Day 71 advanced data with pandas

Day 71 Goals: what you will make by the end of the day

**Learn Data Exploration with Pandas by Analysing the Post-University Salaries of Graduates by Major**

**Introducing the Google Colab Notebook**

PyCharm is a fantastic IDE, but when we're exploring and visualising a dataset, you'll find the Python notebook format better suited.

Open your first Google Colab Notebook in through your [Google Drive](https://drive.google.com/). You can find the Python Notebook under New → More → Google Colaboratory

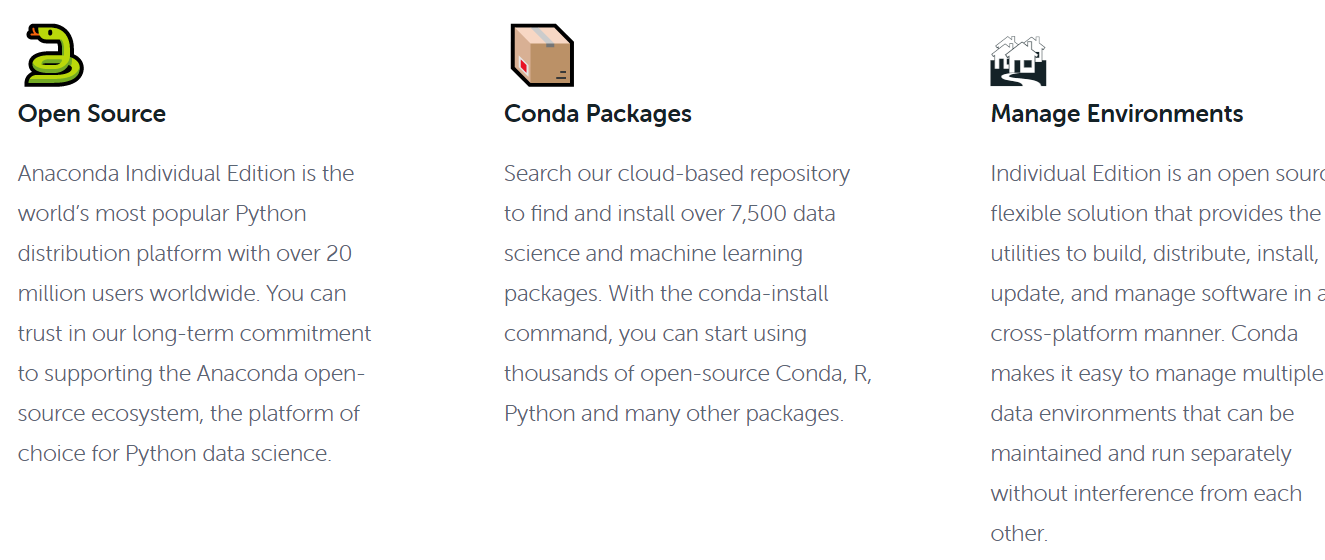
If you cannot access the Google Colab Notebooks or would like to run everything locally on your computer, then I recommend installing Anaconda and using the bundled Jypyter Notebook instead. Either way works. Google Colab is essentially just an online version of Jupyter.

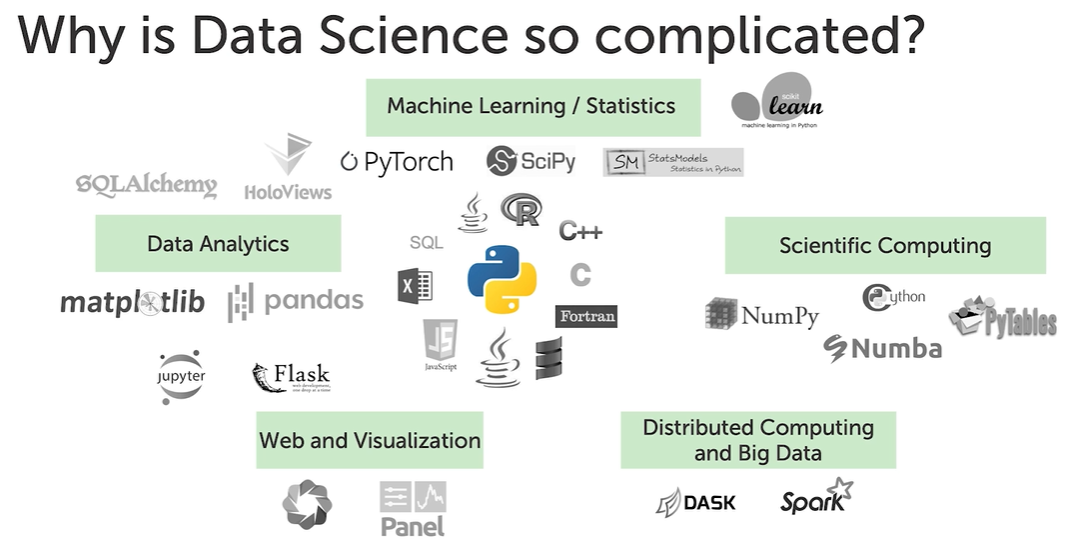
<https://www.anaconda.com/products/individual>

Individual Edition

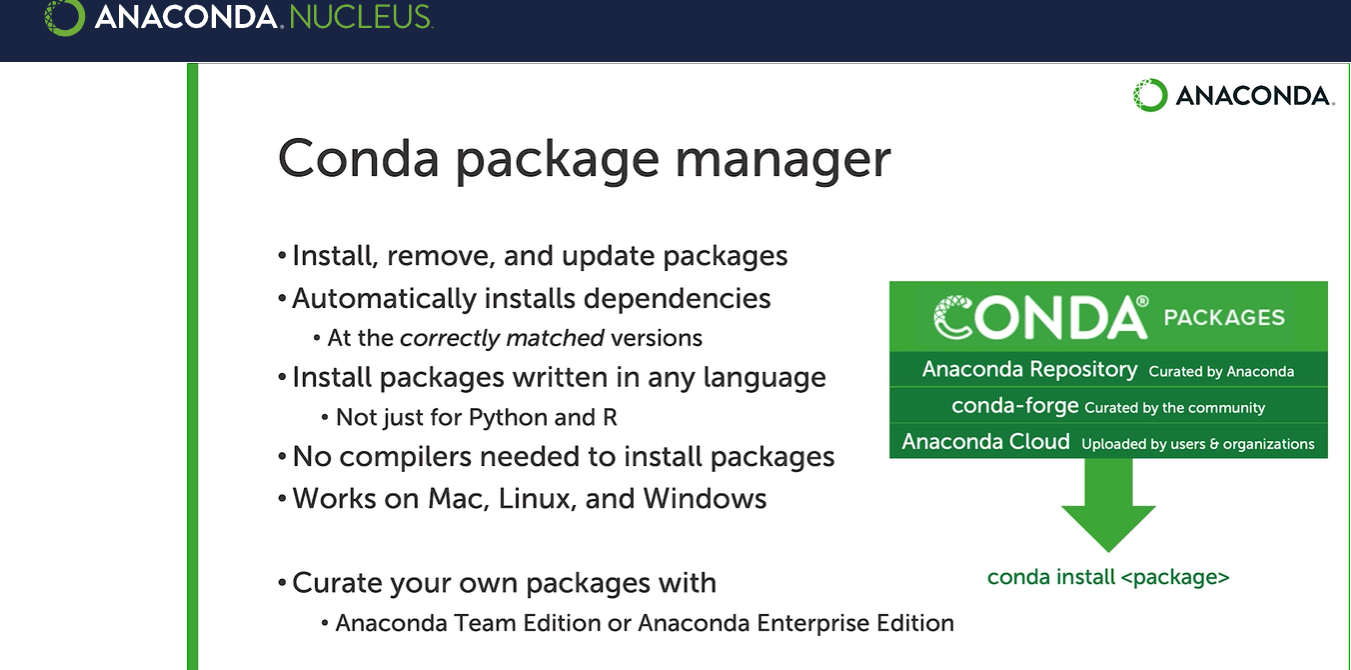
Your data science toolkit

With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.





<https://www.anaconda.com/products/individual/installation-success?source=win_installer>



Preliminary Data Exploration and Data Cleaning with Pandas

Now that we've got our data loaded into our dataframe, we need to take a closer look at it to help us understand what it is we are working with. This is always the first step with any data science project. Let's see if we can answer the following questions:

* How many rows does our dataframe have?
* How many columns does it have?
* What are the labels for the columns? Do the columns have names?
* Are there any missing values in our dataframe? Does our dataframe contain any bad data?

We've already used the .head() method to peek at the top 5 rows of our dataframe. To see the number of rows and columns we can use the shape attribute:

1. df.shape

Do you see 51 rows and 6 columns printed out below the cell?

We saw that each column had a name. We can access the column names directly with the columns attribute.

Missing Values and Junk Data

Before we can proceed with our analysis we should try and figure out if there are any missing or junk data in our dataframe. That way we can avoid problems later on. In this case, we're going to look for NaN (Not A Number) values in our dataframe. NAN values are blank cells or cells that contain strings instead of numbers. Use the .isna() method and see if you can spot if there's a problem somewhere.

1. df.isna()

Did you find anything? Check the last couple of rows in the dataframe:

1. df.tail()

Aha! We have a row that contains some information regarding the source of the data with blank values for all the other other columns.

**Delete the Last Row**

We don't want this row in our dataframe. There's two ways you can go about removing this row. The first way is to manually remove the row at index 50. The second way is to simply use the .dropna() method from pandas. Let's create a new dataframe without the last row and examine the last 5 rows to make sure we removed the last row:

1. clean\_df = df.dropna()
2. clean\_df.tail()

Accessing Columns and Individual Cells in a Dataframe

**Find College Major with Highest Starting Salaries**

To access a particular column from a data frame we can use the square bracket notation, like so:

1. clean\_df['Starting Median Salary']

You should see all the values printed out below the cell for just this column:

To find the highest starting salary we can simply chain the .max() method.

1. clean\_df['Starting Median Salary'].max()

The highest starting salary is $74,300. But which college major earns this much on average? For this, we need to know the row number or **index** so that we can look up the name of the major. Lucky for us, the .idxmax() method will give us index for the row with the largest value.

1. clean\_df['Starting Median Salary'].idxmax()

which is 43. To see the name of the major that corresponds to that particular row, we can use the .loc (location) property.

1. clean\_df['Undergraduate Major'].loc[43]

Here we are selecting both a column ('Undergraduate Major') and a row at index 43, so we are retrieving the value of a particular cell. You might see people using the double square brackets notation to achieve exactly the same thing:

1. clean\_df['Undergraduate Major'][43]

If you don't specify a particular column you can use the .loc property to retrieve an entire row:

1. clean\_df.loc[43]

**Challenge**

Now that we've found the major with the highest starting salary, can you write the code to find the following:

* What college major has the highest mid-career salary? How much do graduates with this major earn? (Mid-career is defined as having 10+ years of experience).
* Which college major has the lowest starting salary and how much do graduates earn after university?
* Which college major has the lowest mid-career salary and how much can people expect to earn with this degree?

I'll provide the solution and the code snippets in the next lesson =)

Solution: Highest and Lowest Earning Degrees

I hope you gave the last challenge a good go before checking the solution below.

**The Highest Mid-Career Salary**

1. print(clean\_df['Mid-Career Median Salary'].max())
2. print(f"Index for the max mid career salary: {clean\_df['Mid-Career Median Salary'].idxmax()}")
3. clean\_df['Undergraduate Major'][8]

If you have multiple lines in the same cell, only the last one will get printed as an output automatically. If you'd like to see more than one thing printed out, then you still have to use a print statement on the lines above.

**The Lowest Starting and Mid-Career Salary**

1. print(clean\_df['Starting Median Salary'].min())
2. clean\_df['Undergraduate Major'].loc[clean\_df['Starting Median Salary'].idxmin()]

Here I've nested the code that we've seen in the previous lesson in the same line. We can also use the .loc property to access an entire row. Below I've accessed the row at the index of the smallest mid-career salary:

1. clean\_df.loc[clean\_df['Mid-Career Median Salary'].idxmin()]

Sadly, education is actually the degree with the lowest mid-career salary and Spanish is the major with the lowest starting salary.

Sorting Values & Adding Columns: Majors with the Most Potential vs Lowest Risk

**Lowest Risk Majors**

A low-risk major is a degree where there is a small difference between the lowest and highest salaries. In other words, if the difference between the 10th percentile and the 90th percentile earnings of your major is small, then you can be more certain about your salary after you graduate.

How would we calculate the difference between the earnings of the 10th and 90th percentile? Well, Pandas allows us to do simple arithmetic with entire columns, so all we need to do is take the difference between the two columns:

1. clean\_df['Mid-Career 90th Percentile Salary'] - clean\_df['Mid-Career 10th Percentile Salary']

Alternatively, you can also use the .subtract() method.

1. clean\_df['Mid-Career 90th Percentile Salary'].subtract(clean\_df['Mid-Career 10th Percentile Salary'])

The output of this computation will be another Pandas dataframe column. We can add this to our existing dataframe with the .insert() method:

1. spread\_col = clean\_df['Mid-Career 90th Percentile Salary'] - clean\_df['Mid-Career 10th Percentile Salary']
2. clean\_df.insert(1, 'Spread', spread\_col)
3. clean\_df.head()

The first argument is the position of where the column should be inserted. In our case, it's at position 1, so the second column.

**Sorting by the Lowest Spread**

To see which degrees have the smallest spread, we can use the .sort\_values() method. And since we are interested in only seeing the name of the degree and the major, we can pass a list of these two column names to look at the .head() of these two columns exclusively.

1. low\_risk = clean\_df.sort\_values('Spread')
2. low\_risk[['Undergraduate Major', 'Spread']].head()

Does .sort\_values() sort in ascending or descending order? To find out, check out the Pandas documentation: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_values.html>

You can also bring up the quick documentation with shift + tab on your keyboard directly in the Python notebook.

#### Challenge

Using the .sort\_values() method, can you find the degrees with the highest potential? Find the top 5 degrees with the highest values in the 90th percentile.

Also, find the degrees with the greatest spread in salaries. Which majors have the largest difference between high and low earners after graduation.

I've got the solution for you in the next lesson.

Here's the solution to the challenge from the previous lesson:

**Majors with the Highest Potential**

1. highest\_potential = clean\_df.sort\_values('Mid-Career 90th Percentile Salary', ascending=False)
2. highest\_potential[['Undergraduate Major', 'Mid-Career 90th Percentile Salary']].head()

**Majors with the Greatest Spread in Salaries**

1. highest\_spread = clean\_df.sort\_values('Spread', ascending=False)
2. highest\_spread[['Undergraduate Major', 'Spread']].head()

Often times you will want to sum rows that below to a particular category. For example, which category of degrees has the highest average salary? Is it STEM, Business or HASS (Humanities, Arts, and Social Science)?

To answer this question we need to learn to use the .groupby() method. This allows us to manipulate data similar to a Microsoft Excel Pivot Table.

We have three categories in the 'Group' column: STEM, HASS and Business. Let's count how many majors we have in each category:

1. clean\_df.groupby('Group').count()

**Mini Challenge**

Now can you use the .mean() method to find the average salary by group?

Here's the solution:

**Number formats in the Output**

The above is a little hard to read, isn't it? We can tell Pandas to print the numbers in our notebook to look like 1,012.45 with the following line:

1. pd.options.display.float\_format = '{:,.2f}'.format

Ah, that's better, isn't it?

**Extra Credit:**

The PayScale dataset used in this lesson was from 2008 and looked at the prior 10 years. Notice how Finance ranked very high on post-degree earnings at the time. However, we all know there was a massive financial crash in that year. Perhaps things have changed. Can you use what you've learnt about web scraping in the prior lessons (e.g., Day 45) and share some updated information from [PayScale's website](https://www.payscale.com/college-salary-report/majors-that-pay-you-back/bachelors) in the comments below?

<https://www.payscale.com/college-salary-report/majors-that-pay-you-back/bachelors>

Learning Points & Summary

**Today's Learning Points**

* Use .head(), .tail(), .shape and .columns to explore your DataFrame and find out the number of rows and columns as well as the column names.
* Look for NaN (not a number) values with .findna() and consider using .dropna() to clean up your DataFrame.
* You can access entire columns of a DataFrame using the square bracket notation: df['column name'] or df[['column name 1', 'column name 2', 'column name 3']]
* You can access individual cells in a DataFrame by chaining square brackets df['column name'][index] or using df['column name'].loc[index]
* The largest and smallest values, as well as their positions, can be found with methods like .max(), .min(), .idxmax() and .idxmin()
* You can sort the DataFrame with .sort\_values() and add new columns with .insert()
* To create an Excel Style Pivot Table by grouping entries that belong to a particular category use the .groupby() method