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**

* 1. **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)

* 1. **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)

* 1. **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)

* 1. **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)

* 1. **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**

1. **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()

1. **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()

1. **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()

1. **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.
  1. **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')

* 1. **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)

* 1. **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)

* 1. **Close the connection with the database.**

# Close the SQLite connection

conn.close()