1. **Supervised ML**

**Train Test Split**

|  |  |
| --- | --- |
| Import | from sklearn.model\_selection import train\_test\_split |
| Assigning | X\_train,X\_test,y\_train,y\_test=  train\_test\_split  (X,y,train\_size=0.70,random\_state=0) |

**Linear Regression**

|  |  |
| --- | --- |
| Import | from sklearn.linear\_model import LinearRegression |
| Assigning variable as Model | model1=LinearRegression() |
| fit 🡪Train the Data | model1.fit(X\_train,y\_train) |
| test | y\_pred=model1.predict(X\_test) |
| Coefficient | model1.coef\_ |
| Intercept | model1.intercept\_ |
| Import r2 score  only for Regression type output data | from sklearn.metrics import r2\_score |
| r2 score | r2\_score(y\_test,y\_pred) |
| Import MSE  loss or error | from sklearn.metrics import mean\_squared\_error |
| MSE | mean\_squared\_error(y\_test,y\_pred) |
| RMSE | RMSE=np.sqrt(MSE) |

**Multi Linear Regression**

|  |  |
| --- | --- |
| X – variable | X=dataset.drop([‘output\_column’], axis = 1) |
| y – variable | y = dataset.loc[‘output\_column’] |
| plotting the regression line | plt.scatter(X\_test,y\_test)  plt.plot(X\_test,y\_pred, color = 'red') |

**One Hot Encoding**

|  |  |
| --- | --- |
| Creating dummies | data\_dummy = pd.get\_dummies(Dataset) |
| auto removing 1st column dummies | data\_dummy2 = pd.get\_dummies(Dataset, drop\_first = True) |
| assigning values for Ordinal column | Label = {"Ex":4,"Gd":3,"TA":2, "Fa":1} |
| replacing numerical values in the column | Data['col\_name\_Oe'] = Data['col\_name']. map (Label) |
| shortcut with numpy | Data [' output\_col''] = np.where(Dataset['output\_col'] == 'yes',1,0) |

**Standard Scaling**

|  |  |
| --- | --- |
| Import | from sklearn.preprocessing import StandardScaler |
| model | sc = StandardScaler() |
| fit | sc.fit(X\_train) |
| transform | X\_train\_sc = sc.transform(X\_train)  X\_test\_sc = sc.transform(X\_test) |

**Logistic Regression**

|  |  |
| --- | --- |
| Import | from sklearn.linear\_model import LogisticRegression |
| assign calling with object | model1 = LogisticRegression() |
| fit | model1.fit(X\_train,y\_train) |
| predict | y\_pred = model1.predict(X\_test) |
| Import accuracy score | from sklearn.metrics import accuracy\_score |
| check accuracy | accuracy\_score(y\_test,y\_pred) |
| Import Confusion Matrix | from sklearn.metrics import confusion\_matrix |
| Confusion Matrix | confusion\_matrix(y\_test,y\_pred) |
| see heat map of confusion matrix | sns.heatmap(name\_of\_matrix, annot=True) |
| Import Classification Report | **from** **sklearn.metrics** **import** classification\_report |
| Classification Report | (classification\_report  (y\_test,y\_pred)) |

**KNN (K Nearest Nambour)**

|  |  |
| --- | --- |
| Import | from sklearn.neighbors import KNeighborsClassifier |
| assign | model1 = KNeighborsClassifier() |
| assigning k value | model=NearestNeighbors(n\_neighbors=1) |
| fit | model1.fit(X\_train\_sc,y\_train) |
| Prediction | y\_pred = model1.predict(X\_test\_sc) |

**Multicollinearity**

|  |  |
| --- | --- |
| 1st Correlation | Data.corr() |
| Import Multicollinearity  VIF Import | from statsmodels.stats.outliers\_influence import variance\_inflation\_factor |

**Decision Tree**

|  |  |
| --- | --- |
| Importing the data from sklearn | from sklearn.datasets import load\_iris |
| Importing the data from seaborn | iris1 = sns.load\_dataset('iris') |
| assigning variable to the data | iris = load\_iris() |
| Decision Tree for classification problems | from sklearn.tree import DecisionTreeClassifier |
| Decision Tree for regression problems | sklearn.tree import DecisionTreeRegressor |
| model assigning | model1 = DecisionTreeClassifier() |
| Plotting the decision tree | from sklearn import tree  plt.figure(figsize = (15,10))  tree.plot\_tree(model1, filled = True)  plt.show() |
| post pruning  it is done after the Decision tree | model2 = DecisionTreeClassifier(max\_depth = 2)  model2.fit(X\_train,y\_train) |

**Pruning**

**Hyper Parameter Tuning**

|  |  |
| --- | --- |
| Hyper Parameters | parameter = {  'criterion' : ["gini", "entropy", "log\_loss"],  'splitter' : ["best","random"],  'max\_depth' : [1,2,3,4,5],  'max\_features' : ["auto","sqrt","log2"]  } |
| Import Grid Search CV | from sklearn.model\_selection import GridSearchCV |
| calling Grid Search CV with details  Ex: - model1 = random forest | model2 = GridSearchCV(model1, param\_grid = parameter, cv=5, scoring = "accuracy") |
| Training the model | model2.fit(X\_train,y\_train) |
| choice of the parameter selection by the model | model2.best\_params\_ |
| Prediction | y\_pred = model2.predict(X\_test) |

**Random Forest**

|  |  |
| --- | --- |
| Import regression problem | from sklearn.ensemble import RandomForestRegressor |
| Import Classification Problem | from sklearn.ensemble import RandomForestClassifier |
| Model assigning | model2 = RandomForestRegressor  (n\_estimators = 150) |
| fit (train) | model2. fit(X\_train\_sc, y\_train) |
| Prediction | y\_pred2 = model2.predict(X\_test\_sc) |

**Adaboost**

|  |  |
| --- | --- |
| base model | base\_model2 = DecisionTreeClassifier() |
| Import | from sklearn.ensemble import AdaBoostClassifier |
| adaboost model building | new\_model = AdaBoostClassifier(base\_estimator = base\_model2, n\_estimators = 150,learning\_rate=1.0) |
| fit | new\_model.fit(X\_train\_sc,y\_train) |
| predict | y\_pred = new\_model.predict(X\_test\_sc) |
| classification report | print(classification\_report(y\_test,y\_pred)) |

R2 score

|  |  |
| --- | --- |
| Import r2 score  only for Regression type output data | from sklearn.metrics import r2\_score |
| r2 score | r2\_score(y\_test,y\_pred) |

Classification Report and Accuracy Score

|  |  |
| --- | --- |
| Import accuracy score | from sklearn.metrics import accuracy\_score |
| check accuracy | accuracy\_score(y\_test,y\_pred) |
| Import Confusion Matrix | from sklearn.metrics import confusion\_matrix |
| Confusion Matrix | confusion\_matrix(y\_test,y\_pred) |
| see heat map of confusion matrix | sns.heatmap(name\_of\_matrix, annot=True) |
| Import Classification Report | from sklearn.metrics import classification\_report |
| Classification Report | (classification\_report  (y\_test,y\_pred)) |

1. **Unsupervised ML**

**K means Clustering**

|  |  |
| --- | --- |
| Import | from sklearn.cluster import KMeans |
| loop to get intersection point values in the elbow graph | inter = []  for i in range(1,11):  algo = KMeans(n\_clusters = i)  algo.fit(X)  inter.append(algo.inertia\_) |
| elbow graph | pd.DataFrame(inter).plot()  plt.show() |
| Model building | k\_model = KMeans(n\_clusters = 3)  k\_model.fit(X) |
| clusters | l1 = k\_model.labels\_ |
| Dataset with clusters  Dataset = clusters | df = clusters.assign(cluster = l1) |
| plotting the values with clusters | sns.scatterplot('Age','Spending Score (1-100)',hue = 'cluster', data = clusters,palette = 'bright')  plt.show() |

**PCA**

**Principle Component Analysis**

|  |  |
| --- | --- |
| Normalisation | sc = StandardScaler() |
| Only for Input data: X | x\_sc = sc.fit\_transform(X) |
| Covariance Matrix | cov\_mat = np.cov(x\_sc.T) |
| eig\_vals = eigen values  eig\_vecs = eigen vecters  np.linalg.eig(cov\_mat) = numpy. linear algibra.eigen(covariance matrix) | eig\_vals,eig\_vecs = np.linalg.eig(cov\_mat) |
| Explained Variance (Individual)  tot = variable refers total  list comprehension(var\_exp) | tot = sum(eig\_vals)  var\_exp = [(i/tot)\*100 for i in sorted(eig\_vals,  reverse = True)] |
| Cumulative Variance Explained | cum\_var\_exp = np.cumsum(var\_exp) |
| Plotting Bar plot and Step plot in 1 single plot to visualise the Principal Components to find the n\_components for the PCA | plt.figure(figsize = (10,5)) # size of the plot  plt.bar(range(len(var\_exp)), var\_exp, label = 'individual explained variance', color = 'red')  plt.step(range(len(cum\_var\_exp)), cum\_var\_exp, label = 'cumulative explained variance')  plt.ylabel('Cum Explained Variance')  plt.xlabel('Principle Components')  plt.legend()  plt.show() |
| Data Split | X\_train,X\_test,y\_train,y\_test = train\_test\_split(x\_sc,y,train\_size = 0.8) |
| Import PCA | from sklearn.decomposition import PCA |
| PCA (Principal Component Analysis) | pca = PCA(n\_components = 20)  pca\_X\_train = pca.fit\_transform(X\_train) pca\_X\_test = pca.transform(X\_test) |