Using Jupyter Notebooks with Pandas and Matplotlib

Why is this important?

Visualization is a key part of data analysis. While Pandas provides some basic plotting capabilities, Matplotlib is a powerful library for creating a wide variety of static, animated, and interactive visualizations in Python. Combining Jupyter Notebooks with Pandas and Matplotlib allows for interactive data exploration and visualization, making it easier to understand and communicate insights from data.

Matplotlib and Pandas some of the most popular libraries for data manipulation and visualization in Python, making them essential tools for anyone working with data in Python.

Note: If you're taking software development at NAIT you're probably going to cover this in depth in future courses. This is just a brief introduction to get you started.

What are we going to do?

We're going to install Jupyter, Pandas, and Matplotlib in a virtual environment, and create a simple notebook that uses Pandas to analyze some data and Matplotlib to visualize it.

We're going to build on the ideas of the previous example and add some visualizations using Matplotlib.

Steps

1. Create a virtual environment and install jupyter and pandas

1. create the virtual environment, and activate it:

```
python -m venv ./venv
.\venv\Scripts\activate # Or source ./venv/bin/activate on macOS/Linux
```

2. install jupyter and pandas:

```
pip install jupyter pandas matplotlib
```

3. Save the dependencies to a requirements file:

```
pip freeze > requirements.txt
```

2. Create a new Jupyter notebook just like last class.

1. Start the Jupyter Notebook server if it's not already running:

```
jupyter notebook
```

2. In the Jupyter interface, click on "New" and select "Python 3" to create a new notebook.
3. Rename your notebook to `matplotlib_example.ipynb`

3. Import the libraries and load the data

In the first cell of your new notebook, import the necessary libraries and load the data:

```
import pandas as pd
import matplotlib.pyplot as plt
# Load the data
data =
pd.read_csv('data/Property_Assessment_Data_(Current_Calendar_Year)_2025103
0.csv')
# take a look at the first few rows
data.head()
```

4. Let's get filter to get the top 6 most expensive residential properties in the city.

1. let's filter the data A reminder of how to filter data in pandas:

```
redietial_data = data[data["Tax Class"] == "Residential"]
```

This will filter the data to only include rows where the "Tax Class" column is "Residential".

2. Let's sort the data by "Assessed Value" in descending order and get the top 6 most expensive properties:

```
top_6_expensive = redietial_data.sort_values(by="Assessed Value",
ascending=False)[:6]

top_6_expensive.head(10) # shows the first 10 rows, but there should only
be 6
```

Note: the `[:6]` at the end of the line selects the top 6 rows after sorting.

5. Now let's create a bar chart of the assessed values of these top 6 properties.

1. First, let's set up the data for the bar chart: We're going to create a new column in the `top_6_expensive` DataFrame that combines the "House Number" and "Street Name" to create a full address for each property. Then, we'll use this new column as the x-axis labels for our bar chart, and the "Assessed Value" as the heights of the bars.

```
# create a new column for full address
# since 'House Number' is an integer we need to convert it to a string
first
# this is what the .astype(str) part is doing
top_6_expensive['Property Address'] = top_6_expensive['House
Number'].astype(str) + ' ' + top_6_expensive['Street Name']

# set up data for the bar chart
property_names = top_6_expensive['Property Address']
assessed_values = top_6_expensive['Assessed Value']
```

2. Now, let's create the bar chart using Matplotlib:

```
plt.figure(figsize=(10, 6))
plt.bar(property_names, assessed_values, color='skyblue')
plt.xlabel('Property Address')
plt.ylabel('Assessed Value')
plt.title('Top 6 Most Expensive Residential Properties')
plt.xticks(rotation=45, ha='right')  # Rotate x labels for better
readability
plt.show()
```

- Here `plt` is the common alias for `matplotlib.pyplot`, which provides a MATLAB-like interface for creating plots, this is a very common convention in the Python data science community.
- The `plt.bar()` function creates a bar chart.
  - the first argument is the x-axis labels (property names),
  - the second argument is the heights of the bars (assessed values),
  - the `color` argument sets the color of the bars.
- The `plt.xlabel()`, `plt.ylabel()`, and `plt.title()` functions set the labels and title of the chart.
- The `plt.xticks()` function is used to rotate the x-axis labels for better readability.
- Finally, `plt.show()` displays the plot.

6. let's create a line chart of this data.

A line chart can be useful to see trends over a sequence. In this case, we can plot the assessed values of the top 6 properties in order.

```
plt.figure(figsize=(10, 6))
plt.plot(property_names, assessed_values, marker='o', linestyle='-',
color='orange')
plt.xlabel('Property Address')
plt.ylabel('Assessed Value')
plt.title('Top 6 Most Expensive Residential Properties - Line Chart')
plt.xticks(rotation=45, ha='right')  # Rotate x labels for better
readability
```

```
plt.grid(True)
plt.show()
```

Let's break down the code:

- The `plt.plot()` function creates a line chart.
  - the first argument is the x-axis labels (property names),
  - the second argument is the y-axis values (assessed values),
  - the `marker` argument specifies the style of the markers at each data point,
  - the `linestyle` argument specifies the style of the line connecting the points,
  - the `color` argument sets the color of the line and markers.
- the `plt.grid(True)` function adds a grid to the chart for better readability.
- The rest of the code is similar to the bar chart example, setting labels, title, rotating x-axis labels, and displaying the plot.

Note: Again if you're in software development at NAIT you'll probably cover this in depth in future courses. This is just a brief introduction to get you started.

7. (Optional) Let's do some more analysis with pandas and matplotlib and get the average assessed value by neighbourhood and plot it.

This is an optional step if you want to explore more with pandas and matplotlib.

Let's group the data by "Neighbourhood" and calculate the average assessed value for each neighbourhood. Then, we'll plot a bar chart of the average assessed values by neighbourhood.

```
# Group by Neighbourhood and calculate average assessed value
avg_assessed_by_neighbourhood = redietial_data.groupby('Neighbourhood')
['Assessed Value'].mean().sort_values(ascending=False)[:10]
```

This is doing a few things things here let's talk about it.

- `redietial_data.groupby('Neighbourhood')` groups the data by the "Neighbourhood" column.
- `['Assessed Value'].mean()` calculates the mean (average) of the "Assessed Value" for each neighbourhood.
- `.sort_values(ascending=False)[:10]` sorts the average assessed values in descending order and selects the top 10 neighbourhoods with the highest average assessed values.

Now, let's plot the bar chart:

```
plt.figure(figsize=(10, 6))
plt.bar(avg_assessed_by_neighbourhood.index,
avg_assessed_by_neighbourhood.values, color='lightgreen')
plt.xlabel('Neighbourhood')
plt.ylabel('Average Assessed Value')
plt.title('Top 10 Neighbourhoods by Average Assessed Value')
```

```
plt.xticks(rotation=45, ha='right')  # Rotate x labels for better
readability
plt.show()
```

- Here, `avg_assessed_by_neighbourhood.index` provides the neighbourhood names for the x-axis labels.
- `avg_assessed_by_neighbourhood.values` provides the average assessed values for the heights of the bars.
- The rest of the code is similar to the previous bar chart example, setting labels, title, rotating x-axis labels, and displaying the plot.

Exercises

- Create a bar chart of the top 5 "Assessed Value" properties in the "Commercial" tax class.
- Create a bar chart of the assessed values of properties in a specific neighbourhood (e.g., YOUR NEIGHBOURHOOD HERE).
- Create a line chart showing the trend of average assessed values across all neighbourhoods.

Summary

Visualization is an essential part of data analysis, and Matplotlib is a powerful library for creating a wide variety of visualizations in Python. By combining Jupyter Notebooks with Pandas and Matplotlib, you can interactively explore and visualize data, making it easier to understand and communicate insights.

Appendix: handy jupyter shortcuts

Command	Description
`Shift + Enter`	Run the current cell and move to the next cell
`Ctrl + Enter`	Run the current cell and stay in it
`Alt + Enter`	Run the current cell and insert a new one below
`Esc`	Enter command mode (blue border)
`Enter`	Enter edit mode (green border)
`A` (in command mode)	Insert a new cell **above**
`B` (in command mode)	Insert a new cell **below**
`D, D` (press D twice)	Delete the selected cell
`Z`	Undo the last cell deletion
`M`	Change cell to **Markdown**
`Y`	Change cell to **Code**
`L`	Toggle line numbers in the current cell

Command	Description
`Shift + M`	Merge selected cells
`Ctrl + S`	Save the notebook
`0, 0` (press 0 twice)	Restart the kernel
`I, I` (press I twice)	Interrupt the kernel
`Ctrl + /`	Toggle comment on selected lines (in edit mode)
`Ctrl + Shift + -`	Split a cell at the cursor
`Tab`	Autocomplete or show function signature
`Shift + Tab`	Show tooltip/documentation for an object
`!command`	Run a shell command (e.g. `!ls`, `!pip install`)
`%time`	Measure execution time of a single line of code
`%%time`	Measure execution time of a whole cell
`%who`	List all variables in the namespace
`%whos`	Detailed list of variables with types and sizes
`%pwd`, `%cd`, `%ls`	File system navigation commands
`%matplotlib inline`	Display plots inline (common for matplotlib)
`?object` or `help(object)`	Show documentation for an object