CS 105 Final Project Team 1

By: Bhavika Patel, Bryan Yu, Chlinton Kuang, Daniel Lin, Danielle Lee

---

Project Description

Our topic is on income. We plan to train and test various models (supervised and unsupervised) to predict whether or not a person's income will exceed $50k based on variables such as age, education, marital status, occupation, sex, etc.

For our dataset, we will be using the "Census Income" dataset (also known as "Adult") found on the UCI Machine Learning Repository. The dataset contains continuous attributes like age, capital gains, and hours of work a week, and categorical attributes like education, occupation, relationship, and native country. The dataset was extracted from the US census in 1994, which may introduce bias into our models.

We will be using chi squared test and forward feature subset selection to determine which features are more influential to one's income. We will then train our models on the subset of features and observe which one has the highest accuracy.

Dataset: https://archive.ics.uci.edu/dataset/2/adult

Slideshow: https://docs.google.com/presentation/d/1DIvmG-mGqYy0Pzr6K4lh22z1bJeboqJAfj2sXRfzS5M/edit?usp=sharing

---

Contributions

```
**Bhavika Patel**: i worked on the decision tree
```

Bhavika Patel: i worked on the decision tree

Bryan Yu: I loaded in the repository and worked on the expository data analysis as well as the visual models and explanations. I also contributed to editing/polishing the code as well as making the result reporting.

Chlinton Kuang: I worked on the k-means clustering

Daniel Lin: I wrote the code for data standardization, normalization, one-hot-encoding, forward feature selection, and chi square test feature selection. I also created and trained the VotingClassifier, and contributed to finding the best k value for KNN

Danielle Lee: I worked on a few of the explanations and visualizations for the EDA. I also contributed to creating the KNN model for forward features and chi-squared features. I also worked on creating the slides for the presentation.

Techniques

Techniques	
Forward Feature Selection	We will use one-hot encoding to convert our categorical attributes into numerical ones. We'll st
Chi Squared Feature Selection	We will also use one-hot encoding to covert categorical attributes into numerical ones. Due to t
K-Nearest Neighbor	This is a supervised learning model that is fast and easy to implement. We will find the best po
Decision Trees	This is another supervised learning model that we plan to implement because it can easily be u
K-Means Clustering	This is one of the unsupervised learning models that we plan on implementing. We are planning
Voting Classifier	This is another supervised (ensemble learning) model that we plan on using. We are going to c

Data Cleaning

```
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.3-py3-none-any.whl (7.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.3
```

```
from ucimlrepo import fetch_ucirepo
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import tree
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import normalize
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay, silhouette_sc
from sklearn.model_selection import train_test_split
from scipy.stats import chi2_contingency, chi2
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_scc
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, export_graphviz, plot_tree
from sklearn.ensemble import VotingClassifier
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

```
# fetch dataset
adult = fetch_ucirepo(id=2)

# data (as pandas dataframes)
X = adult.data.features
y = adult.data.targets

# metadata
print(adult.metadata)

# variable information
print(adult.variables)
```

```
{'uci_id': 2, 'name': 'Adult', 'repository_url': 'https://archive.ics.uci.edu
              name     role         type      demographic  \
0              age  Feature      Integer              Age
1        workclass  Feature  Categorical           Income
2           fnlwgt  Feature      Integer             None
3        education  Feature  Categorical  Education Level
4    education-num  Feature      Integer  Education Level
5   marital-status  Feature  Categorical            Other
6       occupation  Feature  Categorical            Other
7     relationship  Feature  Categorical            Other
8             race  Feature  Categorical             Race
9              sex  Feature       Binary              Sex
10    capital-gain  Feature      Integer             None
11    capital-loss  Feature      Integer             None
12  hours-per-week  Feature      Integer             None
13  native-country  Feature  Categorical            Other
14          income   Target       Binary           Income

                                          description units missing_values
0                                                 N/A  None             no
1   Private, Self-emp-not-inc, Self-emp-inc, Feder...  None            yes
2                                                None  None             no
3    Bachelors, Some-college, 11th, HS-grad, Prof-...  None             no
4                                                None  None             no
5   Married-civ-spouse, Divorced, Never-married, S...  None             no
6   Tech-support, Craft-repair, Other-service, Sal...  None            yes
7   Wife, Own-child, Husband, Not-in-family, Other...  None             no
8   White, Asian-Pac-Islander, Amer-Indian-Eskimo,...  None             no
9                                       Female, Male.  None             no
10                                               None  None             no
11                                               None  None             no
12                                               None  None             no
13  United-States, Cambodia, England, Puerto-Rico,...  None            yes
14                                   >50K, <=50K.  None             no
```

```
# all features
print(X.columns)
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country'],
      dtype='object')
```

```
df = pd.DataFrame(X)
incomes = pd.DataFrame(y) #data is messy. For some reason, there are periods that
incomes = incomes.replace(to_replace = [">50K.", "<=50K."], value = [">50K", "<=5
df['incomes'] = incomes['income']
orig_df = df.copy()
df.head()
```

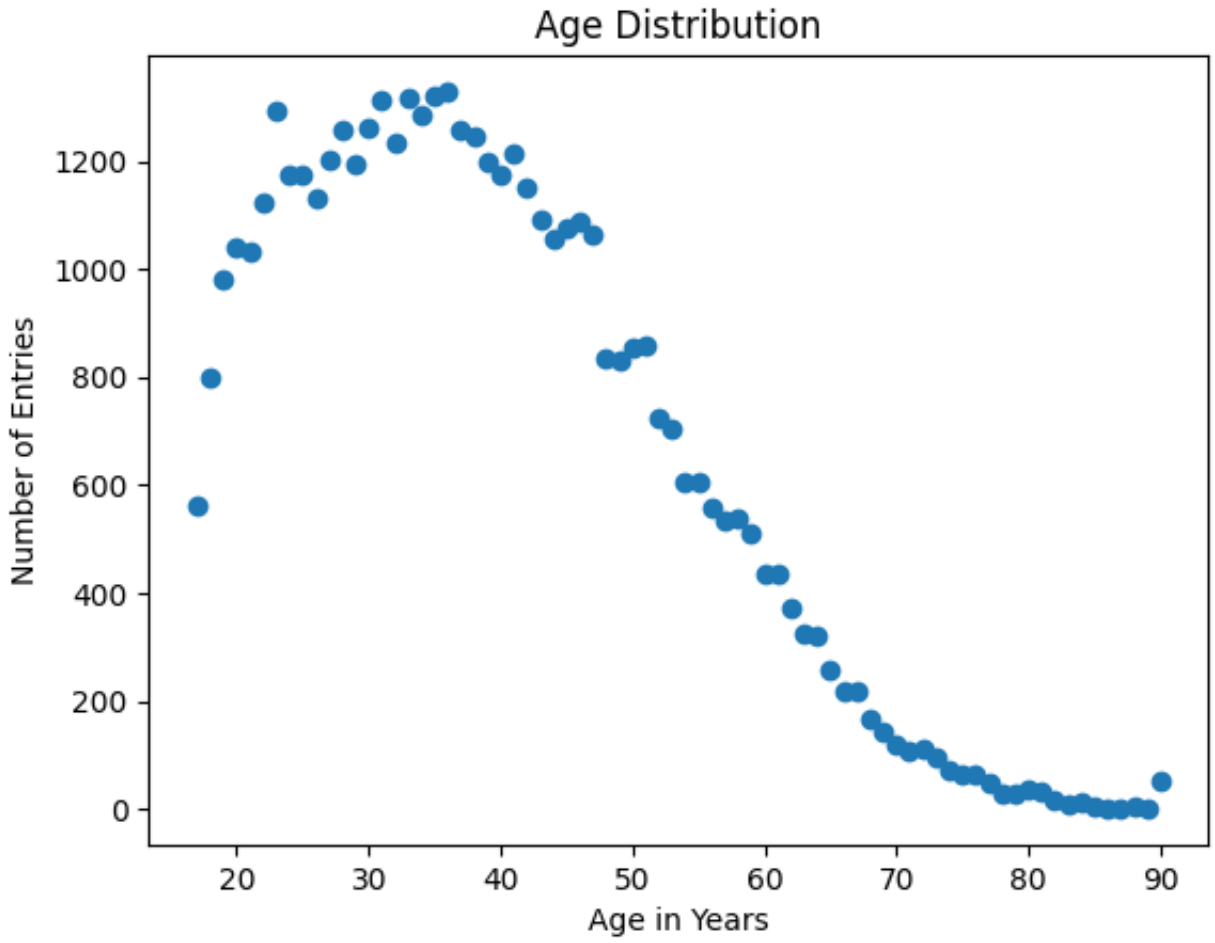
	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relatio
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Hu
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Hu
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	

```
#columns with missing values: workclass, occupation, native-country
#drop NA columns for now
df.dropna(inplace = True)
df.drop('fnlwgt',axis=1, inplace = True)
cleaned_df = df.copy()
df.shape #note: some of the columns became reformatted entirely
```

```
(47621, 14)
```

EDA

```
age_dist = df.groupby('age').size()
ages = range(min(df['age']), max(df['age']) + 1)


#Visual 1: Age distribution
plt.scatter(ages, age_dist)
plt.title('Age Distribution')
plt.xlabel('Age in Years')
plt.ylabel('Number of Entries')

plt.show()
```

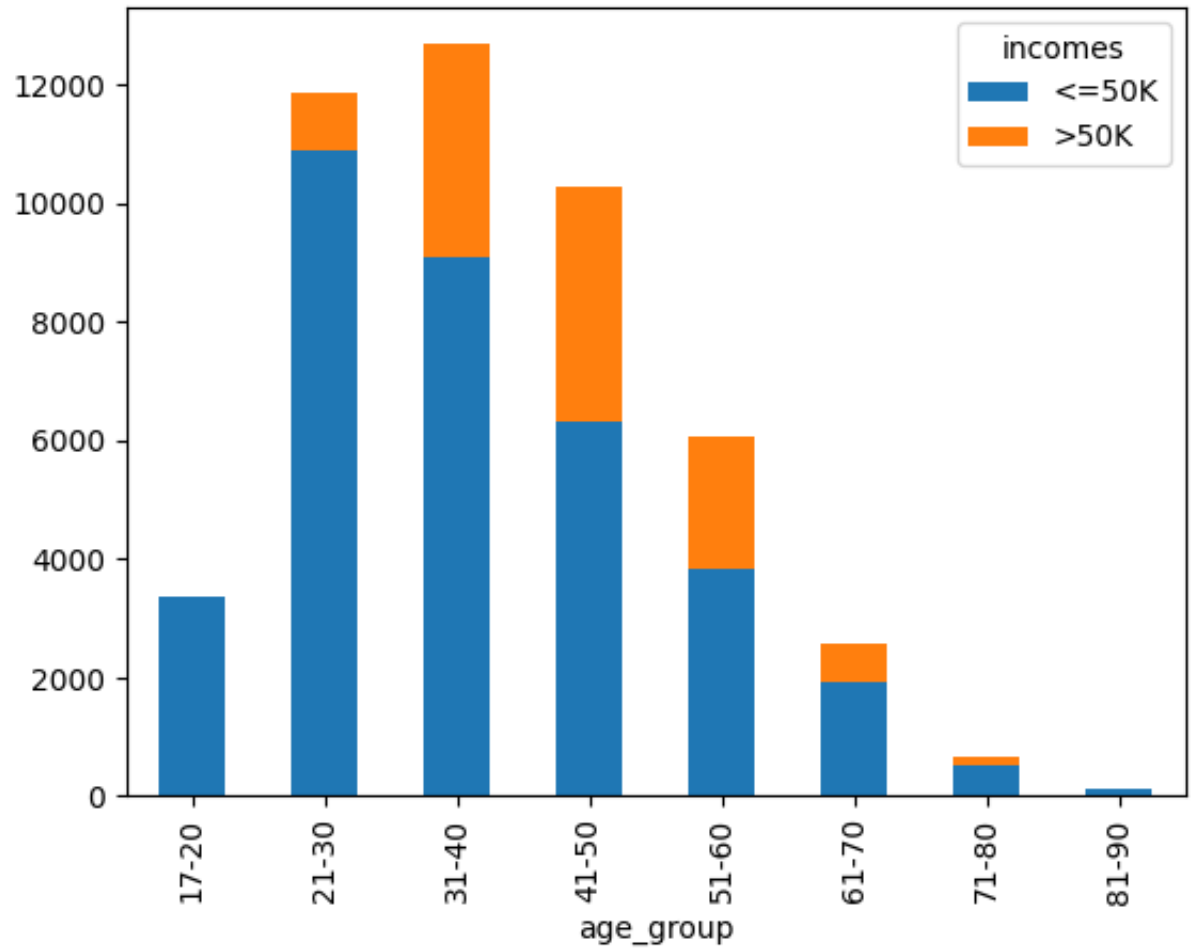


Explanation:

This is the age distribution of people who filled out the income portion of the census that we gathered the data from. It shows that people ranging from about 30 to 40 years old had the most entries. From that point, the number of entries decreased as the age of participants increased. It is important to take into account the age distribtion as it might skew our income results. This may be due to people 20 years old and younger still pursuing an education and working part-time and there are not many part-time jobs that provide an income of 50k or above. Also, people 60 years old and older are mostly retired or no longer working so their income will be less than 50k or even no income at all.

```
#Visual 2: Age vs income
#Note: binning automatically starts from numbers ending in "0" for some reason
df["age_group"] = pd.cut(df["age"], bins = range(10, 91, 10), right = True,
                  labels = ["17-20", "21-30", "31-40", "41-50", "51-60", "61-70",
                            "71-80", "81-90"])
df['obs'] = range(1, len(df) + 1)

age_vs_income = df.pivot_table(
    index = "age_group", columns = "incomes",
    values = "obs", aggfunc = "count"
)
age_vs_income = age_vs_income.fillna(0)
#NA means there are no respondents that match; filling with 0 is fine

(age_vs_income).plot.bar(stacked = True)
```

```
<Axes: xlabel='age_group'>
```

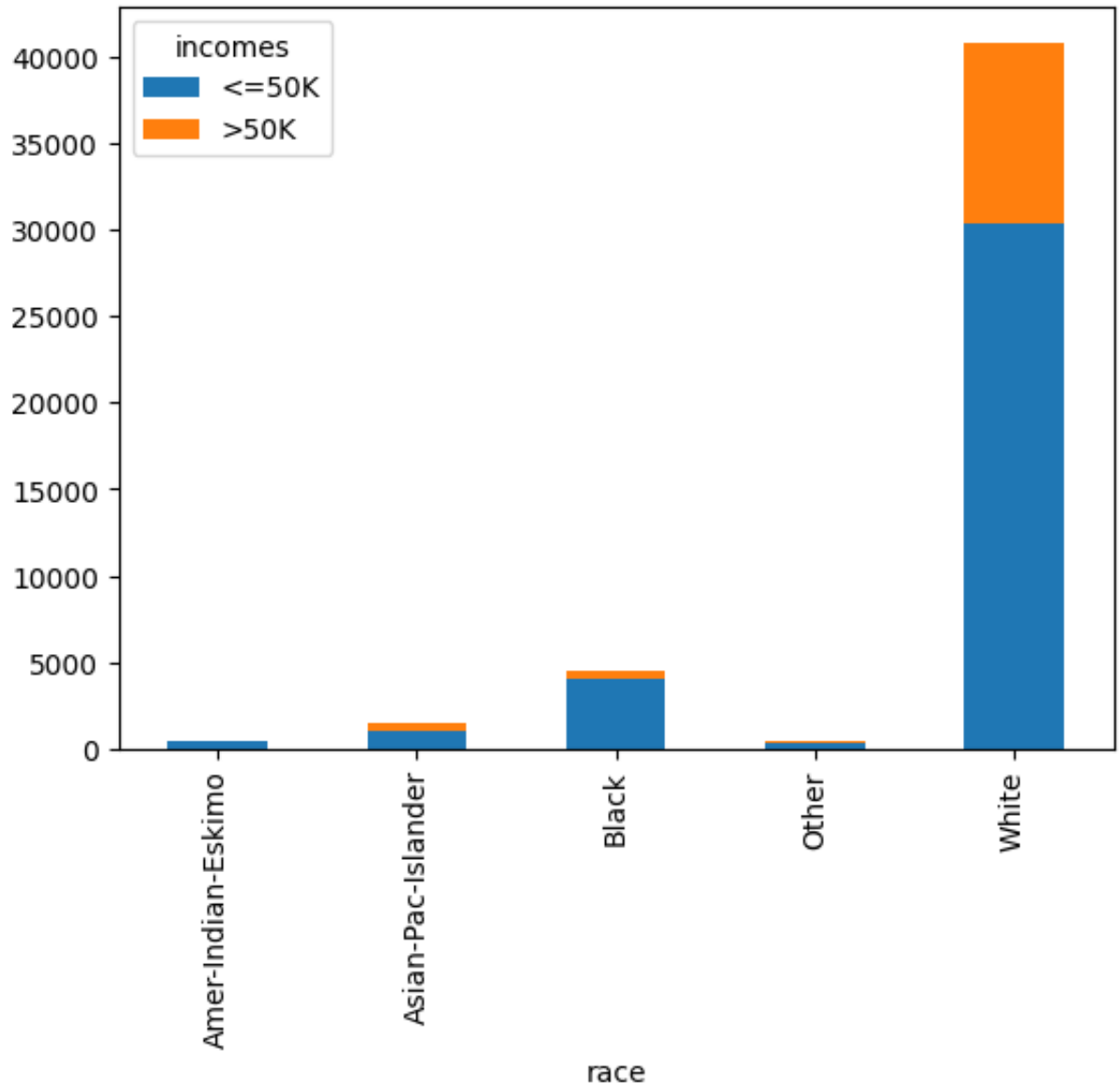


Explanation:

The stacked bar chart displays the number of people with incomes greater than 50k or less than 50k based on their age group. Relative to each age group, there is a greater proportion of people who make less than 50k than greater than 50k. In the youngest age group, 17-20 years old, and the oldest age group, 81-90 years old, they all have incomes less than 50k which makes sense due to the reasoning mentioned in the previous explanation above.

```
#Visual 3: Race vs Income
race_vs_income = df.pivot_table(
    index = "race", columns = "incomes",
    values = "obs", aggfunc = "count"
)
race_vs_income = race_vs_income.fillna(0) #NAN means that no respondent fits the
(race_vs_income).plot.bar(stacked = True)
```

```
<Axes: xlabel='race'>
```



Explanation:

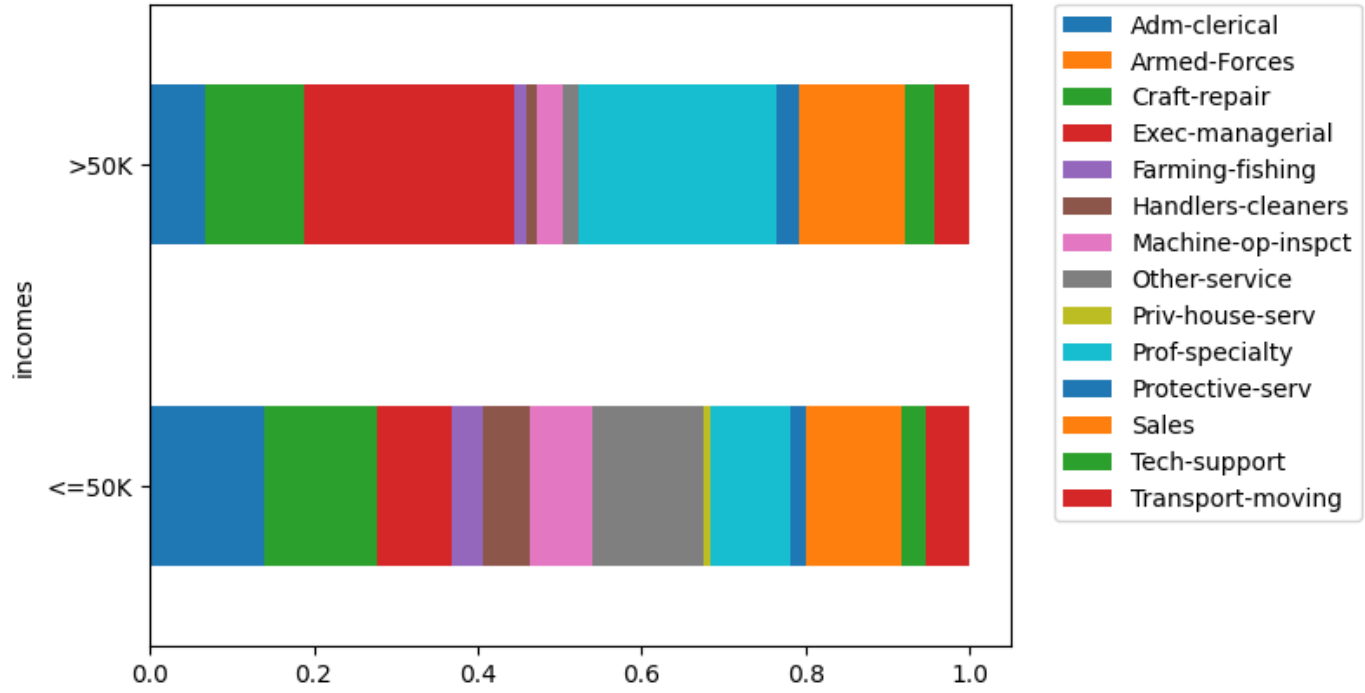
The stacked bar chart shows the the number of people with incomes greater than 50k or less than 50k based on their race. The results show that majority of the data comes from white people and there is not much diversity in the census. This is an important factor to consider because higher income for white people can be potentially attributed to white privilege allowing greater access to higher paying jobs or promotions. There is only a very small proportion of people of American-Indian-Eskimo, Asian-Islander, Black, and other races that have an income greater than 50k. In short, it seems that an overwhelming majority of those surveyed were White, and but a larger proportion of them make more than 50K compared to most other ethnicities proportions regardless (if you look at the Asian-Pacific-Islander bar, about 1/3 of them make more than 50K.)

```
#Visual 4: Income vs Occupation
cleaned_df2 = cleaned_df[cleaned_df.occupation != "?"]

income_occupation = pd.crosstab(cleaned_df2.iloc[:,13],cleaned_df2.iloc[:,5])
income_occupation
```

occupation	Adm-clerical	Armed-Forces	Craft-repair	Exec-managerial	Farming-fishing	Handlers-cleaners	Machine-op-inspct
incomes							
<=50K	4824	10	4713	3160	1313	1928	2636
>50K	765	4	1376	2898	172	138	370

```
income_occupationP = income_occupation.divide(cleaned_df2.iloc[:,13].value_counts
income_occupationP.plot.barh(stacked=True).legend(bbox_to_anchor=(1.05, 1),loc='u
```

```
<matplotlib.legend.Legend at 0x79d1524f3b80>
```

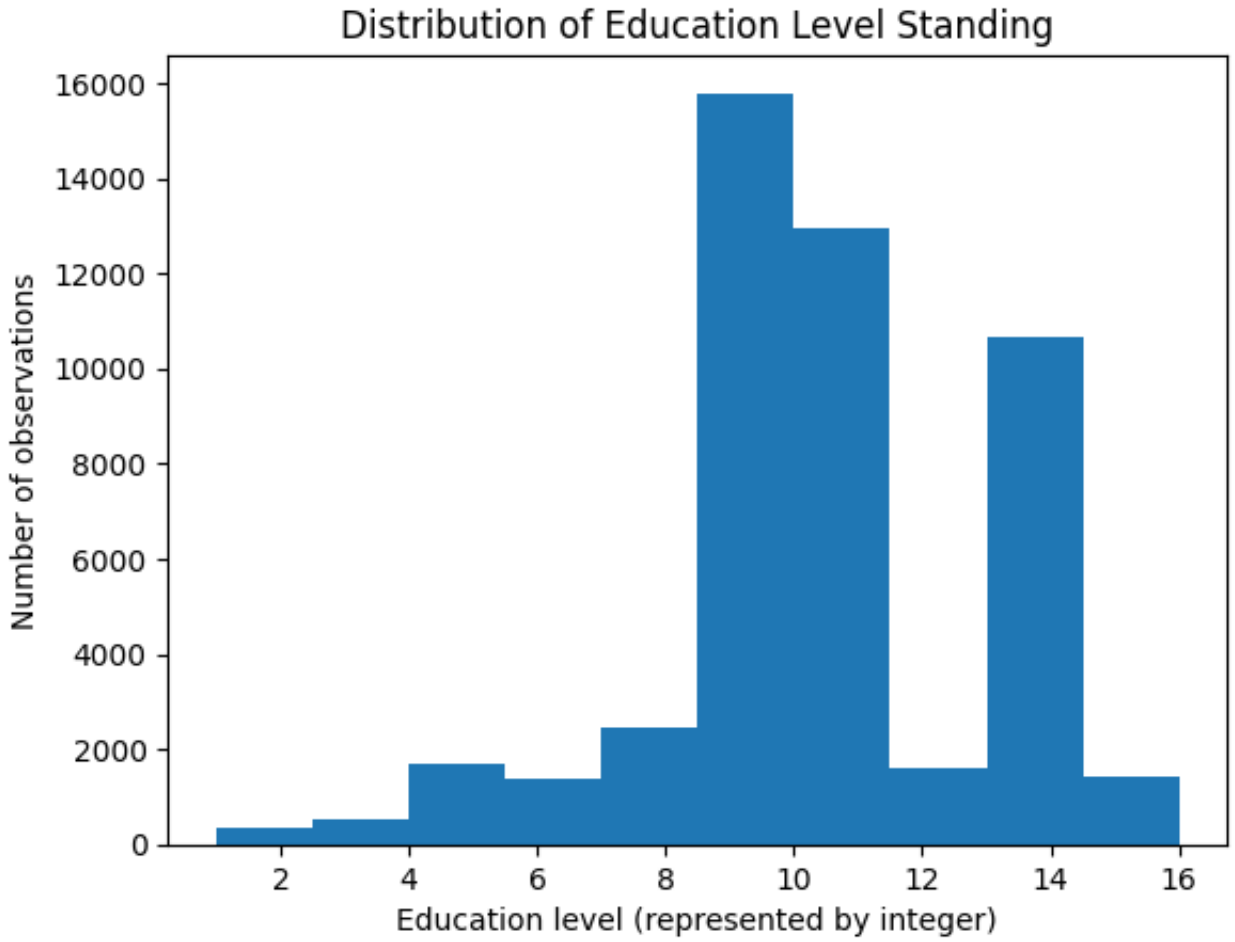


Explanation:

The horizontal stacked bar chart shows the proportion of occupations relative to their income if they make greater than 50k or less than 50k. Out of those who earn greater than 50k, the top three occupations are executive managerial, professional spciality, and sales careers. Out of those who earn less than 50k, the top three occupations are administration-clerical, craft-repair, and other services careers. This provides greater insight on which careers provide greater or less income.

```
#Visual 5: histogram showing distribution of amount of education completed

# education-num represents degree of education
```

```
# 1 = preschool      5 = 9th      9 = HS Grad         13 = Bachelors
# 2 = 1st-4th        6 = 10th    10 = Some-college    14 = Masters
# 3 = 5th-6th        7 = 11th    11 = Assoc-voc       15 = Prof-school
# 4 = 7th-8th        8 = 12th    12 = Assoc-acdm      16 = Doctorate

edu_num  = orig_df['education-num']

#note: the original df['education-num] will calculate the proportion of it equal
edu_num.head() #originally 12.8, 12.8, 8.53, 6.4, 12.8


plt.hist(edu_num)
plt.title("Distribution of Education Level Standing")
plt.xlabel("Education level (represented by integer)")
plt.ylabel("Number of observations")

plt.show()
```



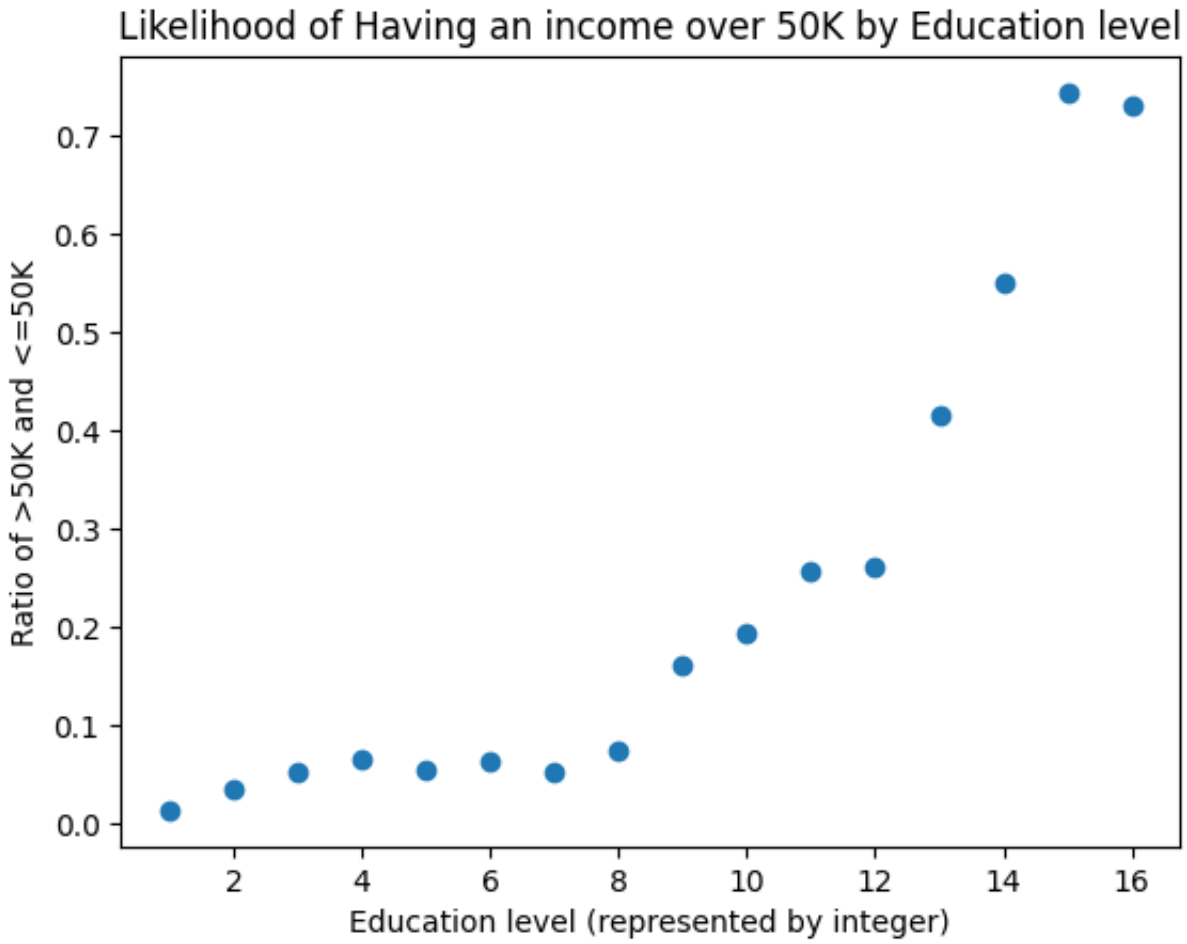
As seen over here, a vast majority of subjects surveyed either completed high school entirely, some college, or got an associate's degree (represented by an education level number of 9 to 11). However, others might opt for a 4-year Bachelor's degree, which, as shown in the histogram, is also more commonly represented (more so than the associate's degree part of it) than other demographics. While this information is useful for telling us where the majority stands in terms of education level, we are also dealing with categorical data, meaning the numbers are only assigned due to amount of time spent getting to that degree of education ('13' is Bachelor's since it would take around 4 years of education, while '14' is Master's since it would take around 6). Since it's categorical and the numbers have no valid meaning, it would be difficult to model the correlation between education level and income. To refute that point, however, these numbers are the best way of measuring different degrees of education, and so we can use it to model the effort spent vs income received.

This data is also not normally distributed; it is noticeably left-skewed as seen in the histogram since most respondents completed their education beyond high school.

```
#Visual 6: Education level vs Income
binary_income = df["incomes"]
binary_income = binary_income.replace([">50K", "<=50K"], [1, 0])
#binary_income

edu_num_tab = pd.crosstab(edu_num, binary_income)
edu_num_tab
```

incomes	0	1
education-num		
1	77	1
2	231	8
3	468	26
4	852	60
5	695	40
6	1251	85
7	1656	90
8	586	47
9	12970	2474
10	8471	2041
11	1513	521
12	1159	407
13	4608	3273
14	1176	1434
15	210	609
16	157	425

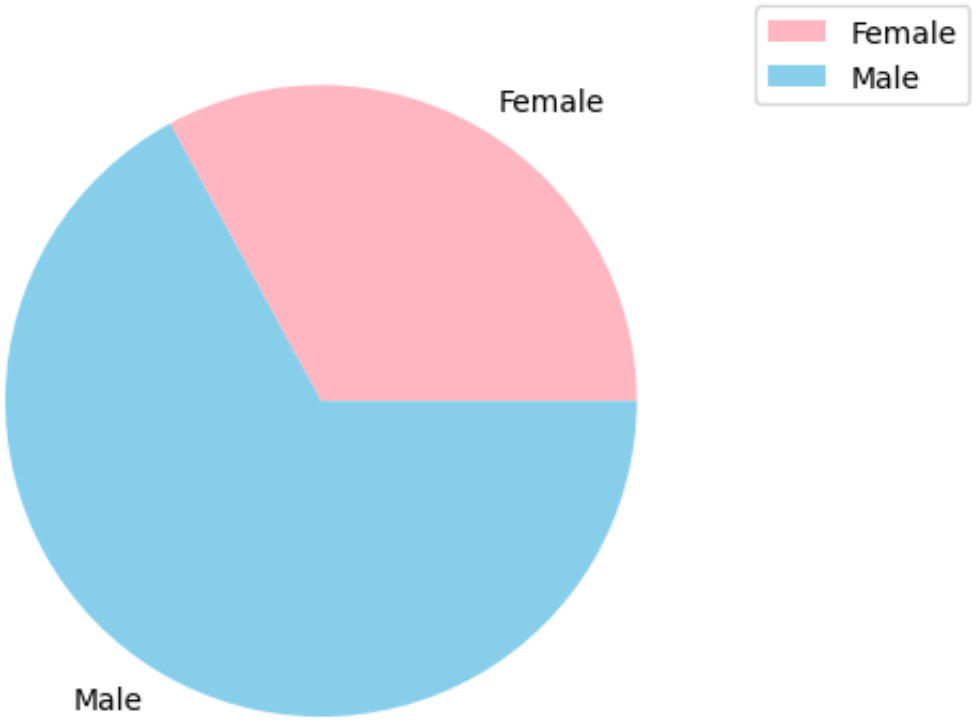
```
edu_num_tab['mean'] = edu_num_tab[1] / (edu_num_tab[0] + edu_num_tab[1])
#a, b = np.polyfit(edu_num_tab.index, edu_num_tab['mean'], 1)

plt.scatter(edu_num_tab.index, edu_num_tab['mean'])

plt.title("Likelihood of Having an income over 50K by Education level")
plt.xlabel("Education level (represented by integer)")
plt.ylabel("Ratio of >50K and <=50K")
#plt.plot(edu_num_tab.index, a* edu_num_tab.index +b)
plt.show()
```



As seen in the scatter plot above (which models the education level and the likelihood of having a higher income), it seems as though education level and income are positively correlated. However, it seems as though a straight line is not the best way to model the relationship as it seems to increase much more beyond a bachelor's degree.

Most higher paying jobs require at least a Bachelor's degree, while much higher paying jobs usually require a Master's degree or higher. Those without a degree can mostly afford low- to minimum wage jobs, which explains why a majority of those wihout college degrees are making under 50K a year.

```
#Visual 7: Distribution of Gender
gender_tab = pd.crosstab(df["sex"],df["incomes"])
gender_tab["total"] = gender_tab["<=50K"] + gender_tab[">50K"]

genders = ["Female", "Male"]
gen_colors = ["lightpink", "skyblue"]
plt.pie(gender_tab["total"], labels = genders, colors = gen_colors)
plt.legend(bbox_to_anchor=(1.05, 1),loc='upper left', borderaxespad=0.)
plt.show()
```



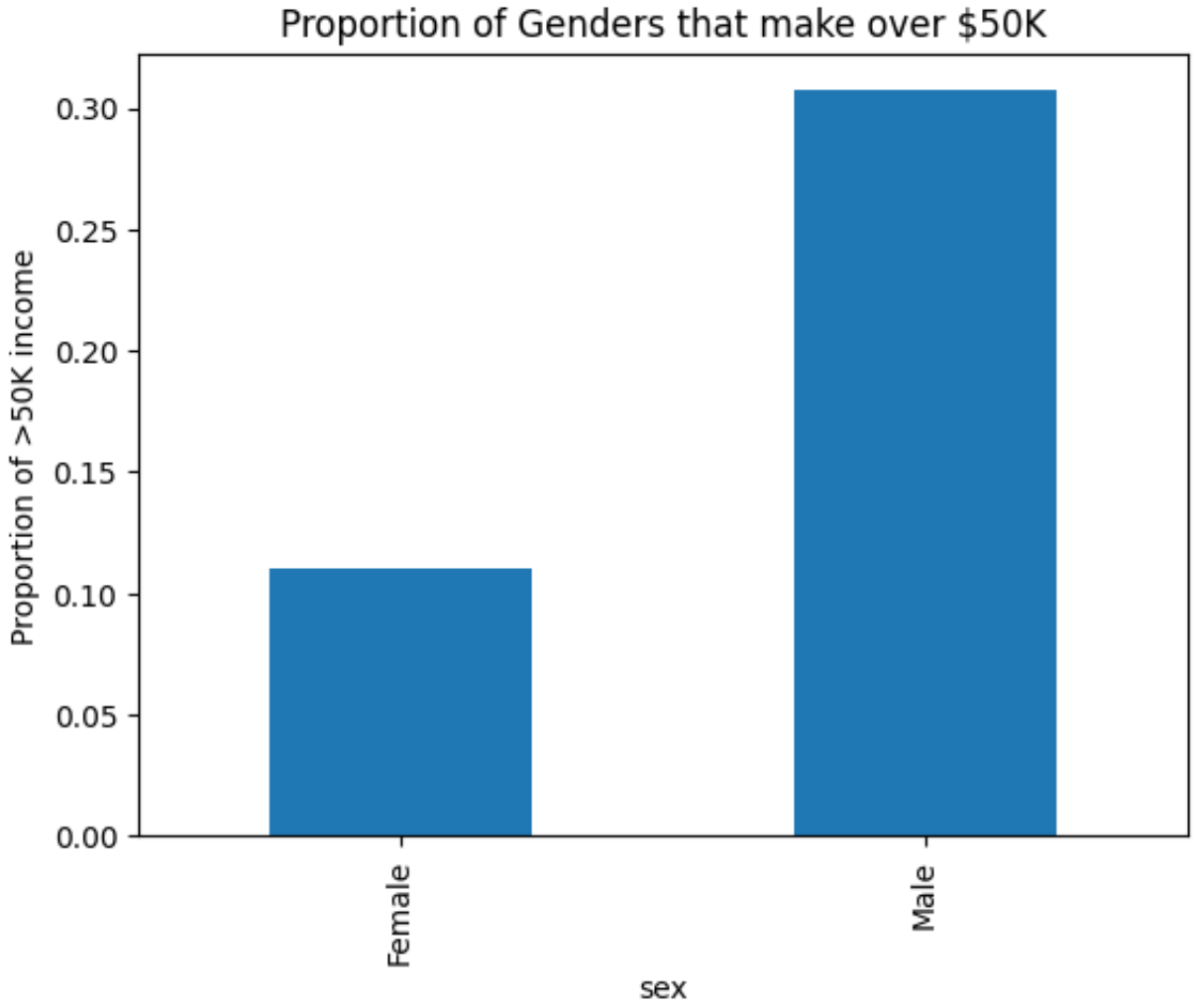
```
gender_tab
```

incomes	<=50K	>50K	total
sex			
Female	13948	1736	15684
Male	22132	9805	31937

As seen in both the pie chart and table above, males are much more represented in the dataset compared to females. There are about two men per woman within the dataset, and so inferences about gender may apply more to men than women (may be biased and therefore not generalizable).

```
#Visual 8: Gender vs pay
hi_income_given_gender = gender_tab[">50K"] / gender_tab["total"]

plt.title("Proportion of Genders that make over $50K")
plt.xlabel("Gender")
plt.ylabel("Proportion of >50K income")
hi_income_given_gender.plot.bar()
```

```
<Axes: title={'center': 'Proportion of Genders that make over $50K'},
xlabel='sex', ylabel='Proportion of >50K income'>
```

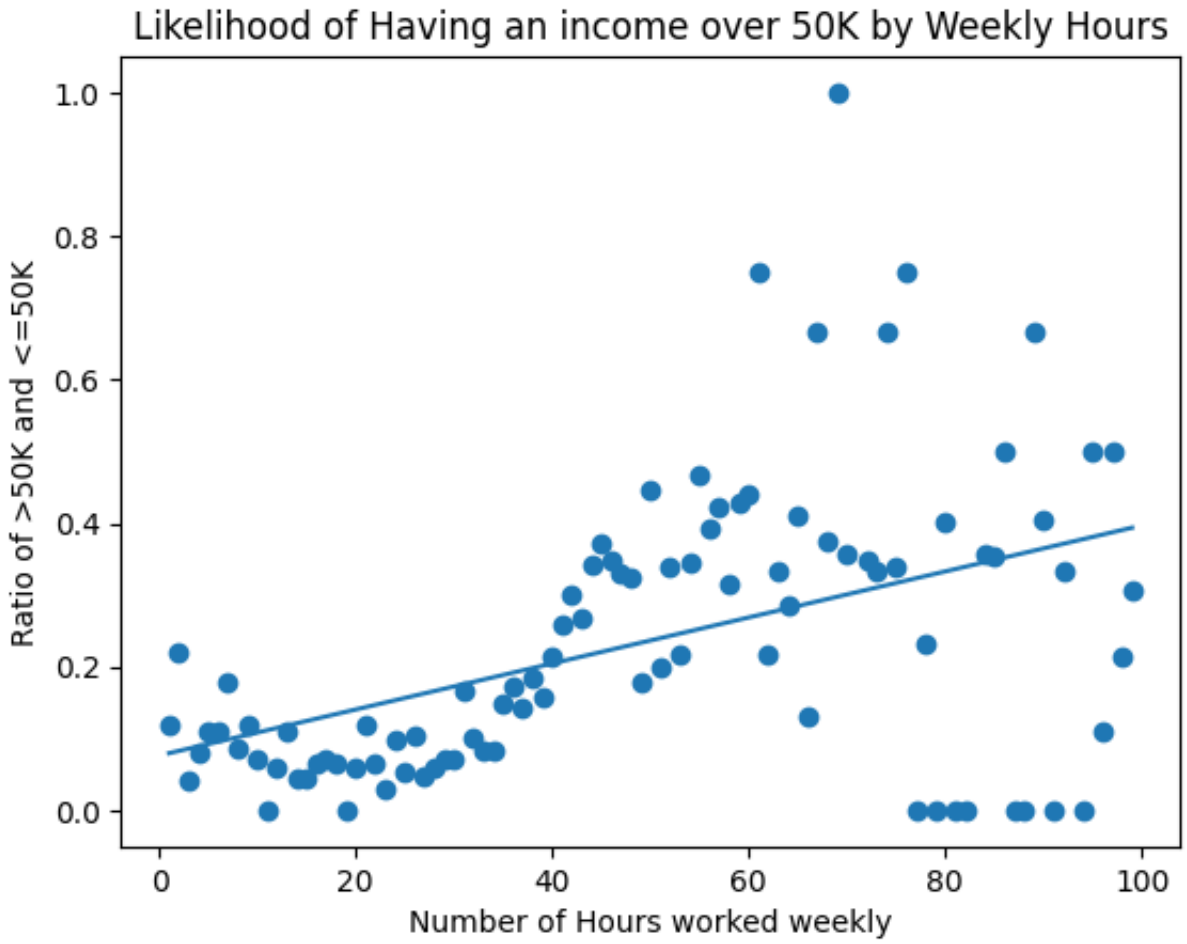


With the bar graph showing it is evident that males from this sample population are much more likely than females to make over 50K yearly. Only about 11% of women make more than 50K in this sample while about 31% of men make over 50K.

```
#Visual 9: Weekly hours distribution
hour_wk_tab = pd.crosstab(df["hours-per-week"], binary_income)
hour_wk_tab.tail()
```

incomes	0	1
hours-per-week		
95	1	1
96	8	1
97	1	1
98	11	3
99	90	40

```
hour_wk_tab['mean'] = hour_wk_tab[1] / (hour_wk_tab[0] + hour_wk_tab[1])
c, d = np.polyfit(hour_wk_tab.index, hour_wk_tab['mean'], 1)

plt.scatter(hour_wk_tab.index, hour_wk_tab['mean'])

plt.title("Likelihood of Having an income over 50K by Weekly Hours")
plt.xlabel("Number of Hours worked weekly")
plt.ylabel("Ratio of >50K and <=50K")
plt.plot(hour_wk_tab.index, c* hour_wk_tab.index +d)
plt.show()
```



As depicted by the regression line, hours worked and likelihood of having an income of >50K are weakly positively correlated. While it is true that more hours means more money, it does not consider the overall hourly wage. For example, some people make over 50K despite not working at all during the week. There are also those working between 60 to 80 hours making more than 50K while those who work close to 100 hours are less likely to make more than 50K.

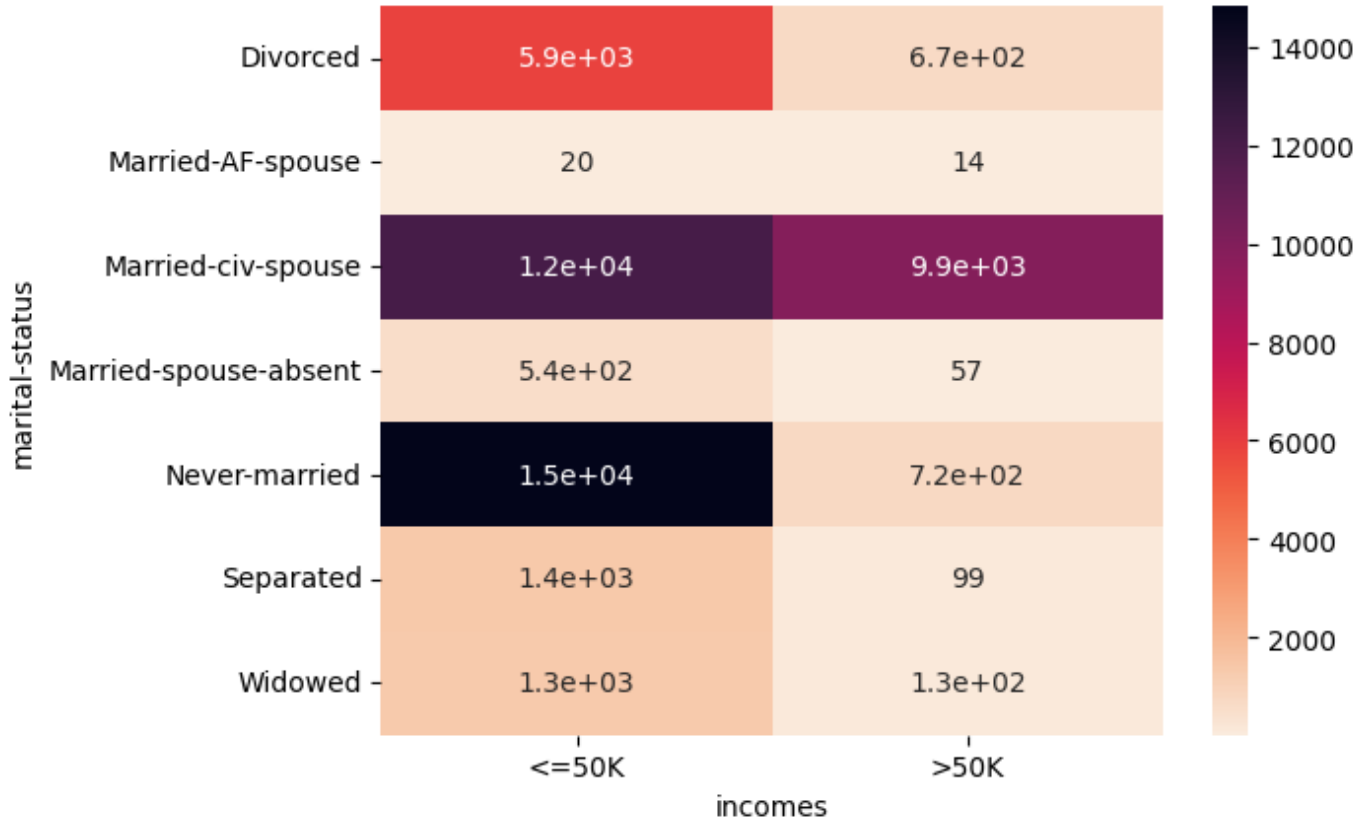
Keeping in mind that most of the people in the data make 50K or less per year, and that most people work 20-40 hours per week, making assumptions and conclusions about higher-hour or higher-salary individuals won't be reliable (some are considered outliers that impact the graph, in this case).

```
#Visual 10: marital status vs income
married_income = pd.crosstab(df["marital-status"], df["incomes"])
married_income
```

incomes	<=50K	>50K
marital-status		
Divorced	5860	666
Married-AF-spouse	20	14
Married-civ-spouse	12109	9857
Married-spouse-absent	543	57
Never-married	14833	722
Separated	1398	99
Widowed	1317	126

```
%matplotlib inline
sns.heatmap(married_income, annot = True, cmap = "rocket_r")
#note: AF means armed force, civ is civilian
```

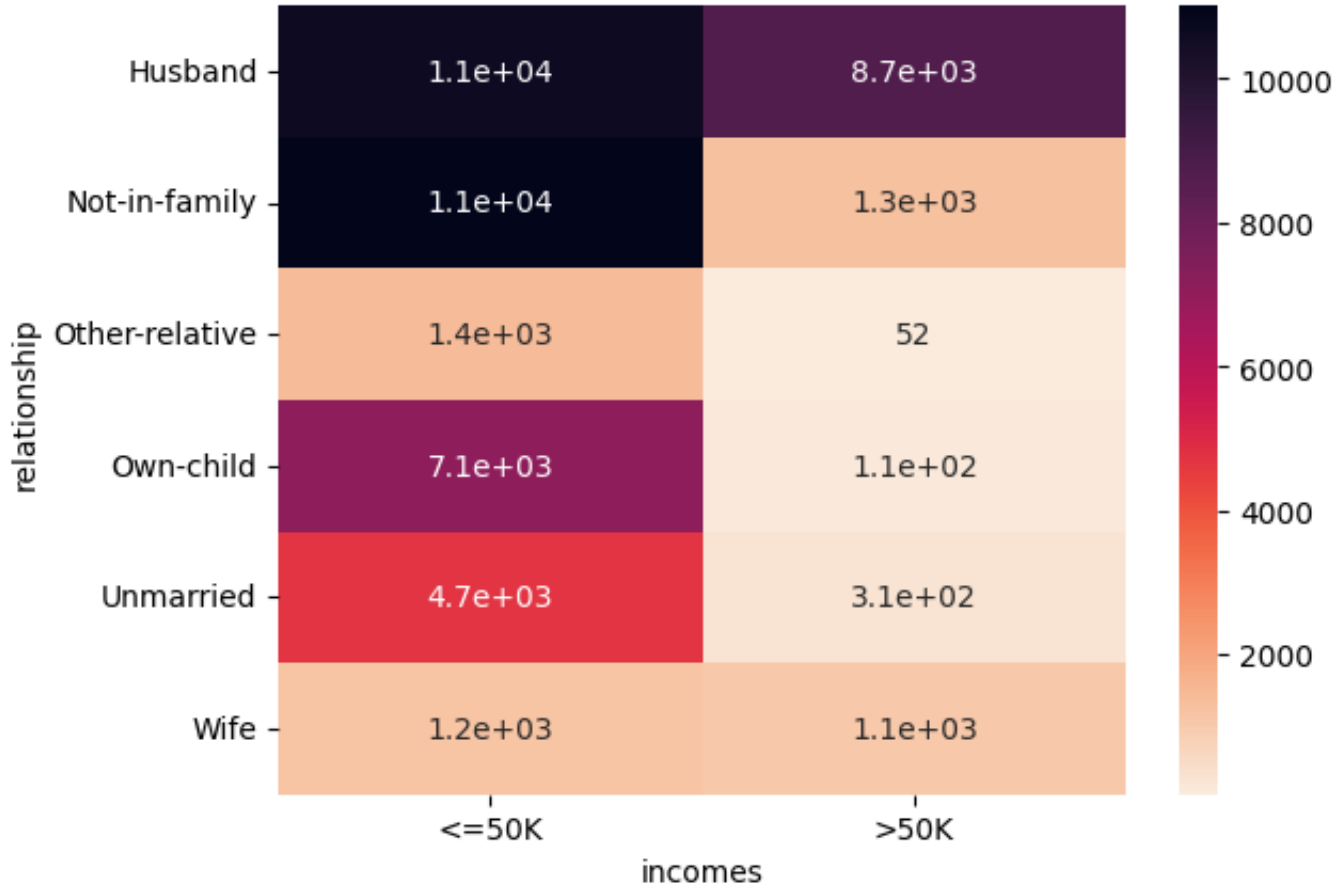
```
<Axes: xlabel='incomes', ylabel='marital-status'>
```



Basically what this heatmap tells us is that those who never married and are making less than 50K yearly are the most commonly represented group within the dataset. Those who ARE making more than 50K are most likely to be married WITH a spouse (though slightly more married-with-spouse individuals will be making 50K or less), because they likely have families to raise as well.

```
#Visual 11: Relationship status vs Income
relation_income = pd.crosstab(df["relationship"], df["incomes"])
sns.heatmap(relation_income, annot = True, cmap = "rocket_r")
```

```
<Axes: xlabel='incomes', ylabel='relationship'>
```



In general, men who are husbands are the most represented group, and are the most likely to make more than 50K out of any other group. Other groups' majorities are noticeably making less than 50K, with the exception of women who are wives, in which roughly half make more than 50K while the other half of wives make less than or equal to 50K. Married men are likely to make more money in order to provide for the entire family due to traditional roles.

```
#Visual 12: Distribution of Countries and their income
cleaned_df3 = cleaned_df[cleaned_df["native-country"] != "?"]
cleaned_df3['obs'] = range(1, len(cleaned_df3) + 1)
countries = cleaned_df3.pivot_table(
    index = "incomes", columns = "native-country",
    values = "obs", aggfunc = "count"
)
countries
```

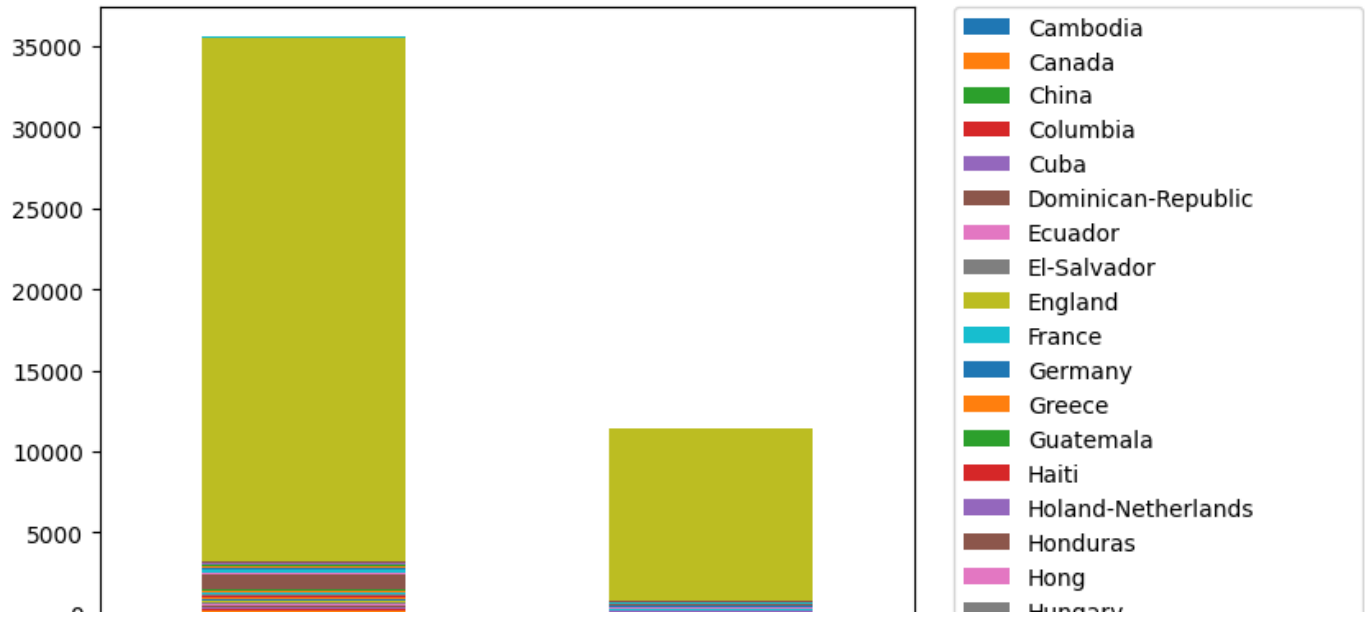
```
<ipython-input-23-ee87b9352fea>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  cleaned_df3['obs'] = range(1, len(cleaned_df3) + 1)
```

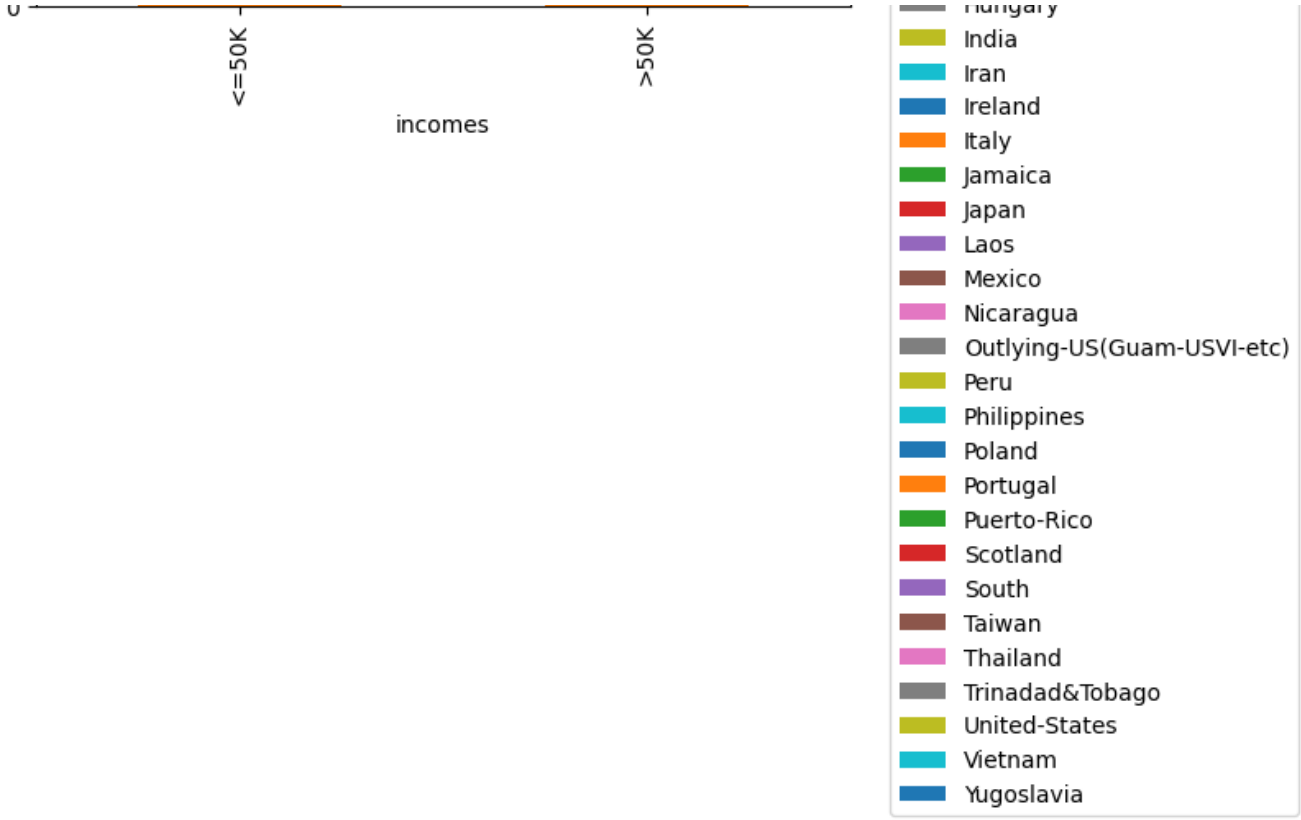
native-country	Cambodia	Canada	China	Columbia	Cuba	Dominican-Republic	Ecuador	El-Salvador
incomes								
<=50K	18.0	114.0	84.0	81.0	102.0	95.0	38.0	142.0
>50K	9.0	63.0	36.0	4.0	34.0	5.0	6.0	11.0

2 rows × 41 columns

```
countries.plot.bar(stacked = True).legend(bbox_to_anchor=(1.05, 1),loc='upper lef
```

```
<matplotlib.legend.Legend at 0x79d1523bbb50>
```





A vast majority of those surveyed are from the United States. Additionally, most of those surveyed make 50K or under.

Feature Selection

In addition to feature selection, we will also be normalizing numerical columns so that the range of each numerical attribure is between 0 and 1. This would be extremely helpful for models like KNN, where the distance calculations could be biased due to the varying ranges of different attributes.

We will also be quantifying our categorical columns with one-hot-encoding so each column is replaced with numerous columns with binary values. The number of binary value columns will of course depend on the number of unique categories the original column had.

```
df = cleaned_df.copy()


# Normalization

def normalize(x, min, max):
  return (x-min)/(max-min)

for cname in df.select_dtypes(include=['int']).columns:
  df[cname] = df[cname].apply(normalize, args=(df[cname].min(), df[cname].max()))


# One-Hot-Encoding
encoder = OneHotEncoder(handle_unknown = 'ignore')
for c in df.select_dtypes(include=['object']).columns:
    one_hot = pd.get_dummies(df[c], prefix=c)
    df = df.drop(c, axis=1)
    df = pd.concat([df, one_hot],axis=1)
    df[one_hot.columns] = df[one_hot.columns].astype(int)


print(df.columns)
```

```
Index(['age', 'education-num', 'capital-gain', 'capital-loss',
       'hours-per-week', 'workclass_?', 'workclass_Federal-gov',
       'workclass_Local-gov', 'workclass_Never-worked', 'workclass_Private',
       ...
       'native-country_Scotland', 'native-country_South',
       'native-country_Taiwan', 'native-country_Thailand',
       'native-country_Trinadad&Tobago', 'native-country_United-States',
       'native-country_Vietnam', 'native-country_Yugoslavia', 'incomes_<=50K'
       'incomes_>50K'],
      dtype='object', length=109)
```

```
forward_selection_features = ['capital-gain', 'relationship_Not-in-family', 'educ
chi_square_features = []
```

Forward Selection

(we commented the code out because running it takes a while; results saved below)

```
'''
X = df.drop(['incomes_<=50K','incomes_>50K'],axis=1)
y = df['incomes_>50K']

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_sta
feature_subset = set()

clf1 = LogisticRegression(random_state = 16)
clf2 = DecisionTreeClassifier(random_state = 16)
clf3 = KNeighborsClassifier(n_neighbors=np.sqrt(df.shape[0]).astype(int))
model = VotingClassifier(estimators=[('LR',clf1),('DT',clf2),('KNN',clf3)], votin

cur_best_score = 0;
while True:
    best_feature = None;
    for feature in X_train.columns:
      if feature not in feature_subset:
        feature_subset.add(feature)

        model.fit(X_train[list(feature_subset)], y_train)
        y_pred = model.predict(X_val[list(feature_subset)])
        score = accuracy_score(y_val, y_pred)
        if score >= cur_best_score:
          cur_best_score = score
          best_feature = feature

        feature_subset.remove(feature)

    if best_feature is None:
      break

    feature_subset.add(best_feature)
    print(f'Current Subset: [{list(feature_subset)}]')
    print(f'Validation acc: {cur_best_score}')

forward_selection_features = list(feature_subset)
```

'''

```
'\nX = df.drop(['incomes_<=50K','incomes_>50K'],axis=1)\ny = df['incomes_>50K
']\n\nX_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state = 16)\nfeature_subset = set()\n\nclf1 = LogisticRegression(rando
m_state = 16)\nclf2 = DecisionTreeClassifier(random_state = 16)\nclf3 = KNeig
hborsClassifier(n_neighbors=np.sqrt(df.shape[0]).astype(int))\nmodel = Voting
Classifier(estimators=[('LR',clf1),('DT',clf2),('KNN',clf3)], voting='soft')\
n\ncur_best_score = 0;\nwhile True:\n    best_feature = None;\n    for featur
e in X_train.columns:\n      if feature not in feature_subset:\n        featu
```

Forward Feature Selection printed results (saved bc re-running it will take a long time)

Current Subset: [['capital-gain']]

Validation acc: 0.8041994750656168

Current Subset: [['capital-gain', 'capital-loss']]

Validation acc: 0.8260367454068241

Current Subset: [['capital-gain', 'education_Doctorate', 'capital-loss']]

Validation acc: 0.8297112860892388

Current Subset: [['capital-gain', 'education_Doctorate', 'education_Prof-school', 'capital-loss']]

Validation acc: 0.8309711286089239

Current Subset: [['capital-gain', 'relationship_Not-in-family', 'education_Doctorate', 'education_Prof-school', 'capital-loss']]

Validation acc: 0.8313910761154856

Current Subset: [['education_Masters', 'capital-gain', 'relationship_Not-in-family', 'education_Doctorate', 'education_Prof-school', 'capital-loss']]

Validation acc: 0.8359055118110236

Current Subset: [['education_Masters', 'capital-gain', 'relationship_Not-in-family', 'education_Doctorate', 'education_Prof-school', 'marital-status_Married-civ-spouse', 'capital-loss']]

Validation acc: 0.8386351706036745

Current Subset: [['capital-gain', 'relationship_Not-in-family', 'education_Doctorate', 'education_Prof-school', 'marital-status_Married-civ-spouse', 'education_Masters', 'capital-loss', 'education_Bachelors']]

Validation acc: 0.8520734908136482

Current Subset: [['capital-gain', 'relationship_Not-in-family', 'education_Doctorate', 'education_Prof-school', 'marital-status_Married-civ-spouse', 'education_Masters', 'capital-loss', 'occupation_Farming-fishing', 'education_Bachelors']]

Validation acc: 0.8530183727034121

Current Subset: [['capital-gain', 'relationship_Not-in-family', 'education_Doctorate', 'education_Prof-school', 'marital-status_Married-civ-spouse', 'education_Masters', 'capital-loss', 'occupation_Transport-moving', 'occupation_Farming-fishing', 'education_Bachelors']]

Validation acc: 0.8539632545931759

Current Subset: [['capital-gain', 'relationship_Not-in-family', 'occupation_?', 'education_Doctorate', 'education_Prof-school', 'marital-status_Married-civ-spouse', 'education_Masters', 'capital-loss', 'occupation_Transport-moving', 'occupation_Farming-fishing', 'education_Bachelors']]

Validation acc: 0.8548031496062992

Best Accuracy: 0.8548

Explanation:

Considering the optimal feature subset for one model may not be the exact same as the optimal subset for another model, we decided to use a VotingClassifier (with all models we plan to use) to determine the best feature subset. We believed with a VotingClassifier, every model we plan to use will contribute to determining the feature subset, resulting in the most optimal feature subset overall.

The forward feature selection was stopped as soon as new feature additions stop improving the accuracy score by 0.001 or 0.1% two consecutive times. This is to ensure that the accuracy score has plateaued and we do not add unnesscessary dimensions to our training data

The selected features are ['capital-gain', 'relationship_Not-in-family', 'education_Doctorate', 'education_Prof-school', 'marital-status_Married-civ-spouse', 'education_Masters', 'capital-loss', 'occupation_Transport-moving', 'occupation_Farming-fishing', 'education_Bachelors'] with the 'capital' and 'education' related features yielding higher accuracy score improvement with their additions.

Chi-Squared Selection

```
X = df.drop(['incomes_<=50K','incomes_>50K',
             'age', 'education-num', 'capital-gain', 'capital-loss','hours-per-we
y = df['incomes_>50K']
chi_square_features += ['age', 'education-num', 'capital-gain', 'capital-loss','h

x2_DF = pd.DataFrame()

for c in X.columns:
    contingency_table = pd.crosstab(df[c],y)
    chi, p, dof, exp = chi2_contingency(contingency_table)
    d = {'Feature Name':[c], 'Chi Square Value':[chi], 'P Value':[p], 'DoF':[dof]
    new_row = pd.DataFrame(d)
    x2_DF = pd.concat([x2_DF, new_row], axis=0, ignore_index = True)


x2_DF = x2_DF.sort_values(['Chi Square Value'], ascending= [False])
x2_DF
```

	Feature Name	Chi Square Value	P Value	DoF
27	marital-status_Married-civ-spouse	9456.646863	0.000000	1
47	relationship_Husband	7750.793420	0.000000	1
29	marital-status_Never-married	4828.284563	0.000000	1
50	relationship_Own-child	2398.552847	0.000000	1
59	sex_Male	2206.880028	0.000000	1
...	...	...	...	...
81	native-country_Ireland	0.091017	0.762888	1
77	native-country_Hong	0.041832	0.837941	1
65	native-country_Cuba	0.011722	0.913784	1
34	occupation_Armed-Forces	0.004462	0.946741	1
75	native-country_Holand-Netherlands	0.000000	1.000000	1

102 rows × 4 columns

```
chi_square_features += list(x2_DF.head(10)['Feature Name'])
print(chi_square_features)
```

```
['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week', 'm
```

Explanation:

Because there are a lot of categorical attributes in our data, we also decided to use the chi-square of independence for feature selection by selecting the attributes with the highest chi-square value and share the most significant relationship with our target variable (>=$50 income)

However, since chi-square test cannot be done on our numerical columns, we decided to include every numerical features (which is not a lot) into our final subset of features so we do not drop potentially useful numerical attributes.

Chi-square test is done on each cateogrical column of the dataframe post one-hot-encoding. The 10 features with the highest chi-square value/lowest p-value is added to the final subset, completing the feature selection.

One worry is that the chi-square test does not work well with large sample sizes (we do have around 40,000 entries for each column). With a large sample size, any slight difference will be considered statistically significant. The resulting chi-square values are extremely high and p-values are extremely low (as shown in the table above). However, even though the chi-square and p-values may not be reliable, we believe they are still able to show relative significance between all the features.

Supervised Learning Models

```
forward_selection_features
```

```
['capital-gain',
 'relationship_Not-in-family',
 'education_Doctorate',
 'education_Prof-school',
 'marital-status_Married-civ-spouse',
 'education_Masters',
 'capital-loss',
 'occupation_Transport-moving',
 'occupation_Farming-fishing',
 'education_Bachelors']
```

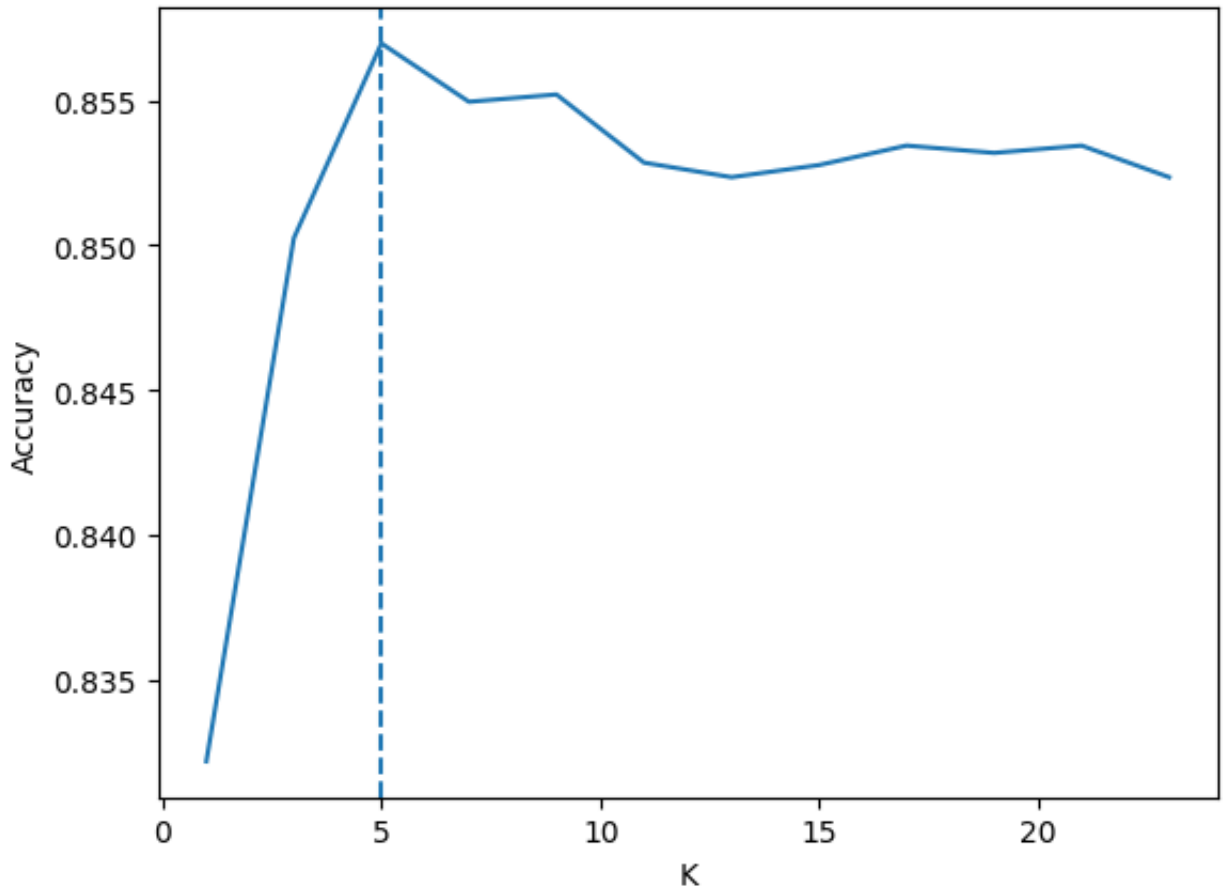
```
chi_square_features
```

```
['age',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'marital-status_Married-civ-spouse',
 'relationship_Husband',
 'marital-status_Never-married',
 'relationship_Own-child',
 'sex_Male',
 'sex_Female',
 'occupation_Exec-managerial',
 'relationship_Not-in-family',
 'occupation_Prof-specialty',
 'education_Bachelors']
```

KNN (Forward Selection Features)

```
def find_best_k(X, y, max_k):
  accuracies = []
  max_accuracy = 0
  ideal_k = 0;

  for k in range(1,max_k,2):

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ran
    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    score = sk_accuracy_score(y_test, y_pred)
    accuracies.append(score)

    if score > max_accuracy:
      max_accuracy = score
      ideal_k = k

  plt.xlabel("K")
  plt.ylabel('Accuracy')
  plt.plot(range(1,max_k,2),accuracies)
  plt.axvline(x=ideal_k, linestyle='--')
  plt.show()
  print(f'Ideal K value: {ideal_k}')

  return ideal_k
```

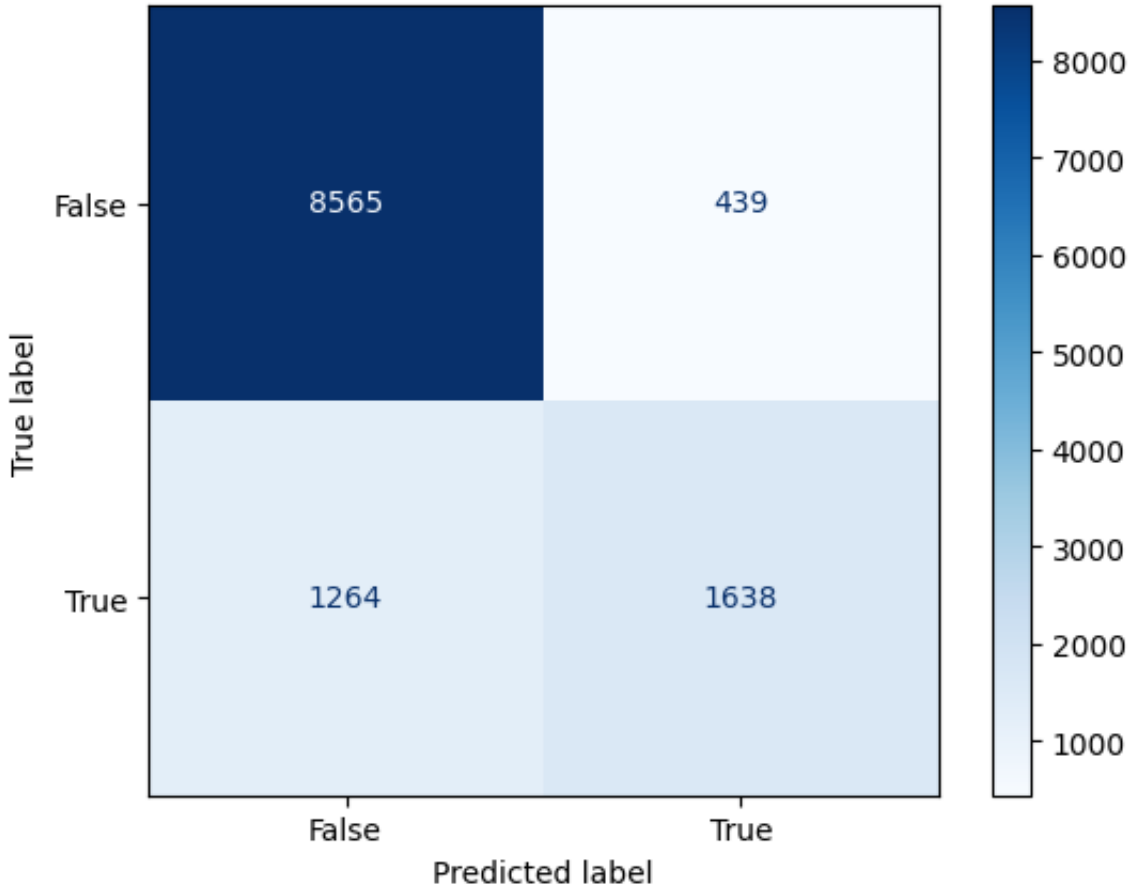
```
X = df[forward_selection_features]
y = df['incomes_>50K']

ideal_k = find_best_k(X, y, 25)
```




```
Ideal K value: 5
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_
knn_ff = KNeighborsClassifier(n_neighbors=ideal_k)

knn_ff.fit(X_train, y_train)
y_pred = knn_ff.predict(X_test)

print('Confusion Matrix')
ConfusionMatrixDisplay.from_estimator(knn_ff, X_test, y_test,cmap='Blues',display
plt.show()

print(classification_report(y_test, y_pred))
```

Confusion Matrix



	precision	recall	f1-score	support
0	0.87	0.95	0.91	9004
1	0.79	0.56	0.66	2902
accuracy			0.86	11906
macro avg	0.83	0.76	0.78	11906
weighted avg	0.85	0.86	0.85	11906

Explanation:

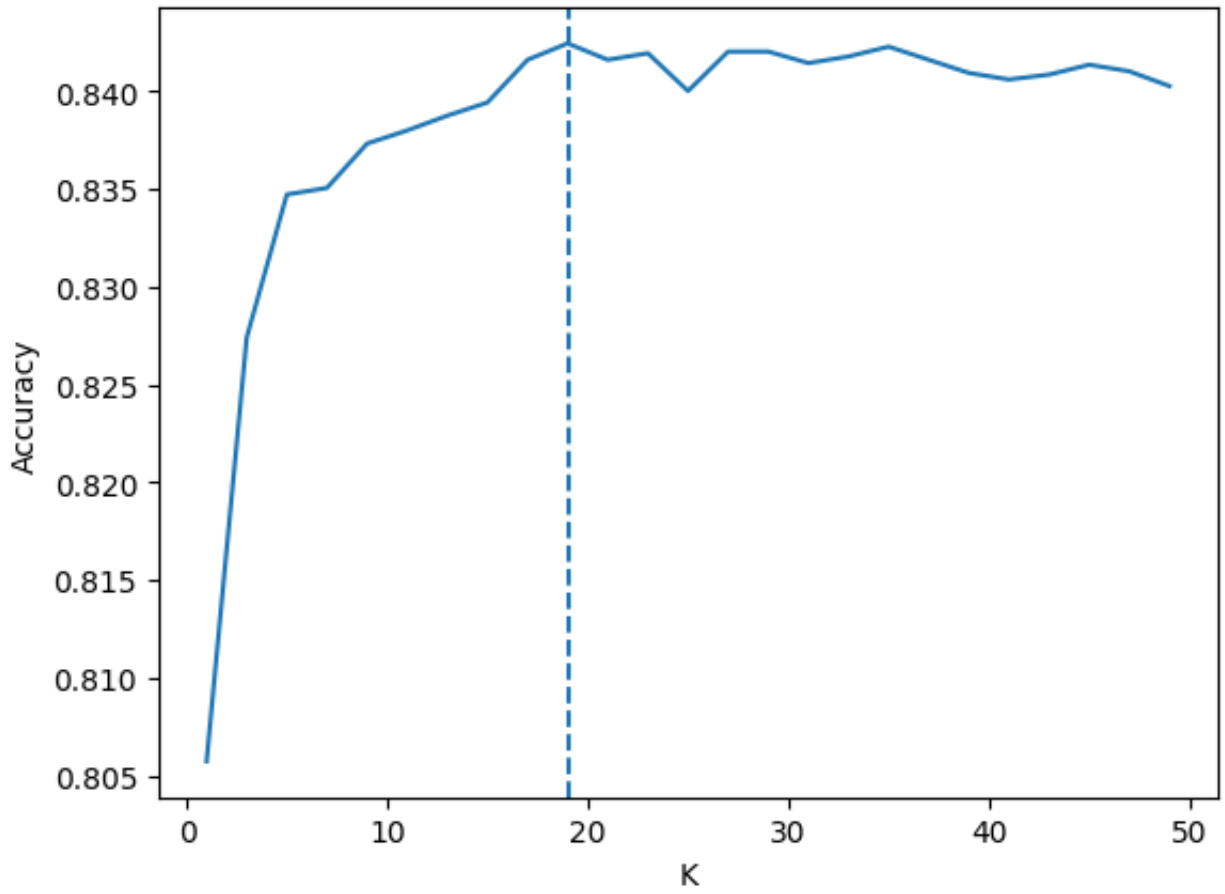
With the features selected by forward subset selection, KNN managed to achieve a maximum accuracy of 0.86. Looking at the classification report and the confusion matrix, the most concerning thing is the low recall rate for >=50k income. This could have been due to the low sample size of < 50k income. If the data was tested on a different set of data with mostly high income entries, we could see a large drop in accuracy for this model.

For selecting the k value, we ran the KNN model with odd values ranging from 1 to 25. We originally wanted to run with values ranging all the way to the maximum possible k value based on our data but that would take too long. After running knn on values ranging from 1 to 100. We found that 1 to 25 sufficiently shows the end behavior of the accuracy vs k graph.

We used 5 as our ideal k value as it resulted in the highest accuracy.

KNN (Chi-Squared Features)

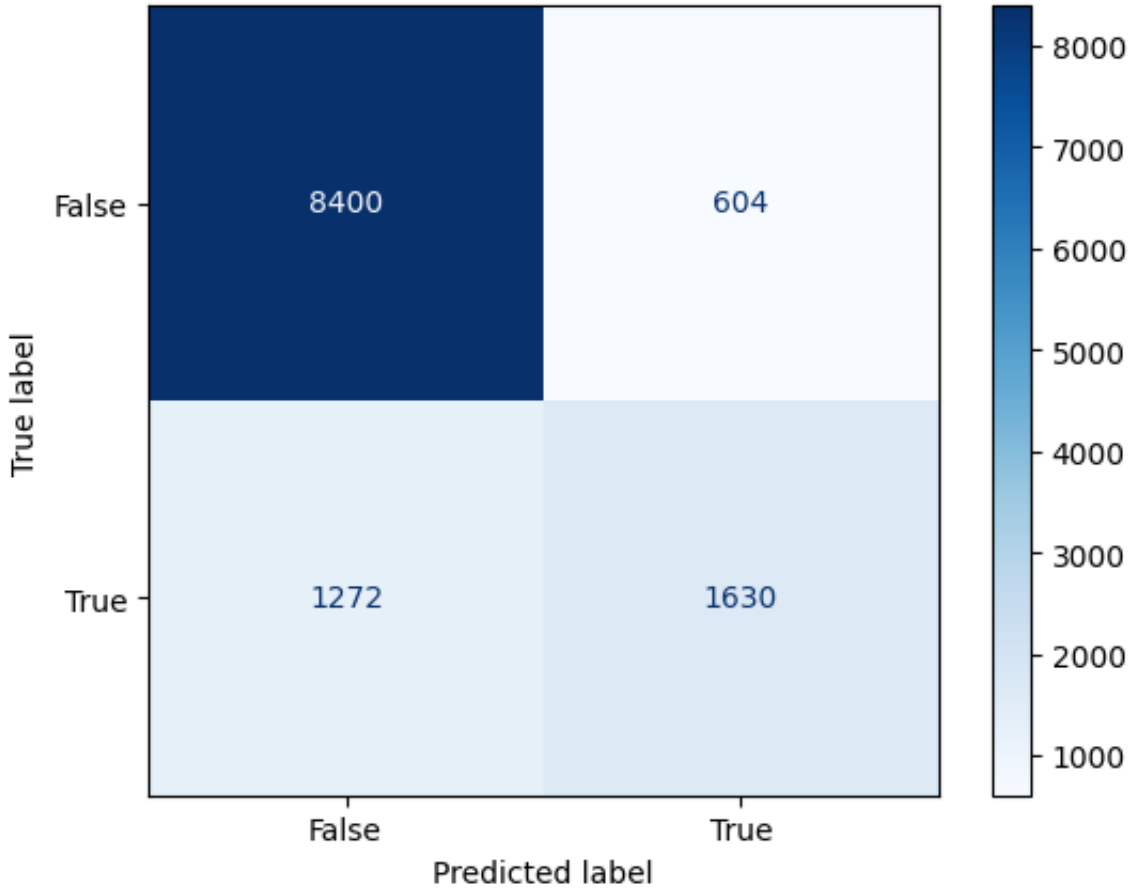
```
X = df[chi_square_features]
y = df['incomes_>50K']

ideal_k = find_best_k(X, y, 50)
```




```
Ideal K value: 19
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_
knn_chi = KNeighborsClassifier(n_neighbors=ideal_k)

knn_chi.fit(X_train, y_train)
y_pred = knn_chi.predict(X_test)

print('Confusion Matrix')
ConfusionMatrixDisplay.from_estimator(knn_chi, X_test, y_test,cmap='Blues',displa
plt.show()

print(classification_report(y_test, y_pred))
```

Confusion Matrix



	precision	recall	f1-score	support
0	0.87	0.93	0.90	9004
1	0.73	0.56	0.63	2902
accuracy			0.84	11906
macro avg	0.80	0.75	0.77	11906
weighted avg	0.83	0.84	0.84	11906

Explanation:

With features selected from chi square test, we got a model with an accuracy of 0.84, which is slightly lower than the KNN trained with forward selected feature subset. This could be due the higher number of features in the chi square test subset, resulting in higher dimensionality (which could negatively affect KNN's performance).

Like the previous KNN, this KNN model suffers the same low recall rate for >=50k income, which could be a problem if the models was run on a different dataset with higher numbers of >=50k income entries.

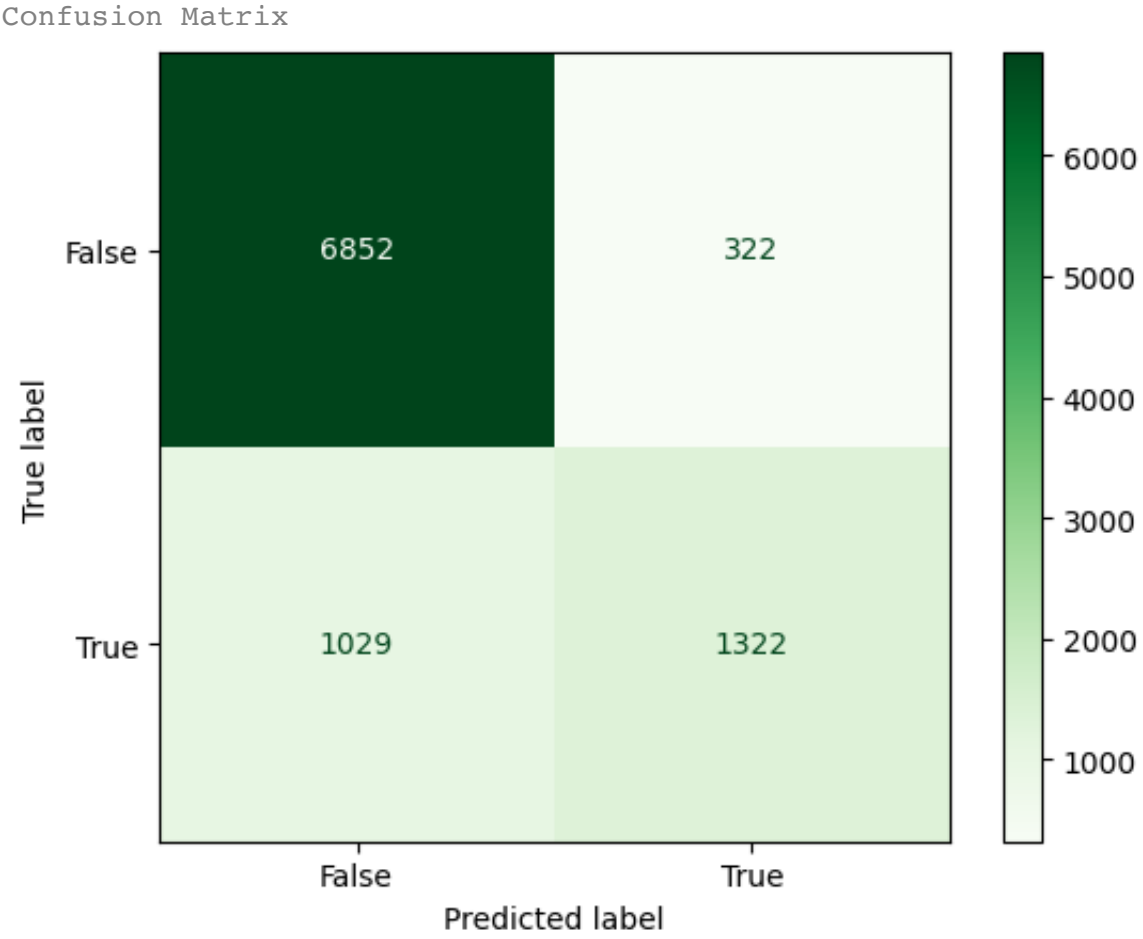
For selecting the k value, we ran the KNN model with odd values ranging from 1 to 50. We originally wanted to run with values ranging all the way to the maximum possible k value based on our data but that would take too long. After running knn on values ranging from 1 to 100. We found that 1 to 50 sufficiently shows the end behavior of the accuracy vs k graph and would allow us to find a maximum in accuracy.

We used 19 as our ideal k value as it resulted in the highest accuracy.

Voting Classifier (Forward Selection Features)

```
def train_voting_classifier(X,y):
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

  clf1 = LogisticRegression(random_state = 16)
  clf2 = DecisionTreeClassifier(random_state = 16)
  clf3 = KNeighborsClassifier(n_neighbors=5)
  vc = VotingClassifier(estimators=[('LR',clf1),('DT',clf2),('KNN',clf3)], voting

  vc.fit(X_train, y_train)
  y_pred = vc.predict(X_test)

  print('Confusion Matrix')
  ConfusionMatrixDisplay.from_estimator(vc, X_test, y_test,cmap='Greens',display_
  plt.show()

  print(classification_report(y_test, y_pred))
```

```
X = df[forward_selection_features]
y = df['incomes_>50K']
train_voting_classifier(X,y)
```



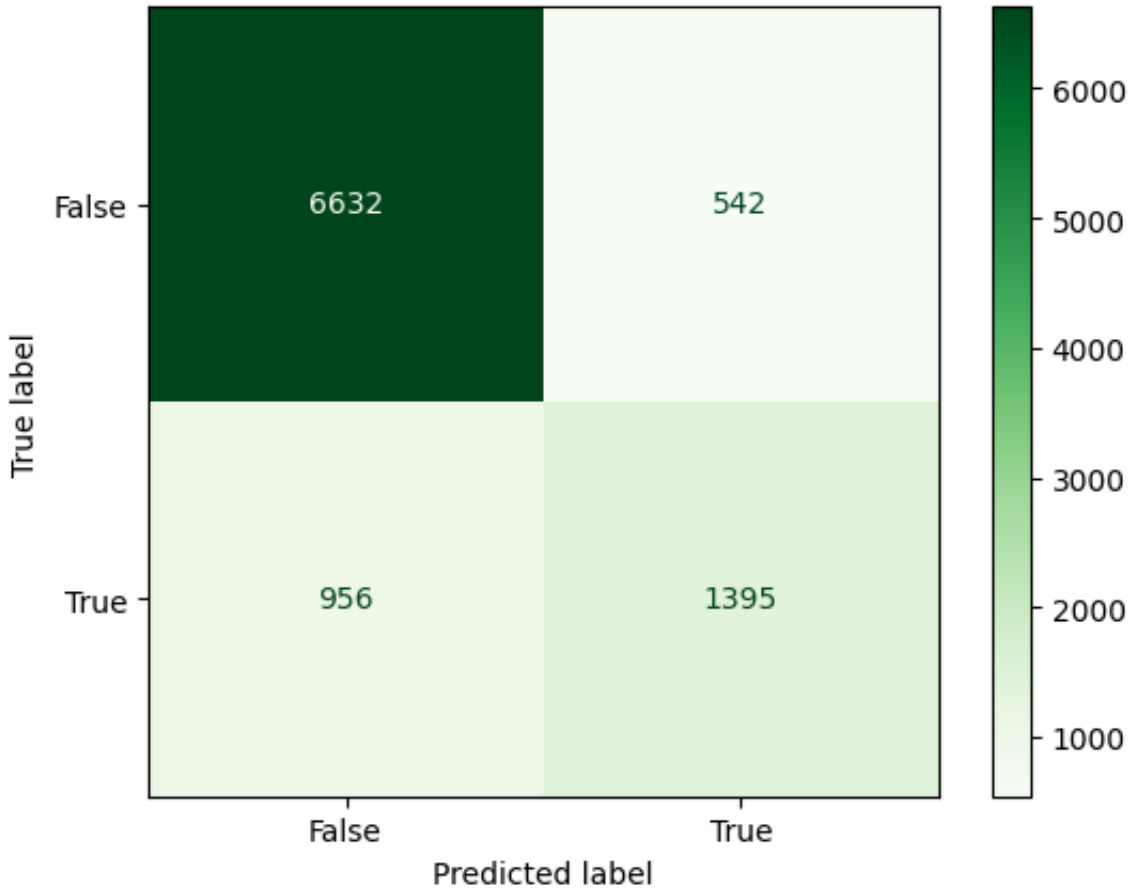
	precision	recall	f1-score	support
0	0.87	0.96	0.91	7174
1	0.80	0.56	0.66	2351
accuracy			0.86	9525
macro avg	0.84	0.76	0.79	9525
weighted avg	0.85	0.86	0.85	9525

Voting Classifier (Chi Square Features)

```
X = df[chi_square_features]
y = df['incomes_>50K']
```

```
train_voting_classifier(X,y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
Confusion Matrix
```



	precision	recall	f1-score	support
0	0.87	0.92	0.90	7174
1	0.72	0.59	0.65	2351
accuracy			0.84	9525
macro avg	0.80	0.76	0.77	9525
weighted avg	0.84	0.84	0.84	9525

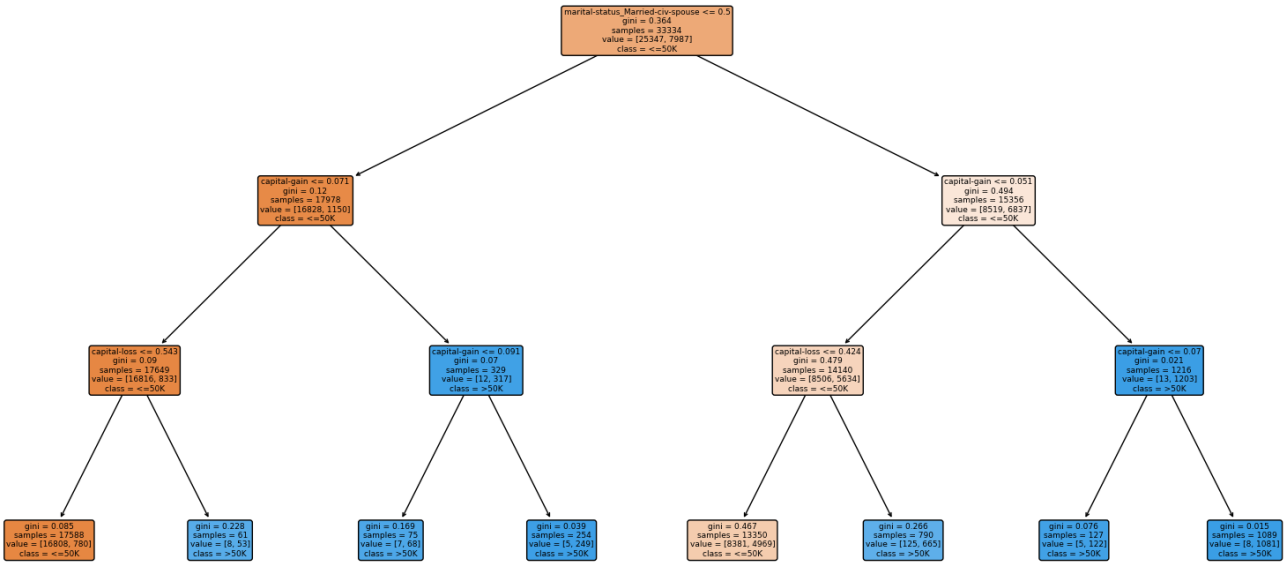
Explanation:

For the voting classifier, we ran it with both features subset selected through forward selection and chi square test selection. We found that the voting classifier trained with forward selected features have a higher accuracy (higher by 0.02). Like the previous KNN models, there is a low recall rate for >=50k income, which is a prevalent problem for many of our models. This could be due to the low amount of >=50k income entries.

We included a logistic regression model, a KNN model, and a decision tree model in our voting classifer. For the KNN model, we used the ideal K value with based on the feature subset (determined previously). We tried both soft voting and hard voting and found no significant difference in accuracy and classification report (soft voting depends on each model's probabiility score while hard voting depends on each model's actual prediction)

Decision Tree

```
# Assume 'df' is your DataFrame and it's already been loaded correctly.

X = df[forward_selection_features]
y = df['incomes_>50K']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

# Initialize the DecisionTreeClassifier with specific constraints
model = DecisionTreeClassifier(max_depth=3, min_samples_split=100, min_samples_le
model = model.fit(X_train, y_train)

feature_names_list = X.columns.tolist()

# Now use plot_tree with the converted list of feature names
plt.figure(figsize=(20, 10))  # Set the size of the figure
plot_tree(model,
          feature_names=feature_names_list,
          class_names=['<=50K', '>50K'],
          filled=True,
          rounded=True)
plt.show()
```

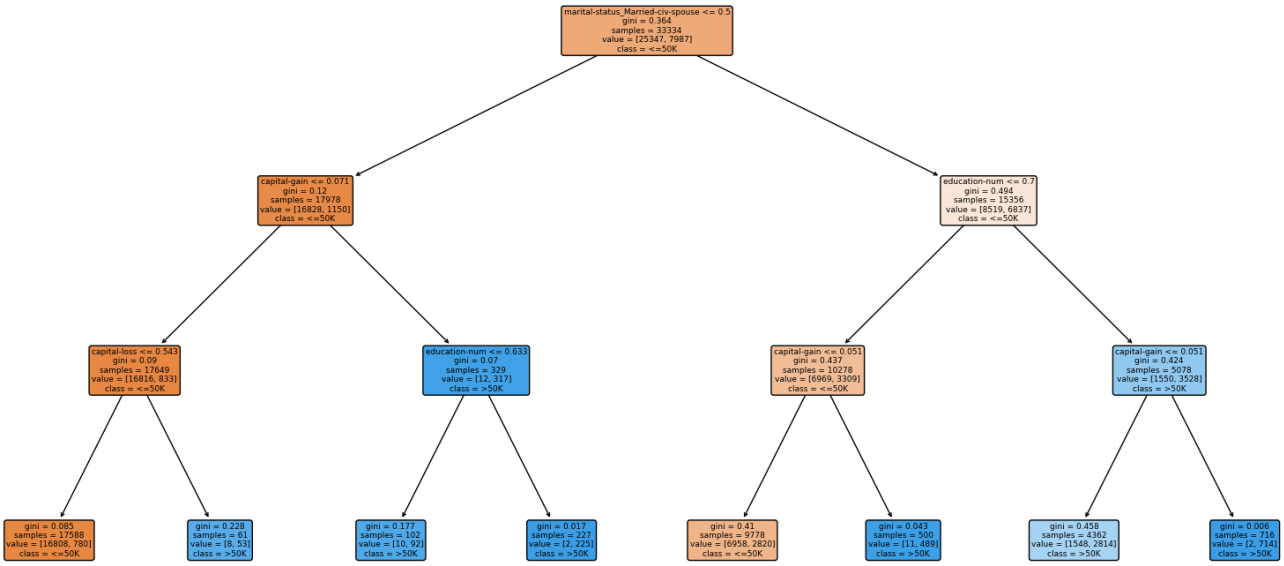



```
y_pred = model.predict(X_test)
tree_forward_score = accuracy_score(y_test, y_pred)
print('Accuracy:', tree_forward_score)
```

```
Accuracy: 0.8138167564919158
```

1. This decision tree model shows utilizes forward selection. It has an accuracy of 81.3%. The tree starts with the most significant feature, capital gain, to split the data, and continues to branch out based on other features like marital status and capital loss. Forward selection has helped in identifying the most predictive features from the dataset to create this decision tree. The root node splits the data based on whether the marital status variable is less than or equal to 0.5. The Gini score is a measure of how often a randomly chosen element would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. The lower the Gini score, the better the purity of the node with respect income classification. The samples value indicates the number of observations in the dataset that fall into each node, while the value array represents the count of samples for each class, with the first number typically representing the count for the class <= 50k and the second for > 50k.

```
# Assume 'df' is your DataFrame and it's already been loaded correctly.

X = df[chi_square_features]
y = df['incomes_>50K']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

# Initialize the DecisionTreeClassifier with specific constraints
model = DecisionTreeClassifier(max_depth=3, min_samples_split=100, min_samples_le
model = model.fit(X_train, y_train)

feature_names_list = X.columns.tolist()

# Now use plot_tree with the converted list of feature names
plt.figure(figsize=(20, 10))  # Set the size of the figure
plot_tree(model,
          feature_names=feature_names_list,
          class_names=['<=50K', '>50K'],
          filled=True,
          rounded=True)
plt.show()
```




```
y_pred = model.predict(X_test)
tree_chisq_score = accuracy_score(y_test, y_pred)
print('Accuracy:', tree_chisq_score)
```

```
Accuracy: 0.8385245327920487
```

2. We're showcasing a decision tree model that's been developed using the Chi-Square selection technique for feature selection. The model's accuracy stands at a robust 83.9%, indicating a high level of predictive power. At each node, the tree splits the dataset based on the attribute that provides the most significant Chi-square value, indicating the best distinction between classes. The splits, like 'capital-gain < 0.071' and 'marital-status_Married-civ-spouse <= 0.5', lead to the classification of individuals into those who earn above or below $50K.

Unsupervised Learning Models

```
# Forward Feature Selection
forward_selection_features
```

```
['capital-gain',
 'relationship_Not-in-family',
 'education_Doctorate',
 'education_Prof-school',
 'marital-status_Married-civ-spouse',
 'education_Masters',
 'capital-loss',
 'occupation_Transport-moving',
 'occupation_Farming-fishing',
 'education_Bachelors']
```

```
# Chi Squared Feature Selection
chi_square_features
```

```
['age',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'marital-status_Married-civ-spouse',
 'relationship_Husband',
 'marital-status_Never-married',
 'relationship_Own-child',
 'sex_Male',
 'sex_Female',
 'occupation_Exec-managerial',
 'relationship_Not-in-family',
 'occupation_Prof-specialty',
 'education_Bachelors']
```

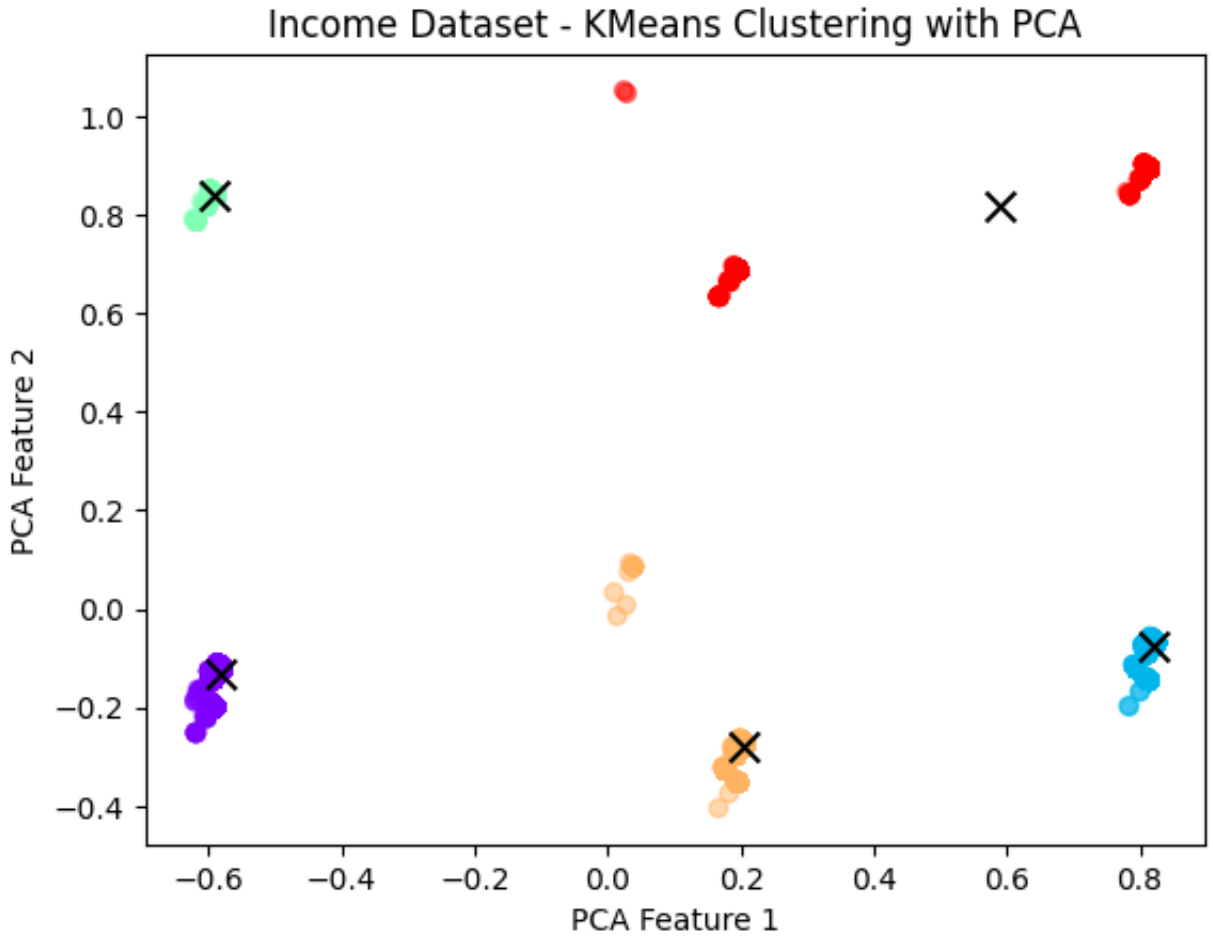
K-Means Clustering

```
# forward selection features
incomes = df["incomes_>50K"]
X = df[forward_selection_features]

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)

explained_variance = pca.explained_variance_ratio_
print(explained_variance)

# apply k-means clustering
# We will attempt to cluster into 5 groups,
# optimal k (k=5) found using inertia elbow method
kmeans = KMeans(n_clusters=5, random_state=42)

# train the model
kmeans.fit(X_pca)

# get the centroids and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
```

```
[0.45662505 0.18907969]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
```

```
silhouette_avg = silhouette_score(X_pca, labels)
print(f"Silhouette Score: {silhouette_avg}")

# Since we have more than two features, we will only plot the first two features
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='rainbow', alpha=0.5)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=100, c='black')
plt.title('Income Dataset - KMeans Clustering with PCA')
plt.xlabel('PCA Feature 1')
plt.ylabel('PCA Feature 2')
plt.show()
```

```
Silhouette Score: 0.9518725813716782
```

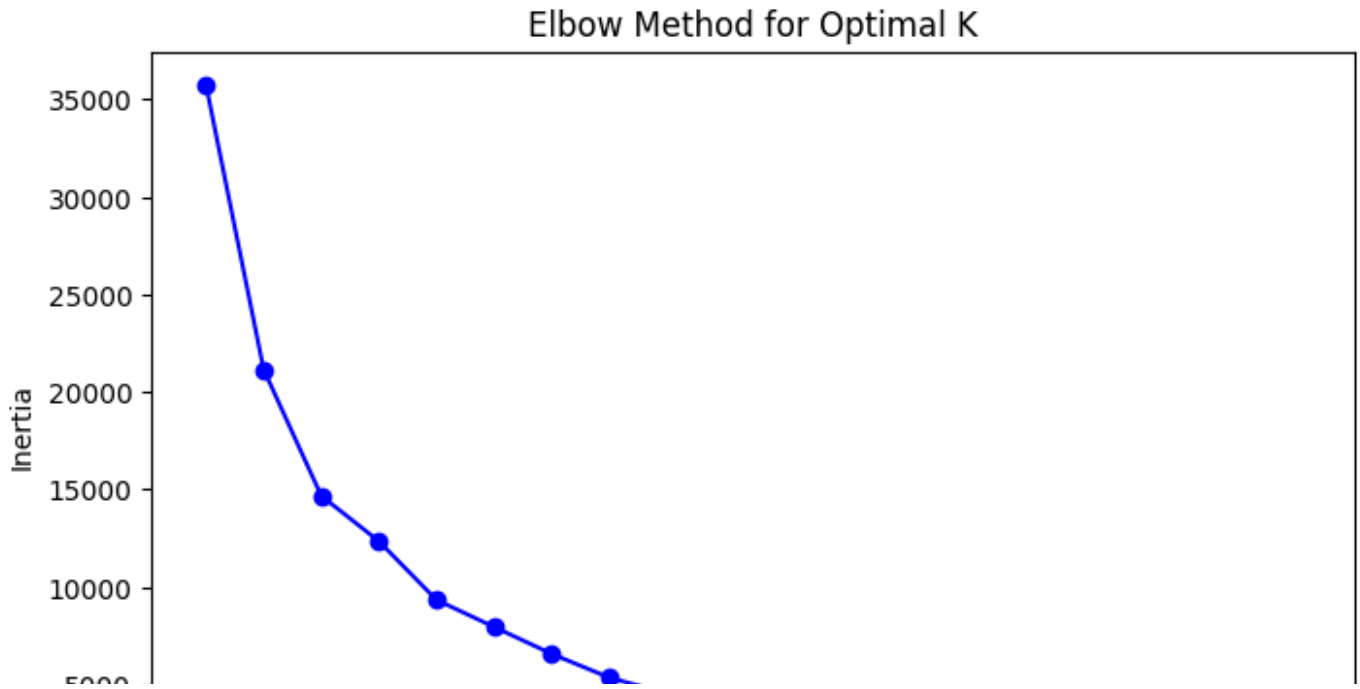



```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

K = 20
inertia = []

for n_clusters in range(1, K+1):
```

```
        kmeans = KMeans(n_clusters=n_clusters)
        kmeans.fit(X)
        inertia.append(kmeans.inertia_)

    # Plotting the elbow curve
    plt.figure(figsize=(8, 5))
    plt.plot(range(1, K+1), inertia, marker='o', linestyle='-', color='b')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal K')
    plt.xticks(range(1, K+1))
    plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
```



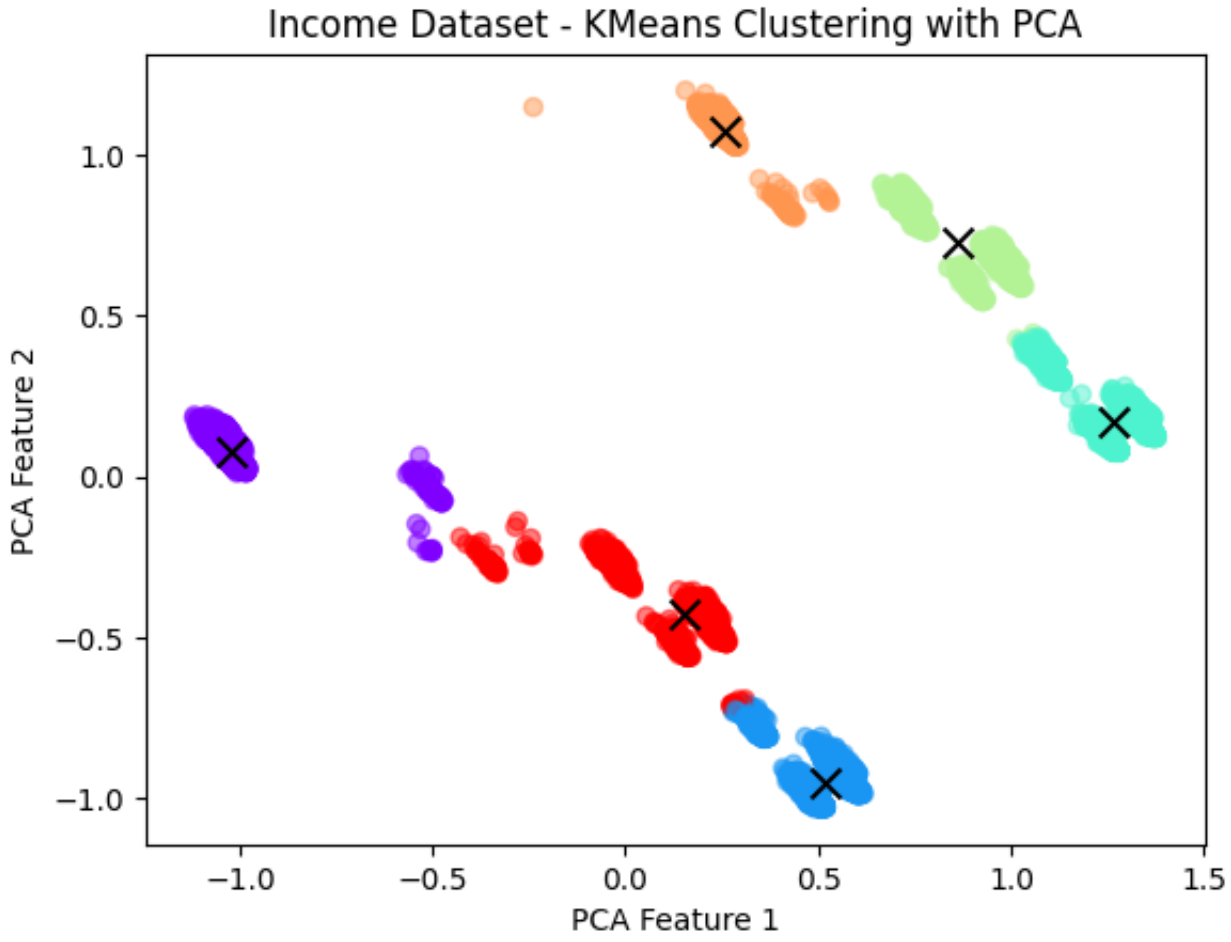
Above shows k-means clustering ran for the features pinpointed in forward feature selection. All of the variables used were numeric, courtesy of one-hot-encoding performed above. An optimal k value of 5 was determined using the elbow method in an inertia vs number of clusters graph. Inertia refers to the sum of squared distances between each data point. Lower inertia is preferred, meaning better defined and more cohesive clusters. With many dimensions/features, we decided to use a principal component analysis (PCA) value of 2 that simplifies the data down to two features, in order to maximize variance. The features chosen seem to account for 0.67 of total variance. From there, the k-means clustering algorithm was ran and a silhouette score of 0.952 was found. The silhouette score we found is quite close to 1, indicating that the separations between the clusters are clear and that everything is well clustered.

```
# chi-squared features
X1 = df[chi_square_features]
y = df["incomes_>50K"]

pca1 = PCA(n_components=2)
X_pca1 = pca1.fit_transform(X1)

explained_variance1 = pca1.explained_variance_ratio_
print(explained_variance1)

# apply k-means clustering
# split into 6 clusters, found optimal using inertia elbow method
kmeans1 = KMeans(n_clusters=6, random_state=42)

# train the model
kmeans1.fit(X_pca1)

# get the centroids and labels
centroids1 = kmeans1.cluster_centers_
labels1 = kmeans1.labels_
```

```
[0.42554784 0.16488606]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
```

```
silhouette_avg1 = silhouette_score(X_pca1, labels1)
print(f"Silhouette Score: {silhouette_avg1}")

# Since we have more than two features, we will only plot the first two features
plt.scatter(X_pca1[:, 0], X_pca1[:, 1], c=labels1, cmap='rainbow', alpha=0.5)
plt.scatter(centroids1[:, 0], centroids1[:, 1], marker='x', s=100, c='black')
plt.title('Income Dataset - KMeans Clustering with PCA')
plt.xlabel('PCA Feature 1')
plt.ylabel('PCA Feature 2')
plt.show()
```

```
Silhouette Score: 0.8506958757785694
```

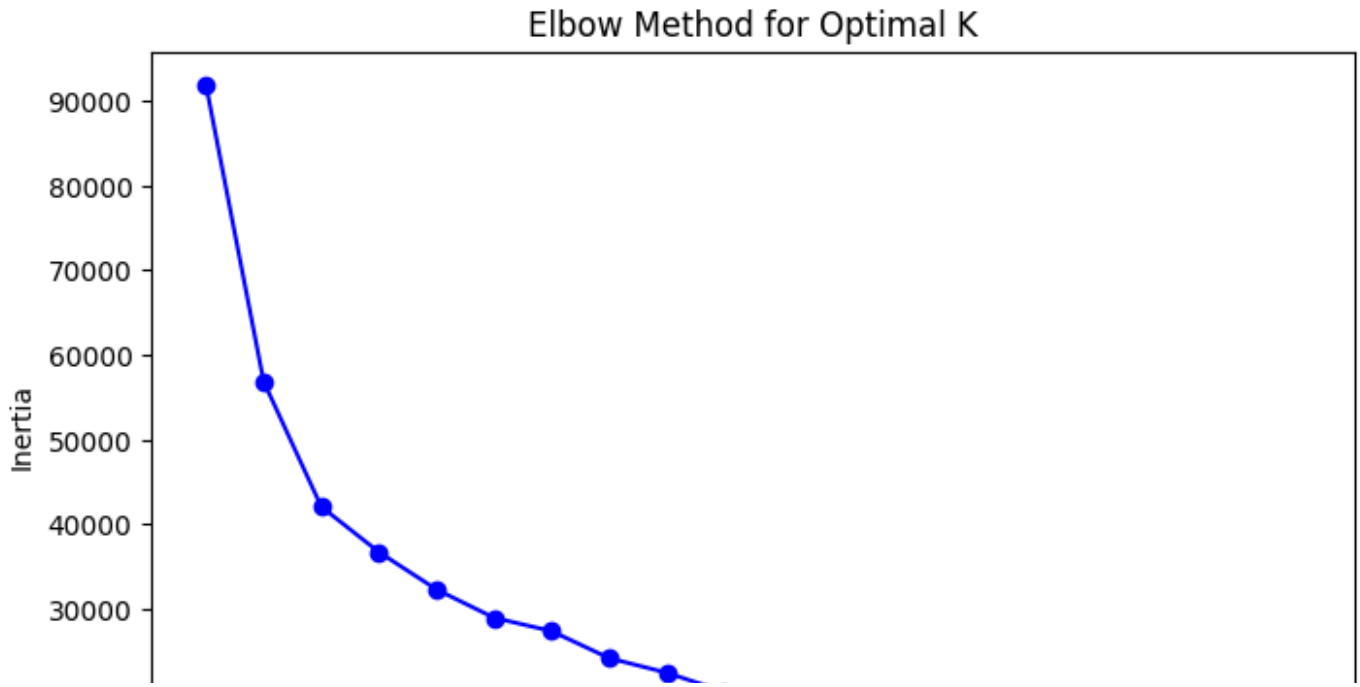



```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

K = 20  # Define your maximum number of clusters
inertia = []

for n_clusters in range(1, K+1):
```

```
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X1)
    inertia.append(kmeans.inertia_)

# Plotting the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(range(1, K+1), inertia, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.xticks(range(1, K+1))
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
```



Above shows k-means clustering ran for the features pinpointed in chi-squared feature selection. All of the variables used were numeric, courtesy of one-hot-encoding performed above. An optimal k value of 6 was determined using the elbow method in an inertia vs number of clusters graph. Inertia refers to the sum of squared distances between each data point. Lower inertia is preferred, meaning better defined and more cohesive clusters. With many dimensions/features, we decided to use a principal component analysis (PCA) value of 2 that simplifies the data down to two features. The features chosen seem to account for 0.60 of total variance. From there, the k-means clustering algorithm was ran and a silhouette score of 0.85 was found. The silhouette score we found is quite close to 1, indicating that the separations between the clusters are clear and that everything is well clustered. However, the k-means clustering algorithm appears to work better for the forward selection, as the peak silhouette score is higher.

```
from sklearn.cluster import KMeans

X = df[forward_selection_features]
y = df['incomes_>50K']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

kmeans = KMeans(n_clusters=2, random_state=0).fit(X_train)

y_pred = kmeans.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
              precision    recall  f1-score   support

           0       0.54      0.30      0.39     10733
           1       0.10      0.24      0.14      3554

    accuracy                           0.29     14287
   macro avg       0.32      0.27      0.27     14287
weighted avg       0.43      0.29      0.33     14287
```

```
from sklearn.cluster import KMeans

X = df[chi_square_features]
y = df['incomes_>50K']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

kmeans = KMeans(n_clusters=2, random_state=0).fit(X_train)

y_pred = kmeans.predict(X_test)
print(classification_report(y_test, y_pred))
```

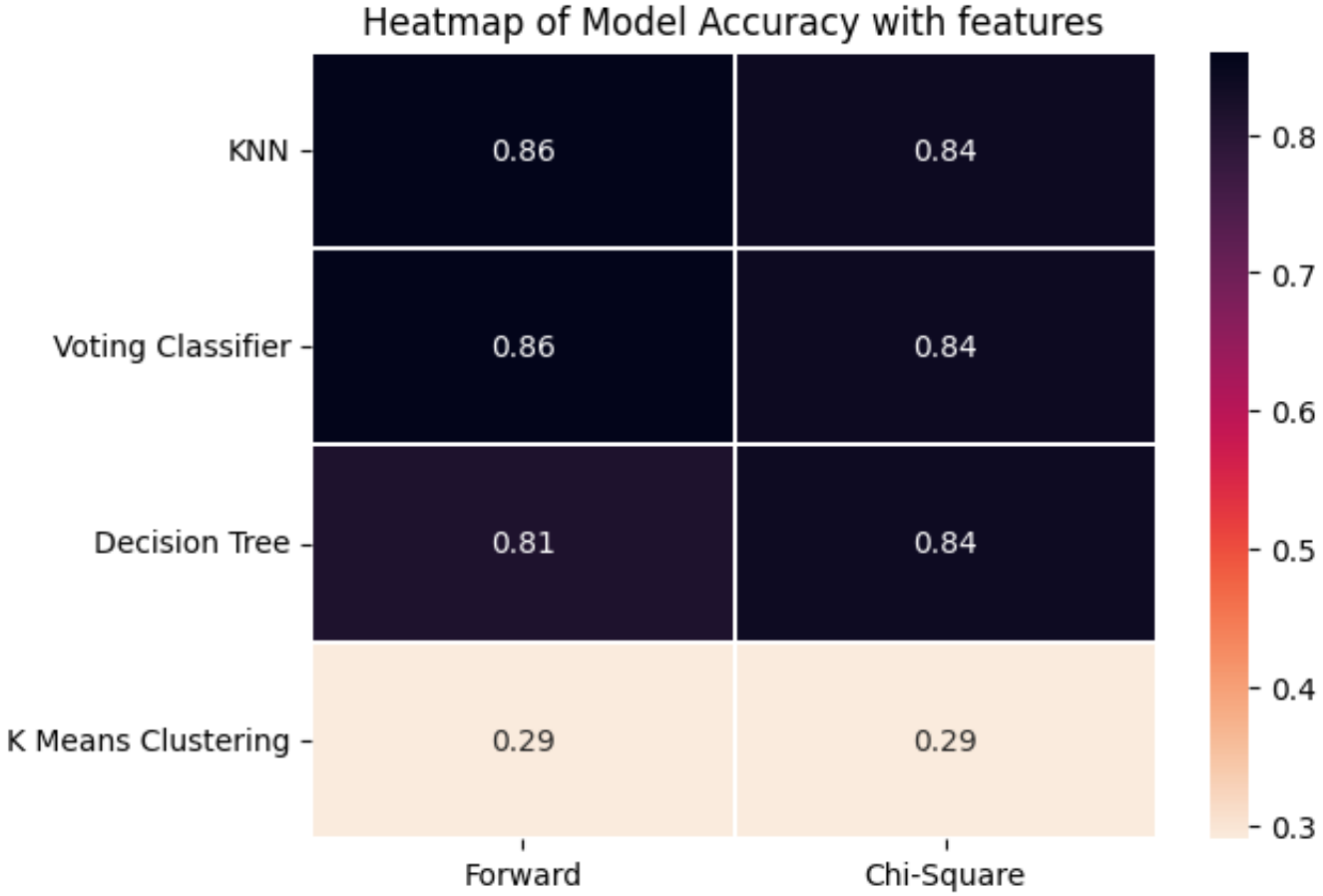
```
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
      warnings.warn(
                  precision    recall  f1-score   support

               0       0.54      0.33      0.41     10733
               1       0.07      0.15      0.10      3554

        accuracy                           0.29     14287
       macro avg       0.31      0.24      0.25     14287
    weighted avg       0.43      0.29      0.33     14287
```

Now that the clustering algorithm has been shown to work well with silhouette scores near 1, we need to interpret the clustering within the context of binary classification of income. To do this, we strayed away from the ideal k values of 5 for feature selection and 6 for chi squared and used those same predictors but for k = 2 for both. K = 2 was chosen because we want to predict a binary target variable-whether income is above or below 50k. The accuracy of both models was found to be .29, which indicates that nonsupervised learning methods like k-means clustering are not ideal for binary classification tasks.

Project Report of Results

Results & how it supports or not support hypotheses

```
feature_selection = ["Forward", "Chi-Square"]
model_names = ["KNN", "Voting Classifier", "Decision Tree", "K Means Clustering"]
#vector order: KNN, Voting Classifier, Decision Tree, k means clustering
forward_accuracies = [0.86, 0.86,tree_forward_score, 0.29]
chisq_accuracies = [0.84, 0.84, tree_chisq_score, 0.29]
accuracies = [forward_accuracies, chisq_accuracies]

accuracy_df = pd.DataFrame(accuracies, columns= model_names, index=feature_select
accuracy_df = accuracy_df.transpose()
sns.heatmap(accuracy_df, annot = True, cmap = "rocket_r", linewidths = 0.25)
plt.title("Heatmap of Model Accuracy with features")
plt.show()
#Note: K means clustering silhouette score can range from -1 to 1.
#However, negative values indicates incorrect clustering, while 0 means
#that cluster spacing doesn't matter.
```



Explanation

As seen in the heatmap above, KNN and Voting Classifiers have the overall best accuracies (86% and 84% for forward and chi square features respectively) compared to other models. While the Decision Tree yielded somewhat accurate results (81% for forward features and 84% for Chi-Square), it is outranked by KNN and Voting Classifier for forward features. K-Means clustering is overall ineffective (both chi-square and forward features yielded silhouette scores of 0.29), likely due to the large number of features in it. As a result, we would probably use KNN or Voting Classifiers to do classification on this dataset.

Conclusion

After training and modelling the data, we found that education levels and capital gain/loss were among the strongest predictors of whether someone would earn more than 50K yearly or not. While most of the models were similar according to our testing and training, we found that Voting Classifiers and KNN (they were tied in first place) were the overall most effective ways of classification in this dataset. However, we found that most of the sample population were White, Male, and from the USA, which questions the generalizability of this dataset. Even taking into consideration these biases, however, we were still able to determine that income was heavily correlated with education levels (capital gain/loss is likely already obvious).