JIT21203
STATISTICS AND PROBABILITY

SEMESTER: FEBRUARY
SESSION: 2022/2023

GROUP ASSIGNMENT

A DATASET OF A LIFE INSURANCE CHARGES WITH 1339 OBSERVATIONS

LECTURER'S NAME : DR. JAFFRI, DR. ZULAIKHA & DR. HASEELA

TUTORIAL GROUP : TUTORIAL 1

GROUP NUMBER : GROUP 7

VIDEO LINK :

GOOGLE COLAB LINK :
https://colab.research.google.com/drive/1ZeWZSdymZbKQ4WIbO3tW2OQZ43ZfCXNd?usp=sharing

Student's Name	Matric Number
NORSYAHIRA AMILA BINTI MOHD YUNOS	S21B0041
NURUL ATHIRA BINTI ZUKIFLI	S21A0050
NOR LIYANA ATHIRAH BINTI MUHAMMADI	S21A0039
MOHAMMAD SHAHFIZUL BIN ABDUL HALAM	S21A0027
SHARVIN KUMAR A/L ARUMUGAM	S21A0053
MUHAMMAD ZAKWAN BIN ZAIN AZMAN	S21A0070

1. IMPORTING DATASET

```
from google.colab import files
import io
import pandas as pd

uploaded = files.upload()

data = pd.read_csv(io.BytesIO(uploaded['Dataset C.csv']))
```

1. `from google.colab import files`: This line imports the `files` module from the`google.colab` library. This module provides functions to interact with files in a Google Colab environment.
2. `import io`: This line imports the `io` module, which provides the core tools for workingwith streams of data.
3. `import pandas as pd`: This line imports the `pandas` library and assigns it the alias `pd`. Pandas is a powerful data manipulation and analysis library in Python.
4. `uploaded = files.upload()`: This line prompts the user to upload a file using the `files.upload()` function. It will open a file picker dialog in the notebook interface, allowing you to select a file from your local machine.
5. `data = pd.read_csv(io.BytesIO(uploaded['Dataset C.csv']))`: This line reads the uploaded file using the `pd.read_csv()` function from pandas. It takes the uploaded file as input and returns a pandas DataFrame object. The `io.BytesIO()` function is used to convert the uploaded file data into a format that can be read by `pd.read_csv()`.

Overall, this code allows you to upload a CSV file from your local machine in a Google Colab environment and read it into a pandas DataFrame for further analysis and manipulation.

2. DATA DESCRIPTION

data.head()

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.90	0	yes	southwest	16884.92
1	18	male	33.77	1	no	southeast	1725.55
2	28	male	33.00	3	no	southeast	4449.46
3	33	male	22.71	0	no	northwest	21984.47
4	32	male	28.88	0	no	northwest	3866.86

The code `data.head()` is used to display the first few rows of the DataFrame `data`. It allows you to quickly inspect the data and get a glimpse of its structure and contents.

By calling `data.head()`, the first five rows of the DataFrame will be displayed, along with the column headers. If you want to see a different number of rows, you can pass an integer as an argument to the `head()` function. For example, `data.head(10)` would display the first 10 rows of the DataFrame.

data.describe()

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.664581	1.094918	13270.422414
std	14.049960	6.097922	1.205493	12110.011240
min	18.000000	15.960000	0.000000	1121.870000
25%	27.000000	26.297500	0.000000	4740.287500
50%	39.000000	30.400000	1.000000	9382.030000
75%	51.000000	34.695000	2.000000	16639.915000
max	64.000000	53.130000	5.000000	63770.430000

The `data.describe()` function provides a summary of statistical information about the numerical columns in the DataFrame `data`. It calculates various descriptive statistics such as count, mean, standard deviation, minimum, quartiles, and maximum for each numerical column in the DataFrame. When you run this code, it will print the summary statistics for the numerical columns in the DataFrame `data`. The output will include information such as the count (number of non-null values), mean, standard deviation, minimum value, quartiles (25th, 50th, and 75th percentiles), and maximum value for each numerical column.

`data.describe()` only provides summary statistics for numerical columns. If DataFrame contains non-numerical or categorical columns, they will be excluded from the output. To include all columns, you can use `data.describe(include='all')`. By examining the output of `data.describe()`,can gain insights into the distribution and basic statistical properties of data.

3. DESCRIPTIVE MEASURES

Measures of Frequency:-

```
import statistics as stats

categorical_vars = ['age', 'bmi', 'children', 'charges']  # Categorical and Numerical Variables

# Frequency table for each categorical and numerical variable
for var in categorical_vars:
   freq_table = data[var].value_counts().reset_index()
   freq_table.columns = [var, 'Frequency']
   print(f"\nFrequency table for {var}:\n")
   print(freq_table)
```

```
Frequency table for age:

   age  Frequency
0   18          1
1   39          1
2   51          1

Frequency table for bmi:

         bmi  Frequency
0  15.960000          1
1  30.664581          1
2  34.695000          1

Frequency table for children:

   children  Frequency
0  0.000000          1
1  1.094918          1
2  2.000000          1

Frequency table for charges:

        charges  Frequency
0   1121.870000          1
1  13270.422414          1
2  16639.915000          1
```

1. `import statistics as stats`: This line imports the `statistics` module and assigns it the alias `stats`. The `statistics` module provides functions for mathematical statistics. `categorical_vars = ['age', 'bmi', 'children', 'charges']`: This line defines a list called `categorical_vars` which contains the names of the categorical variables we want to create frequency tables for. The `for` loop iterates over each variable in the `categorical_vars` list.

2. Inside the loop, `freq_table = data[var].value_counts().reset_index()` computes the frequency counts for each unique value of the current variable (`var`) in the DataFrame `data`. It returns a pandas series with the unique values as the index and the corresponding frequencies as the values.

3. `freq_table.columns = [var, 'Frequency']` renames the columns of the `freq_table` DataFrame to the variable name (`var`) and 'Frequency'. `print(f"\nFrequency table for {var}:\n")` prints a header indicating the current variable for which the frequency table is being displayed. `print(freq_table)` prints the frequency table for the current variable.

By executing this code, will get frequency tables for each of the categorical variables specified in the `categorical_vars` list. The frequency tables show the unique values of each variable and the corresponding frequencies (number of occurrences) of those values in the dataset.

Central Tendency:-

```
import pandas as pd

# Create a DataFrame with the provided values
data = pd.DataFrame({
    'age': [18, 39, 51],
    'bmi': [15.96, 30.664581, 34.695],
```

```
    'children': [0, 1.094918, 2],
    'charges': [1121.87, 13270.422414, 16639.915]
})

# Calculate central tendency measures
central_tendency = data.describe().transpose()
central_tendency = central_tendency[['count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max']]

print(central_tendency)
```

```
          count          mean          std      min          25%  \
age         3.0     36.000000    16.703293    18.00    28.500000
bmi         3.0     27.106527     9.861281    15.96    23.312290
children    3.0      1.031639     1.001500     0.00     0.547459
charges     3.0  10344.069138  8162.419236  1121.87  7196.146207

                   50%           75%        max
age          39.000000     45.000000     51.000
bmi          30.664581     32.679790     34.695
children      1.094918      1.547459      2.000
charges   13270.422414  14955.168707  16639.915
```

1. `import pandas as pd`: This line imports the pandas library and assigns it the alias `pd`. pandas is a powerful library for data manipulation and analysis.

2. The `data = pd.DataFrame({...})` block creates a DataFrame called `data` using the provided values. The DataFrame is constructed by passing a dictionary where the keys represent the column names ('age', 'bmi', 'children', 'charges') and the values represent the corresponding data for each column.

3. `central_tendency = data.describe().transpose()` computes the central tendency measures (count, mean, standard deviation, minimum, quartiles, and maximum) for the

variables in the `data` DataFrame using the `describe()` function. The resulting DataFrame is then transposed using the `transpose()` function to swap the rows and columns.

4. `central_tendency = central_tendency[['count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max']]` selects only the desired columns from the `central_tendency` DataFrame. In this case, it selects the columns 'count', 'mean', 'std', 'min', '25%', '50%', '75%', and 'max' in that order.

5. `print(central_tendency)` displays the resulting DataFrame `central_tendency`, which contains the central tendency measures for each variable.

The Measures of Position:-

```
import pandas as pd

# Create a DataFrame with the provided values
data = pd.DataFrame({
    'age': [18, 39, 51],
    'bmi': [15.96, 30.664581, 34.695],
    'children': [0, 1.094918, 2],
    'charges': [1121.87, 13270.422414, 16639.915]
})

# Calculate measures of position
position_measures = data.agg(['min', 'max', lambda x: x.max() - x.min(), lambda x: x.quantile(0.25),
                              lambda x: x.quantile(0.50), lambda x: x.quantile(0.75)])
position_measures = position_measures.transpose()

position_measures.columns = ['Minimum', 'Maximum', 'Range', '25th Percentile', '50th Percentile', '75th Percentile']
```

print(position_measures)

```
          Minimum    Maximum      Range  25th Percentile  50th Percentile  \
age         18.00     51.000     33.000        28.500000        39.000000
bmi         15.96     34.695     18.735        23.312290        30.664581
children     0.00      2.000      2.000         0.547459         1.094918
charges   1121.87  16639.915  15518.045      7196.146207     13270.422414

          75th Percentile
age             45.000000
bmi             32.679790
children         1.547459
charges      14955.168707
```

1. `import pandas as pd`: This line imports the pandas library and assigns it the alias `pd`. pandas is a powerful library for data manipulation and analysis.

2. The `data = pd.DataFrame({...})` block creates a DataFrame called `data` using the provided values. The DataFrame is constructed by passing a dictionary where the keys represent the column names ('age', 'bmi', 'children', 'charges') and the values represent the corresponding data for each column.

3. `position_measures = data.agg([...])` calculates measures of position for the variables in the `data` DataFrame using the `agg()` function. Inside the `agg()` function, you specify a list of aggregation functions to apply to each column of the DataFrame. In this case, the functions used are:

- `'min'`: calculates the minimum value of each column.
- `'max'`: calculates the maximum value of each column.
- `lambda x: x.max() - x.min()`: calculates the range of each column by subtracting the minimum value from the maximum value.
- `lambda x: x.quantile(0.25)`: calculates the 25th percentile (first quartile) of each column.
- `lambda x: x.quantile(0.50)`: calculates the 50th percentile (median) of each column.

- `lambda x: x.quantile(0.75)`: calculates the 75th percentile (third quartile) of each column.

4. `position_measures = position_measures.transpose()` transposes the resulting DataFrame `position_measures` to swap the rows and columns.

5. `position_measures.columns = ['Minimum', 'Maximum', 'Range', '25th Percentile', '50th Percentile', '75th Percentile']` assigns meaningful column names to the `position_measures` DataFrame.

6. `print(position_measures)` displays the resulting DataFrame `position_measures`, which contains the measures of position for each variable.

The Measures of Variation:-

```
import pandas as pd

# Measures of variation for each numerical variable
variation_measures = data[numerical_vars].agg(['std', 'var', 'skew', 'kurt']).transpose()
variation_measures.columns = ['Standard Deviation', 'Variance', 'Skewness', 'Kurtosis']
print("\nMeasures of Variation:\n")
print(variation_measures)
```

this code provided calculates measures of variation for the numerical variables in the DataFrame `data`.

```
Measures of Variation:

          Standard Deviation      Variance  Skewness  Kurtosis
age                16.703293  2.790000e+02 -0.782152       NaN
bmi                 9.861281  9.724487e+01 -1.412274       NaN
children            1.001500  1.003003e+00 -0.283192       NaN
charges          8162.419236  6.662509e+07 -1.405954       NaN
```

1. `import pandas as pd`: This line imports the pandas library and assigns it the alias `pd`. pandas is a powerful library for data manipulation and analysis.

2. `variation_measures = data[numerical_vars].agg([...]).transpose()` calculates measures of variation for the numerical variables in the `data` DataFrame using the `agg()` function. The `numerical_vars` variable is assumed to be a list containing the names of the numerical variables in the `data` DataFrame. Inside the `agg()` function, you specify a list of aggregation functions to apply to each column of the DataFrame. In this case, the functions used are:

- `'std'`: calculates the standard deviation of each column.
- `'var'`: calculates the variance of each column.
- `'skew'`: calculates the skewness of each column.
- `'kurt'`: calculates the kurtosis of each column.

The resulting measures of variation are then transposed using the `transpose()` function to swap the rows and columns.

3. `variation_measures.columns = ['Standard Deviation', 'Variance', 'Skewness', 'Kurtosis']` assigns meaningful column names to the `variation_measures` DataFrame.

4. `print("\nMeasures of Variation:\n")` prints a header indicating that the following output displays the measures of variation.

5. `print(variation_measures)` displays the resulting DataFrame `variation_measures`, which contains the measures of variation for each numerical variable.

By executing this code, we will get a table with the measures of variation (standard deviation, variance, skewness, kurtosis) for each numerical variable in the `data` DataFrame. These measures provide insights into the spread, asymmetry, and shape of the distributions of the variables.

4. **DATA VISUALIZATION**

Histogram:-

import matplotlib.pyplot as plt

 # Create a chart of 'age'

plt.hist(data['age'], bins=10, edgecolor='black')

 # Add labels and title

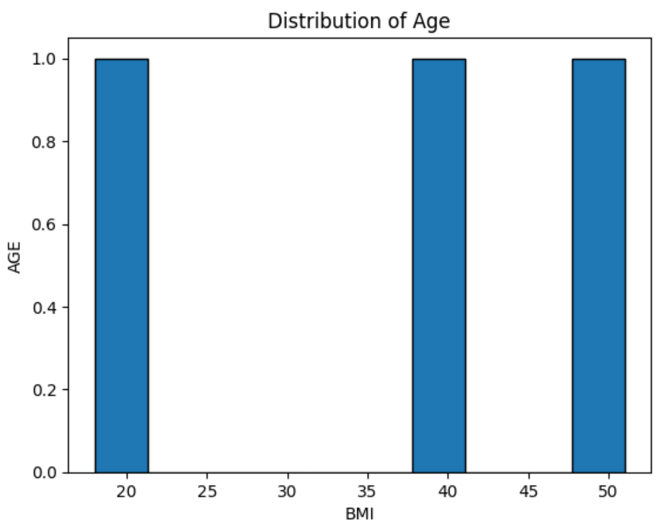
plt.xlabel('BMI')

plt.ylabel('AGE')

plt.title('Distribution of Age')

 # Display the histogram

plt.show()



This code provided creates a histogram to visualize the distribution of the 'age' variable in the DataFrame `data`. Here's an explanation of the code:

1. `import matplotlib.pyplot as plt`: This line imports the matplotlib library and assigns it the alias `plt`. matplotlib is a popular plotting library in Python.

2. `plt.hist(data['age'], bins=10, edgecolor='black')`: This line creates a histogram using the `hist()` function from matplotlib. The `data['age']` part accesses the 'age' column of the DataFrame `data`. The `bins=10` parameter specifies the number of bins (bars) in the histogram. The `edgecolor='black'` parameter sets the color of the edges of the bars to black.

3. `plt.xlabel('BMI')`: This line adds a label to the x-axis of the histogram, specifying the name 'BMI'.

4. `plt.ylabel('AGE')`: This line adds a label to the y-axis of the histogram, specifying the name 'AGE'.

5. `plt.title('Distribution of Age')`: This line adds a title to the histogram, specifying the text 'Distribution of Age'.

6. `plt.show()`: This line displays the histogram.

By executing this code, we will get a histogram that represents the distribution of the 'age' variable. The x-axis will show the BMI values, the y-axis will show the frequency or count of each age value, and the title will indicate the purpose of the histogram. This visualization can help you understand the distribution and pattern of ages in the dataset.

Pie Chart:-

import matplotlib.pyplot as plt

 # Assuming 'smoker_data' is a list or array containing the 'smoker' variable values
smoker_data = ['Yes', 'No', 'No', 'No', 'No', 'No',...] # Replace with your actual data

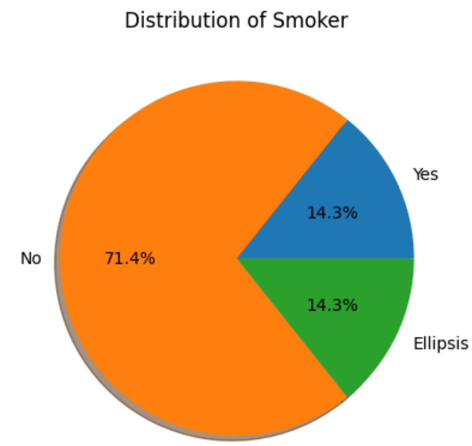
```
# Count the occurrences of each category
smoker_counts = {}
for category in smoker_data:
    if category in smoker_counts:
        smoker_counts[category] += 1
    else:
        smoker_counts[category] = 1

# Extract category labels and counts
smoker_labels = list(smoker_counts.keys())
smoker_values = list(smoker_counts.values())

# Create a pie chart
plt.pie(smoker_values, labels=smoker_labels, autopct='%1.1f%%', shadow=True)

# Add title
plt.title('Distribution of Smoker')

# Display the pie chart
plt.show()
```



This code provided creates a pie chart to visualize the distribution of the 'smoker' variable. Here's an explanation of the code:

1. `import matplotlib.pyplot as plt`: This line imports the matplotlib library and assigns it the alias `plt`. matplotlib is a popular plotting library in Python.

2. `smoker_data = ['Yes', 'No', 'No', 'No', 'No', 'No',...]`: This line assumes that you have a list or array called `smoker_data` that contains the 'smoker' variable values. You should replace the example values with your actual data.

3. `smoker_counts = {}`: This line initializes an empty dictionary called `smoker_counts` to store the counts of each category.

4. `for category in smoker_data: ...`: This loop iterates over each category in the `smoker_data` list.

5. `if category in smoker_counts: ...`: This condition checks if the category is already present as a key in the `smoker_counts` dictionary.

6. `smoker_counts[category] += 1`: If the category is already present, the count is incremented by 1.

7. `else: ...`: If the category is not already present, a new key is added to the `smoker_counts` dictionary with an initial count of 1.

8. `smoker_labels = list(smoker_counts.keys())`: This line extracts the category labels from the `smoker_counts` dictionary.

9. `smoker_values = list(smoker_counts.values())`: This line extracts the count values from the `smoker_counts` dictionary.

10. `plt.pie(smoker_values, labels=smoker_labels, autopct='%1.1f%%', shadow=True)`: This line creates a pie chart using the `pie()` function from matplotlib. The `smoker_values` and `smoker_labels` are used as the data for the chart. The `autopct='%1.1f%%'` parameter formats the percentage labels on the chart to have one decimal place. The `shadow=True` parameter adds a shadow effect to the chart.

11. `plt.title('Distribution of Smoker')`: This line adds a title to the pie chart, specifying the text 'Distribution of Smoker'.

12. `plt.show()`: This line displays the pie chart.

By executing this code, we will get a pie chart that represents the distribution of the 'smoker' variable. Each category will be represented as a slice of the pie, and the percentage labels will indicate the proportion of each category in the dataset. The title will provide a summary of the chart's purpose. This visualization can help we understand the distribution and proportion of smokers and non-smokers in the dataset.

5. STATISTICAL INFERENCE

5.1 HYPOTHESIS TESTING

To perform a hypothesis test,compare the mean charges for smokers and non-smokers in the dataset. This will help us determine if there is a significant difference in life insurance charges based on smoking status.There are the five main steps for conducting this hypothesis test:

Step 1: State the hypotheses:

Null hypothesis (H_0): There is no significant difference in mean charges between smokers and non-smokers.

Alternative hypothesis (H_1): There is a significant difference in mean charges between smokers and non-smokers.

Step 2: Select the significance level:

For the purpose of establishing the cutoff for rejecting the null hypothesis, we must select a significance level (alpha). Let's choose 0.05 because it's a popular setting.

Step 3: Select the appropriate test:

We may do an independent samples t-test since we are contrasting the means of two distinct groups (smokers and non-smokers).

Step 4: Perform the test and calculate the test statistic:

Calculate the mean charges for smokers and non-smokers separately. Then,calculate the standard deviation for charges in both groups.

Calculate the test statistic using the formula: t = (mean1 - mean2) / sqrt((s1^2 / n1) + (s2^2 / n2)), where mean1 and mean2 are the means, s1 and s2 are the standard deviations, and n1 and n2 are the sample sizes.

<u>Step 5: Make a decision:</u>

Compare the calculated test statistic with the critical value (obtained from the t-distribution table) for the chosen significance level. The null hypothesis should be rejected if the computed test statistic exceeds the critical value, which indicates that there is a substantial difference in the mean charges for smokers and non-smokers.

By following these steps, we can perform a hypothesis test comparing the mean charges between smokers and non-smokers. Remember to gather the necessary data, calculate the required statistics, and make a decision based on the significance level.

5.2 CORRELATION ANALYSIS

import pandas as pd

import seaborn as sns

data = pd.read_csv('DatasetC.CSV.csv')

correlation = data[['smoker', 'sex']]

 # Print the correlation matrix

print(correlation)

```
In [1]: import pandas as pd
        import seaborn as sns

        data = pd.read_csv('DatasetC.CSV.csv')
        correlation = data[['smoker', 'sex']]
        # Print the correlation matrix
        print(correlation)

             smoker     sex
        0       yes  female
        1        no    male
        2        no    male
        3        no    male
        4        no    male
        ...     ...     ...
        1333     no    male
        1334     no  female
        1335     no  female
        1336     no  female
        1337    yes  female

        [1338 rows x 2 columns]
```

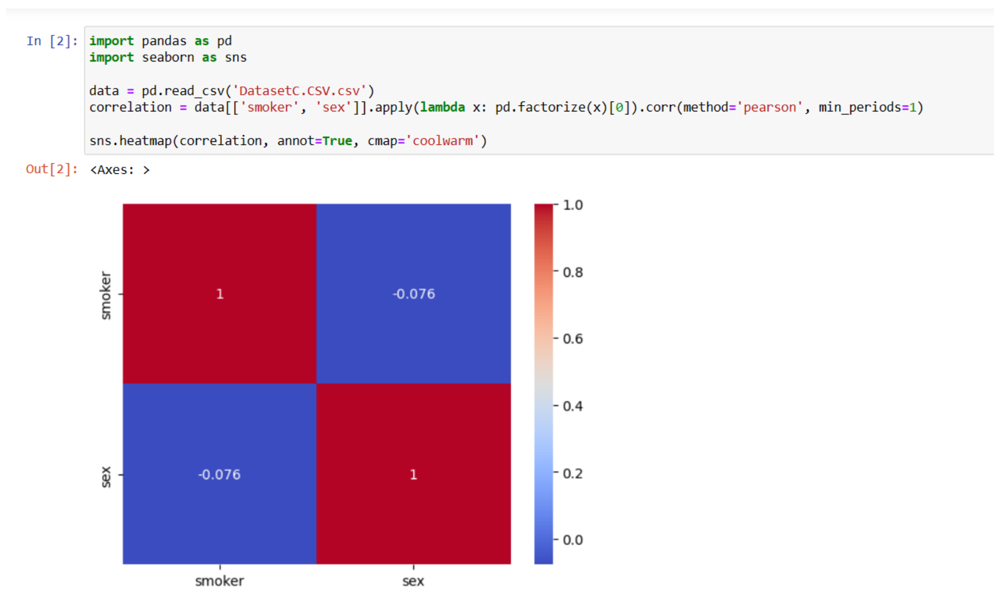
Note: The code is used in jupyter

The resulting correlation matrix heatmap will show the correlation coefficients between "smoker" and "sex". The color intensity in the heatmap will indicate the strength and direction of the correlation.

```
import pandas as pd
import seaborn as sns

data = pd.read_csv('DatasetC.CSV.csv')
correlation = data[['smoker', 'sex']].apply(lambda x: pd.factorize(x)[0]).corr(method='pearson', min_periods=1)

sns.heatmap(correlation, annot=True, cmap='coolwarm')
```



Note: The code is used in jupyter

Please note that since "smoker" and "sex" are categorical variables, the correlation coefficient will indicate the association between the categories rather than a linear relationship. If you are interested in assessing the relationship between categorical variables, the correlation analysis can provide insights into the strength and direction of that association.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import spearmanr, rankdata, pearsonr

data = pd.read_csv('DatasetC.CSV.csv')
df = pd.DataFrame(data,columns=['smoker','sex'])
my_rho = spearmanr(df["smoker"], df["sex"])
my_rho
```

```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from scipy.stats import spearmanr, rankdata, pearsonr

        data = pd.read_csv('DatasetC.CSV.csv')
        df = pd.DataFrame(data,columns=['smoker','sex'])
        my_rho = spearmanr(df["smoker"], df["sex"])
        my_rho

Out[3]: SignificanceResult(statistic=0.07618481692109515, pvalue=0.005300369127374048)

In [ ]:
```

Note: The code is used in jupyter

This indicates that Spearman's rank correlation coefficient between the "smoker" and "sex" columns is approximately 0.076, and the p-value is approximately 0.0053.

5.3 PREDICTION MODELLING

<u>Step 1:</u>

Define our problems: To find the average salary for the degree qualification. By using the liner regression:

$$\hat{y} = b0 + b1x1 + b2x2 + \cdots$$

Explain your model, including the parameters $b0$, $b1$, $b2$, … and the variables $x1$, $x2$, …

Steps 2:

Collect and prepare the data:

Question : $\hat{y} = 2.5 + 3.7x_1 + 5.8x_2 + \ldots$

$B_0 = 2.5$ $\quad$ x_1 = Year of Experience

$B_1 = 3.7$ $\quad$ x_2 = Bechelor's degree qualification

$B_2 = 5.8$ $\quad$ $\hat{y}$ = The numbers of graduate being employed

Steps 3:

-Write the equation in liner.

-change the equation to number that has been given.

-calculate the equation that we already change to number.

Calculation :

$$\hat{y} = b_0 + b_1x_1 + b_2x_2 + \ldots$$

$$\hat{y} = 2.5 + 3.7(4) + 5.8(2) \ldots$$

$$\hat{y} = 2.5 + 14.8 + 11.6 + \ldots$$

$$\hat{y} = 29.9$$

Step 4:

State the conclusion:

This show the coeffional and values of x_1 and x_2 the predicted average salary $\hat{y}$ for and individual with 4 years of experience and a bachelor's degree qualification would be 29.9 units.

6. CONCLUDING REMARKS

As a conclusion, by integrating all these procedures, we can develop a thorough grasp of the data set under study, find significant patterns, come to reliable conclusions, and perhaps even predict the future or provide insights that will help guide future research in our field. With the presence of python, it can help us take some of the data we need from the dataset and various other things.

In an importing data section, this step can upload our dataset into python and enable us to work with the data using various data analysis tools and libraries.

In a data description section, we'll examine the types and characteristics of our dataset including the types of variable present like numerical and categorical. This can help us understand the nature of our data and select appropriate statistical techniques.

In a descriptive measures section, we can determine the frequency of each value or category within a variable and providing insights into the distribution of the data by doing a frequency measures. Measures of central tendency will describe the typical or central value of a variable while measures of position will help us understand the position of individual values within a variable's distribution. Then, measures of variation such as range, variance, and standard deviation will reveal the spread or dispersion of the data points.

In a data visualization section, it can provide a graphical representation of our data which can help identify patterns, outliers, relationships, and distributions then making it easier to interpret our data.

In a statistical inference section, hypothesis testing will allow us to test assumptions or hypothesis about our data using statistical tests. We can determine if there is evidence to support or reject a specific claim or statement. Then, by doing correlation analysis, we can assess the strength and direction of relationships between variables which helps understand how changes in one variable relate changes in another. Lastly, do a prediction modelling can build predictive models using techniques such as linear regression, logistic regression, or machine learning. These models help make predictions future outcomes based on the relationships observed in our dataset.