LONDON
METROPOLITAN
UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

CC5067NI-Smart Data Discovery

60% Individual Coursework

2023-24 Spring

Student Name: Rakesh Chaurasiya

London Met ID: 22085821

College ID: NP01CP4S230153

Assignment Due Date: Monday, May 13, 2024

Assignment Submission Date: Sunday, May 12, 2024

Word Count: 2900

Table of Contents

Table of Figures

Table of Tables

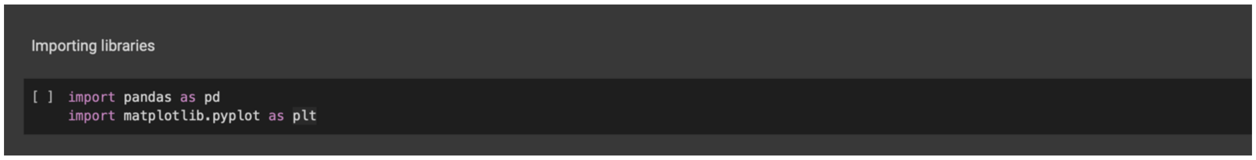
1. Data Understanding

The given dataset is about the different salary ranges of jobs that fall in Data Science and the factors that affect them. The dataset has 11 columns which consist of data such as the work year, experience level of jobs, job titles, job salaries in USD, etc. It has numeric as well as string data. The salary values have been converted into USD, for uniformity, and seem to depend on the other columns in varying amounts. For example, the same job title has different salaries based on experience levels. The table below has short descriptions of each column and their datatype:

S.N.	Column Name	Description	Data Type
1	work_year	The year in which the data was taken.	int64
2	experience_level	The experience level of jobs, which are SE(Senior/Expert), MI(Medium/Intermediate), EN (Entry level), EX (Executive Level).	Object
3	employment_type	The type of employment, which are FT (Full time), CT(Contract), FL(Freelance), PT (Part time).	Object
4	job_title	The name of the jobs.	Object
5	salary	The salary of jobs in the local currency.	int64
6	salary_currency	The local currency of job salary.	String
7	salary_in_usd	The salary of jobs converted into USD.	Int64
8	employee_residence	The country of residence of employees.	Object
9	remote_ratio	The ratio of the number of employees that work remotely to that of employees that work in office.	Int64
10	company_location	The country of location of the company.	Object
11	company_size	Size of the company which is L(large), M(Medium), S(Small).	Object

Table 1 : Descriptions and datatypes of the columns

2. Data Preparation



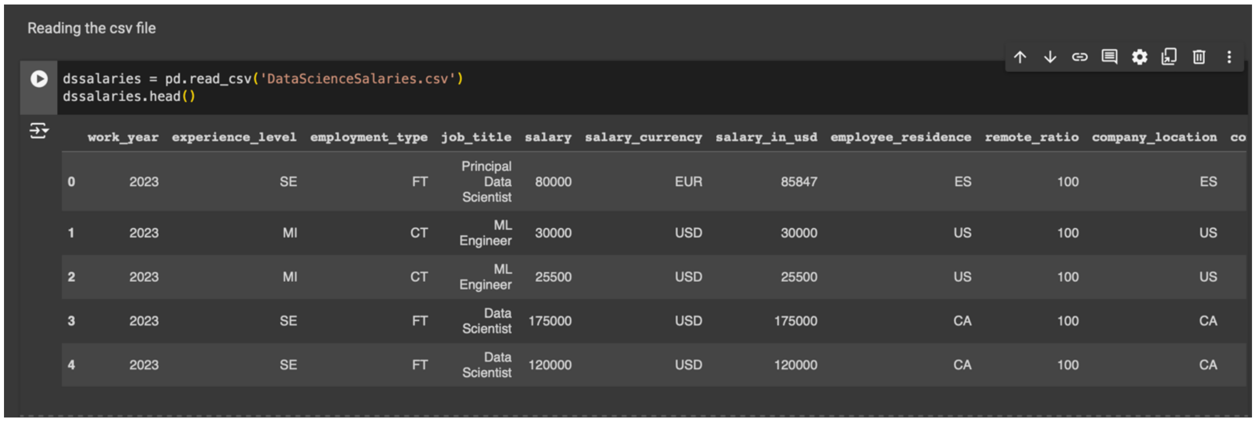
Importing libraries

```
import pandas as pd
import matplotlib.pyplot as plt
```

Figure 1 : Importing the pandas and pyplot libraries

This code imports the pandas and pyplot libraries for the tasks below.

2.1. Write a python program to load data into pandas DataFrame



Reading the csv file

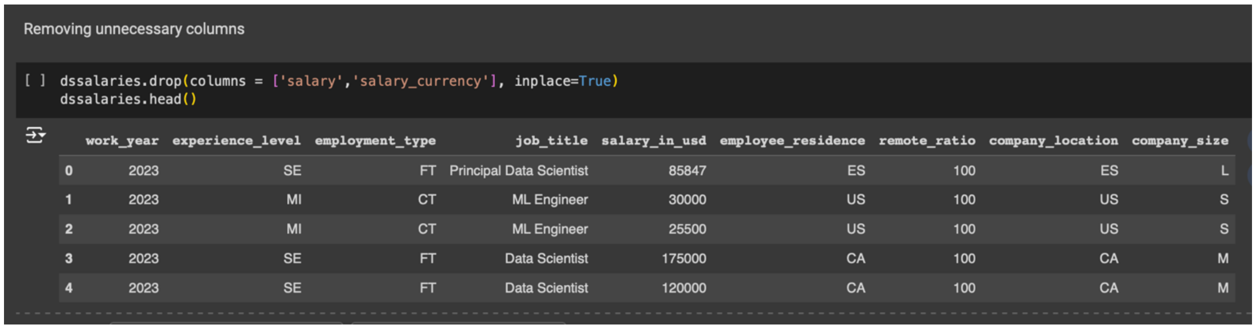
```
dssalaries = pd.read_csv('DataScienceSalaries.csv')
dssalaries.head()
```

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	co
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	ES	100	ES	
1	2023	MI	CT	ML Engineer	30000	USD	30000	US	100	US	
2	2023	MI	CT	ML Engineer	25500	USD	25500	US	100	US	
3	2023	SE	FT	Data Scientist	175000	USD	175000	CA	100	CA	
4	2023	SE	FT	Data Scientist	120000	USD	120000	CA	100	CA	

Figure 2 : Loading the data into pandas dataframe

This code reads the 'DataScienceSalaries.csv' csv file into a Pandas DataFrame called 'dssalaries' and displays the first 5 rows using the head() function.

2.2. Write a python program to remove unnecessary columns i.e., salary and salary currency.



Removing unnecessary columns

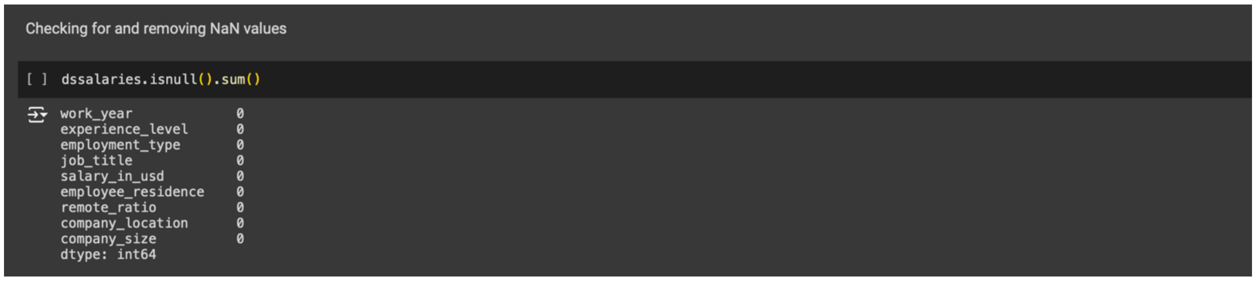
```
[ ] dssalaries.drop(columns = ['salary','salary_currency'], inplace=True)
    dssalaries.head()
```

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	85847	ES	100	ES	L
1	2023	MI	CT	ML Engineer	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	US	100	US	S
3	2023	SE	FT	Data Scientist	175000	CA	100	CA	M
4	2023	SE	FT	Data Scientist	120000	CA	100	CA	M

Figure 3 : Removing the unnecessary columns

This code removes the 'salary' and 'salary_currency' columns from the original DataFrame permanently by setting the inplace parameter to True

2.3. Write a python program to remove the NaN missing values from updated dataframe.



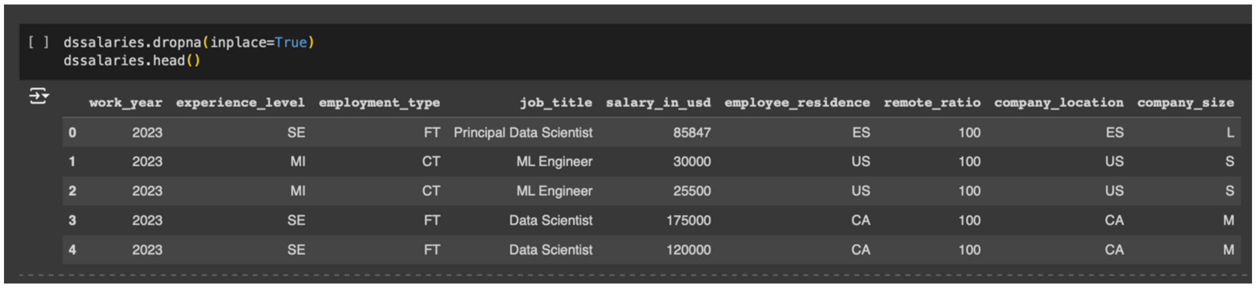
Checking for and removing NaN values

```
[ ] dssalaries.isnull().sum()
```

```
work_year             0
experience_level      0
employment_type       0
job_title             0
salary_in_usd         0
employee_residence    0
remote_ratio          0
company_location      0
company_size          0
dtype: int64
```

Figure 4 : Checking for NaN values in the dataframe

This code checks the DataFrame for missing values (NaN) and calculates the sum of missing values for each column.



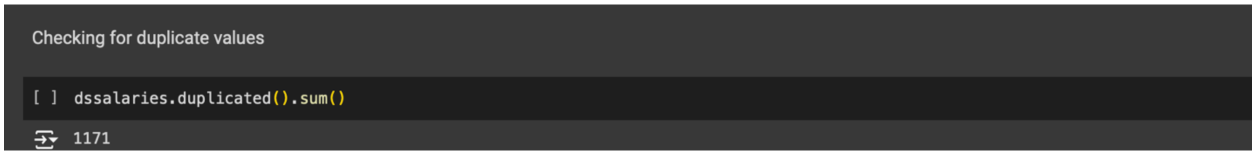
```
[ ] dssalaries.dropna(inplace=True)
    dssalaries.head()
```

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	85847	ES	100	ES	L
1	2023	MI	CT	ML Engineer	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	US	100	US	S
3	2023	SE	FT	Data Scientist	175000	CA	100	CA	M
4	2023	SE	FT	Data Scientist	120000	CA	100	CA	M

Figure 5 : Removing the NaN values from the dataframe

This code removes the rows with missing values from the DataFrame, which in this case is none.

2.4. Write a python program to check duplicates value in the dataframe.


Checking for duplicate values

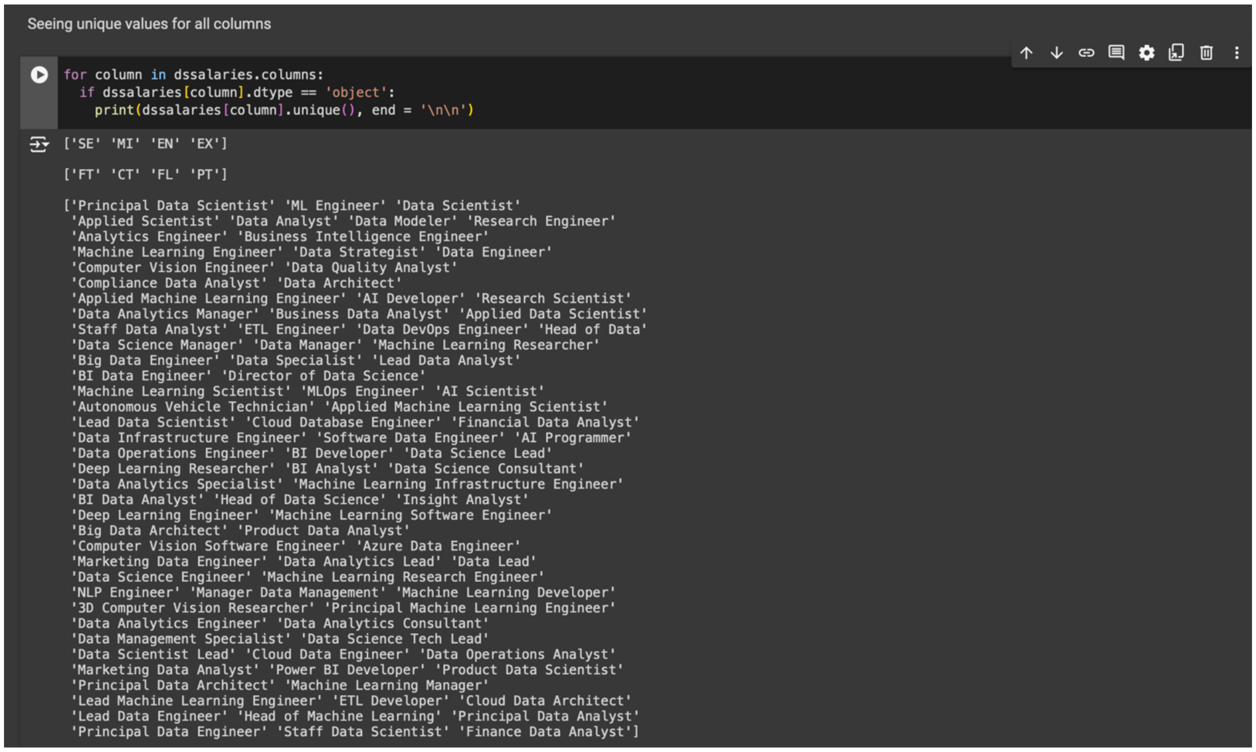
```
[ ] dssalaries.duplicated().sum()
```

```
1171
```

Figure 6 : Checking for duplicate values in the dataframe

This code checks for duplicate entries in the DataFrame and counts the number of duplicate rows found

2.5. Write a python program to see the unique values from all the columns in the dataframe.

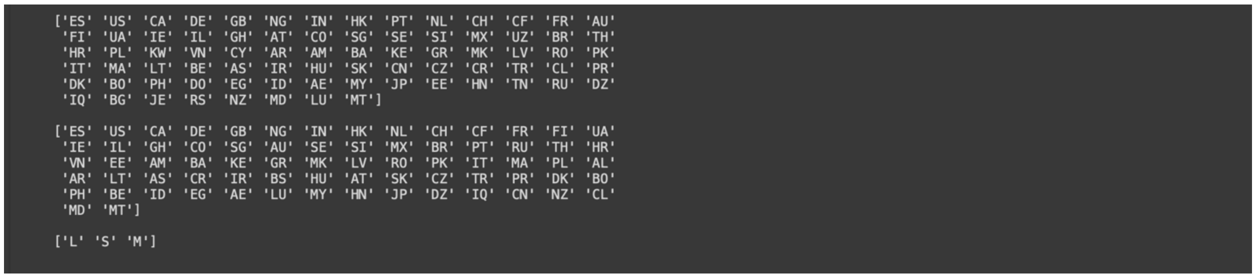


Seeing unique values for all columns

```
for column in dssalaries.columns:
  if dssalaries[column].dtype == 'object':
    print(dssalaries[column].unique(), end = '\n\n')
```

```
['SE' 'MI' 'EN' 'EX']

['FT' 'CT' 'FL' 'PT']

['Principal Data Scientist' 'ML Engineer' 'Data Scientist'
 'Applied Scientist' 'Data Analyst' 'Data Modeler' 'Research Engineer'
 'Analytics Engineer' 'Business Intelligence Engineer'
 'Machine Learning Engineer' 'Data Strategist' 'Data Engineer'
 'Computer Vision Engineer' 'Data Quality Analyst'
 'Compliance Data Analyst' 'Data Architect'
 'Applied Machine Learning Engineer' 'AI Developer' 'Research Scientist'
 'Data Analytics Manager' 'Business Data Analyst' 'Applied Data Scientist'
 'Staff Data Analyst' 'ETL Engineer' 'Data DevOps Engineer' 'Head of Data'
 'Data Science Manager' 'Data Manager' 'Machine Learning Researcher'
 'Big Data Engineer' 'Data Specialist' 'Lead Data Analyst'
 'BI Data Engineer' 'Director of Data Science'
 'Machine Learning Scientist' 'MLOps Engineer' 'AI Scientist'
 'Autonomous Vehicle Technician' 'Applied Machine Learning Scientist'
 'Lead Data Scientist' 'Cloud Database Engineer' 'Financial Data Analyst'
 'Data Infrastructure Engineer' 'Software Data Engineer' 'AI Programmer'
 'Data Operations Engineer' 'BI Developer' 'Data Science Lead'
 'Deep Learning Researcher' 'BI Analyst' 'Data Science Consultant'
 'Data Analytics Specialist' 'Machine Learning Infrastructure Engineer'
 'BI Data Analyst' 'Head of Data Science' 'Insight Analyst'
 'Deep Learning Engineer' 'Machine Learning Software Engineer'
 'Big Data Architect' 'Product Data Analyst'
 'Computer Vision Software Engineer' 'Azure Data Engineer'
 'Marketing Data Engineer' 'Data Analytics Lead' 'Data Lead'
 'Data Science Engineer' 'Machine Learning Research Engineer'
 'NLP Engineer' 'Manager Data Management' 'Machine Learning Developer'
 '3D Computer Vision Researcher' 'Principal Machine Learning Engineer'
 'Data Analytics Engineer' 'Data Analytics Consultant'
 'Data Management Specialist' 'Data Science Tech Lead'
 'Data Scientist Lead' 'Cloud Data Engineer' 'Data Operations Analyst'
 'Marketing Data Analyst' 'Power BI Developer' 'Product Data Scientist'
 'Principal Data Architect' 'Machine Learning Manager'
 'Lead Machine Learning Engineer' 'ETL Developer' 'Cloud Data Architect'
 'Lead Data Engineer' 'Head of Machine Learning' 'Principal Data Analyst'
 'Principal Data Engineer' 'Staff Data Scientist' 'Finance Data Analyst']
```

Figure 7 : Seeing unique values from each column – 1

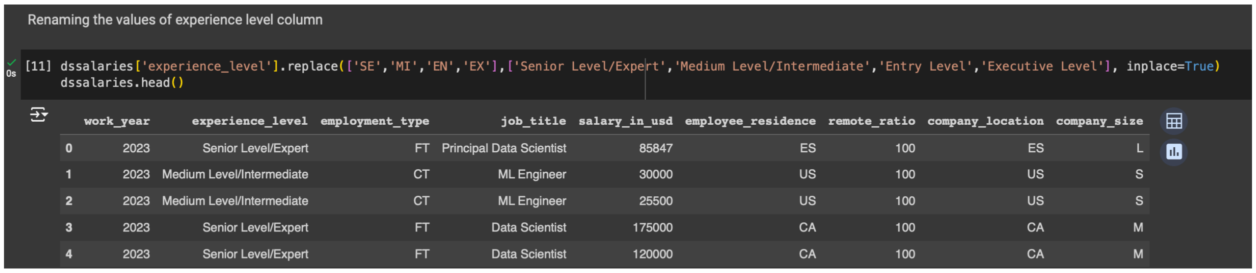


```
['ES' 'US' 'CA' 'DE' 'GB' 'NG' 'IN' 'HK' 'PT' 'NL' 'CH' 'CF' 'FR' 'AU'
 'FI' 'UA' 'IE' 'IL' 'GH' 'AT' 'CO' 'SG' 'SE' 'SI' 'MX' 'UZ' 'BR' 'TH'
 'HR' 'PL' 'KW' 'VN' 'CY' 'AR' 'AM' 'BA' 'KE' 'GR' 'MK' 'LV' 'RO' 'PK'
 'IT' 'MA' 'LT' 'BE' 'AS' 'IR' 'HU' 'SK' 'CN' 'CZ' 'CR' 'TR' 'CL' 'PR'
 'DK' 'BO' 'PH' 'DO' 'EG' 'ID' 'AE' 'MY' 'JP' 'EE' 'HN' 'TN' 'RU' 'DZ'
 'IQ' 'BG' 'JE' 'RS' 'NZ' 'MD' 'LU' 'MT']

['ES' 'US' 'CA' 'DE' 'GB' 'NG' 'IN' 'HK' 'NL' 'CH' 'CF' 'FR' 'FI' 'UA'
 'IE' 'IL' 'GH' 'CO' 'SG' 'AU' 'SE' 'SI' 'MX' 'BR' 'PT' 'RU' 'TH' 'HR'
 'VN' 'EE' 'AM' 'BA' 'KE' 'GR' 'MK' 'LV' 'RO' 'PK' 'IT' 'MA' 'PL' 'AL'
 'AR' 'LT' 'AS' 'CR' 'IR' 'BS' 'HU' 'AT' 'SK' 'CZ' 'TR' 'PR' 'DK' 'BO'
 'PH' 'BE' 'ID' 'EG' 'AE' 'LU' 'MY' 'HN' 'JP' 'DZ' 'IQ' 'CN' 'NZ' 'CL'
 'MD' 'MT']

['L' 'S' 'M']
```

Figure 8 : Seeing unique values from each column – 2

This code iterates through each column in a DataFrame. If the data type of a column is 'object', it prints the unique values in that column.

2.6. Rename the experience level columns as below.



Renaming the values of experience level column

```
[11] dssalaries['experience_level'].replace(['SE','MI','EN','EX'],['Senior Level/Expert','Medium Level/Intermediate','Entry Level','Executive Level'], inplace=True)
     dssalaries.head()
```

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	Senior Level/Expert	FT	Principal Data Scientist	85847	ES	100	ES	L
1	2023	Medium Level/Intermediate	CT	ML Engineer	30000	US	100	US	S
2	2023	Medium Level/Intermediate	CT	ML Engineer	25500	US	100	US	S
3	2023	Senior Level/Expert	FT	Data Scientist	175000	CA	100	CA	M
4	2023	Senior Level/Expert	FT	Data Scientist	120000	CA	100	CA	M

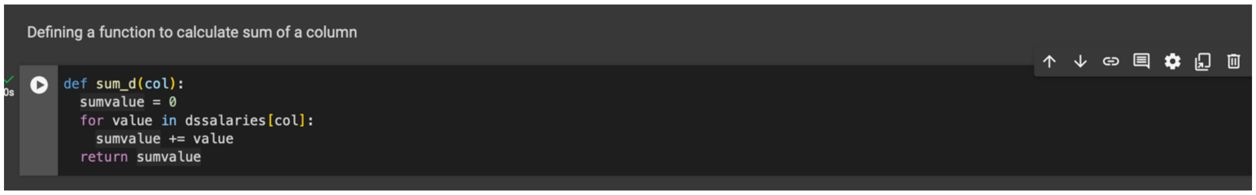
Figure 9 : Renaming the experience level columns

This code replaces the values in the 'experience_level' column of the DataFrame with corresponding values as per the requirement above.

3. Data Analysis

3.1. Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of any chosen variable.

3.1.1. Sum

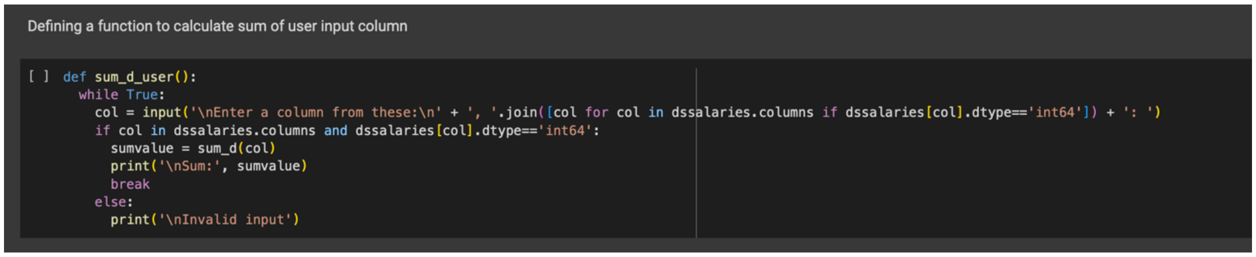


Defining a function to calculate sum of a column

```
def sum_d(col):
  sumvalue = 0
  for value in dssalaries[col]:
    sumvalue += value
  return sumvalue
```

Figure 10 : Defining a function to calculate the sum of a column

This code defines a function named sum_d that takes a column as input. Inside the function, it initializes a variable sumvalue to zero to store the sum of values. It then iterates over the values in the specified column of the DataFrame, adding each value to sumvalue. Finally, it returns the calculated sum



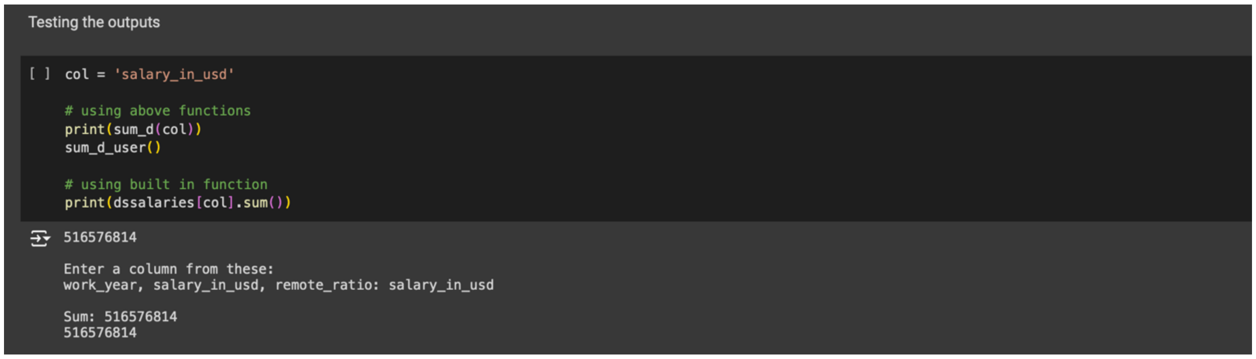
Defining a function to calculate sum of user input column

```
[ ] def sum_d_user():
      while True:
        col = input('\nEnter a column from these:\n' + ', '.join([col for col in dssalaries.columns if dssalaries[col].dtype=='int64']) + ': ')
        if col in dssalaries.columns and dssalaries[col].dtype=='int64':
          sumvalue = sum_d(col)
          print('\nSum:', sumvalue)
          break
        else:
          print('\nInvalid input')
```

Figure 11 : Defining a function to ask for user input to calculate the sum of a column

This code defines another function named sum_d_user without any parameters. Within this function, a while loop is used to repeatedly prompt the user to input a column name from the DataFrame. The user's input is checked to ensure it exists in the DataFrame and is of integer type (dtype='int64'). If a valid column name is entered, the sum_d function is called with the input column name, and the calculated sum is printed. If the input is invalid, an error message is displayed, and the user is asked to enter a valid column name again.

.



Testing the outputs

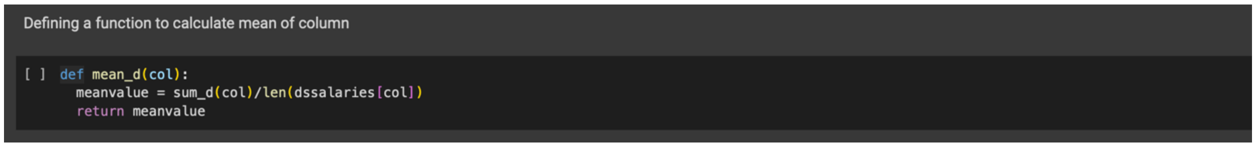
```
col = 'salary_in_usd'

# using above functions
print(sum_d(col))
sum_d_user()

# using built in function
print(dssalaries[col].sum())
```

```
516576814

Enter a column from these:
work_year, salary_in_usd, remote_ratio: salary_in_usd

Sum: 516576814
516576814
```

Figure 12 : Testing the outputs of sum functions

This code compares the functions above with built-in sum function. It shows that the total sum of all salaries in the dataset is $516,576,814.

3.1.2. Mean

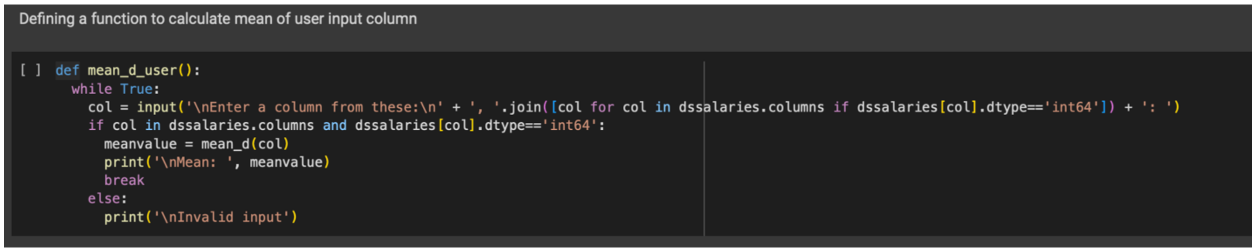


Defining a function to calculate mean of column

```
[ ] def mean_d(col):
      meanvalue = sum_d(col)/len(dssalaries[col])
      return meanvalue
```

Figure 13 : Defining a function to calculate the mean of a column

This function calculates the mean of values in the column specified. It utilizes the previously defined sum_d function to calculate the sum of values in the column, then divides this sum by the number of values in the column (len(dssalaries[col])) to calculate the mean. The mean value is then returned.

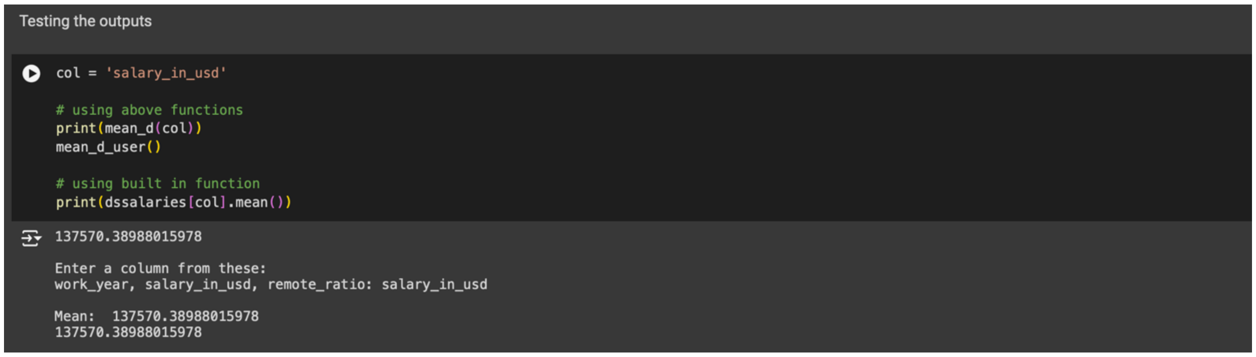


Defining a function to calculate mean of user input column

```
[ ] def mean_d_user():
      while True:
        col = input('\nEnter a column from these:\n' + ', '.join([col for col in dssalaries.columns if dssalaries[col].dtype=='int64']) + ': ')
        if col in dssalaries.columns and dssalaries[col].dtype=='int64':
          meanvalue = mean_d(col)
          print('\nMean: ', meanvalue)
          break
        else:
          print('\nInvalid input')
```

Figure 14 : Defining a function to ask for user input to calculate the mean of a column

This function asks the user to input a column name from the DataFrame for which they want to calculate the mean. It repeatedly asks the user until an existing column name of integer type is entered. Once a valid column name is entered, the mean_d function is called with the input column name, and the calculated mean value is printed.



Testing the outputs

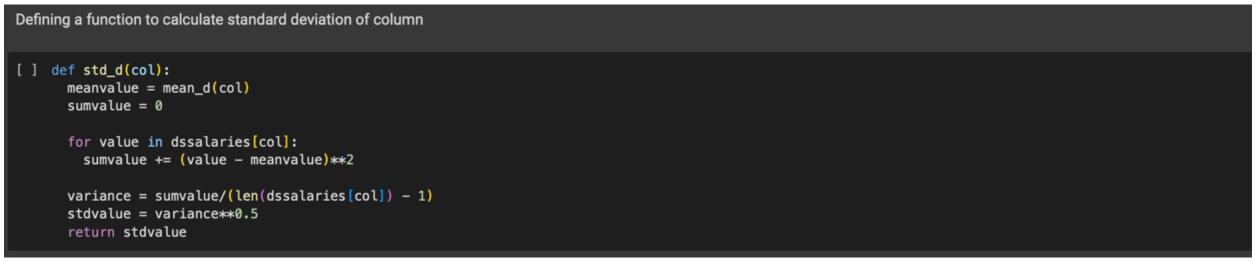
```
col = 'salary_in_usd'

# using above functions
print(mean_d(col))
mean_d_user()

# using built in function
print(dssalaries[col].mean())
```

```
137570.38988015978

Enter a column from these:
work_year, salary_in_usd, remote_ratio: salary_in_usd

Mean:  137570.38988015978
137570.38988015978
```

Figure 15 : Testing the outputs of mean functions

This code compares the functions above with built-in mean function. It is seen here that the mean of salary_in_usd is 137570.389, meaning that the average salary is around $137,570.

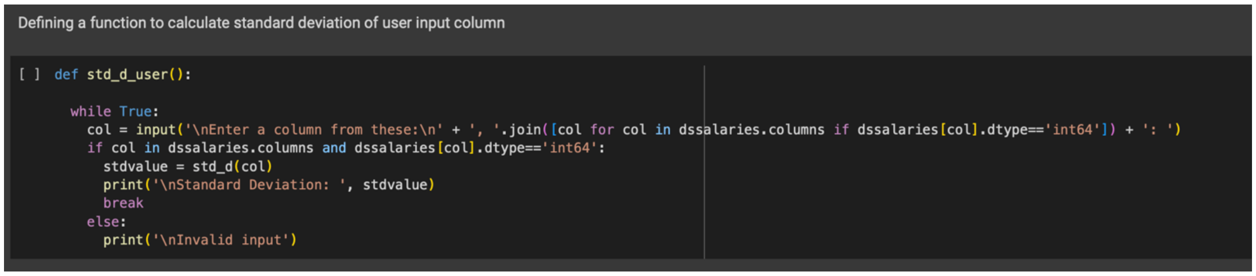
3.1.3. Standard deviation


Defining a function to calculate standard deviation of column

```
[ ] def std_d(col):
      meanvalue = mean_d(col)
      sumvalue = 0

      for value in dssalaries[col]:
        sumvalue += (value - meanvalue)**2

      variance = sumvalue/(len(dssalaries[col]) - 1)
      stdvalue = variance**0.5
      return stdvalue
```

Figure 16 : Defining a function to calculate the standard deviation of a column

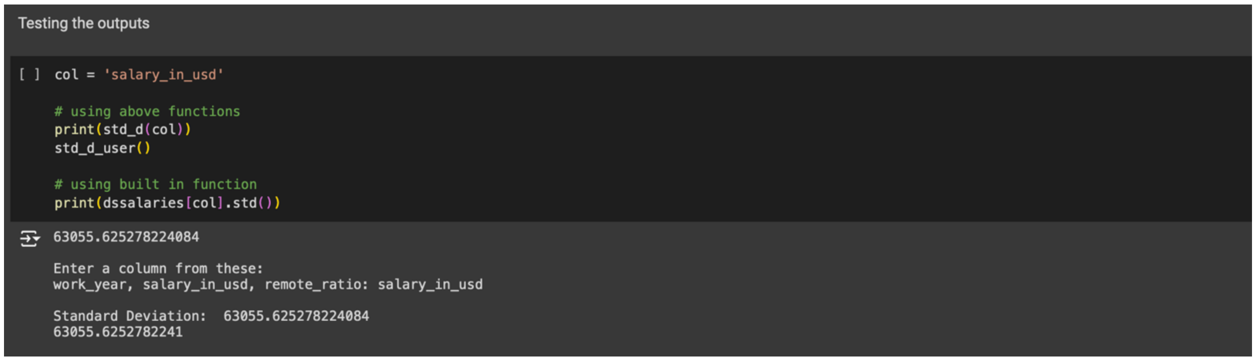
This function calculates the standard deviation of values in the column. It first calculates the mean of the column using the previously defined mean_d function. Then, it iterates through the values in the column, computing the sum of squared differences between each value and the mean. It then calculates the variance by dividing that sum by the sample size (len(dssalaries[col]) - 1). Finally, it calculates the standard deviation by taking the square root of the variance, and returns the standard deviation value.


Defining a function to calculate standard deviation of user input column

```
[ ] def std_d_user():

      while True:
        col = input('\nEnter a column from these:\n' + ', '.join([col for col in dssalaries.columns if dssalaries[col].dtype=='int64']) + ': ')
        if col in dssalaries.columns and dssalaries[col].dtype=='int64':
          stdvalue = std_d(col)
          print('\nStandard Deviation: ', stdvalue)
          break
        else:
          print('\nInvalid input')
```

Figure 17 : Defining a function to ask for user input to calculate the standard deviation of a column

This function asks the user to input a column name from the DataFrame for which they want to calculate the standard deviation. It keeps on asking for input until an existing column name of integer type (dtype=='int64') is entered. Once a valid column name is provided, the std_d function is called with the input column name, and the calculated standard deviation value is printed.



Testing the outputs

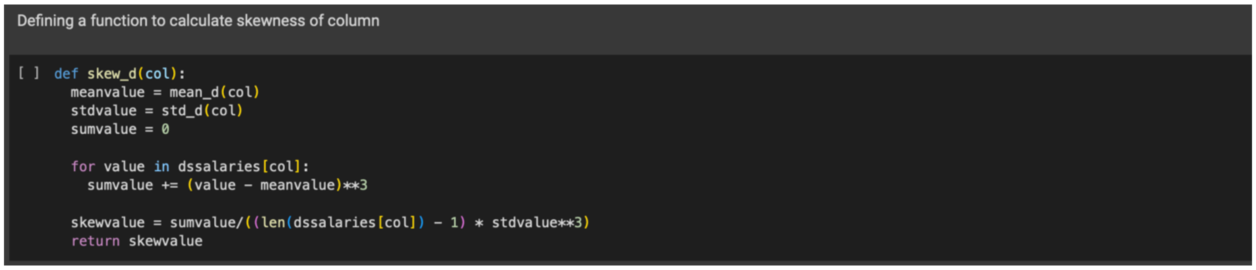
```
[ ]  col = 'salary_in_usd'

     # using above functions
     print(std_d(col))
     std_d_user()

     # using built in function
     print(dssalaries[col].std())
```

```
63055.625278224084

Enter a column from these:
work_year, salary_in_usd, remote_ratio: salary_in_usd

Standard Deviation:  63055.625278224084
63055.6252782241
```

Figure 18 : Testing the outputs of standard deviation functions

This code compares the functions above with built-in standard deviation function. It is seen that the standard deviation is 63055.62, meaning that values are dispersed widely around the mean. This implies that there are not many salaries that fall near the mean.

3.1.4. Skewness

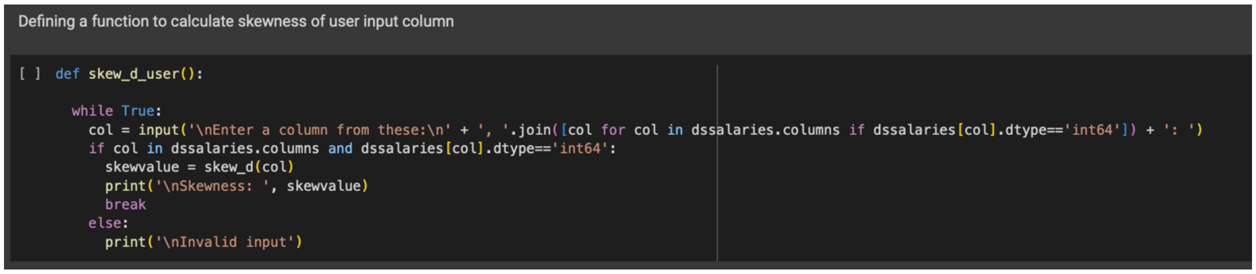


Defining a function to calculate skewness of column

```
def skew_d(col):
  meanvalue = mean_d(col)
  stdvalue = std_d(col)
  sumvalue = 0

  for value in dssalaries[col]:
    sumvalue += (value - meanvalue)**3

  skewvalue = sumvalue/((len(dssalaries[col]) - 1) * stdvalue**3)
  return skewvalue
```

Figure 19 : Defining a function to calculate the skewness of a column

This function calculates the skewness of values in the column. It first calculates the mean and standard deviation of the column using the previously defined mean_d and std_d functions. Then, it iterates through the values in the column, calculating the sum of cubed differences between each value and the mean. Then it divides this sum by the product of the sample size minus one and the cube of the standard deviation, and returns the skewness value.

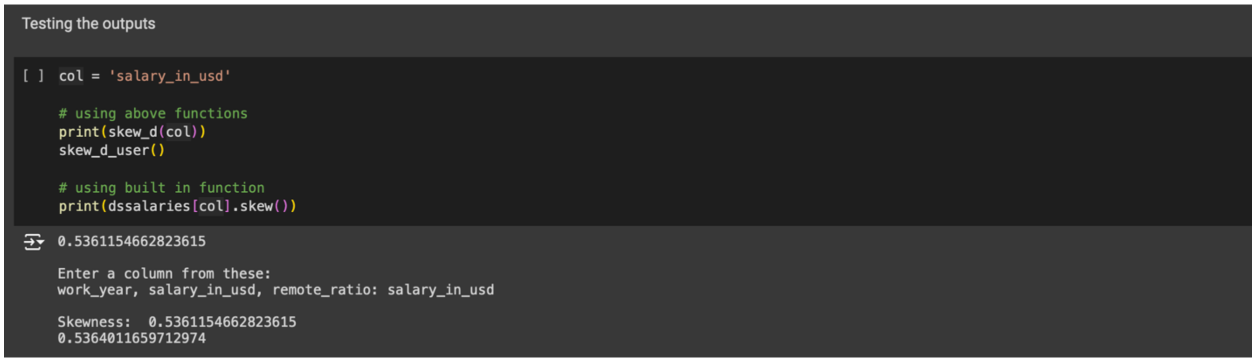


Defining a function to calculate skewness of user input column

```
def skew_d_user():

  while True:
    col = input('\nEnter a column from these:\n' + ', '.join([col for col in dssalaries.columns if dssalaries[col].dtype=='int64']) + ': ')
    if col in dssalaries.columns and dssalaries[col].dtype=='int64':
      skewvalue = skew_d(col)
      print('\nSkewness: ', skewvalue)
      break
    else:
      print('\nInvalid input')
```

Figure 20 : Defining a function to ask for user input to calculate the skewness of a column

This function asks the user to input a column name from the DataFrame for which they want to calculate the skewness until an existing column name of integer type (dtype=='int64') is entered. Once a valid column name is provided, the skew_d function is called with the input column name, and the calculated skewness value is printed.



Testing the outputs

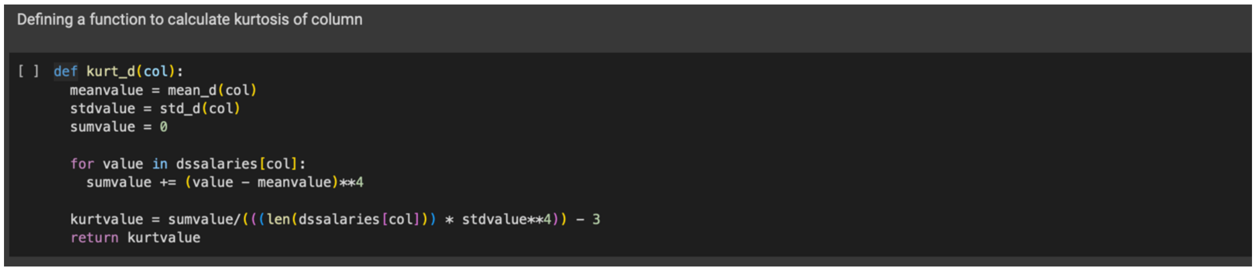
```
col = 'salary_in_usd'

# using above functions
print(skew_d(col))
skew_d_user()

# using built in function
print(dssalaries[col].skew())
```

```
0.5361154662823615

Enter a column from these:
work_year, salary_in_usd, remote_ratio: salary_in_usd

Skewness:  0.5361154662823615
0.5364011659712974
```

Figure 21 : Testing the outputs of skewness functions

This code compares the functions above with built-in skewness function. The picture here shows that the skewness of salary_in_usd is 0.53, meaning that the data is positively skewed. This implies that many salaries are more than the median value of the column.

3.1.5. Kurtosis



Defining a function to calculate kurtosis of column

```
[ ] def kurt_d(col):
      meanvalue = mean_d(col)
      stdvalue = std_d(col)
      sumvalue = 0

      for value in dssalaries[col]:
        sumvalue += (value - meanvalue)**4

      kurtvalue = sumvalue/(((len(dssalaries[col])) * stdvalue**4)) - 3
      return kurtvalue
```

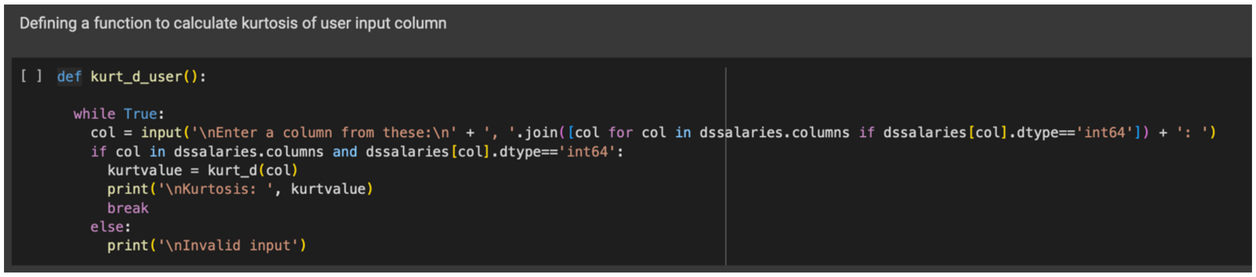
Figure 22 : Defining a function to calculate the kurtosis of a column

Kurtosis is a measurement that indicates the frequency of occurrence of outliers in a dataset. It is mainly of 3 types:

- Leptokurtic: A dataset is leptokurtic if it has a high number of outliers, or has a kurtosis value greater than 3.
- Mesokurtic: A dataset is mesokurtic if it has a moderate number of outliers in a dataset, or has a kurtosis value of approximately 3.
- Platykurtic: A dataset is platykurtic if it has very few outliers, or has a kurtosis value less than 3.

Kurtosis values are often calculated according to excess kurtosis, where the actual kurtosis value is subtracted by 3. A normal dataset would have this value of kurtosis as 0. (Frost, 2022)

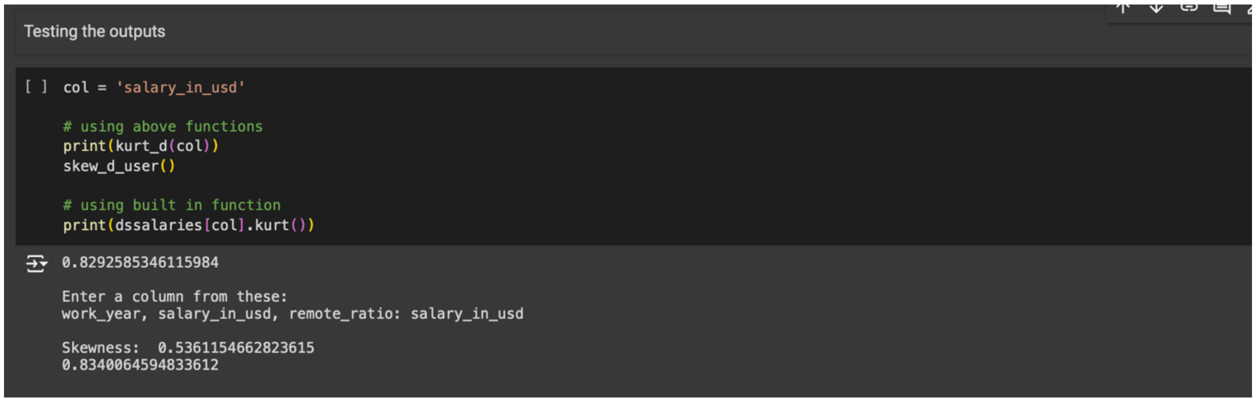
This function calculates the kurtosis of values in the column. It first calculates the mean and standard deviation of the column using the previously defined mean_d and std_d functions. Then, it iterates through the values in the column, computing the sum of 4th power differences between each value and the mean. Then, it divides this sum by the product of the sample size, the fourth power of the standard deviation, and subtracts 3 from the result. This calculation is based on the definition of excess kurtosis, where a normal distribution has a kurtosis of 0.


Defining a function to calculate kurtosis of user input column

```
[ ] def kurt_d_user():

      while True:
        col = input('\nEnter a column from these:\n' + ', '.join([col for col in dssalaries.columns if dssalaries[col].dtype=='int64']) + ': ')
        if col in dssalaries.columns and dssalaries[col].dtype=='int64':
          kurtvalue = kurt_d(col)
          print('\nKurtosis: ', kurtvalue)
          break
        else:
          print('\nInvalid input')
```

Figure 23 : Defining a function to ask for user input to calculate the kurtosis of a column

This function asks the user to input a column name from the DataFrame for which they want to calculate the kurtosis until an existing column name of integer type (dtype=='int64') is entered. Once a valid column name is provided, the kurt_d function is called with the input column name, and the calculated kurtosis value is printed.


Testing the outputs

```
[ ] col = 'salary_in_usd'

    # using above functions
    print(kurt_d(col))
    skew_d_user()

    # using built in function
    print(dssalaries[col].kurt())
```

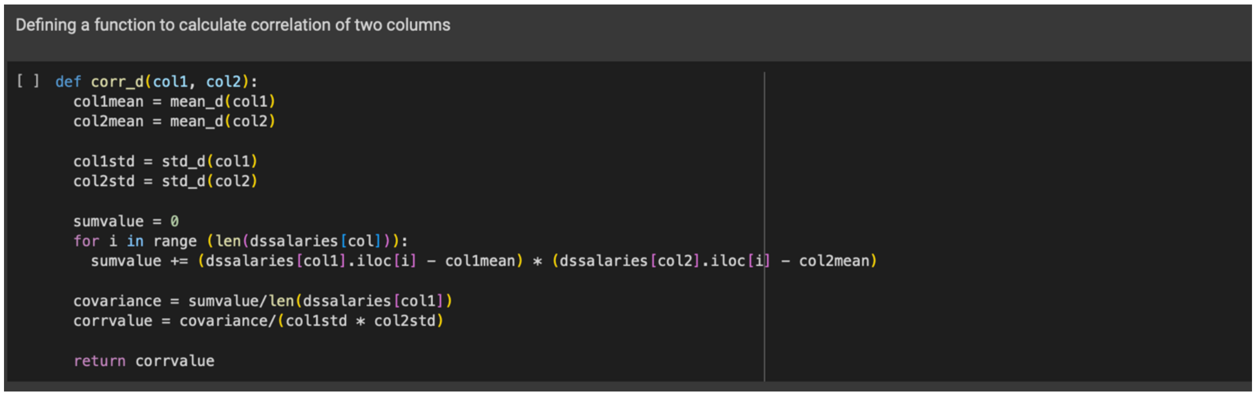
```
0.8292585346115984

Enter a column from these:
work_year, salary_in_usd, remote_ratio: salary_in_usd

Skewness:  0.5361154662823615
0.8340064594833612
```

Figure 24 : Testing the outputs of kurtosis functions

This code compares the functions above with built-in kurtosis function. Here, it can be seen that the kurtosis of salary_in_usd is 0.83, which means the distribution is leptokurtic, i.e., there are many salaries with very high and very low values.

3.2. Write a Python program to calculate and show correlation of all variables.

3.2.1. Correlation

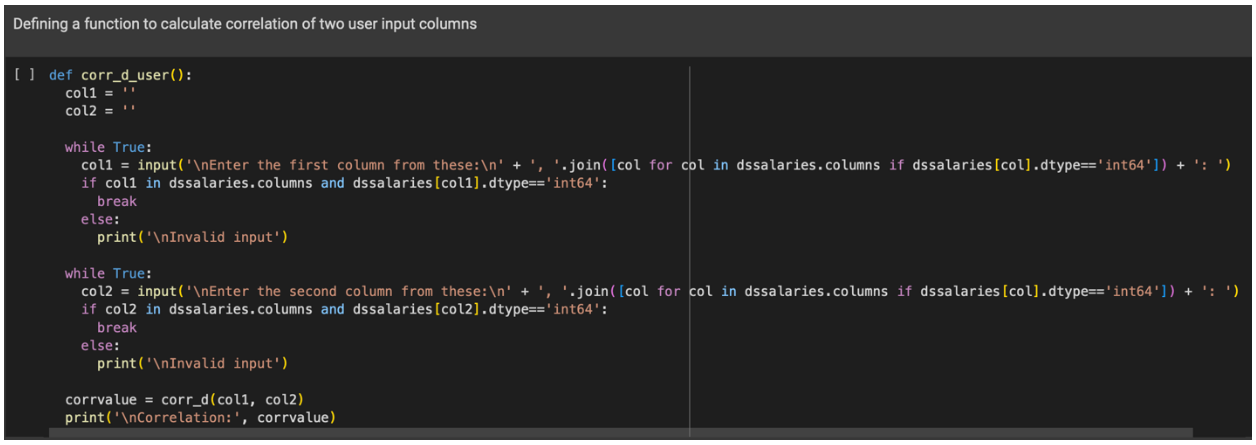

Defining a function to calculate correlation of two columns

```
[ ] def corr_d(col1, col2):
      col1mean = mean_d(col1)
      col2mean = mean_d(col2)

      col1std = std_d(col1)
      col2std = std_d(col2)

      sumvalue = 0
      for i in range (len(dssalaries[col])):
        sumvalue += (dssalaries[col1].iloc[i] - col1mean) * (dssalaries[col2].iloc[i] - col2mean)

      covariance = sumvalue/len(dssalaries[col1])
      corrvalue = covariance/(col1std * col2std)

      return corrvalue
```

Figure 25 : Defining a function to calculate the correlation of two columns

Correlation is the measurement of the strength of the relationship and rate of change between two variables. It ranges from -1 to 1, where -1 means that if one variable increases, the other decreases and vice versa, while 1 means that if one variable increases, the other also increases. 0 means that there is no relationship between the two variables. (Bhandari, 2023)

This function calculates the correlation between two columns. It first calculates the means and standard deviations of both columns using the previously defined mean_d and std_d functions. Then, it iterates through the values of both columns, calculating the sum of the product of the differences between each value and their respective means. it divides the sum by the length of the first column to get the covariance. Finally, it divides the covariance by the product of the standard deviations of both columns to get the correlation, and returns this value.

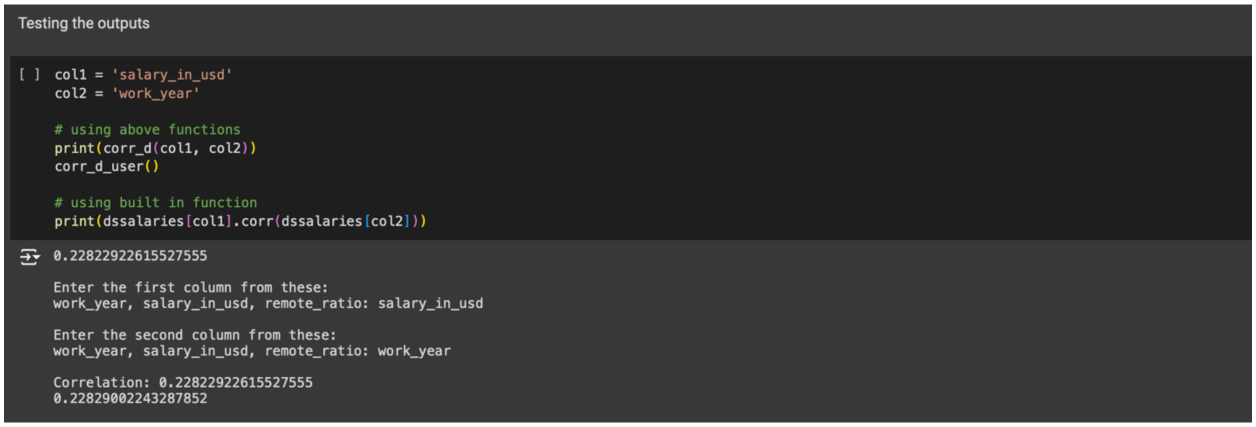


Defining a function to calculate correlation of two user input columns

```
[ ] def corr_d_user():
      col1 = ''
      col2 = ''

      while True:
        col1 = input('\nEnter the first column from these:\n' + ', '.join([col for col in dssalaries.columns if dssalaries[col].dtype=='int64']) + ': ')
        if col1 in dssalaries.columns and dssalaries[col1].dtype=='int64':
          break
        else:
          print('\nInvalid input')

      while True:
        col2 = input('\nEnter the second column from these:\n' + ', '.join([col for col in dssalaries.columns if dssalaries[col].dtype=='int64']) + ': ')
        if col2 in dssalaries.columns and dssalaries[col2].dtype=='int64':
          break
        else:
          print('\nInvalid input')

      corrvalue = corr_d(col1, col2)
      print('\nCorrelation:', corrvalue)
```

Figure 26 : Defining a function to ask for user input to calculate the correlation of two columns

This function prompts the user to input two column names from the DataFrame for which they want to calculate the correlation, until existing column names of integer type (dtype=='int64') are entered for both columns. Once valid column names are entered, the corr_d function is called with the input colums, and the calculated correlation value is printed.



Testing the outputs

```
[ ] col1 = 'salary_in_usd'
    col2 = 'work_year'

    # using above functions
    print(corr_d(col1, col2))
    corr_d_user()

    # using built in function
    print(dssalaries[col1].corr(dssalaries[col2]))
```

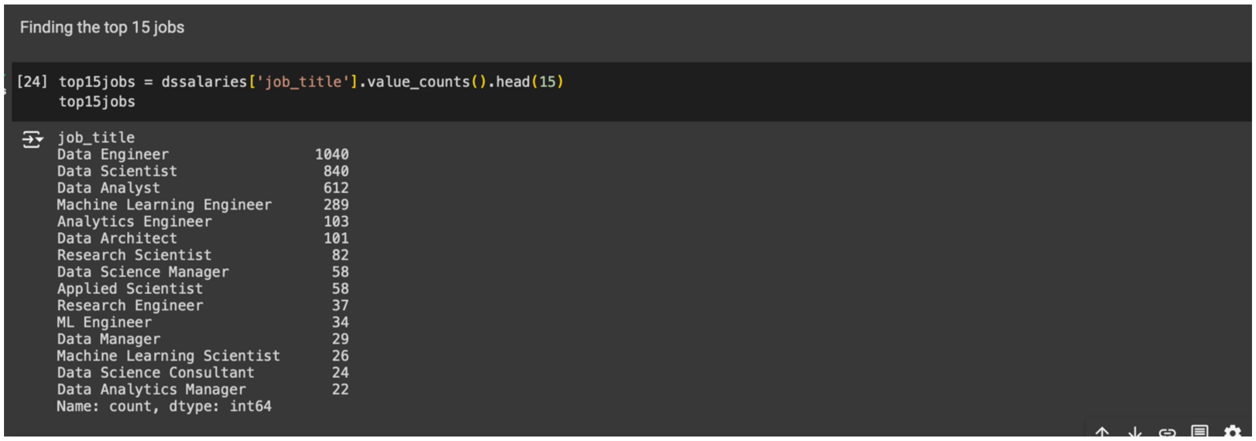
```
0.22822922615527555

Enter the first column from these:
work_year, salary_in_usd, remote_ratio: salary_in_usd

Enter the second column from these:
work_year, salary_in_usd, remote_ratio: work_year

Correlation: 0.22822922615527555
0.22829002243287852
```

Figure 27 : Testing the outputs of correlation functions

This code compares the functions above with built-in correlation function. It is seen that the correlation between salary_in_usd and work_year is positive 0.22, which means that the two columns are loosely related and one increases if the other increases and vice versa.

4. Data Exploration

4.1. Write a python program to find out top 15 jobs. Make a bar graph of sales as well.



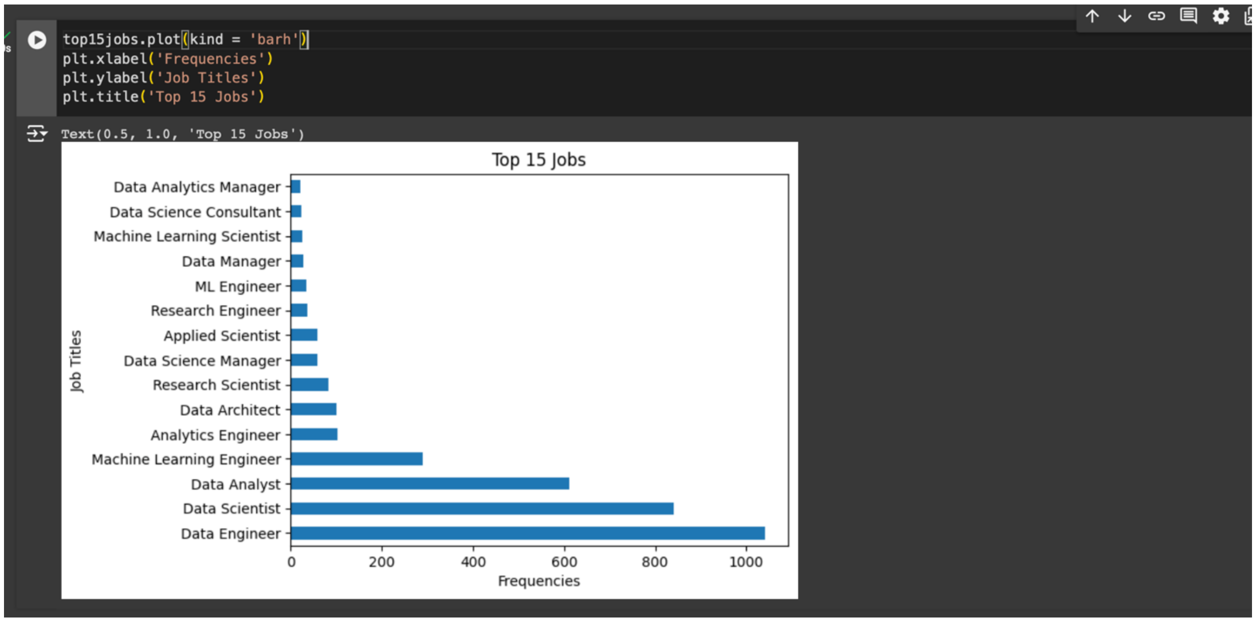
Finding the top 15 jobs

```
[24] top15jobs = dssalaries['job_title'].value_counts().head(15)
     top15jobs
```

```
job_title
Data Engineer                1040
Data Scientist                840
Data Analyst                  612
Machine Learning Engineer     289
Analytics Engineer            103
Data Architect                101
Research Scientist             82
Data Science Manager           58
Applied Scientist              58
Research Engineer              37
ML Engineer                    34
Data Manager                   29
Machine Learning Scientist     26
Data Science Consultant        24
Data Analytics Manager         22
Name: count, dtype: int64
```

Figure 28 : Finding out the top 15 jobs

This code creates a list containing the top 15 most common jobs from the 'job_title' column of the DataFrame. It does this by using the `value_counts()` method to count the occurrences of each unique job title, and then selecting the first 15 rows using the `head(15)` method.



```
top15jobs.plot(kind = 'barh')
plt.xlabel('Frequencies')
plt.ylabel('Job Titles')
plt.title('Top 15 Jobs')
```

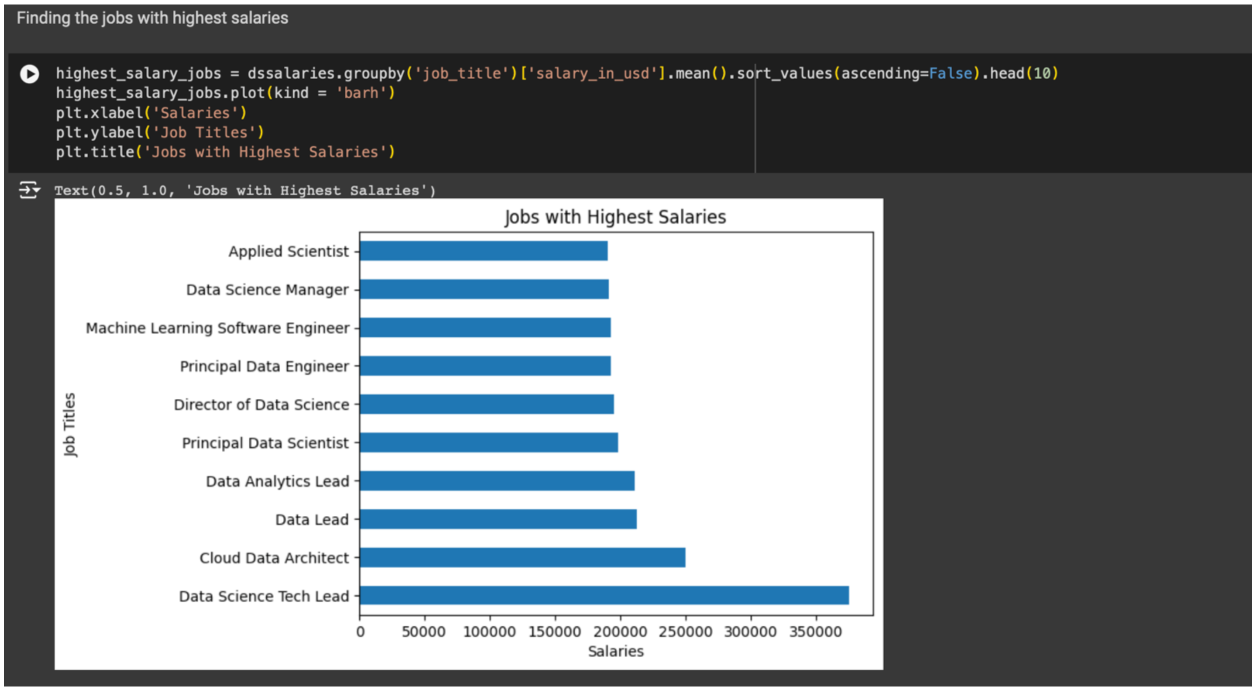
```
Text(0.5, 1.0, 'Top 15 Jobs')
```

Figure 29 : Plotting a bar graph of the top 15 jobs

Here, top15jobs.plot(kind='barh') plots the data from the top15jobs list as a horizontal bar graph ('barh' stands for horizontal bar plot). plt.xlabel('Frequencies') sets the label for the x-axis as 'Frequencies', and plt.ylabel('Job Titles') sets the label for the y-axis as 'Job Titles'. plt.title('Top 15 Jobs') sets the title of the plot as 'Top 15 Jobs'.

It is seen here that Data Engineer is the most common job, followed by jobs like Data Scientist, Data Analyst, etc.

4.1.2. Which job has the highest salaries? Illustrate with bar graph.


Finding the jobs with highest salaries

```
highest_salary_jobs = dssalaries.groupby('job_title')['salary_in_usd'].mean().sort_values(ascending=False).head(10)
highest_salary_jobs.plot(kind = 'barh')
plt.xlabel('Salaries')
plt.ylabel('Job Titles')
plt.title('Jobs with Highest Salaries')
```

```
Text(0.5, 1.0, 'Jobs with Highest Salaries')
```

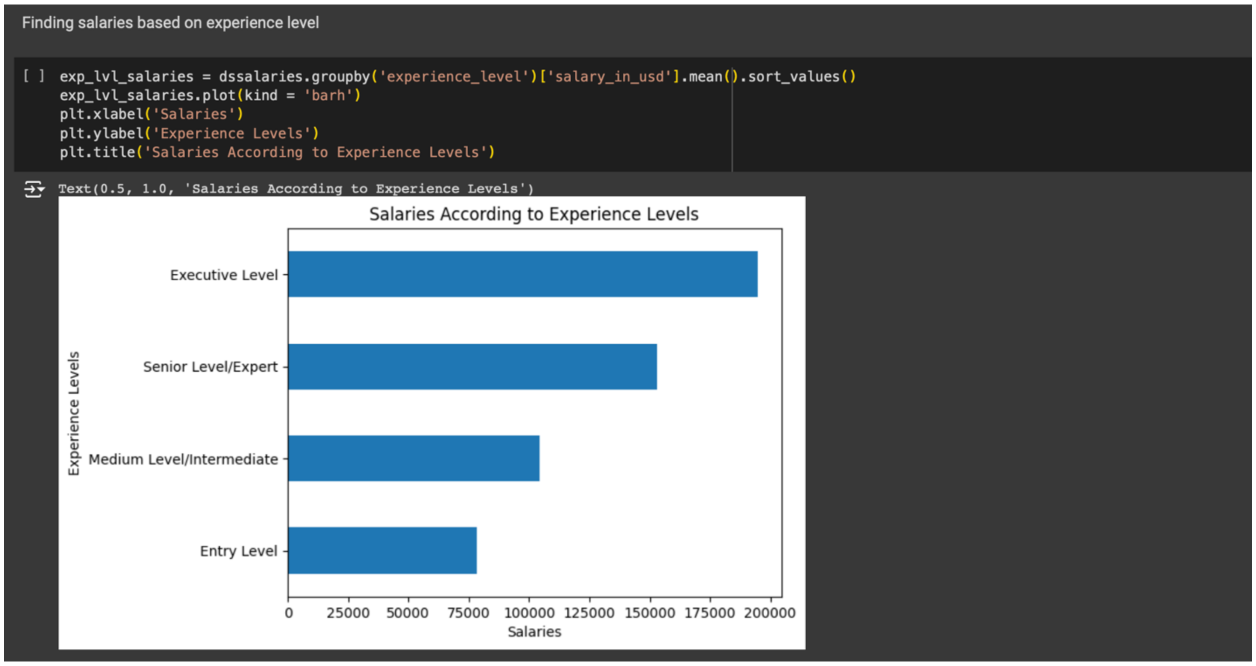
Figure 30 : Plotting a bar graph of the jobs with highest salaries

Here, dssalaries.groupby('job_title') uses the groupby() method to group the DataFrame by the 'job_title' column. After grouping, ['salary_in_usd'] selects the column 'salary_in_usd'. .mean() calculates the mean salary in each job title. After calculating the mean salary for each job title group, .sort_values(ascending=False) sorts the result in descending order of mean salary. Finally, .head(10) selects the top 10 rows from the output, which are the top 10 job titles with the highest average salaries.

Then, highest_salary_jobs.plot(kind='barh') plots the data as a horizontal bar plot. plt.xlabel('Salaries') sets the label for the x-axis as 'Salaries' and plt.ylabel('Job Titles') sets the label for the y-axis as 'Job Titles'. plt.title('Jobs with Highest Salaries') sets the title of the plot as 'Jobs with Highest Salaries'.

Here it is seen that Data Science Tech Lead has the highest salaries, followed by jobs like Cloud Data Architect, Data Lead, etc.

4.3. Write a python program to find out salaries based on experience level. Illustrate it through bar graph.



Finding salaries based on experience level

```
[ ] exp_lvl_salaries = dssalaries.groupby('experience_level')['salary_in_usd'].mean().sort_values()
    exp_lvl_salaries.plot(kind = 'barh')
    plt.xlabel('Salaries')
    plt.ylabel('Experience Levels')
    plt.title('Salaries According to Experience Levels')
```

```
Text(0.5, 1.0, 'Salaries According to Experience Levels')
```

Figure 31 : Plotting a bar graph of salaries based on experience levels

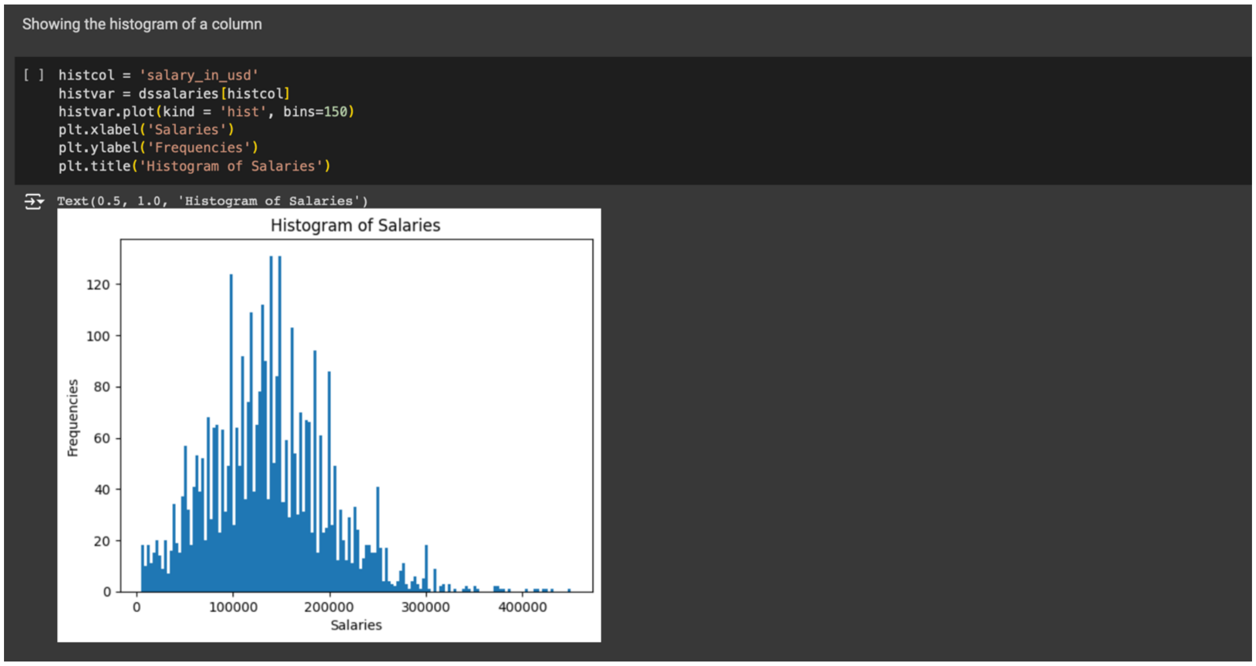
Here, ssalaries.groupby('experience_level') uses the groupby() method to group the DataFrame by the 'experience_level' column. After grouping by experience levels, ['salary_in_usd'] selects the column 'salary_in_usd'. Then, mean() calculates the mean salary within each experience level. Finally, sort_values() sorts the output in ascending order of mean salary.

Then, exp_lvl_salaries.plot(kind='barh') plots the data from as a horizontal bar plot. plt.xlabel('Salaries') sets the label for the x-axis as 'Salaries'., and plt.ylabel('Experience Levels') sets the label for the y-axis as 'Experience Levels'. plt.title('Salaries According to Experience Levels') sets the title of the plot as 'Salaries According to Experience Levels'.

Here, it is seen that executive level has the highest salaries while entry level has the lowest.

4.4. Write a Python program to show histogram and box plot of any chosen different variables. Use proper labels in the graph.

4.4.1. Histogram



Showing the histogram of a column

```
histcol = 'salary_in_usd'
histvar = dssalaries[histcol]
histvar.plot(kind = 'hist', bins=150)
plt.xlabel('Salaries')
plt.ylabel('Frequencies')
plt.title('Histogram of Salaries')
```

```
Text(0.5, 1.0, 'Histogram of Salaries')
```

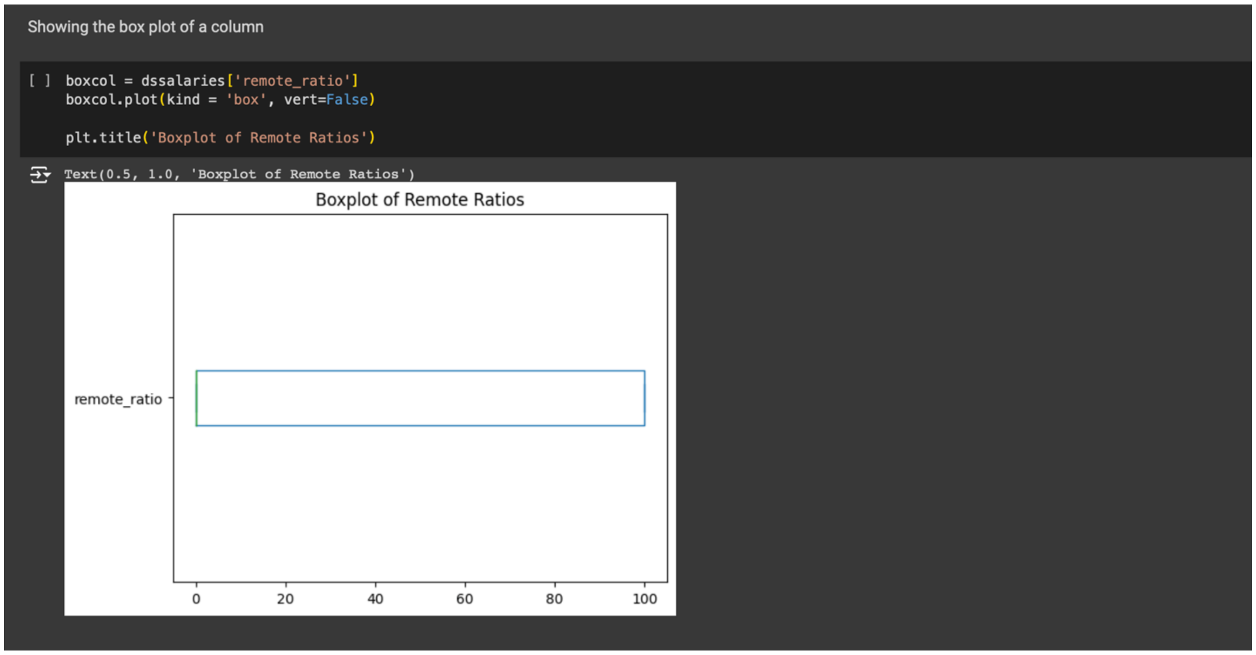
Figure 32 : Plotting a histogram of salaries in USD

Histogram is a graph that displays the distribution of a numeric data. It is similar to a bar graph but is used to visualize the frequencies of continuous data. (GeeksforGeeks, 2024)

This code first selects ‘salary_in_usd’ as the histogram column. Then, plot(kind='hist', bins=150) plots the histogram of the data. The kind='hist' specifies the type of plot as a histogram, while bins=150 specifies the number of bins (the blue lines) into which the data will be divided. plt.xlabel('Salaries') sets the label for the x-axis as 'Salaries'. plt.ylabel('Frequencies') sets the label for the y-axis as 'Frequencies'. plt.title('Histogram of Salaries') sets the title of the plot as 'Histogram of Salaries'.

Here, it is seen that most of the salaries fall in the range of $100,000 to $200,000 and fewer fall in other ranges.

4.4.2. Boxplot


Showing the box plot of a column

```
[ ] boxcol = dssalaries['remote_ratio']
    boxcol.plot(kind = 'box', vert=False)

    plt.title('Boxplot of Remote Ratios')
```

```
Text(0.5, 1.0, 'Boxplot of Remote Ratios')
```

Figure 33 : Plotting a box plot of work years

Box plots are graphical representations of data that use boxes to display mainly 5 different 'numbers' of a distribution: median (the middle line of the box), first quartile (the bottom/left edge of the box), third quartile (the top/right edge of the box), minimum value (the bottom/left whisker), and maximum value (the top/right whisker). (Saul Mcleod, 2023)

This line first selects 'remote_ratio' as the column for the box plot. boxcol.plot(kind='box', vert=False) plots the box plot of the data, where kind='box' specifies the type of plot as a box plot, and vert=False specifies that the box plot is horizontal. plt.title('Boxplot of Remote Ratios') sets the title of the plot as 'Boxplot of Remote Ratios', providing a brief description of the visualized data.

Here, we can see that 0 is the left whiskers well as the first quartile; while 100 is both the right whisker and the third quartile. The median is not defined and there are no outliers as well.

References

GeeksforGeeks, 2024. *Histogram – Definition, Types, Graph, and Examples.* [Online] Available at: https://www.geeksforgeeks.org/histogram/ [Accessed 12 May 2024].

Saul Mcleod, P., 2023. *Box Plot Explained: Interpretation, Examples, & Comparison.* [Online]
Available at: https://www.simplypsychology.org/boxplots.html [Accessed 12 May 2024].

Bh, P., n.d. [Online].

Bhandari, P., 2023. *Correlation Coefficient | Types, Formulas & Examples.* [Online] Available at: https://www.scribbr.com/statistics/correlation-coefficient/ [Accessed 12 May 2024].

Frost, J., 2022. *Kurtosis: Definition, Leptokurtic & Platykurtic.* [Online] Available at: https://statisticsbyjim.com/basics/kurtosis/#comments [Accessed 12 May 2024].