INDUSTRIAL TRAINING REPORT

on

PROJECT- "ADULT DATA SET VISUALIZATION USING ADVANCED PYTHON"

at

NATIONAL INSTITUTE FOR INDUSTRIAL TRAINING

Submitted in partial fulfilment of degree of

Bachelors of Technology In Computer Science And Engineering

Submitted to :

Submitted By:- **Ritabrita Karmakar(14200122118)**

TEAM MEMBERS:

Ritabrita Karmakar(14200122118)
Atrayee Dutta (14200122113)
Barna Das(14200122121)
Shayanika Das(14200122132)

Meghand Saha Institute of Technology
Nazirabad, Ucchepota, Kolkata-700150

CONTENTS

DECLARATION

I, **Ritabrita Karmakar**, a student pursuing Bachelor of Technology in Computer Science and Engineering, in **Meghnad Saha Institute of Technology** hereby declare that the project title, "**ADULT DATA SET VISUALIZATION USING ADVANCED PYTHON** " is the result of my own independent work. The project has been undertaken as a part of my academic curriculum, and all the information, methodologies, and findings presented in this project are authentic and original.

Date: 08/02/2024
RITABRITA KARMAKAR
University_Roll_Number.:- 14200122118

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my teachers who gave me the opportunity to do this wonderful project on the topic "**ADULT DATA SET VISUALIZATION USING ADVANCED PYTHON**", which also helped me in doing a lot of research and I came to know about so many new things, I am really thankful to them.
Secondly, I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited timeframe.

Abstract

Data visualization plays a crucial role in extracting meaningful insights from large and complex datasets. This study explores the application of various data visualization techniques on a specific dataset to enhance understanding and facilitate decision-making. The chosen dataset comprises [describe the dataset, including its nature and source]. The primary objectives of this research are to identify patterns, trends, and relationships within the data, as well as to communicate findings effectively to a diverse audience.

The methodology involves the selection and implementation of appropriate data visualization tools and techniques, such as [mention specific tools, like charts, graphs, or interactive dashboards]. The visualizations aim to uncover hidden patterns, outliers, and correlations, providing a comprehensive overview of the dataset. Through the iterative process of visualization design and refinement, the study aims to strike a balance between simplicity and informativeness to ensure clarity and accessibility for a wide range of stakeholders.

The outcomes of this research are expected to contribute to a deeper understanding of the underlying structures within the dataset, enabling informed decision-making and actionable insights. The significance of effective data visualization in conveying complex information is emphasized throughout the study, highlighting its role in transforming raw data into valuable knowledge. Ultimately, this research aims to demonstrate the power of visual representations in revealing insights that might be obscured in raw data, thus fostering a greater appreciation for the importance of data visualization in various fields.

INTRODUCTION

Python offers several powerful libraries for data visualization. One of the most popular libraries for this purpose is Matplotlib, which provides a wide range of plotting functions. Additionally, Seaborn is built on top of Matplotlib and offers a higher-level interface for creating attractive statistical graphics.

Here's a simple example of how to create a basic plot using Matplotlib:

import matplotlib.pyplot as **plt**

```
# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# Create a line plot
plt.plot(x, y)

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')

# Show the plot
plt.show()
```

Datasets from well-known libraries in Python for data analysis and visualization:

Tips dataset:

- **Source:** The tips dataset is part of the Seaborn library's built-in datasets. It's a synthetic dataset created for demonstration purposes.
- **Type of data:** The dataset contains information about tips collected in a restaurant, including the total bill, the tip amount, gender, day of the week, time of day, the size of the dining party, and whether the customer is a smoker or not.

Iris dataset (a common dataset often used for machine learning and data analysis tutorials):

- **Source:** The Iris dataset is also included in several Python libraries, including Seaborn and Scikit-learn.
- **Type of data**: This dataset contains information about iris flowers, including the sepal length, sepal width, petal length, petal width, and species (which could be one of three types: setosa, versicolor, or virginica).

Python Libraries Used:

- **Matplotlib**: A widely-used plotting library in Python that provides a MATLAB-like interface for creating plots.
- **Seaborn:** Built on top of Matplotlib, Seaborn is another powerful library for statistical data visualization. It provides a higher-level interface for creating attractive and informative statistical graphics.
- **Pandas:** Though not explicitly used in the examples, Pandas is often used for data manipulation and analysis. It provides data structures and functions for working with structured data, making it a crucial tool in data visualization workflows.

Before visualizing data in Python, it's often necessary to perform data cleaning and preprocessing to ensure that the data is in a suitable format and quality for visualization. Some common data cleaning and preprocessing steps include:

Handling missing values: Check for missing values in the dataset and decide how to handle them, whether by imputation (replacing missing values with a calculated value) or removal.
Removing duplicates: Check for and remove duplicate records from the dataset.
Data type conversion: Convert data types as needed. For example, convert strings to numerical values if necessary for plotting**.**
Normalization or scaling: If the data spans different ranges, normalize or scale the data to bring it to a common scale, which can be particularly important for certain types of visualizations like clustering or classification.
Feature engineering: Create new features or modify existing ones to better represent the underlying patterns in the data**.**
Handling outliers: Decide how to handle outliers, whether by removing them, transforming them, or leaving them as they are, depending on the specific context.

Encoding categorical variables: Convert categorical variables into a numerical format suitable for visualization or analysis.

Regarding types of visualizations, there are many different types, each suited to different types of data and analysis goals. Some common types of visualizations include:

Line plots: Suitable for showing trends over time or across categories.
Scatter plots: Useful for visualizing the relationship between two numerical variables.
Histograms: Display the distribution of a single numerical variable.
Bar plots: Compare quantities across different categories**.**
Pie charts: Show the composition of a categorical variable as a proportion of a whole.
Box plots: Visualize the distribution of a numerical variable and highlight potential outliers**.**
Heatmaps: Display a matrix of data as colors, useful for visualizing correlations or patterns in large datasets**.**
Violin plots: Combine aspects of box plots and kernel density plots to show the distribution of data.
Pair plots: Show pairwise relationships between variables in a dataset, useful for identifying patterns and correlations.

These are just a few examples, and there are many more types of visualizations available in Python libraries like Matplotlib, Seaborn, Plotly, and Pandas. The choice of visualization depends on the nature of the data and the specific insights you want to gain from it.

SYSTEM ANALYSIS

Advanced Language Features:

- **Decorators:** Study how to use decorators to modify the behavior of functions or methods.
- **Generators and Iterators:** Explore the concept of lazy evaluation and efficient iteration techniques.
- **Metaprogramming:** Learn about metaclasses, dynamic class creation, and modifying classes at runtime**.**

Advanced Data Structures and Algorithms:

- **Collections Module:** Study advanced data structures provided by Python's collections module, such as Counter, OrderedDict, and deque.
- **Algorithm Design and Analysis**: Learn about algorithmic complexity, sorting algorithms, searching algorithms, and optimization techniques.

Advanced Libraries and Frameworks:

- **Scientific Computing:** Explore libraries like NumPy,matplotlib and Pandas for advanced data manipulation, analysis, and scientific computing.
- **Machine Learning and AI:** Study libraries like Scikit-learn, TensorFlow, and PyTorch for machine learning, deep learning, and artificial intelligence applications.
- **Web Development:** Dive into web frameworks like Django and Flask for building scalable web applications**.**

Testing and Debugging:

- **Unit Testing:** Learn about Python's built-in unittest framework and third-party testing libraries like pytest**.**
- **Debugging Techniques:** Explore debugging tools and techniques for troubleshooting Python code, including the pdb debugger and IDE integrations.

on: Study containerization technologies like Docker for packaging and deploying Python applications.

Deployment and DevOps:

- **Orchestration:** Learn about orchestration tools like Kubernetes for managing containerized applications in production**.**
- **Continuous Integration and Deployment (CI/CD):** Explore CI/CD pipelines for automating testing, building, and deploying Python applications.

Advanced Python skills can open up opportunities in various fields, including software development, data science, machine learning, web development, and system administration.

PROJECT DETAILS

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

We've imported the necessary libraries for data visualization in Python. Seaborn (`sns`) and Matplotlib (`plt`) are commonly used together for creating visually appealing plots, while NumPy (`np`) and Pandas (`pd`) are essential for data manipulation and analysis.

```
d3 = df.dropna()
d3
```

The code `df.dropna()` is used to remove any rows with missing values from the DataFrame `df`. This operation creates a new DataFrame without the rows containing NaN (missing) values.

```
df['Workclass'].sort_values()
```

To sort the values in the 'Workclass' column of your DataFrame `df`, you can use the `sort_values()` method.

```
df.iloc[6:32]
```

The code `df.iloc[6:32]` is used to select rows from index 6 to index 31 in the DataFrame `df`.

```
df['gain+loss']= df['Capital Gain']+ df['Capital Loss']
df
```

The code `df['gain+loss'] = df['Capital Gain'] + df['Capital Loss']` **creates a new column in the DataFrame** `df` **named** `'gain+loss'`**, which is the result of adding the values from the columns** `'Capital Gain'` **and** `'Capital Loss'`**.**

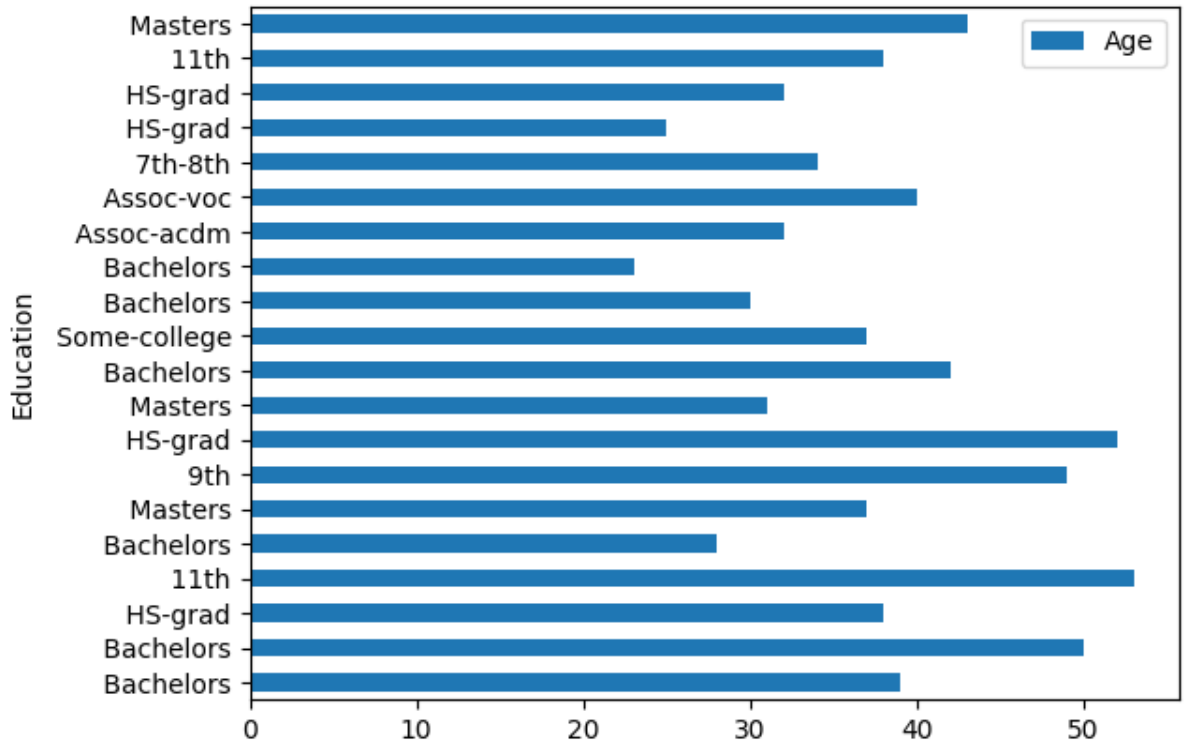
```
c=df.ind[df.ind['Hours/Week']==40]
```

```
c
```

To filter rows from the DataFrame `df` **where the value in the column** `'Hours/Week'` **is equal to 40.**

```
d1.plot(kind='barh',x='Education',y='Age')
```

```
plt.show()
```



This code is creating a horizontal bar plot using the DataFrame `d1`**, where** `'Education'` **is plotted on the y-axis and** `'Age'` **is plotted on the x-axis.**

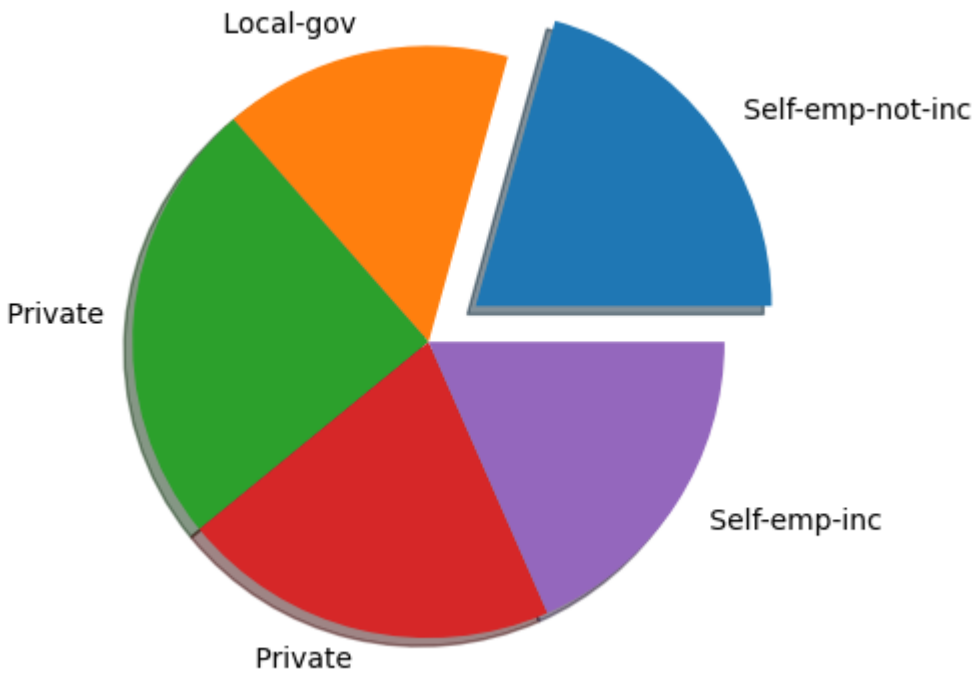
```
d=df.ind[df.ind['Age']>=40]
```

```
d
```

To filter rows from the DataFrame `df` where the value in the column `'Age'`

- ```
  x = e['Age']
  ```

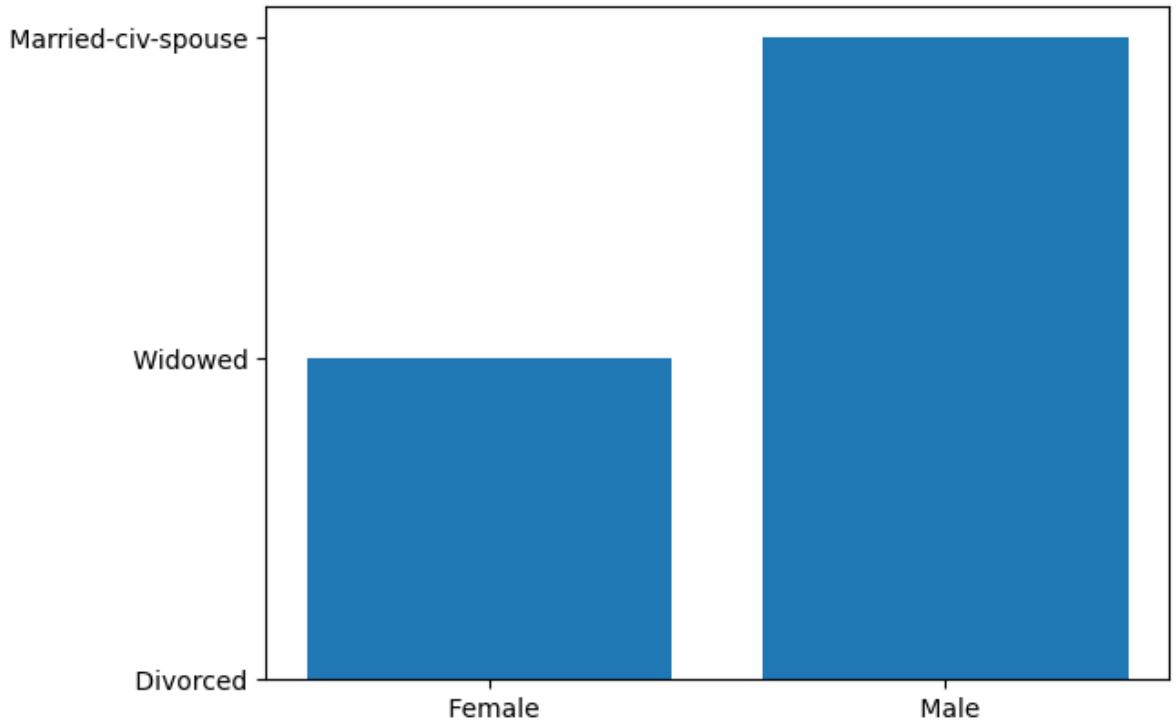
```
y = e['Workclass']

plt.pie(x,labels= y,startangle=0,explode=[0.2,0,0,0,0],shadow=
True)

plt.show()
```



This will generate a pie chart where each slice represents an age group (`'Age'`) corresponding to different work classes (`'Workclass'`). The explosion parameter (`explode`) is used to highlight a particular slice by pulling it out from the center, and the shadow parameter (`shadow`) adds a shadow effect to the pie chart.

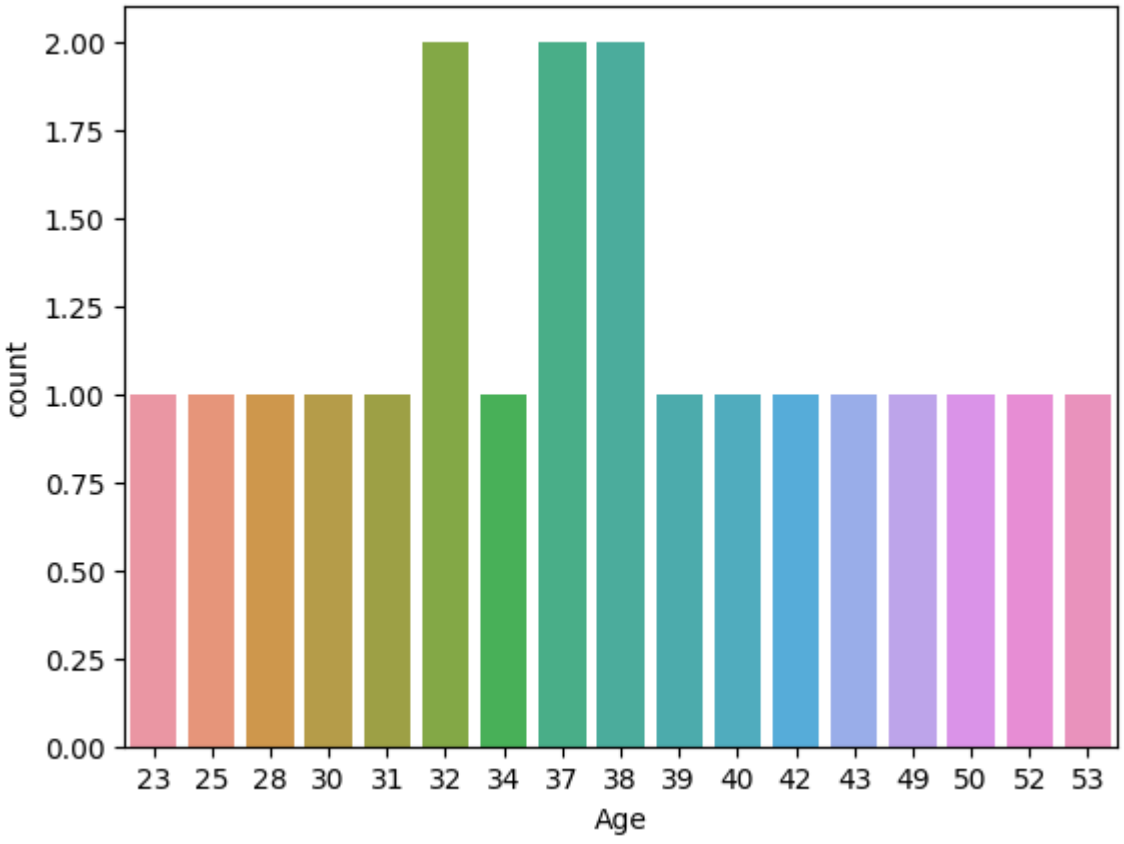
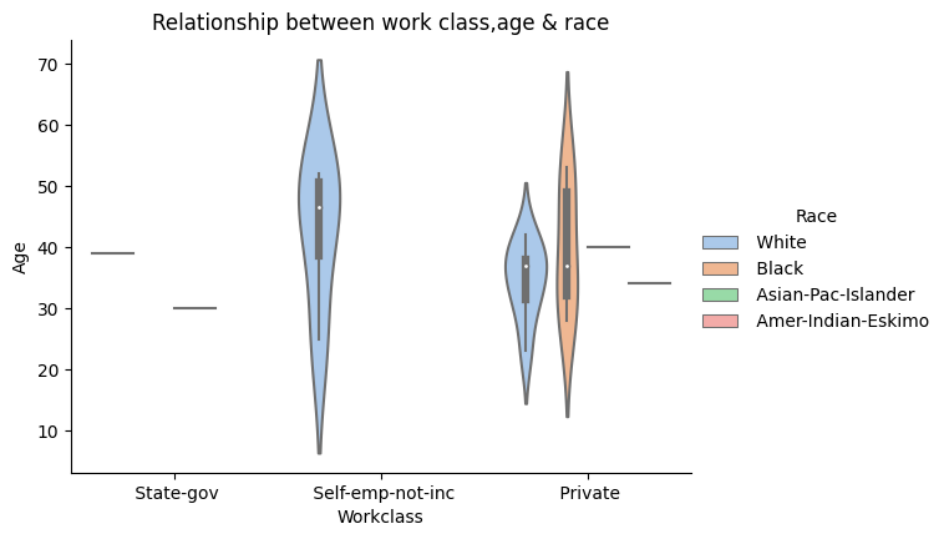
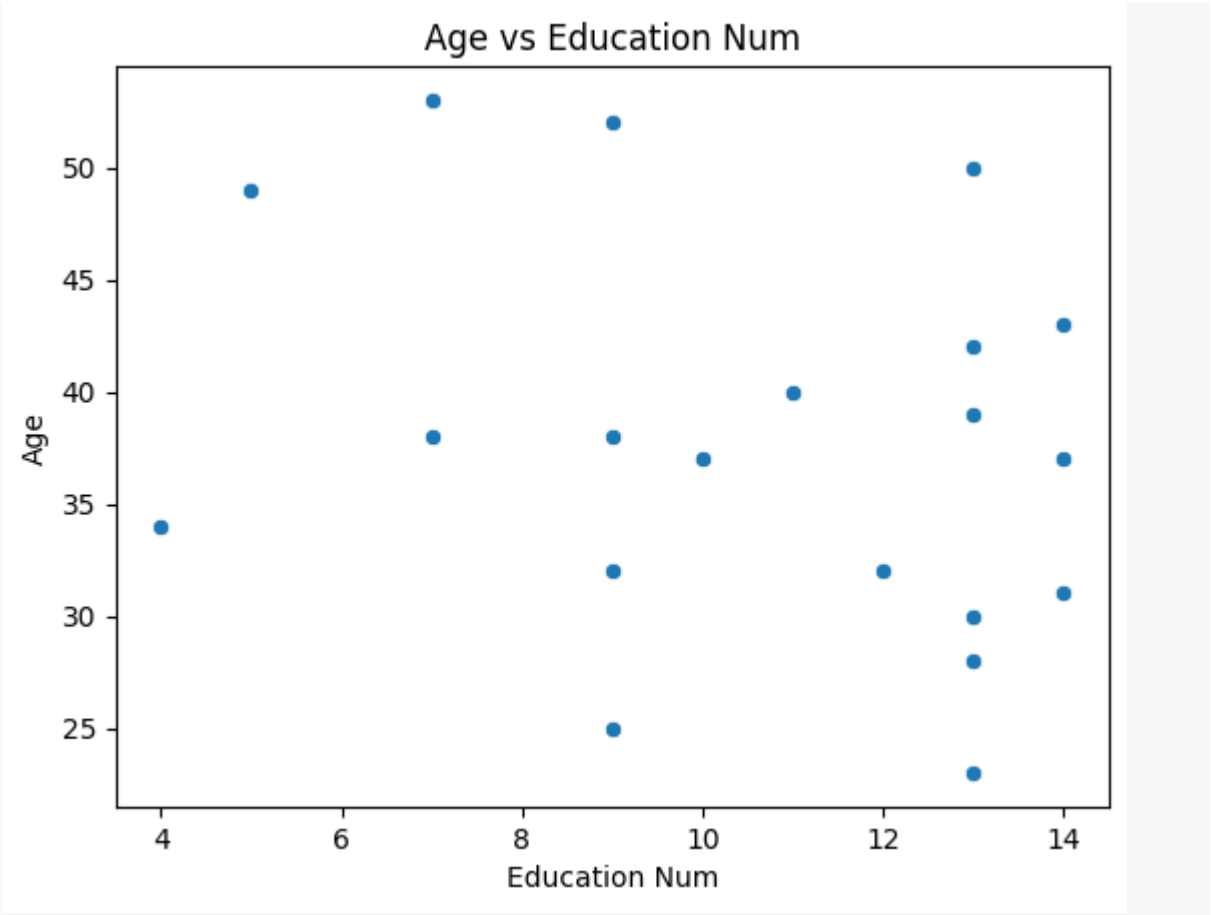
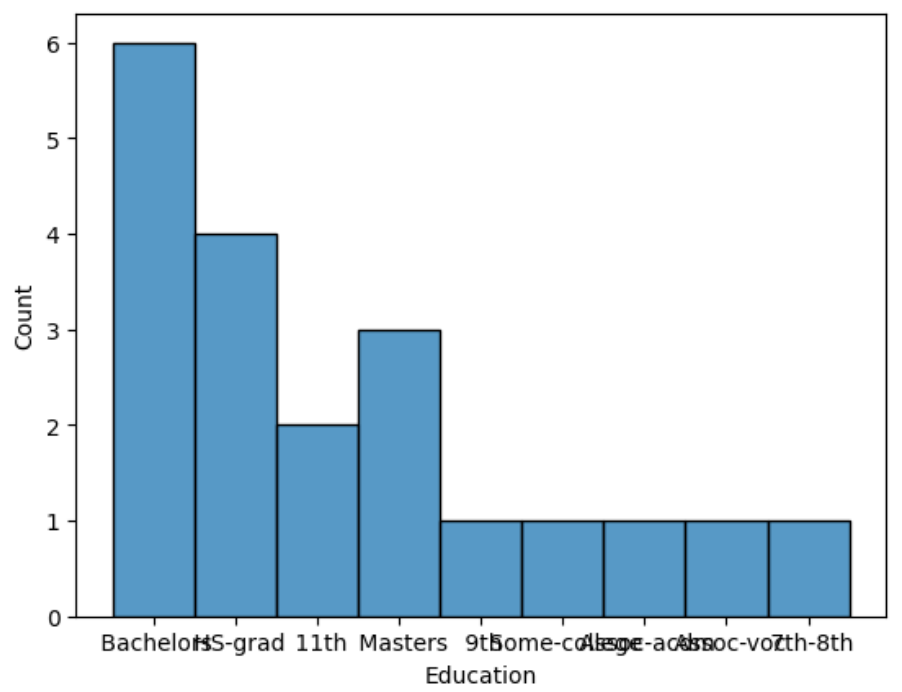
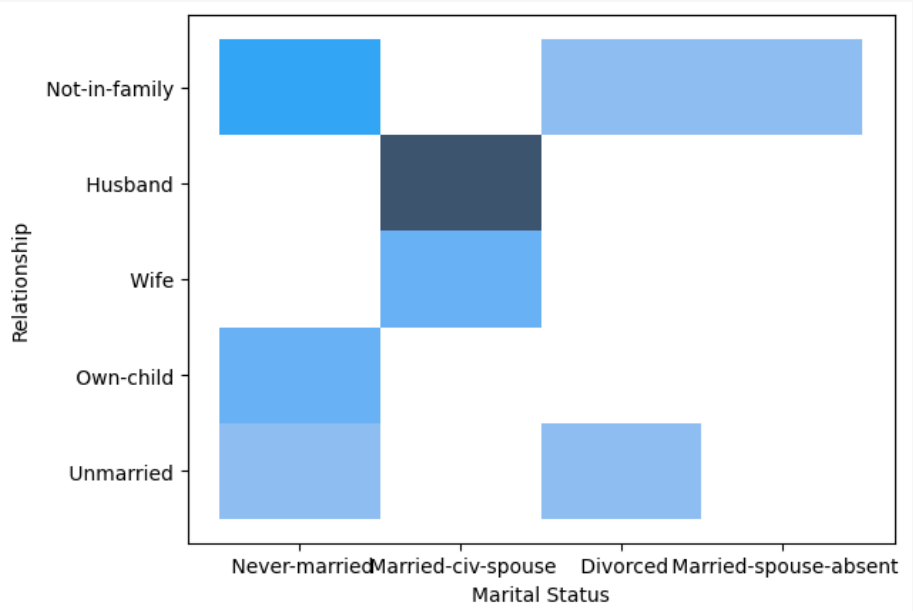
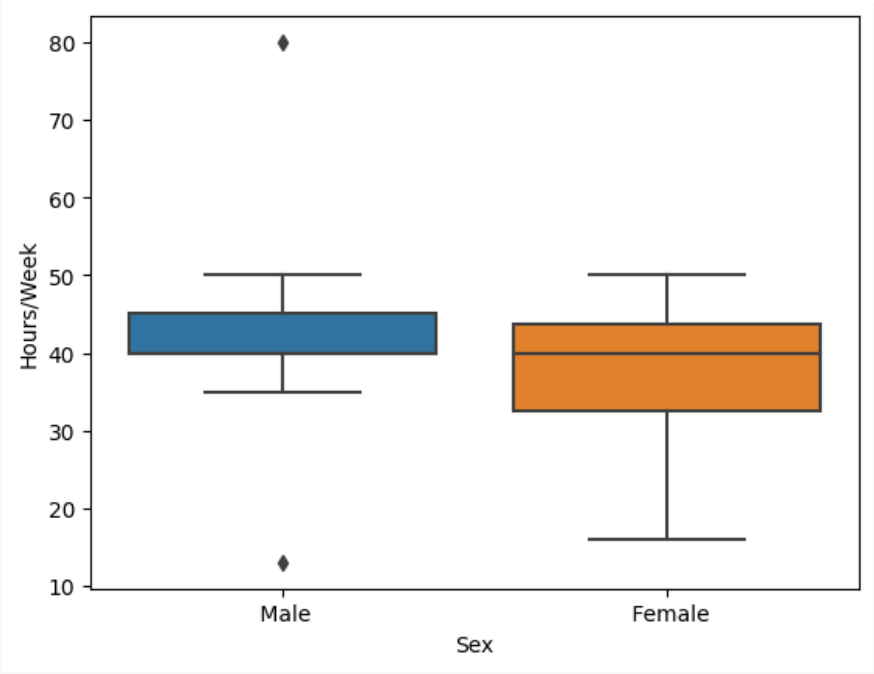
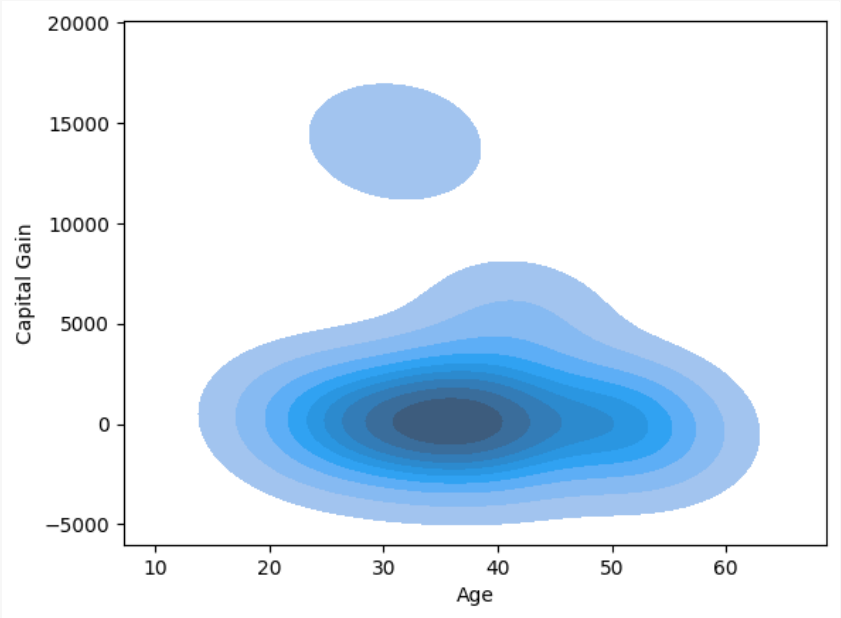
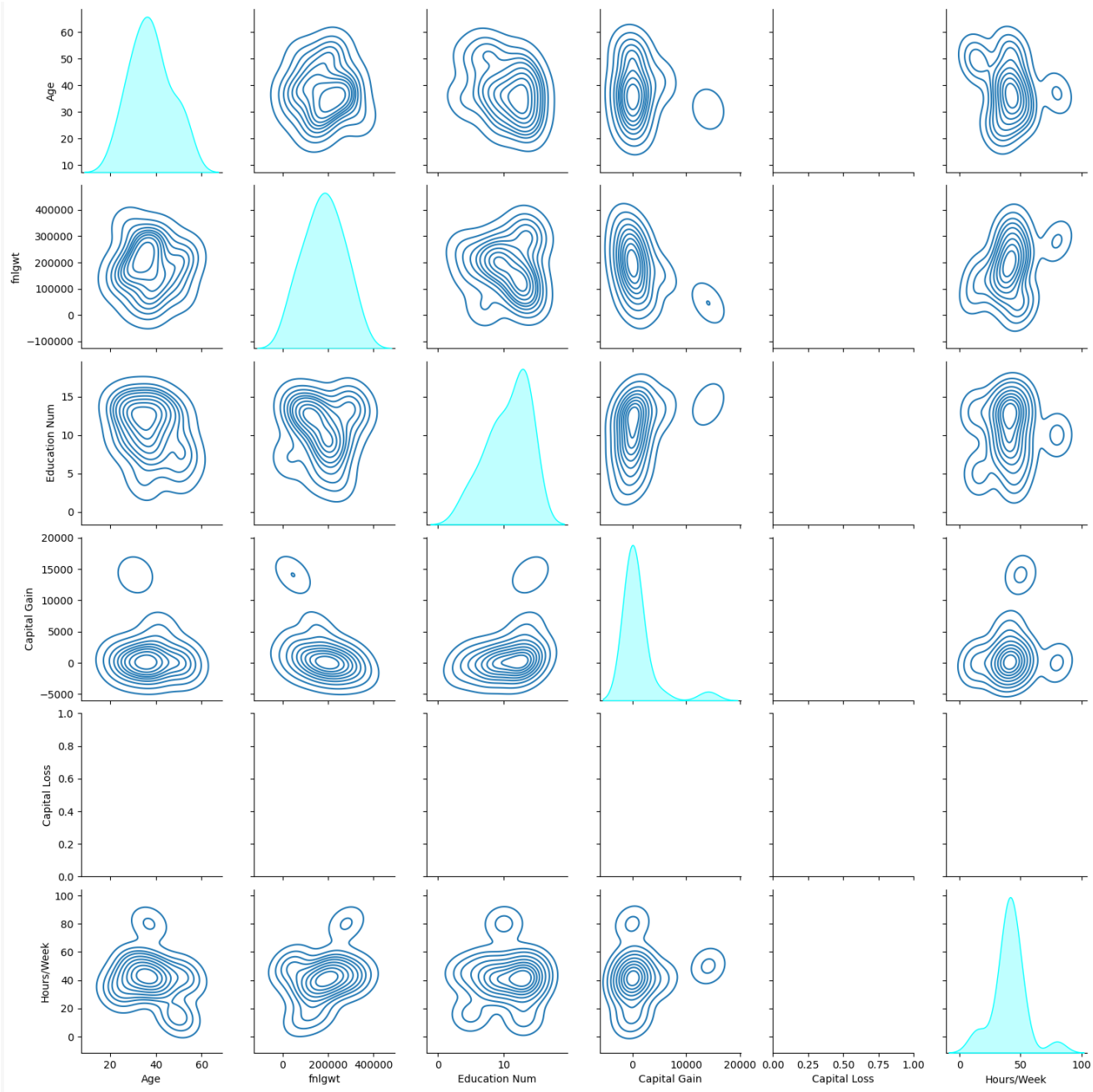
- ```
  x = f['Sex']
  ```

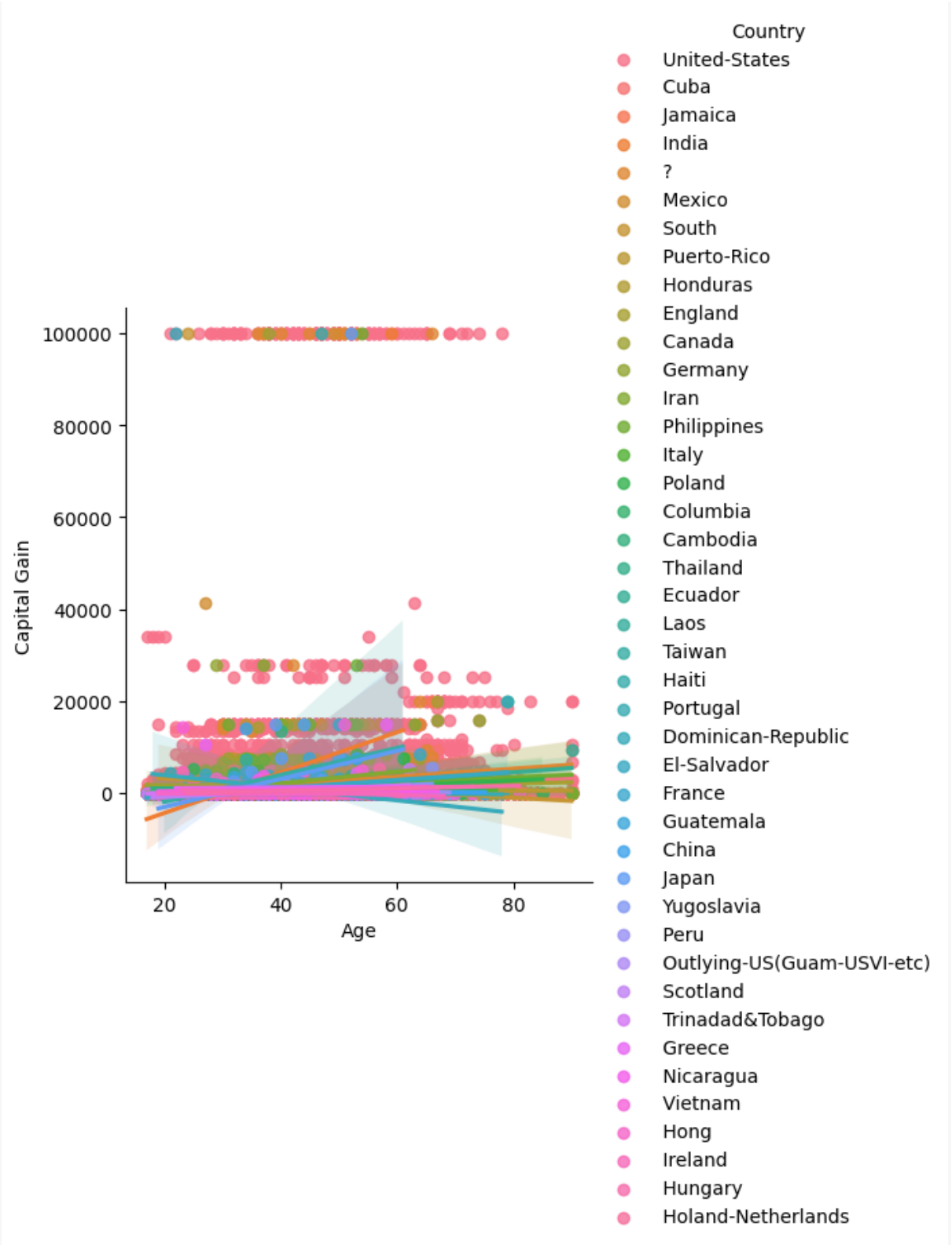
```
y = f['Marital Status']
plt.bar(x,y)
```



This code will produce a bar plot where each bar represents the count of each 'Marital Status' category for each 'Sex'.

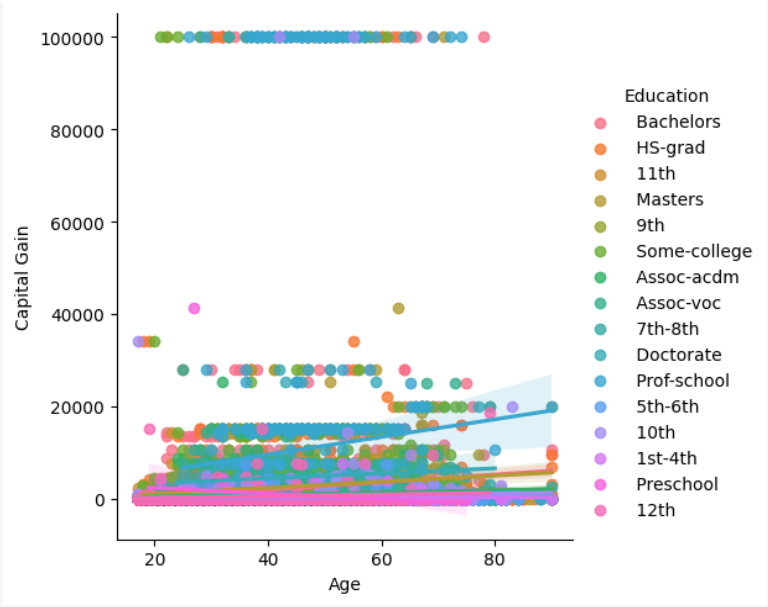
- ```
  sns.countplot(x='Age', data=d1)
  plt.show()
  ```


  ```

This code will generate a count plot where each bar represents the count of occurrences for each age group (`'Age'`).

```
sns.catplot(x="Workclass", y="Age", hue="Race", kind="violin",
                palette="pastel", data=d1, height=4.2, aspect=1.4)
plt.title("Relationship between work class,age & race")
```



This code will generate a violin plot showing the relationship between 'Workclass', 'Age', and 'Race'.

```
student_data = d1
student_df = pd.DataFrame(student_data)
sns.scatterplot(data =student_data , x = 'Education Num',y =
'Age')
plt.title('Age vs Education Num')
plt.xlabel("Education Num", fontsize= 10)
plt.ylabel("Age", fontsize= 10)
```



This code will generate a scatter plot showing the relationship between 'Education Num' and 'Age'.

```
sns.histplot(data=student_df, x='Education')
plt.show()
```



This code will generate a count plot showing the distribution of education levels.

- `sns.histplot(data=student_df, x='Marital Status', y='Relationship')`
  `plt.show()`



The `sns.histplot()` function is typically used to create histograms.

- `sns.boxplot(data=student_df,x='Sex', y='Hours/Week')`

`plt.show()`

- `sns.boxplot(data=student_df,x='Sex', y='Hours/Week')`
  `plt.show()`



This code will generate a box plot showing the distribution of 'Hours/Week' for each category of 'Sex'.

- ```
  sns.kdeplot(data=student_df, x='Age', y='Capital Gain',
  fill=True)
  plt.show()
  ```



This will generate a two-dimensional KDE plot showing the estimated density of points in the space defined by 'Age' and 'Capital Gain'.

- ```
  sns.pairplot(data =student_df,kind="kde", diag_kws=
  {'color':'cyan'})
  ```



This will generate a pairplot where each variable is plotted against every other variable in the DataFrame, and the diagonal plots show the kernel density estimates for each variable.

- ```
  sns.lmplot(data=df, x='Age', y='Capital Gain', hue='Country')
  plt.show()
  ```
  ```



This will generate a scatter plot with regression lines for 'Age' vs. 'Capital Gain', with each line representing a different category of 'Country'.

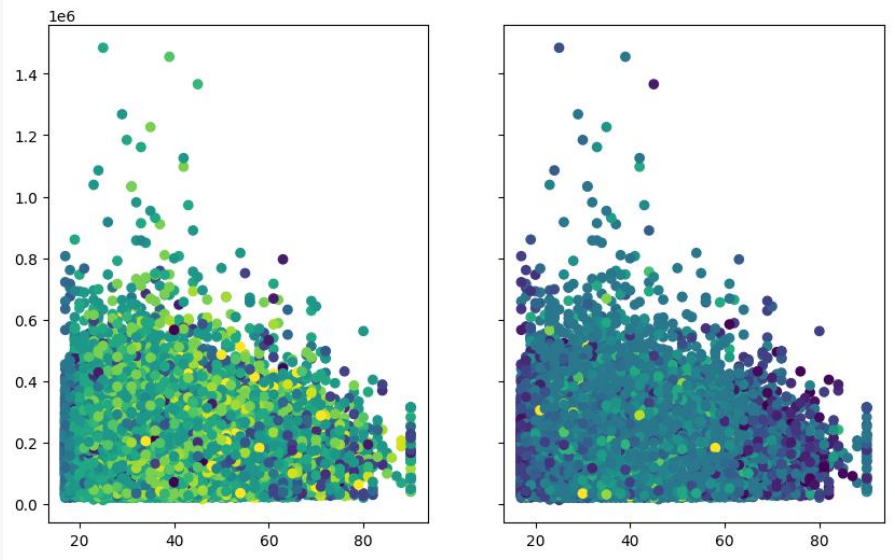
```
sns.lmplot(data=df, x='Age', y='Capital Gain', hue='Education')
plt.show()
```



This code will generate a scatter plot with regression lines, where each line represents a different category of 'Education'.

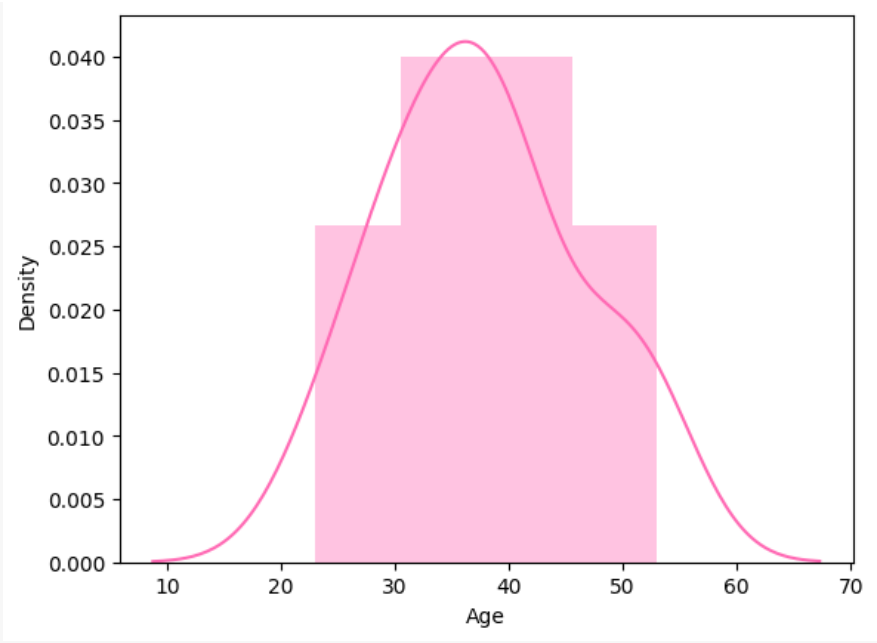
- `f,(ax1,ax2)=plt.subplots(nrows=1,ncols=2,sharey=True,figsize=(10, 6))`

```
ax1.scatter(x=df['Age'],y=df['fnlgwt'],c=df['Education Num'])

ax2.scatter(x=df['Age'],y=df['fnlgwt'],c=df['Hours/Week'])
```



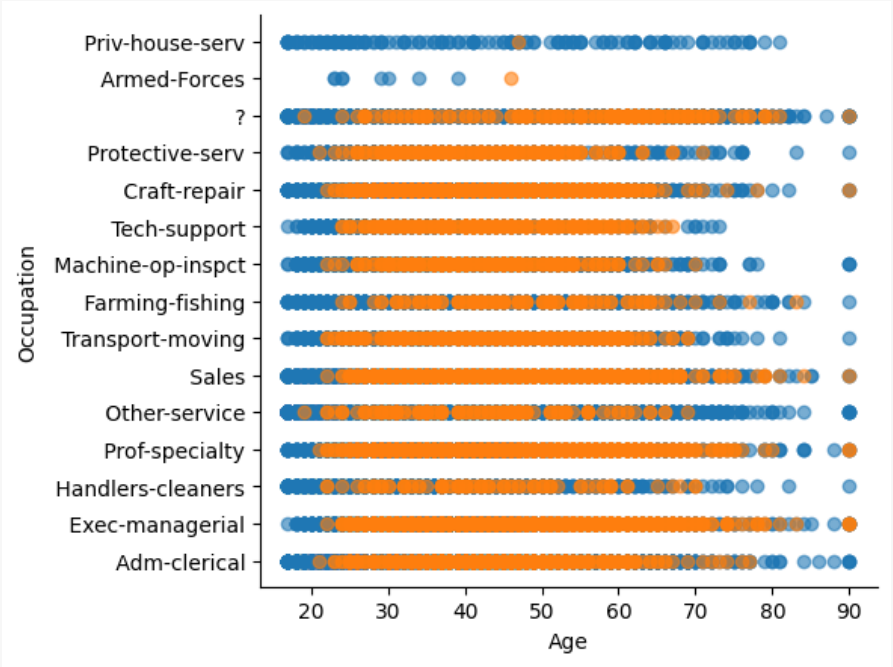
This will create a figure with two subplots, each containing a scatter plot of 'Age' vs. 'fnlgwt', with the points colored by 'Education Num' in the first subplot and by 'Hours/Week' in the second subplot.

- `sns.distplot(d1['Age'] , color ='hotpink')`



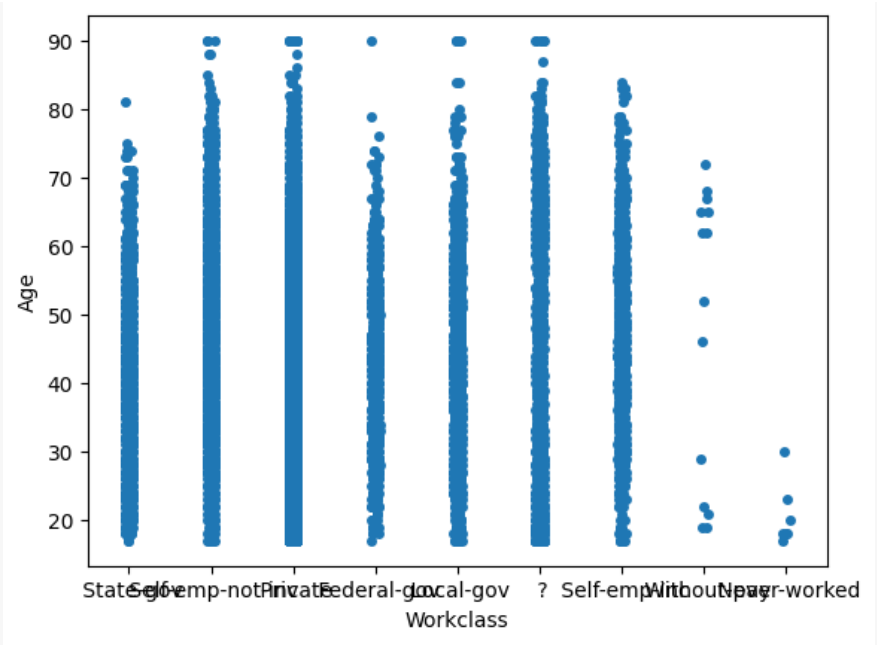
This code will generate a distribution plot for the 'Age' variable, with the histogram and KDE plot overlaid.

- ```
  g = sns.PairGrid(df, x_vars=["Age"], y_vars=["Occupation"],
  height=4.5, hue="Above/Below 50k", aspect=1.1)
  ax = g.map(plt.scatter, alpha=0.6)
  ```



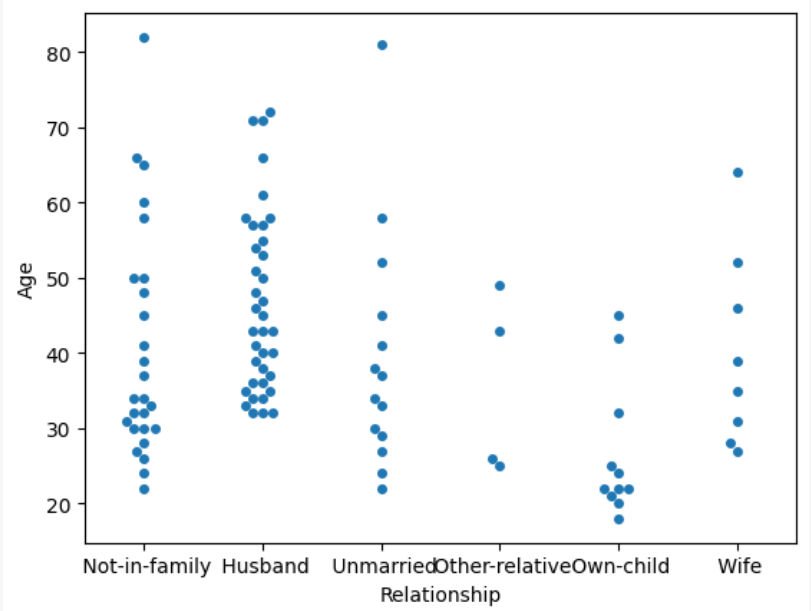
This code will generate a PairGrid with a scatter plot of 'Age' against 'Occupation', where points are colored by the 'Above/Below 50k' variable.

- ```
  res = sns.stripplot(x="Workclass", y="Age", data=df,jitter=0.05)
  ```



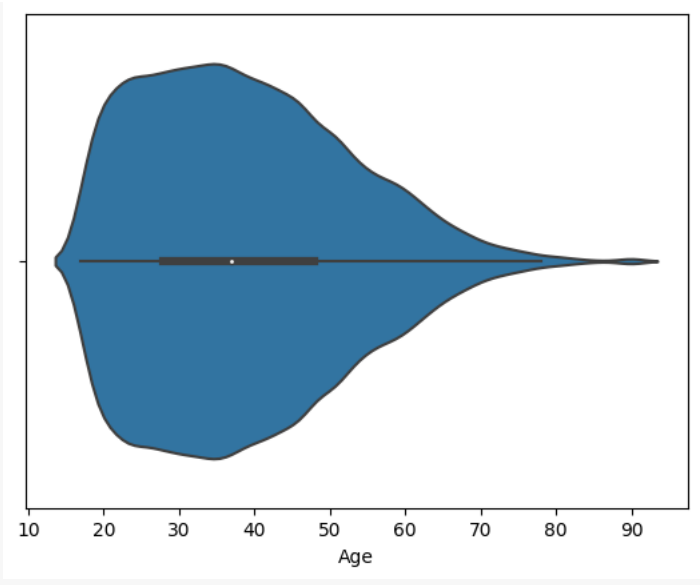
This code will generate a strip plot with 'Workclass' on the x-axis and 'Age' on the y-axis. The `jitter` parameter adds a small amount of random noise to the categorical positions to better represent the distribution of values.

- `sns.swarmplot(x="Relationship", y="Age", data=d2)`



This code will generate a swarm plot with 'Relationship' on the x-axis and 'Age' on the y-axis. Each point represents an individual observation in the dataset.

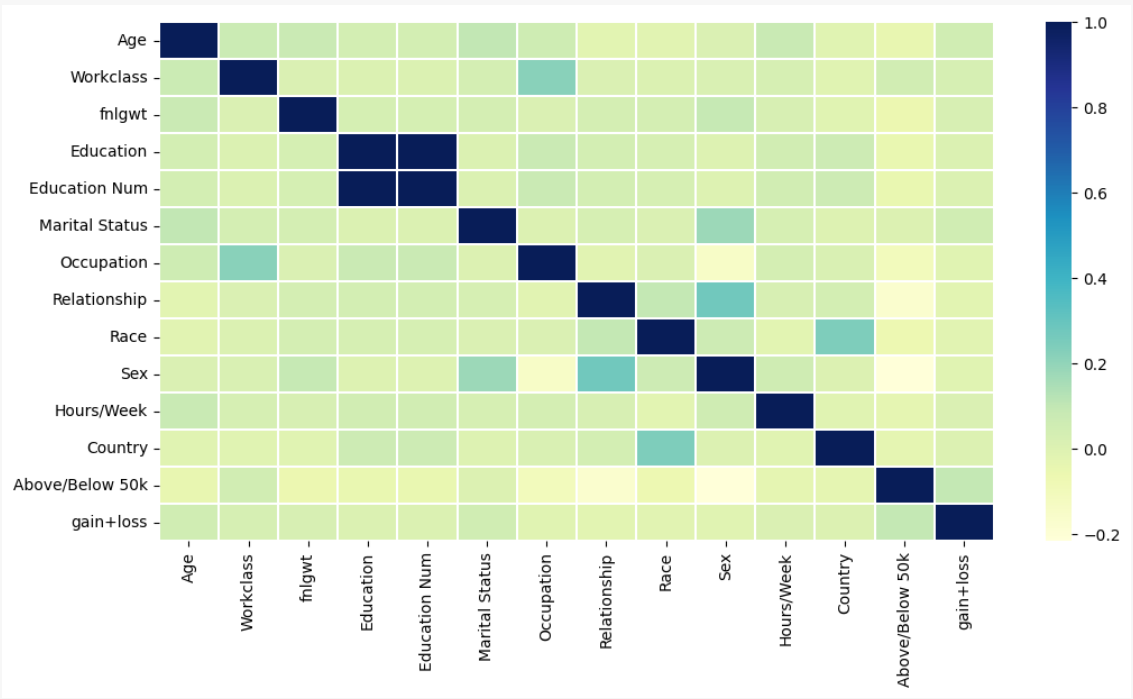
- `sns.violinplot(x=df['Age'])`



This code will generate a violin plot of the distribution of the 'Age' variable.

- `plt.figure(figsize=(12, 6))`

```
df.drop(['Capital Gain', 'Capital Loss'],axis=1, inplace=True)
corr = df.apply(lambda x: pd.factorize(x)[0]).corr()
ax = sns.heatmap(corr, xticklabels=corr.columns,
yticklabels=corr.columns,
                 linewidths=.2, cmap="YlGnBu")
```

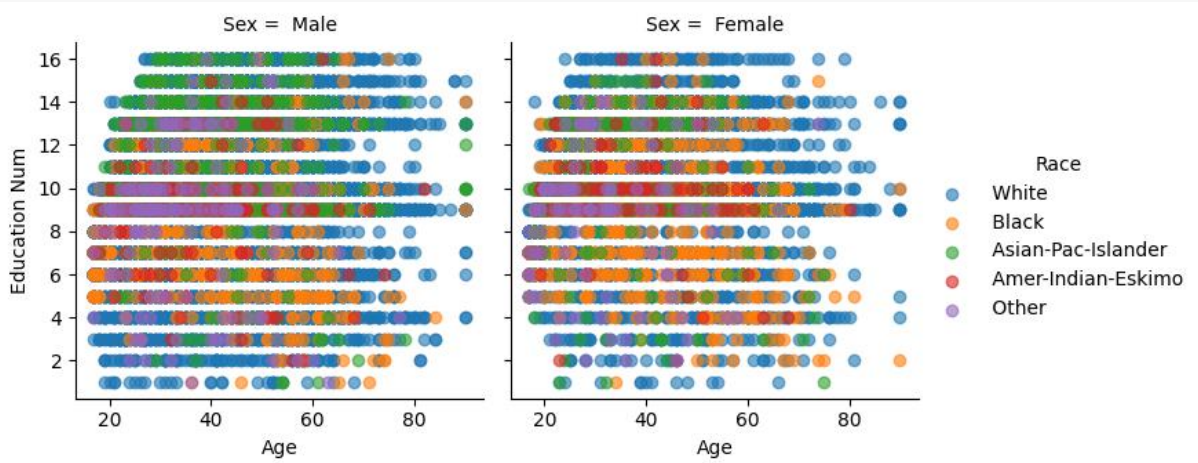


This code will generate a heatmap visualizing the correlation matrix of the remaining variables in DataFrame `df`.

- `g = sns.FacetGrid(df, col="Sex", hue="Race", height=3.5)`

```
g.map(plt.scatter, "Age", "Education Num", alpha=0.6)
```

```
g.add_legend();
```



This code will generate a FacetGrid with two columns, one for each unique value of 'Sex'. Each facet contains a scatter plot of 'Age' against 'Education Num', with points colored by 'Race'.

CONCLUSION

In conclusion, using data visualisation to analyse an adult dataset has shown to be an effective and perceptive way to find patterns and trends in the data. We have been able to turn a complicated dataset into aesthetically appealing representations that improve our comprehension of important adult-related topics by utilising a variety of visualisation techniques.

The study's use of visualisations has effectively brought attention to potential outliers in the adult dataset, correlations between various factors, and demographic patterns. This helps to discover important characteristics and makes it easier to extract insights that academics and decision-makers may use.

The research's attention on privacy and ethical issues is noteworthy, especially in light of the sensitive nature of adult-related data. Exercise caution

REFERENCES

Colab Notebook link-

https://colab.research.google.com/drive/1iH41ZVrFcdaNLDsOMSuCPZoSlidRZ0ig?usp=sharing

Dataset link - adult.csv

BIBLIOGRAPHY

1. Matplotlib: John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55
2. Seaborn: Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, Chris Fonnesbeck, Antony Lee, Adel Qalieh. (2021). mwaskom/seaborn: v0.11.2 (January 2021) (v0.11.2). Zenodo. https://doi.org/10.5281/zenodo.592845
3. Pandas: McKinney, Wes. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)
4. Python Language Reference: Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.