Demographic Factors Impacting Income

Subhabrata Ganguli, Matt Purkeypile, Aryaz Zomorodi

University of San Diego

Abstract

An examination of US Census data to determine if there are demographic factors that influence income. Data was selected from the UC Irvine Machine Learning Repository, which has been utilized for multiple studies.

Demographic Factors Impacting Income

Wealth has a profound impact on one’s life. It dictates living in poverty to jet-setting around the globe, and everything in between. For most, the primary driver of wealth is their income. (Obviously, there are other drivers such as wealth through inheritance.) This paper aims to examine what demographic factors influence wealth.

Within the United States, the US Census Bureau is an excellent source of demographic information. As they state (*About the Bureau*, 2023), “The Census Bureau's mission is to serve as the nation's leading provider of quality data about its people and economy.” Therefore, we used data from the Bureau to see if we could determine what demographic factors influence income, using a variety of models.

Data Selection

The selection of the data set focused on a set that would be relatively easy to understand, which would allow us to focus on the analysis and models- not understanding the underlying data itself. We also wanted a data source that was cited by other studies, as evidence of its use and to have other academic resources to consult.

The data for this project consists of adult data taken from a relatively clean pull of census data, known as “census income” (Becker & Kohavi, n.d.). Highlights of the data set, as described at the source:

- Description: Predict whether income exceeds $50K/yr based on census data.
- Number of variables: 14
- Size of data set: 48,842 (32,561 main set, 16,281 test set)
- Missing data: yes
- Dataset Characteristics: Multivariate
- Subject Area: Social Science
- Associated Tasks: Classification

- Feature Type: Categorical, Integer

With this dataset, we then moved forward to see if we could determine which demographic factors would influence income.

Data Cleaning and Preparation

As previously stated, a relatively clean data set was intentionally selected. Data cleaning and preparation itself can be an entire project - and was not the focus of this exercise.

As an example, United States Security and Exchange Commission (SEC) form 13-F was submitted and published as plain text up until about a decade ago (US Securities and Exchange Commission, 2023). These filings essentially contain a list of holdings made by investment managers. This presented consumers of the data with significant challenges being in plain text, as no formats had to be adhered to. In these plain text forms, holdings were identified by their CUSIP- which is a 9-digit identifier. These are broken into 3 parts, with the 9th digit being a check digit (*What Is a CUSIP Number, and How Do I Find a Stock or Bond CUSIP?*, n.d.). While it sounds simple enough, there are challenges in just identifying the CUSIP in the text:

- The CUSIP may be broken into the three parts: “XXXXXX YY Z”
- Distinguishing the CUSIP from a 8 or 9 digit word: “XXXXXXYYZ” vs “ABCDEFGHI”.
- The check digit could give some confidence on a CUSIP versus other text, but was not always included.
- The CUSIP could be broke across multiple lines:

  XXXXXX

  YY

  Z
- …and many other variations

As mentioned, the data set we selected was relatively clean and that selection was intentional. Missing values in the data set were represented by “?”, and we first replaced those with “not a number” (NaN). We were then able to complete our cleanup by removing rows that had those NaN values. This left us with a completely clean data set to analyze.

Exploratory Data Analysis

We originally envisioned that the project would be centered around how income factored into education. More specifically, we wanted to see how education factored into hourly income. This was to account for variations in hours worked. For example, maybe a small business owner works long hours to keep the business going. On the other hand, maybe that business was up in running and required minimal oversight- leading to low hours. Another possibility we considered was a low-income person not being fully employed and working part-time. Hence, looking at the hourly income instead of just the income to account for these variations. Therefore, the original title of this paper was “Education Versus Hourly Income”.

As we did our initial data analysis, it became clear that more factored into income besides education alone. In hindsight, this shouldn’t have been a surprise. This can be seen by some of the exploratory data analysis we did, as outlined in the following figures.

Figure 1

Income by education

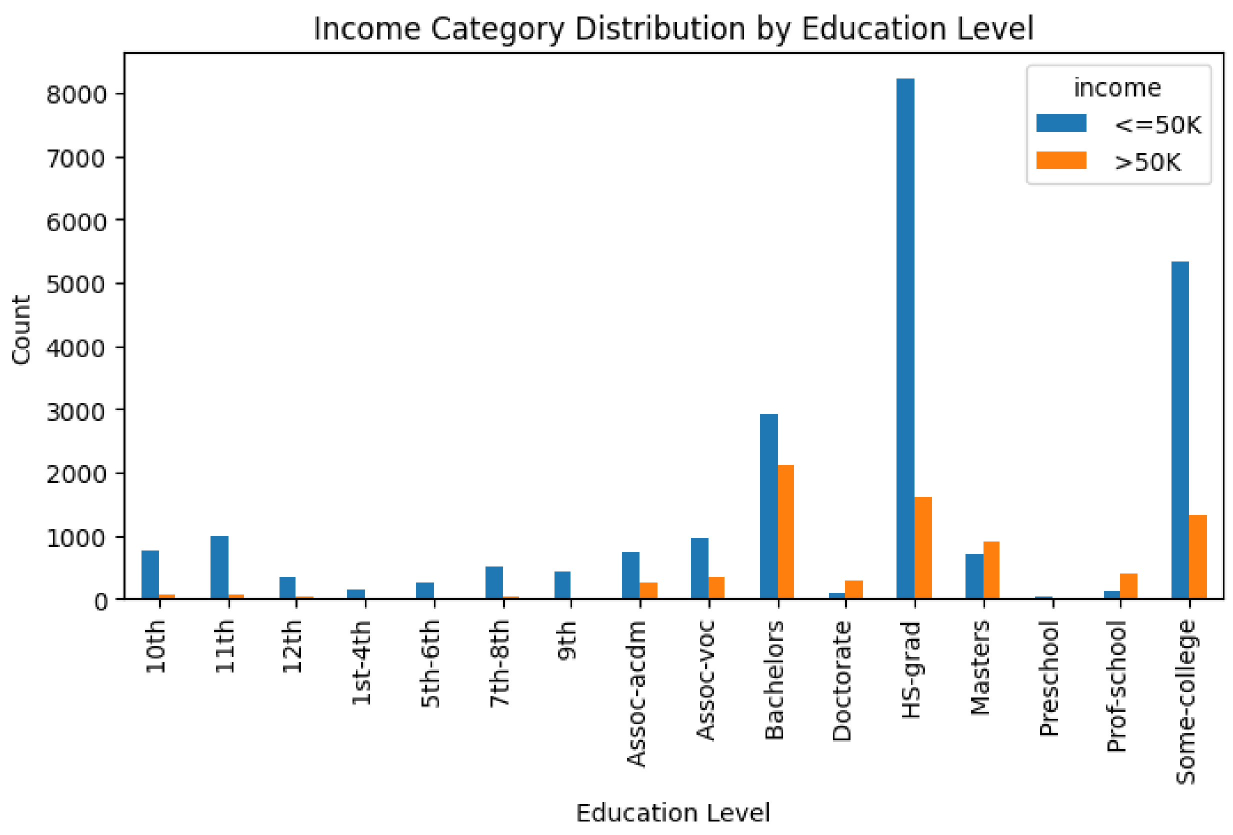


Figure 2

Income by race

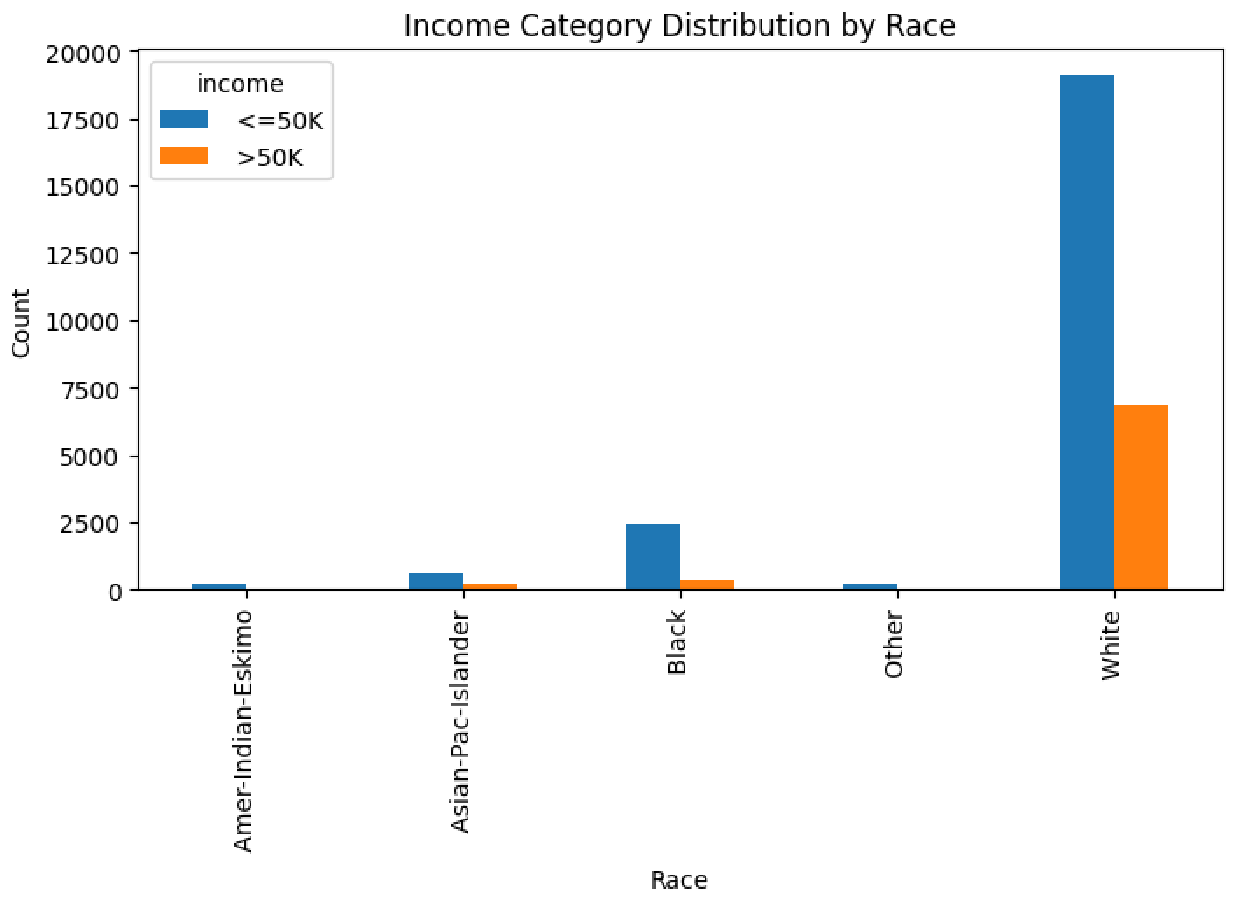


Figure 3

Income by occupation

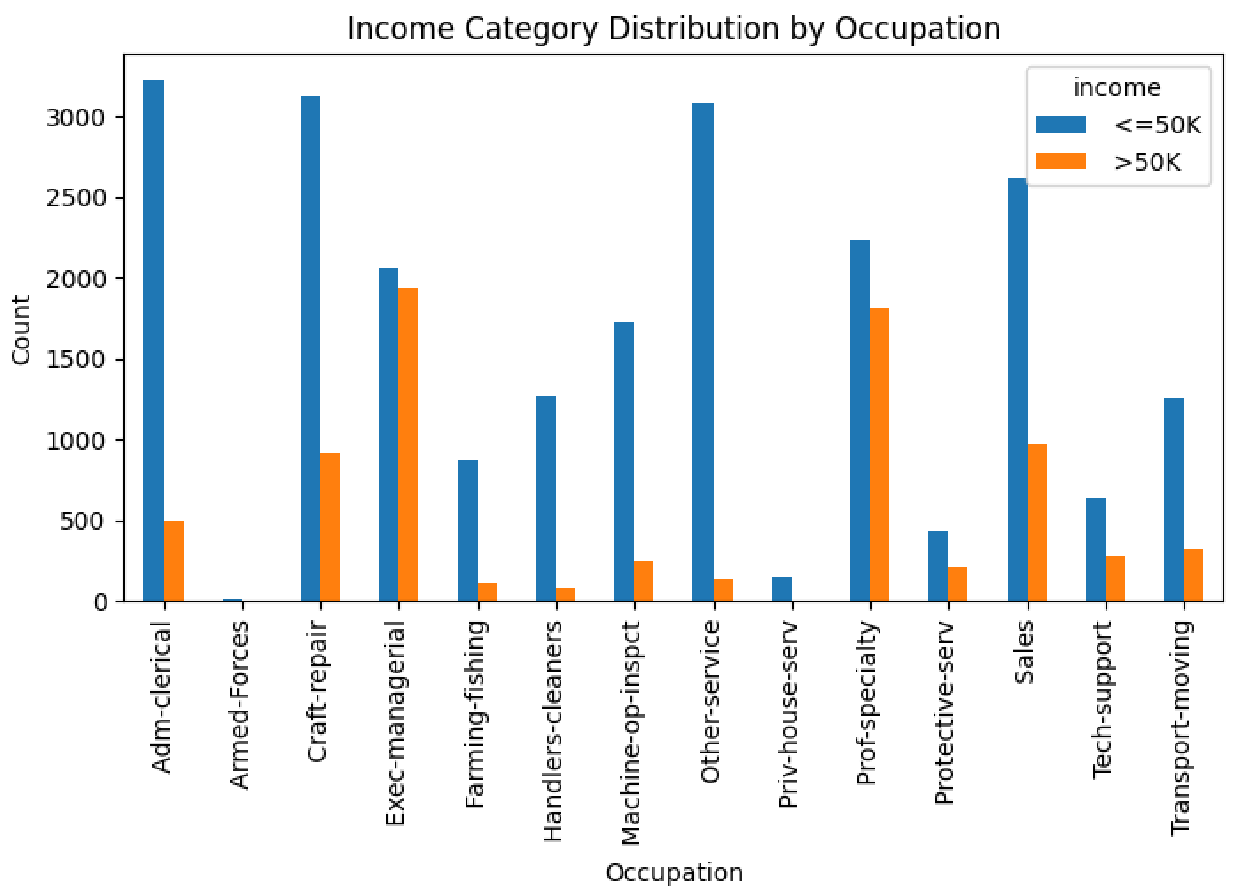


Figure 4

Income by age category

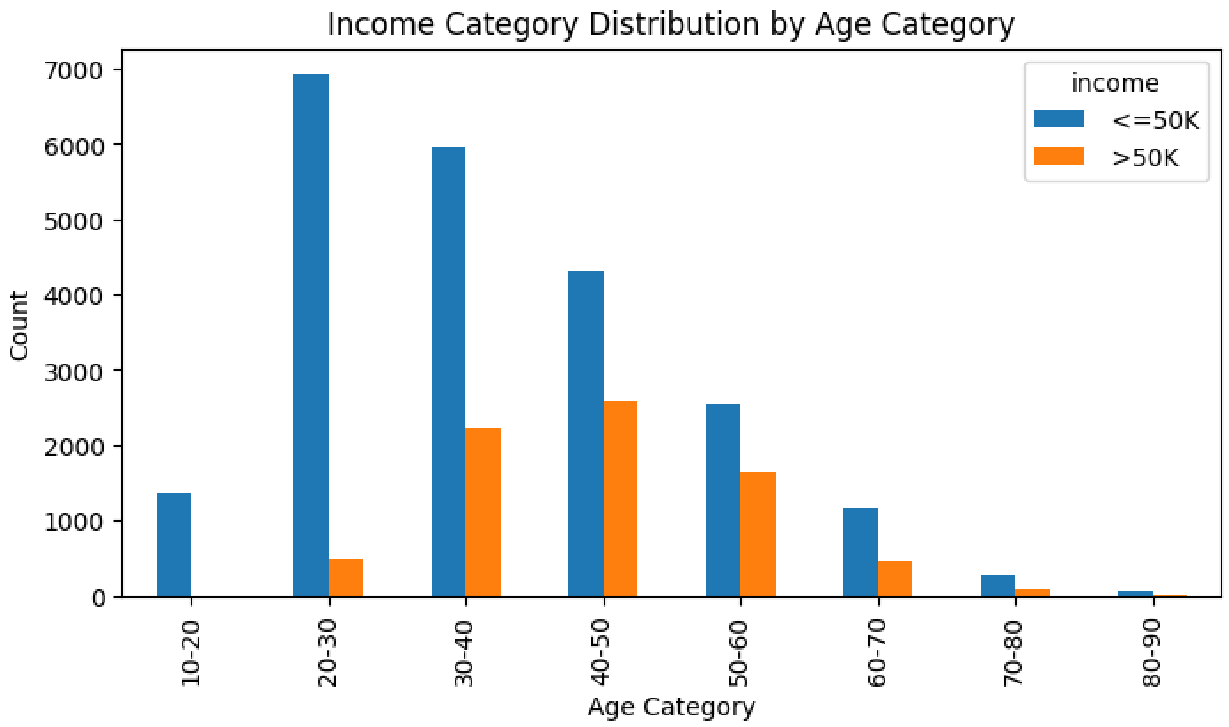
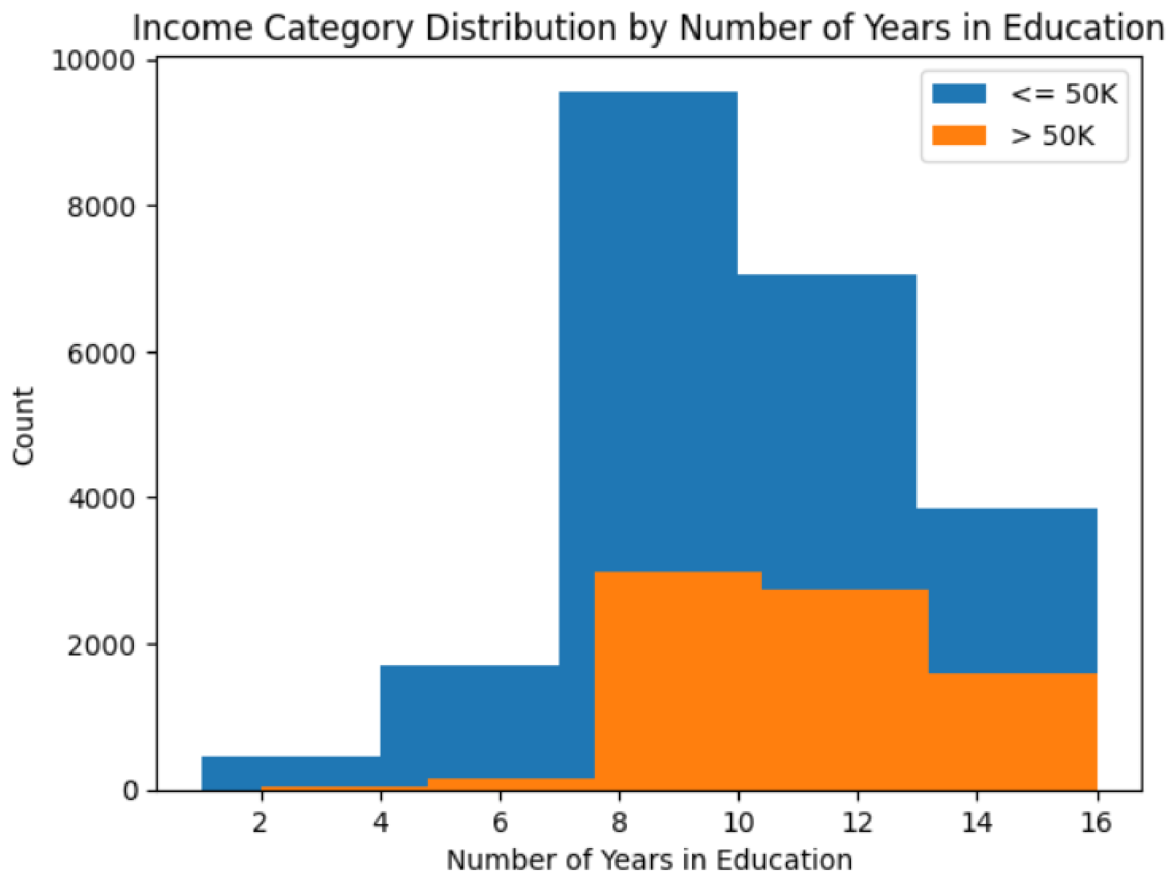


Figure 5

Income by age, stacked



Consequently, the paper was renamed from “Education versus Hourly Income” to “Demographic Factors Impacting Income”. We believe the expanded scope of this title more accurately reflects the work uncovered during this exploratory data analysis.

Model Selection

The team selected 5 classifiers for analyzing the data. The rationale behind selecting five was that it struck a good balance between letting us compare various approaches, without being overwhelmed. Only selecting one or two is not a large enough sample size for a good comparison. On the other hand, there are diminishing returns when looking at too many approaches. Thus, we selected the following:

- The first classifier selected was the support vector machine classifier (*Scikit-Learn SVM Tutorial With Python (Support Vector Machines)*, n.d.):
    - Performs classification by constructing hyperplanes in multidimensional space.
    - Based on the concept of decision planes that define decision boundaries.
    - Employs an iterative training algorithm, which is used to minimize an error function and is implemented using a kernel.
- For our second choice we selected logistic regression (Kanade, 2022):
    - Supervised machine learning algorithm that accomplishes binary classification tasks by predicting the probability of an outcome, event, or observation.
    - Analyzes the relationship between one or more independent variables and classifies data into discrete classes.
- The third classifier selected was the decision tree classifier (*1.10. Decision Trees — Scikit-Learn 1.4.1 Documentation*, n.d.):
    - Supervised learning method used for classification and regression
    - Creates a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
    - A tree can be seen as a piecewise constant approximation
- As the fourth one we choose a random forest classifier (Donges, n.d.):
    - Supervised learning method where the “forest” it builds is an ensemble of decision trees.
    - Usually trained with the “bagging method”.

    - The general idea of the bagging method which is a combination of learning models increases the overall result.
    - Builds multiple decision trees and merges them to get a more accurate and stable prediction.
- Finally, the gradient boosting classifier was our fifth choice (*Gradient Boosting: A Step-By-Step Guide*, n.d.):
    - The algorithm starts by building a decision stump and then assigning equal weights to all the data points.
    - Increases the weights for all the points that are misclassified and lowers the weight for those that are easy to classify or are correctly classified.
    - A new decision stump is made for these weighted data points.

The following figures illustrate each of these and are cited from the same references as the above bullets.

Figure 6

Support vector (Scikit-Learn SVM Tutorial With Python (Support Vector Machines), n.d.).

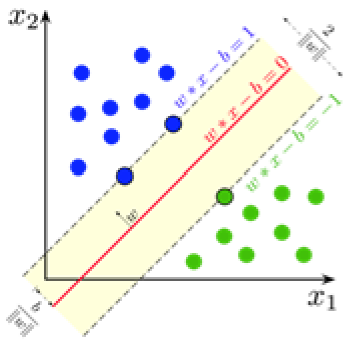


Figure 7

Logic regression (Kanade, 2022)

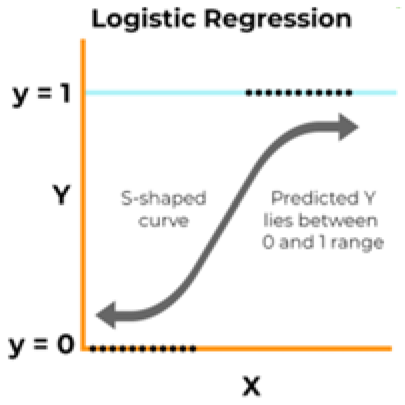


Figure 8

Decision Tree, from (1.10. Decision Trees — Scikit-Learn 1.4.1 Documentation, n.d.).

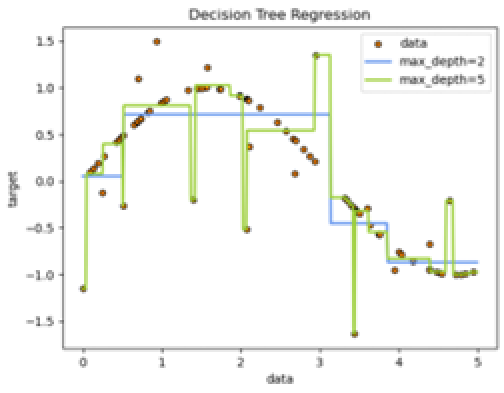


Figure 9

Random forest (Donges, n.d.).

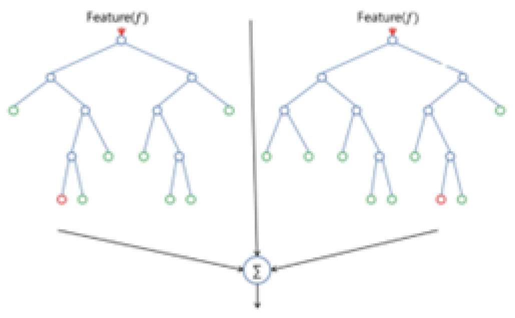
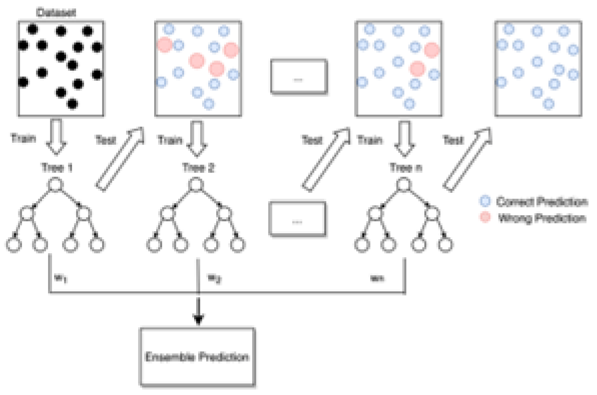


Figure 10

Gradient boosting classifier (*Gradient Boosting: A Step-By-Step Guide*, n.d.).



As previously mentioned, there was a choice of 14 variables in the data set to examine. Examining each of them would be a bit overwhelming. Therefore, we narrowed the set down to the variables we thought would have a significant influence on income. As a result, we considered the following features of the data for our analysis:

- Education
- Race
- Occupation
- Age Category (age divided into bins of 10)
- Education Number (number of years spent in education)

The data was split into two parts: one for training, and another for testing. 80% of the dataset was used for training, while the remaining 20% was used for testing. We broke each of these sets further into two subsets: x and y. We then proceeded to run the models, and the results and analysis are presented in the following section.

Model Analysis

The five models were then run on the data. The following are the results of our analysis for each of the five classifiers.

Table 1

Supported Vector Machine Results

```
            precision    recall  f1-score   support

       <=50K       0.82      0.93      0.87      4503
        >50K       0.65      0.38      0.48      1530

    accuracy                           0.79      6033
   macro avg       0.73      0.66      0.68      6033
weighted avg       0.77      0.79      0.77      6033
```

Table 2

Logic Regression Results

```
             precision    recall  f1-score   support

       <=50K       0.82      0.93      0.87      4503
        >50K       0.66      0.40      0.50      1530

    accuracy                           0.80      6033
   macro avg       0.74      0.67      0.69      6033
weighted avg       0.78      0.80      0.78      6033
```

Table 3

Decision Tree Results

```
              precision    recall  f1-score   support

       <=50K       0.82      0.92      0.87      4503
        >50K       0.64      0.41      0.50      1530

    accuracy                           0.79      6033
   macro avg       0.73      0.67      0.69      6033
weighted avg       0.78      0.79      0.78      6033
```

Table 4

Random Forest Results

```
              precision    recall  f1-score   support

       <=50K       0.82      0.91      0.87      4503
        >50K       0.63      0.43      0.51      1530

    accuracy                           0.79      6033
   macro avg       0.73      0.67      0.69      6033
weighted avg       0.78      0.79      0.78      6033
```

Table 5

Gradient Boosting Machines Results

```
              precision    recall  f1-score   support

       <=50K       0.82      0.92      0.87      4503
        >50K       0.65      0.42      0.51      1530

    accuracy                           0.80      6033
   macro avg       0.74      0.67      0.69      6033
weighted avg       0.78      0.80      0.78      6033
```

This was the most shocking part of the exercise: there wasn't a dramatic difference between the five models. This could be because these demographic factors all had a somewhat similar impact on income? In that case, it could be that these models were just different takes on the same slice of "truth".

Conclusion and Recommendations

All of our models had similar results, so an area for future research would be to dig deeper into why that is. Perhaps different combinations of the demographic factors could lead to different results? Another area to dig deeper into is the data selection itself. Is the sample we used truly random? If it is not, then the results could be skewed. One way to examine this would be to select other sets from the same period and see if our analysis stands true still- as opposed to splitting the dataset into training and testing data as we did.

Another interesting area for future work would be to examine these factors over time. For example, has education played more of a role in determining income as the United States transformed from a largely agrarian society through the industrial revolution, and finally into the information revolution we're in today? One could naturally hypothesize yes, that education does play more of a role now. However, society as a whole has become wealthier over time- so maybe this isn't true.

Our exercise of looking at demographic factors influencing income was certainly an interesting one. We were able to determine several factors that influenced income, although not at an extremely high predictive rate.

References

About the Bureau. (2023, August 2). U.S. Census Bureau. Retrieved February 17, 2024, from https://www.census.gov/about.html

Becker, B., & Kohavi, R. (n.d.). *Census Income*. Retrieved February 13, 2024, from https://archive.ics.uci.edu/dataset/2/adult

Donges, N. (n.d.). *What Is Random Forest? A Complete Guide*. Built In. Retrieved February 21, 2024, from https://builtin.com/data-science/random-forest-algorithm

Gradient Boosting: A Step-by-Step Guide. (n.d.). Analytics Vidhya. Retrieved February 21, 2024, from https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/

Kanade, V. (2022, April 18). *Logistic Regression: Equation, Assumptions, Types, and Best Practices*. Spiceworks. Retrieved February 21, 2024, from https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/

1.10. Decision Trees — scikit-learn 1.4.1 documentation. (n.d.). Scikit-learn. Retrieved February 21, 2024, from https://scikit-learn.org/stable/modules/tree.html

Scikit-learn SVM Tutorial with Python (Support Vector Machines). (n.d.). DataCamp. Retrieved February 21, 2024, from https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

US Securities and Exchange Commision. (2023, May 25). *Frequently Asked Questions About Form 13F*. Retrieved February 13, 2024, from https://www.sec.gov/divisions/investment/13ffaq

What Is a CUSIP Number, and How Do I Find a Stock or Bond CUSIP? (n.d.). Investopedia. Retrieved February 13, 2024, from https://www.investopedia.com/terms/c/cusipnumber.asp

Appendix A: Code and Output

Appendix A: Code and Output

Some of this is covered in the preceding text. This is dump of our notebook, including the output. For easier to read formatting, please see the readme: https://github.com/suvoganguli/FinalProject/blob/main/README.md

1 Adult dataset

Final_Project

February 20, 2024

For the Final Project, we will do statistical analysis on the Census Income dataset available at the UC Irvine Machine Learning Repository.

Here is the information on the dataset: - Dataset Characteristics: Multivariate - Subject Area: Social Science - Associated Tasks: Classification - Feature Type: Categorical, Integer - No. of Instances: 48842 - No. of Features: 14

```
# import libraries

import pandas as pd
from matplotlib import gridspec
import math
import matplotlib.pyplot as plt
import random
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from scipy import stats
from sklearn.svm import SVC
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report from sklearn.linear_model import
LogisticRegression
from sklearn.metrics import classification_report

# columns of interest
column_names = [
   'age',
   'workclass',
   'fnlwgt',
   'education',
   'education-num',
   'marital-status',
   'occupation',
```

[3]:

1

```

   'relationship',
   'race',
   'sex',
   'capital-gain',
   'capital-loss',
   'hours-per-week',
   'native-country',
   'income',

]

# read data

df = pd.read_csv('adult/adult.data', names=column_names)
# get smaller chunk of data if desired

do_split = False if do_split:

   df_shuffled = df.sample(frac=1,random_state = 51)
   result = np.array_split(df_shuffled, 50)
   data = result[0]
   n = data.shape[0]

data.index = range(0,n) else:

data = df


# data cleanup
# initial data shape
print(data.shape)
# replace missing values

data.replace("?", np.NaN, inplace=True) data.replace(" ?", np.NaN, inplace=True)
```

```
# data preprocessing
# drop rows with missing values data.dropna(inplace=True)

# final data shape
print(data.shape)
```

[4]:

```
(32561, 15)
(30162, 15)
```

[5]:

```
# get information about the size of the dataset

data.head()
```

2

```
age
```

```
1.   0  39
2.   1  50
3.   2  38
```

```
3 53

workclass  fnlwgt   education  education-num  \

4

0 1 2 3 4

0 1 2 3 4

28

Private

338409

      State-gov
Self-emp-not-inc
        Private
```

```
77516

Bachelors          13
Bachelors          13
  HS-grad           9
     11th           7
Bachelors          13
    marital-status
     Never-married
Married-civ-spouse
          Divorced
Married-civ-spouse
Married-civ-spouse
       occupation
     Adm-clerical
  Exec-managerial
Handlers-cleaners
Handlers-cleaners
   Prof-specialty
 relationship
Not-in-family
      Husband
Not-in-family
      Husband
         Wife
 race
White    Male
White    Male
White    Male
Black    Male
Black   Female
          83311
         215646
Private  234721
capital-gain  capital-loss  hours-per-week  native-country  income
2174           0
   0           0
   0           0
   0           0
   0           0
40   United-States
13   United-States
40   United-States
40   United-States
40            Cuba
<=50K
<=50K
<=50K
<=50K
<=50K
```

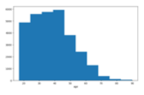
```
sex \
```

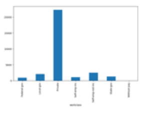
[5]:

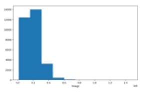
```
# Plotting function for discrete variables

import math

def discrete_plots(df, columns, num_cols): n_plots = len(columns)
n_cols = num_cols
n_rows = int(math.ceil(n_plots/n_cols)) gs = gridspec.GridSpec(n_rows, n_cols) fig =
plt.figure(figsize=(8,100))

for i in range(n_plots):
ax = fig.add_subplot(gs[i])
if df.dtypes[columns[i]] != 'int64':

df[columns[i]].value_counts().sort_index().plot(kind='bar', ax=ax) else:

df[columns[i]].hist(ax=ax, grid=False) ax.set_xlabel(columns[i])

   fig.tight_layout()
   fig.supylabel('Count')
   plt.show()
discrete_plots(data,column_names,1)
```

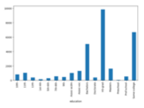
[6]:

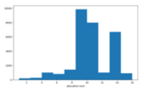
```
3
```

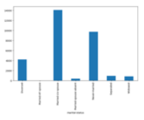


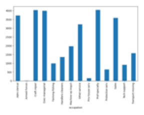


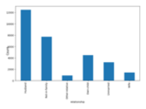


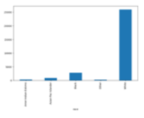


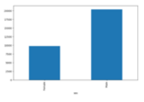


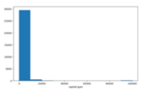


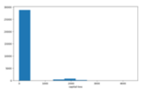


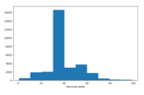


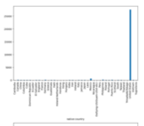


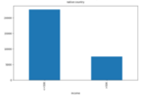












4

1.1 Features affecting income level

We consider the following features which can potentially affect the income level.

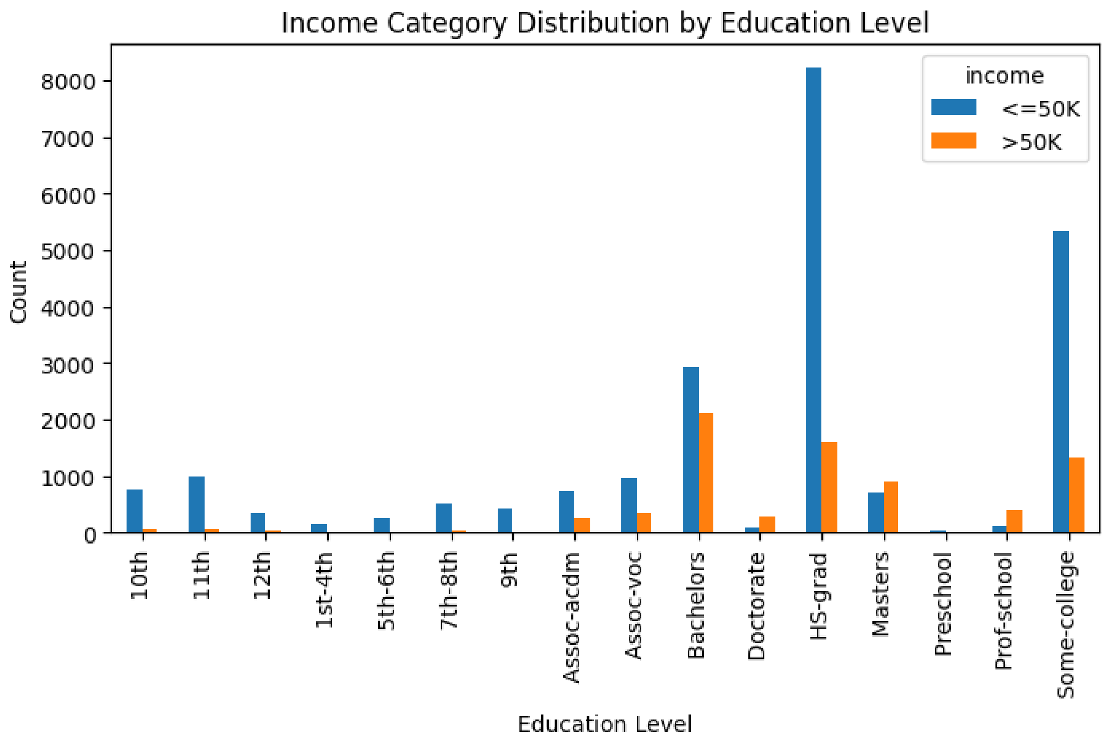
1. Education
2. Race
3. Occupation
4. Age Category (age divided into bins of 10) 5. Number of years in education

The corresponding barplots are shown below.

```
# Group the data by education level and income category

education_income_counts = data.groupby(['education', 'income']).size().unstack() # Grouped bar
plot

education_income_counts.plot(kind='bar', stacked=False, figsize=(8, 4))

# Add labels and title
plt.title('Income Category Distribution by Education Level')
plt.xlabel('Education Level')
plt.ylabel('Count')
```

[7]:

[7]: Text(0, 0.5, 'Count')

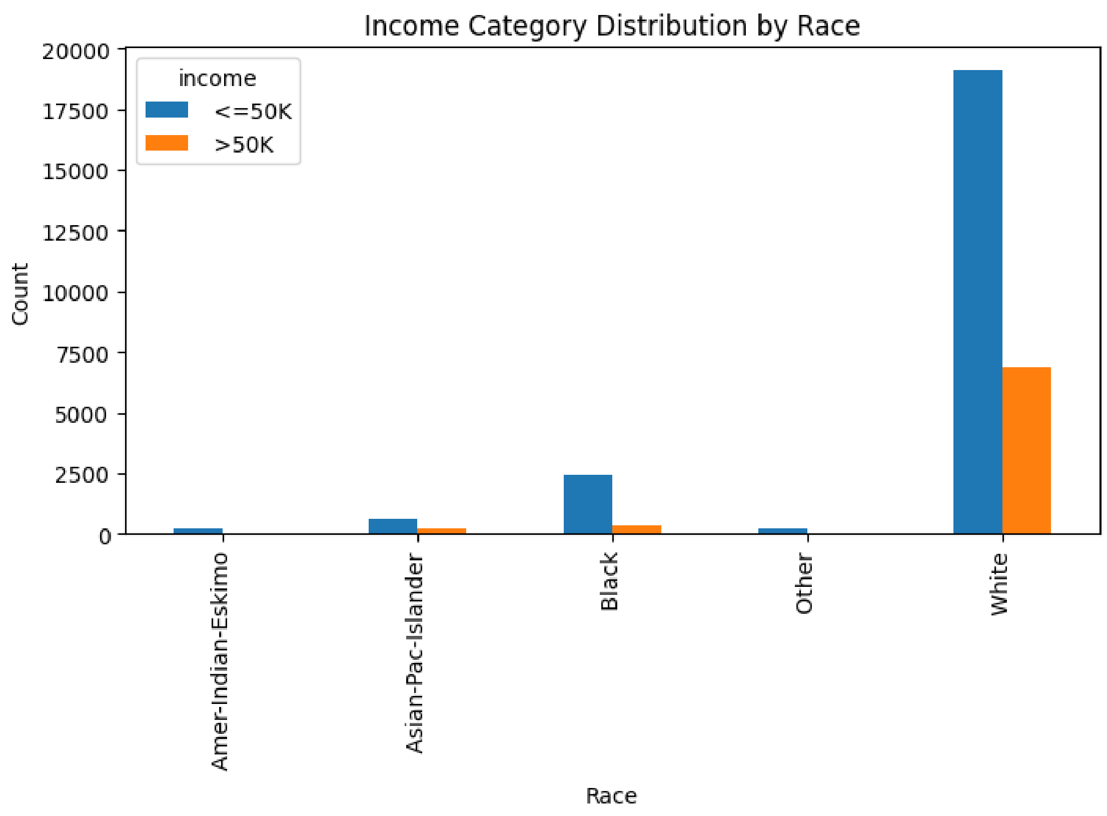


5

```
# Group the data by race and income category

race_income_counts = data.groupby(['race', 'income']).size().unstack() # Grouped bar plot

race_income_counts.plot(kind='bar', stacked=False, figsize=(8, 4))

# Add labels and title
plt.title('Income Category Distribution by Race')
plt.xlabel('Race')
plt.ylabel('Count')
```

[8]:

```
[8]: Text(0, 0.5, 'Count')
```



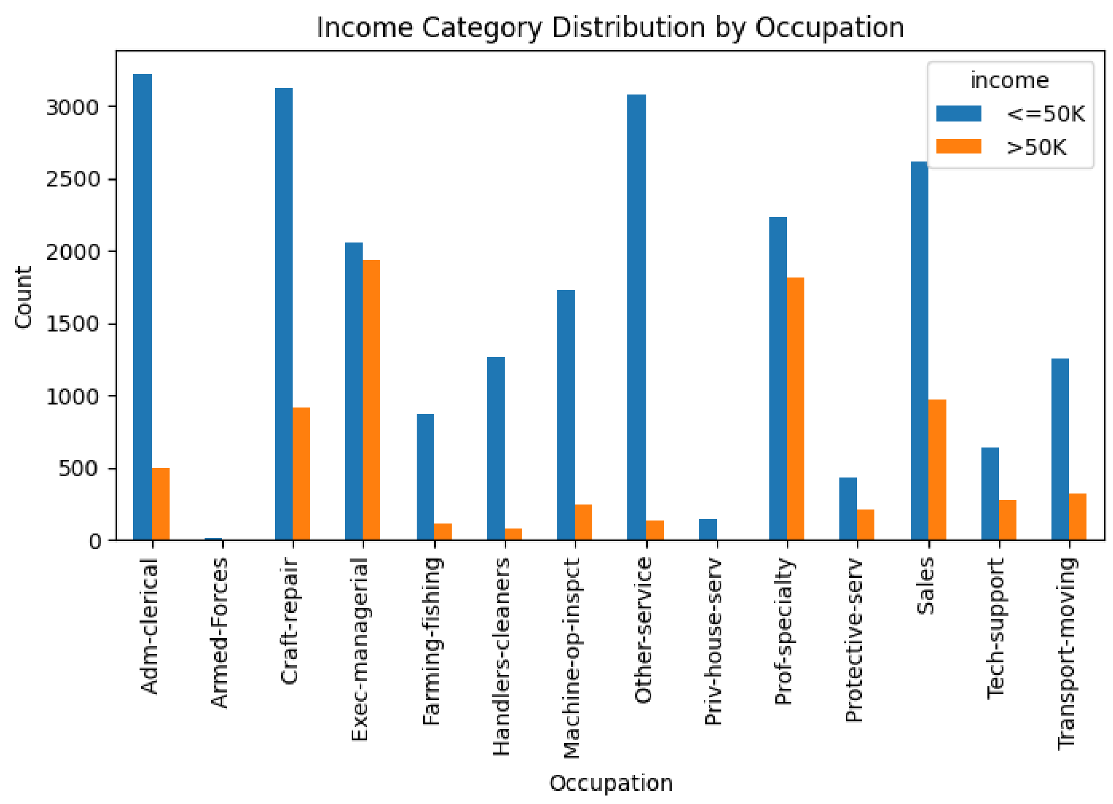
```
# Group the data by occupation and income category

occupation_income_counts = data.groupby(['occupation', 'income']).size(). ↪unstack()

# Grouped bar plot
```

[9]:

6

```
occupation_income_counts.plot(kind='bar', stacked=False, figsize=(8, 4))

# Add labels and title
plt.title('Income Category Distribution by Occupation')
plt.xlabel('Occupation')
plt.ylabel('Count')
```

[9]: Text(0, 0.5, 'Count')



```
# converting age to categorical data
print('Min age = ', np.min(data['age']))
print('Max age = ', np.max(data['age']))
bins = [10,20,30,40,50,60,70,80,90]
labels = ['10-20','20-30','30-40','40-50','50-60','60-70','70-80','80-90']

data['age_category'] = pd.cut(data['age'], bins=bins, labels=labels,␣ ↪right=False)

data.head()
```

[10]:

7

Minage= 17 Maxage= 90

[10]:

```
age
```

```
1.   0  39
2.   1  50
```

```
3.   2  38
```

```
3 53

workclass  fnlwgt   education  education-num  \

4

0 1 2 3 4

28

Private

338409

      State-gov
Self-emp-not-inc
        Private

77516

Bachelors           13
Bachelors           13
  HS-grad            9
    11th            7
Bachelors           13
   marital-status
    Never-married
Married-civ-spouse
         Divorced
Married-civ-spouse
Married-civ-spouse
      occupation
    Adm-clerical
  Exec-managerial
Handlers-cleaners
Handlers-cleaners
   Prof-specialty
 relationship
Not-in-family
     Husband
Not-in-family
     Husband
        Wife
 race
White    Male
White    Male
White    Male
```

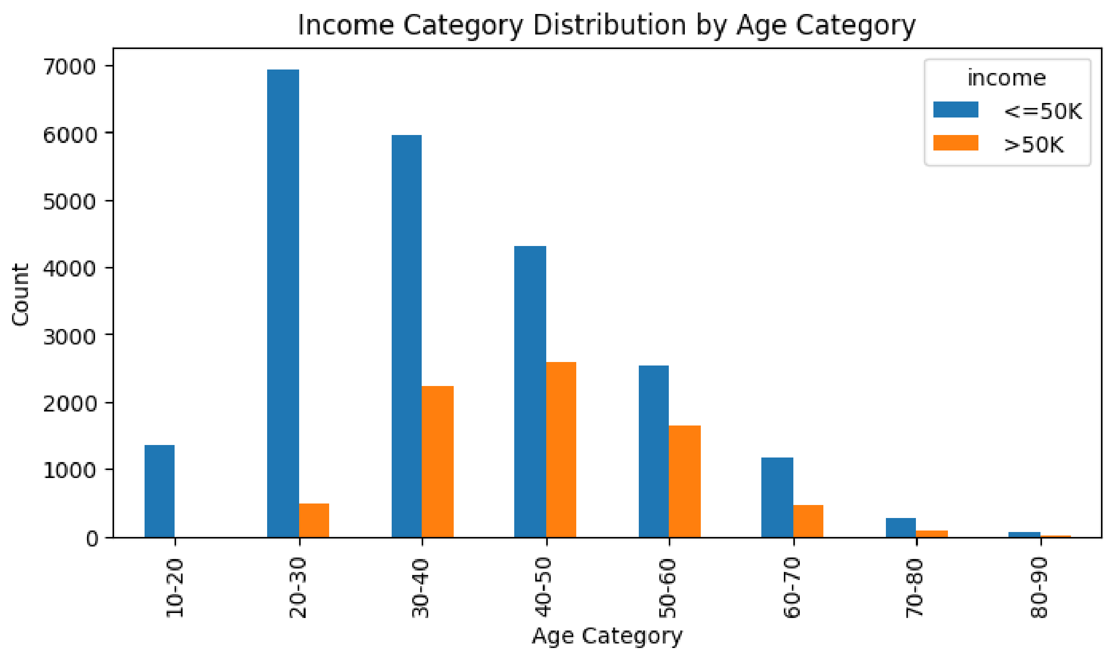
```
Black    Male
Black   Female
      83311
     215646
Private  234721
capital-gain  capital-loss  hours-per-week  native-country  income  \
0       2174          0
1          0          0
2          0          0
3          0          0
4          0          0
 age_category
0      30-40
1      50-60
2      30-40
3      50-60
4      20-30
40   United-States
13   United-States
40   United-States
40   United-States
40          Cuba
<=50K
<=50K
<=50K
<=50K
<=50K

sex \
```

```
# Group the data by age and income category

age_income_counts = data.groupby(['age_category', 'income'], observed=False). ↪size().unstack()

# Grouped bar plot

age_income_counts.plot(kind='bar', stacked=False, figsize=(8, 4))

# Add labels and title
plt.title('Income Category Distribution by Age Category')
plt.xlabel('Age Category')
plt.ylabel('Count')
```

[11]:

[11]: Text(0, 0.5, 'Count')

8

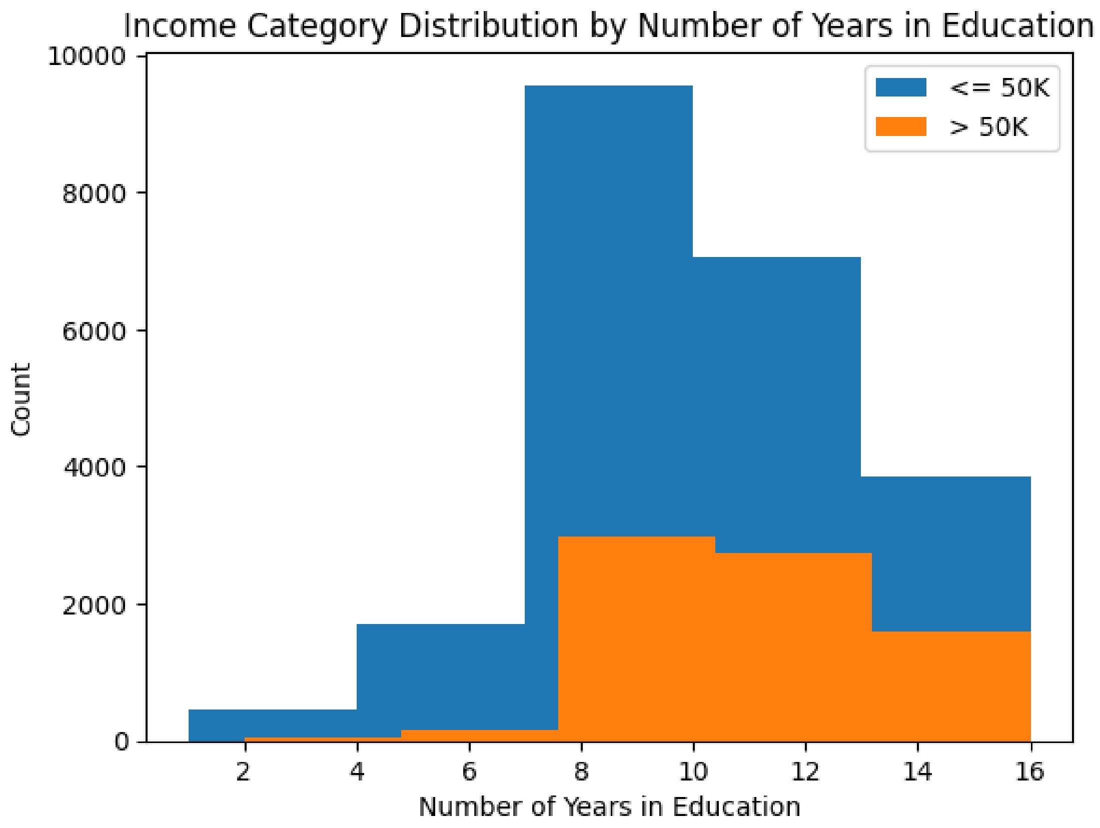


```
grouped_data = {}
for category, values in zip(data['income'], data['education-num']):

   grouped_data.setdefault(category, []).append(values)
# Plotting histogram for each category

plt.figure()
for category, values in grouped_data.items():

   plt.hist(values, bins=5, label=category)
# Add labels and title
plt.title('Income Category Distribution by Number of Years in Education')
plt.legend(['<= 50K','> 50K'])
plt.xlabel('Number of Years in Education')
plt.ylabel('Count')
```

[12]:

[12]: Text(0, 0.5, 'Count')

9



2 Value Counts

The code below shows the value counts for each of the independent variables we have considered above for the statistical analysis.

```
# income counts

income_counts = data['income'].value_counts()
# sorting income count by descending order
income_counts_sorted = income_counts.sort_values(ascending=False) # display the sorted income
levels and their counts print(income_counts_sorted)

# ---------------
# education counts

education_counts = data['education'].value_counts()
# sorting education count by descending order
education_counts_sorted = education_counts.sort_values(ascending=False) # display the sorted
education levels and their counts print(education_counts_sorted)
```

[21]:

10

```
# ---------------

# race counts

race_counts = data['race'].value_counts()
# sorting race count by descending order
race_counts_sorted = race_counts.sort_values(ascending=False) # display the sorted race and their
counts print(race_counts_sorted)

# ---------------
# occupation counts

occupation_counts = data['occupation'].value_counts()
# sorting occupation count by descending order
occupation_counts_sorted = occupation_counts.sort_values(ascending=False) # display the sorted
occupation and their counts print(occupation_counts_sorted)

# ---------------
# age category counts

age_counts = data['age_category'].value_counts()
# sorting occupation count by descending order age_counts_sorted =
age_counts.sort_values(ascending=False) # display the sorted occupation and their counts
print(age_counts_sorted)
```

```
income
 <=50K    22654
 >50K      7508
Name: count, dtype: int64
education
 HS-grad         9840
 Some-college    6678
 Bachelors       5044
 Masters         1627
 Assoc-voc       1307
 11th            1048
 Assoc-acdm      1008
 10th             820
 7th-8th          557
 Prof-school      542
 9th              455
 12th             377
 Doctorate        375
 5th-6th          288
 1st-4th          151
```

11

[36]:

```
 Preschool       45
Name: count, dtype: int64
race
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
25933
 2817
  895
  286
  231
Name: count, dtype: int64
occupation
 Prof-specialty      4038
 Craft-repair        4030
 Exec-managerial     3992
 Adm-clerical        3721
 Sales               3584
 Other-service       3212
 Machine-op-inspct   1966
 Transport-moving    1572
 Handlers-cleaners   1350
 Farming-fishing      989
 Tech-support         912
 Protective-serv      644
 Priv-house-serv      143
 Armed-Forces
Name: count, dtype: int64
age_category
30-40   8211
20-30   7415
40-50   6900
50-60   4185
60-70   1634
10-20   1369
70-80    357
80-90     56
Name: count, dtype: int64
```

9

3 Correlation Analysis

In this section, we perform a correlation analysis between the potential independent variables we considered above and the dependent variable. After the analysis, we have decided that we will excluding the ‘race’ variable since the correlation is very small.

```
# import library

from scipy.stats import chi2_contingency

def cramers_v(x, y):
confusion_matrix = pd.crosstab(x, y)
```

12

```
   chi2 = chi2_contingency(confusion_matrix)[0]
   n = confusion_matrix.sum().sum()
   phi2 = chi2 / n
   r, k = confusion_matrix.shape

phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
rcorr = r - ((r-1)**2)/(n-1)
kcorr = k - ((k-1)**2)/(n-1)
return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

correlation_education = cramers_v(data['education'], data['income']) print(f"Correlation with
Education: {correlation_education:.4f}")

correlation_race = cramers_v(data['race'], data['income']) print(f"Correlation with Race:
{correlation_race:.4f}")

correlation_occupation = cramers_v(data['occupation'], data['income']) print(f"Correlation with
Occupation: {correlation_occupation:.4f}")

correlation_age_category = cramers_v(data['age_category'], data['income']) print(f"Correlation with
Age Category: {correlation_age_category:.3f}")

correlation_education_num = cramers_v(data['education-num'], data['income'])
print(f"Correlation with Number of Years in Education:␣

↪{correlation_education_num:.4f}")
```

```
Correlation with Education: 0.3667
Correlation with Race: 0.0998
Correlation with Occupation: 0.3490
Correlation with Age Category: 0.308
Correlation with Number of Years in Education: 0.3667
```

4 Machine learning:

We have used the following five classifiers for our analysis: - Support Vector Machine Classifier - Logistic Regression - Decision Tree Classifier - Random Forest Classifier - Gradient Boosting Classifier

Our final list of independent variables are as follows:

1. Education
2. Occupation
3. Age Category (age divided into bins of 10)
4. Education Number (number of years spent in education)

We split the dataset into 80% for train and 20% for test. The training data consists of X_train and y_train, and the test data consists of X_test and y_test.

After training the classifiers using X_train and Y_train, we test it using X_test and y_test. 13

[23]:

The classification report for each classifier are printed as the end of each code cell. All of the classifiers have very similar performance.

```
# SVC Classifier

# Get X and y

X = pd.get_dummies(data[['education', 'occupation', 'age_category',␣ ↪'education-num']])

y = data['income']
# Split data into training and testing sets
np.random.seed(123)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣

↪random_state=42)
svc = SVC(random_state=42)

svc.fit(X_train, y_train) # Predict

y_pred = svc.predict(X_test)
# Classification report
report = classification_report(y_test, y_pred)
print(report)
```

```
precision

     <=50K      0.82
```

```
        >50K       0.65
    accuracy
   macro avg       0.73
weighted avg       0.78
recall  f1-score   support
  0.93      0.87      4503
  0.39      0.49      1530
            0.79      6033
  0.66      0.68      6033
  0.79      0.77      6033
```

```
# Logistic Regression
logistic = LogisticRegression(random_state=42, max_iter = 10000)
logistic.fit(X_train, y_train)

# Predict

y_pred = logistic.predict(X_test)
# Classification report
report = classification_report(y_test, y_pred)
print(report)
```

[24]:

```
precision

recall  f1-score   support

14
```

[26]:

```
<=50K >50K

    accuracy
   macro avg
weighted avg
0.82      0.93
0.66      0.39
0.74      0.66
0.78      0.79
0.87      4503
0.49      1530
0.79      6033
0.68      6033
0.78      6033
```

```
# import library
```

```
from sklearn.tree import DecisionTreeClassifier

# Initializing and training the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
# Making predictions
y_pred = dt_classifier.predict(X_test)
# Evaluating the model
# accuracy = accuracy_score(y_test, y_pred)
# print(f'Decision Tree Classifier Accuracy: {accuracy}')
# Classification report
report = classification_report(y_test, y_pred)
print(report)
```

[25]:

```
<=50K >50K

   accuracy
  macro avg
weighted avg
precision   recall
    0.82     0.93
    0.65     0.41
    0.74     0.67
    0.78     0.79
f1-score  support
   0.87     4503
   0.50     1530
   0.79     6033
   0.69     6033
   0.78     6033
```

```
# Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

# Initializing and training the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
# Making predictions

y_pred = rf_classifier.predict(X_test) # Evaluating the model
```

15

```
# accuracy_rf = accuracy_score(y_test, y_pred_rf)
# print(f'Random Forest Classifier Accuracy: {accuracy_rf}')
# Classification report
report = classification_report(y_test, y_pred)
print(report)
```

```
<=50K >50K

   accuracy
  macro avg
weighted avg
precision    recall
    0.82      0.92
    0.65      0.41
    0.73      0.67
    0.78      0.79
f1-score   support
   0.87      4503
   0.50      1530
   0.79      6033
   0.68      6033
   0.78      6033
```

```
# Gradient Boosting Machines Classifer

from sklearn.ensemble import GradientBoostingClassifier

# Initializing and training the GBM Classifier

gbm_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=0. ↪1,
random_state=42)

gbm_classifier.fit(X_train, y_train) # Making predictions

y_pred = gbm_classifier.predict(X_test)
# Evaluating the model
# accuracy_gbm = accuracy_score(y_test, y_pred_gbm)
# print(f'Gradient Boosting Machines Classifier Accuracy: {accuracy_gbm}')
# Classification report
report = classification_report(y_test, y_pred)
print(report)
```

[27]:

[ ]:

```
<=50K >50K
```

```
   accuracy
  macro avg
weighted avg
precision    recall
     0.83      0.92
     0.64      0.43
     0.74      0.67
     0.78      0.79
f1-score   support
    0.87      4503
    0.51      1530
    0.79      6033
    0.69      6033
    0.78      6033

16
```

Appendix B: Source Code

Source code can be found on github at https://github.com/suvoganguli/FinalProject.