Bahria University,

Karachi Campus

A logo with text on it

Description automatically generated

LAB EXPERIMENT NO.

11

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **01** | Web Scraping and Text Analysis  Develop a Python script to scrape textual data from websites and perform sentiment analysis on this data. This could involve analyzing customer reviews, forum discussions, or news articles. |
| **02** | Analyzing Publicly Available APIs  Use publicly available APIs (like Twitter API, OpenWeatherMap, or others) to fetch data and analyze specific trends or patterns. |
| 03 | Data Mining From Files.  Perform data mining from a collection of local files (like logs, text files, or CSVs) to extract meaningful information without using a database. |

Submitted On:

15-05-2024

(Date: DD/MM/YYYY)

**Task No. 01:** Web Scraping and Text Analysis

Develop a Python script to scrape textual data from websites and perform sentiment analysis on this data. This could involve analyzing customer reviews, forum discussions, or news articles.

**Requirement Steps:**

* Use requests and BeautifulSoup for web scraping to extract data.
* Clean and preprocess the data using libraries like nltk or spacy.
* Perform sentiment analysis using a pre-trained model from textblob or build your own classifier with scikit-learn.
* Visualize the results using matplotlib or seaborn.

**Solution:**

import requests

from bs4 import BeautifulSoup

import nltk

from textblob import TextBlob

import matplotlib.pyplot as plt

import streamlit as st

nltk.download('punkt')

def scrape\_pakwheels(url):

    response = requests.get(url)

    soup = BeautifulSoup(response.text, 'html.parser')

    reviews = soup.find\_all('div', dir='ltr')

    review\_texts = []

    for review in reviews:

        review\_text = review.get\_text(strip=True)

        review\_texts.append(review\_text)

    return review\_texts

def clean\_text(text):

    tokens = nltk.word\_tokenize(text)

    tokens = [token.lower() for token in tokens if len(token) > 1]

    return ' '.join(tokens)

def analyze\_sentiment(text):

    blob = TextBlob(text)

    polarity = blob.sentiment.polarity

    if polarity > 0:

        sentiment = 'Positive'

    elif polarity < 0:

        sentiment = 'Negative'

    else:

        sentiment = 'Neutral'

    return polarity, sentiment

def plot\_sentiments(sentiments):

    plt.hist(sentiments, bins=20, edgecolor='black')

    plt.title('Sentiment Analysis of PakWheels Reviews')

    plt.xlabel('Sentiment Polarity')

    plt.ylabel('Number of Reviews')

    st.pyplot()

def main():

    st.title('PakWheels Reviews Sentiment Analysis')

    # Input URL

    url = st.text\_input('Enter PakWheels URL for Reviews:')

    if st.button('Analyze Reviews'):

        review\_texts = scrape\_pakwheels(url)

        cleaned\_reviews = [clean\_text(review) for review in review\_texts]

        sentiments = [analyze\_sentiment(review) for review in cleaned\_reviews]

        # Display sentiments

        st.subheader('Sentiment Analysis Results:')

        for i, (text, (polarity, sentiment)) in enumerate(zip(review\_texts, sentiments)):

            with st.expander(f"Comment {i+1} - Sentiment: {sentiment}, Polarity: {polarity}"):

                st.write(text)

        # Plot sentiments

        st.subheader('Sentiment Polarity Distribution:')

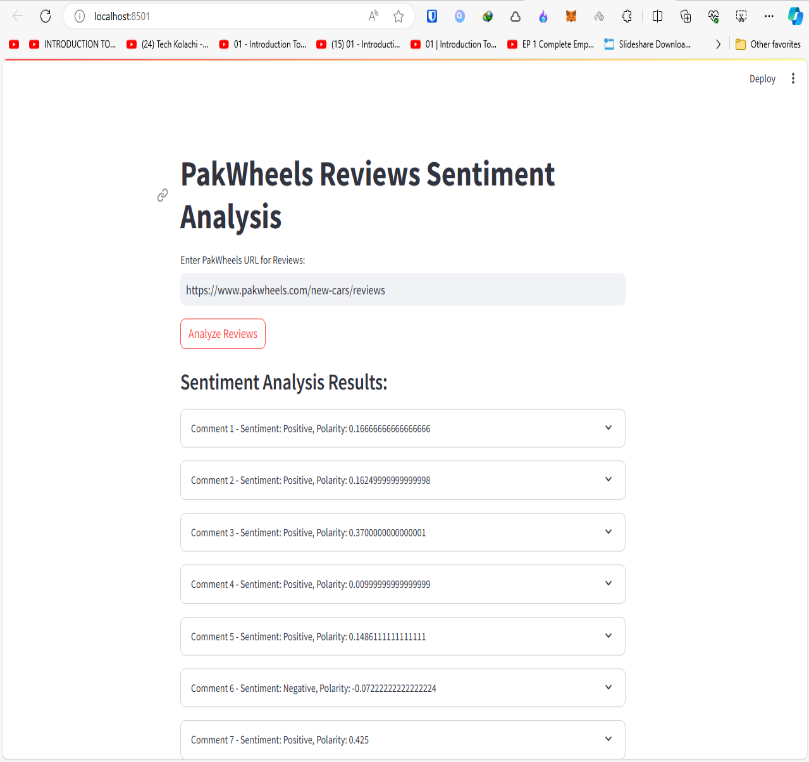
        plot\_sentiments([polarity for polarity, \_ in sentiments])

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Output:**

A screenshot of a computer

Description automatically generated

A graph of blue bars

Description automatically generated with medium confidence

**Task No. 02:** Analyzing Publicly Available APIs

Use publicly available APIs (like Twitter API, OpenWeatherMap, or others) to fetch data and analyze specific trends or patterns.

**Requirements Steps:**

* Fetch data from APIs using the requests library.
* Analyze data to find trends (e.g., frequency of specific words on Twitter, changes in weather patterns).
* Plot data using visualization libraries to interpret these trends.

**Solution:**

import streamlit as st

import requests

import pandas as pd

import matplotlib.pyplot as plt

# OpenWeatherMap API Key

API\_KEY = '8b860b511548557e8bd3725ba45170cc'

# Function to fetch and visualize weather data

def fetch\_and\_visualize\_weather\_data(city):

    url = f"http://api.openweathermap.org/data/2.5/forecast?q={city}&appid={API\_KEY}&units=metric"

    response = requests.get(url)

    data = response.json()

    if data['cod'] != '200':

        st.error("Failed to fetch data. Please check the city name or try again later.")

        return

    weather\_list = data['list']

    parsed\_data = {

        "date": [],

        "temperature": [],

        "humidity": [],

        "weather": []

    }

    for entry in weather\_list:

        parsed\_data["date"].append(entry['dt\_txt'])

        parsed\_data["temperature"].append(entry['main']['temp'])

        parsed\_data["humidity"].append(entry['main']['humidity'])

        parsed\_data["weather"].append(entry['weather'][0]['description'])

    df = pd.DataFrame(parsed\_data)

    # Plot data

    fig, ax = plt.subplots(2, 1, figsize=(10, 8), sharex=True)

    ax[0].plot(df['date'], df['temperature'], marker='o', linestyle='-')

    ax[0].set\_title('Temperature Over Time')

    ax[0].set\_ylabel('Temperature (°C)')

    ax[0].tick\_params(axis='x', rotation=45)

    ax[1].plot(df['date'], df['humidity'], marker='o', linestyle='-', color='orange')

    ax[1].set\_title('Humidity Over Time')

    ax[1].set\_ylabel('Humidity (%)')

    ax[1].tick\_params(axis='x', rotation=45)

    plt.tight\_layout()

    st.pyplot(fig)

    # Display DataFrame

    st.write(df)

# Streamlit app

st.title('Weather Data Analysis')

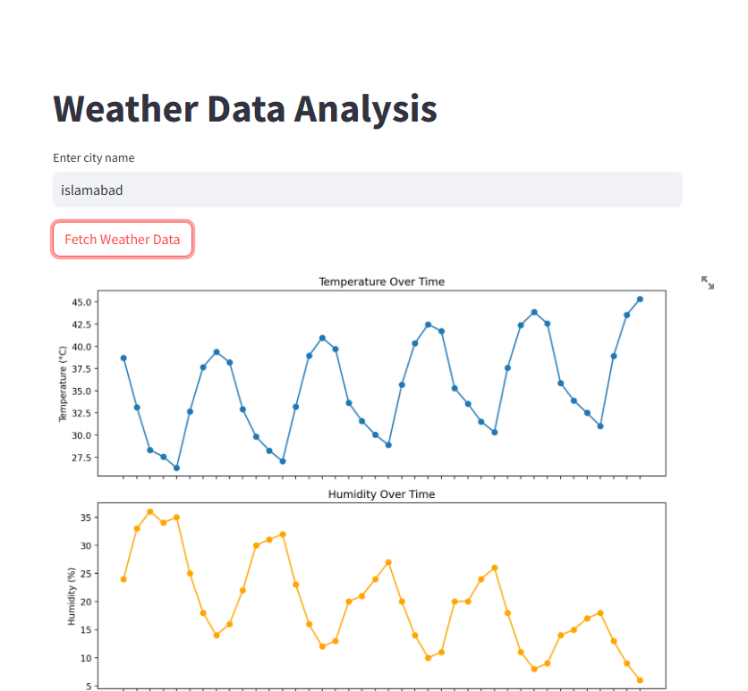
# Input city

city = st.text\_input('Enter city name', 'New York')

if st.button('Fetch Weather Data'):

    fetch\_and\_visualize\_weather\_data(city)

    st.balloons()

**Output:**

A screenshot of a weather report

Description automatically generated

**Task No. 03:** Data Mining From Files.

Perform data mining from a collection of local files (like logs, text files, or CSVs) to extract meaningful information without using a database.

**Requirements Steps:**

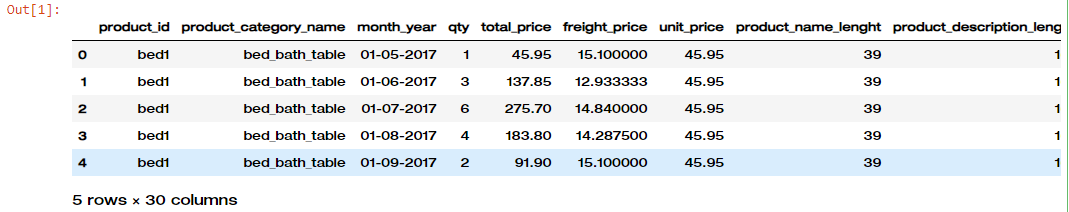
* Read data from various file formats using Python (csv, open, pandas).
* Process and analyze data to detect anomalies, patterns, or generate summaries.
* Use statistical methods or machine learning models to derive insights.
* Visualize the findings using appropriate Python libraries.

**Solution:**

import pandas as pd

file\_path = 'retail\_price.csv'

data = pd.read\_csv(file\_path)



data.head()

missing\_values = data.isnull().sum()

data['month\_year'] = pd.to\_datetime(data['month\_year'], format='%d-%m-%Y')

data.dtypes, missing\_values

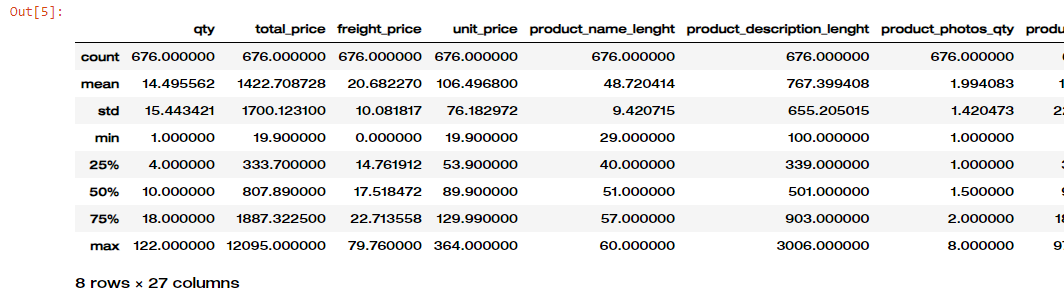
A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

summary = data.describe()

summary

import matplotlib.pyplot as plt

import seaborn as sns

# Set up the plotting style

sns.set(style='whitegrid')

# Plot total sales over time

plt.figure(figsize=(12, 6))

sns.lineplot(data=data, x='month\_year', y='total\_price', estimator='sum')

plt.title('Total Sales Over Time')

plt.xlabel('Date')

plt.ylabel('Total Sales')

plt.show()

# Plot quantity sold over time

plt.figure(figsize=(12, 6))

sns.lineplot(data=data, x='month\_year', y='qty', estimator='sum')

plt.title('Quantity Sold Over Time')

plt.xlabel('Date')

plt.ylabel('Quantity Sold')

plt.show()

A graph showing a line

Description automatically generated

A graph showing a line

Description automatically generated

from scipy import stats

# Calculate z-scores

data['z\_score\_total\_price'] = stats.zscore(data['total\_price'])

data['z\_score\_qty'] = stats.zscore(data['qty'])

# Detect anomalies

anomalies = data[(abs(data['z\_score\_total\_price']) > 3) | (abs(data['z\_score\_qty']) > 3)]

anomalies.head()

A table with numbers and text

Description automatically generated

plt.figure(figsize=(14, 10))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Matrix Heatmap')

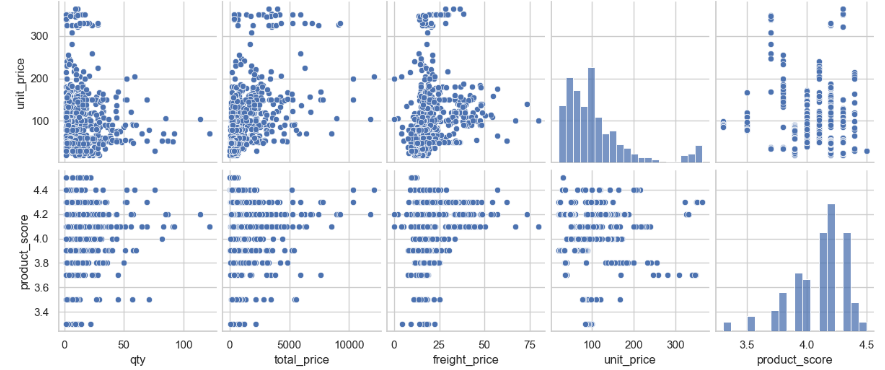
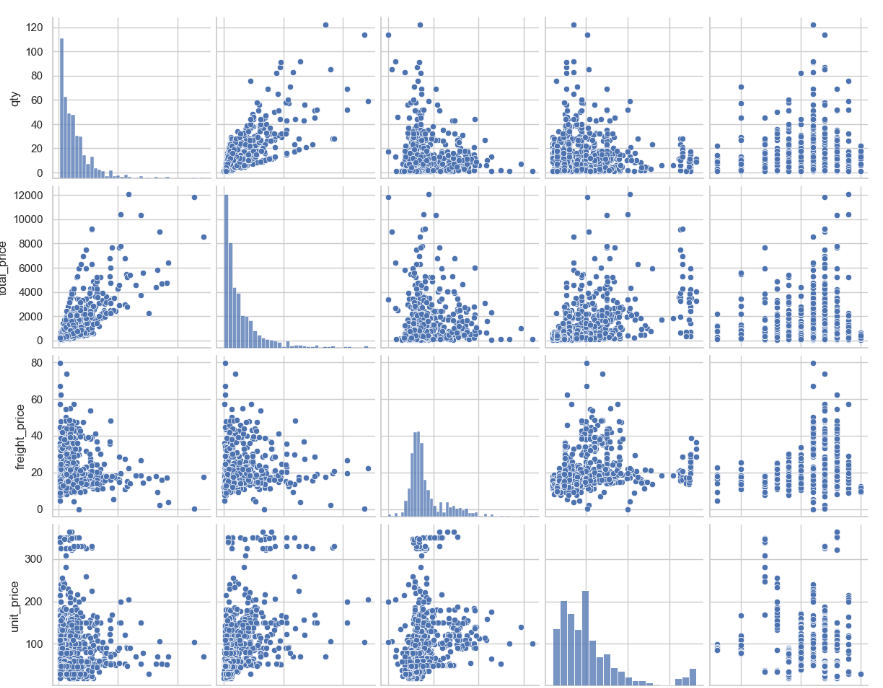
plt.show()

key\_vars = ['qty', 'total\_price', 'freight\_price', 'unit\_price', 'product\_score']

sns.pairplot(data[key\_vars])

plt.suptitle('Pairplot of Key Variables', y=1.02)

plt.show()



from sklearn.cluster import KMeans

features = data[['qty', 'total\_price', 'freight\_price', 'unit\_price', 'product\_score']]

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(features)

# Apply KMeans clustering

kmeans = KMeans(n\_clusters=3, random\_state=42)

clusters = kmeans.fit\_predict(scaled\_features)

data['cluster'] = clusters

# Visualize clusters

plt.figure(figsize=(12, 6))

sns.scatterplot(data=data, x='total\_price', y='qty', hue='cluster', palette='viridis')

plt.title('Clusters based on Total Price and Quantity Sold')

plt.xlabel('Total Price')

plt.ylabel('Quantity Sold')

plt.show()

**OUTPUT:**

A graph showing different colored dots

Description automatically generated