TABLE OF CONTENT

CONTENT PG.NO

1. Abstract…………………………………………………………………2
2. Libraries………………………………………………………………..3-8
3. Data Visualization techniques………………………………………….8-13
4. Data Representation…………………………………………………….14-16
5. Preview…………………………………………………………………16-22
6. Conclusion……………………………………………………………….22

**SALARY ANALYSIS USING EXPLORATIVE DATA ANALYSIS**

Abstraction:

This study presents a comprehensive analysis of employee data to uncover key trends and insights related to workforce dynamics. By examining various attributes such as Work\_year,Experience,Employee\_type,job\_title,salary,currency,employee residenece any many more; the analysis aims to identify patterns that can inform strategic HR decisions. The dataset encompasses a diverse range of employees from multiple departments, providing a holistic view of the organization. Through statistical methods and data visualization techniques, the analysis reveals correlations between employee demographics and performance outcomes, highlights areas for potential improvement in diversity and inclusion, and assesses the impact of tenure on career progression and satisfaction. The findings offer actionable recommendations for enhancing employee engagement, optimizing talent management, and fostering a more inclusive and productive work environment.

Dataset : The dataset has been collected from the company named: DOTCH & ENDEAVOUR

LIBRARIES :

1. Pandas:

Pandas is a powerful and widely-used Python library for data manipulation and analysis. It provides data structures like Series (one-dimensional) and DataFrame (two-dimensional) to handle and analyze data efficiently.

Series: Similar to a list or array but with additional capabilities like indexing.

DataFrame: A table-like structure with labeled axes (rows and columns).

Key features:

Data Loading: Easily load data from various file formats like CSV, Excel, SQL databases, and more.

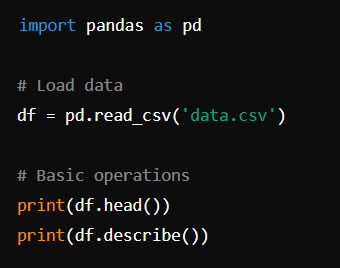
Data Manipulation: Perform operations such as filtering, grouping, merging, and reshaping data.

Data Cleaning: Handle missing values, duplicates, and perform transformations.

Time Series Analysis: Work with time-indexed data, resample, and perform time-based operations.

Integration: Integrates well with other libraries like NumPy, Matplotlib, and more.

Example usage:



2. NumPy

NumPy (Numerical Python) is the foundational package for numerical computing in Python. It provides support for arrays, matrices, and many mathematical functions to operate on these data structures.

Key features:

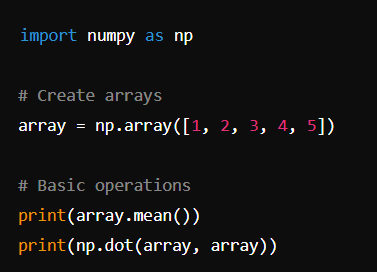
ndarray: A powerful n-dimensional array object.

Mathematical Functions: Extensive library of mathematical functions.

Linear Algebra: Functions for linear algebra, Fourier transforms, and random number generation.

Performance: Efficient operations on large datasets due to its implementation in C.

Example usage:



3. Seaborn:

Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Key features:

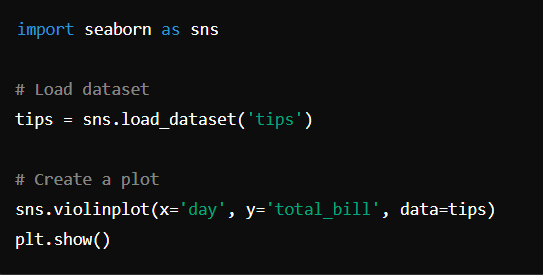
Built-in Themes: Attractive default themes and color palettes.

Statistical Plotting: Functions to create complex visualizations like violin plots, box plots, and more.

Data Handling: Works well with Pandas DataFrames, making it easy to visualize data directly.

Customizability: Easily customize plots to meet specific needs.

Example usage:



4. Plotly Express:

Plotly Express is a high-level interface for Plotly, designed to be easy to use and to create beautiful and interactive visualizations quickly.

Key features:

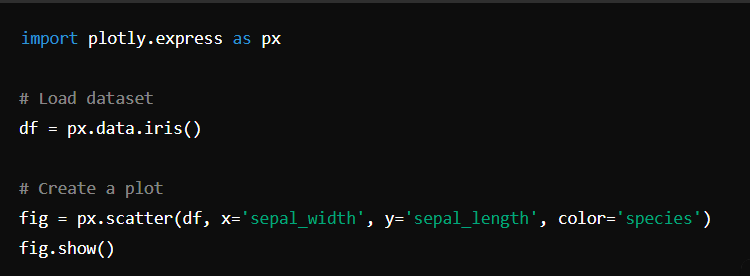
Simplicity: Simple syntax and high-level API to create common visualizations.

Interactivity: Interactive plots with zoom, pan, and hover capabilities.

Integration: Integrates well with Pandas DataFrames.

Variety of Plots: Supports a wide range of plot types including scatter plots, bar charts, line plots, and more.

Example usage:



5. Matplotlib:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Key features:

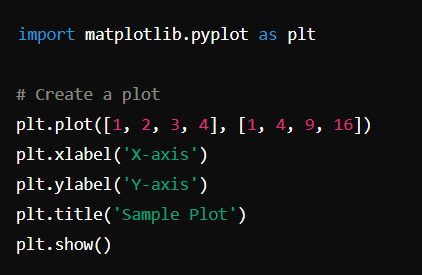
Plot Types: Supports a wide variety of plots such as line plots, scatter plots, bar charts, histograms, etc.

Customization: Highly customizable plots with fine-grained control over elements.

Integration: Works well with NumPy and Pandas for data handling.

Subplots: Create complex multi-plot layouts.

Example Usage:



6. pycountry

pycountry is a library for ISO country, subdivision, language, currency, and script definitions and their translations.

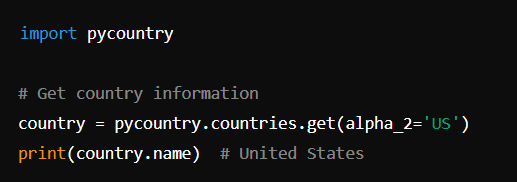
Key features:

ISO Codes: Access ISO codes for countries, subdivisions, languages, etc.

Translations: Provides translations for names in various languages.

Data Handling: Useful for applications requiring standardized country/language data.

Example usage:



1. WordCloud

WordCloud is a library for generating word clouds from text data. Word clouds are visual representations where the size of each word indicates its frequency or importance.

Key features:

Customization: Customize the appearance of the word cloud with different colors, shapes, and sizes.

Stopwords: Automatically removes common stopwords.

Integration: Works with text data from various sources.

These libraries collectively provide a robust toolkit for data analysis, manipulation, and visualization in Python.

Data Visualization Techniques :

1. Bar Graph
2. Cat Plot
3. Histogram
4. Pie Chart
5. Tree Map
6. Scatterplot

1.Bar Graph:

A bar graph (or bar chart) displays data using rectangular bars. Each bar represents a category of data, and the height (or length) of the bar is proportional to the value it represents.

Use: To compare different categories or groups.

figure\_size()

sns.set\_style("darkgrid")

sns.barplot(x='job\_title',y=0,data = jobs[:10],palette = 'magma')

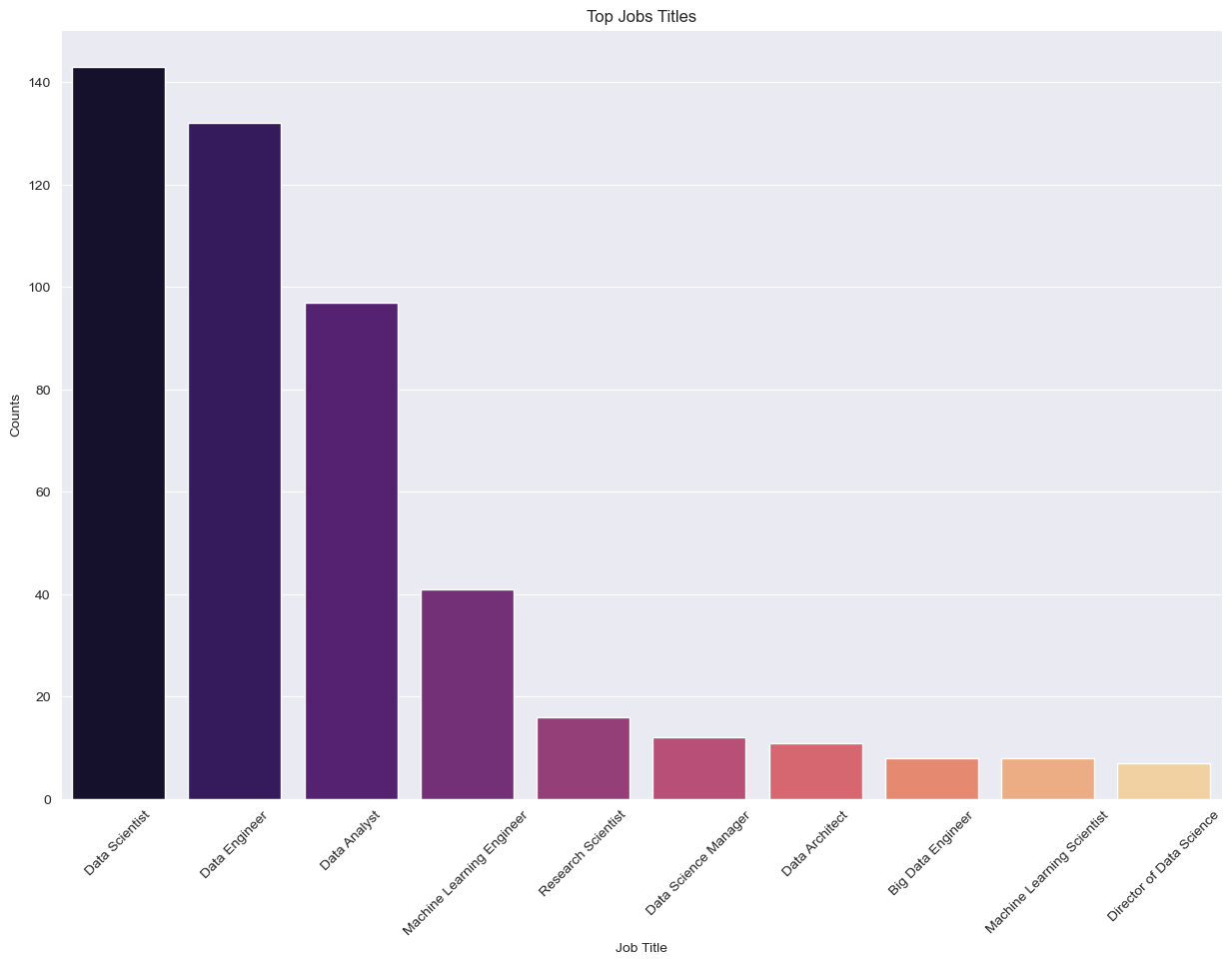
plt.title('Top Jobs Titles')

plt.xlabel('Job Title')

plt.ylabel('Counts')

plt.xticks(rotation=45)

plt.show()



2.Catplot:

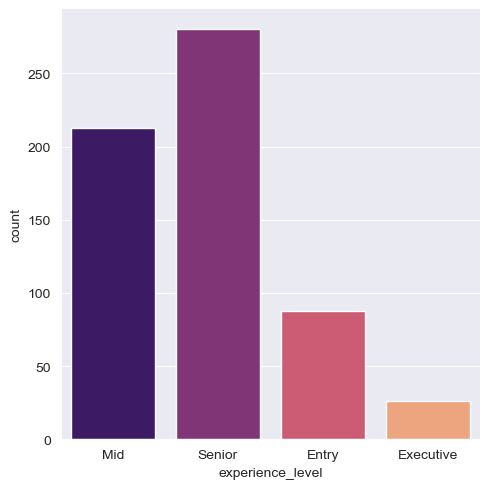
A catplot in Seaborn (a Python data visualization library) is a figure-level function for drawing categorical plots. It provides a unified high-level interface for drawing various types of categorical plots (e.g., bar plots, box plots, violin plots).

Use: To visualize the distribution of categorical data across multiple levels of a variable.

figure\_size()

sns.catplot(x='experience\_level',data = df,kind = 'count',palette = 'magma')

plt.show()

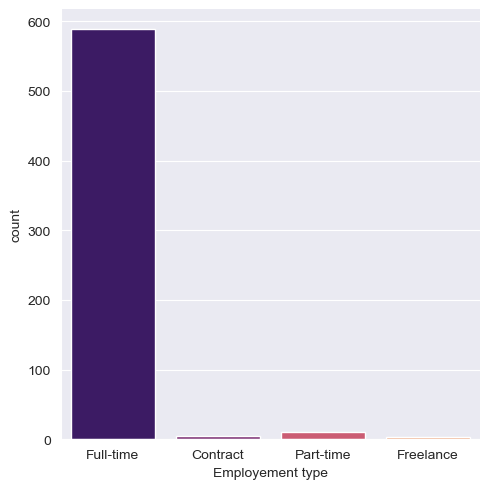


figure\_size()

sns.catplot(x='employment\_type',data = df,kind = 'count',palette = 'magma')

plt.xlabel('Employement type')

plt.show()



3.Histogram:

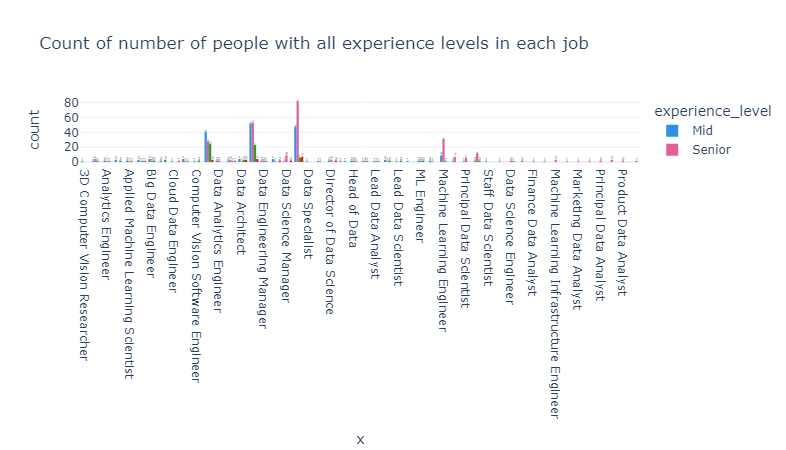
A histogram displays the distribution of a numerical dataset. It divides the data into bins (intervals) and shows the frequency (count) of data points within each bin.

Use: To understand the distribution and frequency of numerical data.

px.histogram(df, x=df.job\_title.sort\_values(), color = 'experience\_level', height = 800, barmode = 'group',

             color\_discrete\_sequence=px.colors.qualitative.Dark24, template = "plotly\_white",

             text\_auto  = True, title = 'Count of number of people with all experience levels in each job')



4.Pie Chart:

A pie chart is a circular graph divided into slices to illustrate numerical proportions. Each slice represents a category, and the size of the slice is proportional to the category's value.

Use: To show the proportion of parts to a whole.

figure\_size()

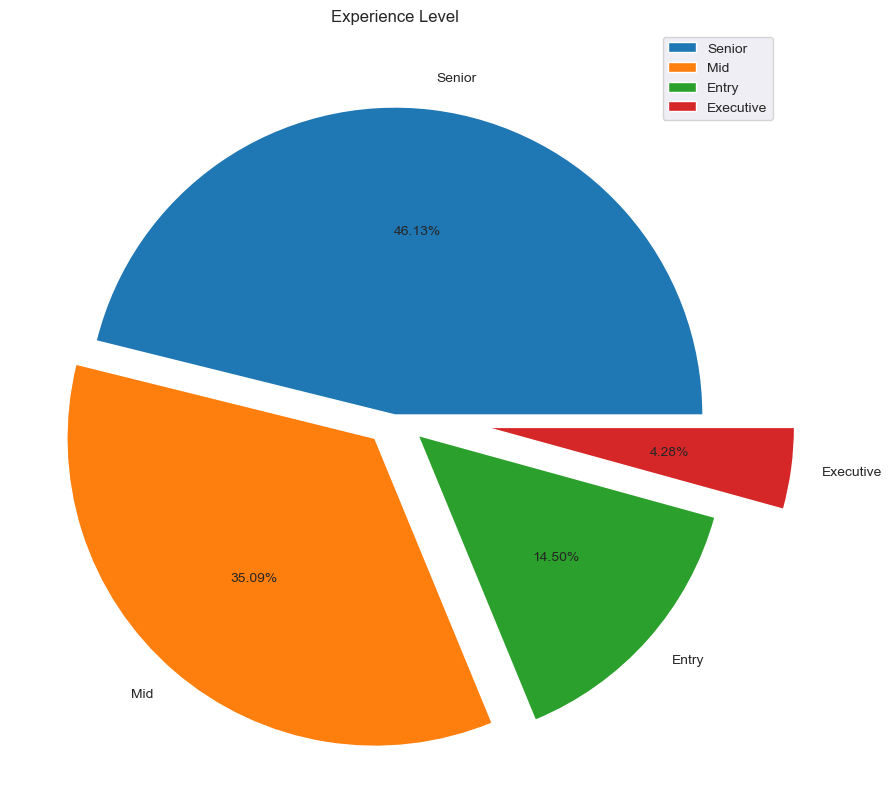
explode = [0,0.1,0.1, 0.3]

plt.pie(x = levels.values,labels =levels.index,autopct='%1.2f%%',explode = explode)

plt.title('Experience Level')

plt.legend()

plt.show()



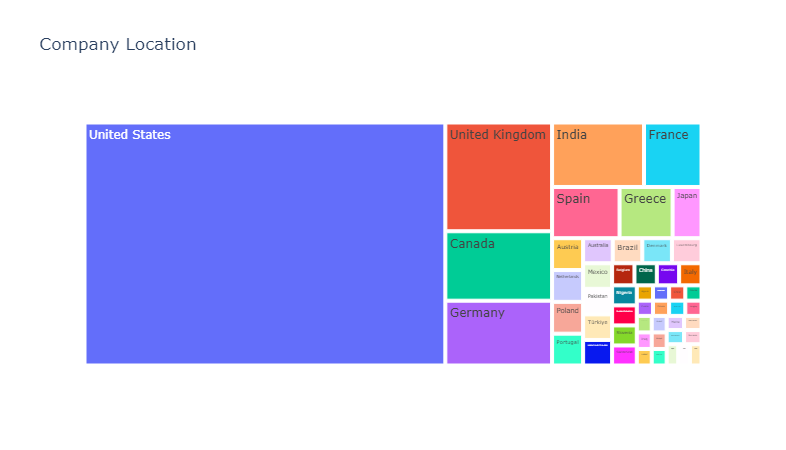
5.Treemap:

A treemap is a visualization that displays hierarchical data using nested rectangles. Each rectangle represents a branch of the hierarchy, and its size is proportional to a specified dimension.

Use: To represent hierarchical data as a proportion of a whole.

ax2= px.treemap(df,path=['company\_location'],title="Company Location")

ax2.show()



6.Scatter Plot:

A scatter plot displays points plotted in a Cartesian plane, where each point represents the values of two variables. It shows the relationship between these two variables.

Use: To identify correlations or patterns between two numerical variables.

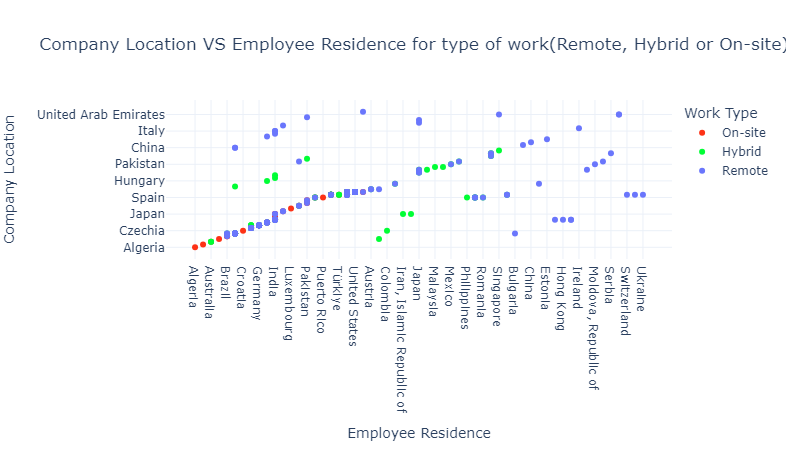
figure\_size()

px.scatter(df, x=df.employee\_residence.sort\_values(), y = df.company\_location.sort\_values(), color = 'remote\_ratio',

           labels ={"x":'Employee Residence', "y":'Company Location', "remote\_ratio":'Work Type'},

           color\_discrete\_sequence=px.colors.qualitative.Light24, template = 'plotly\_white',

           title = 'Company Location VS Employee Residence for type of work(Remote, Hybrid or On-site)')



Data Representation

Data representation in data science refers to the ways in which data is organized, structured, and visualized to facilitate analysis and interpretation. Proper data representation is crucial for extracting meaningful insights and making data-driven decisions. Here are some key aspects of data representation:

**1. Tabular Data**

Description: Data is organized in rows and columns, similar to a spreadsheet.

Use: Suitable for structured data with clear relationships between variables.

Example: A table with columns for customer ID, name, age, and purchase amount.

**2. Time Series Data**

Description: Data points are collected or recorded at specific time intervals.

Use: To analyze trends, patterns, and seasonal variations over time.

Example: Stock prices recorded daily, monthly sales data.

**3. Categorical Data**

Description: Data is divided into categories or groups.

Use: To analyze data that can be divided into discrete groups.

Example: Gender (male, female), product categories (electronics, clothing).

**4. Text Data**

Description: Data in the form of text, often unstructured.

Use: For natural language processing (NLP) tasks such as sentiment analysis, text classification.

Example: Customer reviews, social media posts.

**5. Graph Data**

Description: Data is represented as nodes (entities) and edges (relationships) in a graph.

Use: To model relationships and interactions between entities.

Example: Social networks, transportation networks.

**6. Geospatial Data**

Description: Data that includes geographical information.

Use: To analyze spatial patterns and relationships.

Example: Maps, locations of stores, regions affected by natural disasters.

**7. Image Data**

Description: Data in the form of images or visual representations.

Use: For image processing and computer vision tasks.

Example: Medical imaging, satellite images.

**8. Multidimensional Data**

Description: Data with multiple dimensions or attributes.

Use: For complex data analysis, often represented in data cubes.

Example: Sales data with dimensions like time, region, and product category.

**9. Hierarchical Data**

Description: Data is organized into a tree-like structure with parent-child relationships.

Use: To represent nested relationships.

Example: Organizational charts, file directory structures.

**Importance of Data Representation**

Clarity: Makes complex data easier to understand.

Insight: Helps identify trends, patterns, and anomalies.

Communication: Facilitates sharing findings with stakeholders.

Decision-Making: Supports data-driven decisions by providing clear evidence.

Best Practices

Choose the right representation: Select the most appropriate method based on the data type and the analysis goal.

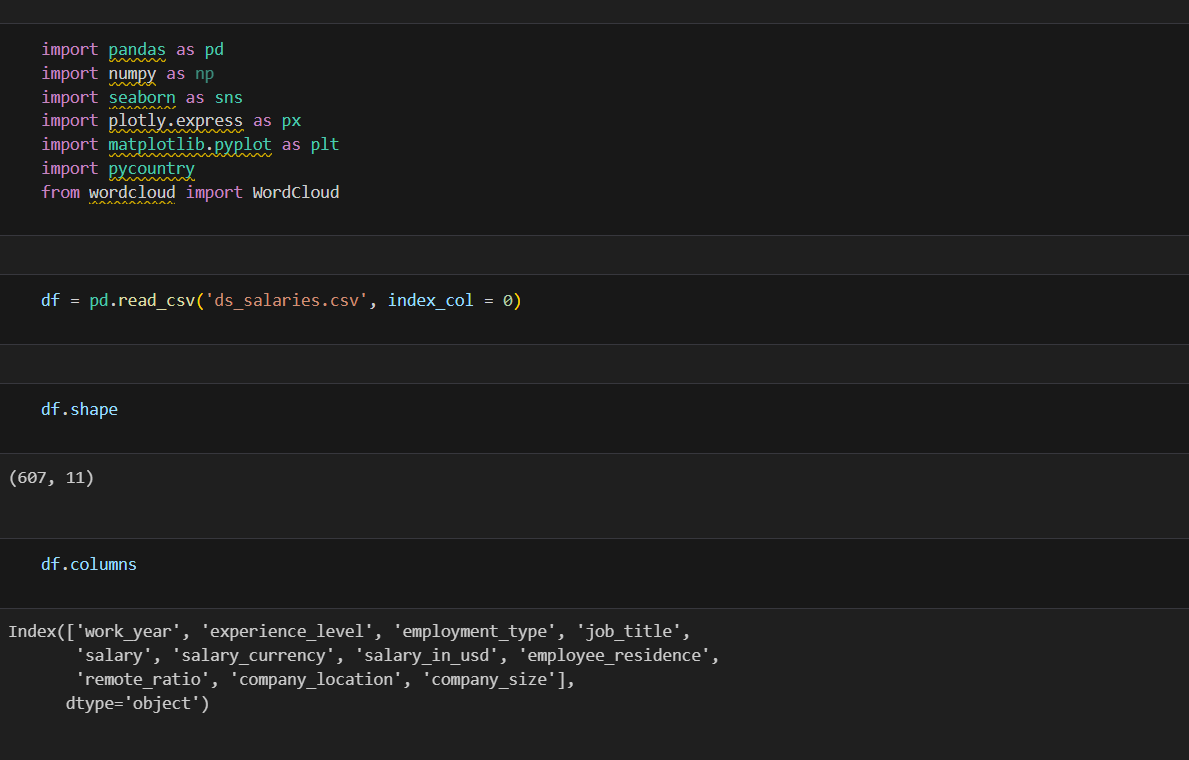
Keep it simple: Avoid overcomplicating visualizations.

Be accurate: Ensure that the representation accurately reflects the data.

Use labels and legends: Make visualizations easy to interpret.

Consider the audience: Tailor the representation to the knowledge level and needs of the audience

Preview:



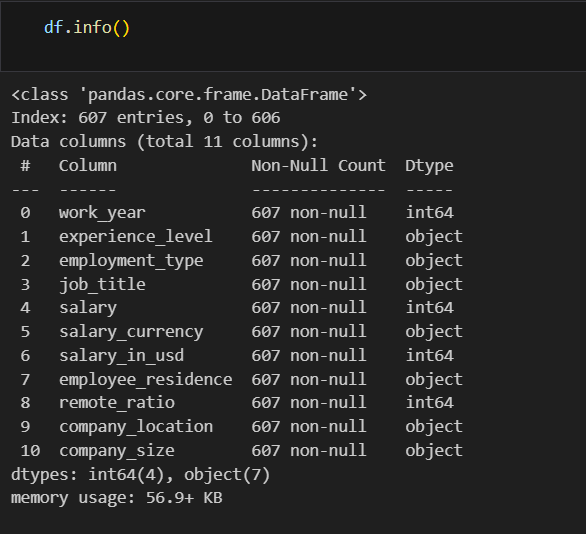
Step 1: We have imported the dependencies

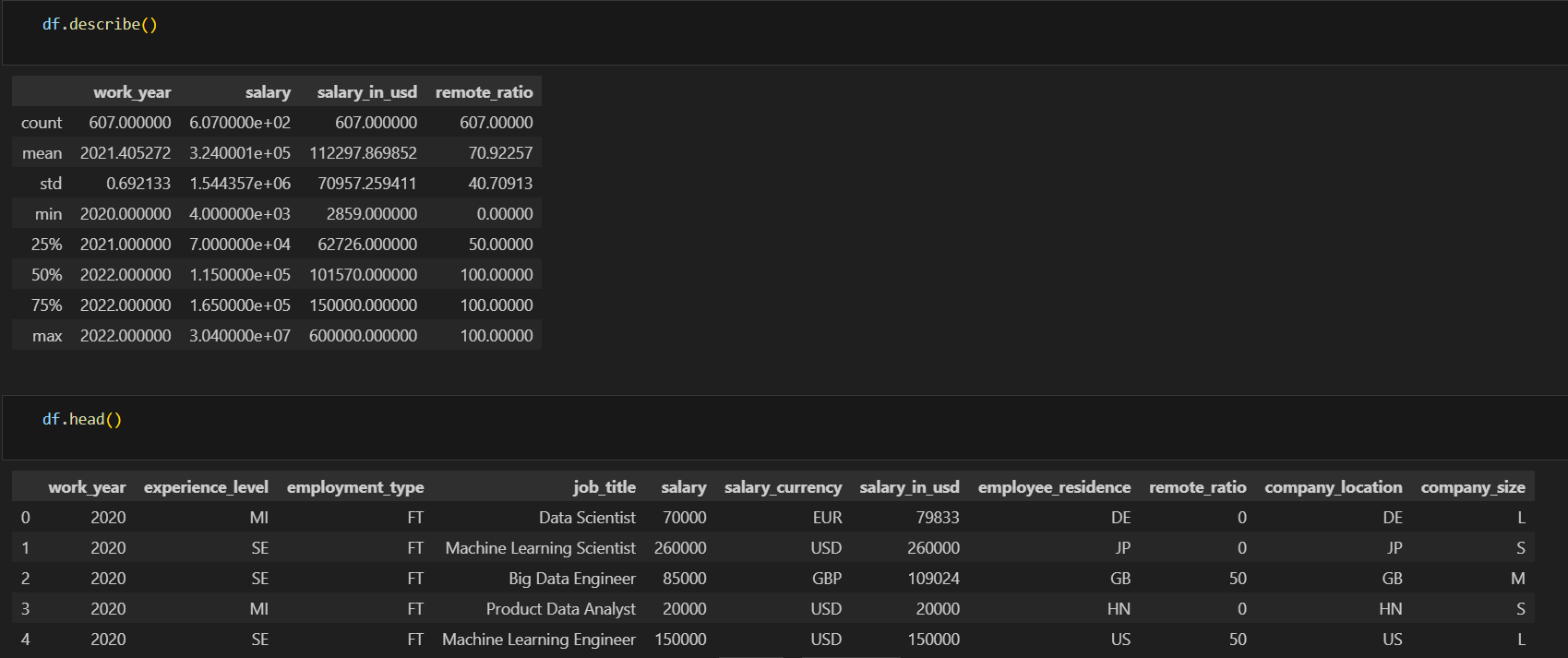
Step 2: The code reads a CSV file named ds\_salaries.csv into a Pandas DataFrame and sets the first column of the CSV file as the index of the DataFrame.

Step 3:

df.shape : Returns a tuple representing the dimensions of the DataFrame. The first element is the number of rows, and the second element is the number of columns. This helps understand the size of the dataset.

df.columns : Returns an Index object containing the column labels of the DataFrame. This provides the names of the columns in the dataset.





**1. df.describe()**

* **Purpose**: Provides summary statistics of the numerical columns in the DataFrame..
* **Use Case**: Helpful for getting a quick overview of the central tendency, dispersion, and shape of the dataset’s distribution.

**2. df.info()**

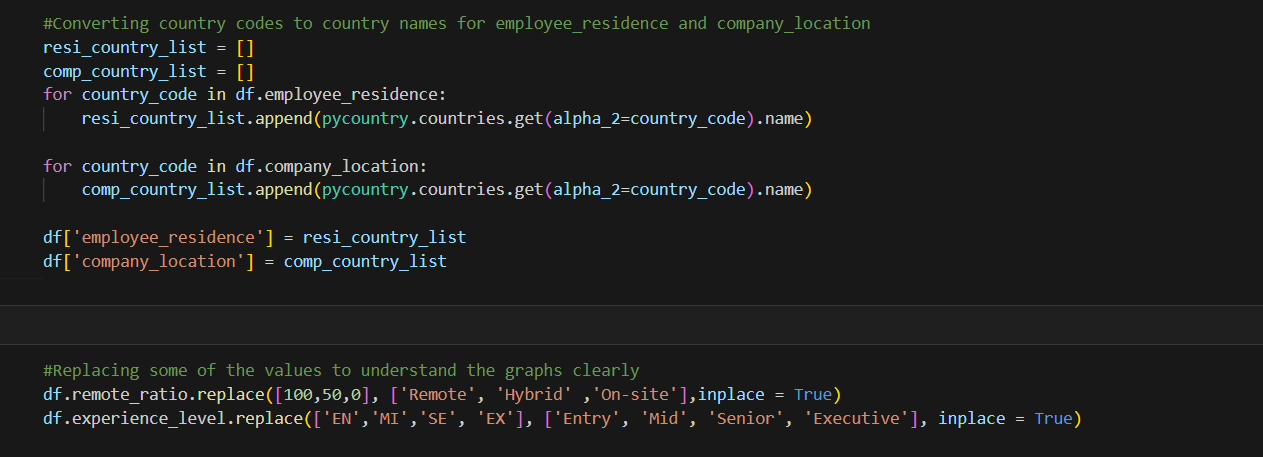
* **Purpose**: Provides a concise summary of the DataFrame.

### Use Case: Useful for understanding the data structure, identifying columns with missing values, and checking data types.

### 3. df.head()

* **Purpose**: Displays the first few rows of the DataFrame.
* **Use Case**: Helpful for quickly inspecting the beginning of the DataFrame to understand the data format and content.

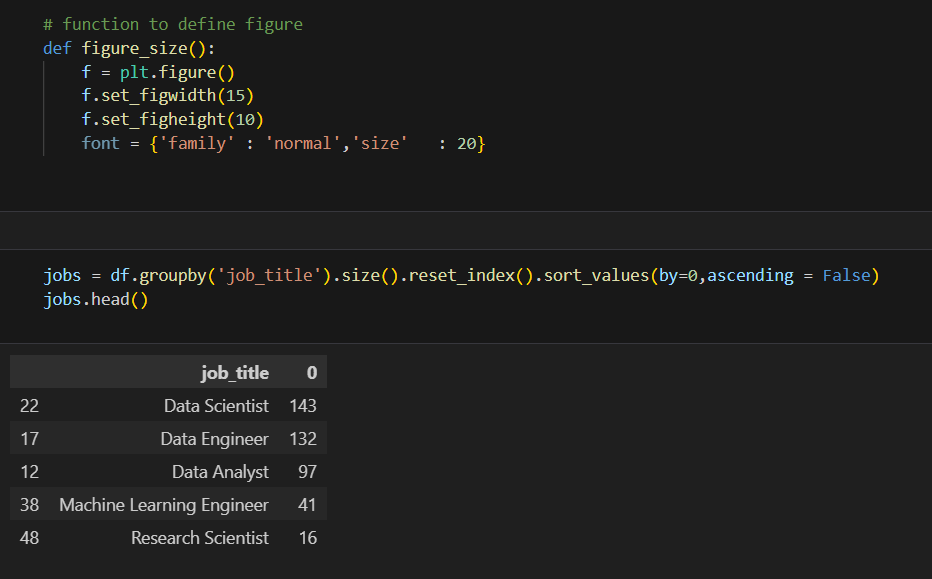
Step 4:



This code snippet converts two-letter country codes into full country names for the ‘employee\_residence’ and ‘company\_location’ columns in a ‘DataFrame df.’ It first initializes two empty lists, resi\_country\_list and ‘comp\_country\_list’, and then iterates over each country code in the employee\_residence and company\_location columns, appending the corresponding country name to the respective list using the pycountry library. Finally, it replaces the original columns in the DataFrame with the new lists of full country names.

The next cell containing the code replaces numerical values in the `remote\_ratio` column and abbreviated strings in the `experience\_level` column of the DataFrame `df` with more descriptive labels for better understanding in graphs.

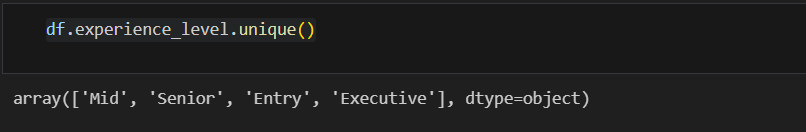
Step 5:



This function `figure\_size()` defines a Matplotlib figure with a specified width and height, and sets the font properties to have a normal family with a font size of 20.

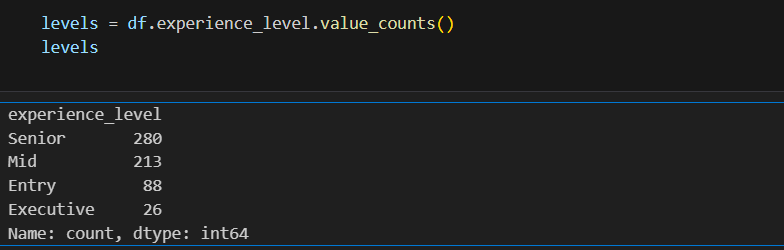
The Next cell containing the code groups the DataFrame `df` by the `job\_title` column, counts the number of occurrences for each job title, resets the index, sorts the results in descending order based on the count, and then displays the first few rows of the sorted DataFrame with `jobs.head()`.

Step 6:



For the Bar graph the code `df.experience\_level.unique()` returns an array of unique values present in the `experience\_level` column of the DataFrame `df`, providing a list of all distinct experience levels in the dataset.

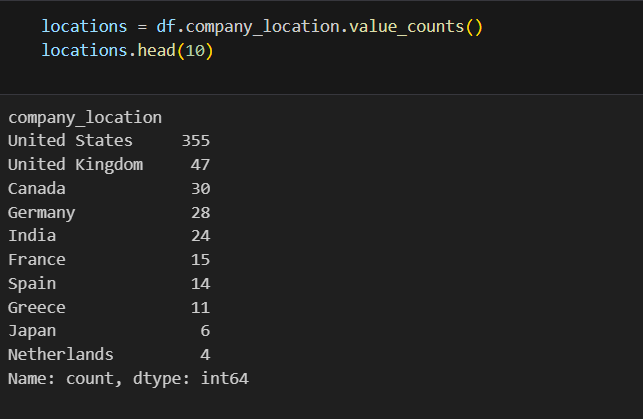
Step 7:

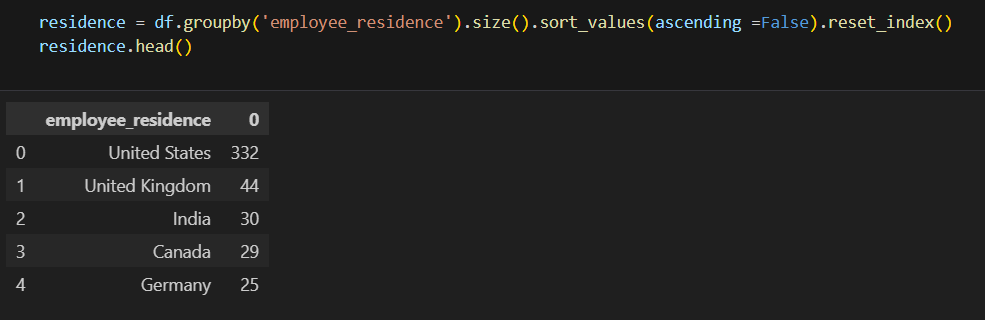


The code `df.experience\_level.unique()` returns an array of unique values present in the `experience\_level` column of the DataFrame `df`, providing a list of all distinct experience levels in the dataset.



Step 8:





This code groups the DataFrame `df` by the `employee\_residence` column, counts the number of occurrences for each unique residence, sorts these counts in descending order, resets the index of the resulting DataFrame, and then displays the first few rows with `residence.head()`.

CONCLUSION:

Our exploratory data analysis of the data science salary dataset reveals several key insights about the current state of the field. Firstly, we observed a diverse range of job titles, with roles such as Data Scientist, Data Analyst, and Machine Learning Engineer being among the most common. The distribution of salaries across these roles indicates that higher experience levels generally correspond to higher salaries, with executives earning the most and entry-level positions earning the least.

Geographical analysis showed significant variations in salaries based on both employee residence and company location, highlighting the impact of regional economic conditions and cost of living. Notably, remote and hybrid work arrangements have become prevalent, offering flexibility but also influencing salary scales.

Additionally, the analysis of remote work ratios revealed that fully remote roles are quite common in the data science field, reflecting a broader trend towards remote work in the tech industry. This shift could have long-term implications for salary structures and job accessibility across different regions.

In summary, the data science field offers lucrative opportunities, particularly for those with higher experience levels and specialized skills. The trends in remote work and geographic salary variations underscore the importance of considering both location and work arrangements when evaluating job opportunities. These insights can help both job seekers and employers make more informed decisions in the evolving landscape of data science careers.