Practical Introduction to Data Science

Assessment No 02

Question 1:

(i) Explanation of the Code

Loading and Summarizing a Dataset

The Python program follows essential data analysis and visualization methods based on NumPy, Pandas, Matplotlib and Seaborn. This code displays numerous important functions which include:

- Seaborn provides the dataset to the program through sns.load_dataset("NAMEHIDDEN").
- The dataset gets divided into separate groups based on the "dataset" column followed by computation of mean and variance using agg(["mean", "var"]) function.
- Descriptive statistics need the aggregated["mean","var"] function to determine EDA's fundamental statistics.

Data Grouping and Aggregation

- Groupby("dataset") allows users to analyze separate groups of information in the dataset.
- Summary statistics obtained through agg(["mean", "var"]) show different levels of data distribution for separate groups.
- This technique serves applications in hypothesis testing and data pattern recognition operations.

Visualization Using Matplotlib

- The code generates 2x2 grid of subplots through plt.subplots(2, 2, figsize=(10, 8)) for examining four different sets of data.
- The loop selects subset data from mydataset using filtering (mydataset[mydataset["dataset"] == dataset]) for each dataset "A", "B", "C", and "D".
- Each dataset receives a scatter plot creation for its x vs. y relationship by the code.
- Using scatter plots remains essential for detecting relationships and identifying patterns in two numerical variables while spotting unusual points or trend patterns.

Legends and Titles for Better Interpretation

- The visual elements of each subplot get their titles through ax.set_title commands together with their legends that appear using ax.legend for better understanding.
- The function plt.tight_layout() maintains proper spacing throughout the plot so elements do not clash.

(ii) Confounding Variables

An extraneous variable can influence both an independent variable and a dependent variable simultaneously to produce incorrect research findings.

Example:

- A research presents evidence that shows ice cream sales and drowning incidents have a direct relationship.
- Independent Variable (X): Ice cream sales
- Dependent Variable (Y): Drowning incidents
- Confounding Variable (Z): Temperature

The sales of ice cream increase when temperature levels rise. Higher temperatures induce more individuals to engage in swimming which raises the chances of drowning accidents. The variable temperature interferes with the relationship between ice cream sales and drowning incidents through its impact on both variables resulting in an unjustified connection between them. The correlation produces erroneous results to describe the actual relationship between ice cream sales and drowning incidents.When temperature becomes an accounted factor the original link between these two variables vanishes indicating a single influencing relationship between temperature and both ice cream sales and drowning occurrences.

We can respresent question 1 using a UML class diagram which would be helpful to show the relationships between the libraries.

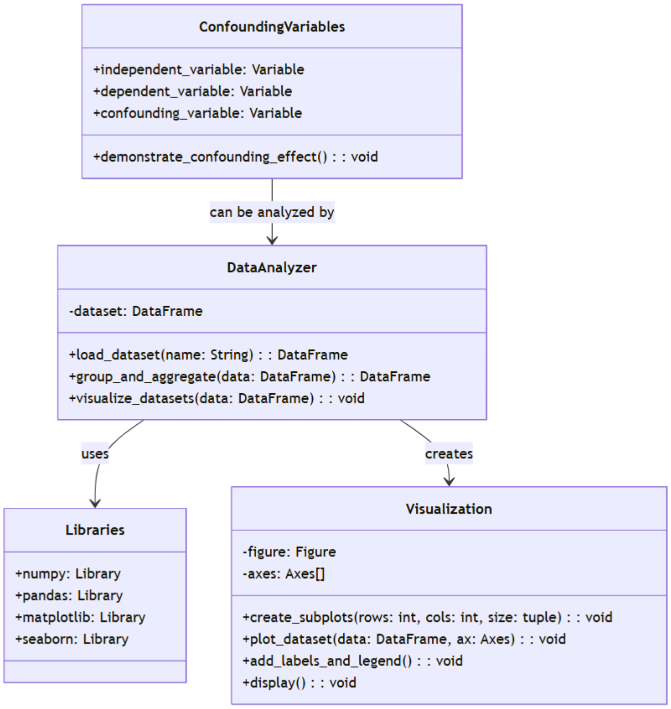
ConfoundingVariables
+independent_variable: Variable
+dependent_variable: Variable
+confounding_variable: Variable
+demonstrate_confounding_effect() : : void
can be analyzed by
DataAnalyzer
-dataset: DataFrame
+load_dataset(name: String) : : DataFrame
+group_and_aggregate(data: DataFrame) : : DataFrame
+visualize_datasets(data: DataFrame) : : void
uses
creates
Libraries
+numpy: Library
+pandas: Library
+matplotlib: Library
+seaborn: Library
Visualization
-figure: Figure
-axes: Axes[]
+create_subplots(rows: int, cols: int, size: tuple) : : void
+plot_dataset(data: DataFrame, ax: Axes) : : void
+add_labels_and_legend() : : void
+display() : : void

Figure 1: relationships between the libraries and components

Question No 02:

(i) Triple Store vs. NoSQL Database

A triple store functions as a database system which manages RDF (Resource Description Framework) data structures consisting of three-part subject-predicate-object tuples. The system was built for executing semantic queries through the SPARQL specification.

The closest NoSQL database equivalent to Neo4j operates as a graph database. While both triple stores and graph databases handle relationships their structural differences along with their separate query languages create distinction between them.

Triple Store (SPARQL-based)

- Uses RDF triples (subject-predicate-object).
- Users write SPARQL queries to retrieve organized data from semantic web resources including DBpedia through this system.
- All data in this framework operates under an open-world assumption because unidentified data entries indicate that content may be true.

Graph Database (Cypher-based, e.g., Neo4j)

- The database includes nodes for entities and edges for explicit relation storage.
- Queries are developed with Cypher for Neo4j databases as well as Gremlin through Apache TinkerPop.
- The data storage method places importance on quick relationship navigation processes.

Contrast with Another NoSQL Database

We also examined document-oriented noSQL database systems during the practical work together with MongoDB. Unlike graph databases and triple stores:

- The documents stored in document databases take the form of JSON-like structures based on key-value pairing.
- A document database does not demand an established schema and holds great flexibility as a result.
- Complex relationships which appear in forms of graphs and triples do not have built-in support within their structure.

The main difference between these database types stems from their approach to managing relations because document databases manage hierarchical information without needing extensive data networks.

(ii) Query Analysis

Typically, we can represent following query, For the SPARQL query part, an object diagram would help illustrate the specific resources and relationships in the query

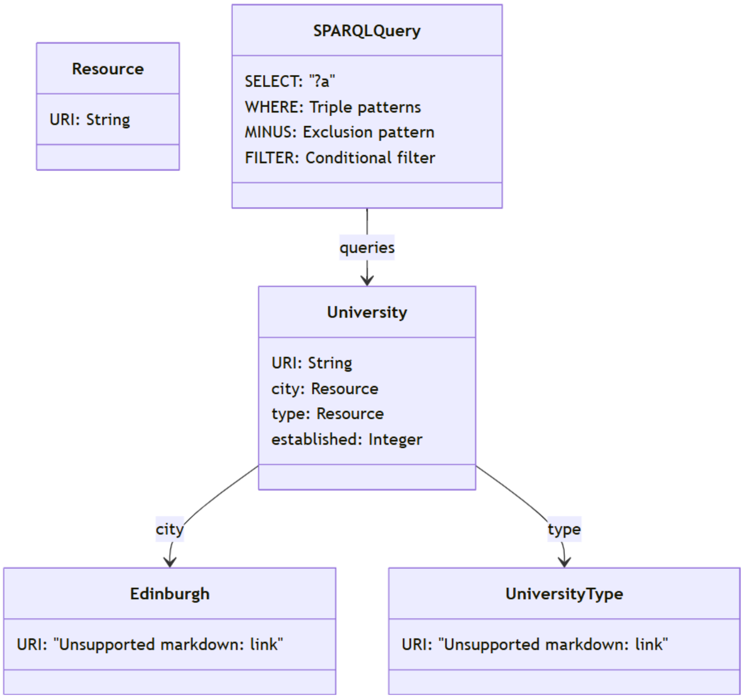

Figure 2: an object diagram showing relationships in the query

(a) Purpose of the Query

The SPARQL query:

```
SELECT DISTINCT ?a WHERE {
   ?a <http://dbpedia.org/ontology/city> <http://dbpedia.org/resource/Edinburgh> ;
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/ontology/University> .
   MINUS { ?a <http://dbpedia.org/property/established> "1964"^^xsd:integer }
}
```

This query is designed to:

- You can locate universities which operate in Edinburgh through searching.
- The query should eliminate all universities which were created in 1964.

A simple execution of the query demonstrates the output results. Some universities founded in 1964 should be appearing in results but we might encounter them because of the following possibilities:

- The established property exists with multiple data types within the database system (strings and dates).
- The filter demonstrates incorrect functioning when the established field is saved as a string format instead of the integer format xsd:integer.
- Triple store data storage faces two difficulties when a value could be absent from the database or located under different predicate names.

(b) Meaning of the Semicolon (;)

The semicolon at the end of line 3:

```
?a <http://dbpedia.org/ontology/city> <http://dbpedia.org/resource/Edinburgh> ;
```

This line belongs to the same triple pattern due to the next statement. Through the use of this statement a single subject (?a) can be associated with different predicates. Maintaining a common variable requires the use of semicolon otherwise we would need to duplicate?a on each line.

(c) Identifying the Predicates

The role of predicates allows for establishing associations between subjects and objects. This query contains the following predicates as its relational components.

- <http://dbpedia.org/ontology/city> → Links a university (?a) to Edinburgh.
- <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> → Specifies that ?a is of type University.
- <http://dbpedia.org/property/established> → Represents the establishment year of a university.

(d) Modifying the Query to Exclude All Years After 1963

To exclude any universities established after 1963, we modify the query:

```
SELECT DISTINCT ?a WHERE {

   ?a <http://dbpedia.org/ontology/city> <http://dbpedia.org/resource/Edinburgh> ;

      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/ontology/University> .

   FILTER ( ?established <= "1963"^^xsd:integer )

}
```

Modification of the query becomes necessary for excluding universities founded after 1963.

The FILTER statement replaces MINUS because it excludes all years that exceed 1963.

The query incorporates this criterion to omit all universities which obtained their establishment year after 1964.

Question No 03:

For the Bayes' Rule derivation an activity diagram would be useful to illustrate the classification process step by step we will follow:

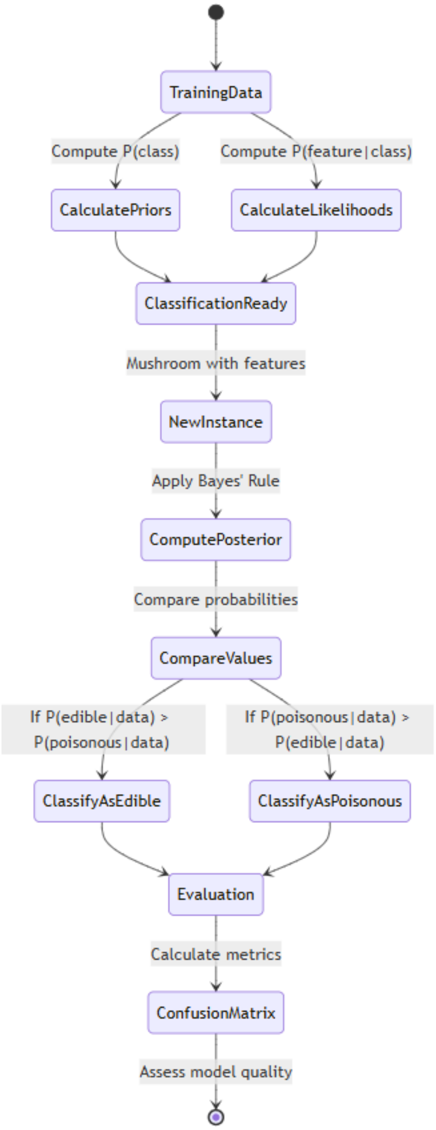

Figure 3: Bayes' Rule step by step by an activity diagram

(i) Derivation of Bayes' Rule

We start with the given equation:

$$p(x,y)=p(x|y)p(y)=p(y|x)p(x)$$

Since both expressions represent the joint probability of x and y, we can rearrange:

$$p(y|x)= p(x|y)p(y) /p(x)$$

This is the standard form of Bayes' Theorem**.**

Our Assumption:

- Conditional Probability Definition**:** We assume the probability of y given x is proportional to the probability of x given y.
- Law of Total Probability**:** The denominator p(x) can be expanded using all possible classes:

$$p(x)=p(x|y1)p(y1)+p(x|y2)p(y2)+\dots$$

- Independence Assumption for Naïve Bayes**:** When multiple features exist, we assume they are conditionally independent given the class.

ii) Understanding Bayes' Rule in Naïve Bayes Classification

In a Naïve Bayes classifier, we calculate:

$$p(class|data)\propto p(data|class)p(class)$$

What Affects These Values?

- Prior Probability p(class): How likely a class is before seeing the data.
- Likelihood p(data|class): The probability of the given data occurring under a specific class.
- Evidence p(data): This is the denominator, but it cancels out when comparing different classes.

We can compute them by

$$p(data|class)=p(feature1|class)\times p(feature2|class)\times\dots$$

The method calculates class probabilities at high speed regardless of the number of features present.

(iii) Why is Naïve Bayes Considered Naïve?

- Naïve Bayes applies the unrealistic premise that all features automatically become independent when conditioned on the category.
- The algorithm fails to recognize how features affect one another since it does not handle associative relationships between elements (e.g. image color and texture).
- Naïve Bayes maintains effective performance even though it operates based on a straightforward method when handling text classification tasks.

(iv) Classifying a Mushroom with bruise=true and veilColour=orange

Using Table 1 (bruise) and Table 2 (veilColour)**:**

Step 1: Compute Prior Probabilities

$$p(\text{edible}) = 253/400$$

$$p(\text{edible}) = 0.6325$$

$$p(\text{poisonous}) = 147/400$$

$$p(\text{poisonous}) = 0.367$$

Step 2: Compute Likelihoods

For bruise=true:

$$p(\text{bruise=true}|\text{edible}) = 64/251$$

$$p(\text{bruise=true}|\text{edible}) =0.2550$$

$$p(\text{bruise=true}|\text{poisonous})=92/149$$

$$p(\text{bruise=true}|\text{poisonous})=0.6174$$

For veilColour=orange:

$$p(\text{veilColour=orange}|\text{edible})=94/253$$

$$p(\text{veilColour=orange}|\text{edible})=0.3715$$

$$p(\text{veilColour=orange}|\text{poisonous})=22/147$$

$$p(\text{veilColour=orange}|\text{poisonous})=0.1497$$

Step 3: Compute Posterior Probabilities

Using Bayes' Rule:

$$p(\text{edible}|\text{bruise=true,veilColour=orange})\propto p(\text{bruise=true}|\text{edible})\times p(\text{veilColour=orange}|\text{edible})\times p(\text{edible})$$

$$p(\text{edible}|\text{bruise=true,veilColour=orange})\propto 0.2550\times 0.3715\times 0.6325=0.0598$$

$p(poisonous|bruise=true,veilColour=orange) \propto p(bruise=true|poisonous) \times p(veilColour=orange|poisonous) \times p(poisonous)$

$p(poisonous|bruise=true,veilColour=orange) \propto 0.6174 \times 0.1497 \times 0.3675 = 0.0340$

Since $p(edible|data) > p(poisonous|data)$, the classifier predicts the mushroom is **edible**.

(v) Evaluating the Model

Confusion Matrix

	Predicted Edible (True)	**Predicted Poisonous (False)**
Actual Edible (True)	45	10
Actual Poisonous (False)	14	21

Sensitivity (Recall for Edible)

Sensitivity=TP / (TP+FN) = 45/55

Sensitivity =0.8181

Specificity (Recall for Poisonous)

Specificity= TN / (TN+FP) = 21/35

Specificity=0.6

Accuracy:

Accuracy = (45+21)/90

Accuracy= 0.733

The wrong classification of poisonous mushrooms as edible substances presents a danger.

Accuracy should be considered together with fairness although it becomes dangerous when the system does not detect harmful mushrooms.

This situation requires peritorizing specificity above accuracy in terms of mushroom identification.