|  |
| --- |
| **W4 – Exploratory Data Analysis** |

Save your W4 notebook with the following naming conventions.

ID\_Name\_SecNo\_W4.ipynb,

for example

**6113333\_JohnWick\_541\_W4.ipynb**

**What is Exploration Data Analysis (EDA)**

Exploratory Data Analysis, or EDA for short, is the process of cleaning and reviewing data to derive insights such as descriptive statistics and correlation and generate hypotheses for experiments. EDA results often inform the next steps for the dataset, whether that be generating hypotheses, preparing the data for use in a machine learning model, or even throwing the data out and gathering new data!

**A close-up of a book shelf

Description automatically generated with low confidence**

**Remember from the last worksheet (W3), we can explore the data using various Pandas methods or attributes such as .head(), .info(), .describe(), .shape, etc.**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**A picture containing text, screenshot, font, number

Description automatically generated**

**Apart from exploring initial statistics summary, it is also effective to visualize the data.**

**Try the following codes.**

|  |  |
| --- | --- |
| **import pandas as pd**  **import matplotlib.pyplot as plt**  **import seaborn as sns**  **%matplotlib inline**  **books = pd.read\_csv('clean\_books.csv')**  **sns.histplot(data=books, x='rating')**  **plt.show()**  **#You may add binwdith to histplot to change percentage point.** |  |

**For .value\_counts(), remember we can select a specific column to get a closer look at the information. Try.**

**books['genre'].value\_counts()**

**A new version of Pandas also supports**

**books.value\_counts('genre')**

1. **Load clean\_umployment.csv to unemployment and complete the following tasks.**

* Import the seaborn visualization libraries.
* Create a histogram of the distribution of 2021 unemployment percentages across all countries in unemployment; show a full percentage (binwidth) point in each bin.

**Data Validation**

Data validation is an important early step in EDA. We want to understand whether data types and ranges are as expected before we progress too far in our analysis!

A screenshot of a computer

Description automatically generated with medium confidence

**Updating data types**

The .astype() function allows us to change data types without too much effort. Here, we can redefine the year column by selecting the column and calling the .astype() method, indicating we'd like to change the column to an integer. Then we use the .dtypes() attribute to check that the year column data is now stored as integers.

A screenshot of a computer code

Description automatically generated with low confidenceA screenshot of a computer

Description automatically generated with low confidence

**Validating Categorial Data**

We can validate categorical data by comparing values in a column to a list of expected values using .isin(), which can either be applied to a Series as we'll show here or to an entire DataFrame. For example, if the values in the genre column are limited to "Fiction" and "Non Fiction" by passing these genres as a list of strings to .isin(). The function returns a Series of the same size and shape as the original but with True and False in place of all values, depending on whether the value from the original Series was included in the list passed to .isin(). We can see that some values are False.

A screenshot of a computer

Description automatically generated with medium confidence

Note we can use ~ to express it is not in … For example,

~books['genre'].isin(['Fiction','Non Ffiction'])

A screenshot of a computer

Description automatically generated with medium confidence

**Working with Numerical Data.**

We can select and view only the numerical columns in a DataFrame by calling the select\_dtypes method and passing "number" as the argument.

A picture containing text, screenshot, software, font

Description automatically generated

And we can view a more detailed picture of the distribution of year data using Seaborn's boxplot function. The boxplot shows the boundaries of each quartile of year data: as we saw using min and max, the lowest year is 2009 and the highest year is 2019. The 25th and 75th percentiles are 2010 and 2016 and the median year is 2013.

A picture containing text, screenshot, rectangle, diagram

Description automatically generated

It is possible to view the year data grouped by a categorical variable such as genre by setting the y keyword argument. It looks like the children's books in our dataset have slightly later publishing years in general, but the range of years is the same for all genres.

A picture containing screenshot, text, diagram, rectangle

Description automatically generated

# Let’s Practice.

# Validating continents

Your colleague has informed you that the data on unemployment from countries in Oceania is not reliable, and you'd like to identify and exclude these countries from your unemployment data. The .isin() function can help with that!

Your task is to use .isin() to identify countries that are *not* in Oceania. These countries should return True while countries in Oceania should return False. This will set you up to use the results of .isin() to quickly filter out Oceania countries using Boolean indexing.

Load clearn\_unemployment.csv to unemployment

1. **Define a Series of Booleans describing whether or not each continent is outside of Oceania; call this Series not\_oceania.**

**Use Boolean indexing to print the unemployment DataFrame without any of the data related to countries in Oceania.**

1. **Print the minimum and maximum unemployment rates, in that order, during 2021.**

**Create a boxplot of 2021 unemployment rates, broken down by continent.**

We can explore the characteristics of subsets of data further with the help of the .groupby() function, which groups data by a given category, allowing the user to chain an aggregating function like .mean() or .count() to describe the data within each group.

A screenshot of a computer

Description automatically generated with medium confidence

The .agg() function, short for aggregate, allows us to apply aggregating functions. By default, it aggregates data across all rows in a given column and is typically used when we want to apply more than one function.

**A screenshot of a computer

Description automatically generated with low confidence**

We can even use a dictionary to specify which aggregation functions to apply to which columns. The keys in the dictionary are the columns to apply the aggregation, and each value is a list of the specific aggregating functions to apply to that column.

**A screenshot of a computer code

Description automatically generated with low confidence**

# Named summary columns.

By combining .agg() and .groupby(), we can apply these new exploration skills to grouped data. Maybe we'd like to show the mean and standard deviation of rating for each book genre along with the median year. We can create named columns with our desired aggregations by using the .agg() function and creating named tuples inside it.

**A screenshot of a computer code

Description automatically generated with medium confidence**

# Visualizing categorical summaries

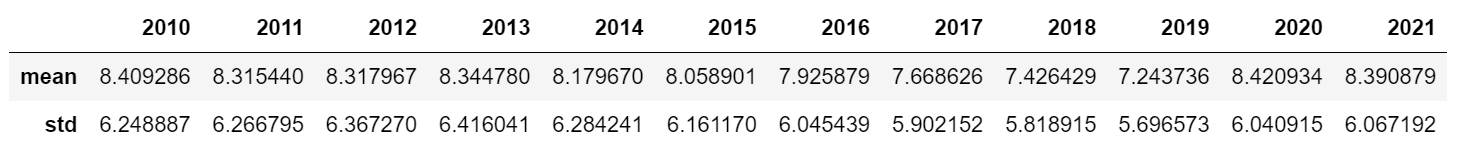
We can display similar information visually using a barplot. In Seaborn, bar plots will automatically calculate the mean of a quantitative variable like rating across grouped categorical data, such as the genre category we've been looking at. In Seaborn, bar plots also show a 95% confidence interval for the mean as a vertical line on the top of each bar.

**A picture containing text, screenshot, rectangle, colorfulness

Description automatically generated**

**Let’s Practice.**

1. **Print the mean and standard deviation of the unemployment rates for each year. (using .agg() as shown in the example above)**

****

**Print the mean and standard deviation of the unemployment rates for each year (not all data is shown below).**

**A picture containing text, screenshot, font, document

Description automatically generated**

# Named aggregations.

You've seen how .groupby() and .agg() can be combined to show summaries across categories. Sometimes, it's helpful to name new columns when aggregating so that it's clear in the code output what aggregations are being applied and where.

Your task is to create a DataFrame called continent\_summary which shows a row for each continent. The DataFrame columns will contain the mean unemployment rate for each continent in 2021 as well as the standard deviation of the 2021 employment rate. And of course, you'll rename the columns so that their contents are clear!

From the given code –

continent\_summary = unemployment.groupby("continent").agg()

1. **Create a column called mean\_rate\_2021 which shows the mean 2021 unemployment rate for each continent.**

**Create a column called std\_rate\_2021 which shows the standard deviation of the 2021 unemployment rate for each continent.**

# Visualizing categorical summaries

As you've learned in this chapter, Seaborn has many great visualizations for exploration, including a bar plot for displaying an aggregated average value by category of data.

In Seaborn, bar plots include a vertical bar indicating the 95% confidence interval for the categorical mean. Since confidence intervals are calculated using both the number of values and the variability of those values, they give a helpful indication of how much data can be relied upon.

Your task is to create a bar plot to visualize the means and confidence intervals of unemployment rates across the different continents.

1. **Create a bar plot showing continents on the x-axis and their respective average 2021 unemployment rates on the y-axis.**

A picture containing text, screenshot, diagram, line

Description automatically generated

So, why is it important to deal with missing data? Well, it can affect distributions. As an example, we collect the heights of students at a high school. If we fail to collect the heights of the oldest students, who were taller than most of our sample, then our sample mean will be lower than the population mean. Put another way, our data is less representative of the underlying population. In this case, parts of our population aren't proportionately represented. This misrepresentation can lead us to draw incorrect conclusions, like thinking that, on average, students are shorter than they really are.

A picture containing text, screenshot, font, number

Description automatically generated

To highlight the impact of missing values, let's look at salaries by experience level using a full version of the dataset. Now, let's compare this to the same data with some missing values. The y-axis shows that the largest salary is around 150000 dollars less in the second plot!

A picture containing diagram, text, screenshot, plan

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

A white background with black text

Description automatically generated with low confidence

To calculate our missing values threshold we multiply the length of our DataFrame by five percent, giving us an upper limit of 30 as shown in the example below.

A picture containing text, screenshot, font

Description automatically generated

We can use Boolean indexing to filter for columns with missing values less than or equal to this threshold, storing them as a variable called cols\_to\_drop. Printing cols\_to\_drop shows four columns. We drop missing values by calling .dropna(), passing cols\_to\_drop to the subset argument. We set inplace to True so the DataFrame is updated.

A screenshot of a computer code

Description automatically generated with low confidence

We then filter for the remaining columns with missing values, giving us four columns. To impute the mode for the first three columns, we loop through them and call the .fillna() method, passing the respective column's mode and indexing the first item, which contains the mode, in square brackets.

A screenshot of a computer program

Description automatically generated with low confidence

Checking for missing values again, we see salary\_USD is now the only column with missing values and the volume has changed from 60 missing values to 41. This is because some rows may have contained missing values for our subset columns as well as salary, so they were dropped.

A picture containing text, screenshot, software, font

Description automatically generated

We'll impute median salary by experience level by grouping salaries by experience and calculating the median. We use the .to\_dict() method, storing the grouped data as a dictionary. Printing the dictionary returns the median salary for each experience level, with executives earning the big bucks!

A picture containing text, screenshot, font, line

Description automatically generated

We then impute using the .fillna() method, providing the Experience column and calling the .map() method, inside which we pass the salaries dictionary.



A screenshot of a computer

Description automatically generated with low confidence

Let’s Practice. Load airline\_unclean.csv to planes with index\_col = 0 argument

1. **Print the number of missing values in each column of the DataFrame.**

* **Calculate how many observations five percent of the planes DataFrame is equal to.**
* **Create cols\_to\_drop by applying boolean indexing to columns of the DataFrame with missing values less than or equal to the threshold.**
* **Use this filter to remove missing values and save the updated DataFrame.** A picture containing text, font, screenshot, receipt

  Description automatically generated A picture containing text, font, screenshot, receipt

  Description automatically generated

# Strategies for remaining missing data.

So far, the five percent rule has worked nicely for the planes dataset, eliminating missing values from nine out of 11 columns!

Now, you need to decide what to do with the "Additional\_Info" and "Price" columns, which are missing 300 and 368 values respectively.

You'll first take a look at what "Additional\_Info" contains, then visualize the price of plane tickets by different airlines.

Run the following code to create boxplot for Price by Airline, and to observe mean value.

# Check the values of the Additional\_Info column

print(planes["Additional\_Info"].value\_counts())

# Create a box plot of Price by Airline

sns.boxplot(data=planes, x='Airline', y='Price')

sns.set(rc={"figure.figsize":(8, 6)}) #width=8, #height=6

plt.show()

A picture containing diagram, screenshot, text

Description automatically generated

## Question

How should you deal with the missing values in "Additional\_Info" and "Price"?

1. Remove the “Additional\_Info” column and impute the mean for missing values of “Price”
2. Remove “No info” values from “Additional\_Info” and impute the median for missing values of “Price”
3. Remove the “Additional\_Info” column and impute the mean by “Airline” for missing values of “Price”
4. Remove the “Additional\_Info” column and impute the median by “Airline” for missing values of “Price”

Let’s drop the “Additional\_Info” column, with **planes = planes.drop(columns = ['Additional\_Info'])**

1. **Group planes by airline and calculate the median price.**

**Convert the grouped median prices to a dictionary.**

**Conditionally impute missing values for "Price" by mapping values in the "Airline column" based on prices\_dict.**

**Check for remaining missing values.**

**Working with Categorial Data.**

Now let's explore how to create and analyze categorical data. Recall that we can use the select\_dtypes method to filter any non-numeric data. Chaining .head() allows us to preview these columns in our salaries DataFrame, showing columns such as Designation, Experience, Employment\_Status, and Company\_Size. You can follow along by loading data from ds\_salaries\_clean.csv to salaries DataFrame.

A screenshot of a computer program

Description automatically generated

Let's examine frequency of values in the Designation column. The output is truncated by pandas automatically since there are so many different job titles!

A screenshot of a computer program

Description automatically generated

We can count how many unique job titles there are using pandas .nunique() method. There are 50 in total!

A close-up of a box

Description automatically generated

If we plot the graph using a bar chart, the fifth most popular job title, Research Scientist, appears less than 20 times.

A graph of different colored squares

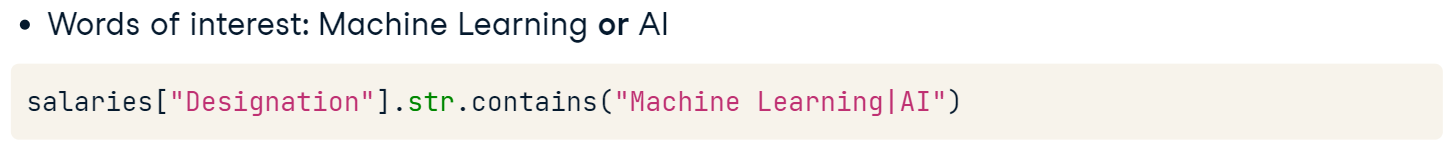
Description automatically generated

The current format of the data limits our ability to generate insights. We can use the pandas.Series.string.contains() method, which allows us to search a column for a specific string or multiple strings. Say we want to know which job titles have Scientist in them. We use the str.contains() method on the Designation column, passing the word Scientist. This returns True or False values depending on whether the row contains this word.

A screenshot of a computer

Description automatically generated

What if we want to filter for rows containing one or more phrases?

****

**A close-up of a white background

Description automatically generated**

Now we have a sense of how this method works, let's define a list of job titles we want to find. We start by creating a list with the different categories of data roles, which will become the values of a new column in our DataFrame.

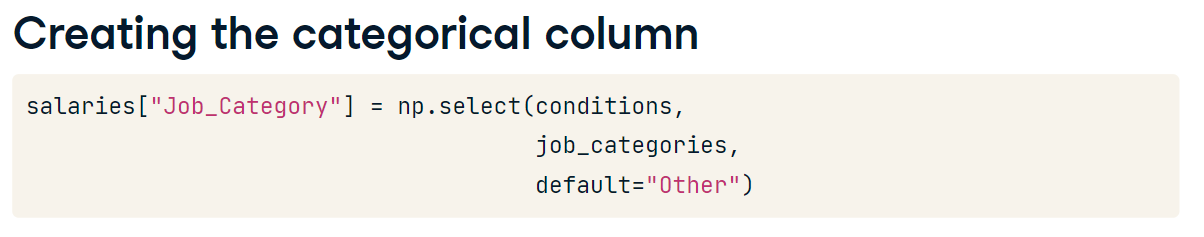
We then need to create variables containing our filters. We will look for Data Scientist or NLP for data science roles. We'll use Analyst or Analytics for data analyst roles. We repeat this for data engineer, machine learning engineer, managerial, and consultant roles.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated



A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

**Try (We select columns with object data type)**

# Filter the DataFrame for object columns

non\_numeric = planes.select\_dtypes("object")

# Loop through columns

for col in non\_numeric.columns:

  # Print the number of unique values

  print(f"Number of unique values in {col} column: ", non\_numeric[col].nunique())

**A number of unique values in the "Duration" column of planes is shown.**

**Calling planes["Duration"].head(), we see the following values:**

0 19h

1 5h 25m

2 4h 45m

3 2h 25m

4 15h 30m

Name: Duration, dtype: object

Looks like this won't be simple to convert to numbers. However, you could categorize flights by duration and examine the frequency of different flight lengths!

You'll create a "Duration\_Category" column in the planes DataFrame. Before you can do this you'll need to create a list of the values you would like to insert into the DataFrame, followed by the existing values that these should be created from.

1. **Create short\_flights, a string to capture values of "0h", "1h", "2h", "3h", or "4h".**

**Create medium\_flights to capture any values between five and nine hours.**

**Create long\_flights to capture any values from 10 hours to 16 hours inclusive.**

# Adding duration categories

Now that you've set up the categories and values you want to capture, it's time to build a new column to analyze the frequency of flights by duration!

The variables flight\_categories, short\_flights, medium\_flights, and long\_flights that you previously created are available to you.

Additionally, the following packages have been imported: pandas as pd, numpy as np, seaborn as sns, and matplotlib.pyplot as plt.

1. **Create conditions, a list containing subsets of planes["Duration"] based on short\_flights, medium\_flights, and long\_flights.**

**Create the "Duration\_Category" column by calling a function that accepts your conditions list and flight\_categories, setting values not found to "Extreme duration".**

**Create a plot showing the count of each category.**

**Working with Numerical Data.**

Time to switch our focus on to working with numeric data.

A screenshot of a computer program

Description automatically generated

To obtain Salary in USD we'll need to perform a few tasks. First, we need to remove the commas from the values in the Salary\_In\_Rupees column. Next, we change the data type to float. Lastly, we'll make a new column by converting the currency.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A diagram of data flow

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

Complete the following tasks.

1. **Print the first five values of the "Duration" column.**

**Remove "h",  "m", " " from the column and make the data format to be hh.mm**

**Convert the column to float data type.**

**Plot a histogram of "Duration" values using**

sns.histplot(data=planes, x="Duration")

# Adding descriptive statistics

Now "Duration" and "Price" both contain numeric values in planes DataFrame, you would like to calculate summary statistics for them that are conditional on values in other columns.

1. **Add a column to planes containing the standard deviation of "Price" based on "Airline".**

**Calculate the median for "Duration" by "Airline", storing it as a column called "airline\_median\_duration".**

**Find the mean "Price" by "Destination", saving it as a column called "price\_destination\_mean".**

# Handling Outliers

# A pool with a building and a building with a pool Description automatically generatedA screenshot of a computer Description automatically generated

We can define an outlier mathematically. First, we need to know the interquartile range, or IQR, which is the difference between the 75th and 25th percentiles.

# A close-up of a logo Description automatically generated

Recall that these percentiles are included in box plots, like this one showing salaries of data professionals.

# A diagram with text and numbers Description automatically generated

Once we have the IQR, we can find an upper outlier by looking for values above the sum of the 75th percentile plus 1.5 times the IQR. Lower outliers have values below the sum of the 25th percentile minus 1.5 times the IQR.

# A white background with black text Description automatically generated

# A screenshot of a computer program Description automatically generated

# A screenshot of a computer code Description automatically generated

# A screenshot of a computer Description automatically generated

# A screenshot of a computer Description automatically generated

# A white background with black text Description automatically generated

# A white background with black text Description automatically generated

# A screenshot of a computer code Description automatically generated

# Let’s Practice.

1. **Plot the distribution of "Price" column from planes using sns.histplot()**

**Display the descriptive statistics for flight duration.**

# Removing outliers

While removing outliers isn't always the way to go, for your analysis, you've decided that you will only include flights where the "Price" is not an outlier.

Therefore, you need to find the upper threshold and then use it to remove values above this from the planes DataFrame.

1. **Find the 75th and 25th percentiles, saving as price\_seventy\_fifth and price\_twenty\_fifth respectively.**

**Calculate the IQR, storing it as prices\_iqr.**

**Calculate the upper and lower outlier thresholds.**

**Remove the outliers from planes.**