Practical 1

Problem Statement:

To use PCA Algorithm for dimensionality reduction on wine dataset and visualize data after applying PCA.

Dataset: wine Dataset.

dataset Link: https://media.geeksforgeeks.org/wp-content/uploads/Wine.csv

Objective:

1. Understand working of PCA algorithm
2. Visualize Data after applying PCA algorithm.

Outcome:

PCA Algorithm for dimensionality reduction should be understood.

Theory:

PCA is an unsupervised linear dimensionality reduction algorithm to find a more meaningful basis or coordinate system for our data and works based on covariance matrix to find the strongest features of your samples.

PCA is used for -

1. Dimensionality Reduction
2. Increasing Performance
3. Visualizing Higher Dimensional Data
4. Obscure Data
5. Create Independent Features

Requirements:

1) Get the Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the dataset.

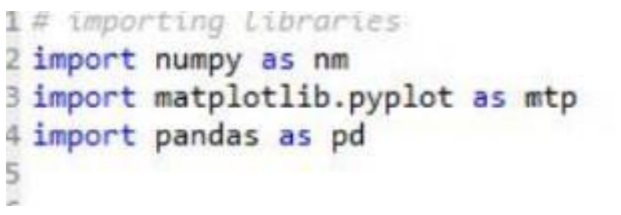
Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another dataset. To use the dataset in our code, we usually put it into a CSV file. However, sometimes, we may also need to use an HTML or xlsx file.

.CSV File

CSV stands for "Comma-Separated Values" files; it is a file format which allows us to save the tabular data, such as spreadsheets. It is useful for huge datasets and can use these datasets in programs.
For real-world problems, we can download datasets online from various sources such ashttps://www.kaggle.com. We can also create our dataset by gathering data using various API with Python and put that data into a .csv file.

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:
Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:
import numpy as nm
Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.
Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:
import matplotlib.pyplot as mpt
Here we have used mpt as a short name for this library.
Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:
Here, we have used pd as a short name for this library. Consider the below image



```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory and import wine dataset.

```
df = pd.read_csv('winequality.csv')
```

4) Scaling of data

Feature scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale. The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values.

```
scaler = MinMaxScaler()
```

5) Applying Principal Component Analysis

PCA is an unsupervised linear dimensionality reduction algorithm to find a more meaningful basis or coordinate system for our data and works based on covariance matrix to find the strongest features of your samples. We call PCA function on dataset.

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=3)
```

6) Visualize dataset using matplotlib library

After a PCA, the observations are expressed in principal component scores. Therefore, it is important to visualize the spread of the data along the new axes (principal components) to interpret the relations in the dataset.

```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1])#first two columns
plt.xlabel('first principle component')
plt.ylabel('second principle component')
```

Algorithm:

1. Getting the dataset
Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.

2. Representing data into a structure
Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

3. Standardizing the data
In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.
If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.

4. Calculating the Covariance of Z
To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.

5. Calculating the Eigen Values and Eigen Vectors
Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

6. Sorting the Eigen Vectors
In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

7. Calculating the new features Or Principal Components
Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

8. Remove less or unimportant features from the new dataset.
The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Conclusion:

Principal component analysis is a technique to summarize data, and is highly flexible depending on your use case. It can be valuable in both displaying and analysing a large number of possibly dependent variables. Techniques of performing principal component analysis range from arbitrarily selecting principal components, to automatically finding them until a variance is reached.

Oral Questions:

1. What all libraries are used for performing PCA?
2. How the data is imported in python?
3. Explain working of PCA?
4. Explain how PCA is visualized?

Practical 2

Problem Statement:

To Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:

1. Pre-process the dataset.
2. Identify outliers.
3. Check the correlation.
4. Implement linear regression and ridge, Lasso regression models.
5. Evaluate the models and compare their respective scores like R2, RMSE,

Dataset: Uber fares Dataset.

Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

Objective:

1. Understand working of linear regression and ridge, Lasso regression models.
2. Evaluate the models and compare their respective scores like R2, RMSE, etc.

Outcome:

Linear regression, ridge and Lasso regression models for prediction of test data should be understood and accuracy is evaluated.

Theory:

Linear regression is a type of statistical analysis used to predict the relationship between two variables. It assumes a linear relationship between the independent variable and the dependent variable, and aims to find the best-fitting line that describes the relationship. The line is determined by minimizing the sum of the squared differences between the predicted values and the actual values.
Linear regression is commonly used in many fields, including economics, finance, and social sciences, to analyse and predict trends in data. It can also be extended to multiple linear regression, where there are multiple independent variables, and logistic regression, which is used for binary classification problems.

Linear regression (in scikit-learn) is the most basic form, where the model is not penalized for its choice of weights, at all. That means, during the training stage, if the model feels like one particular feature is particularly important, the model may place a large weight to the feature.
This sometimes leads to overfitting in small datasets. Hence, following methods are invented.

Lasso is a modification of linear regression, where the model is penalized for the sum of absolute values of the weights. Thus, the absolute values of weight will be (in general) reduced, and many will tend to be zeros. During training, the objective function become:

$$\frac{1}{2m}\sum_{i=1}^{m}(y-Xw)^2+alpha\sum_{j=1}^{p}|w_j|$$

As you see, Lasso introduced a new hyperparameter, alpha, the coefficient to penalize weights.

Ridge takes a step further and penalizes the model for the sum of squared value of the weights. Thus, the weights not only tend to have smaller absolute values, but also really tend to penalize the extremes of the weights, resulting in a group of weights that are more evenly distributed. The objective function becomes:

$$\sum_{i=1}^{n}(y-Xw)^2+alpha\sum_{j=1}^{p}w_j^{\,2}$$

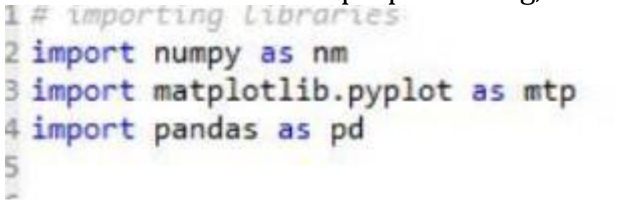
Requirements:

1) Get the Uber fare dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the dataset.

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:



```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project.

```
df = pd.read_csv('uber-fares-dataset')
print(df.head())
```

4) Data Preprocessing and Cleaning

Here we handle missing values, if necessary, remove outliers, and converting data types as needed.

a) Checking where is null value

```
df.isnull().sum()
```

b) Replace null value by mean and median value

```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace =
True)
```

c) Display the data types of each column in the DataFrame

```
  df.dtypes
```

d) Change the Incorrect data type.

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime)
df.dtypes
```

d) drop the column 'key' ,'Unnamed: 0' and 'pickup_daetime'

```
df = df.drop(['key' ,'Unnamed: 0', 'pickup datetime'],axis=1)
```

e) Outliers are data points that significantly deviate from the overall pattern or distribution of the dataset. They can be exceptionally high or low values that may distort statistical analysis or model performance, and they are important to identify and handle appropriately to avoid biased results or skewed interpretations of the data.

```
# Boxplot to check the outliers
# Resolving the outliers
#IQR to fill in the blanks
```

f) Correlation

quantifies the relationship between two variables. It indicates the degree to which the variables tend to change together, ranging from -1 (strong negative correlation) to 1 (strong positive correlation), with 0 indicating no linear relationship.

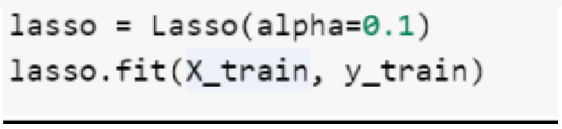
```
corr = df.corr()
```

5) Splitting out data to train and test using sklearn

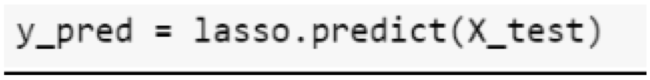
```
X_train, X_test, y_train, y_test = train_test_split(x_std, y_std,
test_size=0.2, random_state=0)
```

6) Applying simple regression , lasso and Ridge model and Evaluate models

```
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)
evaluate model performance on test data
y_pred = l_reg.predict(X_test)
```



```
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
```



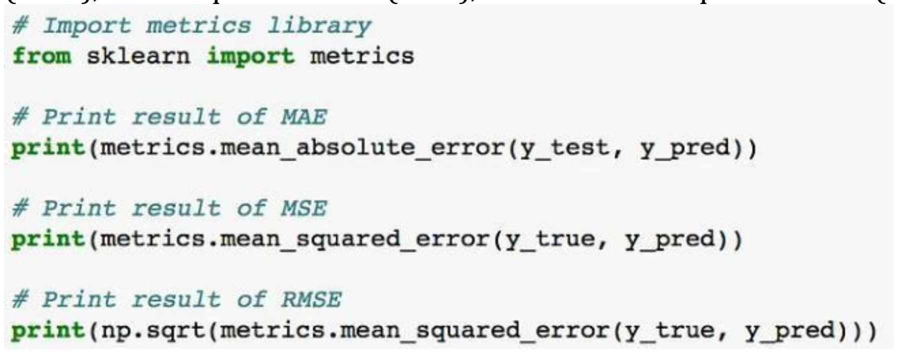
```
y_pred = lasso.predict(X_test)
```

```
    model = RidgeRegression( iterations = 1000,
                             learning_rate = 0.01, l2_penality = 1 )
    model.fit( X_train, Y_train )

    # Prediction on test set
    Y_pred = model.predict( X_test )
```

7) Model Evaluation

There are three primary metrics used to evaluate linear models. These are: Mean absolute error (MAE), Mean squared error (MSE), or Root mean squared error (RMSE).



```
# Import metrics library
from sklearn import metrics

# Print result of MAE
print(metrics.mean_absolute_error(y_test, y_pred))

# Print result of MSE
print(metrics.mean_squared_error(y_true, y_pred))

# Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_true, y_pred)))
```

Algorithm:

Linear regression

1. Data Pre-processing
2. Fitting the Simple Linear Regression to the Training Set
3. Prediction of test set result
4. visualizing the Training set results
5. visualizing the Test set results

Lasso regression

1. Data Pre-processing
2. Fitting the lasso regression to the Training Set
3. Prediction of test set result
4. visualizing the Training set results
5. visualizing the Test set results

Ridge regression

1. Data Pre-processing
2. Fitting the ridge regression to the Training Set
3. Prediction of test set result
4. visualizing the Training set results
5. visualizing the Test set results

Conclusion:

Linear regression the most basic form, where the model is not penalized for its choice of weights, at all. Ridge and Lasso's regression are a powerful technique for regularizing linear regression models and preventing overfitting. They both add a penalty term to the cost function, but with different approaches.

Oral Questions:

1. What is gradient descent?
2. What is cost function?
3. How Linear regression works?
4. What is difference between linear, lasso and ridge regression?

Practical 3

Problem Statement:

Implementation of Support Vector Machines (SVM) for classifying images of hand written digits into their respective numerical classes (0 to 9)

Objective:

1. Understand working of SVM algorithm
2. Evaluate the SVM models based on accuracy

Outcome:

Support Vector machine for prediction of test data should be understood and accuracy is evaluated.

Theory:

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

Types of Support Vector Machine Algorithms

1. Linear SVM

When the data is perfectly linearly separable only then we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line(if 2D).

2. Non-Linear SVM

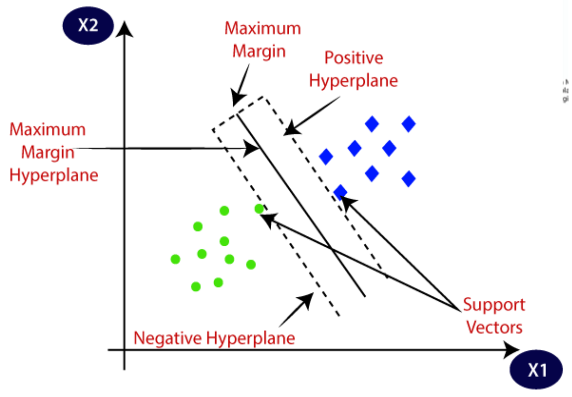
When the data is not linearly separable then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.

Important Terms

Two main terms in svm:

Support Vectors: These are the points that are closest to the hyperplane. A separating line will be defined with the help of these data points.

Margin: it is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins hard margin and soft margin. I will talk more about these two in the later section.



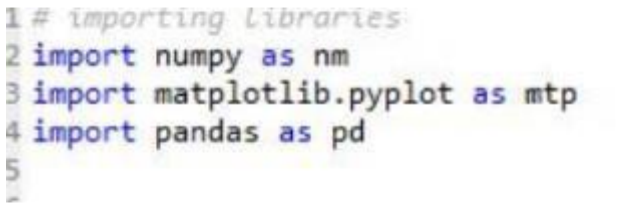
Requirements:

1) Get the dataset of images called Digits Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the dataset.

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:
Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:
import numpy as nm
Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.
Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:
import matplotlib.pyplot as mpt
Here we have used mpt as a short name for this library.
Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:
Here, we have used pd as a short name for this library. Consider the below image



```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project.

```
from sklearn import datasets   ## importing datasets from sklearn
digits = datasets.load_digits()  ### loading data from scikit_learn library
```

4) Data understanding and exploration

Here we understand and visualize missing values, if necessary, remove outliers, and converting data types as needed.

a) Display the data types of each column in the DataFrame

```
numbers.info()
```

b) Describe the distribution of your data

```
numbers.describe(percentiles = [0.05,0.10,0.25,0.50,0.75,0.90,0.99])
```

c) Checking for null values

```
round(100*(numbers.isnull().sum()/(len(numbers.index))),2).sort_values(ascending =
False)
```

d) Check unique entries of label column

```
np.unique(numbers['label'])
numbers['label'].value_counts()
```

e) Visualising the column - label

```
sns.countplot(numbers['label'],palette = 'icefire')
```

5) Data Preparation

a) Set average feature values

```
pd.set_option('display.max_rows', 999)
round(numbers.drop('label', axis=1).mean(), 2).sort_values(ascending = False)
```

b) Splitting into x and y

```
X = numbers.drop("label", axis = 1)
y = numbers['label']
```

c) scaling the features

```
X_scaled = scale(X)
```

d) Splitting out data to train and test using sklearn

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
train_size=0.2,test_size = 0.8, random_sta
```

6) Building SVM model

kernel is the hyperparameter that we want to tune, Linear Kernel is used when the data is Linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are many Features in a particular Data Set.

```
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)
```

7) Predicting test data

y_pred = model_linear.predict(X_test)

8) Model Evaluation

To evaluate the performance of the model, we calculate the accuracy of the model using the accuracy score method from the scikit-learn metrics module. The accuracy score measures the percentage of correctly classified data points out of all the data points. The accuracy score is calculated by comparing the predicted labels with the actual labels and then dividing the number of correct predictions by the total number of data points.

a) Calculating the accuracy of the model

```
accuracy = accuracy_score(y_pred, y_test)
```

Algorithm:

Support vector machine

1. Import the dataset
2. Explore the data to figure out what they look like
3. Pre-process the data
4. Split the data into attributes and labels
5. Divide the data into training and testing sets
6. Train the SVM algorithm
7. Make some predictions
8. Evaluate the results of the algorithm

Conclusion:

SVM is a supervised learning algorithm which separates the data into different classes using a hyperplane. The chosen hyperplane is one with the greatest margin between the hyperplane and all points, this yields the greatest likelihood of accurate classification.

Oral Questions:

1. What is a SVM algorithm?
2. Why is SVM the best algorithm?
3. What are the steps of SVM algorithm?
4. What does SVM do in machine learning?

Practical 4

Problem Statement:

Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters using the elbow method.

Dataset:

Iris.csv Dataset.

Dataset link: https://www.kaggle.com/datasets/uciml/iris

Objective:

1. Understand working of K-Means clustering algorithm on the Iris dataset
2. determine the optimal number of clusters using the elbow method

Outcome:

After implementing K-Means clustering on the Iris.csv dataset and using the elbow method, students will effectively determine the optimal number of clusters and understand the practical application of this clustering algorithm on real-world datasets.

Theory:

K Means algorithm is a centroid-based clustering (unsupervised) technique. This technique groups the dataset into k different clusters having an almost equal number of points. Each of the clusters has a centroid point which represents the mean of the data points lying in that cluster. The idea of the K-Means algorithm is to find k-centroid points and every point in the dataset will belong to either of the k-sets having minimum Euclidean distance.

Types of Support Vector Machine Algorithms

1. K-Means Clustering:

K-Means clustering is a partitioning method that divides a dataset into K distinct, non-overlapping clusters. The objective is to minimize the distance between data points within the same cluster and maximize the distance between clusters.

- **Strengths:**
  Simple and easy to implement.
  Scalable for large datasets.
- **Weaknesses:**
  The number of clusters K must be specified beforehand.
  Sensitive to initial centroid placement, which can lead to local optima.
  Assumes clusters are spherical and equally sized.

2. Elbow Method:

The Elbow method is a heuristic used in determining the optimal number of clusters K for the K-Means clustering algorithm. The method involves visualizing the variation explained as a function of the number of clusters, and picking the "elbow" of the curve as the number of clusters to use.

Key Points:

1. **WCSS Calculation:** For each value of K, compute the within-cluster sum of squares (WCSS). This represents the total distance of data points from their respective cluster centroids.
2. **Plotting:** Plot the curve of WCSS against the number of clusters K.
3. **Elbow Identification:** The point where the rate of decrease of WCSS sharply changes (representing an "elbow" in the graph) is considered an optimal value for K.

Rationale:
As the number of clusters increases, the total variance or WCSS will naturally decrease as data points will be closer to their respective centroids. However, after a certain number of clusters, the reduction in WCSS will be marginal, indicating that adding more clusters may not provide much additional value. The "elbow" of the curve represents the point of diminishing returns.

Limitations:

- The "elbow" may not always be clear and distinct.
- It's a heuristic, so it might not always yield the absolute optimal number of clusters.

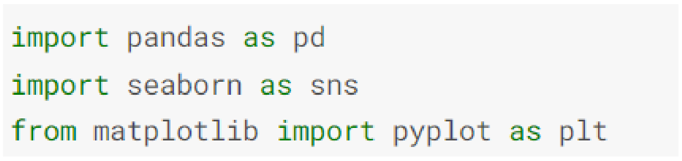
Requirements:

1) Get the Iris dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the dataset.

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:
Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:
import matplotlib.pyplot as mpt
Here we have used mpt as a short name for this library.
Seaborn: It is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions
import seaborn as sns
Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:
Here, we have used pd as a short name for this library. Consider the below image



```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
```

3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project.

```
data = pd.read_csv(    'Iris.csv'    )### loading data
```

4) Data understanding and exploration

Here we understand and visualize missing values, if necessary, remove outliers, and converting data types as needed.

a) Display the data types of each column in the DataFrame

```
  data.info()
```

b) Describe the distribution of your data

```
numbers.describe(include = 'all')
```

5) Data Preparation

a) scaling the features

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

6) Implementing K-Means Clustering:

```
from sklearn.cluster import KMeans

# Applying kmeans to the dataset
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
y_kmeans = kmeans.fit_predict(X)
```

7) Determining the Number of Clusters Using the Elbow Method:

```
import matplotlib.pyplot as plt

wcss = []  # Within-cluster sum of squares

for i in range(1, 11):

    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)

    kmeans.fit(X)

    wcss.append(kmeans.inertia_)
```

8) Visualizing the Clusters:

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s=100, c='red',
label='Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s=100, c='blue',
label='Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s=100, c='green',
label='Cluster 3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300,
c='yellow', label='Centroids'))
```

Algorithm:

K-Means Clustering

1. Initialization: Select K data points as initial centroids, either randomly or using some heuristic.
2. Assignment: Assign each data point to the nearest centroid, forming K clusters.
3. Update: Calculate the mean of all data points within each cluster and move the centroid to that mean location.
4. Convergence: Repeat the assignment and update steps until centroids no longer change, or a set number of iterations is reached.
The measure of "distance" is usually the Euclidean distance, but other distance metrics can also be used.

Elbow Method:

1. Computing the sum of squared distances from each point to its assigned center for different values of K. This is often referred to as within-cluster sum of squares (WCSS).
2. Plotting the curve of WCSS as a function of the number of clusters.
3. Observing the "elbow" point in the plot, where the rate of decrease sharply changes, indicating an optimal value for K.

The underlying idea is that increasing the number of clusters will naturally reduce the distance from data points to their respective centroids, leading to a decrease in WCSS. However, after a certain point, the reduction in WCSS becomes marginal, providing diminishing returns in clustering quality. This point, where adding another cluster doesn't give much better fit to the data, is the "elbow".

Conclusion:

We have learned about K Means clustering algorithm from scratch to implementation. First, we have seen clustering and then finding K value in K Means. For that we have seen Elbow method.

Oral Questions:

1. What is clustering?
2. Where is clustering used in real life?
3. When to use k-means and K medians?
4. What is difference between classification and clustering?

Practical 5

Problem Statement:

Implement Random Forest Classifier model to predict the safety of the car.

Dataset: car evaluation dataset

Dataset link: https://www.kaggle.com/datasets/elikplim/car-evaluation-data-set

Objective:

1. To understand and implement the Random Forest Classifier
2. Predict the safety of cars based on given features.

Outcome:

After implementing the Random Forest Classifier on the Car Evaluation Dataset, students will adeptly utilize ensemble methods to predict car safety and evaluate model performance based on provided features.

Theory:

Random Forest Classifier is a supervised ensemble learning method that constructs a multitude of decision trees during training and outputs the mode of the classes for classification or the mean/average prediction of the individual trees for regression. It introduces randomness into the model when growing the trees and combines their outputs to improve performance, reduce overfitting, and give a more accurate prediction.

Key Characteristics:

1. Ensemble Method: It combines multiple decision trees to create a more robust and accurate model.
2. Bagging: Random Forest uses bootstrap aggregation, where subsets of the dataset are selected with replacement and used to train individual trees.
3. Feature Randomness: In each split test, only a random subset of the features is considered, introducing further diversity among the trees.
4. Reduced Overfitting: By averaging out biases, the combined prediction is often more accurate and less prone to overfitting.
5. Handle Missing Values: Random Forest can handle missing values and still produce accurate predictions.
6. Feature Importance: The model provides insights into which features have the most impact on predictions.
7. Versatility: It can be used for both regression and classification tasks.
However, like all models, Random Forest has its disadvantages, including the potential for increased complexity and longer computation times compared to simpler algorithms, especially when dealing with very large datasets.

Requirements:

1 Data Acquisition:

- Download the Car Evaluation Dataset from the provided link or Kaggle.

2. Data Exploration:

- Load the dataset using tools like pandas.
- Conduct a preliminary examination using methods like head(), describe(), and info().
- Identify any missing values or anomalies that need addressing.

3. Data Preprocessing:

- **Handling Missing Values**: If any, use techniques like imputation.
- **Encoding Categorical Variables**: Since Random Forest requires numerical input, use methods like Label Encoding or One-Hot Encoding.
- **Feature Scaling**: Although not always necessary for Random Forest, scaling (e.g., Min-Max Scaling, Standard Scaling) can be applied for consistent feature representation.
- **Train-Test Split**: Partition the data into training and test sets to evaluate the model's performance.

4. Model Building:

- Initialize the Random Forest Classifier using scikit-learn.
- Train the classifier on the training data.

5. Model Evaluation:

- Use the trained model to predict the safety of cars in the test set.
- Calculate and evaluate metrics such as accuracy, precision, recall, and the F1 score.
- (Optional) Visualize the results using a confusion matrix, ROC curve, etc.

6. Feature Importance Evaluation:

- Extract feature importance scores from the trained Random Forest model to understand which features have the most influence on predictions.

7. Hyperparameter Tuning (Optional but Recommended):

- Use tools like GridSearchCV or RandomizedSearchCV to find the best hyperparameters for the Random Forest Classifier, optimizing for better performance.

8. Model Refinement (Based on Evaluation and Hyperparameter Tuning):

- If the initial performance is unsatisfactory, refine the model. This can involve:
    a. Adjusting model hyperparameters.
    b. Incorporating more features or removing irrelevant ones.
    c. Trying advanced preprocessing techniques.

Algorithm:

Random Forest Classifier:

1. Input:

- Dataset D of size N.
- The number T of trees to be generated.

- The size m of the random subset of features to be considered at each split (usually $\sqrt{total^2 - feautures^2}$ for classification).
- Optional: Size n for the subsample of the training data, with or without replacement (if not provided, then n = N).

2. Output: A list of 'T' decision trees.

3. Procedure:

For i = 1 to T do:

- Create a bootstrap sample 'D_i' of size n by randomly selecting n samples from 'D' with replacement.
- Grow a decision tree 'Tree_i' on 'D_i' as follows:
    a. For each node of the tree, randomly select 'm' features without replacement.
    b. Split the node using the feature that provides the best split (according to some criterion, often Gini impurity or entropy for classification).
    c. Recurse for the split subsets until the maximum depth is reached, or another stopping criterion is met (e.g., minimum samples per leaf).
- Add 'Tree_i' to the list of trees.

4. Prediction: Given a new sample 'S':

a. For each tree Tree_i in the list of trees, get a prediction 'P_i'.
b. For classification, the final prediction is the mode of all the predictions {'P_1, P_2, ... P_T'}. For regression, it would be the mean.

Conclusion:

We have learned about Random Forest Classifier, being an ensemble method, provides more robust predictions than a single decision tree. Its capability to determine feature importance helps in understanding which features most influence predictions.

Oral Questions:

1. What is the primary reason to use Random Forest over a single decision tree?
2. How does increasing the number of trees in a Random Forest impact its performance?
3. Explain the concept of bootstrap sampling.
4. Why is feature randomness introduced during the tree-building process in Random Forest?

Practical 6

Problem Statement:

Build a Tic-Tac-Toe game using reinforcement learning in Python by using following tasks
a. Setting up the environment
b. Defining the Tic-Tac-Toe game
c. Building the reinforcement learning model
d. Training the model
e. Testing the model

Objective:

To design and implement a Tic-Tac-Toe game where an agent learns optimal gameplay using Q-learning.

Outcome:

After implementing the reinforcement learning on Tic-Tac-Toe game, students will adeptly utilize reinforcement learning methods to solve very complex problems that cannot be solved by conventional techniques.

Theory:

Reinforcement Learning is a type of machine learning paradigm where an agent learns how to behave in an environment by performing certain actions and receiving rewards or penalties in return. It's inspired by behavioural psychology and involves agents who take actions in an environment to maximize some notion of cumulative reward.

Key Characteristics:

1) **Agent**: The decision-maker or learner.
2) **Environment**: Everything the agent interacts with.
3) **State (s)**: A situation returned by the environment.
4) **Action (a)**: What the agent can do.
5) **Policy (π)**: A strategy that defines the learning agent's way of behaving at a given time. It's a map from states to actions.
6) **Reward (R)**: A feedback from the environment. It can be immediate or delayed.
7) **Value Function (V)**: It predicts the expected cumulative future reward for a given state. Essentially, it tells the worth of a state.
8) **Q-function (or Action-Value Function)**: Represents the expected return of taking an action in a particular state, then following policy π.

Learning Process:

- The agent observes the current state and takes an action.
- The environment, upon receiving the action, transitions to a new state.
- The environment then gives a reward (or penalty) to the agent based on the action taken.

- The agent uses this reward to update its knowledge or policy.

The goal of the agent is to maximize the expected cumulative reward over time, often referred to as the **discounted future reward**. The agent does this by learning the optimal policy that gives the highest reward over the long run.

Exploration vs. Exploitation:

In RL, there's a continuous dilemma of Exploration vs. Exploitation:

Exploration: Trying out new actions to see their effects.

Exploitation: Using the known actions that yield the maximum reward.

Algorithm:

1 . Setting up the environment

a) Install Required Libraries:

```
pip install numpy
```

b) Initialize Variables:

States: All possible positions on the Tic-Tac-Toe board.

Actions: All possible moves a player can make.

Rewards: Assign rewards for winning, losing, drawing, or making a move.

2. Defining the Tic-Tac-Toe game

- **Board Repesentation:** Use a 3x3 matrix or a list of 9 elements to represent the board.
- **Move Function:** Given a state and an action, return the resulting state.
- **Check End Function:** Check if the game has ended, and return the result (win, draw, or lose).
- **Reset Function:** Resets the game to its initial state.

3. Building the reinforcement learning model

1. Q-Table Initialization: Create a table with states as rows and actions as columns. Initially, all values are zeros.

2. Policy Function: Given a state, decide which action to take. You can start with a random policy and refine it as you go.

3. Q-Learning Algorithm:

- Choose an action using the policy.
- Take the action, then observe the reward and the new state.
- Update the Q-value of the taken action using the observed reward and the highest Q-value of the new state.
- Loop until the game ends.

4. Training the model

1. **Episodic Learning:** Play the game multiple times (episodes). In each episode:
   - Start with an initial state.
   - Play the game until it ends, updating the Q-values using the Q-learning algorithm.
   - Adjust the policy gradually to rely more on the Q-values and less on random choices.

2. **Adjust Learning Rate and Discount Factor:** Start with a high learning rate and decrease it over time. Use a discount factor (usually denoted as gamma) to balance immediate vs. future rewards.C
3. **Exploration-Exploitation Trade-off:** Initially, make the agent explore more (choose random moves). Gradually, make it exploit its learned knowledge.

5. Testing the model

1. **Evaluation Metric:** Decide on a metric to evaluate the model's performance. For example, the percentage of games won over a set number of games.
2. **Play Games:** Use the trained model to play a certain number of games. Ideally, the agent should win most of the games against a random opponent and play optimally against a smart opponent.
3. **Observe and Adjust:** If the performance isn't satisfactory, you may need to adjust hyperparameters, train for more episodes, or modify the reward structure.

Conclusion:

Building a Tic-Tac-Toe game using reinforcement learning in Python allows for the development of an AI agent capable of understanding game dynamics and making optimal moves. Through iterative training and testing, the model can achieve competent gameplay, potentially matching or surpassing human performance.

Oral Questions:

1. What is reinforcement learning and how does it differ from supervised and unsupervised learning?
2. What are some real-world applications of reinforcement learning?
3. How does the Q-learning algorithm work, and what is the Q-function?
4. What is the role of a reward signal in reinforcement learning?