Exploratory Data Analysis Report

Shweta Ajay Shinde

Masters in Data Analytics, San Jose State University

Data 230: Data Visualization

Instructor: Venkata Duvvuri

8th Sept 2024

Exploratory Data Analysis Report

1.Introduction

This report presents an exploratory analysis of the data set contained in **Salaries.csv**. This dataset includes details about various salaries, and the primary objective is to explore and understand the data's structure and key attributes using google colab.

2. Data Loading and Inspection

2.1 Loading the Dataset

First, we'll need to load the dataset. We will first read the CSV file into a Pandas Data Frame for which we will use **pd.read_csv** command.

```
import pandas as pd
from google.colab import drive

# Reading the CSV file into a DataFrame
salaries = pd.read_csv('/content/drive/MyDrive/Salaries.csv')
```

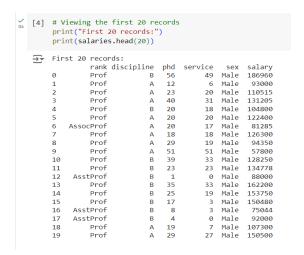
2.2 Viewing the Data

To get an initial understanding of the dataset, we can view the first few records. We will use **head** () command to view data. 'salaries' is the name given to the data frame.

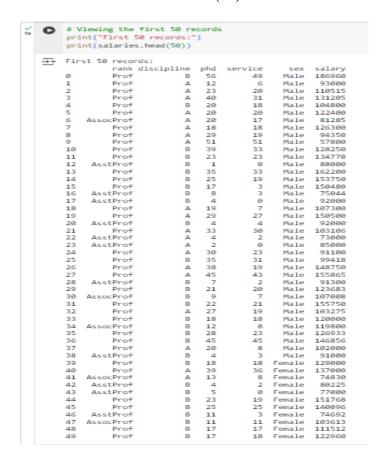
• First 10 records:-salaries.head(10)

EXPLORATORY DATA ANALYSIS REPORT

• First 20 records: salaries.head(20)



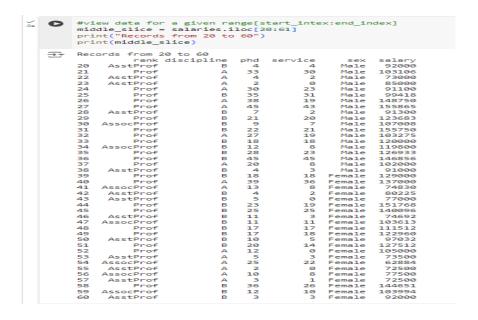
• First 50 records: salaries.head(50)



2.3 Viewing the Records in given range

To view the records in given range, use the **iloc()** method. **salaries.iloc(start_index:end_index)**

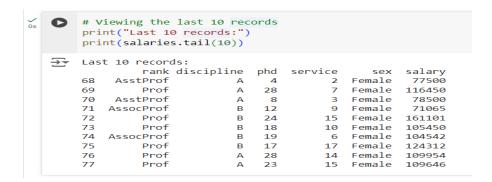
salaries.iloc(20:61): This will slice the row from salary to include rows from index 20 up to 60, but not including index 61.



2.4 Viewing the Last Few Records

To view the last few records, use the tail() method.

Last 10 records: salaries.tail(10)



3. Data Summary

3.1 Number of Records

To find out how many records (rows) are in the Data Frame we use **shape** command Number of records:-salaries.shape[0]

```
# Number of records
num_records = salaries.shape[0]
print(f"Number of records: {num_records}")

Number of records: 78
```

Here shape returns tuple of array dimensions. (rows, columns), we just want rows, so we get **shape[0]**

```
print(salaries.shape)
(78, 6)
```

f in print(**f**"Number:{num_records}") indicates an f-string. F-strings let you insert variables directly into strings using curly braces {}, making it easy to include values in your text.

3.2 Number of Elements

The total number of elements can be calculated as the product of the number of rows and columns. For this we use the **size** command.

Number of elements: salaries.size

```
# Number of elements
num_elements = salaries.size
print(f"Number of elements: {num_elements}")

Number of elements: 468
```

3.3 Column Names

To get the column names we use **columns.tolist()**

tolist() will convert the column names of the Data Frame to a list.

```
# Column names
column_names = salaries.columns.tolist()
print(f"Column names: {column_names}")

Column names: ['rank', 'discipline', 'phd', 'service', 'sex', 'salary']
```

Or we can use **list(salaries.columns)**

```
# Column names
column_list = list(salaries.columns)
print(f"Column names: {column_list}")

Column names: ['rank', 'discipline', 'phd', 'service', 'sex', 'salary']
```

3.4 Data Types of Columns

To view the types of each column we use **dtype** command.

Column data types: salaries.dtype

```
# Column data types
column_types = salaries.dtypes
print("Column data types:")
print(column_types)

Column data types:
rank object
discipline object
phd int64
service int64
sex object
salary int64
dtype: object
```

Here's a brief description of data type used:

- **object**: Represents categorical data or text. Used for strings and mixed types.
- int64: Represents integer numbers, which can be large, without decimal points.

4. Calculation on Data

4.1 Basic Statistics for the Salary Column

To calculate basic statistics (mean, median, standard deviation, etc.) for the salary column we use **salaries.['salary'].describe()** command which provides summary statistics for the 'salary' column. The **round()** function rounds numbers to the specified number of decimal places given as a parameter.

• **float64**: Represents numbers with decimals, allowing for precise measurements and fractional values.

4.2 Count of Values in the Salary Column

To find out how many values are present in the salary column we use **salaries.['salary'].count()** command which returns the number of non-null values in the 'salary' column.

```
# Count of values in the 'salary' column
salary_count = salaries['salary'].count()
print(f"Number of values in the 'salary' column: {salary_count}")

Number of values in the 'salary' column: 78
```

4.3 Average Salary

To calculate the average salary, we use **salaries.['salary'].mean()** command. It will return the average salary.

```
# Average salary
average_salary = salaries['salary'].mean()
print(f"Average salary: {average_salary:.2f}")

Average salary: 108023.78
```

Here {average salary:.2f} : .2f displays value up to 2 decimal place

5. Conclusion

Using Pandas, we analyzed a dataset with 78 records and different data types like float, int, object. We calculated important statistics, such as averages and ranges. Pandas made it easy to handle and understand the data.

References

Pandas Development Team. (n.d.). *Getting started with pandas*. Retrieved, from https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html

Google Colab. command line code. *Assignment_3.ipynb*, https://colab.research.google.com/drive/10X39kH4Vwvr_CPYJG2os94gTYWymZ B1M#scrollTo=et99_DyZEUR9