Week 5: Data filtering, sampling and storage

Objective: In this lab session, you will learn data filtering techniques, sampling and storage. We will use Google Colab - Python IDE for most of our tasks.

Go to https://colab.research.google.com/ to start a new Python notebook. You will need to log in to use the Google Colab. You can use your university email or your personal.

Iris Dataset

- The Iris dataset is one of the most well-known datasets.
- The dataset contains 150 samples of iris flowers, each represented by four features:
- Sepal Length (cm): The length of the sepal (the outer part of the flower) in centimetres.
- Sepal Width (cm): The width of the sepal in centimetres.
- Petal Length (cm): The petal's length (the flower's inner part) in centimetres.
- Petal Width (cm): The width of the petal in centimetres.
- Species: The species of the iris flower. There are three classes:

- Iris Setosa: A species known for its small flowers and generally short petals.
- Iris Versicolor: A species with medium-sized flowers and petals.
- Iris Virginica: A species characterised by its larger flowers and longer petals.

```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sqlite3
```

Task1: Load Iris data from Seaborn

```
# Load the Titanic dataset from seaborn
iris = sns.load_dataset('iris')

iris.head()
```

Task2: Basic filtering technique

a. Filter the iris data based on petal length greater than 1.5cm

```
## 1. Filtering Rows Based on Conditions

# Filter flowers with petal length greater than 1.5 cm
filtered_petal_length = iris[iris['petal_length'] > 1.5]
print("\nFlowers with petal length greater than 1.5 cm:")
display(filtered_petal_length)
```

b. Filter setosa species with width greater then 3.0

```
## 2. Filtering with Multiple Conditions

# Filter flowers that are of species 'setosa' and have sepal width
greater than 3.0 cm
```

```
filtered_setosa = iris[(iris['species'] == 'setosa') &
(iris['sepal_width'] > 3.0)]
print("\nSetosa flowers with sepal width greater than 3.0 cm:")
display(filtered_setosa)
```

c. Filter the iris data using the isin() method

```
## 3. Filtering with `isin()`

# Filter flowers that are either 'versicolor' or 'virginica'
filtered_species = iris[iris['species'].isin(['versicolor',
'virginica'])]
print("\nFlowers that are either versicolor or virginica:")
display(filtered_species)
```

d. Filter the iris data using the query() method

```
## 4. Using `query()`

# Filter using query method to get flowers with petal width < 0.5 cm
filtered_petal_width = iris.query('petal_width < 0.5')
print("\nFlowers with petal width less than 0.5 cm:")
display(filtered_petal_width)
```

e. Filter the iris data using the string method

```
## 5. Filtering Using String Methods

# Filter flowers whose species name contains 'a'
filtered_species_name = iris[iris['species'].str.contains('a')]
print("\nFlowers whose species name contains 'a':")
display(filtered_species_name)
```

Task3: Data Sampling

a. Sample 20% of the iris data using random sample

```
# Basic Sampling Techniques
## 1. Random Sampling

# Randomly sample 20% of the data
random_sample = iris.sample(frac=0.2, random_state=42)
print("\nRandom Sample (20% of the data):")
display(random_sample)

# Visualization of the random sample
plt.figure(figsize=(10, 6))
sns.scatterplot(data=random_sample, x='sepal_length', y='sepal_width',
hue='species', style='species', s=100)
plt.title('Random Sample from Iris Dataset')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
```

```
plt.legend(title='Species')
plt.grid()
plt.show()
```

b. Sample 50% of each species of the iris data using stratified sampling technique

```
## 2. Stratified Sampling

# Stratified sampling based on species
stratified_sample = iris.groupby('species',
group_keys=False).apply(lambda x: x.sample(frac=0.5, random_state=42))
print("\nStratified Sample (50% from each species):")
display(stratified_sample)

# Visualization of the stratified sample
plt.figure(figsize=(10, 6))
sns.scatterplot(data=stratified_sample, x='sepal_length',
y='sepal_width', hue='species', style='species', s=100)
plt.title('Stratified Sample from Iris Dataset')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend(title='Species')
plt.grid()
plt.show()
```

c. Using systematic sampling, sample every third sample.

```
## 3. Systematic Sampling

# Systematic sampling by selecting every 3rd sample
systematic_sample = iris.iloc[::3, :]
print("\nSystematic Sample (Every 3rd row):")
display(systematic_sample)

# Visualization of the systematic sample
plt.figure(figsize=(10, 6))
sns.scatterplot(data=systematic_sample, x='sepal_length',
y='sepal_width', hue='species', style='species', s=100)
plt.title('Systematic Sample from Iris Dataset')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend(title='Species')
plt.grid()
plt.show()
```

d. Sample the first 30 samples using convenience sampling

```
# Non-Probability Sampling (Convenience Sampling)
random_sample_size = 30

convenience_sample = iris.head(random_sample_size)
```

```
# Display the convenience sample
display(convenience_sample)

# Visualize the convenience sample
plt.figure(figsize=(10, 5))
sns.scatterplot(data=convenience_sample, x='sepal_length',
y='sepal_width', hue='species', s=100)
plt.title('Convenience Sample of Iris Dataset')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.grid(True)
plt.show()
```

Task4: Storage in SQL db

- Storing and Retrieving Data with SQLite
- We'll explore how to store the Iris dataset in an SQLite database.

a. Create a connection with the database.

```
# Create a connection to SQLite database (it will create a new database
if it doesn't exist)
conn = sqlite3.connect('iris.db')
```

b. Write the systematic_sample dataframe from earlier to a SQL table

```
# 1. Create an systematic_sample table in the database
systematic_sample.to_sql('systematic_sample', conn,
if_exists='replace', index=False)
```

c. Querying the systematic_sample table in the database.

```
# 2. Querying data from the SQLite database
# we will retrieve all data from the 'systematic_sample' table
query_all = "SELECT * FROM systematic_sample"
df_all = pd.read_sql_query(query_all, conn)
display(df_all)
```

d. Close the connection with the database.

```
# Close the SQLite connection
conn.close()
```