NOTES

Ass 6-1

d) Read CSV file :-

- import pandas as pd :-
  - It import pandas library.
  - It is a powerful data manipulat^{n} & analysis library in python.

- df = pd.read_csv ("File Path")
  - df: variable.
  - pd.read_csv: It is a funct^{n} provided by pandas to read csv file into data frames

h) df.shape :-

- Attribute of a pandas Data Frame. it returns a tuple representing the dimensions of the Data Frame
  - 1st element is row & 2nd element is columns.

* Shape :- layout of data.

e) df.dtypes :-

- It returns a series Containing the data types of each Column in the Data Frame

a) - df. isnull(). sum()
- Count of missing values in each column of the data frame

- df['Thal'].mode()
- Find the common values in Thal column of the Data Frame

- df['Thal'].Fillna(str(df['Thal'].mode()), inplace = True)

df['Thal'] : Select the 'Thal' column from DataFrame.

Fillna : It is a method. Used to fill missing value with specified values

inplace = True : This parameter updates the DataFrame directly changing the 'Thal' column in the original DataFrame

str(df['Thal'].mode()) : It calculate the mode of the 'Thal' Column & Convert into string

- inplace = False
- It omitted, it return a new DataFrame with the filled values.

- df['Ca'].fillna(df['Ca'].mean(), inplace = True)

df['Ca'].mean() : It calculates the mean value of the 'Ca' column.

* Inconsistency :- Same data fill in the different table & diff Format

b) df.drop-duplicates (inplace = True)
- Removing Inconsistency
- We need to remove duplicates.

c) - import matplotlib.pyplot as plt :-
- It import MATLAB - like plotting framework

- import numpy as np :-
- It import numpy library
- It is used for mathematical operations & creating arrays.

- plt.boxplot (df.Age) :-
- It creates a box plot for the 'Age' column of the Data Frame

- plt.show () :-
- It display the box plot.

d) - import seaborn as sns :-
- It import seaborn library.
- It is a visualization library.

- sns.histplot ([df.Age], kde = True).

It is used to plot a histogram

It is data that you want to plot a histogram

It estimate the data's probability density funct^{n}

* Histogram :-
- It is a graphical representation of the distribution of numerical data.

g) - mean_age = df['Age'].mean() :-
- Calculate mean

f) zeros_count = (df == 0).sum()

- It creates a DataFrame with 'True' is it equal to 0 & 'False' otherwise.

Add the no of times each column in the DataFrame. It contains the value 0

Ass. 2

Read the file...

- import pandas as pd
  df = pd.read_csv (file path)
  Print (df).

- df. dtypes # Find the data types.

- import numpy as np

- numeric_df = df. Select_dtypes (include = [np. number])
  (include only numeric data)
- print (numeric_df)

This line creates a new DataFrame

a) # Find Std. dev & Varience of every numerical attribute.

- Std_dev = numeric_df. Std () ⟹ Calculate Std. dev
  print (Std_dev)

- var = numeric_df. var () ⟹ Calculate var
  print (var)

b) # Find Covarience & perform Correlation analysis the using Correlation Coefficient.

- Covarience_matrix = numeric_df. Cov ()
  print (Covarience_matrix)

- Correlection_matrix = numeric_df. Corr ()
  Print (Correlection_matrix)

c) # How many independent features are present in the given dataset ?

- independent_features = np.linalg.matrix_rank (numeric_df.corr())

- It is a funct^n.
- It calculate the rank of a matrix, which is the maximum no. of independent columns or rows in the matrix.

It calculates the Correclat^n matrix of the numeric columns in the DataFrame

- print (independent_features)

d) # Unwanted features.

- creates an empty set to hold names of features that might be redundant due to high correlation.

```
unwanted_features = set()
for i in range (len (correlation_matrix.columns)):
    for j in range (i):
        if abs (correlation_matrix.iloc [i,j]) > 0.8:
            colname = correlation_matrix.columns [i]
            unwanted_features.add (colname)
print (unwanted_features)
```

Loop iterates over column indices in Correl^n matrix.

Loop checks previous column indices to avoid repeating comparison.

Checks if Correlat^n coefficient bet^n columns exceeds 0.8

Assigns the name of the column to a variable, representing a highly correlated features.

Adds the name of the highly correlated features.

e) # Perform data discretizatn using equi-frequency binning method on the age attribute.

```
num_bins = 5
df['age_bins'] = pd.qcut(df['Age'], q=num_bins, labels=False)
```

Perform equi-frequency binning on attribute of the Data-Frame

- Set the variable
- It indicate that we want to divide to data into 5 bins

Accesses or creates a column named 'age_bins' in DataFrame 'df' to store discretized value.

Funct^n from pandas library perform quantile-based discretizatn

The no. of quantiles or bins into which to divide the data

o/p: bin indices, not labels, as integers starting from 0

```
import matplotlib.pyplot as plt
plt.figure(figsize=(5, 3))
plt.hist(df['Age'], bins=num_bins, edgecolor='black', alpha=0.7)
plt.title('Equi-Freq. Binning for Age attribute')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

- Data betⁿ 0 & 1 by subtracting the minimum value & dividing by the range

f) from sklearn.preprocessing import MinMaxScaler, StandardScaler

StandardScaler → It transforms data to have mean 0 & std. dev. 1

```
attributes_to_normalize = ['RestBP', 'Chol', 'MaxHR']

# Min-Max normalizatⁿ

min_max_scaler = MinMaxScaler()   → object.

df_min_max_normalized = df.copy()   ← create duplicate of the DataFrame

df_min_max_normalized[attributes_to_normalize] =
        min_max_scaler.fit_transform
        (df[attributes_to_normalize])

df_min_max_normalized[attributes_to_normalize]
```

Apply Min-Max normalizatⁿ to selected columns & stores the normalized values in 'df_min_max_normalized'

Selects specific columns from 'df_min_max_normalized'

```
# Z-score normalization.

z_score_scaler = StandardScaler()

df_z_score_normalized = df.copy()

df_z_score_normalized[attributes_to_normalize]=
        z_score_scaler.fit_transforms
            (df[attributes_to_normalize])

df_z_score_normalized[attributes_to_normalize]
```

10**(len(...)-1) Calculate decimal scaling factor.

```
decimal_scaling_factor = 10 ** (len(str(int(df
[attributes_to_normalize].abs().max().max())))
-1)

df_decimal_scaled = df.copy()

df_decimal_scaled[attributes_to_normalize] =
df[attributes_to_normalize]/decimal_scaling_factor

df_decimal_scaled[attributes_to_normalize]
```

Counts digits in the max value by converting it to integer & a string, & calculates the length of that string.

ASS 4

* WEKA :-
  - It stand for Waskato Environment for knowledge Analysis.
  - It is open source s/w.
  - It provides tools for data preprocessing
  - It is used to preprocess the big data, apply different machine learning algo. on big data & compare various o/p
  - This s/w is easy to work with big data & train a machine using machine learning algo.

* Advantage :-
  - Testing new ides quickly.
  - Easy to learn & solve data mining problem.
  - Code is not required ; everything can be done through the UI.

* Disadvantage :-
  - It can only handle small datasets
  - lack of algo. for deep learning
  - The compilication in converting the string into numeric math.
  - Alternative algo. are not available

* Clustering :-
  - It is groups similar data points together based on characteristics

Counts digits in the max value by converting it to integers & a string & calcul the leng of that string

Supervised & - independent data.

* Simple k-means Clustering :- data is dependent on operⁿ

- k-means clustering is an unsupervised learning algo.
- It is used to solve the clustering problems in machine learning or data science.
- k defines the no of pre-defined clusters that need to be created in the process.

* Advantage :-
- Simplicity & ease to use
- Scalability
- Efficiency & Speed

* Disadvantage :-
- Not suitable for non-linear data
- Dependence on no of clusters
- Sensitive to outliers

* Hierarchical clustering :-
- It creates a hierarchical representation of the clusters in a dataset.
- It is an unsupervised learning
- It starts by treating each data points as an individual cluster.

* Advantage :-
- Simple to implement
- Gives the best o/p in some cases.

* Disadvantage :-
  - Breaks the large clusters.
  - Sensitive to noise & outliers.
  - Difficult to handle different sized clusters.

ASS. 5

* Naive Bayes :-
  - It is a collectn of classificatn algo based on Bayes Theorem.
  - It is a family of algo. where all of them share common principles.

* Random forest :-
  - It is a powerful tree learning technique in machine learning.
  - It is a creating a no. of decision trees during the training phase
  - The greater no. of trees in the forest leads to higher accuracy.

* Decision tree :-
  - It is a supervised learning technique.
  - It is used both classificatn & Regression problem
  - But mostly it is preferred for solving classificatn problem.

Ass 6

* Theory same as Ass 5

Read the dataset

It is used to encode categorical labels with numerical values

- From sklearn.preprocessing (module) import LabelEncoder (class)
encoder = LabelEncoder()
data['chestPain'] = encoder.fit_transform(data['chestPain'])
print(data)

pick column from the Data Frame

- It create instance.
- Used to encode the categorical data.
- Categorical labels into numerical values using LabelEncoder
- Update the Data Frame accordingly

- data['Thal'] = encoder.fit_transform(data['Thal'])
print(data).

Same for AHD

- From sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
From sklearn.svm import SVC
From sklearn.neighbours import kNeighboursClassifier
From sklearn.metrics import accuracy_score

- data.isnull().sum()

- data['Ca'].fillna(data['Ca'].mean(), inplace=True)

- x = data.drop(columns=['AHD'])
  ↳ Creates new DataFrame by dropping column named 'AHD' from dataFrame.

- y = data['AHD']

- x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
  - x_train, x_test, y_train, y_test → Variable storing training & testing sets for features (x) & target variable (y).
  - test_size=0.2 → 20% of data used for testing.
  - random_state=42 → Set random speed for reproducibility.

- dt_classifier = DecisionTreeClassifier()
- svm_classifier = SVC()
- knn_classifier = kNeighboursClassifier()

SVC →
Support vector Classifier.
- It is a specific implement of the support vector machine algo.
- It is a supervised learning algo.
- It makes classificat^n based on data neighbors.

- dt_classifier.fit(x_train, y_train)
- svm_classifier.fit(x_train, y_train)
- knn_classifier.fit(x_train, y_train)

→ train the decision tree classifier.
→ train the svm (support vector machine) classifier.
→ train the k-nearest neighbours classifier.

- dt_predictions = dt_classifier.predict(x_test)
- svm_predictions = svm_classifier.predict(x_test)
- knn_predictions = knn_classifier.predict(x_test)

→ Generate predictions using trained decision tree.
→ same for all.

- dt_accuracy = accuracy_score(y_test, dt_predictions)
- svm_ ——— " ——— , svm_ —"—
- knn_ ——— " ——— , knn_ —"—

→ Calculate accuracy of predictions made by decision tree classifier.
→ same for all.

```
print(dt_accuracy)
print(svm_accuracy)
print(knn_accuracy)
```

—x x x—