Lab 06

Advanced Computing for Policy

```
from ydata_profiling import ProfileReport
import pandas as pd
```

Lab Overview

- Finishing Lab 5: Profiling and data quality checks
- Linting and formatting
- Continuous integration

Task:

- Set up continuous integration to run tests and linting on your code.
- You'll work in your Project teams.

Finishing Lab 5

Profiling

```
data = pd.read_csv('../lab_04/videos_data.csv')
data['Likes_numeric'] = data['Likes'].str.replace(',', '').astype(int)
profile = ProfileReport(data, title="Pandas Profiling Report")
profile.to_widgets()
```

- Some findings:
  - Variables: Likes is a string. Most liked video has 44M likes. Least poular has 433 likes (?)
  - Interactions tab: Most top 200 videos were published after 2017.
  - Missing values: Almost half of the videos are missing the 'Dislikes' column.
- Did you find anything surprising/interesting/useful?

Finishing Lab 5

Data quality checks

- Unit tests for data
- Example 1: Checking variables' types

```
def check_numeric(data, column):
    assert data[column].dtype in ['int64', 'float64'], f"{column} is not numeric"

cols = ['Rank', 'Likes', 'Dislikes']
for col in cols:
    check_numeric(data, col)
```

```
NameError: name 'data' is not defined
```

```
---------------------------------------------------------------
NameError                 Traceback (most recent call last)
Cell In[19], line 6
      4 cols = ['Rank', 'Likes', 'Dislikes']
      5 for col in cols:
----> 6     check_numeric(data, col)
NameError: name 'data' is not defined
```

Finishing Lab 5

Data quality checks (cont.)

- Unit tests for data
- Example 2: Checking outliers

```
def is_outlier(value,q1,q3):
    iqr = q3 - q1 # Interquartile range
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    return value < lower_bound or value > upper_bound

def column_has_outliers(data, column):
    q1 = data[column].quantile(0.25) # First quartile
    q3 = data[column].quantile(0.75) # Third quartile
    return any(data[column].apply(lambda x: is_outlier(x, q1, q3)))

assert not column_has_outliers(data, 'Likes_numeric'), "Likes has outliers"
```

```
NameError: name 'data' is not defined
---------------------------------------------------------------
NameError                 Traceback (most recent call last)
Cell In[20], line 12
      9     q3 = data[column].quantile(0.75) # Third quartile
     10     return any(data[column].apply(lambda x: is_outlier(x, q1, q3)))
---> 12 assert not column_has_outliers(data, 'Likes_numeric'), "Likes has outliers"
NameError: name 'data' is not defined
```

Linting

- A type of static analysis
  - Analyzing code without executing it
- Checks for: Code quality
- We'll be starting with ruff.

Example of Low Quality Code

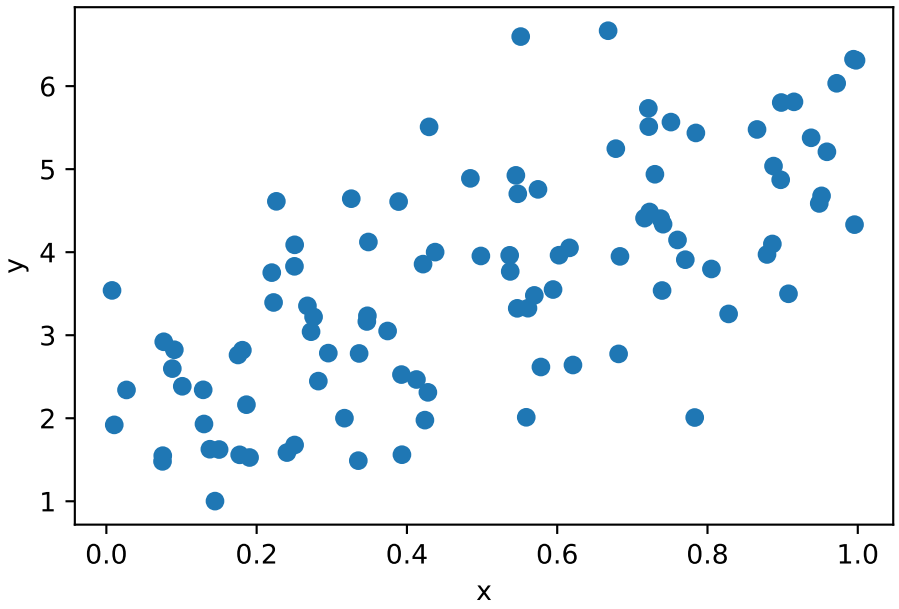
```python
import numpy as np
import pandas as pd

def simulate_data(n):
    x = np.random.uniform(0, 1, n)
    y = 2 + 3 * x + np.random.normal(0, 1, n)
    return x, y

from matplotlib import pyplot as plt

def plot_data(x, y):
    width = 100
    height = 100
    plt.scatter(x, y)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

plot_data(*simulate_data(100))
```



Continuous integration

- You're going to set up your tests and linting to run automatically every time you push code to GitHub.
- This is one of those times where you'll follow instructions without necessarily knowing what's going on
  - You'll learn more about it in this week's reading.

Workflows

- A workflow is an automated process made up of one or more jobs
- We use a YAML file to define our workflow configuration

```yaml
name: Run tests

on: push

jobs:
  tests:
    runs-on: ubuntu-latest
    steps:
```

```yaml
      - name: Clone repository
        uses: actions/checkout@v4
      # https://github.com/actions/setup-python
      - name: Install Python
        uses: actions/setup-python@v5
        with:
          python-version: "3.12"
          cache: pip
      - name: Install dependencies
        run: pip install -r requirements.txt
      - name: Run tests
        # https://pytest-cov.readthedocs.io/en/latest/readme.html
        run: pytest --cov
      # https://github.com/astral-sh/ruff-action
      - name: Run ruff
        uses: astral-sh/ruff-action@v3
        with:
          version: latest
```

Task

Steps

1. Install Ruff
    1. Install the ruff VSCode extension.
    2. Open up your Python files, you'll likely see some warnings.
        - Don't do anything with them yet.
2. Set up a GitHub Actions workflow
    1. In a branch, add a copy of `.github/workflows/tests.yml`.
    2. Create a pull request.
    3. View the results of the Actions run.
    4. If the workflow is failing, review the errors and address them.