1.Demonstrate the following data preprocessing tasks using python libraries.

a) Loading the dataset

b) Identifying the dependent and independent variables

c) Dealing with missing data

a) Loading the Dataset (ie Importing)

To import the dataset use **read_csv()** of the pandas' library which is used to read a CSV file and perform various operations on it.

```
data = pd.read_csv('dataset.csv')
# following code is to display top 5 rows of csv file
print.head()
```

Note: Load csv file to the current working directory or else specify complete path.

Sample csv file (opened in excel)

b) Extracting Dependent And Independent Variables:

To create a matrix of Independent variables and a vector of dependent variables.

At first, decide and confirm that which columns or factors are independent variables(also known as features) that are used to train model which affects target variables.

c) Dealing with missing data

Missing data can create huge problems to the results hence they are necessary to be removed from the dataset.

Missing values usually take the form of **NaN or NONE**.

Pandas treat None and NaN as essentially interchangeable for indicating missing or NULL values.

The cause of missing value is: sometimes most of the fields in columns are empty which needs to be filled by correct data and sometimes there is incorrect data or poor quality of data which adversely affects the outputs. There are several ways to deal with missing values and fill them:

- The first way to deal with NULL values: simply delete the rows or columns which are having NULL values. But most of the time this may lead to loss of information so this method is not so efficient.
- Another important method is to fill the data in place of NULL values. You can calculate the mean of a specific row and column and fill that in place of Null values. The method is very useful in which columns have numerical data such as age, salary, weight, year, etc.
- Fill up missing values with whatever value comes directly after it in the same column.

The decisions depend on the type of data, and the results.

If missing values are less then it's good to delete them and if more than 50% of values are missing then it needs to fill with correct data.

Checking for missing values using isnull()

In order to check null values in Pandas DataFrame, use isnull() function this function return dataframe of Boolean values which are True for NaN values.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset = pd.read_csv("suv_datas.csv")
dataset.head()
new_df1=dataset.fillna(method="ffill")
new_df1
# input
x = dataset.iloc[:, [2, 3]].values
# output
y = dataset.iloc[:, 4].values
print(x)
```

```
print(y)
bool_series=pd.isnull(dataset["Gender"])
dataset[bool_series]
bool_series=pd.notnull(dataset["Gender"])
dataset[bool_series]
dataset[10:25]
new_data=dataset.dropna(axis=0,how='any')
new_data
dataset.replace(to_replace=np.nan,value=-99)
dataset["Gender"].fillna("No Gender",inplace=True)
dataset
print("Old data frame length:", len(dataset))
print("New data frame length:", len(dataset))
print("Number of rows with at least 1 NA value:", len(dataset)-len(new_data))
new_df1=dataset.fillna(method="ffill")
new_df1
new_df3=dataset.dropna(how='all')
new_df3

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset = pd.read_csv("suv_datas.csv")
dataset.head()
new_df1=dataset.fillna(method="ffill")
new_df1
new_df=dataset.fillna(method="bfill")
new_df
print(dataset.isna().sum())
new_df3 = dataset.dropna(how='all')
new_df3
from sklearn import preprocessing
```

```
print(y)
bool_series=pd.isnull(dataset["Gender"])
dataset[bool_series]
bool_series=pd.notnull(dataset["Gender"])
dataset[bool_series]
dataset[10:25]
new_data=dataset.dropna(axis=0,how='any')
new_data
dataset.replace(to_replace=np.nan,value=-99)
dataset["Gender"].fillna("No Gender",inplace=True)
dataset
print("Old data frame length:", len(dataset))
print("New data frame length:", len(dataset))
print("Number of rows with at least 1 NA value:", len(dataset)-len(new_data))
new_df1=dataset.fillna(method="ffill")
new_df1
new_df3=dataset.dropna(how='all')
new_df3

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset = pd.read_csv("suv_datas.csv")
dataset.head()
new_df1=dataset.fillna(method="ffill")
new_df1
new_df=dataset.fillna(method="bfill")
new_df
print(dataset.isna().sum())
new_df3 = dataset.dropna(how='all')
new_df3
from sklearn import preprocessing
```

```
x=dataset.iloc[:,1:4].values
print("\n ORIGINAL VALUES:\n\n",x)
min_max_scaler=preprocessing.MinMaxScaler(feature_range=(0,1))
new_x=min_max_scaler.fit_transform(x)
```

2. Demonstrate the following data preprocessing tasks using python library a) Dealing with categorical data b) Scaling the features c) Splitting dataset into Training and Testing Sets

Encoding Categorical Variables: One common approach is to convert categorical variables into numerical values. There are several methods for encoding categorical variables, such as one-hot encoding and label encoding.

a) **One-Hot Encoding**: This method creates binary columns for each category and represents the presence of a category with a 1 in the corresponding column.

Splitting dataset into training and testing set

The very crucial step of data preprocessing to split the dataset into training and testing sets. If we train the model on the different datasets and test it with a completely different dataset from training then it will create difficulties for the model to estimate the creation between independent and dependent variables and the accuracy of the model will be very less.

So, we always try to make a machine learning model that performs well with the training set as well with the testing set.

- **Training set:** Subset of the dataset to train the model, the outputs are known to us as well to model.
- **Testing set:** Subset of the dataset to test the model, which model predicts the output based on training given to the model.

Feature Scaling

It's a method to standardize the training dataset in a specific range. In feature scaling, all the values are kept in the same range and on the same scale so that no variable dominates the other variable.

For example, we have age and salary in the training dataset then they are not on the same scale as age is 31 and salary is 48000.

As Machine learning model is based on Euclidean Distance and if we did not scale the variable it will cause a problem in results and performance.

EUCLIDEAN DISTANCE BETWEEN A & B = $\sum[(x_2 - x_1)^2 + (y_2 - y_1)^2]$

For feature scaling, we use the **StandardScaler** class from the **preprocessing** library.

Code:

```
import pandas as pd
import numpy as np
data = pd.read_csv("Bankwages.csv")
data.head()
x=data.iloc[:,:-1].values
print(x[:8,:])
y=data.iloc[:,-1].values
print(y)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
print(x_train)
print(x_test)
print(y_train,y_test)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
print(x_train)
print(x_test)
print(y_train,y_test)
data['job']=data['job'].astype('category')
data['job_new']=data['job'].cat.codes
from sklearn.preprocessing import OneHotEncoder
enc=OneHotEncoder()
enc_data=pd.DataFrame(enc.fit_transform(data[['job_new']]).toarray())
enc_data
new_df=data.join(enc_data)
new_df
```

3. Write Python code to select features in machine learning using Python.

1. **Load Dataset**: Reads the dataset from a CSV file into a pandas DataFrame.
2. **Split Dataset**: Splits the dataset into training and testing sets.
3. **Feature Selection**: Uses SelectKBest with ANOVA F-value method to select the top features.
4. **Transform Dataset**: Transforms the original dataset to retain only the selected features.
5. **Print Feature Scores**: Prints the scores of each feature.
6. **Plot Feature Importance**: Plots the feature importance scores for visualization.

This process helps in selecting the most relevant features for the machine learning model, which can lead to better performance and interpretability.

Code:

```
from pandas import read_csv
from numpy import set_printoptions
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from matplotlib import pyplot
path=r'newdaibetes.csv'
names=['preg','plas','pres','skin','test','mass','peds','age','class']
dataframe=read_csv(path,names=names)
dataframe.head()
array=dataframe.values
x=array[:,0:8]
y=array[:,8]
print(x)
print(y)
x_train,x_test,y_train,y_test,=train_test_split(x,y,test_size=0.33,random_state=1)
fs= SelectKBest(score_func=f_classif,k='all')
fs.fit(x_train,y_train)
x_train_fs=fs.transform(x_train)
x_test_fs=fs.transform(x_test)
for i in range(len(fs.scores_)):
print('feature %d:%f'%(i,fs.scores_[i]))
```

```
pyplot.bar([i for i in range(len(fs.scores_))],fs.scores_)
pyplot.show()
```

4. Write Python code to load the data from a CSV file and select the top 10 features using the chi-squared test. The selected features are to be printed on the console.

1. **Loading the Dataset**:
   - Load the dataset from a CSV file into a pandas DataFrame.
   - Inspect the dataset to understand its structure and features.
2. **Preprocessing**:
   - Convert categorical variables into numerical values if needed.
   - Select relevant features and the target variable for analysis.
3. **Feature Selection with Chi-Squared Test**:
   - Use the chi2 function from scikit-learn to calculate chi-squared statistics and p-values for feature selection.
   - Sort the features based on their p-values to identify the most significant features.
4. **Visualization**:
   - Visualize the p-values of features using a bar plot to understand their significance in predicting the target variable.
5. **Selecting Top Features**:
   - Select the top 10 features with the lowest p-values obtained from the chi-squared test.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split
df=pd.read_csv("Churn.csv",sep=",")
df.head()
df.dtypes
df=df[['gender','Contract','PaperlessBilling','Churn']]
df["gender"]=df["gender"].map({"Female":1,"Male":0})
df["Contract"]=df["Contract"].map({"Month-to-month":0,"One year":1,"Two year":2})
df["PaperlessBilling"]=df["PaperlessBilling"].map({"Yes":0,"No":1})
df["Churn"]=df["Churn"].map({"Yes":1,"No":0})
df.head()
```

```
x=df.iloc[:,0:3]
y=df.iloc[:,-1]
f_score=chi2(x,y)
f_score
p_value=pd.Series(f_score[1], index=x.columns)
p_value.sort_values(ascending=True,inplace=True)
p_value
p_value.plot(kind="bar")
plt.xlabel("Features", fontsize=20)
plt.ylabel("p_values", fontsize=20)
plt.title("chi squared test base on p value")
plt.show()
```

5. Build a classification model using Decision Tree algorithm on iris dataset.

1. **Importing Libraries**: The necessary libraries such as NumPy, matplotlib, pandas, seaborn, and scikit-learn are imported.
2. **Loading the Iris Dataset**: The Iris dataset is loaded using either scikit-learn's **load_iris()** function or seaborn's **load_dataset()** function. The dataset is stored in the variable **iris**.
3. **Data Preprocessing**:
   - The features (X) and target variable (y) are separated.
   - The dataset is split into training and testing sets using **train_test_split()**.
4. **Building the Decision Tree Model**:
   - An instance of the DecisionTreeClassifier is created.
   - The model is trained on the training data using **fit()**.
5. **Making Predictions**:
   - Predictions are made on the testing data using **predict()**.
6. **Visualizing the Decision Tree**:
   - The trained decision tree model is visualized using **plot_tree()** from the **tree** module.
7. **Model Evaluation**:
   - The classification report is printed using **classification_report()** to evaluate the model's performance.
   - The confusion matrix is printed using **confusion_matrix()** to assess the model's performance on each class.
   - The accuracy score is computed using **accuracy_score()** to quantify the model's overall accuracy.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import tree
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
iris = load_iris()
iris = sns.load_dataset('iris')
iris.head()
```

```
x=iris.iloc[:,:-1]
y=iris.iloc[:,-1]
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.33, random_state=42)
treemodel = DecisionTreeClassifier()
treemodel.fit(x_train, y_train)
y_pred = treemodel.predict(x_test)
plt.figure(figsize=(20,30))
tree.plot_tree(treemodel, filled=True)
print(classification_report(y_test, y_pred))
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
from sklearn.metrics import accuracy_score
```

6. Apply Naïve Bayes Classification algorithm on any dataset.

Naïve Bayes Classification Algorithm on User_data Dataset:

- **Dataset**: The code loads the 'User_data.csv' dataset.
- **Preprocessing**: It separates the features and target variable, and then scales the features using StandardScaler.
- **Model Training**: The Gaussian Naïve Bayes classifier is trained on the training data.
- **Model Evaluation**: Confusion matrix is printed to evaluate the model's performance.

Code:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
dataset = pd.read_csv('User_data.csv')
x= dataset.iloc[:,[2,3]].values
print(x)
y = dataset.iloc[:,4].values
print(y)
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)
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train , y_train)
y_pred= classifier.predict(x_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

7. Apply KNN Classification algorithm on any dataset.

KNN Classification Algorithm on Social_Network_Ads Dataset:

- **Dataset**: The code loads the 'Social_Network_Ads.csv' dataset.
- **Preprocessing**: It scales the features using StandardScaler.
- **Model Training**: The K-Nearest Neighbors classifier is trained on the training data.
- **Model Evaluation**: Confusion matrix and accuracy score are printed to evaluate the model's performance.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:,-1].values
dataset.head()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)
X
y
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
print(classifier.predict(sc.transform([[46,28000]])))
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
accuracy_score(y_test, y_pred)
```

8. Build a model using linear regression algorithm on any dataset.

Linear Regression Algorithm on Salary_Data Dataset:

- **Dataset**: The code loads the 'Salary_Data.csv' dataset.
- **Model Training**: Linear regression model is trained on the training data.
- **Model Evaluation**: Mean squared error is computed to evaluate the model's performance.
- **Visualization**: Scatter plots are plotted to visualize the regression line on both training and test sets.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Salary_Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:,-1].values
dataset.head()
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=1/3, random_state=0)
print(x_train)
print(x_test)
print(y_train)
print(y_test)
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(x_train,y_train)
y_pred=regressor.predict(x_test)
print(y_test)
print(y_pred)
print(np.concatenate((y_test.reshape(len(y_test),1),y_pred.reshape(len(y_pred),1)),1))
from sklearn.metrics import mean_squared_error
mean=mean_squared_error(y_test,y_pred)
mean
plt.scatter(x_train,y_train,color='red')
```

```
plt.plot(x_train,regressor.predict(x_train),color='blue')
plt.title('salary vs Experience(Training set)')
plt.xlabel('years of Experience')
plt.ylabel('salary')
plt.show()
plt.scatter(x_test, y_test, color = 'red')
plt.plot(x_train, regressor.predict(x_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

9. Build a model using multi linear regression algorithm on any dataset.

Multiple linear regression is a statistical technique that uses multiple linear regression to model more complex relationships between two or more independent variables and one dependent variable. It is used when there are two or more x variables.

- Load the dataset.
- Preprocess the data if needed (one-hot encoding).
- Split the dataset into training and testing sets.
- Train a multi-linear regression model on the training data.
- Evaluate the model's performance using mean squared error.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('50_Startups_dataset.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:,-1].values
dataset.head()
x
y
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[4])],
remainder='passthrough')
x = np.array(ct.fit_transform(x))
print(x)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
y_pred = regressor.predict(x_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_test.reshape(len(y_test),1),y_pred.reshape(len(y_pred),1)),1))
from sklearn.metrics import mean_squared_error
```

```
mean=mean_squared_error(y_test,y_pred)
mean
```

10. Apply Hierarchical Clustering algorithm on any dataset.

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.

In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

1. **Agglomerative:** Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

Code:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
x = dataset.iloc[:, [3, 4]].values
dataset.head()
#Finding the optimal number of clusters using the dendrogram
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
y_pred= hc.fit_predict(x)
```

```
#visulaizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

11. Apply DBSCAN clustering algorithm on any dataset.

Density-Based Clustering refers to one of the most popular unsupervised learning methodologies used in model building and machine learning algorithms. The data points in the region separated by two clusters of low point density are considered as noise. The surroundings with a radius ε of a given object are known as the ε neighborhood of the object. If the ε neighborhood of the object comprises at least a minimum number, MinPts of objects, then it is called a core object.

The primary features of Density-based clustering are given below.

- It is a scan method.
- It requires density parameters as a termination condition.
- It is used to manage noise in data clusters.
- Density-based clustering is used to identify clusters of arbitrary size.

Code:

```
import numpy as np
 import pandas as pd
 import seaborn as sns
 import matplotlib.pyplot as plt
 from sklearn.cluster import DBSCAN
df=pd.read_csv('Mall_Customers.csv')
df.head()
x_train=df[['Age','Annual Income (k$)','Spending Score (1-100)']]
 x_train
 clustering=DBSCAN(eps=12.5, min_samples=4).fit(x_train)
 DBSCAN_dataset=x_train.copy()
 DBSCAN_dataset.loc[:,'Cluster']=clustering.labels_
DBSCAN_dataset.Cluster.value_counts().to_frame()
 outliers = DBSCAN_dataset[DBSCAN_dataset['Cluster'] == -1]
 fig2, axes = plt.subplots(1, 2, figsize=(12, 5))
 sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)',
 data=DBSCAN_dataset[DBSCAN_dataset['Cluster'] != -1],
  hue='Cluster', ax=axes[0], palette='Set2', legend='full', s=200, color='black')
  sns.scatterplot(x='Age', y='Spending Score (1-100)',
```

```
data=DBSCAN_dataset[DBSCAN_dataset['Cluster'] != -1],
hue='Cluster', ax=axes[1], palette='Set2', legend='full', s=200)

axes[0].scatter(outliers['Annual Income (k$)'], outliers['Spending Score (1-100)'], s=50,
label='outliers')
axes[1].scatter(outliers['Age'], outliers['Spending Score (1-100)'], s=50, label='outliers')
axes[0].legend(fontsize='12')
axes[1].legend(fontsize='12')
plt.show()
```