DATA ANALYSIS & VISUALIZATION PROJECT

CTEC 298 – 101 SYMBOLIC COMPUTATION USE BIG DATA

FINAL PRESENTATION

BY: TORRODJAE SOMERVILLE & THERESHA MILLER

INTRODUCTION

- This project applies the full data-science workflow using Python and Tableau
- Each group member used their own dataset from CTEC 128 and reproduced the analysis using Python instead of Excel
- Our work included loading data, cleaning and transforming datasets, creating six required Python plots, and building two Tableau visualizations
- We documented the original datasets, the cleaned datasets, the code used, and the final outputs
- The goal was to show that the same analytical process can be applied to different data sources
- Demonstrated skills across Python, Jupyter Notebook, Matplotlib, Tableau, GitHub, NumPy, and Pandas

CTEC 128 PAPER SUMMARY: GROUP LEADER

- **Topic:**
  - Analyzing COVID-19 case trends and vaccination patterns in Maryland
- **Dataset:**
  - COVID-19 cases by county and date.
  - Vaccination rates by county (first dose and completed series)
  - Focus on Maryland counties only
- **Key Steps:**
  - Removed unneeded columns and incomplete records
  - Calculated **Daily New Cases** from cumulative totals
  - Used mean, median, and standard deviation to describe the data
  - Created charts to compare cases and vaccination trends over time
- **Main Finding:**
  - Short term: vaccinations and high case counts overlapped.
  - Long term: as vaccination rates increased, **daily new cases declined**, showing vaccines helped control spread

CTEC 298 – COURSE WORK SUMMARY (GROUP LEADER)

Introduced to multiple data-focused tutorials and hands-on tasks to understand Python tools:

- Learned and used several major tools & libraries:
  - **Jupyter Notebook** – interactive coding environment (.ipynb)
  - **NumPy** – numerical computing, arrays, math operations
  - **Pandas** – data loading, cleaning, filtering, merging, grouping
  - **Matplotlib** – Python plots (bar, pie, histogram, scatter, stack plot, multiplot)
  - **Tableau** – visual analytics tool for building dashboards
  - **GitHub** – repository for uploading files and managing code
- Imported and uploaded multiple files into GitHub using past CTEC 128 data
- Applied all tools to complete full workflow: load → clean → transform → visualize

ORGINIAL DATASETS:

Original dataset 1 – Cases sheet

- Sheet: MD_COVID-19_-_Cases_by_County
- Example Columns:
  - DATE
  - County columns (Allegany, Anne_Arundel)
  - Total Cases
  - Daily New Cases

```
# Load the Excel file once
excel_file = pd.ExcelFile("FINAL CTEC 128 DATASET.xlsx")

# Read each sheet into its own DataFrame
df_cases = pd.read_excel(excel_file, sheet_name="MD_COVID-19_-_Cases_by_County_2")
df_vac = pd.read_excel(excel_file, sheet_name="Edited US Vac Rec MD")
df_combined = pd.read_excel(excel_file, sheet_name="Sheet1")

# Look at the first few rows of each so we can document the "original datasets"
print("Cases sheet (MD_COVID-19_-_Cases_by_County_2):")
display(df_cases.head())

print("\nVaccination sheet (Edited US Vac Rec MD):")
display(df_vac.head())

print("\nCombined sheet (Sheet1):")
display(df_combined.head())
```

Cases sheet (MD_COVID-19_-_Cases_by_County_2):

	OBJECTID	DATE	Allegany	Anne_Arundel	Baltimore	Baltimore_City	Calvert	Caroline	Carroll	Cecil	...	Prince_Georges	Queen_Annes	Somerset	St_Marys	Tal
0	1	2020-03-15 06:00:00	0	2	3	1	0	0	1	0	...	9	0	0	0	
1	2	2020-03-16 06:00:00	0	1	4	1	0	0	1	0	...	15	0	0	0	
2	3	2020-03-17 06:00:00	0	3	6	1	0	0	1	0	...	14	0	0	0	
3	4	2020-03-18 06:00:00	0	4	10	4	0	0	1	0	...	20	0	0	0	
4	5	2020-03-19 06:00:00	0	5	12	8	1	0	2	0	...	23	0	0	0	

5 rows × 28 columns

ORIGINAL DATASET 2 (VACCINATION DATA)

Original dataset 2 – Vaccination sheet

- Example columns:
  - Date
  - Recip_State
  - Recip_County
  - Administered_Dose1_Pop_Pct
  - Series_Complete_Pop_Pct

```
# Load the Excel file once
excel_file = pd.ExcelFile("FINAL CTEC 128 DATASET.xlsx")

# Read each sheet into its own DataFrame
df_cases = pd.read_excel(excel_file, sheet_name="MD_COVID-19_-_Cases_by_County_2")
df_vac = pd.read_excel(excel_file, sheet_name="Edited US Vac Rec MD")
df_combined = pd.read_excel(excel_file, sheet_name="Sheet1")

# Look at the first few rows of each so we can document the "original datasets"
#print("Cases sheet (MD_COVID-19_-_Cases_by_County_2):")
#display(df_cases.head())

print("\nVaccination sheet (Edited US Vac Rec MD):")
display(df_vac.head())

print("\nCombined sheet (Sheet1):")
display(df_combined.head())
```

Vaccination sheet (Edited US Vac Rec MD):

	Date	DATE 2	MMWR_week	Recip_County	Recip_State	Completeness_pct	Administered_Dose1_Recip	Administered_Dose1_Pop_Pct	Administered_Dose1_Recip_5Plus
0	2023-05-10	May 2023	19	Allegany County	MD	98.2	45566	59.1	15110.0
1	2023-05-10	May 2023	19	Anne Arundel County	MD	98.2	528981	81.1	205628.0
2	2023-05-10	May 2023	19	Baltimore city	MD	98.2	463147	86.1	31892.0
3	2023-05-10	May 2023	19	Baltimore County	MD	98.2	697468	85.9	44800.0
4	2023-05-10	May 2023	19	Calvert County	MD	98.2	76487	95.0	1144180.0

5 rows × 80 columns

DATA CLEANING & WRANGLING OVERVIEW

Goals of cleaning:

- Focus on **Maryland counties only**
- Link **cases** and **vaccination** data
- Build smaller “final” datasets for each plot

Main wrangling steps:

  - Dropped rows without Recip_County
  - Selected only needed columns for county-level analysis
  - Grouped by Recip_County and took **max** cumulative cases
  - Converted DATE and Date columns to real datetime
  - Filtered vaccination sheet to Recip_State == 'MD'
  - Grouped vaccination data by Date and averaged vaccination %
  - Merged **cases by date** with **vaccination by date** into df_daily

```
# 1. Focus on Maryland counties only (remove rows with no county name)
df_counties = df_combined.dropna(subset=['Recip_County']).copy()

# 2. Select only needed columns for county-level analysis
df_counties = df_counties[
    ['Recip_County', 'Cumulative Cases ',
     'Administered_Dose1_Pop_Pct', 'Series_Complete_Pop_Pct']
]

# 3. Group by Recip_County and take the max cumulative cases
df_counties = df_counties.groupby('Recip_County', as_index=False).max()

# 4. Convert DATE column in cases sheet + Date column in vaccination sheet to datetime
df_cases['DATE'] = pd.to_datetime(df_cases['DATE'])
df_vac['Date']  = pd.to_datetime(df_vac['Date'], errors='coerce')

# 5. Filter vaccination sheet to Maryland only (Recip_State == 'MD')
df_vac_md = df_vac[df_vac['Recip_State'] == 'MD'].copy()

# 6. Group vaccination data by Date and average vaccination % for the whole state
vacc_by_date = df_vac_md.groupby('Date', as_index=False)[
    'Administered_Dose1_Pop_Pct'
].mean()

# 7. Build a date + daily new cases dataset from cases sheet
cases_by_date = df_cases[['DATE', 'Daily New Cases']].copy()
cases_by_date = cases_by_date.rename(
    columns={'DATE': 'Date', 'Daily New Cases': 'Daily_New_Cases'}
)

# Normalize dates to remove time so the merge works cleanly
vacc_by_date['Date']  = vacc_by_date['Date'].dt.normalize()
cases_by_date['Date'] = cases_by_date['Date'].dt.normalize()

# 8. Merge cases and vaccination data together by Date --> df_daily
df_daily = pd.merge(cases_by_date, vacc_by_date, on='Date', how='inner')

# Sort final daily dataset by date
df_daily = df_daily.sort_values('Date')

# Show the cleaned + merged dataset
df_daily.head()
```

BAR PLOT: DATA USED

- **Plot 1: Bar Plot – Cumulative Cases by County**
- **Original dataset used:**
- df_combined (from Sheet1)
- **Cleaning / Final dataset (df_counties):**
- Dropped rows with missing Recip_County
- Kept columns:
  - Recip_County
  - Cumulative Cases
  - Administered_Dose1_Pop_Pct
  - Series_Complete_Pop_Pct
- Grouped by Recip_County and took maximum Cumulative Cases

```
display(df_counties.head())
```

	Recip_County	Cumulative Cases	Administered_Dose1_Pop_Pct	Series_Complete_Pop_Pct	Vaccination_Level
0	Allegany County	24690.0	64.7	52.2	50-69%
1	Anne Arundel County	129879.0	91.3	72.9	70%+
2	Baltimore County	188297.0	84.3	75.0	70%+
3	Baltimore city	163926.0	78.0	76.8	70%+
4	Calvert County	16073.0	82.7	94.0	70%+

BAR PLOT: CODE & VISUALIZATION

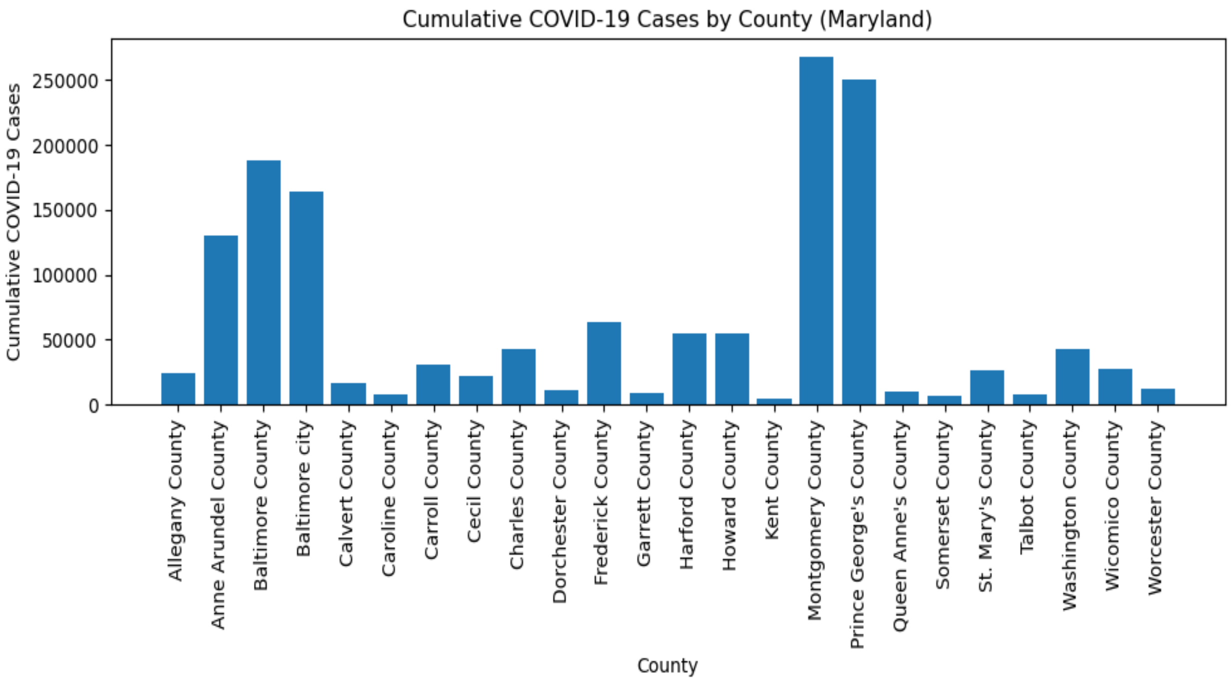
Key Python steps:

- df_counties = df_combined.dropna(subset=['Recip_County'])
- Selected needed columns for counties
- groupby('Recip_County').max() to get one row per county
- plt.bar(df_counties['Recip_County'], df_counties['Cumulative Cases '])

Plot interpretation (what it shows):

- Shows which Maryland counties have the **highest cumulative cases**
- Easy to compare counties side-by-side
- Highlights counties with the greatest COVID-19 burden

```python
plt.figure(figsize=(10,5))
#Plotting bar graph
plt.bar(df_counties['Recip_County'], df_counties['Cumulative Cases '])
plt.xticks(rotation=90)
plt.xlabel("County")
plt.ylabel("Cumulative COVID-19 Cases")
plt.title("Cumulative COVID-19 Cases by County (Maryland)")
plt.tight_layout()
plt.show()
```



PIE CHART: DATA & CATEGORIES

- **Plot 2: Pie Chart – Vaccination Level by County**
- **Original dataset used:** df_counties (cleaned county dataset)
- **Final dataset:**
- Added Vaccination_Level with categories:
  - Below 50%
  - 50–69%
  - 70%+
- Used value_counts() to count how many counties in each category

```
display(df_counties[['Recip_County','Series_Complete_Pop_Pct','Vaccination_Level']].head())
```

	Recip_County	Series_Complete_Pop_Pct	Vaccination_Level
0	Allegany County	52.2	50-69%
1	Anne Arundel County	72.9	70%+
2	Baltimore County	75.0	70%+
3	Baltimore city	76.8	70%+
4	Calvert County	94.0	70%+

PIE CHART: CODE AND VISUALIZATION

Key Python steps:

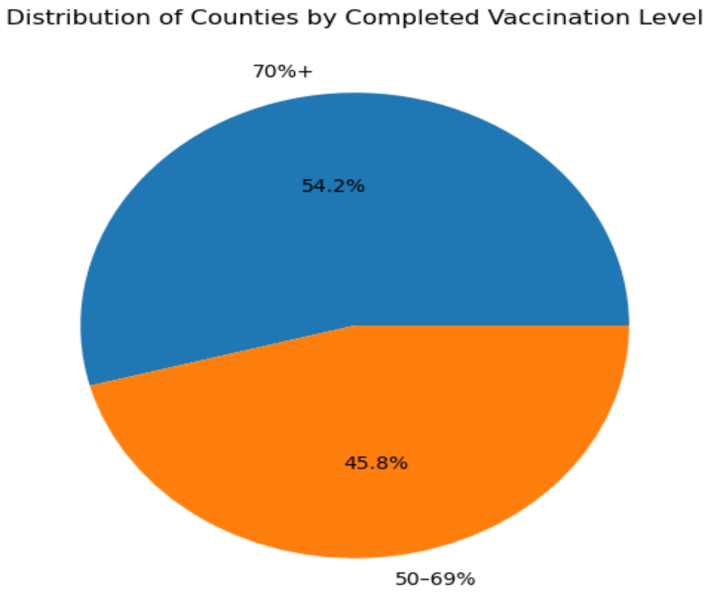
- Defined **vacc_category(pct)** function
- Applied to Series_Complete_Pop_Pct → created Vaccination_Level
- pie_counts = df_counties['Vaccination_Level'].value_counts()
- Built pie chart with **plt.pie(...)**

Plot interpretation:

- Shows **percentage** of counties in each **vaccination level**
- Summarizes how well counties are doing overall with completed vaccination series
- **No** Counties have **below 50% of population Vaccinated**

```
Vaccination_Level
70%+      13
50-69%    11
Name: count, dtype: int64
```

```python
plt.figure(figsize=(6,6))
plt.pie(pie_counts.values, labels=pie_counts.index, autopct='%1.1f%%')
plt.title("Distribution of Counties by Completed Vaccination Level")
plt.show()
```



HISTOGRAM: DATA USED

Plot 3: Histogram – Daily New Cases Distribution

Original dataset used:

- df_cases (MD_COVID-19_-_Cases_by_County_2)

Final dataset (df_hist):

- Kept one column: **Daily New Cases**
- Represents number of new cases per day across Maryland

```
4]: display(df_hist.head())
    display(df_hist.tail())
```

	Daily New Cases
0	0
1	6
2	20
3	28
4	22

	Daily New Cases
1720	34
1721	33
1722	22
1723	23
1724	147

HISTOGRAM: CODE & VISUALIZATION

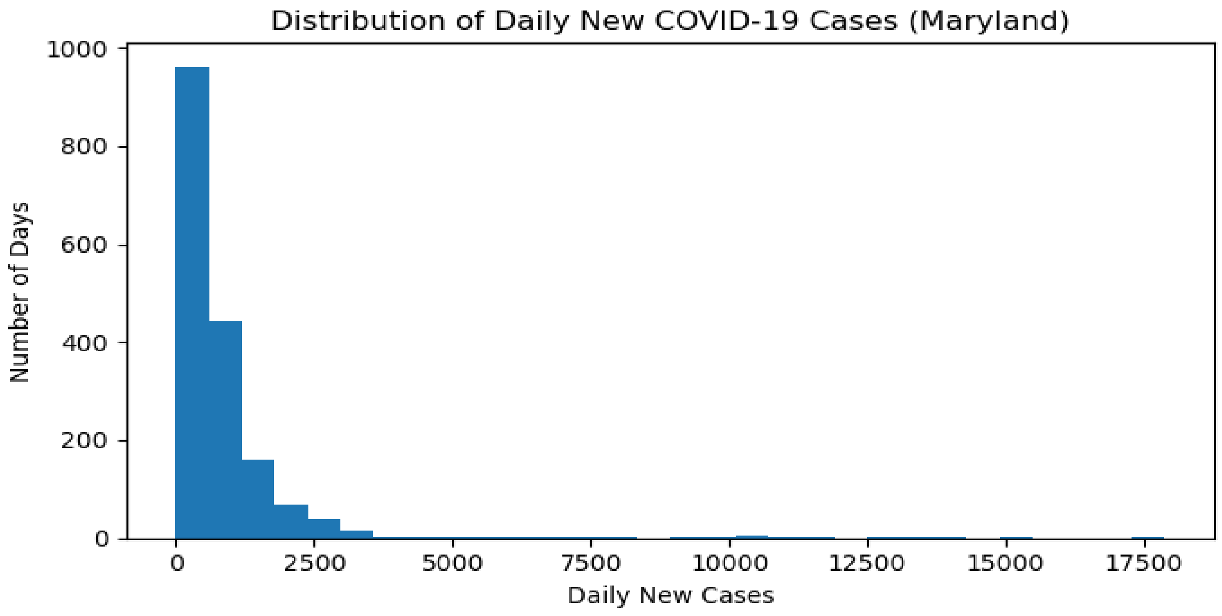
Key Python steps:

- df_hist = df_cases[['Daily New Cases']].copy()
- plt.hist(df_hist['Daily New Cases'], bins=30)

```
plt.figure(figsize=(7,4))
plt.hist(df_hist['Daily New Cases'], bins=30)
plt.title("Distribution of Daily New COVID-19 Cases (Maryland)")
plt.xlabel("Daily New Cases")
plt.ylabel("Number of Days")
plt.tight_layout()
plt.show()
```

Plot interpretation:

- Shows how often certain daily new case counts occurred
- Reveals whether most days were low, moderate, or high in new cases



SCATTER PLOT: DATA USED

Plot 4: Scatter Plot – Vaccination vs Cases (County Level)

Original dataset used:

- df_counties

Final dataset (same DataFrame):

- X-axis: Series_Complete_Pop_Pct (completed vaccination %)
- Y-axis: Cumulative Cases

```
display(df_counties[['Recip_County','Series_Complete_Pop_Pct','Cumulative Cases ']].head())
```

	Recip_County	Series_Complete_Pop_Pct	Cumulative Cases
0	Allegany County	52.2	24690.0
1	Anne Arundel County	72.9	129879.0
2	Baltimore County	75.0	188297.0
3	Baltimore city	76.8	163926.0
4	Calvert County	94.0	16073.0

SCATTER PLOT: CODE & VISUALIZATION

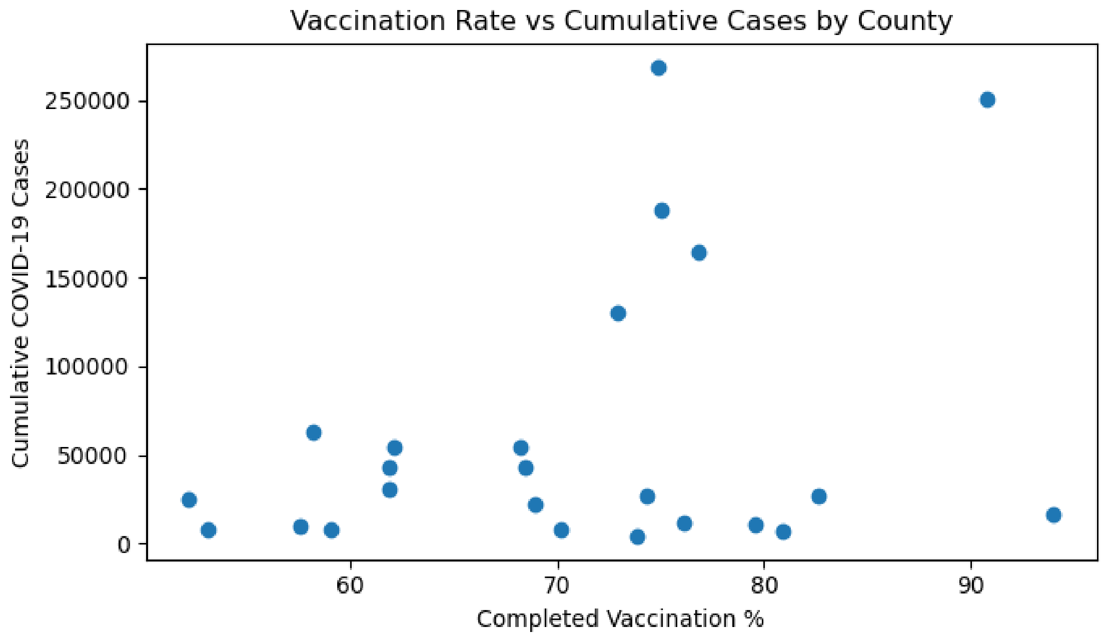
Key Python steps:

- plt.scatter(df_counties['Series_Complete_Pop_Pct'], df_counties['Cumulative Cases '])

Plot interpretation:

- Each point = one county
- Let's us see how case totals relate to vaccination coverage
- Encourages discussion about timing, population size, and county differences

```
plt.figure(figsize=(7,4))
plt.scatter(df_counties['Series_Complete_Pop_Pct'], df_counties['Cumulative Cases '])
plt.xlabel("Completed Vaccination %")
plt.ylabel("Cumulative COVID-19 Cases")
plt.title("Vaccination Rate vs Cumulative Cases by County")
plt.tight_layout()
plt.show()
```



STACKPLOT: DATA USED

Plot 5: Stack Plot – Vaccination & Daily Cases Over Time

Original datasets used:

- df_cases (for Daily New Cases)
- df_vac (for Administered_Dose1_Pop_Pct)

Final merged dataset (df_daily):

- Columns:
- Date
- Daily_New_Cases
- Administered_Dose1_Pop_Pct (average across Maryland counties)
- Dates normalized and merged from both sheets

```
display(df_daily.head())
```

	Date	Daily_New_Cases	Administered_Dose1_Pop_Pct
0	2021-09-17	1525	53.771429
1	2021-09-18	1277	60.272000
2	2021-09-19	1036	60.332000
3	2021-09-20	1139	60.380000
4	2021-09-21	974	60.420000

STACKPLOT: CODE & VISUALIZATION

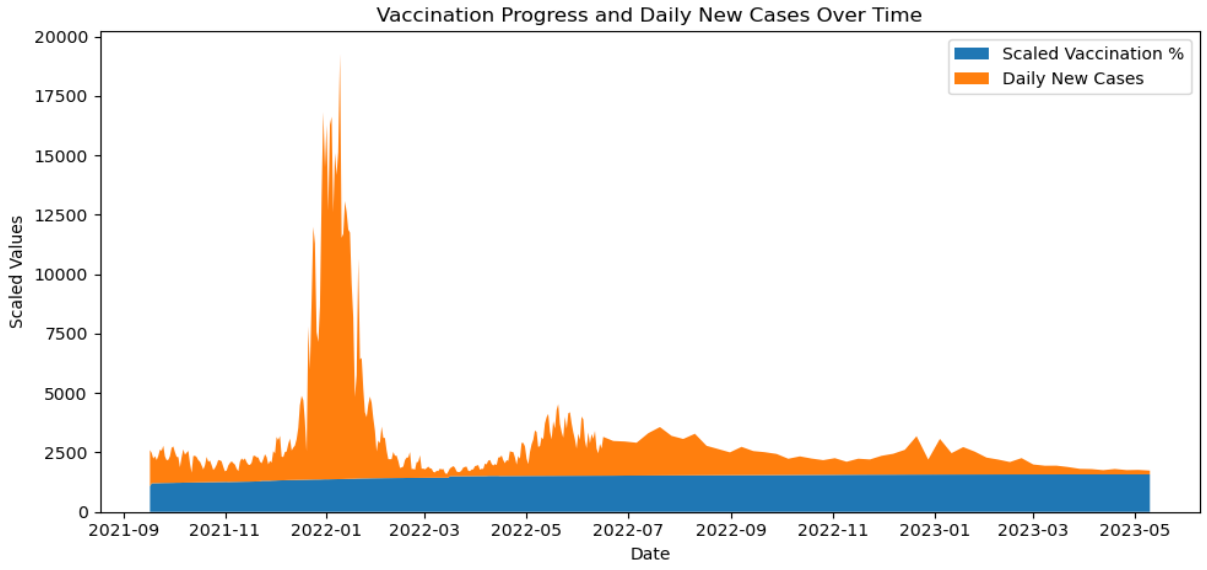
Key Python steps:

- Converted **both** date columns to **datetime**
- Filtered df_vac where Recip_State == 'MD'
- groupby('Date')['Administered_Dose1_Pop_Pct'].mean() for statewide vaccination %
- Built cases_by_date from df_cases[['DATE','Daily New Cases']]
- Renamed and normalized Date columns
- pd.merge(cases_by_date, vacc_by_date, on='Date', how='inner') → df_daily
- Scaled vaccination % for visibility: scaled_vacc = df_daily['Administered_Dose1_Pop_Pct'] * 20
- Used plt.stackplot(Date, scaled_vacc, Daily_New_Cases, ...)

Plot interpretation:

- Shows vaccination progress and daily new cases **on the same timeline**
- Makes it easier to see how trends overlapped and changed over time

```
plt.figure(figsize=(10,5))
# Scale vaccination % upward so it becomes visible on the stackplot
scaled_vacc = df_daily['Administered_Dose1_Pop_Pct'] * 20  # adjust multiplier as needed
daily_cases = df_daily['Daily_New_Cases']

plt.stackplot(
    df_daily['Date'],
    scaled_vacc,
    daily_cases,
    labels=["Scaled Vaccination %", "Daily New Cases"]
)
plt.legend()
plt.title("Vaccination Progress and Daily New Cases Over Time")
plt.xlabel("Date")
plt.ylabel("Scaled Values")
plt.tight_layout()
plt.show()
```



MULTIPLOT: DATA USED

Plot 6: Multiplot – Two Time Series

Dataset used:

- Same df_daily as the stack plot

Columns:

- Date
- Administered_Dose1_Pop_Pct
- Daily_New_Cases

```
display(df_daily[['Date','Administered_Dose1_Pop_Pct','Daily_New_Cases']].head())
display(df_daily[['Date','Administered_Dose1_Pop_Pct','Daily_New_Cases']].tail())
```

	Date	Administered_Dose1_Pop_Pct	Daily_New_Cases
0	2021-09-17	53.771429	1525
1	2021-09-18	60.272000	1277
2	2021-09-19	60.332000	1036
3	2021-09-20	60.380000	1139
4	2021-09-21	60.420000	974

	Date	Administered_Dose1_Pop_Pct	Daily_New_Cases
315	2023-04-12	78.925000	178
316	2023-04-19	78.916667	227
317	2023-04-26	78.937500	180
318	2023-05-03	78.958333	188
319	2023-05-10	78.962500	160

MULTIPLOT: CODE & VISUALIZATION

Key Python steps:

- fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
- **Top axis:** ax1.plot(Date, Administered_Dose1_Pop_Pct)
- **Bottom axis:** ax2.plot(Date, Daily_New_Cases)

Plot interpretation:

- Separates vaccination trend and daily new case trend
- Easier to compare shapes and timing between the two lines
- Supports discussion of how long-term vaccination may relate to changes in daily cases

```python
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,6), sharex=True)
# Top subplot: vaccination percentage over time
ax1.plot(df_daily['Date'], df_daily['Administered_Dose1_Pop_Pct'])
ax1.set_ylabel("Vaccination %")
ax1.set_title("Vaccination Rate Over Time (Maryland)")
# Bottom subplot: daily new cases over time
ax2.plot(df_daily['Date'], df_daily['Daily_New_Cases'])
ax2.set_xlabel("Date")
ax2.set_ylabel("Daily New Cases")
ax2.set_title("Daily New COVID-19 Cases Over Time (Maryland)")

plt.tight_layout()
plt.show()
```

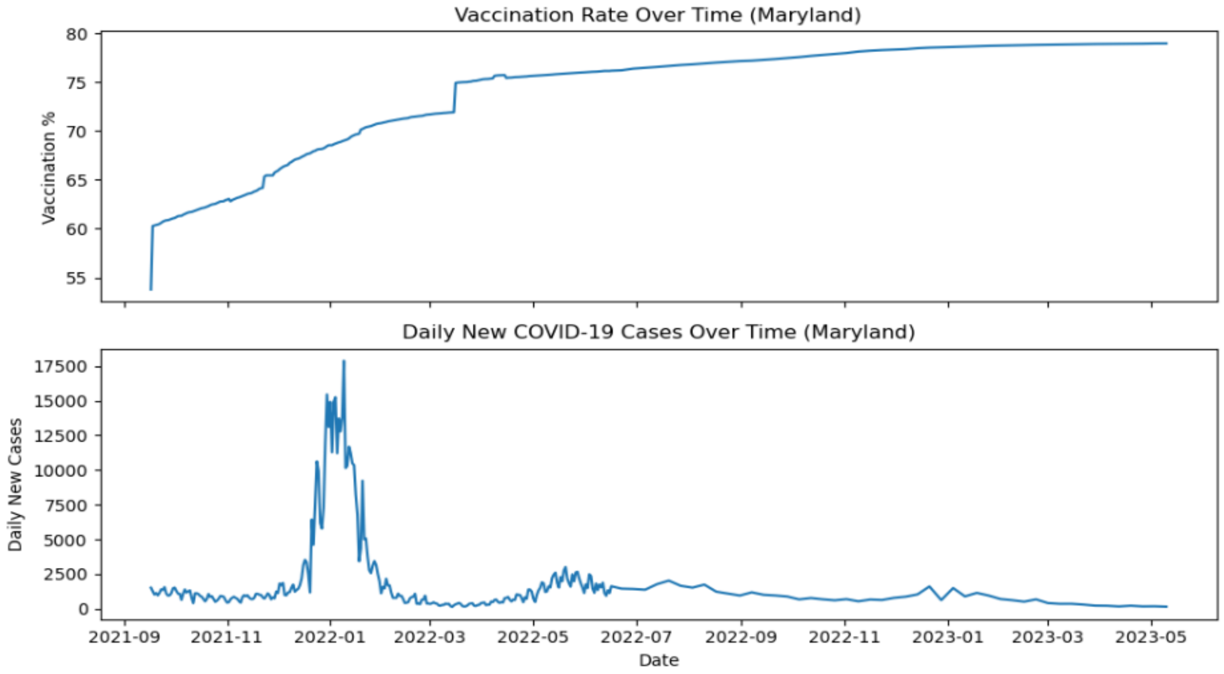


TABLEAU PLOT #1: QUARTERLY COVID-19 CASES (BAR CHART)

- Displays quarterly COVID-19 case totals for **Wicomico** and **Baltimore City**
- Columns **represent Year** → **Quarter** (e.g., 2020 Q1–Q4, etc.)
- Bars are **stacked by category**, showing how totals shift over time
- Allows comparison of case levels between two counties on a quarterly timeline
- Shows how both counties experienced noticeable jumps during early pandemic surges
- Highlights seasonal patterns and differences in how each county was impacted

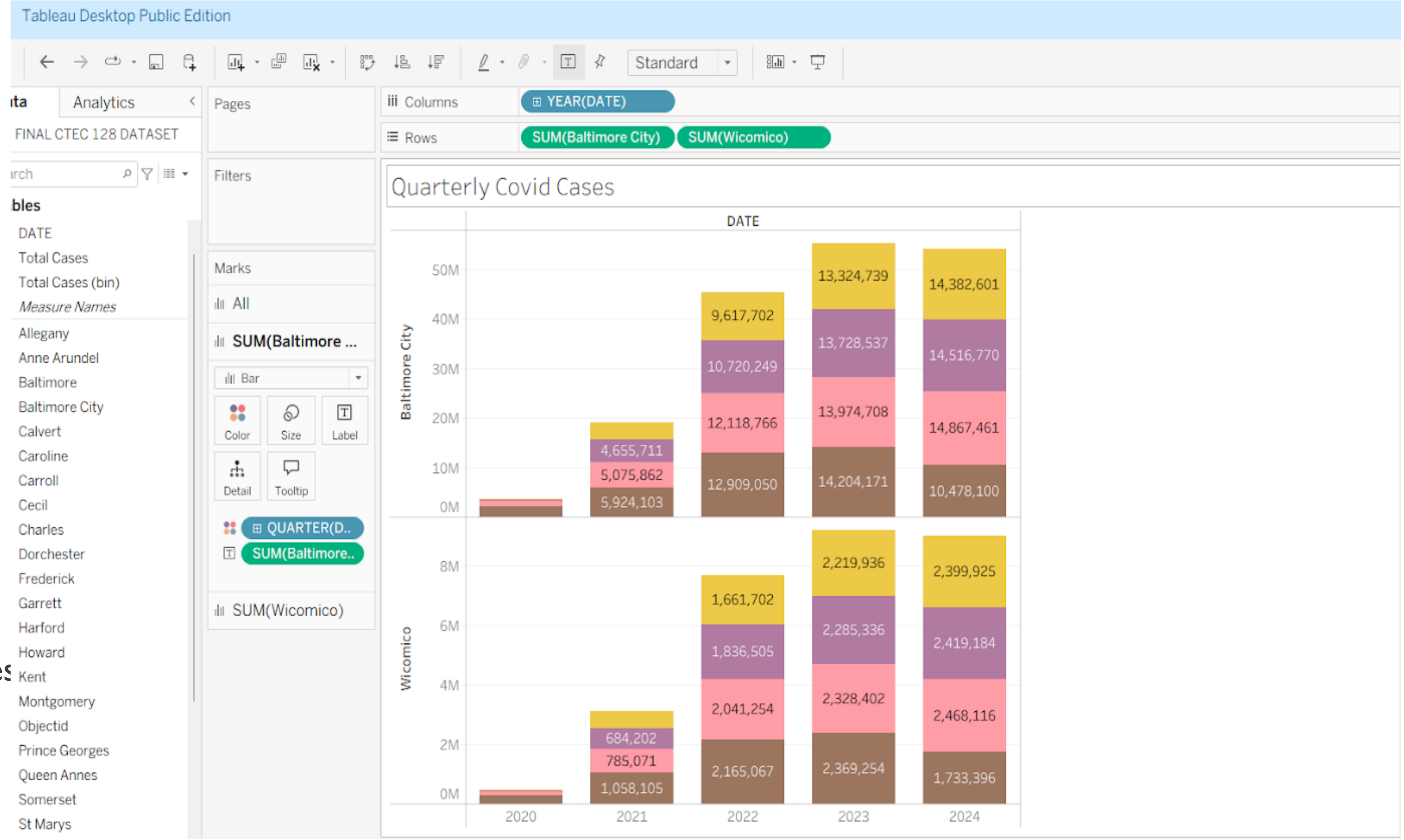
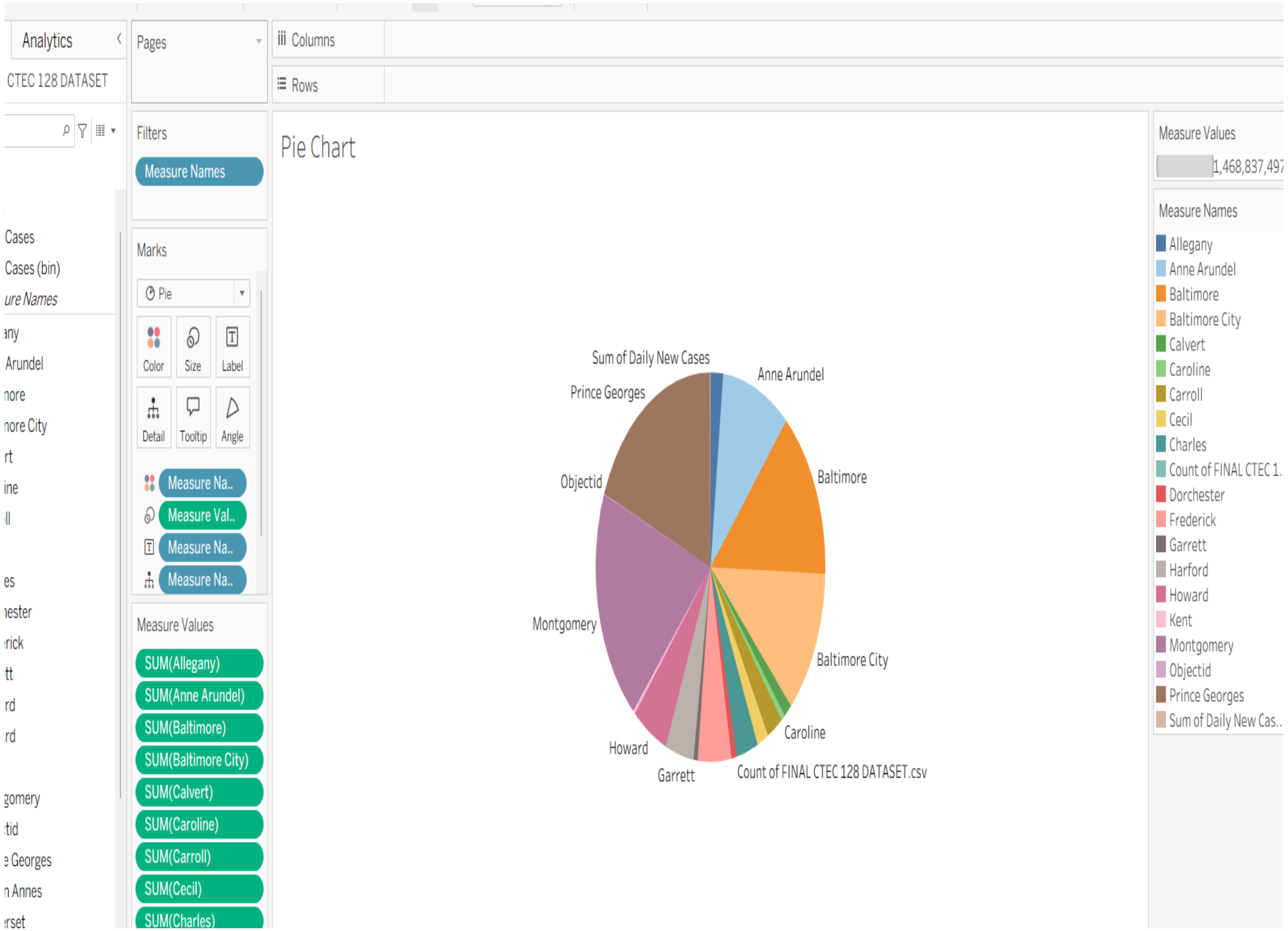


TABLEAU PLOT #2: DAILY NEW CASES BY COUNTY (PIE CHART)

- Pie chart **shows share of total daily new cases** contributed by each Maryland county
- Slice size = **sum of all daily new cases** in the dataset for that county
- Highlights counties with the highest overall impact
- Baltimore City and Baltimore County contribute the largest portions
- Smaller counties show noticeably smaller slices, reflecting lower total case counts
- Offers a clear "big-picture" view of how heavily each area was affected



CTEC 128 PAPER SUMMARY: MEMBER 2

Topic:

- Occupation of African American Maryland residents aged 18-62 with a focus on gender

Dataset

- **Source:** U.S. Census Public Use Microdata Sample (PUMS)
- **Population filtered:**
- Race: African American
- Age: 18–65 (working-age population)
- Employment status: Employed individuals only

Key steps:

- Filtered dataset to include African American respondents aged 18–65 who were employed
- Cleaned data by removing blanks, standardizing occupation labels, and grouping similar job titles
- Consolidated occupations into broader categories (healthcare, service, management, transportation, education, administrative support)

Main Findings

- African Americans are most concentrated in service, transportation, education, and healthcare support roles
- Managerial and healthcare occupations showed higher mean and median earnings
- Service and transportation roles reflected lower earnings overall

CTEC 298 COURSEWORK SUMMARY (MEMBER 2)

- **Coursework Summary**
- **Objective:** Examined occupational representation and earnings among African Americans aged 18–65 using Census PUMS data.
- **Data Preparation:**
  - Filtered for African American working-age adults.
  - Cleaned occupation labels and grouped jobs into major sectors.
- **Methods & Visuals:**
  - Bar and pie charts to show the most common jobs and sector distribution.
  - Histogram and scatter plots to analyze wage trends and age-income relationships.
- **Key Findings:**
  - Representation spans many fields but clusters in service, healthcare support, education, and transportation.
  - Higher earnings seen in management and healthcare roles; lower wages in service-related sectors.
  - Patterns reveal equity gaps and limited access to higher-earning positions.
- **Relevance to Symbolic Computation / Big Data:**
  - Demonstrates cleaning, transforming, and analyzing large datasets.
  - Shows how visualization reveals demographic and occupational patterns.

BAR PLOT: CODE & VISUALIZATION

Data Used

- Dataset loaded from **"Data Visualization.csv"**.
- Key column: OCCP
  - Represents occupation codes for individuals.

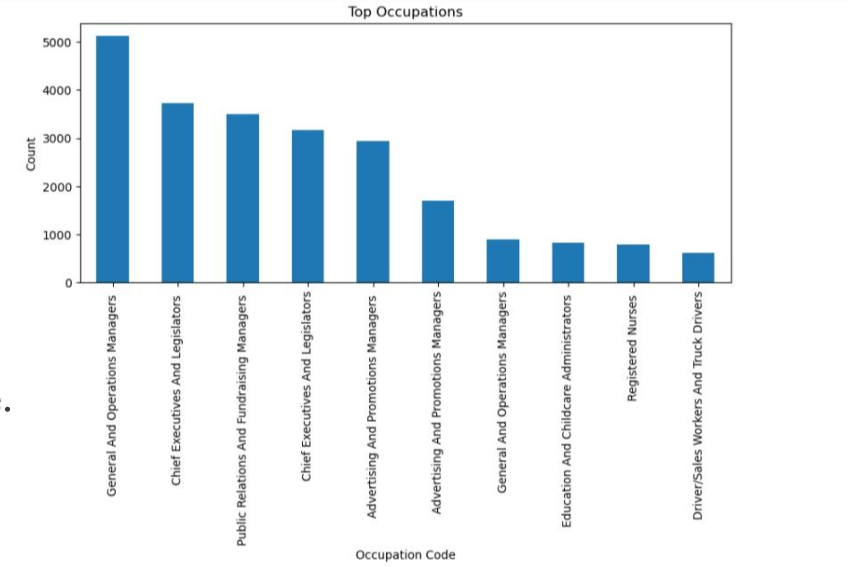
Key Python Steps

- Loaded the dataset using pd.read_csv().
- Counted how many times each occupation appears with value_counts().
- Selected the **top 10 most common occupations** using .head(10).
- Plotted the results as a **bar chart** with labels and a title.

Plot Interpretation

- Displays the **10 most frequent occupations** in the dataset.
- Taller bars represent jobs with **more individuals**.
- Highlights occupational concentration a few codes dominate the workforce.
- Useful for identifying **dominant job categories** within the population.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load your dataset (replace with your actual filename)
df = pd.read_csv("Data Visualization.csv")

# --- 1. Bar Plot: Top Occupations ---
plt.figure(figsize=(10,4))
df['OCCP'].value_counts().head(10).plot(kind='bar')
plt.title("Top Occupations")
plt.xlabel("Occupation Code")
plt.ylabel("Count")
plt.show()
```

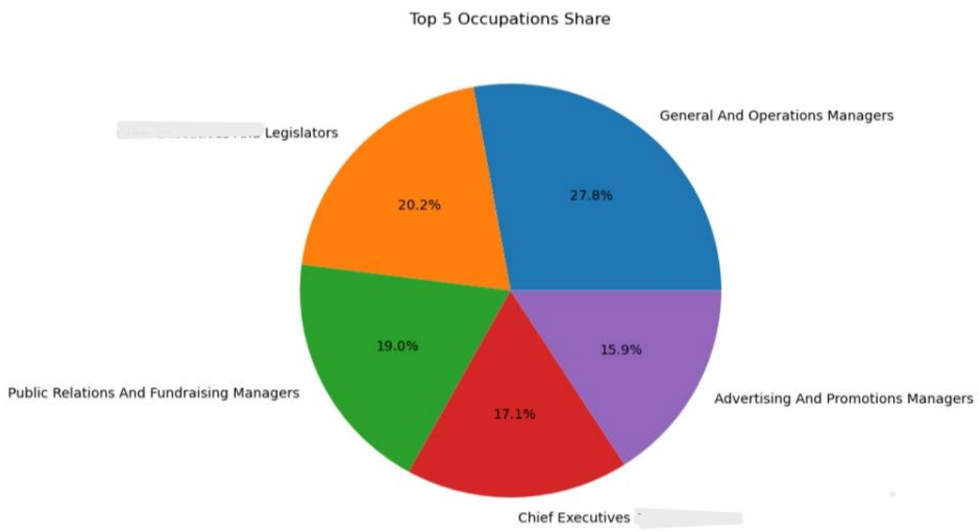


PIE CHART: CODE & VISUALIZATION

Data Used
- Dataset contains an **occupation column (OCCP)**.
- Each value represents a coded occupation for an individual.
- Analysis focuses on **the top 5 occupations by frequency**.

```
# --- 2. Pie Chart: Top 5 Occupations ---
plt.figure(figsize=(7,7))
df['OCCP'].value_counts().head(5).plot(kind='pie', autopct='%1.1f%%')
plt.title("Top 5 Occupations Share")
plt.ylabel("")  # remove y-label
plt.show()
```

Key Python Steps
- Used `value_counts()` to calculate **how many people are in each occupation**.
- Selected the **top 5 occupations** using `.head(5)`.
- Created a **pie chart** to show each occupation's proportional share.
- Used `autopct='%1.1f%%'` to display percentage values on the chart.



Plot Interpretation
- Pie chart shows **how the five most common occupations compare proportionally**.
- Larger slices represent occupations with **more people**.
- Highlights that certain occupations make up a **significant share** of the dataset.
- Helps visualize **dominance or balance** between major occupation categories.

HISTOGRAM: CODE & VISUALIZATION

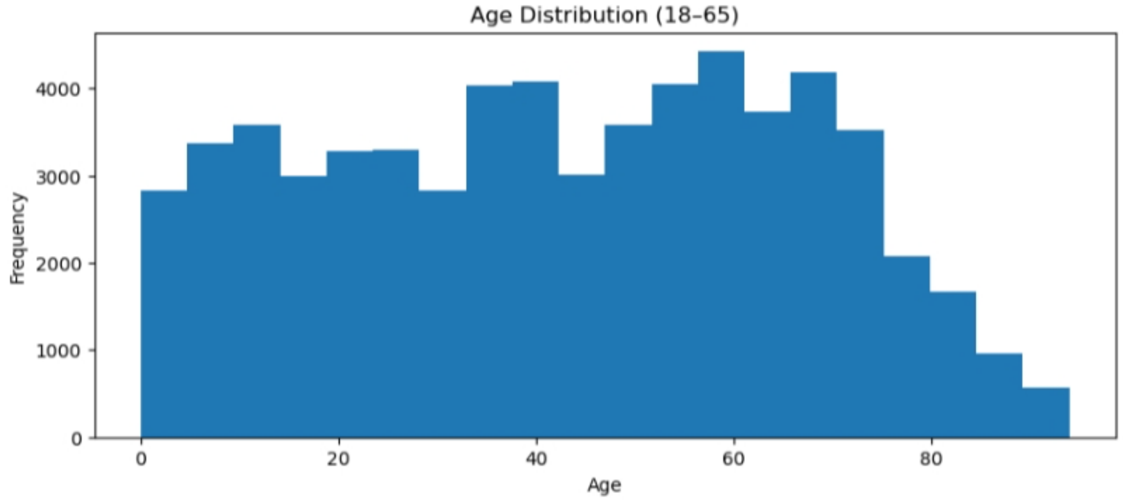
Data Used

- Dataset includes an **AGEP column** representing individual ages.
- Ages are assumed to be within a working-age range (18–65).

```python
# --- 3. Histogram: Age Distribution ---
plt.figure(figsize=(10,4))
df['AGEP'].plot(kind='hist', bins=20)
plt.title("Age Distribution (18-65)")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```

Key Python Steps

- Selected the **AGEP column** for analysis.
- Used `.plot(kind='hist')` to create a histogram.
- Set `bins=20` to divide ages into **20 equal-width groups**.
- Displayed the distribution with labeled axes and a title.



Plot Interpretation

- Histogram shows **how ages are spread across the population**.
- Taller bars represent age ranges with **more individuals**.
- Helps identify:
  - **Most common age groups**
  - Whether the population is **young-, mid-, or older-skewed**
  - Distribution shape (e.g., even, clustered, or declining)

SCATTER PLOT: CODE & VISUALIZATION

Data Used

- Dataset includes:
  - AGEP → Age of individuals
  - WAGP → Annual wages or income
- Used to analyze the **relationship between age and earnings**.

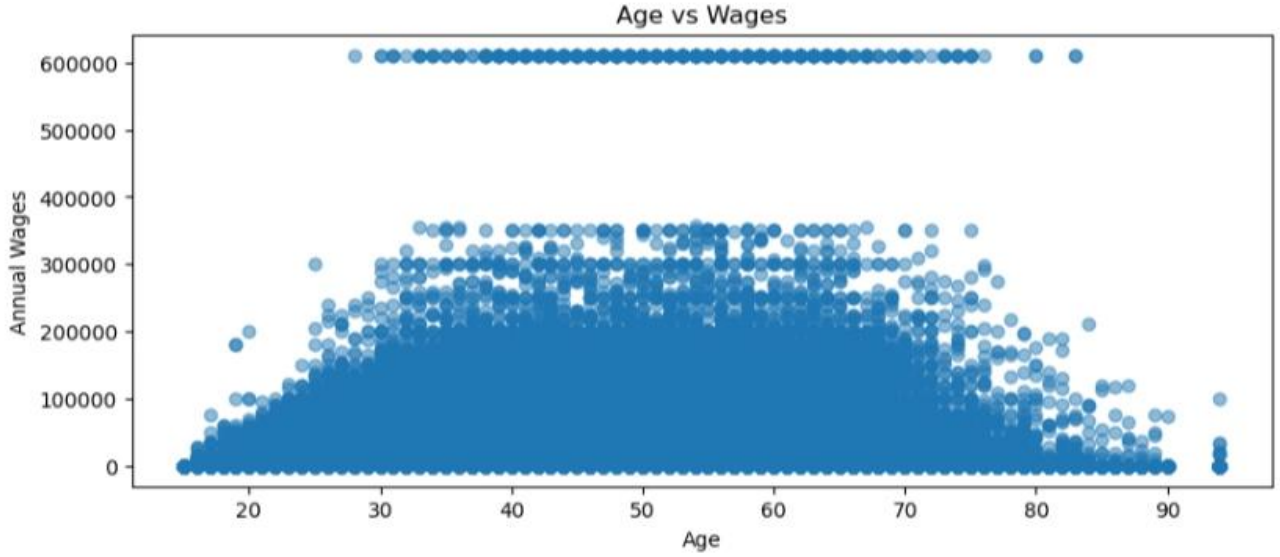
Key Python Steps

- Retrieved age values (`df['AGEP']`) and wage values (`df['WAGP']`).
- Used `plt.scatter()` to create a **scatter plot**.
- Applied `alpha=0.5` to make points **semi-transparent** for clarity.
- Labeled axes and added a title for interpretation.

Plot Interpretation

- Shows **how wages vary across different ages**.
- Each dot represents **one individual's age and income**.
- Helps reveal trends:
  - Wages may **increase with age** up to a point.
  - Possible **wide variation in income** across ages.
- No clear pattern may suggest **many other factors affect wages**.

```python
# --- 4. Scatter Plot: Age vs Wages ---
plt.figure(figsize=(10,4))
plt.scatter(df['AGEP'], df['WAGP'], alpha=0.5)
plt.title("Age vs Wages")
plt.xlabel("Age")
plt.ylabel("Annual Wages")
plt.show()
```



STACK PLOT: CODE & VISUALIZATION

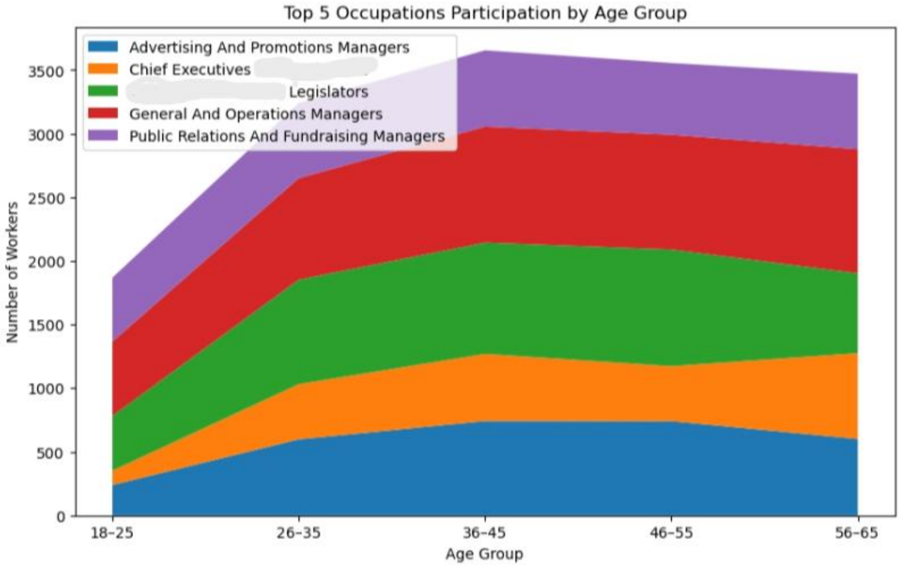
Data Used

- Dataset includes:
  - AGEP → Age of individuals
  - OCCP → Occupation code
- Focuses only on the **top 5 occupations**.
- Ages are grouped into five ranges:
  18–25, 26–35, 36–45, 46–55, 56–65.

```python
# --- 5. Stack Plot: Occupations by Age Group (Top 5 only) ---
# Create age groups
df['age_group'] = pd.cut(df['AGEP'], bins=[18,25,35,45,55,65],
                         labels=["18-25","26-35","36-45","46-55","56-65"])

# Find top 5 occupations overall
top_occp = df['OCCP'].value_counts().head(5).index

# Filter dataset to only those occupations
df_top = df[df['OCCP'].isin(top_occp)]

# Pivot table for stack plot
stack_data = df_top.pivot_table(index='age_group', columns='OCCP', aggfunc='size', fill_value=0)

# Plot
plt.figure(figsize=(10,6))
plt.stackplot(stack_data.index, stack_data.T, labels=stack_data.columns)
plt.title("Top 5 Occupations Participation by Age Group")
plt.xlabel("Age Group")
plt.ylabel("Number of Workers")
plt.legend(loc='upper left')
plt.show()
```

Key Python Steps

- Created **age group categories** using `pd.cut()`.
- Identified the **top 5 occupations** using `value_counts().head(5)`.
- Filtered the dataset to include **only those occupations**.
- Built a **pivot table** to count workers in each occupation by age group.
- Created a **stack plot** using `plt.stackplot()` to compare participation across ages.



Plot Interpretation

- Shows how **the top 5 occupations are distributed across age groups**.
- Each colored area represents **one occupation's workers**.
- Wider sections indicate **more workers in that age group**.
- Helps reveal:
  - Which age groups dominate certain occupations
  - How occupation participation shifts with age
  - Whether some jobs attract younger vs. older workers

MULTIPLOT: CODE & VISUALIZATION

Data Used

- Columns used:
  - AGEP → Age of individuals
  - WAGP → Annual wages/income
- Used to analyze **how income changes with age** and **population size per age**.

Key Python Steps

- Used `groupby('AGEP')['WAGP'].mean()` to calculate **average income by age**.
- Plotted the average income trend in **subplot 1**.
- Counted the number of individuals at each age using `value_counts()`.
- Sorted ages numerically and plotted counts in **subplot 2**.
- Used `plt.subplots()` to place both charts vertically in one figure.

Plot Interpretation

- **Top plot (Average Income by Age):**
  - Shows how earnings **increase, peak, or decline** with age.
  - Indicates whether income rises over time and at what ages it stabilizes.
- **Bottom plot (Worker Count by Age):**
  - Displays **how many people are represented per age**.
  - Helps identify whether certain age groups are **over- or under-represented**.
- **Together they help answer:**
  - Do ages with higher income have **more or fewer workers**?
  - How earnings progression relates to age distribution.

```python
# --- 6. Multiplot: Income vs Age ---
fig, ax = plt.subplots(2, 1, figsize=(10,8))

# Subplot 1 - Average income by age
df.groupby('AGEP')['WAGP'].mean().plot(ax=ax[0])
ax[0].set_title("Average Income by Age")
ax[0].set_xlabel("Age")
ax[0].set_ylabel("Income")

# Subplot 2 - Count of individuals by age
df['AGEP'].value_counts().sort_index().plot(ax=ax[1])
ax[1].set_title("Worker Count by Age")
ax[1].set_xlabel("Age")
ax[1].set_ylabel("Count")

plt.tight_layout()
plt.show()
```

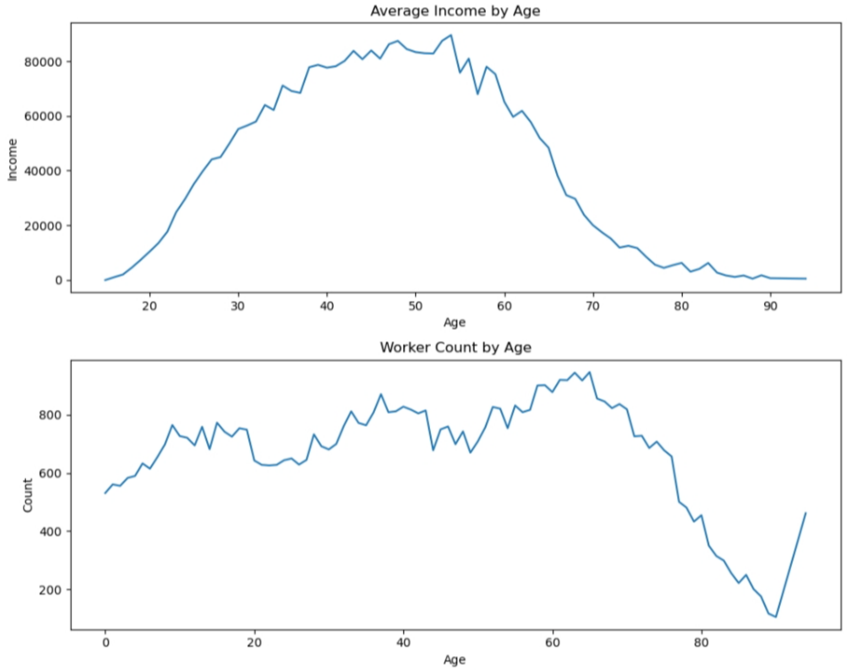


TABLEAU PLOT 1 (BAR CHART)

What It Shows

- A **vertical bar chart** displaying the **10 most common occupations** in your dataset.
- Each bar represents an **occupation code (OCCP)**.
- The **height of the bar** reflects the **number of individuals** in that occupation.

How to Interpret It

- The **taller the bar**, the **more people** work in that occupation.
- Occupation code **Bookkeeping, Accounting, and Auditing Clerks** has the highest count, meaning it's the most common job in your dataset.
- Occupation code **Electricians** has the lowest among the top 10, but still ranks higher than all others not shown.

Key Insights

- You can identify **which occupations dominate** the workforce.
- This helps in understanding **labor market concentration,** where most people are clustered in a few job types.
- If you're comparing regions or demographics, this chart can show **which jobs are most prevalent** in each group.

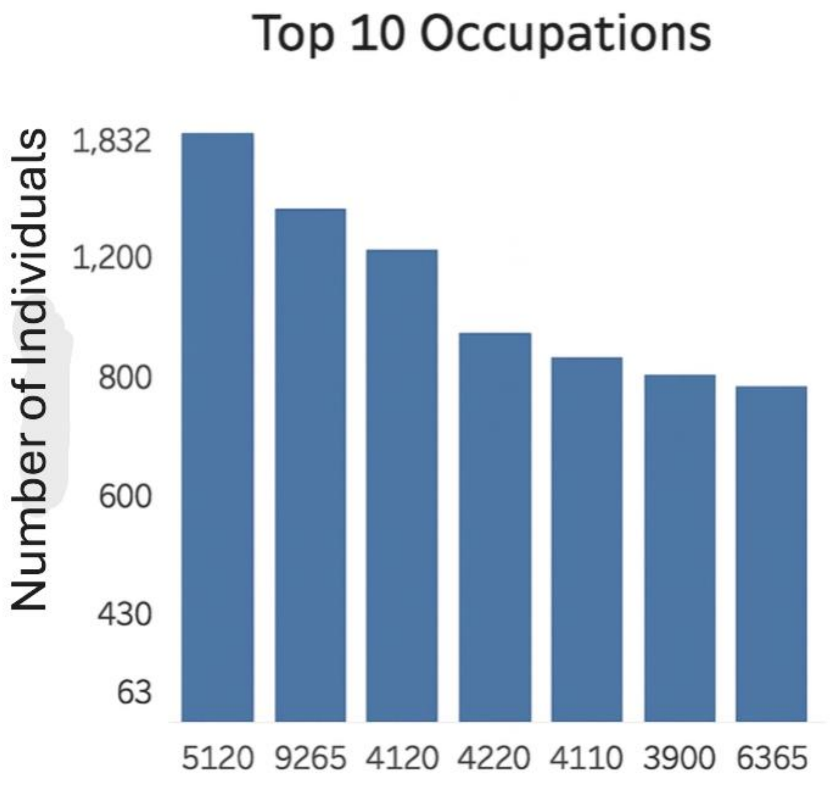


TABLEAU PLOT 2 (SCATTER PLOT WITH CURE)

What It Shows

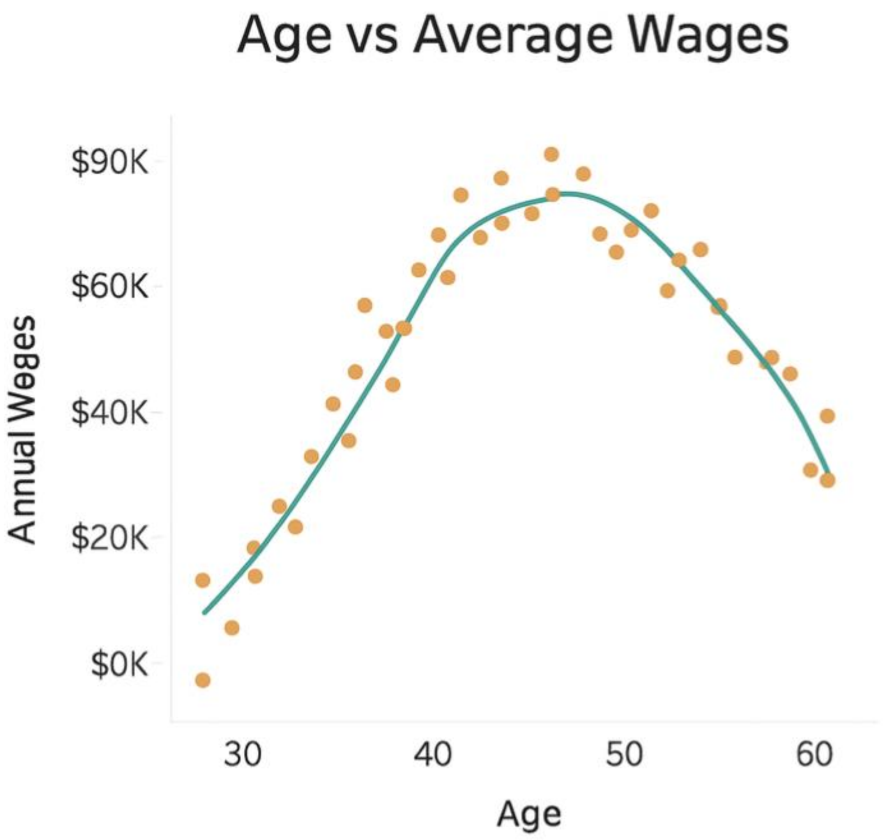
- A **scatter plot** with **orange dots** representing individual data points (age vs wage).
- A **green fitted curve** shows the overall trend in wages as age increases.
- X-axis = **Age** (from 30 to 60), Y-axis = **Annual Wages** (from $0K to $90K).

How to Interpret It

- Wages **increase steadily** from age 30 to around age 45.
- After 45, wages **plateau or slightly decline**, suggesting a peak earning age.
- The curve helps visualize the **average wage trajectory** across age groups.

Key Insights

- This graph reveals the **relationship between age and income**.
- It suggests that **mid-career individuals (around 45)** tend to earn the most.
- Useful for analyzing **career progression**, **income inequality**, or **retirement planning trends**.



CONCLUSION

- This project showed how shared tools and methods can analyze very different datasets
- Python allowed us to load data, clean it, merge datasets, calculate new variables, and build visualizations
- Tableau provided additional visuals for deeper interpretation and clearer comparisons
- Despite using different topics, both datasets followed the same process:
  load → clean → transform → visualize → interpret
- The six Python plots and two Tableau charts highlighted major trends, relationships, and patterns
- Completing this project strengthened our understanding of real-world data workflows
- The skills demonstrated—documentation, reproducible code, and visual communication—prepare us for future academic and professional work

REFERENCES

- Centers for Disease Control and Prevention. *COVID-19 Vaccinations in the United States, County.*
- Maryland Open Data Portal. *MD COVID-19 – Cases by County.*
- Somerville, T. (2024). *Analyzing the Impact of COVID-19 Vaccination Rates on Case Trends in Maryland.* CTEC 128 Data Science Project.
- U.S. Census Bureau. (2023). *American Community Survey: Public Use Microdata Sample (PUMS)*. Retrieved from https://www.census.gov/programs-surveys/acs/microdata/access.html Census.gov+1
- U.S. Bureau of Labor Statistics. (2021). *Labor force characteristics by race and ethnicity* (Table 7 & Table 18). Retrieved from https://www.bls.gov/opub/reports/race-and-ethnicity/2021/home.htm

THANK YOU