Practical Assignment 1:

Linear and Logistic Regression

1. Create 'sales' Data set having 5 columns namely: ID, TV, Radio, Newspaper and Sales.(random 500 entries) Build a linear regression model by identifying independent and target variables. Split the variables into training and testing sets. then divide the training and testing sets into a 7:3 ratio, respectively and print them. Build a simple linear regression model.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt

# 1. Create dataset
np.random.seed(42)

data = pd.DataFrame({
    'ID': np.arange(1, 501),
    'flat': np.random.randint(1, 100, 500),       # Flats available
    'houses': np.random.randint(1, 50, 500),      # Houses available
})

# 2. Generate 'purchases' with some correlation
data['purchases'] = (
    1.5 * data['flat'] +
    2.8 * data['houses'] +
    np.random.normal(0, 10, 500)   # Noise
)

# 3. Define independent (X) and target (y) variables
X = data[['flat', 'houses']]
y = data['purchases']

# 4. Split dataset into 70% training and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)

# 5. Print training and testing sets
print("Training Features (X_train):\n", X_train.head(), "\n")
print("Training Target (y_train):\n", y_train.head(), "\n")
print("Testing Features (X_test):\n", X_test.head(), "\n")
print("Testing Target (y_test):\n", y_test.head(), "\n")

# 6. Build and train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# 7. Print model details
print("\nIntercept:", model.intercept_)
print("Coefficients (flat, houses):", model.coef_)

# 8. Predict on test set
y_pred = model.predict(X_test)
```

```
# 9. Evaluate model performance
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"\nR² Score: {r2:.4f}")
print(f"RMSE: {rmse:.4f}")

# 10. Visualization: Actual vs Predicted Purchases
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
color='red', linewidth=2)
plt.title("Actual vs Predicted Purchases")
plt.xlabel("Actual Purchases")
plt.ylabel("Predicted Purchases")
plt.grid(True)
plt.tight_layout()
plt.show()
```

2. Create 'real_estate' Data set having 4 columns namely: ID,flat, houses and purchases (random 500 entries). Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets and print them. Build a simple linear regression model for predicting purchases.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt

# 1. Create dataset
np.random.seed(42)

data = pd.DataFrame({
    'ID': np.arange(1, 501),
    'flat': np.random.randint(1, 100, 500),        # Flats available
    'houses': np.random.randint(1, 50, 500),       # Houses available
})

# 2. Generate 'purchases' with some correlation
data['purchases'] = (
    1.5 * data['flat'] +
    2.8 * data['houses'] +
    np.random.normal(0, 10, 500)   # Noise
)

# 3. Define independent (X) and target (y) variables
X = data[['flat', 'houses']]
y = data['purchases']

# 4. Split dataset into 70% training and 30% testing
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)

# 5. Print training and testing sets
print("Training Features (X_train):\n", X_train.head(), "\n")
print("Training Target (y_train):\n", y_train.head(), "\n")
print("Testing Features (X_test):\n", X_test.head(), "\n")
print("Testing Target (y_test):\n", y_test.head(), "\n")

# 6. Build and train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# 7. Print model details
print("\nIntercept:", model.intercept_)
print("Coefficients (flat, houses):", model.coef_)

# 8. Predict on test set
y_pred = model.predict(X_test)

# 9. Evaluate model performance
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"\nR² Score: {r2:.4f}")
print(f"RMSE: {rmse:.4f}")

# 10. Visualization: Actual vs Predicted Purchases
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
color='red', linewidth=2)
plt.title("Actual vs Predicted Purchases")
plt.xlabel("Actual Purchases")
plt.ylabel("Predicted Purchases")
plt.grid(True)
plt.tight_layout()
plt.show()
```

3. Build a simple linear regression model for Fish Species Weight Prediction. (download dataset https://www.kaggle.com/aungpyaeap/fish-market?select=Fish.csv

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

# 1. Load dataset
# Replace 'Fish.csv' with your local path or the raw URL
data = pd.read_csv('Fish.csv')

# 2. Select feature and target
X = data[['Length3']]  # Independent variable
y = data['Weight']     # Target variable

# 3. Split into training (70%) and testing (30%) sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

# 4. Print sample data
print("Training feature sample:\n", X_train.head(), "\n")
print("Training target sample:\n", y_train.head(), "\n")
print("Testing feature sample:\n", X_test.head(), "\n")
print("Testing target sample:\n", y_test.head(), "\n")

# 5. Build and train a simple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# 6. Model output
print("Intercept:", model.intercept_)
print("Coefficient for Length3:", model.coef_[0])

# 7. Make predictions on test set
y_pred = model.predict(X_test)

# 8. Evaluate model performance
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"R² Score: {r2:.4f}")
print(f"RMSE: {rmse:.4f}")

# 9. Visualize actual vs predicted weight
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', alpha=0.6, label='Actual
weight')
plt.plot(X_test, y_pred, color='red', linewidth=2,
label='Prediction')
plt.title("Fish Weight Prediction: Actual vs Predicted (Length3)")
plt.xlabel("Length3")
plt.ylabel("Weight")
plt.legend()
```

```
plt.grid(True)
plt.tight_layout()
plt.show()
```

Practical Assignment 2:

Frequent itemset and Association rule mining

1. Download the Market basket dataset.
Write a python program to read the dataset and dis play its information.
Preprocess the data (drop null values etc.)
Convert the categorical values into numeric format.
Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.

```
# Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Step 1: Load the dataset
# Replace 'Market_Basket.csv' with your file path
data = pd.read_csv("basket.csv")

# Step 2: Display dataset information
print("Dataset Information:")
print(data.info())
print("\nFirst 5 Rows:")
print(data.head())

# Step 3: Preprocess data
# Drop null values
data.dropna(inplace=True)

# Step 4: Convert categorical values into numeric format
# Convert transaction data into one-hot encoding
basket = pd.get_dummies(data)

print("\nAfter Encoding:")
print(basket.head())

# Step 5: Apply Apriori Algorithm
# Generate frequent itemsets with minimum support
frequent_itemsets = apriori(basket, min_support=0.05,
use_colnames=True)

print("\nFrequent Itemsets:")
print(frequent_itemsets)

# Step 6: Generate association rules
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1.0)
```

```
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 
'lift']])
```

Practical Assignment 3 :

Text and Social Media Analytics

1. Consider any text paragraph. Preprocess the text to remove any special characters and digits. Generate the summary using the extractive summarization process.

```
import re
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from collections import defaultdict
from heapq import nlargest
def summarize_text(text, num_sentences=3):
"""
Generates an extractive summary of a text after preprocessing.
Args:
text (str): The input text paragraph.
num_sentences (int): The desired number of sentences in the summary.
Returns:
str: The generated extractive summary.
"""
# Step 1: Preprocess the text to remove special characters and digits
cleaned_text = re.sub(r'[^a-zA-Z\s.]', '', text, re.I|re.A)
cleaned_text = re.sub(r'\s+', ' ', cleaned_text).strip()
# Handle cases where all text is removed
if not cleaned_text or cleaned_text.isspace():
return "Not enough text to generate a summary."
# Tokenize the text into sentences
sentences = sent_tokenize(cleaned_text)
if len(sentences) <= num_sentences:
return cleaned_text # Return original text if it's already short
# Tokenize the text into words and remove stopwords
stop_words = set(stopwords.words('english'))
word_frequencies = defaultdict(int)
for word in word_tokenize(cleaned_text):
if word.lower() not in stop_words and word.isalpha():
word_frequencies[word.lower()] += 1
# Check if word_frequencies is empty
if not word_frequencies:
return "Not enough meaningful words to generate a summary."
# Normalize the word frequencies
max_frequency = max(word_frequencies.values())
for word in word_frequencies.keys():
word_frequencies[word] = (word_frequencies[word]/max_frequency)

# Step 2: Score sentences based on word frequency
sentence_scores = defaultdict(int)
```

```
for sentence in sentences:
for word in word_tokenize(sentence):
if word.lower() in word_frequencies:
sentence_scores[sentence] += word_frequencies[word.lower()]
# Step 3: Extract the top N sentences for the summary
summary_sentences = nlargest(num_sentences, sentence_scores, key=sentence_scores.get)
# Join the sentences to form the summary
summary = ' '.join(summary_sentences)
return summary
# Example usage with a sample paragraph
sample_paragraph = """
Artificial intelligence (AI) is a rapidly advancing field of computer science.
Its applications range from simple chatbots to complex autonomous vehicles.
AI systems are designed to perform tasks that would normally require human intelligence,
such as visual perception, speech recognition, and decision-making.
In recent years, the development of deep learning models has significantly boosted AI's
capabilities, leading to major breakthroughs.
The ethical implications of AI, however, are a subject of ongoing debate. Some researchers
believe AI could solve some of the world's most complex problems.
While others worry about its potential impact on employment and data privacy.
The future of AI holds great promise and many challenges, with billions of dollars being
invested in research and development each year.
"""
# Generate a summary with 3 sentences
generated_summary = summarize_text(sample_paragraph, num_sentences=3)
print("Original Text:")
print(sample_paragraph)
print("\n--- Extractive Summary ---")
print(generated_summary)
```

2.Consider review message .Perform sentiment analysis on the message .

```
import numpy as np
import pandas as pd
#import nltk
import matplotlib.pyplot as plt
airtweets = pd.read_csv("Tweets.csv")
plsize plt.rcParams["figure.figsize"]
plt.rcParams["figure.figsize"] = plsize
print("Data in CSV file")
print(airtweets.head())
print("Distribution of sentiments across all tweets ")
airtweets.airline_sentiment.value_counts().plot(kind='pie', autopct='%1.
0f%%', colors=["Black", "Orange", "green"])
airline_sentiment = airtweets.groupby(['airline', 'airline_sentiment']).
airline_sentiment.count().unstack()
airline_sentiment.plot(kind='bar',color=['black', 'blue', 'cyan'])
```