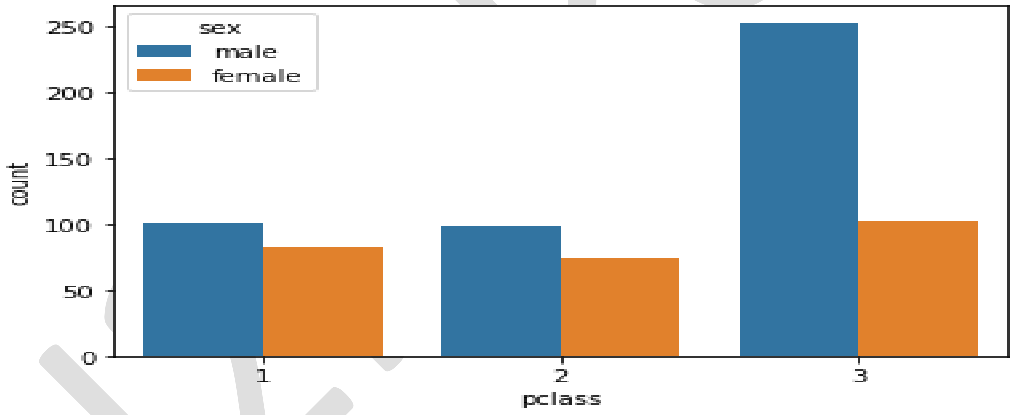


Bar plot- to visualize passenger based on class and age along with the place from which passengers embarked(Boarded in ship) *sns.barplot(x='pclass', y='age',hue='embarked', data=x1)*



Count plot used to visualize the number of passengers boarded in different class and with different genders

sns.countplot(x='pclass',hue='sex',data=x1)

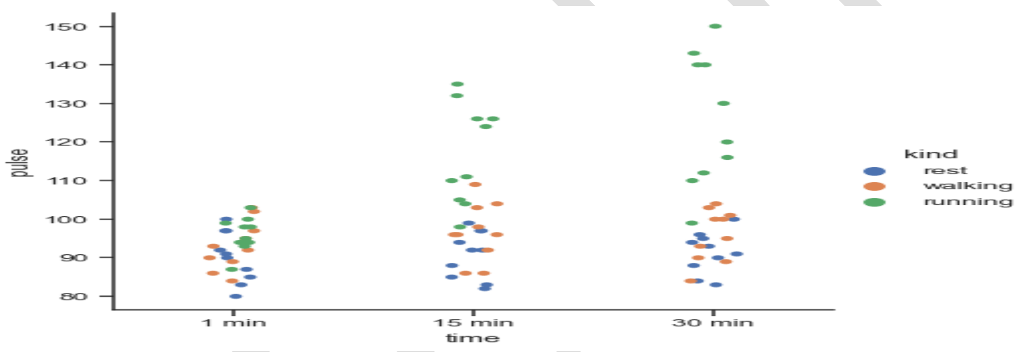


seaborn.catplot(*, x=None, y=None, hue=None, data=None, row=None, col=None, col_wrap=None, estimator=<function mean at 0x7ff320f315e0>, ci=95, n_boot=1000, units=None, seed=None, order=None, hue_order=None, row_order=None, col_order=None, kind='strip', height=5, aspect=1, orient=None, color=None, palette=None, legend=True, legend_out=True, sharex=True, sharey=True, margin_titles=False, facet_kws=None, **kwargs)

This function provides access to several axes-level functions that show the relationship between a numerical and one or more categorical variables using one of several visual representations.
The kind parameter selects the underlying axes-level function to use:

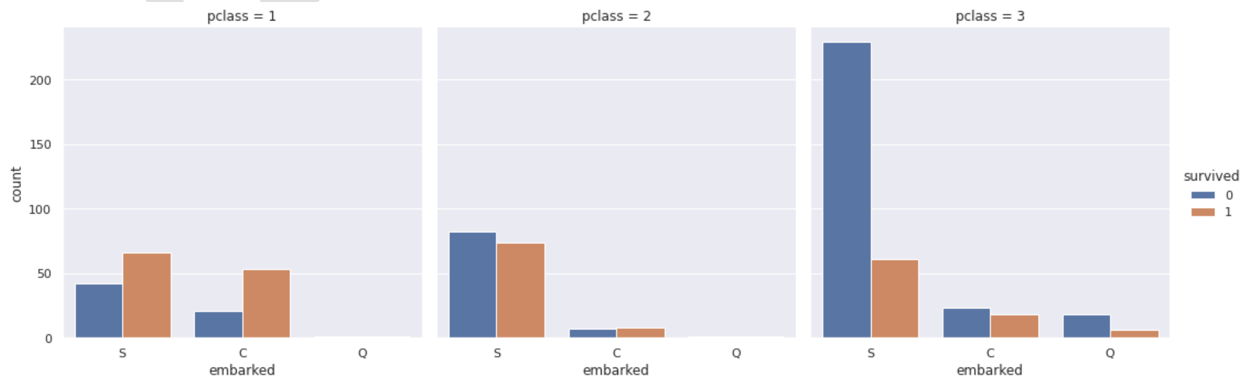
Categorical scatterplots
stripplot() (with kind="strip"; the default)
swarmplot() (with kind="swarm")
Categorical distribution plots:
boxplot() (with kind="box")
violinplot() (with kind="violin")
boxenplot() (with kind="boxen")
Categorical estimate plots:
pointplot() (with kind="point")
barplot() (with kind="bar")
countplot() (with kind="count")

```
import seaborn as sns
 sns.set_theme(style="ticks")
exercise = sns.load_dataset("exercise")
g = sns.catplot(x="time", y="pulse", hue="kind", data=exercise)
```



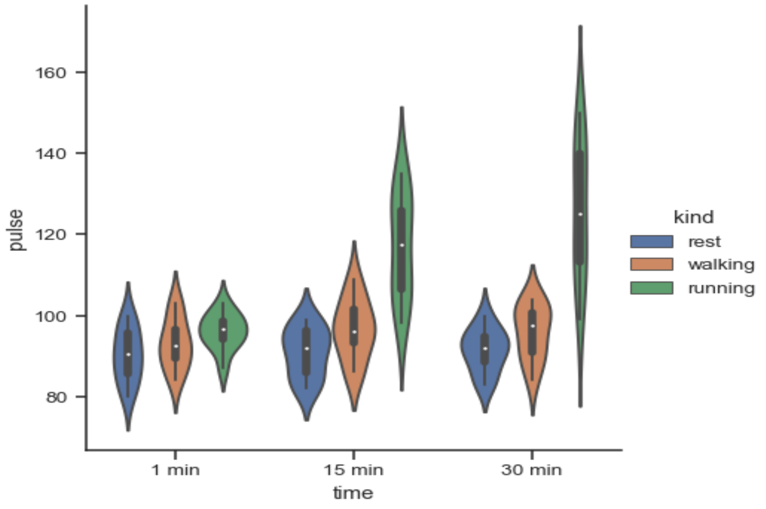
Categorical plot with kind =count can be used to plot the data of passengers based on class ,boarding place and hue with survived or not

```
sns.catplot(x ='embarked', hue ='survived',kind ='count', col ='pclass', data = x1)
```



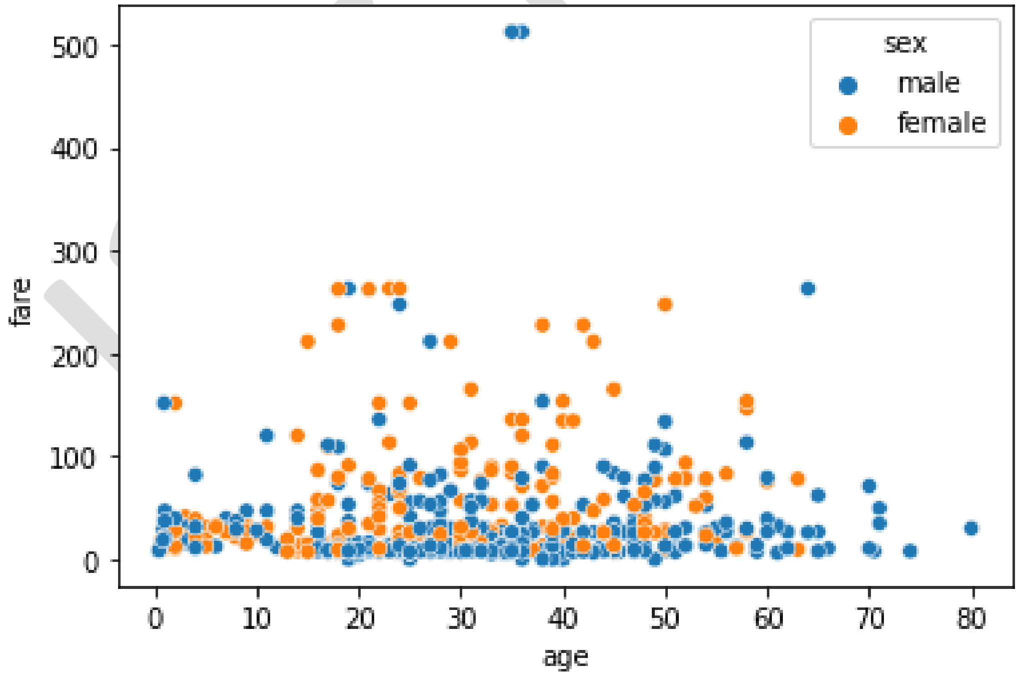
Use a different plot kind to visualize the same data:

```
g = sns.catplot(x="time", y="pulse", hue="kind",data=exercise, kind="violin")
```



Scatter Pot

sns.scatterplot(x='age',y='fare',hue='sex',data=x1)



Outcomes:

After completion of this assignment students can do EDA using visualization library called Seaborn

Software Requirements:

Anaconda with Python 3.7

Conclusion:

Students are able to find the pattern of distribution of mail female passengers based on age, survival number ,Survival based of Passenger class, place of boarding etc using Titanic dataset.

ASSIGNMENT - 9

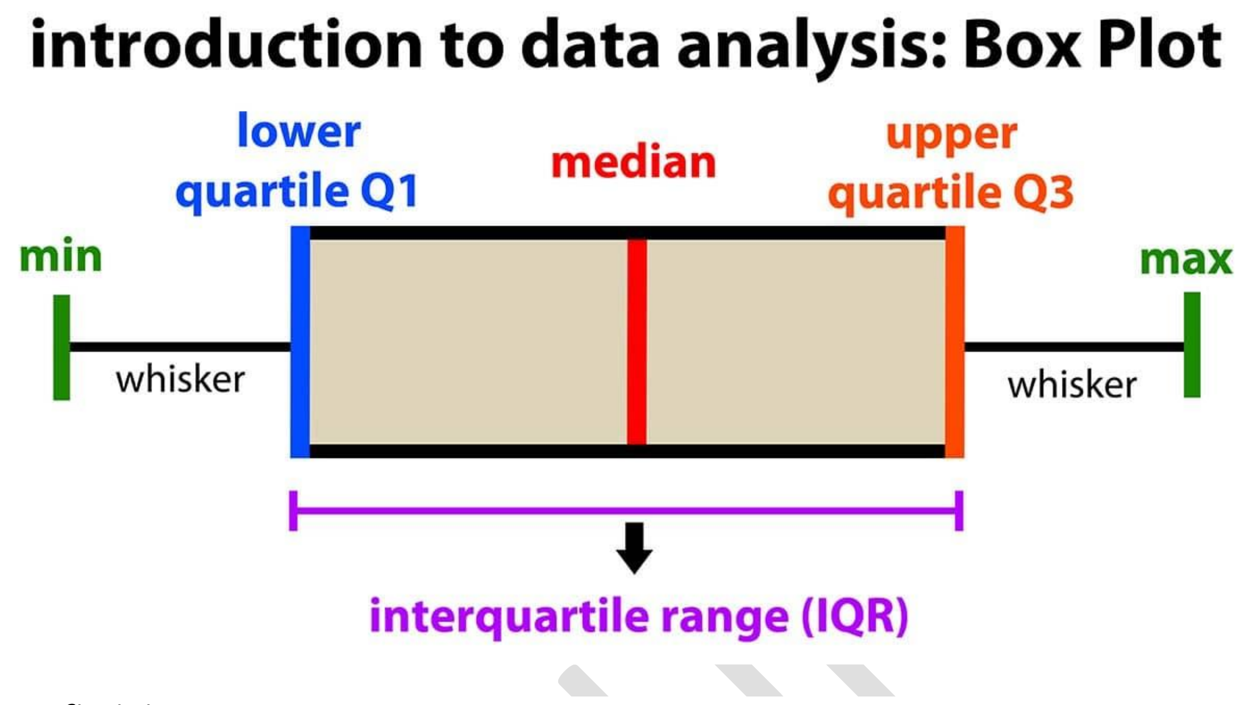
Data Visualization II
1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age')
2. Write observations on the inference from the above statistics.

Theory

What is a box plot?

In descriptive statistics, a box plot or boxplot (also known as box and whisker plot) is a type of chart often used in explanatory data analysis. Box plots visually show the distribution of numerical data and skewness through displaying the data quartiles (or percentiles) and averages.

Box plots show the five-number summary of a set of data: including the minimum score, first (lower) quartile, median, third (upper) quartile, and maximum score.


introduction to data analysis: Box Plot
lower quartile Q1
median
upper quartile Q3
min
max
whisker
whisker
interquartile range (IQR)

Definitions

Minimum Score

The lowest score, excluding outliers (shown at the end of the left whisker).

Lower Quartile

Twenty-five percent of scores fall below the lower quartile value (also known as the first quartile).

Median

The median marks the mid-point of the data and is shown by the line that divides the box into two parts (sometimes known as the second quartile). Half the scores are greater than or equal to this value and half are less.

Upper Quartile

Seventy-five percent of the scores fall below the upper quartile value (also known as the third quartile). Thus, 25% of data are above this value.

Maximum Score

The highest score, excluding outliers (shown at the end of the right whisker).

Whiskers

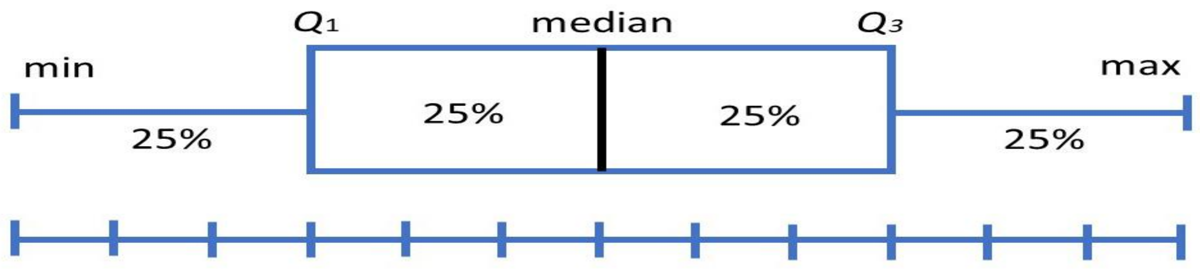
The upper and lower whiskers represent scores outside the middle 50% (i.e. the lower 25% of scores and the upper 25% of scores).

The Interquartile Range (or IQR)

This is the box plot showing the middle 50% of scores (i.e., the range between the 25th and 75th percentile).

Why are box plots useful?

Box plots divide the data into sections that each contain approximately 25% of the data in that set.



Box plots are useful as they provide a visual summary of the data enabling researchers to quickly identify mean values, the dispersion of the data set, and signs of skewness.

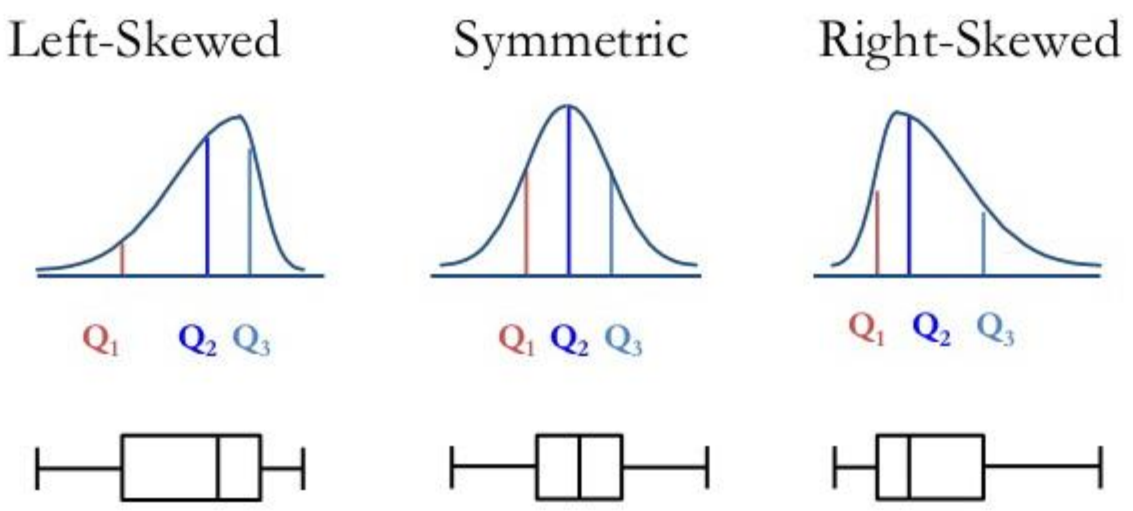
Note the image above represents data which is a perfect normal distribution and most box plots will not conform to this symmetry (where each quartile is the same length).

Box plots are useful as they show the average score of a data set.

The median is the average value from a set of data and is shown by the line that divides the box into two parts. Half the scores are greater than or equal to this value and half are less.

Box plots are useful as they show the skewness of a data set

The box plot shape will show if a statistical data set is normally distributed or skewed.



When the median is in the middle of the box, and the whiskers are about the same on both sides of the box, then the distribution is symmetric.

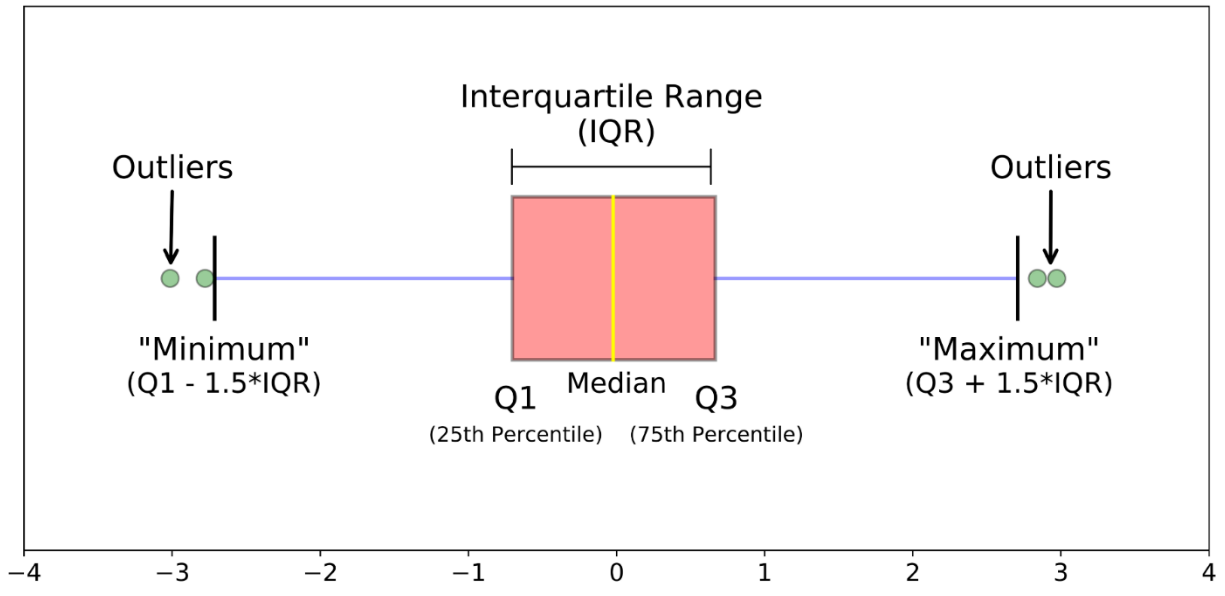
When the median is closer to the bottom of the box, and if the whisker is shorter on the lower end of the box, then the distribution is positively skewed (skewed right).

When the median is closer to the top of the box, and if the whisker is shorter on the upper end of the box, then the distribution is negatively skewed (skewed left).

Box plots are useful as they show outliers within a data set.

An outlier is an observation that is numerically distant from the rest of the data.

When reviewing a box plot, an outlier is defined as a data point that is located outside the whiskers of the box plot.



Boxplot

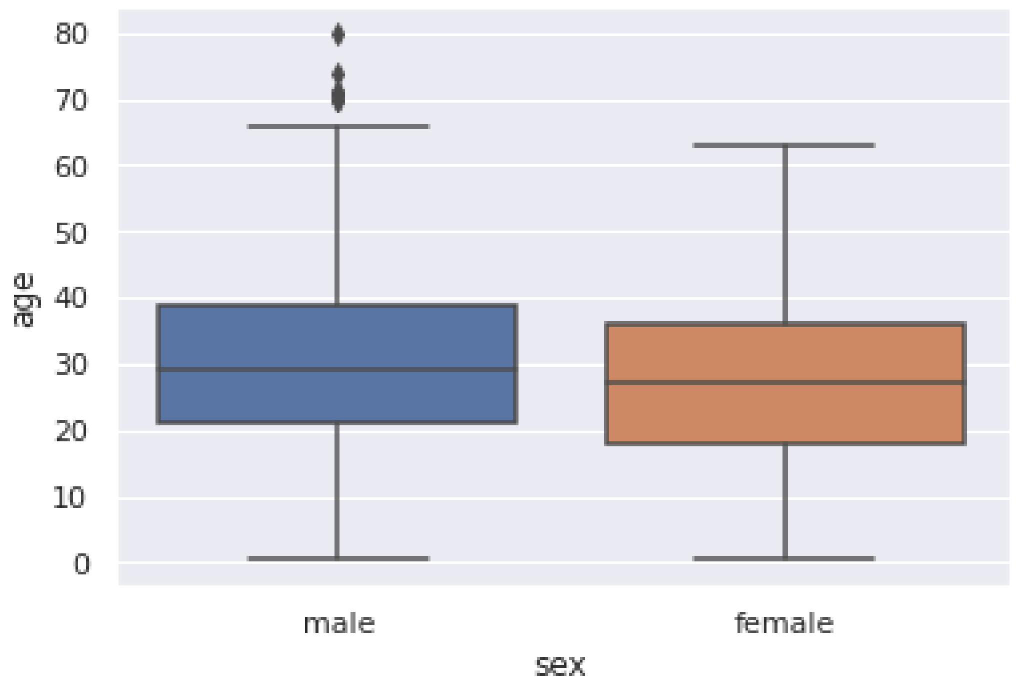
A boxplot is sometimes known as the box and whisker plot.It shows the distribution of the quantitative data that represents the comparisons between variables. boxplot shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution i.e. the dots indicating the presence of outliers.

Syntax:

```
boxplot([x, y, hue, data, order, hue_order, …])
```

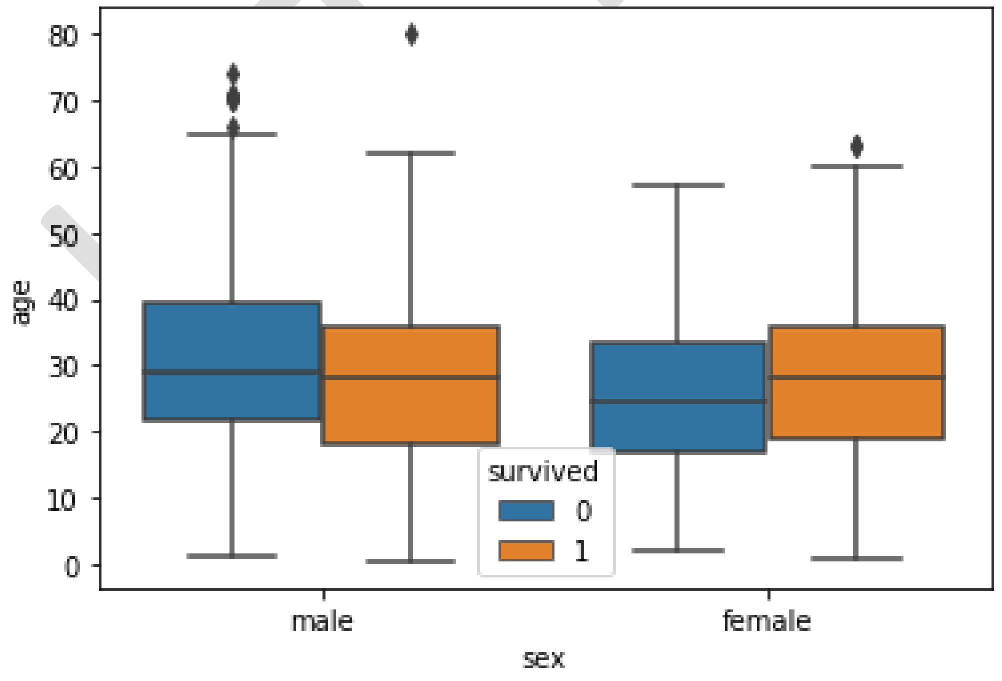
Now let's plot a box plot that displays the distribution for the age with respect to each gender. You need to pass the categorical column as the first parameter (which is sex in our case) and the numeric column (age in our case) as the second parameter. Finally, the dataset is passed as the third parameter, take a look at the following script:

```
sns.boxplot(x='sex', y='age', data=x1)
```



You can make your box plots more fancy by adding another layer of distribution. For instance, if you want to see the box plots of forage of passengers of both genders, along with the information about whether or not they survived, you can pass the survived as value to the hue parameter as shown below:

```
sns.boxplot(x='sex', y='age', data=x1, hue="survived")
```

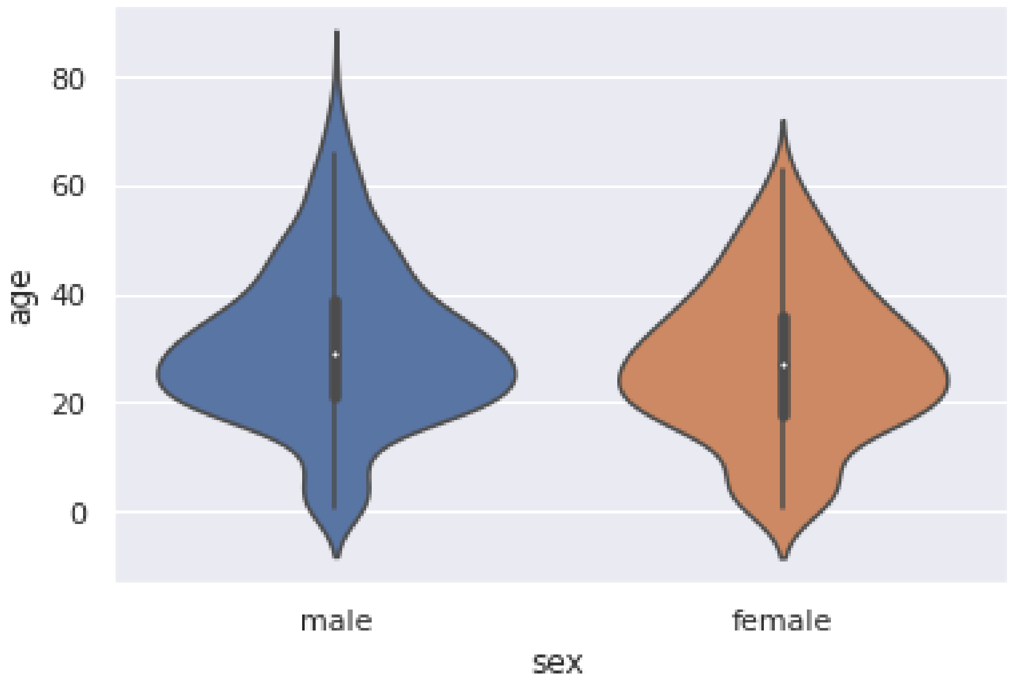


In above chart ,in addition to the information about the age of each gender, you can also see the distribution of the passengers who survived. For instance, you can see that among the male passengers, on average more younger people survived as compared to the older ones.

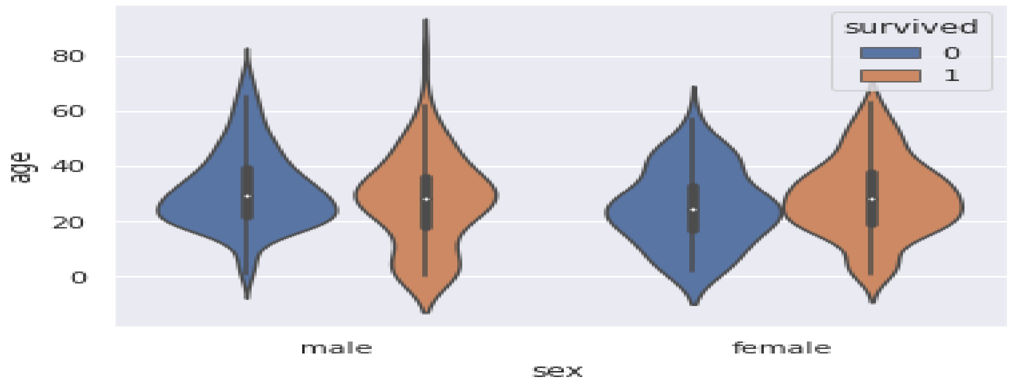
Similarly, you can see that the variation among the age of female passengers who did not survive is much greater than the age of the surviving female passengers.

The Violin Plot The violin plot is similar to the box plot, however, the violin plot allows us to display all the components that actually correspond to the data point. The violinplot() function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset.

```
sns.violinplot(x='sex', y='age', data=x1)
```



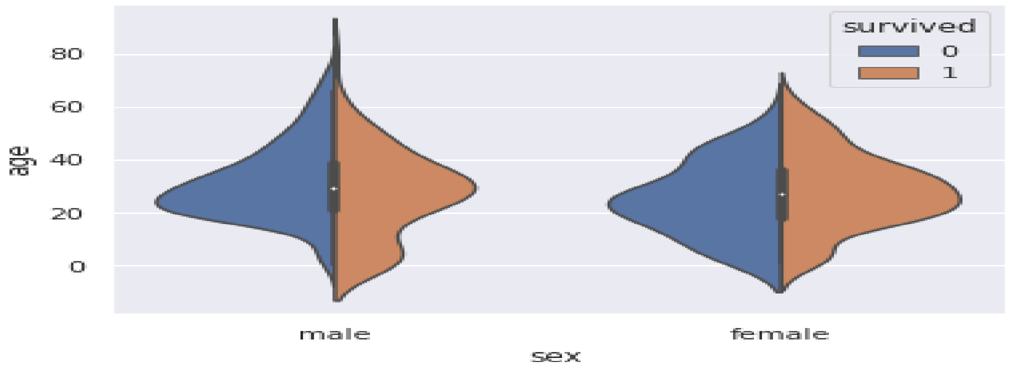
```
sns.violinplot(x='sex', y='age', data=dataset, hue='survived')
```



Now you can see a lot of information on the violin plot. For instance, if you look at the bottom of the violin plot for the males who survived (left-orange), you can see that it is thicker than the bottom of the violin plot for the males who didn't survive (left-blue). This means that the number of young male passengers who survived is greater than the number of young male passengers who did not survive. The violin plots convey a lot of information, however, on the downside, it takes a bit of time and effort to understand the violin plots.

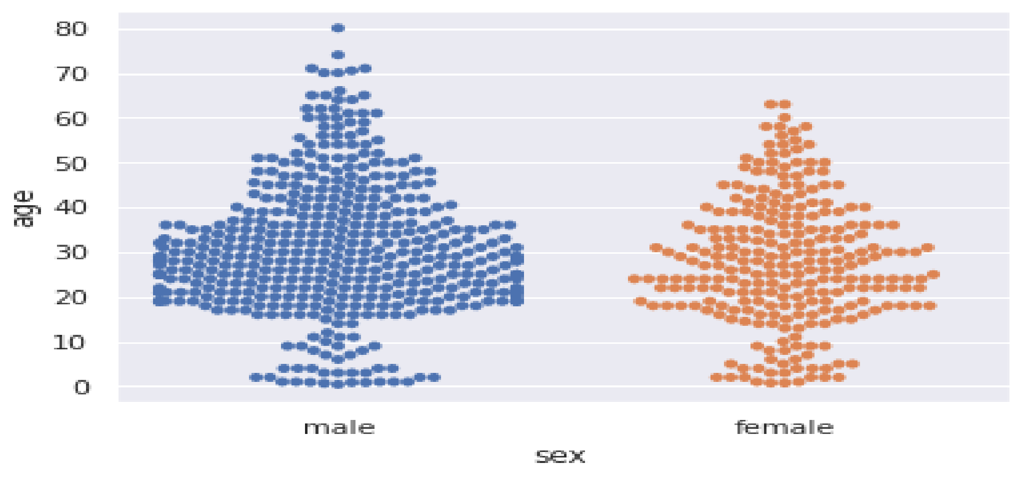
Instead of plotting two different graphs for the passengers who survived and those who did not, you can have one violin plot divided into two halves, where one half represents surviving while the other half represents the non-surviving passengers. To do so, you need to pass True as value for the split parameter of the violinplot() function. Let's see how we can do this:

```
sns.violinplot(x='sex', y='age', data=x1, hue='survived', split=True)
```

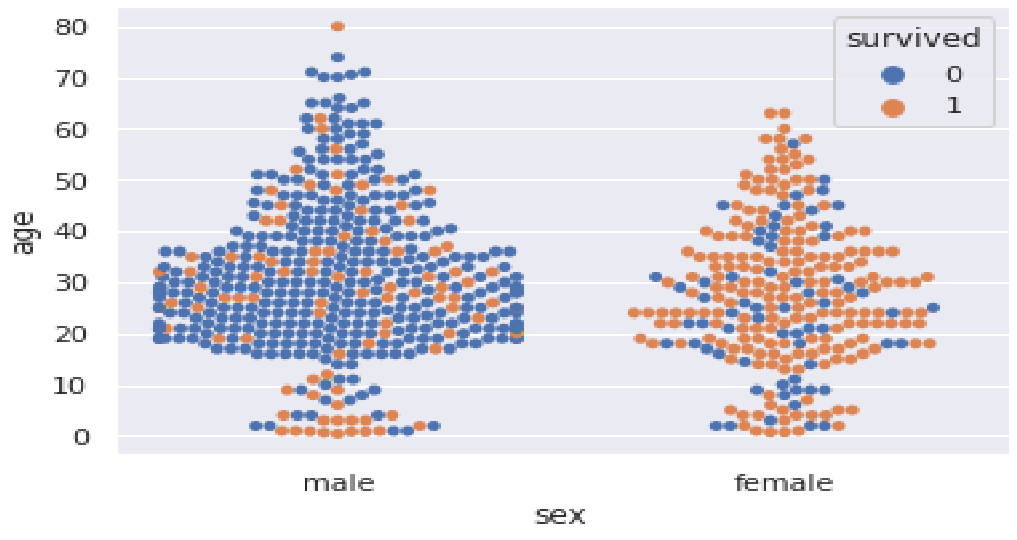


The Swarm Plot The swarm plot is a combination of the strip and the violin plots. In the swarm plots, the points are adjusted in such a way that they don't overlap. Let's plot a swarm plot for the distribution of age against gender. The swarmplot() function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. Look at the following script:

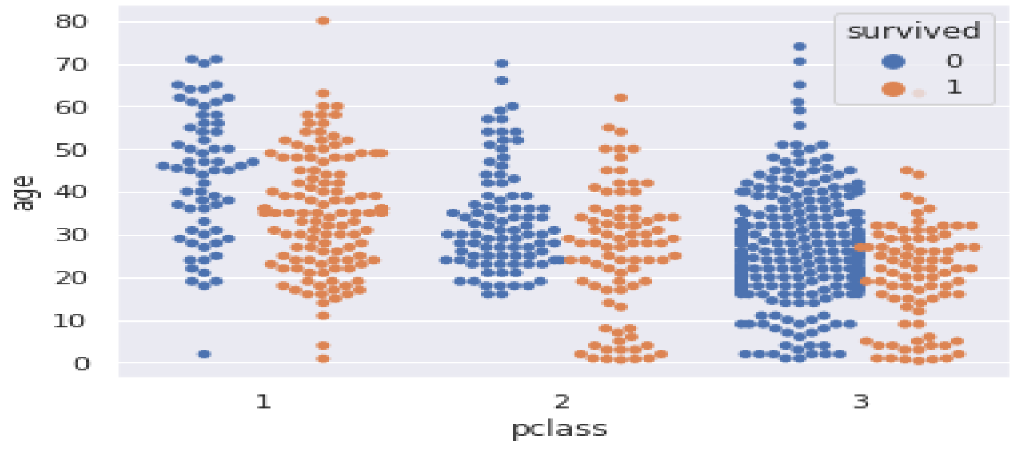
```
sns.swarmplot(x='sex', y='age', data=x1)
```



```
sns.swarmplot(x='sex', y='age', data=x1, hue='survived')
```



```
sns.swarmplot(x='pclass', y='age', data=x1, hue='survived', split=True)
```


survived
age
pclass