**Experiment 1: Web Scraping & API Handling**

Python

```
from bs4 import BeautifulSoup
import requests
import json


# ==========================================
# PART 1: Web Scraping using BeautifulSoup
# ==========================================
print('--- PART 1: Web Scraping Output ---')


# Sample HTML document to parse
html_doc = """<html><head><title>The Dormouse's story</title></head>
<body>
  <p class ="title"><b>The Dormouse's story</b></p>
  <p class ="story">One Upon a time there were three litle sisters; and their names were
    <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
    <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
    <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
    and they lived at the bottom of the well.
  </p>
  <p calss="story">...</p>
</body>
</html>"""


# Parsing the HTML content
soup = BeautifulSoup(html_doc, 'html.parser')
```

```python
# Displaying the formatted HTML structure
print(soup.prettify())

# Accessing specific tags from the HTML
print('\nPage Title:', soup.title.string)
print('Body Content:', soup.body.text.strip())

# ============================================
# PART 2: API Handling using Requests
# ============================================
print('\n--- PART 2: API Handling Output ---')

# Making a GET request to the GitHub API
response = requests.get('https://api.github.com')

# Checking the status code (200 means success)
print('Status Code:', response.status_code)

# Printing the keys of the JSON response to verify data
print('Response JSON Keys:', response.json().keys())
```

---

**Experiment 2: Handling HTTP Requests**

Python

```python
import requests

# ============================================
# Handling HTTP Requests
# ============================================
```

```
# Target URL
url = 'https://abhikdas.me'

try:
    # Making a GET request to the website
    print(f'Sending GET request to {url}...')
    r = requests.get(url)

    # Check the status code of the response
    # 200 indicates a successful request
    print('Response Object:', r)
    print('Status Code:', r.status_code)

    # Accessing and printing the raw HTML content
    # (Printing first 500 characters to keep output clean)
    print('\nResponse Content (First 500 chars):')
    print(r.text[:500])

except requests.exceptions.RequestException as e:
    # Handling errors (e.g., connection issues)
    print(f'An error occurred: {e}')
```

---

**Experiment 3: Data Cleaning (Part 1 - Sample Data)**

Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# =============================================

# PART 1: Data Cleaning on Sample Data

# =============================================


# Step 1: Create a sample dataset with missing values

data = {

    'Name': ['John', 'Mary', 'David', 'Emily', 'Michael'],

    'Age': [25, 31, np.nan, 42, 28],

    'City': ['New York', 'Los Angeles', 'Chicago', np.nan, 'Houston']

}


df = pd.DataFrame(data)

print('Original DataFrame:')

print(df)


# Step 2: Data Exploration

print('\nMissing Values:')

print(df.isnull().sum())


print('\nData Types:')

print(df.dtypes)


print('\nSummary Statistics:')

print(df.describe())


# Step 3: Imputation

# Impute missing values in 'Age' column with Mean
```

```
df['Age'].fillna(df['Age'].mean(), inplace=True)
print('\nImputed Dataset (Age Mean):')
print(df)

# Reset Data for Median Imputation Demo
mf = pd.DataFrame(data)
mf['Age'].fillna(mf['Age'].median(), inplace=True)
print('\nImputed Dataset (Age Median):')
print(mf)

# Impute missing values in 'City' column with Mode
df['City'].fillna(df['City'].mode()[0], inplace=True)
print('\nImputed Dataset (City Mode):')
print(df)

# Impute missing values using Interpolation
iff = pd.DataFrame(data)
iff['Age'].interpolate(method='linear', limit_direction='forward', inplace=True)
print('\nImputed Dataset (Interpolation):')
print(iff)

# Step 4: Visualization
# Plotting Age Distribution
plt.figure(figsize=(8, 4))
plt.hist(df['Age'], bins=10, color='pink', alpha=0.7, edgecolor='red')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
```

```python
plt.show()


# Plotting City Distribution
city_counts = df['City'].value_counts()
plt.figure(figsize=(8, 4))
plt.bar(city_counts.index, city_counts.values, color='skyblue', alpha=0.7, edgecolor='blue')
plt.title('City Distribution')
plt.xlabel('City')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```

---

**Experiment 3: Data Cleaning (Part 2 - Titanic Dataset)**

Python

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# ==========================================
# PART 2: Cleaning Titanic Dataset
# Note: Requires 'train.csv' and 'test.csv'
# ==========================================


# Step 1: Load Datasets
try:
    train = pd.read_csv('train.csv')
    test = pd.read_csv('test.csv')
```

```
print('Train Dataset Head:')
print(train.head())

# Step 2: Exploration
print('\nTrain Data Info:')
print(train.info())

print('\nMissing Values (Train):')
print(train.isnull().sum())
print('\nMissing Values (Test):')
print(test.isnull().sum())

# Step 3: Imputation
# Filling missing 'Age' with Mean
train['Age'].fillna(train['Age'].mean(), inplace=True)
test['Age'].fillna(test['Age'].mean(), inplace=True)

# Interpolating 'Cabin' column
train['Cabin'].interpolate(method='linear', inplace=True)
test['Cabin'].interpolate(method='linear', inplace=True)

# Filling 'Embarked' with Mode
train['Embarked'].fillna(train['Embarked'].mode()[0], inplace=True)

# Filling remaining 'Cabin' NaNs with Mode (if any left after interpolate)
train['Cabin'].fillna(train['Cabin'].mode()[0], inplace=True)
test['Cabin'].fillna(test['Cabin'].mode()[0], inplace=True)
```

```
    print('\nMissing Values After Cleaning (Train):')
    print(train.isnull().sum())

    # Step 4: Visualization
    # Age Distribution (Train)
    plt.figure(figsize=(8, 4))
    plt.hist(train['Age'], bins=30, edgecolor='black')
    plt.title('Age Distribution in Training Set')
    plt.xlabel('Age')
    plt.ylabel('Frequency')
    plt.show()

    # Fare Distribution (Train)
    plt.figure(figsize=(8, 4))
    plt.hist(train['Fare'], bins=30, edgecolor='black')
    plt.title('Fare Distribution')
    plt.xlabel('Fare')
    plt.ylabel('Frequency')
    plt.show()

except FileNotFoundError:
    print("Error: 'train.csv' or 'test.csv' not found. Please upload the datasets.")
```

---

**Experiment 6: Apriori Algorithm**

Python

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
```

```
from mlxtend.preprocessing import TransactionEncoder

# Sample Dataset (List of Transactions)
dataset = [['shrimp', 'almonds', 'avocado', 'vegetables mix', 'green grapes'],
    ['burgers', 'meatballs', 'eggs'],
    ['chutney'],
    ['turkey', 'avocado'],
    ['mineral water', 'milk', 'energy bar', 'whole wheat rice', 'green tea']]

# Transaction Encoder to convert list to One-Hot encoded boolean format
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Generate frequent itemsets using Apriori
frequent_itemsets = apriori(df, min_support=0.05, use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets.head())

# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
print("\nAssociation Rules:")
print(rules.head())
```

---

**Experiment 7: FP-Growth Algorithm**

Python

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
```

```
from mlxtend.frequent_patterns import fpgrowth, association_rules

# Load and Preprocess Data (Assuming df_encoded is ready from Exp 6 logic)
# For demo, we reuse the encoded dataframe logic
# In a real run, ensure 'df_encoded' is available

# ---- FP-Growth Algorithm ----
frequent_itemsets_fpgrowth = fpgrowth(df_encoded, min_support=0.4, use_colnames=True)

# Generate Rules
rules_fpgrowth = association_rules(frequent_itemsets_fpgrowth, metric="confidence", min_threshold=0.5)

# Function to Calculate Accuracy
N = len(df_encoded)
def calc_accuracy(row):
    support_AB = row['support'] * N
    support_A = df_encoded[list(row['antecedents'])].all(axis=1).sum()
    return (support_AB + (N - support_A)) / N

rules_fpgrowth["accuracy"] = rules_fpgrowth.apply(calc_accuracy, axis=1)

print("\n---- FP-Growth Rules with Accuracy ----")
print(rules_fpgrowth[['antecedents','consequents','support','confidence','lift','accuracy']])
```

---

**Experiment 8: Comparison Metrics**

Python

```
# Assuming rules_apriori and rules_fpgrowth are generated
```

```
metrics_cols = ['antecedents', 'consequents', 'support', 'confidence', 'lift', 'leverage',
'conviction']

print("---- Apriori Rules Metrics ----")
print(rules_apriori[metrics_cols])

print("\n---- FP-Growth Rules Metrics ----")
print(rules_fpgrowth[metrics_cols])
```

---

**Experiment 9: Neural Networks**

Python

```
import matplotlib.pyplot as plt

# (Assuming 'model' is trained and 'history' object exists)
# Evaluating the model on test set
loss, mae = model.evaluate(X_test, y_test)
print(f'ANN Test MAE: {mae:.2f}')

# Plotting Training & Validation Loss
plt.figure(figsize=(12, 5))

# Loss Plot
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
```

```
plt.ylabel('MSE Loss')
plt.legend()
plt.grid(True)

# MAE Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Training MAE', color='green')
plt.plot(history.history['val_mae'], label='Validation MAE', color='red')
plt.title('MAE over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.grid(True)
plt.show()
```

---

**Experiment 10: Choropleth Maps**

Python

```
import plotly.express as px
import pandas as pd

# Sample world dataset (GDP per capita by country)
data = {'country': ['United States', 'India', 'China', 'Germany', 'Brazil',
             'Canada', 'Russia', 'South Africa', 'Japan', 'Australia'],
    'gdp_per_capita': [65000, 2100, 12000, 48000, 9000,
                 52000, 11500, 6000, 42000, 55000]}
df = pd.DataFrame(data)

# Create the choropleth map
```

```
fig = px.choropleth(df,
  locations='country',       # Name of countries
  locationmode='country names', # Match using full country name
  color='gdp_per_capita',      # Column to color by
  color_continuous_scale='Viridis',
  title='GDP per Capita by Country (Sample Data)',
  labels={'gdp_per_capita': 'GDP per Capita (USD)'}
)

fig.show()
```

---

**Experiment 13: GeoJSON Visualization**

Python

```
import geopandas as gpd
import folium
import matplotlib.pyplot as plt
from folium.features import GeoJsonTooltip
from IPython.display import IFrame, display

# Step 3: Load the uploaded GeoJSON file
geojson_path = "/content/india_states.geojson"  # uploaded file path
gdf = gpd.read_file(geojson_path)

print("✅ GeoJSON file loaded successfully.")
print("Number of features:", len(gdf))
print("Columns available:", list(gdf.columns))

# Step 4: Quick preview of the GeoDataFrame
```

```
display(gdf.head())

# Step 5: Simple static plot using GeoPandas + Matplotlib
plt.figure(figsize=(10, 10))
gdf.plot(edgecolor="black", linewidth=0.5, cmap="viridis")
plt.title("Cartographic Visualization of Indian States", fontsize=16)
plt.axis("off")
plt.show()

# Step 6: Create an interactive map using Folium
# Center map over India (approximate lat/lon)
center_lat, center_lon = 22.9734, 78.6569

m = folium.Map(location=[center_lat, center_lon], zoom_start=5, tiles="CartoDB positron")

# Detect a suitable property to use for tooltip (e.g., 'st_nm' or 'STATE_NAME')
possible_name_fields = ['st_nm', 'STATE_NAME', 'NAME_1', 'NAME']
name_field = None
for field in possible_name_fields:
    if field in gdf.columns:
        name_field = field
        break

if name_field is None:
    name_field = [col for col in gdf.columns if col != 'geometry'][0]
print(f"Using '{name_field}' as the name field for tooltips.")
```

```
# Add GeoJSON layer with tooltips
folium.GeoJson(
    gdf,
    name="Indian States",
    tooltip=GeoJsonTooltip(fields=[name_field], aliases=["State: "]),
    style_function=lambda x: {
        'fillColor': 'lightgreen',
        'color': 'black',
        'weight': 0.5,
        'fillOpacity': 0.6
    }
).add_to(m)

folium.LayerControl().add_to(m)

# Step 7: Save and display the interactive map
output_map = "/content/india_states_map.html"
m.save(output_map)
print("🌏 Interactive map saved as:", output_map)

display(IFrame(output_map, width=950, height=600))
```

---

**Experiment 14: Text Analysis (TF-IDF)**

Python

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
import nltk
```

```
from nltk.corpus import stopwords

documents = [
    "Data science is an interdisciplinary field focused on extracting insights from data.",
    "Machine learning is a subset of data science that enables systems to learn automatically."
]

# (Preprocessing steps assumed here)
cleaned_docs = documents # Placeholder for preprocessed text

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(cleaned_docs)

# Convert to DataFrame for visualization
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())
print("\nTF-IDF Matrix:\n")
print(tfidf_df.round(3))

# Cosine Similarity
similarity_matrix = cosine_similarity(tfidf_matrix)
similarity_df = pd.DataFrame(similarity_matrix,
                    index=[f'Doc{i+1}' for i in range(len(documents))],
                    columns=[f'Doc{i+1}' for i in range(len(documents))])

print("\nDocument Similarity Matrix:\n")
print(similarity_df.round(3))
```

**Experiment 15: Word Cloud**

Python

```
import matplotlib.pyplot as plt

from wordcloud import WordCloud


# Sample Text

text = "Data Science Machine Learning Artificial Intelligence..."


# (Preprocessing function calling assumed)

processed_text = text # Placeholder


# Generate Word Cloud

wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(processed_text)


# Render the Word Cloud

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Word Cloud Visualization')

plt.show()
```