Exploring Socioeconomic Factors in the Adult Population An Analysis of the Adult Dataset

December 24, 2023

1 Abstract

This project presents a comprehensive exploration of the "Adult Dataset", available on Kaggle, aiming to analyse and understand various socioeconomic factors influencing the annual income of the population.

The dataset encompasses diverse attributes, including demographic information, education levels, occupational details, race, gender, and income status. The research employs statistical and machine learning techniques to unveil patterns, trends, and correlations within the data.

The study begins by simulating and studying descriptive statistics of randomly generated data, and acquiring important insights from it. Data are analysed both in discrete and continuous domains. The study begins by preprocessing the dataset to handle missing values and ensure data quality. After ensuring data usability, several co-relations-finding methods and prediction models are developed and implemented through both mathematical approaches and machine learning methods.

The findings of this study contribute valuable insights into the socioeconomic dynamics of the adult population, providing a basis for informed policy decisions and targeted interventions. By leveraging advanced analytical techniques, this research aims to uncover hidden patterns and relationships within the data, fostering a deeper understanding of the factors shaping individuals' economic outcomes in the adult demographic.

2 Chapter 1

In the beginning, the project covers, detailed analysis of continuous as well as discretized random variables, which includes statistical analysis, central-limit theorem and its visualization, outliner detection, probability calculations, and other visualization techniques. All the above analysis is done on randomly generated data.

Moving forward, the project covers Markov chains simulation. Under the broader umbrella of Markov Chains, various topics like Transition Matrix Simulation, Recurrent Events, Ergodicity of Markov Chain Matrix, and Sensitivity Analysis have been done with appropriate Visualization.

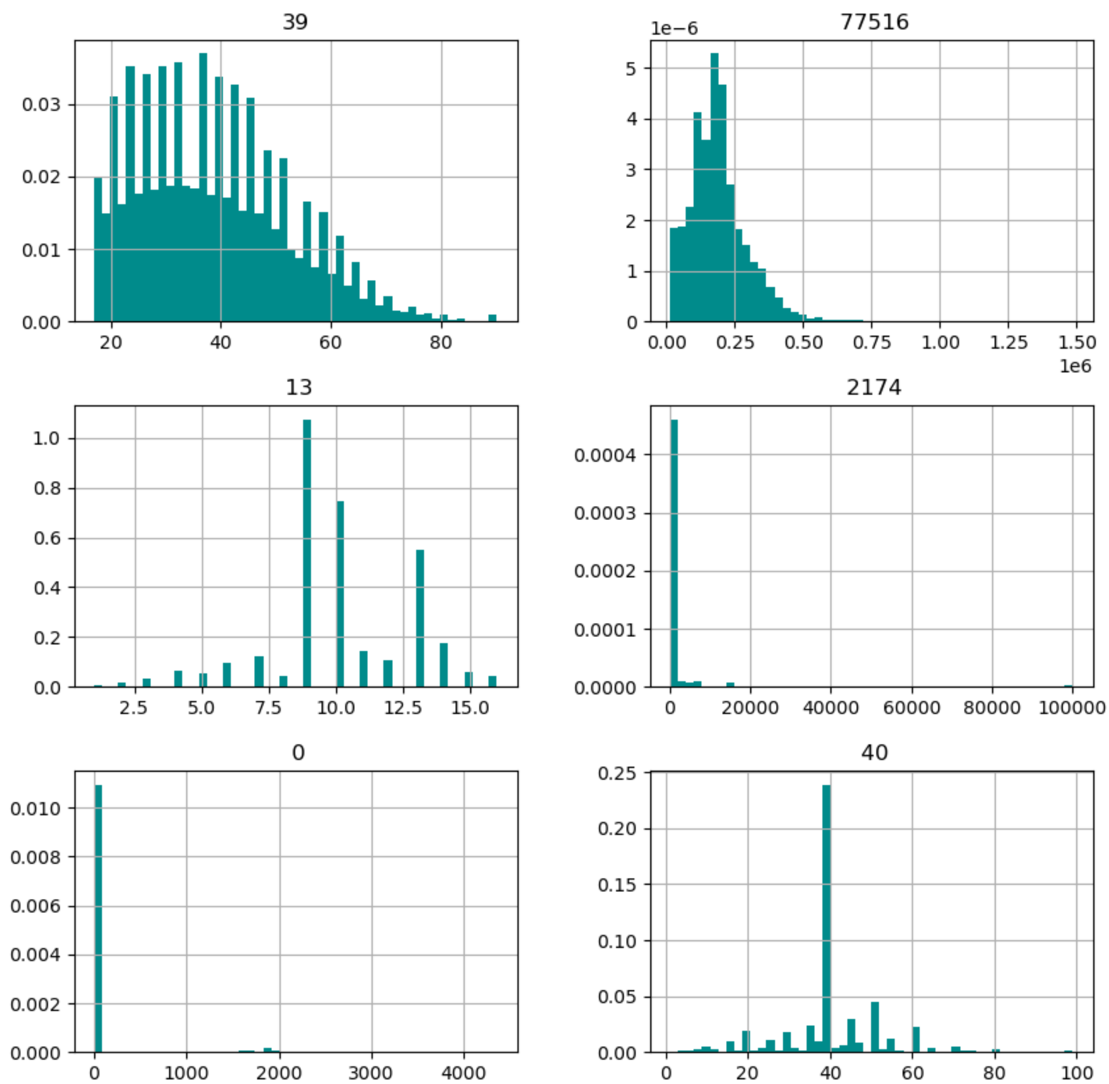
After completing the preliminaries, this project will move on to the Real Data Analysis part, in which we will be covering data analysis done on an Adult-Income dataset (https://www.kaggle.com/datasets/qizarafzaal/adult-dataset). A deep probabilistic analysis will be done on the dataset. The Bayesian inference will be applied to the filtered dataset, with both approaches, the analytical method as well as machine learning method. Joint Distribution Analysis will be further done on this dataset, which will include methods like A/B testing, correlation visualization, and normality testing to check whether the simulated data follows a normal distribution or not. Furthermore, the Kolmogorov-Smirnov test or Shapiro-Wilk test has also been implemented.

By this analysis, we will get to know how various socio-cultural, educational, and work backgrounds of a person determine the annual income of the person.

[1]:
```python
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('./adult.csv')

# Showing data head
data.head(5)
```

[1]:
```
   39          State-gov   77516  Bachelors  13         Never-married  \
0  50   Self-emp-not-inc   83311  Bachelors  13    Married-civ-spouse
1  38            Private  215646    HS-grad   9              Divorced
2  53            Private  234721       11th   7    Married-civ-spouse
3  28            Private  338409  Bachelors  13    Married-civ-spouse
4  37            Private  284582    Masters  14    Married-civ-spouse

        Adm-clerical  Not-in-family  White    Male  2174  0  40  \
0    Exec-managerial        Husband  White    Male     0  0  13
1  Handlers-cleaners  Not-in-family  White    Male     0  0  40
2  Handlers-cleaners        Husband  Black    Male     0  0  40
3     Prof-specialty           Wife  Black  Female     0  0  40
4    Exec-managerial           Wife  White  Female     0  0  40

    United-States   <=50K
0   United-States   <=50K
1   United-States   <=50K
2   United-States   <=50K
3            Cuba   <=50K
4   United-States   <=50K
```

[2]:
```python
# Histogram depecting the Expectation of number of specific peoples belonging to␣
↪particular race or gender, based on the Income
ax = data.hist(figsize=(10, 10), bins=50, xlabelsize=10, ylabelsize=10,␣
↪color='darkcyan', density=True, grid=True)
```



3 Chapter 2

"Adult dataset" has a total of 15 columns (features), and have 32560 rows (records). Columns like age, race, education, number of household members, marital status, sex and many other features have been recorded, to classify whether the given person has income greater than $50K or not.

Data useability of the given data is only 2.94, which is not much, for understanding the data and inferring from it.

Based on the dataset:

- Age Distribution:

The dataset represents a diverse range of ages, with individuals ranging from 17 to 90 years old. The average age is approximately 38.6 years, with a standard deviation of 13.6 years.

- Work Hours:

The average number of hours worked per week is around 40.4 hours, with a minimum of 1 hour and a maximum of 99 hours. The majority of individuals work standard full-time hours, as suggested by the median and upper quartile both being 40 hours.

- Educational Attainment:

The education level of individuals varies, with the dataset including people with education levels ranging from 1 (least educated) to 16 (most educated). The most common education level, represented by the mode, is not explicitly stated in the provided information.

- Occupations and Workplaces:

The dataset includes information on occupations such as 'Exec-managerial', 'Handlers-cleaners', 'Craft-repair', 'Armed-Forces', 'Tech-support', 'Farming-phishing', 'Transport-moving', 'Prof-speciality', 'Adm-clerical', and many unknown jobs.

- Marital Status and Socio-cultural fields:

Categorical variables like marital status, family size, race, and sex, are also taken into consideration. The distribution of occupations and demographic features could provide insights into the workforce composition.

- Income Levels:

The dataset seems to include an income variable represented by '<= 50K'. This suggests a binary classification where individuals earn less than or equal to \$ 50,000 or more than \$ 50,000. The count of individuals falling into each income category can provide an understanding of the income distribution in the dataset.

- Potential Data Quality Issues:

All columns seem to have non-null values, suggesting no missing data in the provided sample. It's important to further investigate and possibly clean column names, as they appear to have leading spaces (' 39', ' State-gov', etc.), which might lead to potential issues during analysis.

[3]:
```python
# Display basic information about the dataset
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   39               32560 non-null  int64
 1    State-gov       32560 non-null  object
 2    77516           32560 non-null  int64
 3    Bachelors       32560 non-null  object
 4    13              32560 non-null  int64
 5    Never-married   32560 non-null  object
 6    Adm-clerical    32560 non-null  object
 7    Not-in-family   32560 non-null  object
 8    White           32560 non-null  object
```

```
 9    Male           32560 non-null  object
 10   2174           32560 non-null  int64
 11   0              32560 non-null  int64
 12   40             32560 non-null  int64
 13   United-States  32560 non-null  object
 14   <=50K          32560 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
None
```

[4]:
```python
# Display summary statistics of numeric columns
print(data.describe())
```

```
                 39         77516            13          2174             0  \
count  32560.000000  3.256000e+04  32560.000000  32560.000000  32560.000000
mean      38.581634  1.897818e+05     10.080590   1077.615172     87.306511
std       13.640642  1.055498e+05      2.572709   7385.402999    402.966116
min       17.000000  1.228500e+04      1.000000      0.000000      0.000000
25%       28.000000  1.178315e+05      9.000000      0.000000      0.000000
50%       37.000000  1.783630e+05     10.000000      0.000000      0.000000
75%       48.000000  2.370545e+05     12.000000      0.000000      0.000000
max       90.000000  1.484705e+06     16.000000  99999.000000   4356.000000

                 40
count  32560.000000
mean      40.437469
std       12.347618
min        1.000000
25%       40.000000
50%       40.000000
75%       45.000000
max       99.000000
```

[5]:
```python
# Check for missing values
print(data.isnull().sum())
```

```
39                 0
 State-gov         0
 77516             0
 Bachelors         0
 13                0
 Never-married     0
 Adm-clerical      0
 Not-in-family     0
 White             0
 Male              0
 2174              0
 0                 0
 40                0
```

```
 United-States    0
 <=50K            0
dtype: int64
```

4 Chapter 3

4.0.1 Methodology for Statistical Analysis

Data Collection:

- Gather data $X = \{x_1, x_2, ..., x_n\}$ from Kaggle dataset.
- Dataset categorize the target variable income into two different class ('$<$50k','$>$50k'), based on dependent variables

  (age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income').

Tagret Variable:

- ['income']

Dependent Variable:

- [age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income']

Descriptive Statistics:

- Calculate mean

  $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$.
- Compute variance

  $Var(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{X})^2$.
- Determine standard deviation

  $SD(X) = \sqrt{Var(X)}$.
- Identify mode as the most frequently occurring value.
- Compute first quantile (Q_1) and third quantile (Q_3), inter-quartile range ($Q_3 - Q_1$).
- Calculating Skewness and Kurtosis.

Probability Distributions:

- Distinguish between discrete $P(X = x)$ and continuous $f(x)$ random variables.
- Define joint distributions $P(X = x, Y = y)$ to study relationships.

Conditional Probability and Expectation:

- Use conditional probability $P(A|B) = \frac{P(A \cap B)}{P(B)}$ and expectation $E(X|A)$.

Markov Chains:

- Define states $S = \{s_1, s_2, ..., s_n\}$ and transition matrix P.
- Analyze steady-state probabilities π and long-term behavior.

Sensitivity Analysis:

- Assess sensitivity using partial derivatives or alternate parameter values.
- Identify key variables impacting outcomes.

Simulation Techniques:

- Implement Monte Carlo simulations using random sampling techniques.
- Estimate probabilities and assess system behavior.

Bayesian Analysis:

- Bayes' rule:

  $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$.
- Incorporate prior knowledge $P(A)$ and likelihood $P(B|A)$ to obtain posterior $P(A|B)$.

Correlation:

- Calculate Pearson correlation coefficient:

  $r_{XY} = \frac{cov(X,Y)}{SD(X)SD(Y)}$.
- Assess significance with hypothesis testing.

Factor Analysis:

- Model observed variables X as linear combinations of latent factors F with loading matrix L: $X = LF + \epsilon$.
- Interpret factors through loadings and eigenvalues.

Validation and Interpretation:

- Interpret results in the context of the problem domain.

5 Chapter 4

5.1 Importing Libraries

[6]:
```python
# Importing Libraries
import math
import random
import itertools
```

```python
import matplotlib
import statistics
import numpy as np
import networkx as nx
import scipy.stats as st
import category_encoders as ce
from collections import Counter
from mpl_toolkits.mplot3d import Axes3D
from factor_analyzer import FactorAnalyzer
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import CategoricalNB
from pandas import read_csv, Series, DataFrame
from scipy.linalg import fractional_matrix_power
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import FactorAnalysis
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from factor_analyzer.factor_analyzer import calculate_kmo,␣
 ↪calculate_bartlett_sphericity
```

5.2 Generating Continous Data

[7]:
```python
# Generating random data based on mean and standard deviation
mu, sigma, n = 0, 1, 1000
data = np.random.normal(mu, sigma, n)
np.set_printoptions(threshold=999)
print(data)
```

```
[ 1.96090017 -0.18366947  0.54789462 ...  1.25190234  0.71866899
  0.46316718]
```

5.3 Statistical Analysis

[8]:
```python
# Calculating Mean, Standard Deviation & Variance
mean, standard_deviation, variance = data.mean(), data.std(), data.std()**2
mode = st.mode(data, keepdims = True)
print('\
        Mean : {:.4f}\n\
        Standard Deviation : {:.3f}\n\
        Variance : {:.3f}'\
        .format(mean, standard_deviation, variance))
```

```
        Mean : 0.0081
        Standard Deviation : 1.021
        Variance : 1.042
```

[9]:
```python
# Calculating Minumum & Maximum Values, Quantile points & Inter-Quantile Range
quantile = np.quantile(data, [0,0.25,0.5,0.75,1])
min_value, first_quantile, second_quantile, thrid_quantile, max_value = quantile
print('\
       Minimum Value :       {:.3f}\n\
       Maximum Value :       {:.3f}\n\n\
       First Quantile :      {:.3f}\n\
       Second Quantile :     {:.3f}\n\
       Third Quantile :      {:.3f}\n\
       Inter Quartile Range: {:.3f}'\
      .format(min_value, max_value, first_quantile, second_quantile,␣
 ↪thrid_quantile, thrid_quantile-first_quantile))
```

```
      Minimum Value :       -3.238
      Maximum Value :       2.644

      First Quantile :      -0.694
      Second Quantile :     0.005
      Third Quantile :      0.707
      Inter Quartile Range: 1.401
```

[10]:
```python
# Calculating Skewness & Kortosis
skewness = st.skew(data, axis=0, bias=True)
kortosis = st.kurtosis(data, axis=0, bias=True)
print('\
        Skewness : {:.4f}'.format(skewness))
print('\
        Kortosis : {:.4f}'.format(kortosis))
```

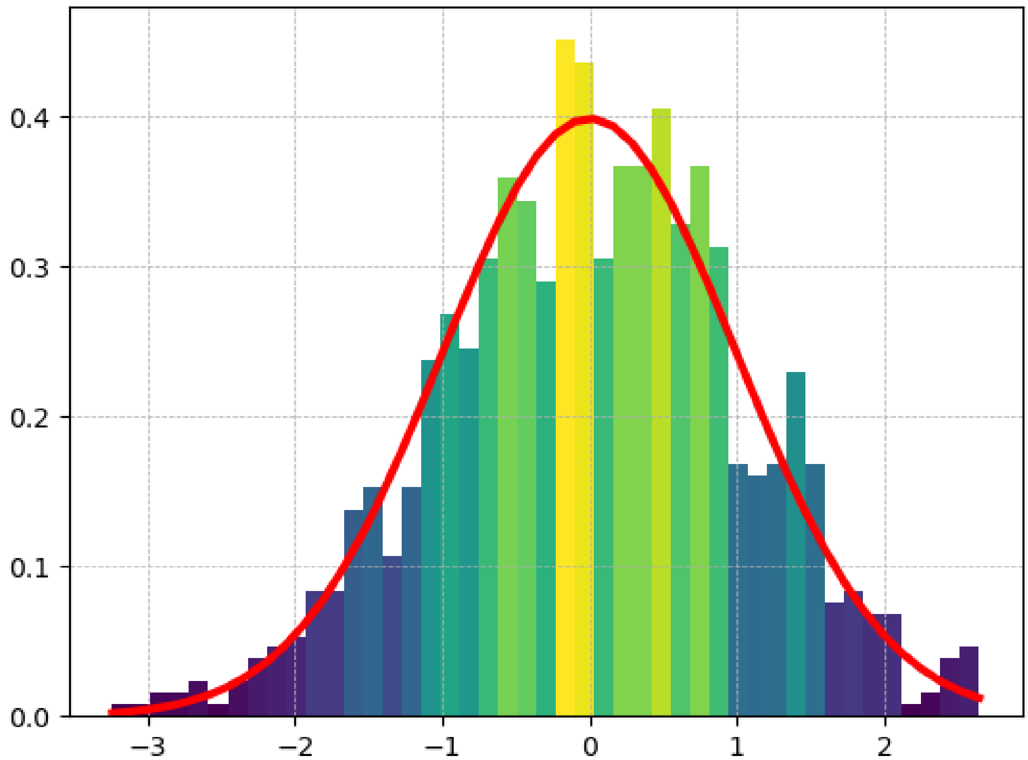
```
        Skewness : -0.1128
        Kortosis : -0.1116
```

5.4 Data Visualization

5.4.1 Weighted Histogram

[11]:
```python
# Ploting Histogram of the Generated Data based on values
count, bins, ignored = plt.hist(data, 45, density=True)
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (bins - mu)**2 / (2 *␣
 ↪sigma**2) ), linewidth=3, color='r')
plt.grid(linestyle='--', linewidth=0.5)

# We'll color code by height, but you could use any scalar
fracs = count / count.max()

# we need to normalize the data to 0..1 for the full range of the colormap
norm = matplotlib.colors.Normalize(fracs.min(), fracs.max())
```

```
# Now, we'll loop through our objects and set the color of each accordingly
for thisfrac, thisignored in zip(fracs, ignored):
    color = plt.cm.viridis(norm(thisfrac))
    thisignored.set_facecolor(color)

plt.show()
```

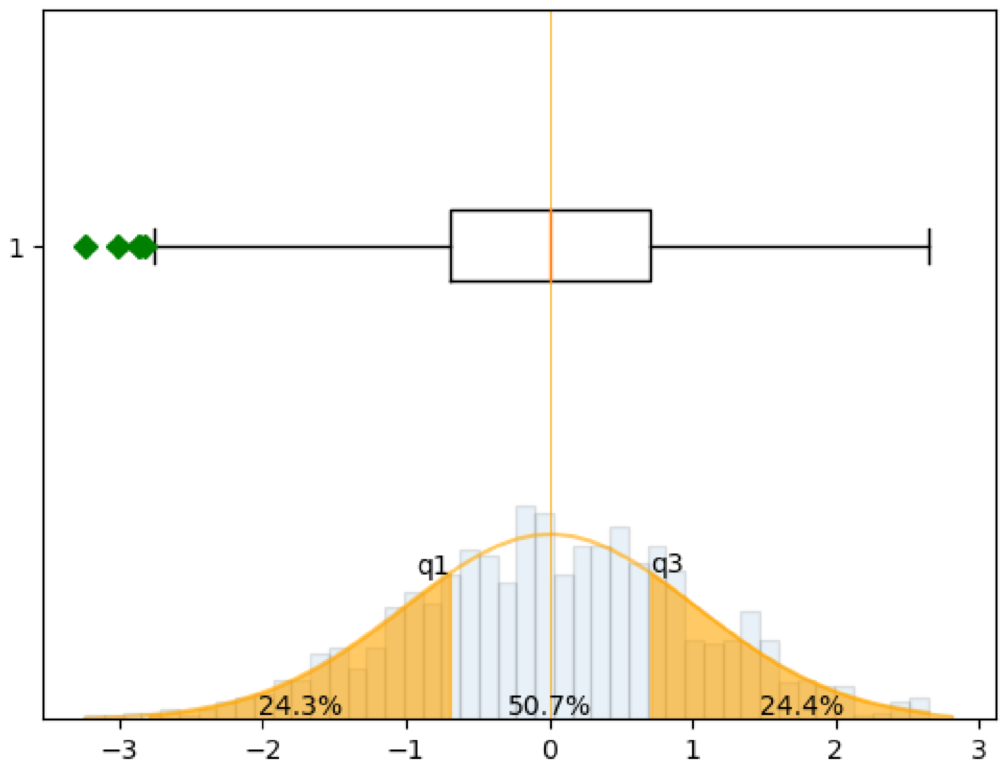


Weighted average normal of the generated data

5.4.2 Data distribution of Normal Distribution

[12]:
```
# Histogram with Box Plot
n, bins, patches = plt.hist(data, 45, density=True, alpha=.1, edgecolor='black' )
mu = data.mean()
sigma = data.std()
pdf = 1/(sigma*np.sqrt(2*np.pi))*np.exp(-(bins-mu)**2/(2*sigma**2))
median, q1, q3 = np.percentile(data, 50), np.percentile(data, 25), np.
 ↪percentile(data, 75)

# Probability density function
plt.plot(bins, pdf, color='orange', alpha=.6)
```

```
# Fill from Q1-1.5*IQR to Q1 and Q3 to Q3+1.5*IQR
iqr = 1.5 * (q3-q1)
x1 = np.linspace(q1 - iqr, q1)
x2 = np.linspace(q3, q3 + iqr)
pdf1 = 1/(sigma*np.sqrt(2*np.pi))*np.exp(-(x1-mu)**2/(2*sigma**2))
pdf2 = 1/(sigma*np.sqrt(2*np.pi))*np.exp(-(x2-mu)**2/(2*sigma**2))
plt.fill_between(x1, pdf1, 0, alpha=.6, color='orange')
plt.fill_between(x2, pdf2, 0, alpha=.6, color='orange')

# Add text to bottom graph.
low_per = 100*(st.norm(mu, sigma).cdf(q1)-st.norm(mu, sigma).cdf(q1-iqr))
mid_per = 100*(st.norm(mu, sigma).cdf(q3)-st.norm(mu, sigma).cdf(q1))
hig_per = 100*(st.norm(mu, sigma).cdf(q3+iqr)-st.norm(mu, sigma).cdf(q3))

plt.annotate("{:.1f}%".format(low_per), xy=(q1-iqr/2, 0), va='bottom',␣
↪ha='center')
plt.annotate("{:.1f}%".format(mid_per), xy=(median, 0), va='bottom', ha='center')
plt.annotate("{:.1f}%".format(hig_per), xy=(q3+iqr/2, 0), va='bottom',␣
↪ha='center')
plt.annotate('q1', xy=(q1, st.norm(mu, sigma).pdf(q1)), ha='right')
plt.annotate('q3', xy=(q3, st.norm(mu, sigma).pdf(q3)), ha='left')

# Boxplot of Histogram
plt.boxplot(data, 0, 'gD', vert=False)
plt.axvline(median, color='orange', alpha=1, linewidth=.5)
plt.axis('auto')
plt.show()
```

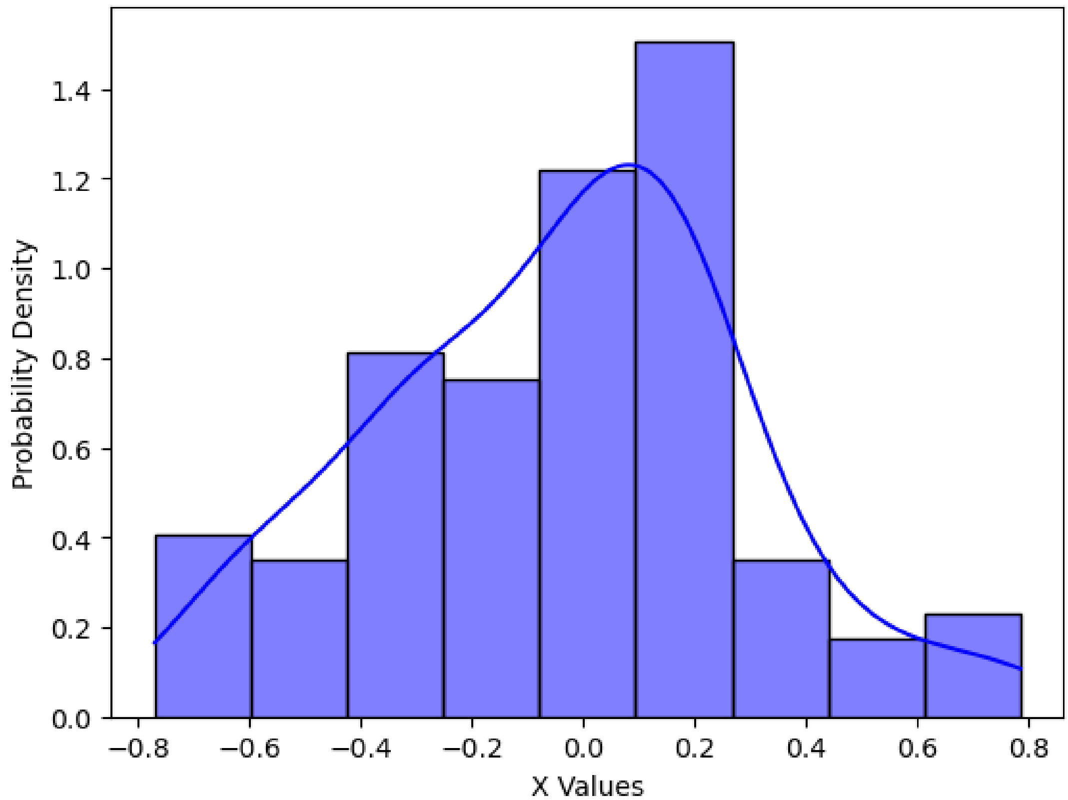


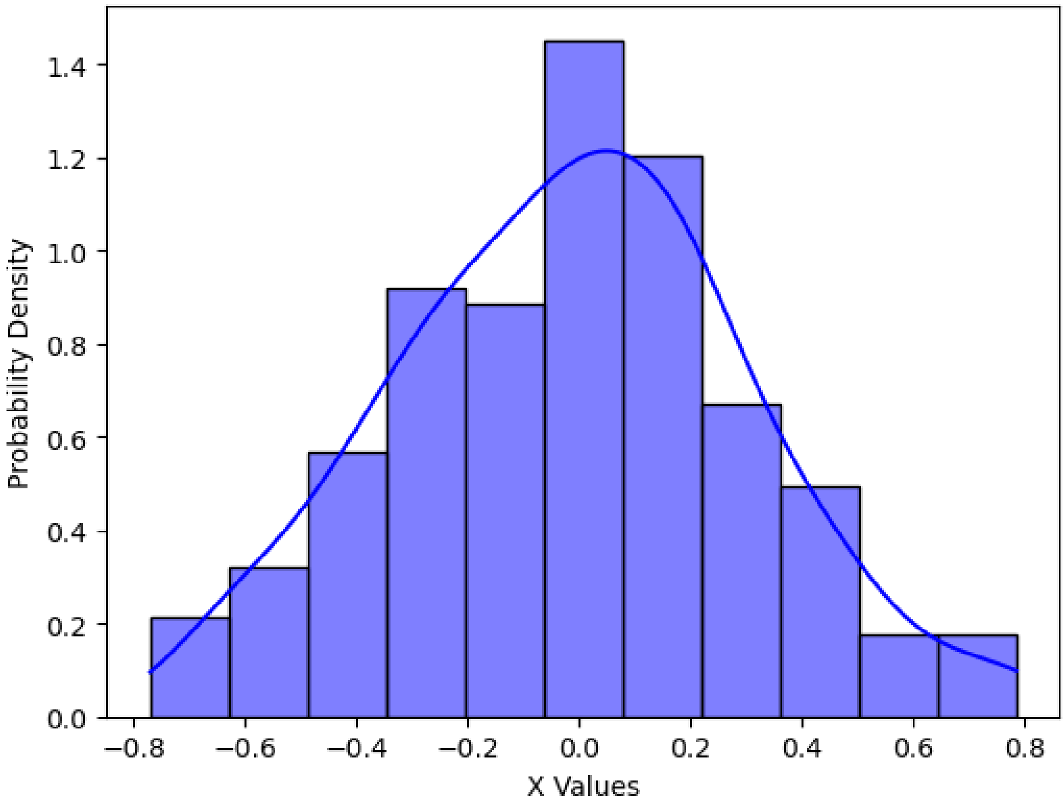
Differenciating outliers from the data.

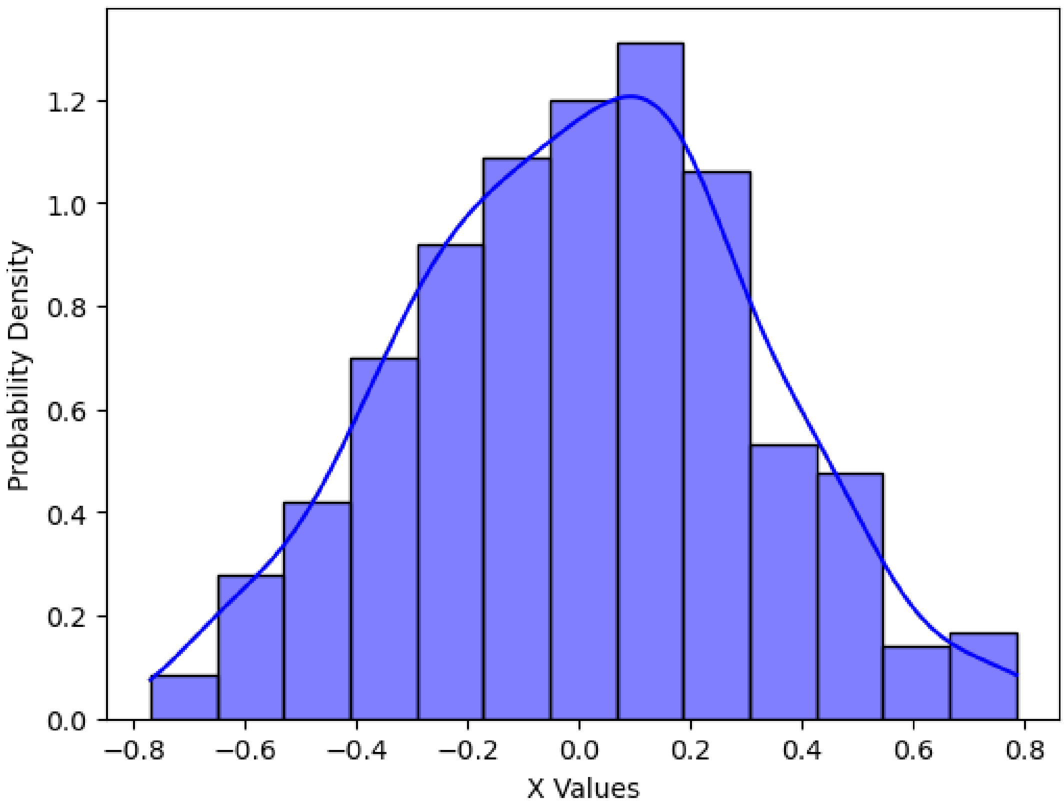
5.5 Central Limit Theorem Verification

[13]:
```
# Sample Mean calculator
mean_list = []
def calc_sample_mean(sample_size, no_of_sample_means):
    for i in range(no_of_sample_means):
        sample = random.sample(list(data),sample_size)
        sample_mean=np.mean(sample)
        mean_list.append(sample_mean)
    return mean_list

def plot_generator(sample_size, no_of_sample_means):
    mean_2=calc_sample_mean(sample_size=10, no_of_sample_means=100) #sample si
    # number of sample_means indicate the number of times the process would be
    sns.histplot(mean_2, color='b',kde=True, stat="density")
    plt.xlabel('X Values')
    plt.ylabel('Probability Density')

    plt.show()
    return 0
```

```
for iteration in range(1,4):
    plot_generator(50*iteration, 100)
```



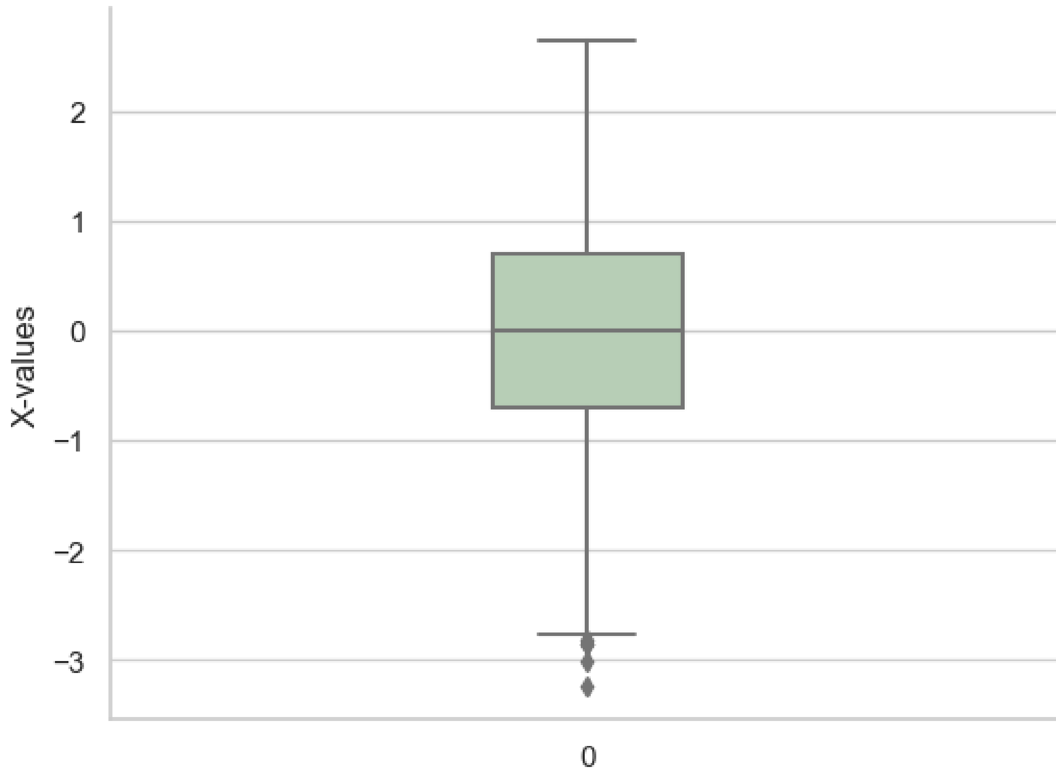

Probability Density
X Values
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8



5.6 Outlier Detection

[14]:
```
sns.set(style="whitegrid")
sns.boxplot(data=data, color="#B3D2B2", width=0.2).set(ylabel='X-values')
sns.despine()

#Detecting outliers using interquartile range
q1 = np.quantile(data,0.25)
q3 = np.quantile(data,0.75)
IQR = q3-q1
outliers = data[((data<(q1-1.5*IQR)) | (data>(q3+1.5*IQR)))]

#Detecting outliers using z-scores method(since we are dealing with normal
upper_limit = mean + 2.73 * standard_deviation
lower_limit = mean - 2.73 * standard_deviation
data[((data > upper_limit) | (data < lower_limit))]
print(outliers)
```

```
[-3.01603001 -3.23848548 -2.85902974 -2.85989311 -2.82122627]
```



5.7 Finding Probability

5.7.1 P(X=0.1)

[15]:
```
st.norm.pdf((0.1 - mu)/ sigma)
```

[15]: 0.39732943162730133

5.7.2 P(X<0.1)

[16]:
```
st.norm.cdf((0.1 - mu)/ sigma)
```

[16]: 0.5358608362133812

5.7.3 P(-0.1<X<0.1)

[17]:
```
st.norm.cdf((0.1 - mu)/ sigma) - st.norm.cdf((-0.1 - mu)/ sigma)
```

[17]: 0.07803366537781253

6

6.1 Generating Discrete Data for Poision Distribution

[18]:
```
# Poisson Discrete Distribution
mu = 5
data_discrete = st.poisson.rvs(mu,size = 1000)
data_discrete
```

[18]:
```
array([4, 6, 7, ..., 7, 7, 5], dtype=int64)
```

6.2 Statistical Analysis

[19]:
```
mean, standard_deviation, variance = data_discrete.mean(), data_discrete.std(),␣
↪data_discrete.std()**2
mode = st.mode(data_discrete, keepdims = True)
print('Mean : {:.4f}\n\
Standard Deviation : {:.3f}\n\
Variance : {:.3f}'.format(mean, standard_deviation, variance))
```

```
Mean : 4.9510
Standard Deviation : 2.220
Variance : 4.929
```

[20]:
```
quantile = np.quantile(data_discrete, [0,0.25,0.5,0.75,1])
min_value, first_quantile, second_quantile, thrid_quantile, max_value = quantile
print('Minimum Value : {:.3f}\n\
Maximum Value : {:.3f}\n\n\
First Quantile : {:.3f}\n\
Second Quantile : {:.3f}\n\
Third Quantile : {:.3f}\n\
'.format(min_value, max_value, first_quantile, second_quantile, thrid_quantile))
```

```
Minimum Value : 0.000
Maximum Value : 13.000

First Quantile : 3.000
Second Quantile : 5.000
Third Quantile : 6.000
```

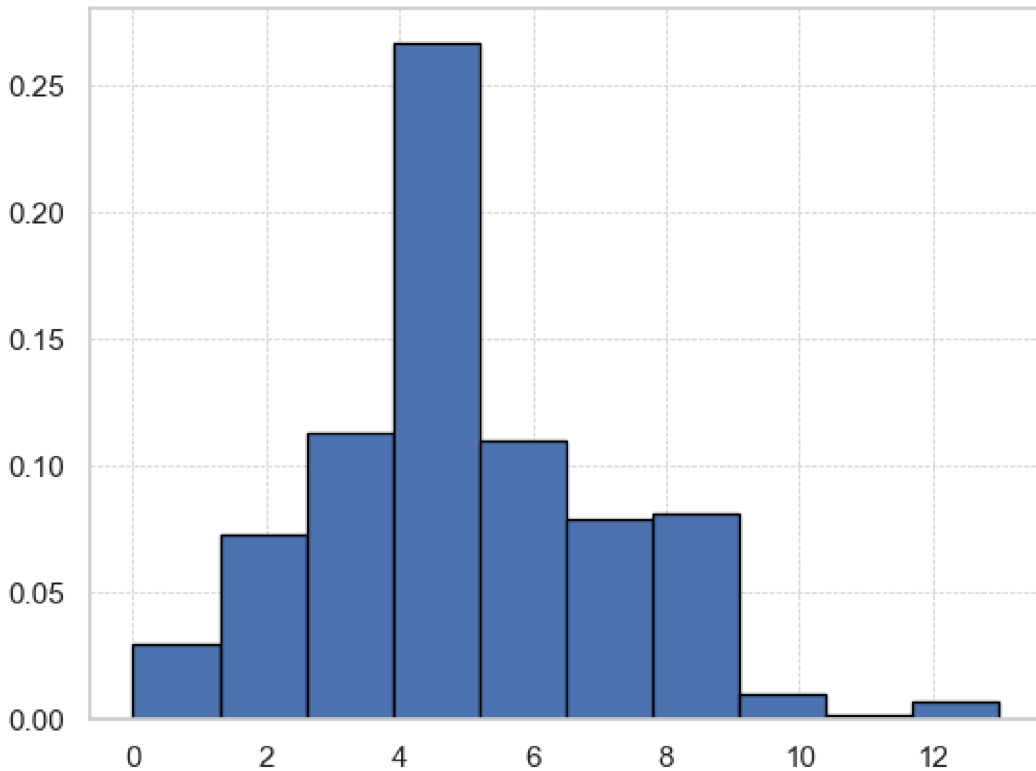
[21]:
```
skewness = st.skew(data_discrete, axis=0, bias=True)
kortosis = st.kurtosis(data_discrete, axis=0, bias=True)
print('Skewness : {:.4f}'.format(skewness))
print('Kortosis : {:.4f}'.format(kortosis))
```

```
Skewness : 0.5054
Kortosis : 0.1766
```

6.3 Data Visualization

6.3.1 Histogram

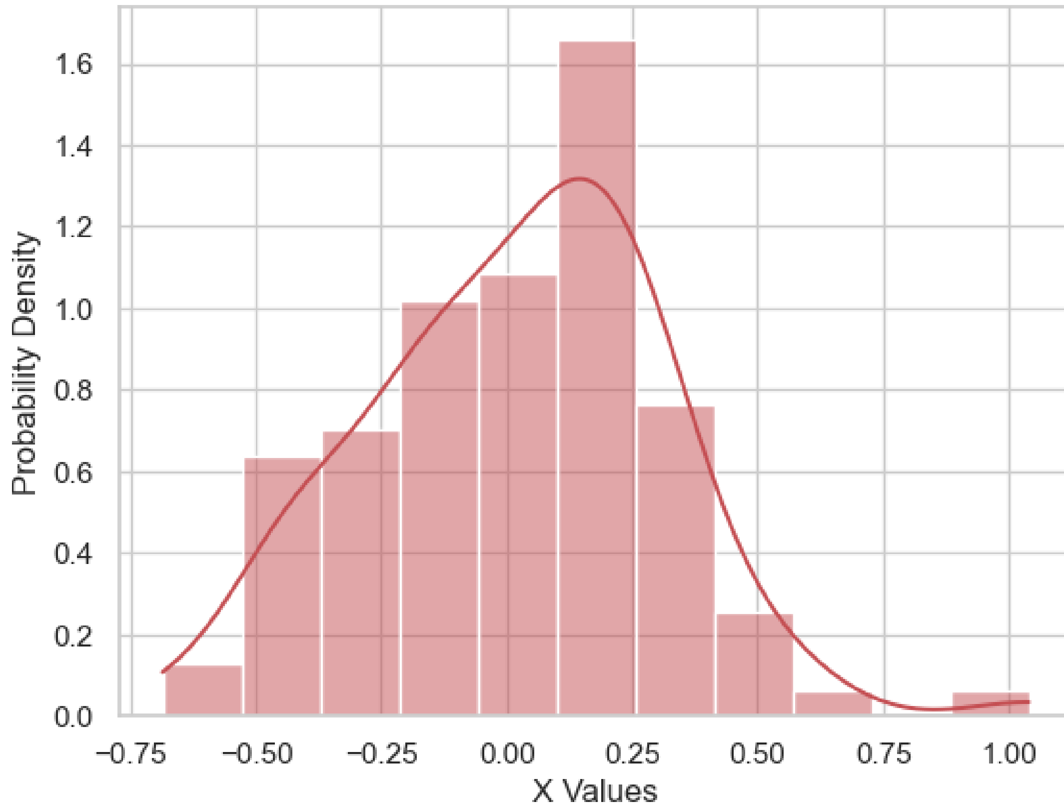
[22]:
```
plt.hist(data_discrete, density=True, edgecolor='black')
plt.grid(linestyle='--', linewidth=0.5)
plt.show()
```

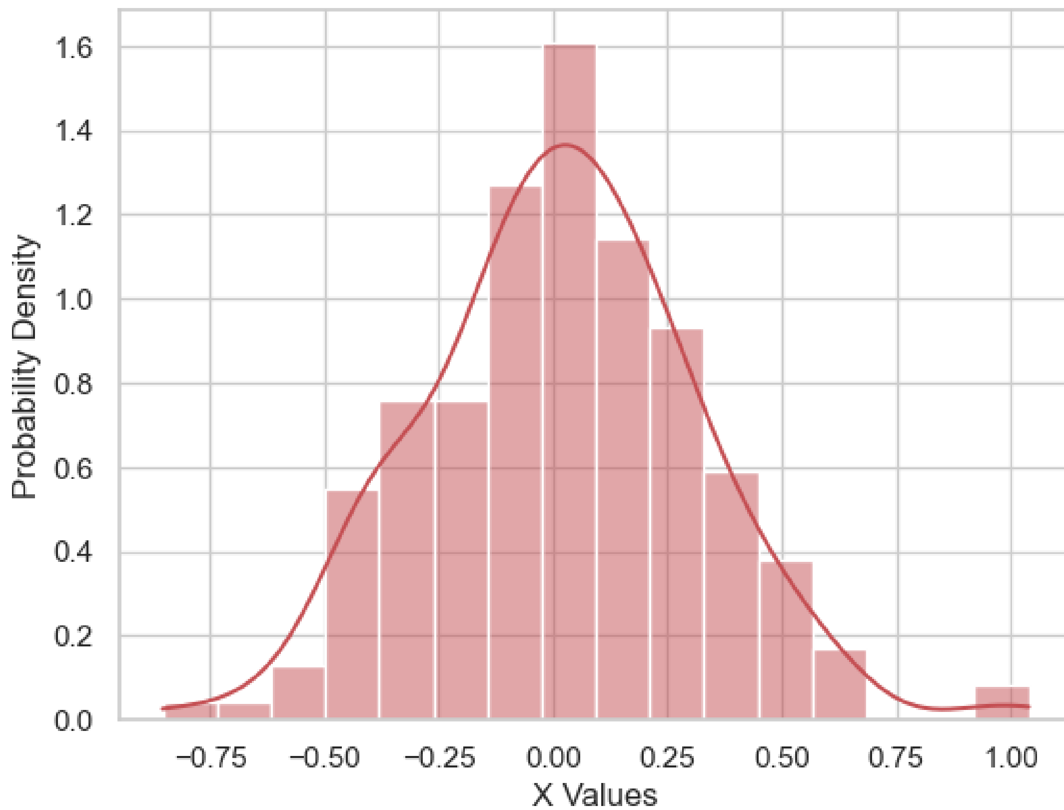


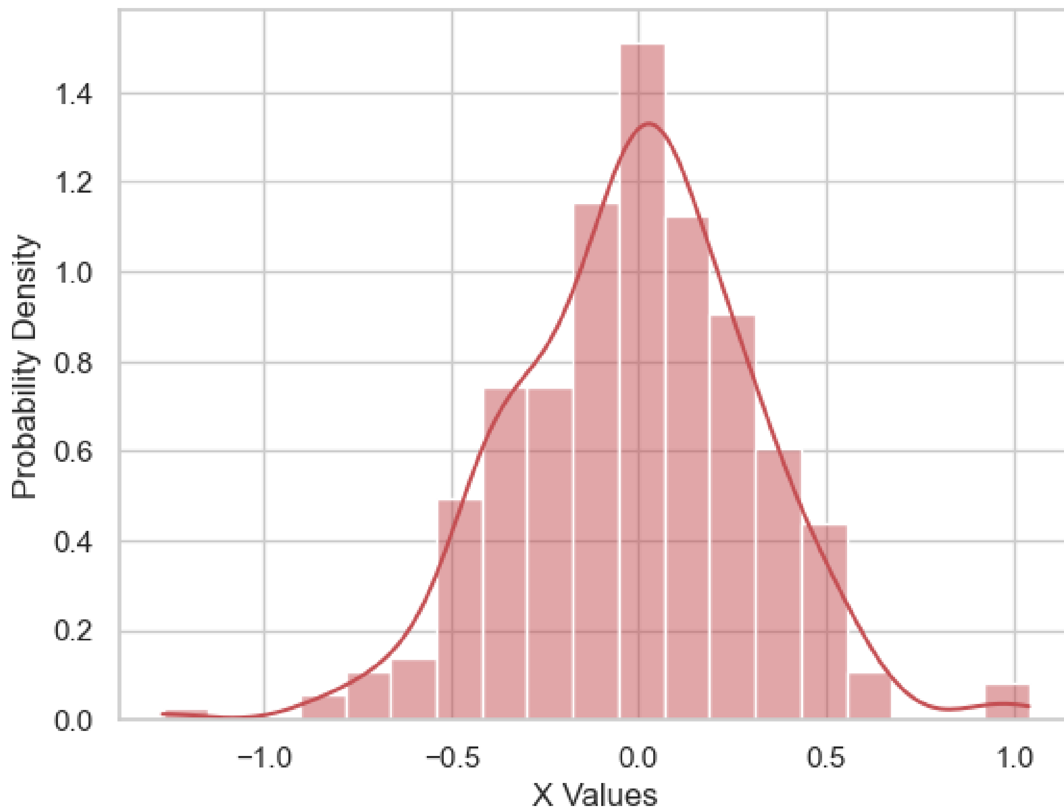
6.4 Poision Random Variable Check

[23]:
```
# Sample Mean calculator
mean_list = []
def calc_sample_mean(sample_size, no_of_sample_means):
    for i in range(no_of_sample_means):
        sample = random.sample(list(data),sample_size)
        sample_mean=np.mean(sample)
        mean_list.append(sample_mean)
    return mean_list

def plot_generator(sample_size, no_of_sample_means):
    mean_2=calc_sample_mean(sample_size=10, no_of_sample_means=100) #sample si
    # number of sample_means indicate the number of times the process would be
    sns.histplot(mean_2, kde=True, color='r', stat="density")
```

```
    plt.xlabel('X Values')
    plt.ylabel('Probability Density')

    plt.show()
    return 0

for iteration in range(1,4):
    plot_generator(50*iteration, 100)
```




Probability Density
X Values
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

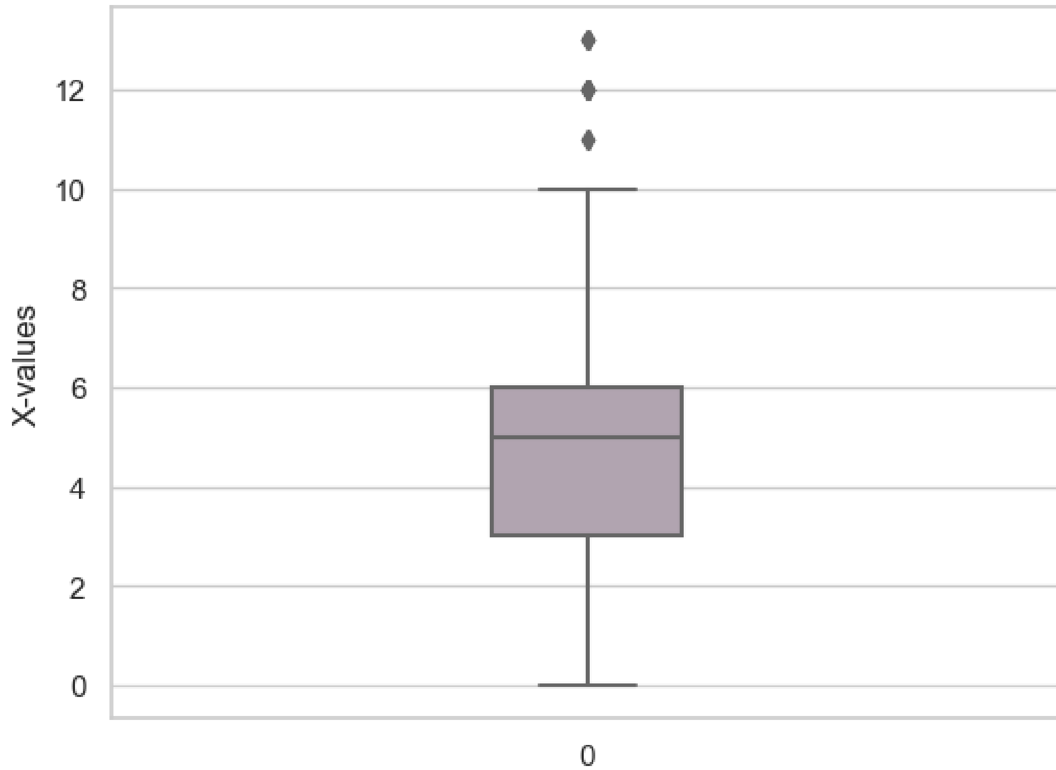


6.5 Boxplot

[24]:
```
#By visualization of box_plot
sns.boxplot(data_discrete, color="#B3A2B2", width=0.2).set(ylabel='X-values')

#Detecting outliers using interquartile range
q1 = np.quantile(data_discrete,0.25)
q3 = np.quantile(data_discrete,0.75)
IQR = q3-q1
outliers = data_discrete[((data_discrete<(q1-1.5*IQR)) | (data_discrete>(q1+1.
↪5*IQR)))]

#Detecting outliers using z-scores method(since we are dealing with normal
upper_limit = mean + 2.73 * standard_deviation
lower_limit = mean - 2.73 * standard_deviation
data_discrete[((data_discrete > upper_limit) | (data_discrete < lower_limit))]
print(outliers)
```

```
[ 9  8 12  8  9 12  8  8  9  8  9  8 12  8  8  8  8 10  8  8 10  9  8  9
 10  8  9  8  8  8  8  8  8  9  8  9 10  9  8  9  9  9  9  8  8  8  9 10
  8  8  9 13 11  8  9  9  8  9  8  8 10  9 10  9  9 10  9 13  8  8  9  8
```

```
12   9   9   9   8   9   8   8   8   8   9   8  10   8   8   8   8   9   9   8   9  10   8   9
10   9   8   8  12   8   8   9   9  10   8   9   8   9  11   8   8   8  12   9   8   8  13   9
 8   9   9   9   9   9   9  10   9]
```



6.6 Probabality Calculation

6.6.1 P(X=1)

[25]:
```
st.poisson.pmf(k=1, mu=5)
```

[25]:
```
0.03368973499542734
```

6.6.2 P(X<=1)

[26]:
```
st.poisson.cdf(k=1, mu=5)
```

[26]:
```
0.04042768199451279
```

6.6.3 P(X>=1)

[27]:
```
1 - st.poisson.cdf(k=1, mu=5)
```

[27]: 0.9595723180054873

7

7.1 Markov Chains

A Markov chain is a mathematical system that experiences transitions from one state to another according to certain probabilistic rules. The defining characteristic of a Markov chain is that no matter how the process arrived at its present state, the possible future states are only dependent on the current state.

$P(X_{n+1}|X_n) = P(X_{n+1}|X_n, X_{n-1}, \ldots, X_1, X_0)$

7.1.1 Transition Matrix Simulation

[28]:
```
# Genrating transition matrix from a fixed path
transitions = ['A', 'B', 'B', 'D', 'C', 'B', 'A', 'A', 'C', 'D', 'A', 'B', 'A',␣
 ↪'C']
uniq_label = pd.Series(transitions).unique().tolist()
num = len(uniq_label)

def rank(c):
    return ord(c) - ord('A')

T = [rank(c) for c in transitions]

#create matrix of zeros
M = [[0]*num for _ in range(num)]
for (i,j) in zip(T,T[1:]):
    M[i][j] += 1

#now convert to probabilities:
for row in M:
    n = sum(row)
    if n > 0:
        row[:] = [f/sum(row) for f in row]

df = pd.DataFrame(M)
df.columns = uniq_label
df.index = uniq_label
transition_matrix = np.array(df)

df
```

[28]:

	A	B	D	C
A	0.2	0.40	0.4	0.00
B	0.5	0.25	0.0	0.25
D	0.0	0.50	0.0	0.50

```
C  0.5  0.00  0.5  0.00
```

Above is a transistion matrix generated based on the path followed by a partical, after following a fixed certain path walk.

7.1.2 Recurrent Events

A state in a Markov chain is recurrent if, once the system enters that state, it will eventually return to that state with probability 1, in other words recurrent event refers to an event that will eventually happen again with probability 1, given that the system starts in a certain state.

[29]:
```
recurrent_matrix = np.array(df)

for i in range(0,10):
    recurrent_matrix = np.matmul(recurrent_matrix, recurrent_matrix)

recurrent_matrix = pd.DataFrame(recurrent_matrix)

recurrent_matrix.columns = uniq_label
recurrent_matrix.index = uniq_label
recurrent_matrix
```

[29]:

	A	B	D	C
A	0.30303	0.30303	0.212121	0.181818
B	0.30303	0.30303	0.212121	0.181818
D	0.30303	0.30303	0.212121	0.181818
C	0.30303	0.30303	0.212121	0.181818

[30]:
```
recurrent_matrix.iloc[0]
```

[30]:
```
A    0.303030
B    0.303030
D    0.212121
C    0.181818
Name: A, dtype: float64
```

There are no state where we can guarantee that the event will occur with a probabality of 1.

7.1.3 Ergodicity

A markov chain matrix is called to be ergodicity markov chain matrix, if it is possible to go from every state to every state, irrespectively of number of steps taken.

[31]:
```
ergodicity=True

df = transition_matrix
state_prob = df[0]

for cel in state_prob:
```

```
    if(cel==0):
        ergodicity=False
        break
if(ergodicity):
    print("This is an Ergodicity Markov Chain Matrix.")
else:
    print("This is not an Ergodicity Markov Chain Matrix.")
```

```
This is not an Ergodicity Markov Chain Matrix.
```

7.1.4 Sensitivity Analysis

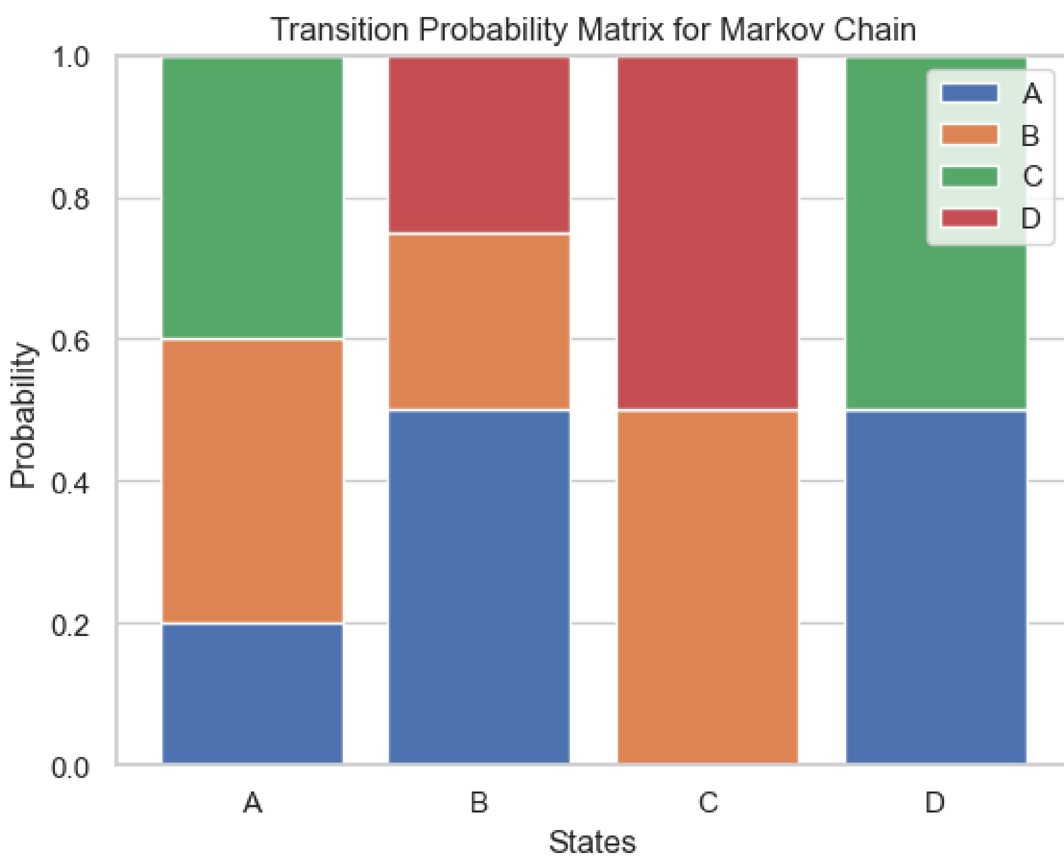
Sensitivity analysis determines how different values of an independent variable affect a particular dependent variable under a given set of assumptions. Sensitivity Analysis can be very usefull in case of checking the robustness of a system.

[32]:
```
# Let's define a steady-state distribution
def steady_state(trans_mat):
    eigval, eigvec = np.linalg.eig(trans_mat.T)
    index = np.argmin(np.abs(eigval - 1.0))
    return np.real(eigvec[:, index] / np.sum(eigvec[:, index]))



# Let's perform sensitivity analysis
def sensitivity_analysis(trans_mat, perturbation):
    perturbed_mat = trans_mat + perturbation

    perturbed_mat /= np.sum(perturbed_mat, axis=1)[:, np.newaxis]

    original_prob = steady_state(trans_mat)
    perturbed_prob = steady_state(perturbed_mat)

    sens_metric = (perturbed_prob - original_prob) / original_prob

    return sens_metric

perturbation = 0.01

sens_result = sensitivity_analysis(transition_matrix, perturbation)

for i, metric in enumerate(sens_result):
    print(f"For state {i + 1}, the Sensitivity Metric with Sign is {metric:.4f}")
```

```
For state 1, the Sensitivity Metric with Sign is -0.0077
For state 2, the Sensitivity Metric with Sign is -0.0086
For state 3, the Sensitivity Metric with Sign is 0.0093
For state 4, the Sensitivity Metric with Sign is 0.0162
```

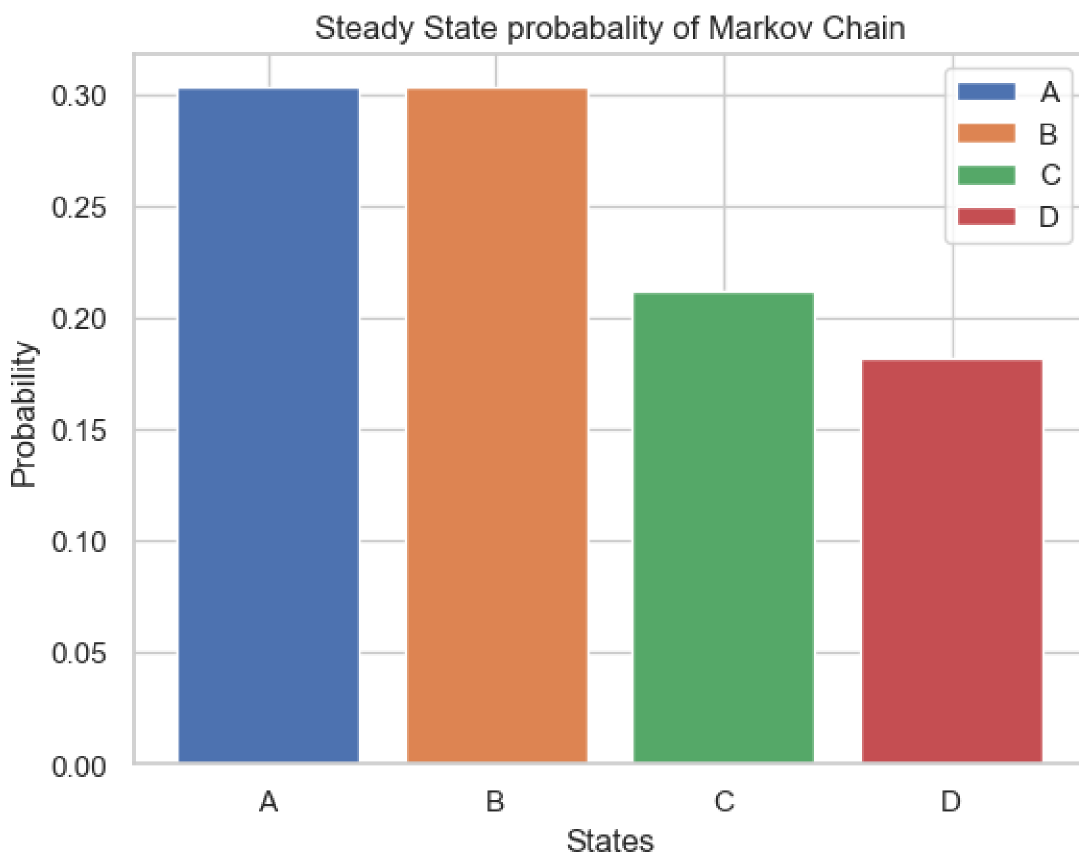
7.1.5 Visualization

[33]:
```
# State labels
state_labels = ['A', 'B', 'C', 'D']

# Plot the bar chart
plt.bar(state_labels, transition_matrix[:,0])
plt.bar(state_labels, transition_matrix[:,1], bottom=transition_matrix[:,0])
plt.bar(state_labels, transition_matrix[:,2], bottom=transition_matrix[:,0] +␣
↪transition_matrix[:,1])
plt.bar(state_labels, transition_matrix[:,3], bottom=transition_matrix[:,0] +␣
↪transition_matrix[:,1] + transition_matrix[:,2])

plt.title('Transition Probability Matrix for Markov Chain')
plt.xlabel('States')
plt.ylabel('Probability')
plt.legend(state_labels)
plt.show()
```



Above shown stacked bar chart represents the transition probabality of the Markov's Chain Process, with respect to all the possible states.

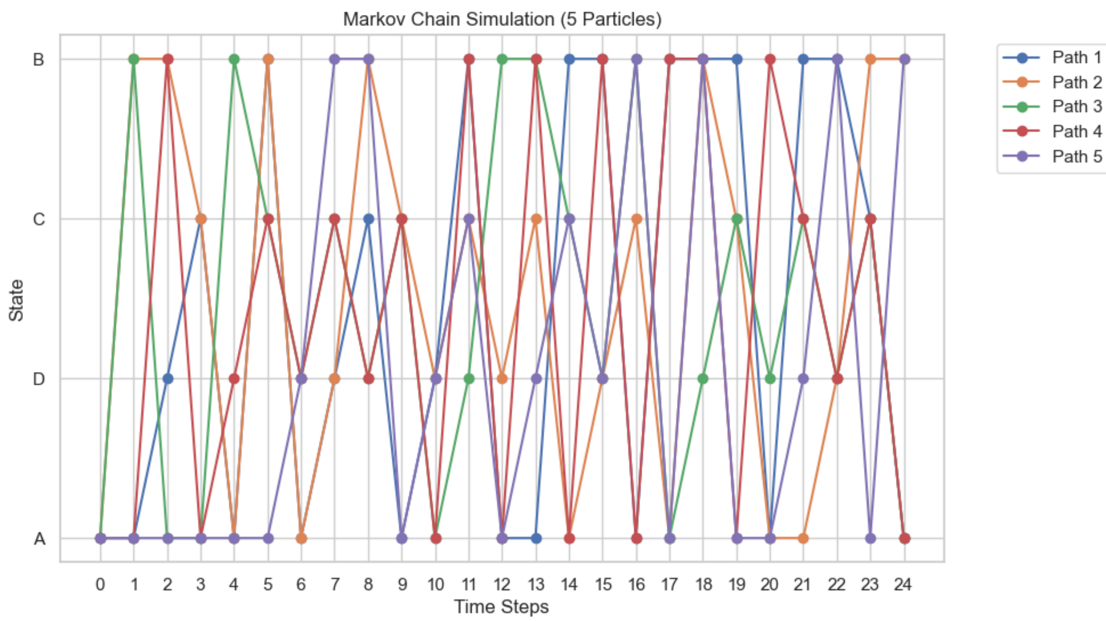
[34]:
```
values = np.array(recurrent_matrix.iloc[0])
labels = ["A", "B", "C", "D"]
colors = ['#4D72B0', '#DD8452', '#55A868', '#C54E52']

plt.bar(labels, values, color=colors, label=labels)
plt.title('Steady State probabality of Markov Chain')
plt.xlabel('States')
plt.ylabel('Probability')
plt.legend(labels)
plt.show()
```



Above bar graph represents the converged probabalities of all the states, at the steady state after a infinie number of iterations.

[35]:
```
def markov_chain(states, transition_matrix, initial_state, num_steps):
    current_state = initial_state
    trajectory = [current_state]
```

```python
    for _ in range(num_steps - 1):
        transition_probabilities = transition_matrix[states.index(current_state)]
        next_state = random.choices(states, weights=transition_probabilities)[0]
        current_state = next_state
        trajectory.append(next_state)

    return trajectory

def simulate_and_plot(num_simulations, states, transition_matrix, initial_state,␣
 ↪num_steps):
    plt.figure(figsize=(10, 6))

    for i in range(num_simulations):
        trajectory = markov_chain(states, transition_matrix, initial_state,␣
 ↪num_steps)
        plt.plot(range(num_steps), trajectory, marker='o', label=f'Path {i+1}')

    plt.title(f'Markov Chain Simulation ({num_simulations} Particles)')
    plt.xlabel('Time Steps')
    plt.ylabel('State')
    plt.xticks(range(num_steps))
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.show()

states = uniq_label
initial_state = 'A'
num_steps = 25
num_simulations = 5

simulate_and_plot(num_simulations, states, transition_matrix, initial_state,␣
 ↪num_steps)
```



Above line chart shows the paths followed by 5 different particals.

7.2 Comparison of Different Simulation Methods

7.2.1 Importance Sampling:

Importance Sampling is a technique designed to improve the estimation of rare events by assigning higher probabilities to outcomes that contribute more to the overall result. ### Control Variates:

Control Variates aim to minimize the variance of an estimator by incorporating the information from a related variable. By identifying a correlated auxiliary variable, the technique allows for a more efficient estimation of the desired quantity. ### Antithetic Variates:

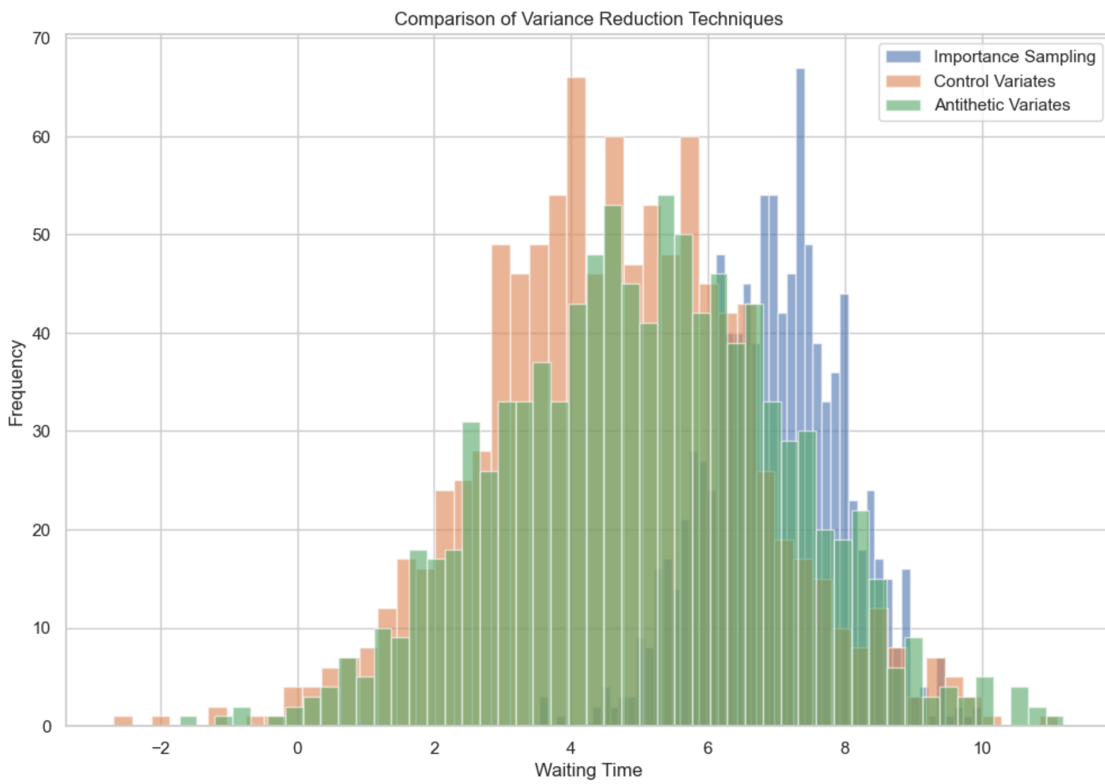
Antithetic Variates exploit the negative correlation between random variables to reduce variance. It involves generating pairs of correlated random variables and averaging their contributions to the estimator.

[36]:
```
# function to simulate waiting times using importance sampling technique
def simulate_importance_sampling(wait_simulation_duration):
    sampled_times = np.random.normal(7, 1, wait_simulation_duration)
    sampling_weights = np.exp(-0.5 * ((sampled_times - 5) / 2)*2) / np.sqrt(2 *
 ↪np.pi * 2*2)
    return sampled_times, sampling_weights

# function to simulate waiting times using control variates technique
def simulate_control_variates(wait_simulation_duration):
    bank_wait_times = np.random.normal(5, 2, wait_simulation_duration)
    atm_wait_times = np.random.normal(2, 1, wait_simulation_duration)
    control_variate = atm_wait_times
```

```
    return bank_wait_times, control_variate

# function to simulate waiting times using antithetic variates technique
def simulate_antithetic_variates(wait_simulation_duration):
    u1 = np.random.normal(0, 1, wait_simulation_duration)
    u2 = -u1
    sampled_times = np.random.normal(5, 2, wait_simulation_duration) + 0.5 * (u1␣
↪+ u2)
    return sampled_times
```

[37]:

```
# No. of simulation runs and wait_simulation_duration
num_simulations, wait_simulation_duration = 1000, 1000

# simulate waiting times using importance sampling technique, control variates,␣
↪antithetic variates
importance_sampled_times, importance_sampling_weights =␣
↪simulate_importance_sampling(wait_simulation_duration)
bank_waiting_times, control_variate_times =␣
↪simulate_control_variates(wait_simulation_duration)
antithetic_sampled_times = simulate_antithetic_variates(wait_simulation_duration)

df = pd.DataFrame({
    'ImportanceSampling': importance_sampled_times,
    'ControlVariates': bank_waiting_times - 0.2 * control_variate_times,
    'AntitheticVariates': antithetic_sampled_times
})

# visualizing the results
plt.figure(figsize=(12, 8))
plt.hist(df['ImportanceSampling'], bins=50, alpha=0.6, label='Importance␣
↪Sampling')
plt.hist(df['ControlVariates'], bins=50, alpha=0.6, label='Control Variates')
plt.hist(df['AntitheticVariates'], bins=50, alpha=0.6, label='Antithetic␣
↪Variates')
plt.title('Comparison of Variance Reduction Techniques')
plt.xlabel('Waiting Time')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



7.3 Simulation for Combinatorial Analysis

Combinatorial analysis is a branch of mathematics that focuses on counting, arranging, and understanding the possibilities and structures that arise in discrete, finite sets.

It includes methods like Permutation and Combinations, Binomial Coefficeint, Graph Theroy, Pigeonhole Principle and Inclusion-Exclusion Principle, Generating Functions, and Recurrence Functions.

Below is a application on Permutation and Combination.

[38]:
```
suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King',␣
 ↪'Ace']
deck = list(itertools.product(ranks, suits))

def draw_hand(num_cards):
    return random.sample(deck, num_cards)

def analyze_hand(hand):
    rank_counts = Counter(card[0] for card in hand)

    for count in rank_counts.values():
```

```
        if count == 2:
            return 'Pair'
        elif count == 3:
            return 'Three-of-a-Kind'
        elif count == 4:
            return 'Four-of-a-Kind'

    sorted_ranks = sorted(set(card[0] for card in hand), key=lambda x: ranks.
 ↪index(x))
    if len(sorted_ranks) == 5 and ranks.index(sorted_ranks[-1]) - ranks.
 ↪index(sorted_ranks[0]) == 4:
        return 'Straight'

    return 'High Card'
```

[39]:
```
num_simulations = 10000
hand_size = 5
hand_types = {'Pair': 0, 'Three-of-a-Kind': 0, 'Four-of-a-Kind': 0, 'Straight':␣
 ↪0, 'High Card': 0}

for _ in range(num_simulations):
    hand = draw_hand(hand_size)
    hand_type = analyze_hand(hand)
    hand_types[hand_type] += 1

probabilities = {hand_type: count / num_simulations for hand_type, count in␣
 ↪hand_types.items()}

print("Simulation Results:")
for hand_type, probability in probabilities.items():
    print(f"{hand_type}: Probability = {probability:.4f}")
```

```
Simulation Results:
Pair: Probability = 0.4719
Three-of-a-Kind: Probability = 0.0193
Four-of-a-Kind: Probability = 0.0001
Straight: Probability = 0.0034
High Card: Probability = 0.5053
```

8 Real Data Analysis

8.0.1 Importing the Dataset

[40]:
```
# Reading the CSV file
df = pd.read_csv("./adult.csv")
```

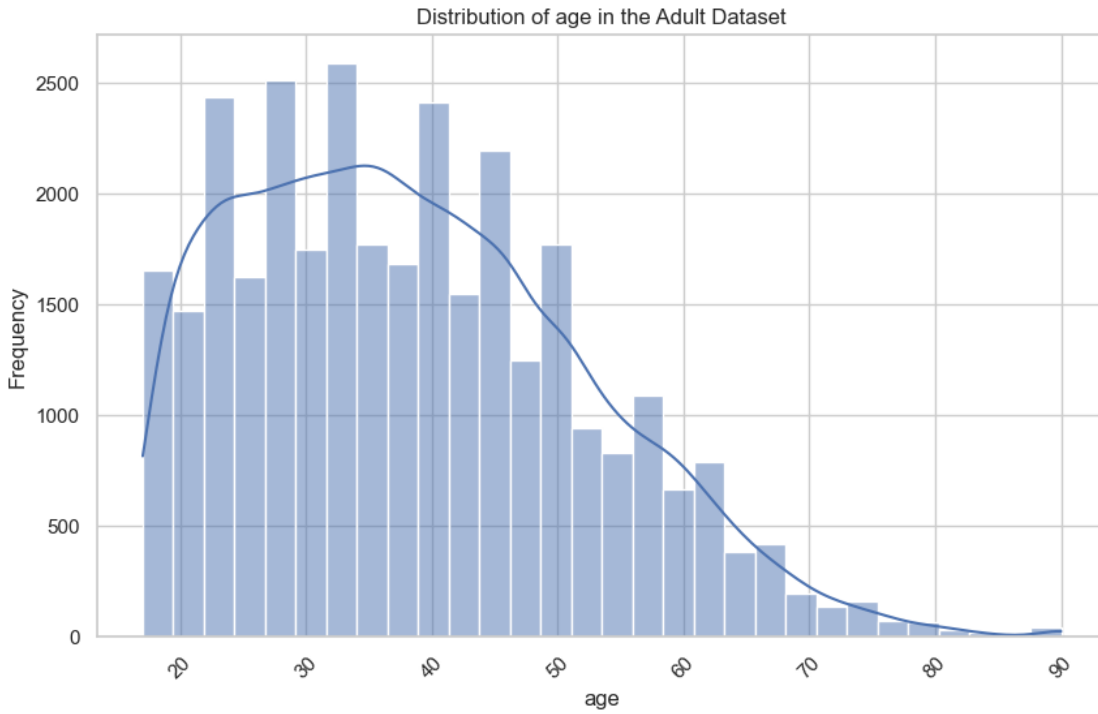
[41]:
```
# Change columns name
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',␣
↪'marital_status', 'occupation', 'relationship',
             'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week',␣
↪'native_country', 'income']

df.columns = col_names
df.columns
```

[41]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income'],
      dtype='object')
```

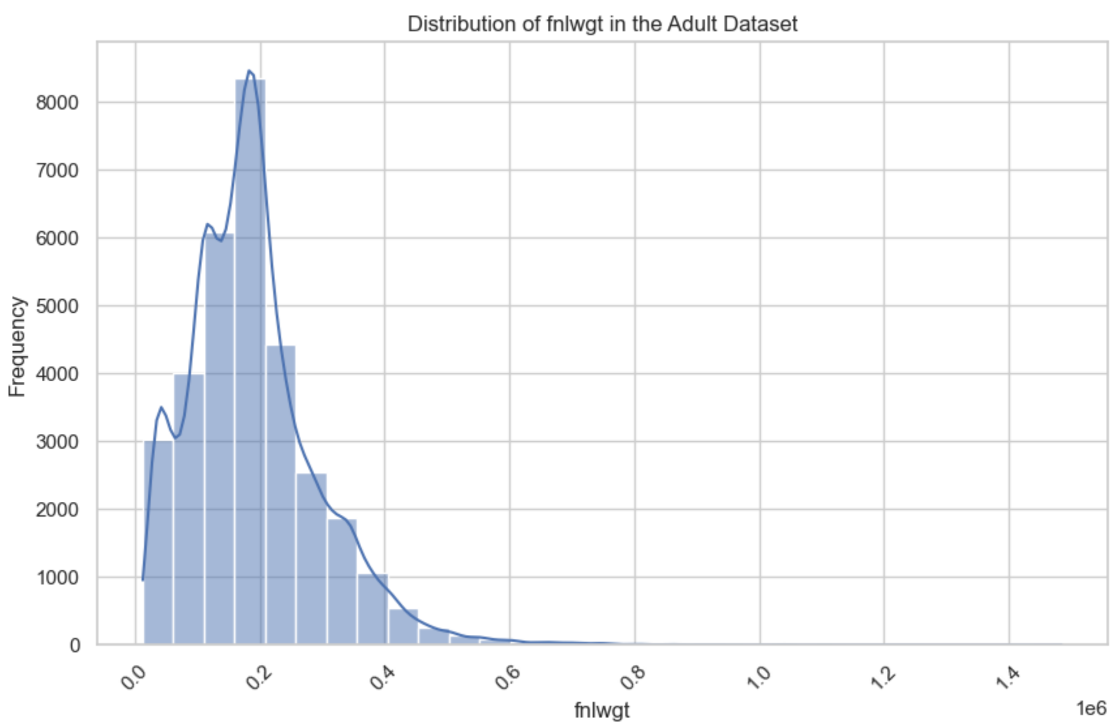
8.0.2 Visualizing Data

Continious Data

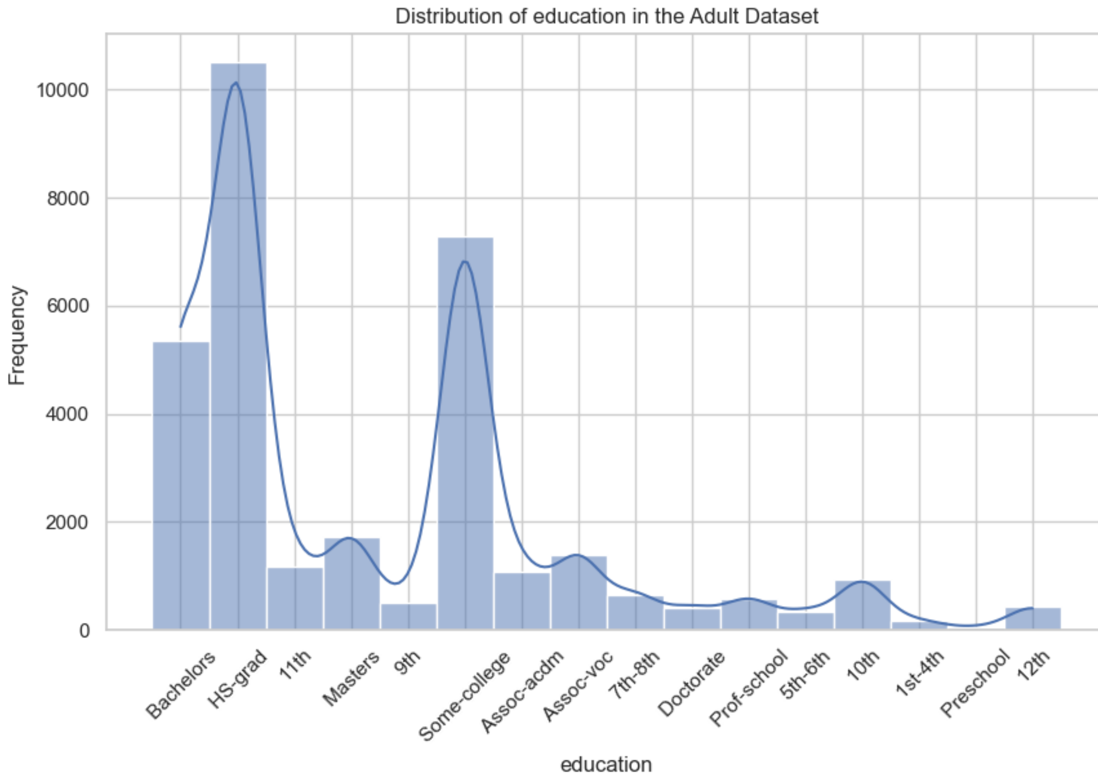
[42]:
```
# Plotting
hist_label = ['age', 'fnlwgt', 'education', 'education_num', 'capital_gain',␣
↪'capital_loss', 'hours_per_week' ]

for i in hist_label:
    plt.figure(figsize=(10, 6))
    sns.histplot(df[i], bins=30, kde=True)
    plt.title(f'Distribution of {str(i)} in the Adult Dataset')
    plt.xlabel(str(i))
    plt.xticks(rotation=45)
    plt.ylabel('Frequency')
    plt.show()
```

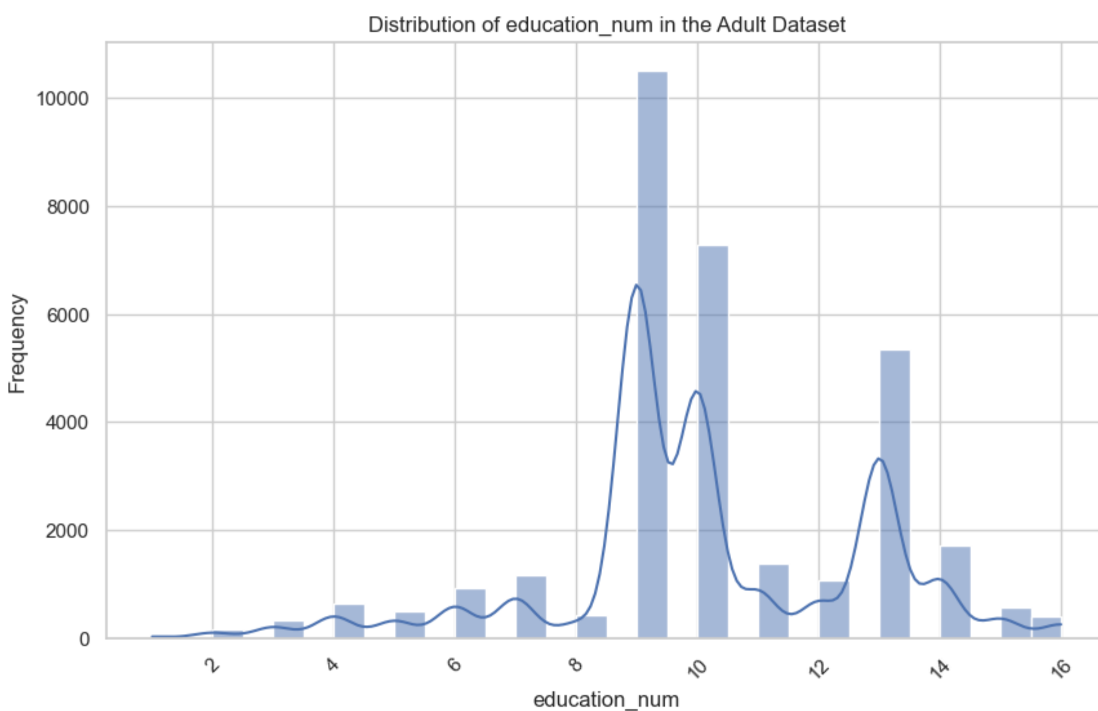


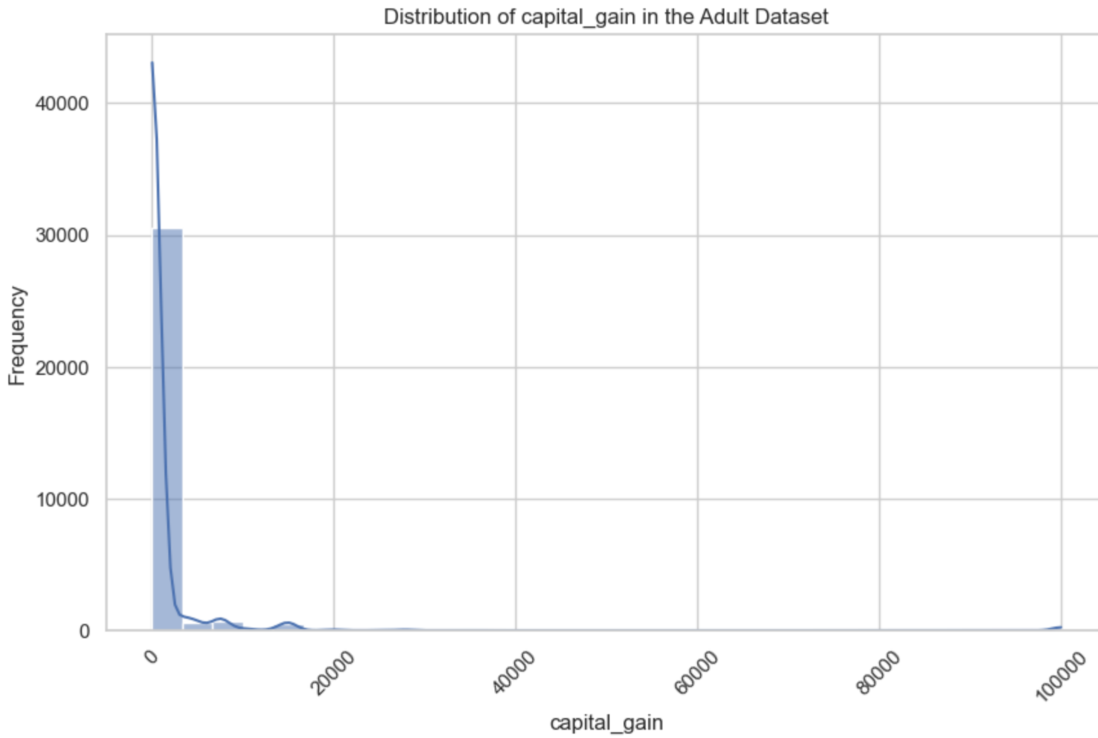
Distribution of age in the Adult Dataset
Frequency
2500
2000
1500
1000
500
0
20
30
40
50
60
70
80
90
age

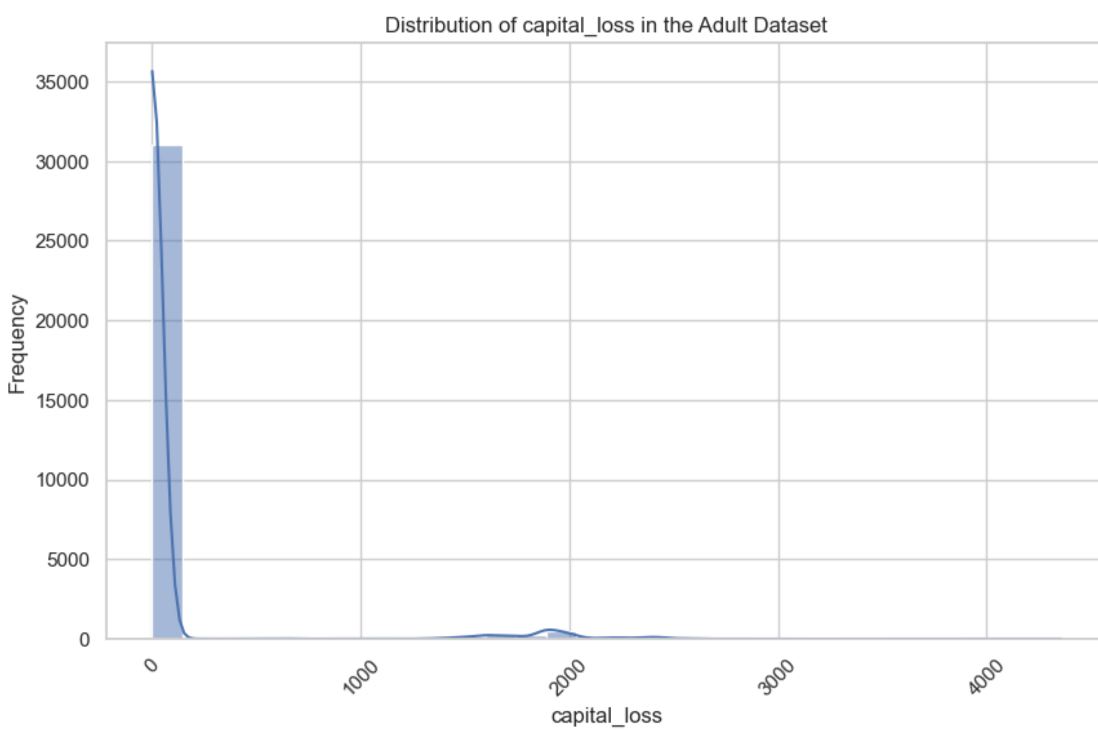


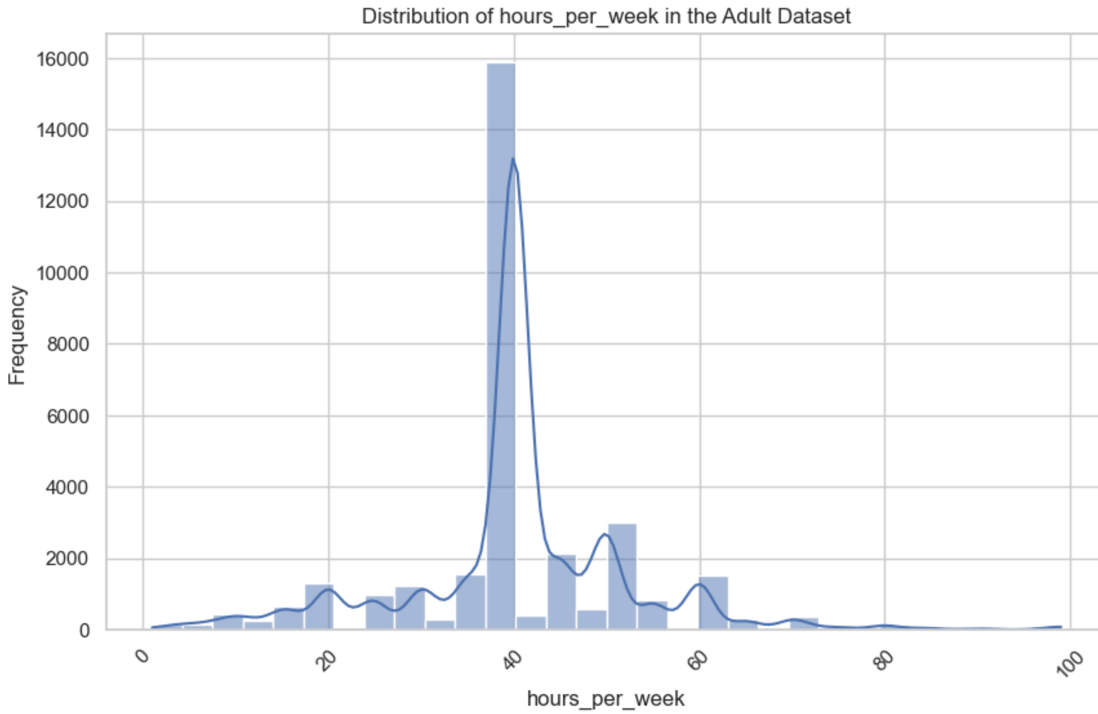
Distribution of fnlwgt in the Adult Dataset
Frequency
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
fnlwgt
1e6


Distribution of education in the Adult Dataset
Frequency
10000
8000
6000
4000
2000
0
Bachelors
HS-grad
11th
Masters
9th
Some-college
Assoc-acdm
Assoc-voc
7th-8th
Doctorate
Prof-school
5th-6th
10th
1st-4th
Preschool
12th
education


Distribution of education_num in the Adult Dataset
Frequency
10000
8000
6000
4000
2000
0
2
4
6
8
10
12
14
16
education_num

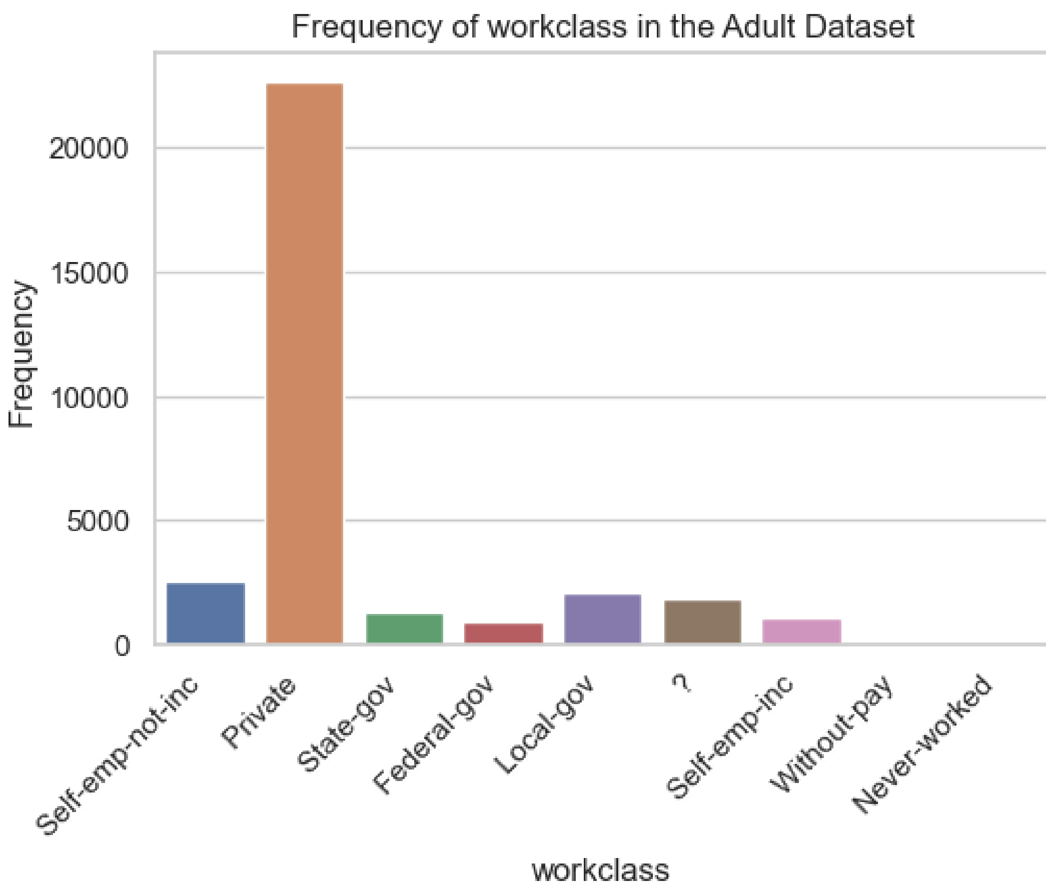

Distribution of capital_gain in the Adult Dataset
Frequency
40000
30000
20000
10000
0
0
20000
40000
60000
80000
100000
capital_gain

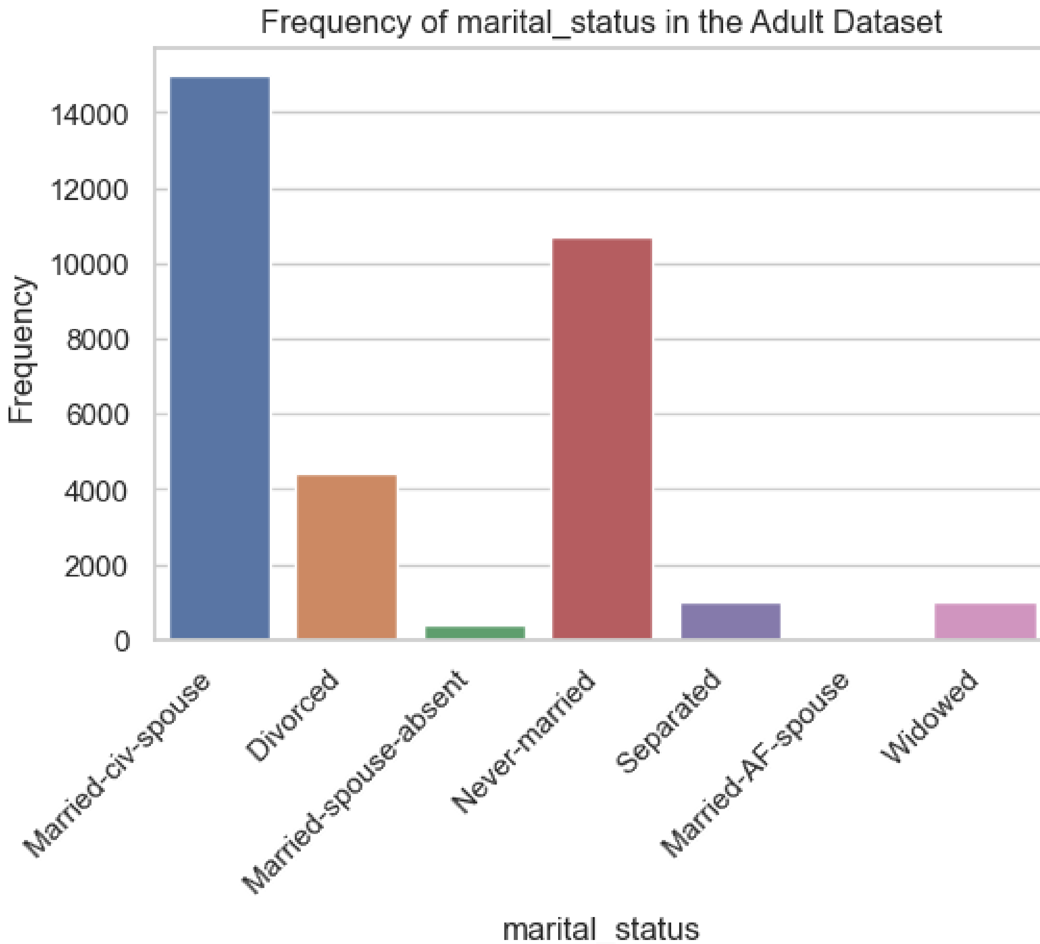

Distribution of capital_loss in the Adult Dataset
Frequency
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
capital_loss

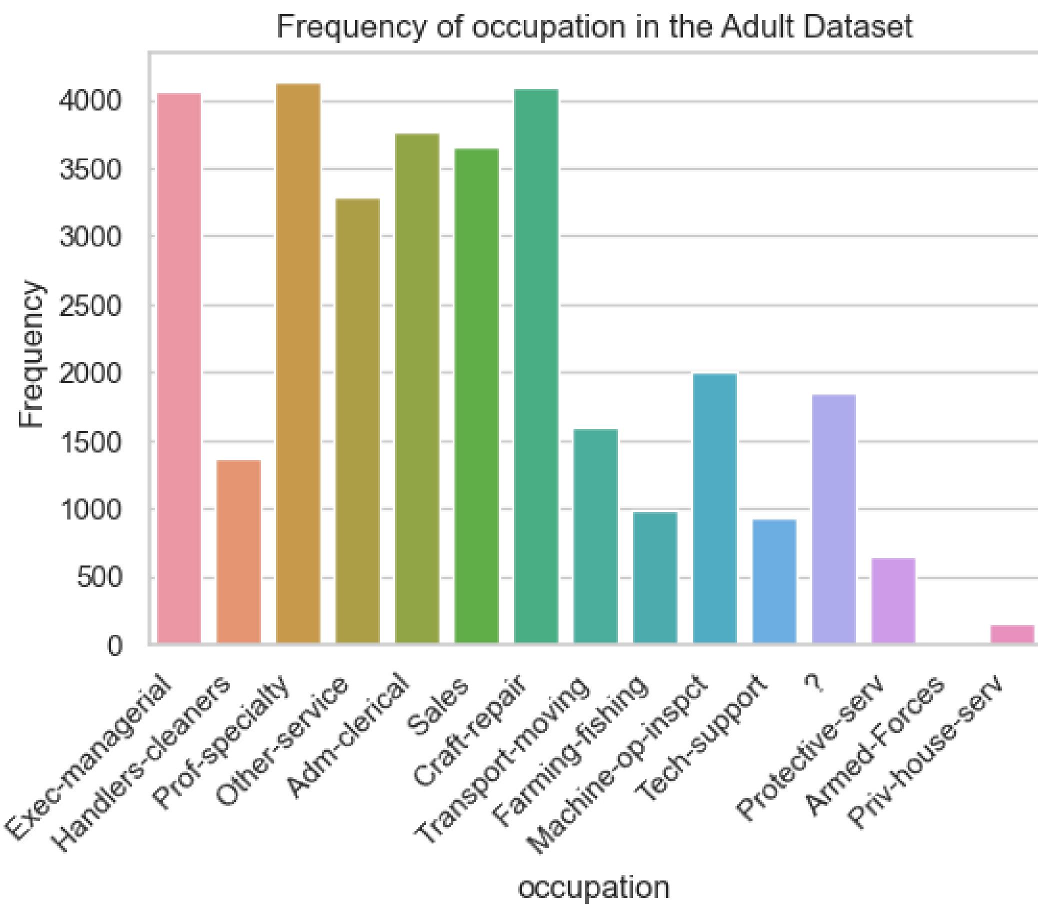


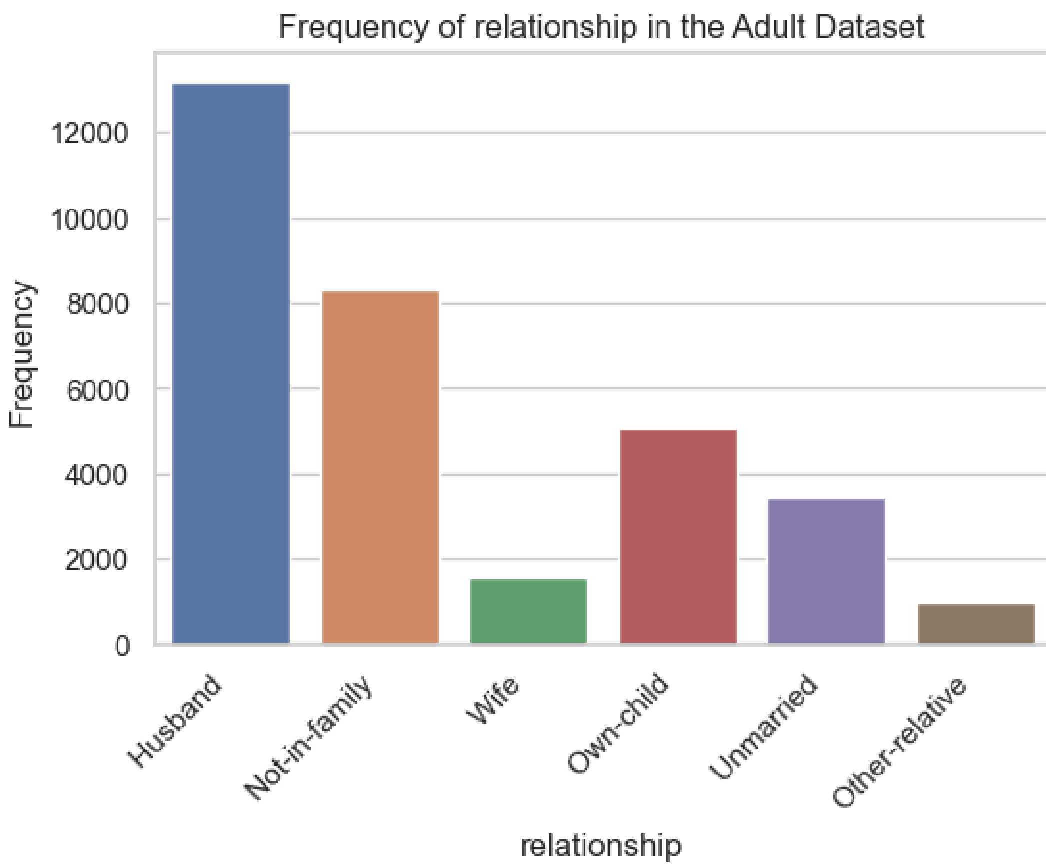
Catagorical Data

[43]:
```
# Plotting
count_label = ['workclass', 'marital_status', 'occupation', 'relationship',␣
↪'race', 'sex']

for j in count_label:
    plt.figure(figsize=(6, 4))
    sns.countplot(x=str(j), data=df)
    plt.title(f'Frequency of {str(j)} in the Adult Dataset')
    plt.xlabel(str(j))
    plt.xticks(rotation=45, ha="right")
    plt.ylabel('Frequency')
    plt.show()
```

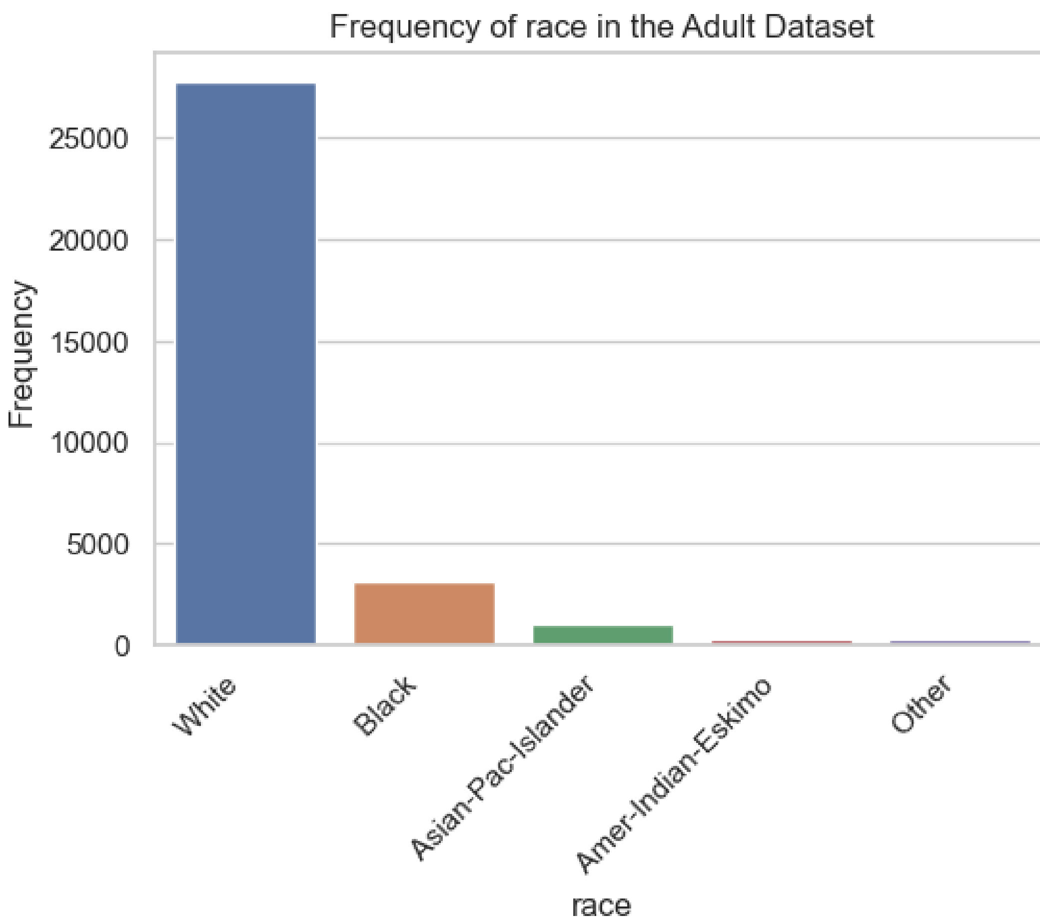

Frequency of workclass in the Adult Dataset
Frequency
20000
15000
10000
5000
0
Self-emp-not-inc
Private
State-gov
Federal-gov
Local-gov
?
Self-emp-inc
Without-pay
Never-worked
workclass

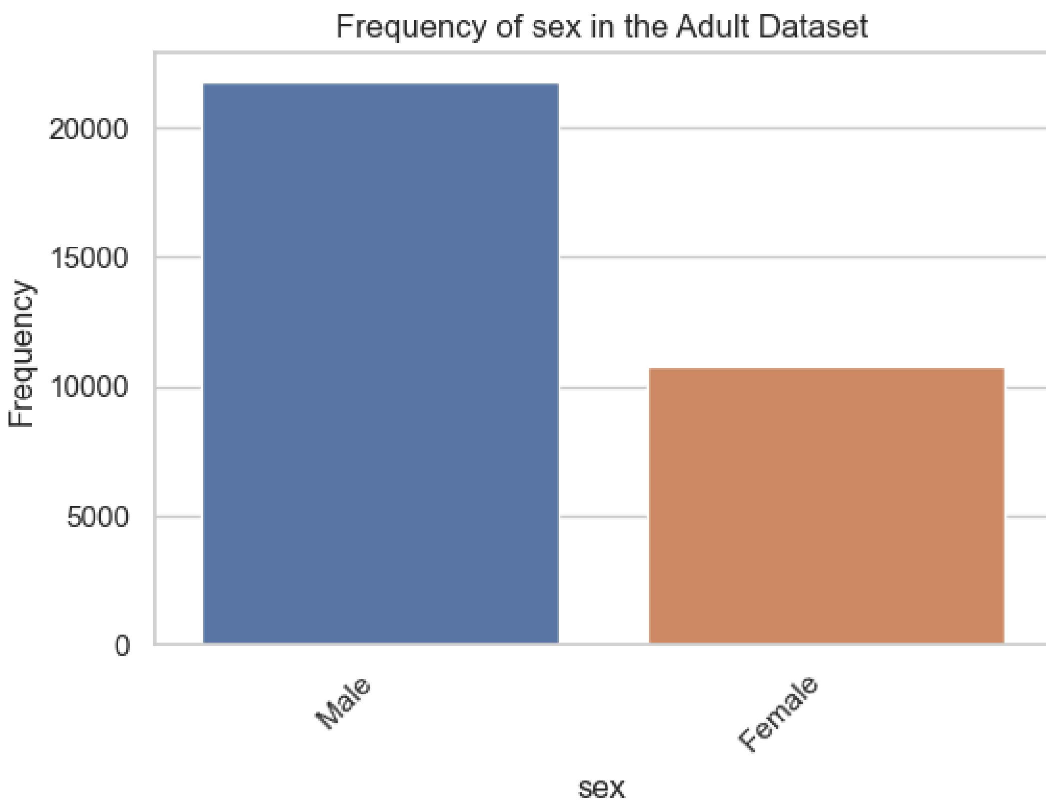

Frequency of marital_status in the Adult Dataset
Frequency
14000
12000
10000
8000
6000
4000
2000
0
Married-civ-spouse
Divorced
Married-spouse-absent
Never-married
Separated
Married-AF-spouse
Widowed
marital_status


Frequency of occupation in the Adult Dataset
Frequency
4000
3500
3000
2500
2000
1500
1000
500
0
Exec-managerial
Handlers-cleaners
Prof-specialty
Other-service
Adm-clerical
Sales
Craft-repair
Transport-moving
Farming-fishing
Machine-op-inspct
Tech-support
?
Protective-serv
Armed-Forces
Priv-house-serv
occupation



Frequency of relationship in the Adult Dataset
Frequency
12000
10000
8000
6000
4000
2000
0
Husband
Not-in-family
Wife
Own-child
Unmarried
Other-relative
relationship


Frequency of race in the Adult Dataset
Frequency
25000
20000
15000
10000
5000
0
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
race



[44]:
```
df.dtypes
```

[44]:
```
age                int64
workclass         object
fnlwgt             int64
education         object
education_num      int64
marital_status    object
occupation        object
relationship      object
race              object
sex               object
capital_gain       int64
capital_loss       int64
hours_per_week     int64
native_country    object
income            object
dtype: object
```

[45]:
```
df.head()
```

[45]:
```
   age          workclass  fnlwgt  education  education_num  \
0   50   Self-emp-not-inc   83311  Bachelors             13
1   38            Private  215646    HS-grad              9
2   53            Private  234721       11th              7
3   28            Private  338409  Bachelors             13
4   37            Private  284582    Masters             14

       marital_status         occupation   relationship   race     sex  \
0  Married-civ-spouse    Exec-managerial        Husband  White    Male
1            Divorced  Handlers-cleaners  Not-in-family  White    Male
2  Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
3  Married-civ-spouse     Prof-specialty           Wife  Black  Female
4  Married-civ-spouse    Exec-managerial           Wife  White  Female

   capital_gain  capital_loss  hours_per_week  native_country  income
0             0             0              13   United-States   <=50K
1             0             0              40   United-States   <=50K
2             0             0              40   United-States   <=50K
3             0             0              40            Cuba   <=50K
4             0             0              40   United-States   <=50K
```

8.0.3 Data Pre-preprocessing & Exploratory Data Analysis

Showing how much unique values are present in the dataset.

[46]:
```python
for i in df.columns:
    print(f"\n{i}")
    print(df[i].unique())
    print("Count of Unique Values: ", df[i].unique().shape[0])
```

```
age
[50 38 53 28 37 49 52 31 42 30 23 32 40 34 25 43 54 35 59 56 19 39 20 45
 22 48 21 24 57 44 41 29 18 47 46 36 79 27 67 33 76 17 55 61 70 64 71 68
 66 51 58 26 60 90 75 65 77 62 63 80 72 74 69 73 81 78 88 82 83 84 85 86
 87]
Count of Unique Values:  73

workclass
[' Self-emp-not-inc' ' Private' ' State-gov' ' Federal-gov' ' Local-gov'
 ' ?' ' Self-emp-inc' ' Without-pay' ' Never-worked']
Count of Unique Values:  9

fnlwgt
[ 83311 215646 234721 ...  34066  84661 257302]
Count of Unique Values:  21647

education
```

```
[' Bachelors' ' HS-grad' ' 11th' ' Masters' ' 9th' ' Some-college'
 ' Assoc-acdm' ' Assoc-voc' ' 7th-8th' ' Doctorate' ' Prof-school'
 ' 5th-6th' ' 10th' ' 1st-4th' ' Preschool' ' 12th']
Count of Unique Values:  16

education_num
[13  9  7 14  5 10 12 11  4 16 15  3  6  2  1  8]
Count of Unique Values:  16

marital_status
[' Married-civ-spouse' ' Divorced' ' Married-spouse-absent'
 ' Never-married' ' Separated' ' Married-AF-spouse' ' Widowed']
Count of Unique Values:  7

occupation
[' Exec-managerial' ' Handlers-cleaners' ' Prof-specialty'
 ' Other-service' ' Adm-clerical' ' Sales' ' Craft-repair'
 ' Transport-moving' ' Farming-fishing' ' Machine-op-inspct'
 ' Tech-support' ' ?' ' Protective-serv' ' Armed-Forces'
 ' Priv-house-serv']
Count of Unique Values:  15

relationship
[' Husband' ' Not-in-family' ' Wife' ' Own-child' ' Unmarried'
 ' Other-relative']
Count of Unique Values:  6

race
[' White' ' Black' ' Asian-Pac-Islander' ' Amer-Indian-Eskimo' ' Other']
Count of Unique Values:  5

sex
[' Male' ' Female']
Count of Unique Values:  2

capital_gain
[    0 14084  5178  5013  2407 14344 15024  7688 34095  4064  4386  7298
  1409  3674  1055  3464  2050  2176  2174   594 20051  6849  4101  1111
  8614  3411  2597 25236  4650  9386  2463  3103 10605  2964  3325  2580
  3471  4865 99999  6514  1471  2329  2105  2885 25124 10520  2202  2961
 27828  6767  2228  1506 13550  2635  5556  4787  3781  3137  3818  3942
   914   401  2829  2977  4934  2062  2354  5455 15020  1424  3273 22040
  4416  3908 10566   991  4931  1086  7430  6497   114  7896  2346  3418
  3432  2907  1151  2414  2290 15831 41310  4508  2538  3456  6418  1848
  3887  5721  9562  1455  2036  1831 11678  2936  2993  7443  6360  1797
  1173  4687  6723  2009  6097  2653  1639 18481  7978  2387  5060]
Count of Unique Values:  119
```

```
capital_loss
[   0 2042 1408 1902 1573 1887 1719 1762 1564 2179 1816 1980 1977 1876
 1340 2206 1741 1485 2339 2415 1380 1721 2051 2377 1669 2352 1672  653
 2392 1504 2001 1590 1651 1628 1848 1740 2002 1579 2258 1602  419 2547
 2174 2205 1726 2444 1138 2238  625  213 1539  880 1668 1092 1594 3004
 2231 1844  810 2824 2559 2057 1974  974 2149 1825 1735 1258 2129 2603
 2282  323 4356 2246 1617 1648 2489 3770 1755 3683 2267 2080 2457  155
 3900 2201 1944 2467 2163 2754 2472 1411]
Count of Unique Values:  92

hours_per_week
[13 40 16 45 50 80 30 35 60 20 52 44 15 25 38 43 55 48 58 32 70  2 22 56
 41 28 36 24 46 42 12 65  1 10 34 75 98 33 54  8  6 64 19 18 72  5  9 47
 37 21 26 14  4 59  7 99 53 39 62 57 78 90 66 11 49 84  3 17 68 27 85 31
 51 77 63 23 87 88 73 89 97 94 29 96 67 82 86 91 81 76 92 61 74 95]
Count of Unique Values:  94

native_country
[' United-States' ' Cuba' ' Jamaica' ' India' ' ?' ' Mexico' ' South'
 ' Puerto-Rico' ' Honduras' ' England' ' Canada' ' Germany' ' Iran'
 ' Philippines' ' Italy' ' Poland' ' Columbia' ' Cambodia' ' Thailand'
 ' Ecuador' ' Laos' ' Taiwan' ' Haiti' ' Portugal' ' Dominican-Republic'
 ' El-Salvador' ' France' ' Guatemala' ' China' ' Japan' ' Yugoslavia'
 ' Peru' ' Outlying-US(Guam-USVI-etc)' ' Scotland' ' Trinadad&Tobago'
 ' Greece' ' Nicaragua' ' Vietnam' ' Hong' ' Ireland' ' Hungary'
 ' Holand-Netherlands']
Count of Unique Values:  42

income
[' <=50K' ' >50K']
Count of Unique Values:  2
```

[47]:
```
# Listing all the Header from the dataset and listing the number of unique values
categorical = [var for var in df.columns if df[var].dtype=='O']
for i in df.columns:
    print(i, df[i].unique().shape[0])
```

```
age 73
workclass 9
fnlwgt 21647
education 16
education_num 16
marital_status 7
occupation 15
relationship 6
race 5
sex 2
capital_gain 119
```

```
capital_loss 92
hours_per_week 94
native_country 42
income 2
```

[48]:
```
# dropping all the columns except the colluns to be worked upon
drop_col = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',␣
↪'marital_status', \
            'occupation', 'relationship', 'capital_gain', 'capital_loss',␣
↪'hours_per_week', 'native_country']
df = df.drop(columns = drop_col)
df
```

[48]:
```
        race     sex  income
0      White    Male   <=50K
1      White    Male   <=50K
2      Black    Male   <=50K
3      Black  Female   <=50K
4      White  Female   <=50K
...      ...     ...     ...
32555  White  Female   <=50K
32556  White    Male    >50K
32557  White  Female   <=50K
32558  White    Male   <=50K
32559  White  Female    >50K

[32560 rows x 3 columns]
```

Check whether there are any null values, and if there are any then remove it.

[49]:
```
df.isnull().sum()
```

[49]:
```
race      0
sex       0
income    0
dtype: int64
```

Converting categorical columns to One-hot encoding

[50]:
```
df = pd.get_dummies(df, columns=['race','sex'])
df = pd.get_dummies(df, columns=['income'], drop_first=True)
```

[51]:
```
df
```

[51]:
```
      race_ Amer-Indian-Eskimo  race_ Asian-Pac-Islander  race_ Black  \
0                        False                      False        False
1                        False                      False        False
2                        False                      False         True
3                        False                      False         True
```

```
4                          False                      False         False
...                          ...                        ...           ...
32555                      False                      False         False
32556                      False                      False         False
32557                      False                      False         False
32558                      False                      False         False
32559                      False                      False         False

       race_ Other  race_ White  sex_ Female  sex_ Male  income_ >50K
0            False         True        False       True         False
1            False         True        False       True         False
2            False        False        False       True         False
3            False        False         True      False         False
4            False         True         True      False         False
...            ...          ...          ...        ...           ...
32555        False         True         True      False         False
32556        False         True        False       True          True
32557        False         True         True      False         False
32558        False         True        False       True         False
32559        False         True         True      False          True

[32560 rows x 8 columns]
```

8.0.4 Bayes' Theorem

- Bayes theorem for single-feature:

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)}$$

- Bayes theorem for multi-feature:

$$P(y|X_1, X_2, ..., X_n) = \frac{P(X_1, X_2, ..., X_n|y) \cdot P(y)}{P(X_1, X_2, ..., X_n)}$$

[52]:
```
df.columns
```

[52]:
```
Index(['race_ Amer-Indian-Eskimo', 'race_ Asian-Pac-Islander', 'race_ Black',
       'race_ Other', 'race_ White', 'sex_ Female', 'sex_ Male',
       'income_ >50K'],
      dtype='object')
```

8.0.5 Single-feature Bayes theorem

Let, X = ['sex_Male'], and y = ['income_ >50K']

We want to find the probability of having income > 50K if he is a Male, $P(y|X)$

[53]:
```
# Bayes theorem for single-feature
```

```
prob_X_given_y = (df[(df["sex_ Male"]==True) & (df["income_ >50K"]==True)].
 ↪shape[0])/(df[df["income_ >50K"]==True].shape[0])
prob_y = df[df["income_ >50K"]==True].shape[0]/df.shape[0]
prob_X = df[df["sex_ Male"]==True].shape[0]/df.shape[0]

print("Prob_X_given_y :", prob_X_given_y)
print("Prob_y         :", prob_y)
print("Prob_X         :", prob_X)
```

```
Prob_X_given_y : 0.8496365259533223
Prob_y         : 0.24081695331695332
Prob_X         : 0.6691953316953317
```

[54]:
```
prob_y_given_X = prob_X_given_y*prob_y/prob_X

print("Prob_y_given_X :", prob_y_given_X)
```

```
Prob_y_given_X : 0.3057506081050071
```

8.0.6 Multi-feature Bayes theorem

Let, $X_1X_2 =$ ['sex_male', 'race_Black'], and $y =$ ['income_ >50']

We want to find the probabality of having income > 50K if he is a Male and also he is Black, $P(y|X_1X_2)$

[55]:
```
# Bayes theorem for multi-feature
prob_X_given_y = (df[df["sex_ Male"]==True][df["race_ Black"]==True][df["income_␣
 ↪>50K"]==True]).shape[0]\
                /(df[df["income_ >50K"]==True].shape[0])
prob_y = df[df["income_ >50K"]==True].shape[0]/df.shape[0]
prob_X = df[df["sex_ Male"]==True][df["race_ Black"]==True].shape[0]/df.shape[0]

print("Prob_X_given_y :", prob_X_given_y)
print("Prob_y         :", prob_y)
print("Prob_X         :", prob_X)
```

```
Prob_X_given_y : 0.037877821706414995
Prob_y         : 0.24081695331695332
Prob_X         : 0.04818796068796069

C:\Users\nihar\AppData\Local\Temp\ipykernel_16272\4138344757.py:2: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  prob_X_given_y = (df[df["sex_ Male"]==True][df["race_
Black"]==True][df["income_ >50K"]==True]).shape[0]\
C:\Users\nihar\AppData\Local\Temp\ipykernel_16272\4138344757.py:2: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  prob_X_given_y = (df[df["sex_ Male"]==True][df["race_
Black"]==True][df["income_ >50K"]==True]).shape[0]\
C:\Users\nihar\AppData\Local\Temp\ipykernel_16272\4138344757.py:5: UserWarning:
```

```
Boolean Series key will be reindexed to match DataFrame index.
  prob_X = df[df["sex_ Male"]==True][df["race_
Black"]==True].shape[0]/df.shape[0]
```

[56]:
```python
prob_y_given_X = prob_X_given_y*prob_y/prob_X

print("Prob_y_given_X :", prob_y_given_X)
```

```
Prob_y_given_X : 0.18929254302103246
```

8.0.7 Joint Distribution Analysis

Joint distribution refers to the probability distribution of two or more random variables. It describes how the probabilities are distributed across all possible combinations of values for the involved variables.

$f_{XY}(x, y) = P(X = x, Y = y)$

[57]:
```python
# List of races
races = ['race_ Amer-Indian-Eskimo', 'race_ Asian-Pac-Islander', 'race_ Black',␣
 ↪'race_ Other', 'race_ White']

# List of genders
genders = ['sex_ Female', 'sex_ Male']

counts={}

# Filter the DataFrame to include only rows where "income_ >50K" is 1
df_filtered = df[df['income_ >50K'] == 1]

# Calculate the joint probabilities
for race in races:
    for gender in genders:
        # Create a joint column
        joint_col = df_filtered[race] & df_filtered[gender]

        # Count the number of 1s in the joint column
        count = joint_col.sum()

        # Store the count in the dictionary
        counts[f'{race} & {gender}'] = count

# Print the counts
for key, value in counts.items():
    print(f'Count of {key} is {value}')
```

```
Count of race_ Amer-Indian-Eskimo & sex_ Female is 12
Count of race_ Amer-Indian-Eskimo & sex_ Male is 24
Count of race_ Asian-Pac-Islander & sex_ Female is 43
```

```
Count of race_ Asian-Pac-Islander & sex_ Male is 233
Count of race_ Black & sex_ Female is 90
Count of race_ Black & sex_ Male is 297
Count of race_ Other & sex_ Female is 6
Count of race_ Other & sex_ Male is 19
Count of race_ White & sex_ Female is 1028
Count of race_ White & sex_ Male is 6089
```

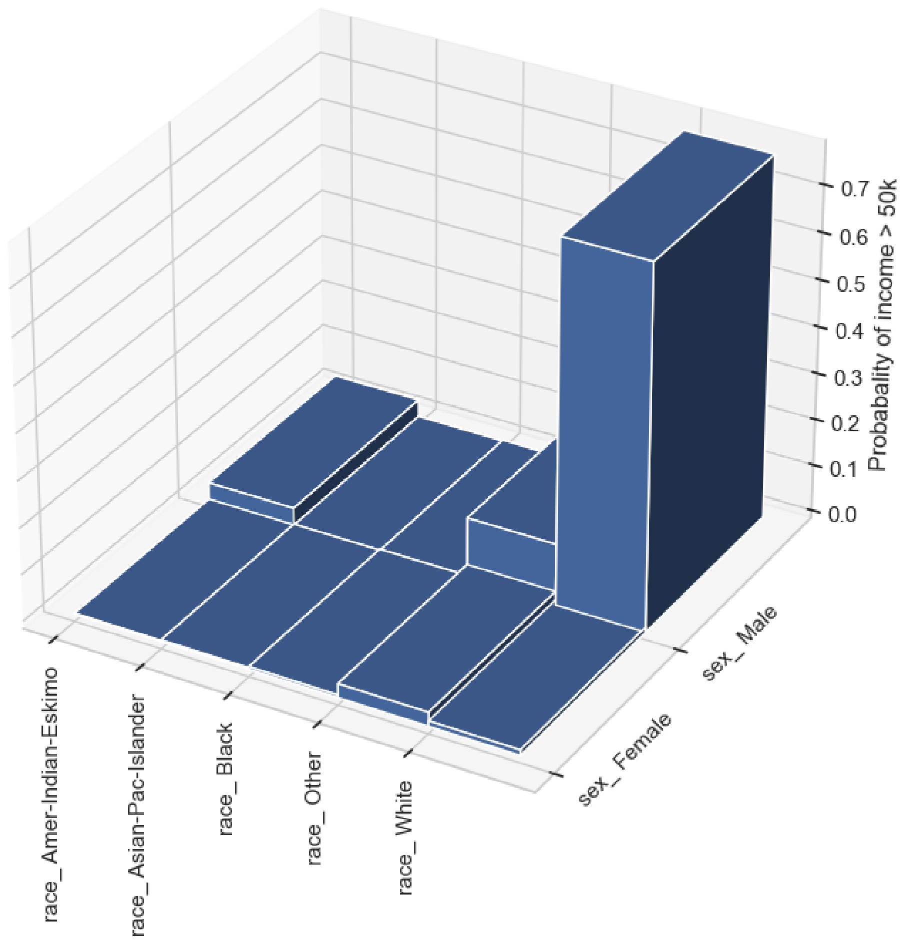
[58]:

```python
# Create a new figure
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
x,y,z = [],[],[]
races = ['race_ Amer-Indian-Eskimo', 'race_ Asian-Pac-Islander', 'race_ Black',␣
↪'race_ Other', 'race_ White']
genders = ['sex_ Female', 'sex_ Male']

# Populate the lists with the data from the counts dictionary
for i, race in enumerate(races):
    for j, gender in enumerate(genders):
        x.append(i)
        y.append(j)
        z.append(counts[f'{race} & {gender}'])

# Create a 3D bar plot
_x = np.arange(len(races))
_y = np.arange(len(genders))
_xx, _yy = np.meshgrid(_x, _y)
x, y = _xx.ravel(), _yy.ravel()

top = np.array(z)/np.sum(z)
bottom = np.zeros_like(top)
width = depth = 1

ax.bar3d(x, y, bottom, width, depth, top, shade=True)

ax.set_xticks(range(len(races)))
ax.set_yticks(range(len(genders)))
ax.set_xticklabels(races, rotation=90, ha='left')
ax.set_yticklabels(genders, rotation=45, ha='left', rotation_mode='anchor')
ax.set_zlabel('Probabality of income > 50k')
ax.dist = 12

# Show the plot
plt.title("Joint Probabality Distribution of Race and Sex")
plt.show()
```

```
C:\Users\nihar\AppData\Local\Temp\ipykernel_16272\3503899099.py:32:
MatplotlibDeprecationWarning: The dist attribute was deprecated in Matplotlib
```

```
3.6 and will be removed two minor releases later.
  ax.dist = 12
```

Joint Probabality Distribution of Race and Sex



As we can see our data is very biased

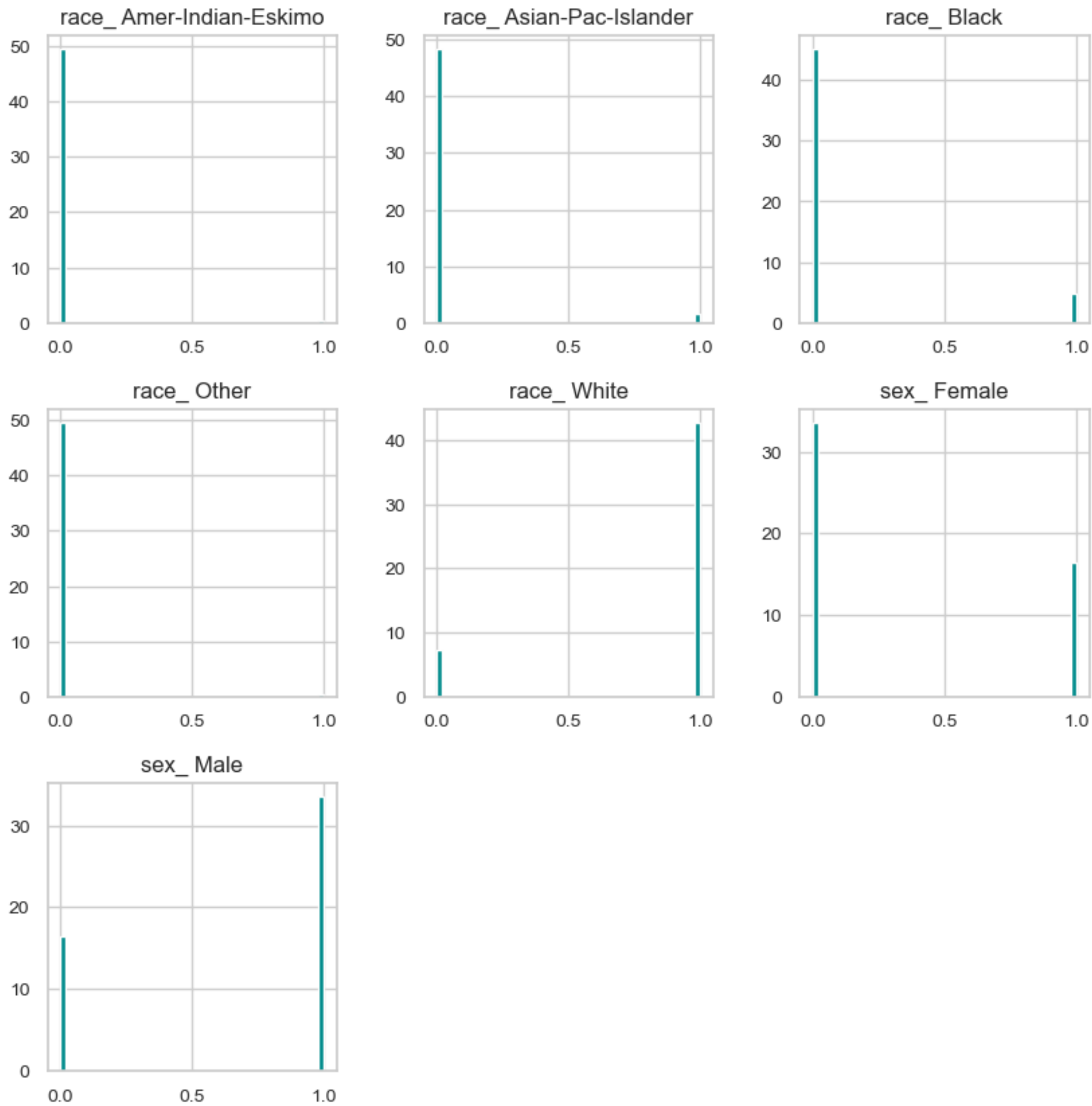
Data splitting into 'X_train', 'X_test', 'y_train', and 'y_test', with a train-test split of 70:30.

[59]:
```
y = df['income_ >50K']
X = df.drop(['income_ >50K'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 0)
```

[60]:
```python
X_train = X_train.astype(int)
X_train
```

[60]:
```
       race_ Amer-Indian-Eskimo  race_ Asian-Pac-Islander  race_ Black  \
20721                         0                         0            1
32097                         0                         0            0
25205                         0                         0            0
23491                         0                         0            0
12367                         0                         0            0
...                         ...                       ...          ...
13123                         0                         0            1
19648                         0                         0            1
9845                          0                         0            1
10799                         0                         0            0
2732                          0                         0            0

       race_ Other  race_ White  sex_ Female  sex_ Male
20721            0            0            0          1
32097            0            1            0          1
25205            0            1            1          0
23491            0            1            0          1
12367            0            1            0          1
...            ...          ...          ...        ...
13123            0            0            0          1
19648            0            0            1          0
9845             0            0            0          1
10799            0            1            1          0
2732             0            1            1          0

[22792 rows x 7 columns]
```

[61]:
```python
# Histogram depecting the Expectation of number of specific peoples belonging to␣
↪particular race or gender, based on the Income
ax = X_train.hist(figsize=(10, 10), bins=50, xlabelsize=10, ylabelsize=10,␣
↪color='darkcyan', density=True)
```



8.0.8 Modeling

[62]:
```python
# instantiate the model
cnb = CategoricalNB()

# fit the model
cnb.fit(X_train, y_train)
```

[62]: CategoricalNB()

[63]:
```python
y_pred = cnb.predict(X_test)
y_pred
```

[63]: array([False, False, False, ..., False, False, False])

[64]:
```
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.7631
```

[65]:
```
print('Training set accuracy: {:.4f}'.format(cnb.score(X_train, y_train)))
print('Test set accuracy: {:.4f}'.format(cnb.score(X_test, y_test)))
```

```
Training set accuracy: 0.7575
Test set accuracy: 0.7631
```

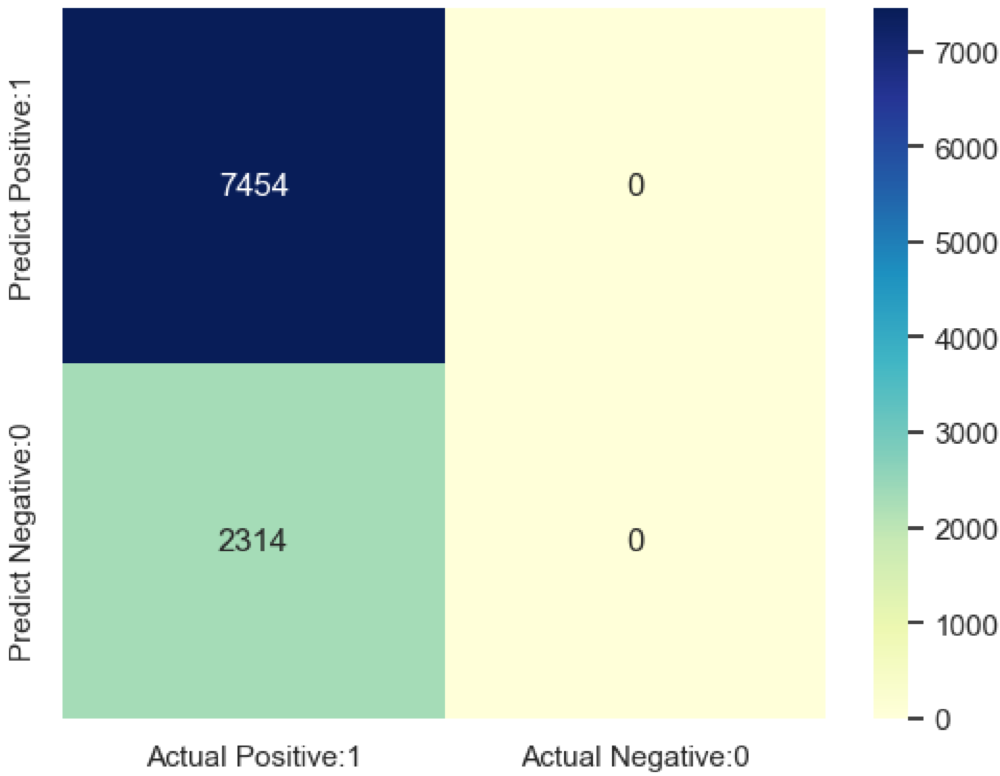
[66]:
```
cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[7454    0]
 [2314    0]]

True Positives(TP) =  7454

True Negatives(TN) =  0

False Positives(FP) =  0

False Negatives(FN) =  2314
```

8.0.9 Visualize confusion matrix with seaborn heatmap

[67]:
```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:
 ↪0'],
                                 index=['Predict Positive:1', 'Predict Negative:
 ↪0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

[67]: <Axes: >



[68]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       False       0.76      1.00      0.87      7454
        True       0.00      0.00      0.00      2314

    accuracy                           0.76      9768
   macro avg       0.38      0.50      0.43      9768
weighted avg       0.58      0.76      0.66      9768


C:\Users\nihar\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\nihar\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\nihar\anaconda3\Lib\site-
```

```
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

8.0.10 Factor Analysis

Factor Analysis is a statistical technique used to explore the underlying structure of a set of observed variables. It aims to identify and quantify the latent (unobservable) factors that may be influencing the observed variables.

$X = \Lambda F + U$

X : is the observed data matrix

Λ : is the factor loading matrix

F : is the matrix of latent factors

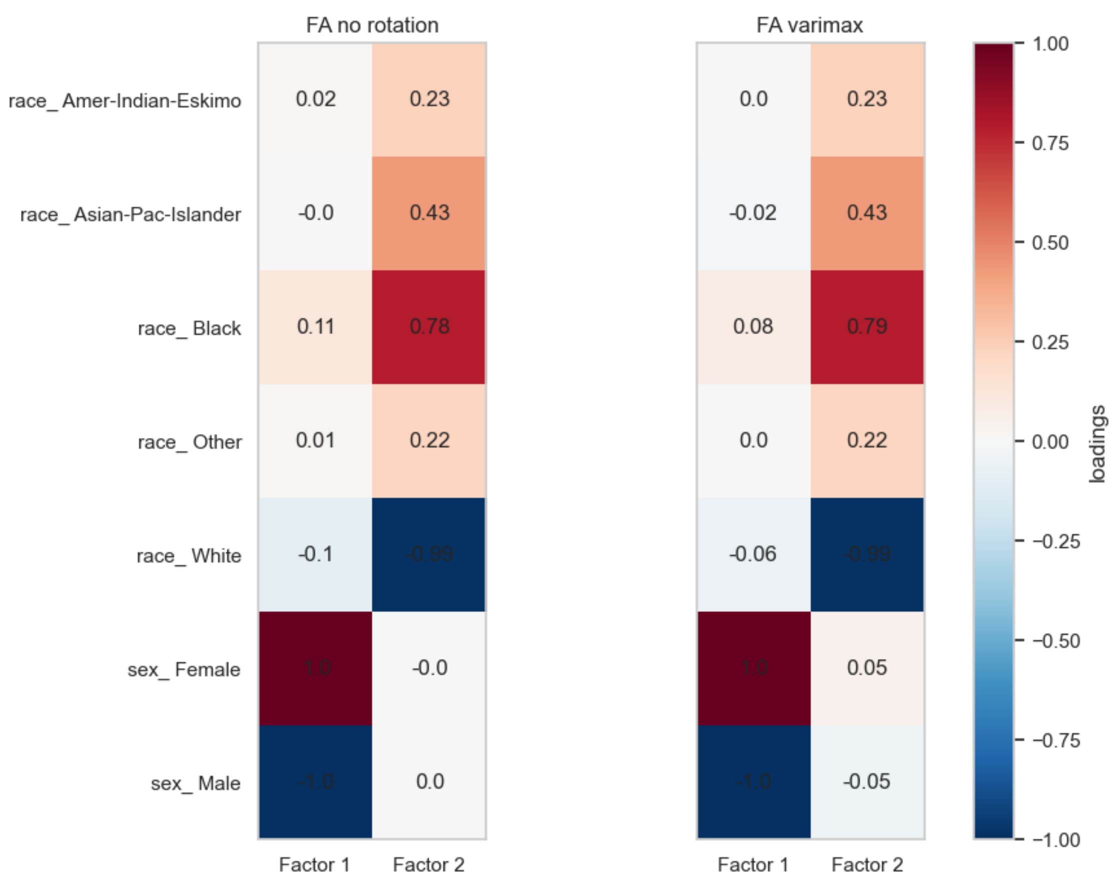
U : is the matrix of unique factors (errors)

[69]:
```python
df = X_train
df.head(5)
```

[69]:
```
       race_ Amer-Indian-Eskimo  race_ Asian-Pac-Islander  race_ Black  \
20721                         0                         0            1
32097                         0                         0            0
25205                         0                         0            0
23491                         0                         0            0
12367                         0                         0            0

       race_ Other  race_ White  sex_ Female  sex_ Male
20721            0            0            0          1
32097            0            1            0          1
25205            0            1            1          0
23491            0            1            0          1
12367            0            1            0          1
```

[70]:
```python
X = StandardScaler().fit_transform(df)
factors = 2

fas = [("FA no rotation", FactorAnalysis(n_components = factors)),
       ("FA varimax", FactorAnalysis(n_components = factors,␣
 ↪rotation="varimax"))]

fig, axes = plt.subplots(ncols=len(fas), figsize=(10, 8))

for ax, (title, fa) in zip(axes, fas):
    fa = fa.fit(X)
    factor_matrix = fa.components_.T
```

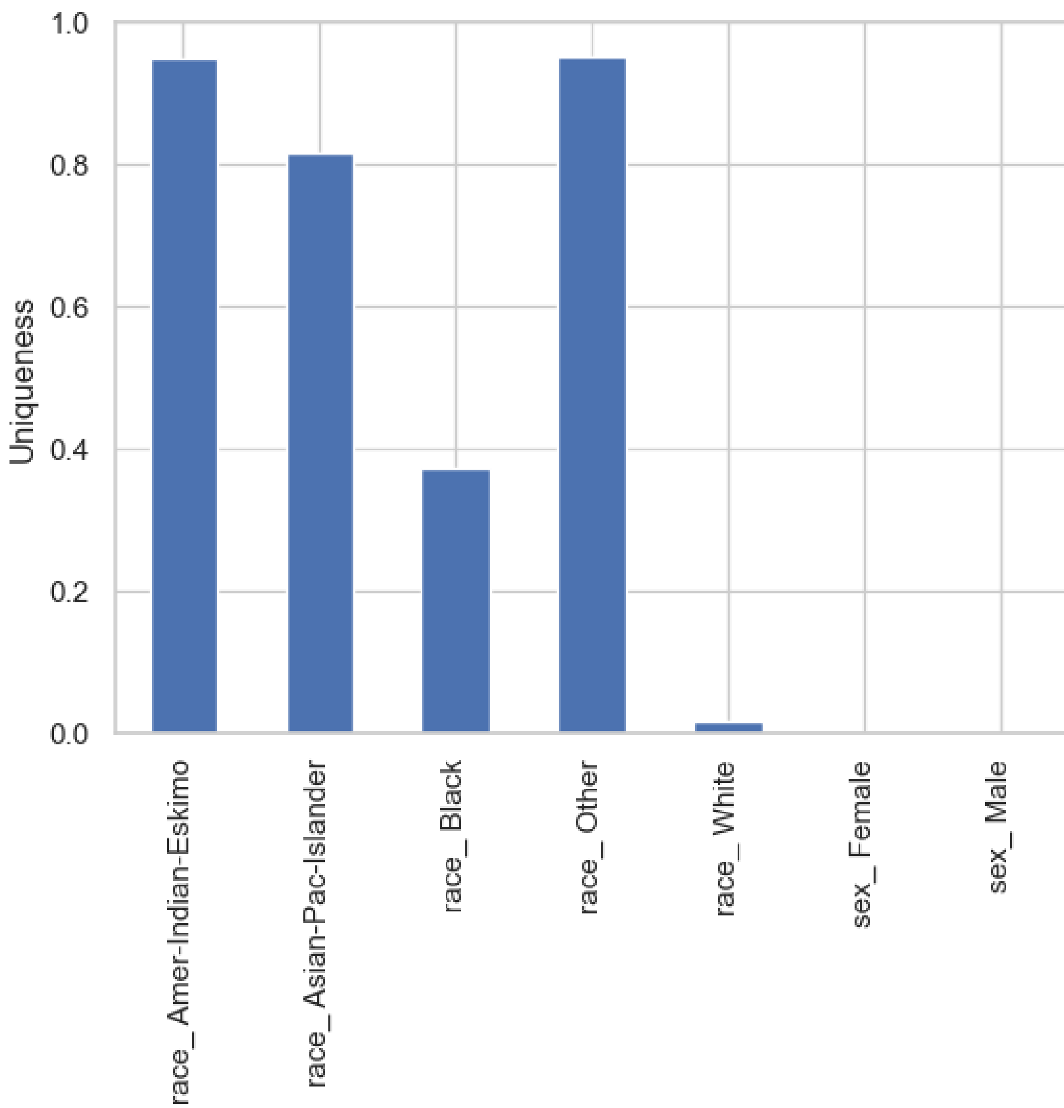
```
    im = ax.imshow(factor_matrix, cmap="RdBu_r", vmax=1, vmin=-1)
    for (i,j), z in np.ndenumerate(factor_matrix):
        ax.text(j, i, str(z.round(2)), ha="center", va="center")
    ax.set_yticks(np.arange(len(df.columns)))
    if ax.get_subplotspec().is_first_col():
        ax.set_yticklabels(df.columns)
    else:
        ax.set_yticklabels([])
    ax.set_title(title)
    ax.set_xticks([0, 1])
    ax.set_xticklabels(["Factor 1", "Factor 2"])
    ax.grid(False)
    plt.plot()

cb = fig.colorbar(im, ax=axes, location='right', label="loadings")
plt.show()
```



[71]:
```
# Uniqueness
fa = FactorAnalysis(n_components = 2, rotation="varimax")
```

```
fa.fit(X)
uniqueness = Series(fa.noise_variance_, index=df.columns)
uniqueness.plot(
    kind="bar",
    ylabel="Uniqueness"
)
```

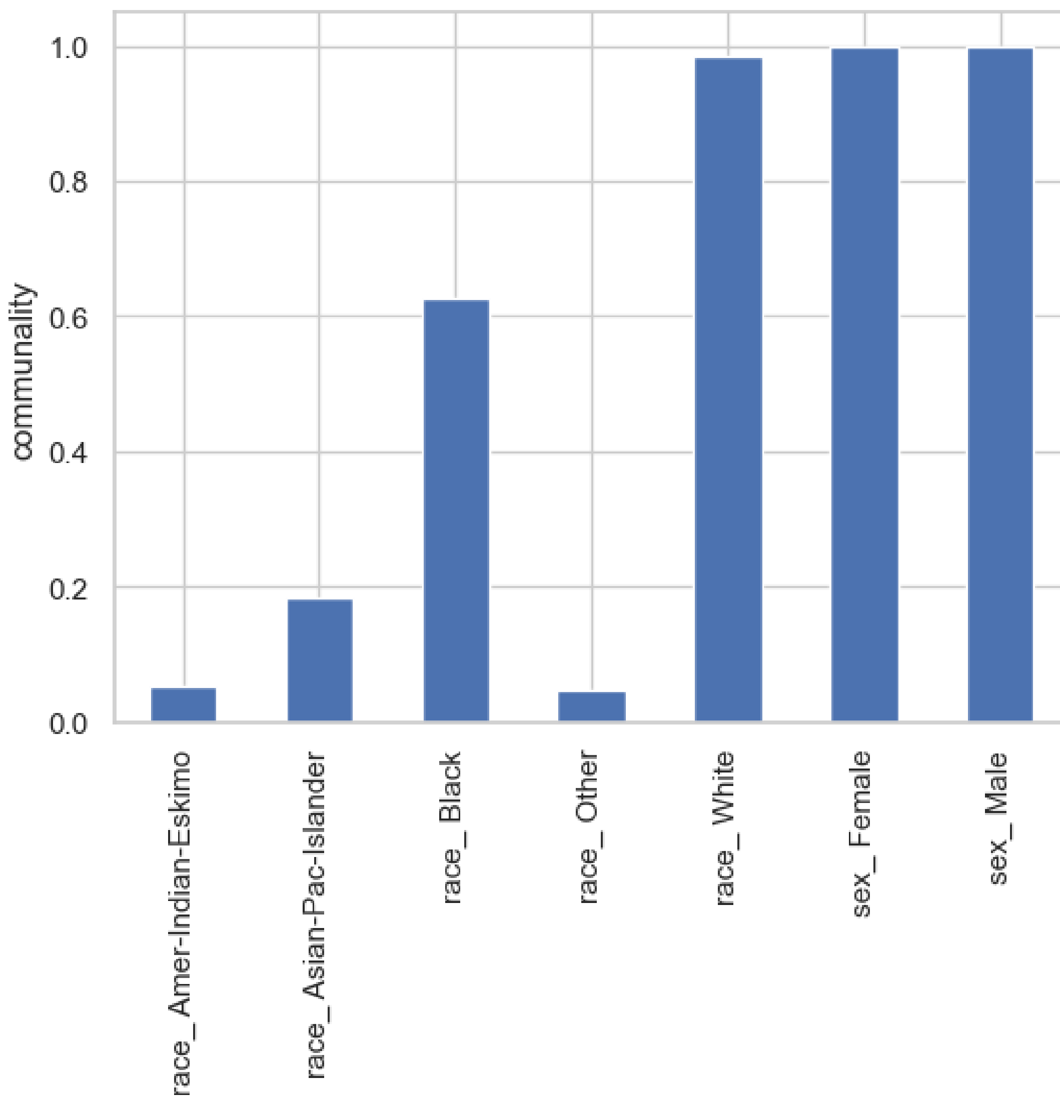
[71]: <Axes: ylabel='Uniqueness'>



[72]:
```
# Communality
communality = Series(np.square(fa.components_.T).sum(axis=1), index=df.columns)
communality.plot(
```

```
    kind="bar",
    ylabel="communality"
)
```

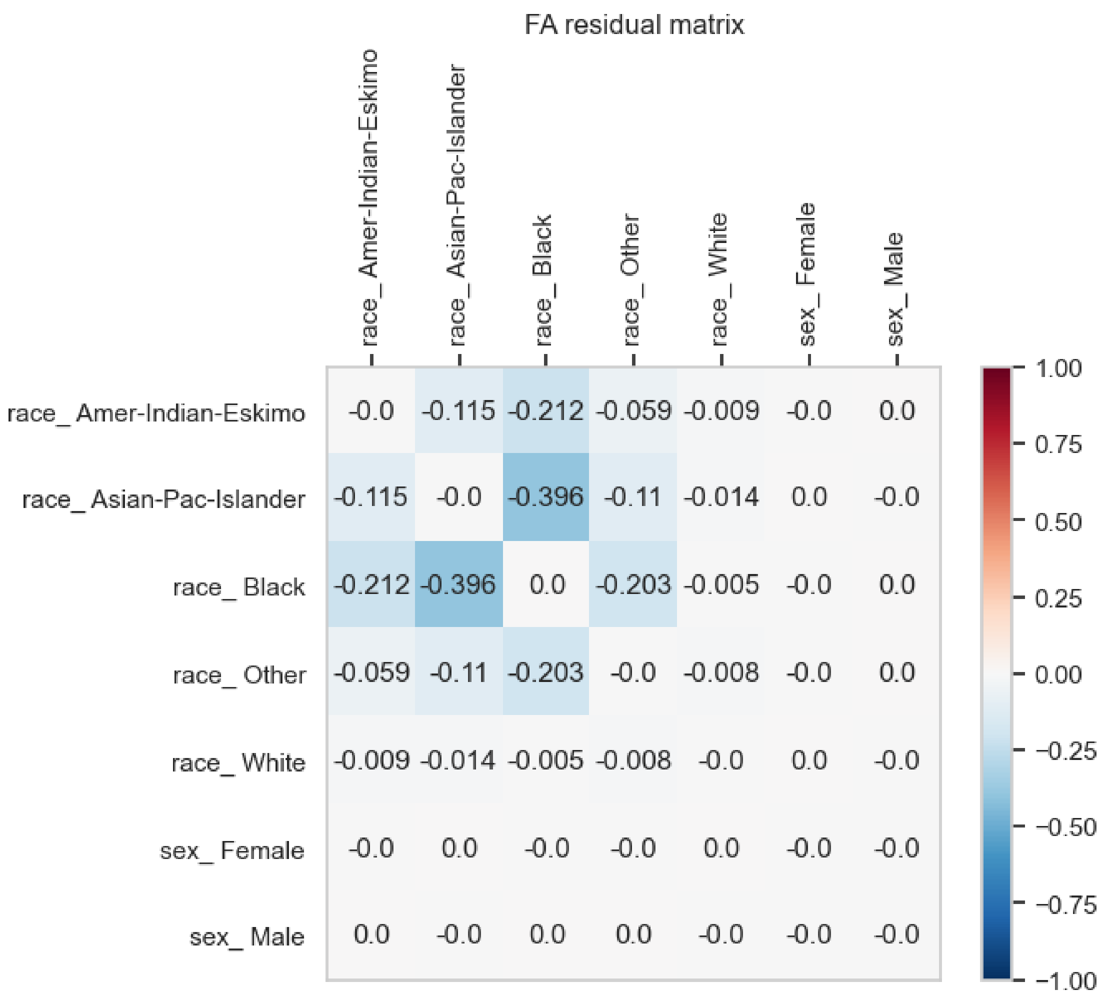
[72]: <Axes: ylabel='communality'>



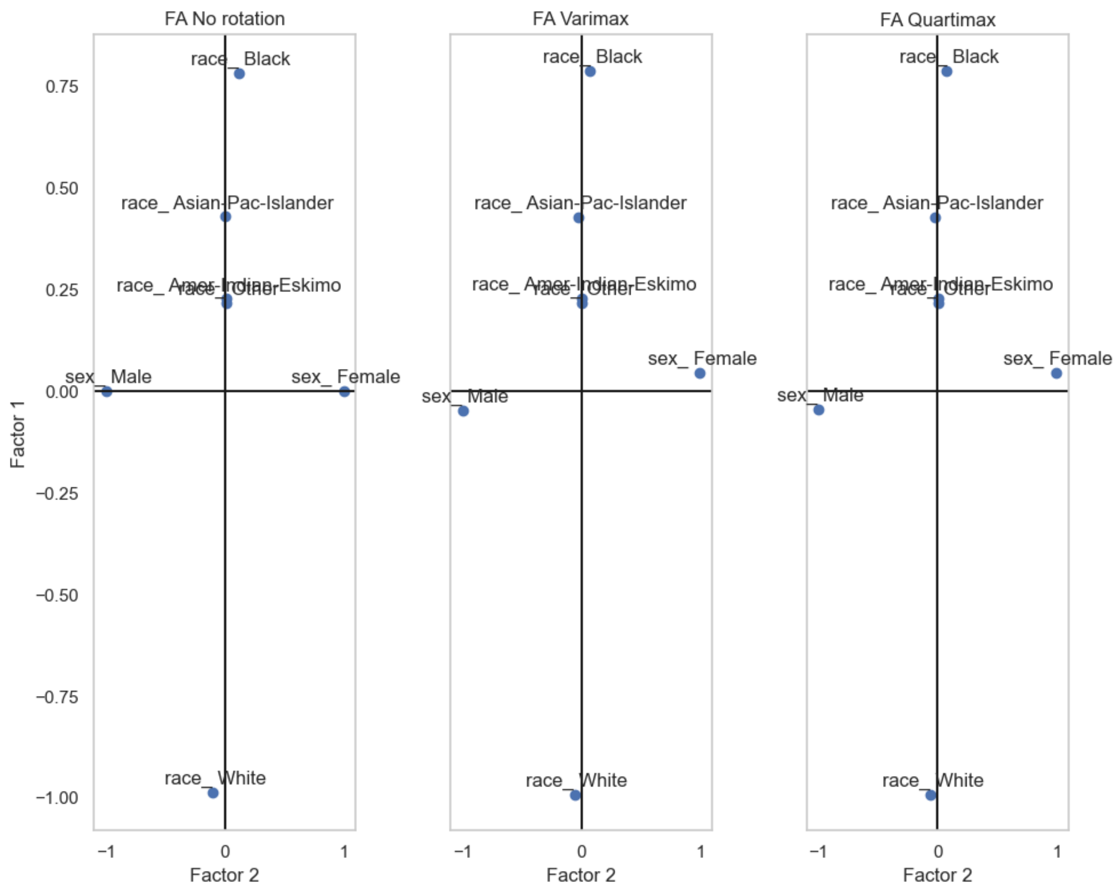
[73]:
```
# calculating the residual FA matrix
lambda_ = fa.components_
psi = np.diag(uniqueness)
s = np.corrcoef(np.transpose(X))
sigma = np.matmul(lambda_.T, lambda_) + psi
residuals = (s - sigma)
```

[74]:
```python
ax = plt.axes()
im = ax.imshow(residuals, cmap="RdBu_r", vmin=-1, vmax=1)
ax.tick_params(axis="x", bottom=False, labelbottom=False, top=True,␣
 ↪labeltop=True)
ax.set_xticks(range(df.columns.shape[0]))
ax.set_xticklabels(df.columns, rotation=90)
ax.set_yticks(range(df.columns.shape[0]))
ax.set_yticklabels(df.columns)
for (i,j), z in np.ndenumerate(residuals):
    ax.text(j, i, str(z.round(3)), ha="center", va="center")

fig.colorbar(im, ax=ax, location='right')
ax.set_title("FA residual matrix")
ax.grid(False)
plt.plot()
```

[74]: []



[75]:
```
methods = [
    ("FA No rotation", FactorAnalysis(2,)),
    ("FA Varimax", FactorAnalysis(2, rotation="varimax")),
    ("FA Quartimax", FactorAnalysis(2, rotation="quartimax")),
]
fig, axes = plt.subplots(ncols=3, figsize=(10, 8), sharex=True, sharey=True)

for ax, (method, fa) in zip(axes, methods):
    fa = fa.fit(X)

    components = fa.components_

    vmax = np.abs(components).max()
    ax.scatter(components[0,:], components[1, :])
    ax.axhline(0, -1, 1, color='k')
    ax.axvline(0, -1, 1, color='k')
    for i,j, z in zip(components[0, :], components[1, :], df.columns):
        ax.text(i+.02, j+.02, str(z), ha="center")
    ax.set_title(str(method))
    if ax.get_subplotspec().is_first_col():
        ax.set_ylabel("Factor 1")
    ax.set_xlabel("Factor 2")
    ax.grid(False)
    ax.dist = 100

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


FA No rotation
FA Varimax
FA Quartimax
race_ Black
race_ Asian-Pac-Islander
race_ Amer-Indian-Eskimo
race_ Other
sex_ Male
sex_ Female
race_ White
Factor 1
Factor 2
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1
0
1

[ ]: