Northeastern

College of Professional Studies
Northeastern University San Jose

MPS Analytics

Course: ALY6020

Assignment:

Module 1 –Project

Submitted on:

September 29, 2023

Submitted to:

Professor: Ahmadi Behzad

Submitted by:

Heejae Roh

Introduction

In this census data, I will conduct KNN (k nearest neighbor) analysis to predict categorical variables classified by salary. Rather than applying KNN right away, I will analyze the data from various angles and look at which columns to include in KNN and what would be the pre-processing for each column. In the process, I will apply each method for judging numerical variables and categorical variables. This will be done by applying the data cleansing learned in this module. Data usually is needed to be lot of cleansing processes, and the data engineering process or cleaning process takes up plenty amount of time during data analysis. I would like to apply data cleansing details in this report.

Dataset Understanding & Data Splitting

The dataset is US census data containing biographical profile (bio) and income. There are 15 columns and 48,841 rows.

Description of the variables/features in the dataset.

#	Column	Dictionary
0	age	people's age. continuous.
1	workclass	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
2	fnlwgt	ignore, continuous.
3	education	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voe, 9th, lth-Bth, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
4	education_num	Years of education. continuous.
5	marital_status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
6	occupation	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
7	relationship	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
8	race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
9	sex	Female, Male.
10	capital_gain	Income earned through capital. continuous.
11	capital_loss	Losses through capital. continuous.
12	hours_per_week	Work hours per week. continuous.
13	native_country	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVJ-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France
14	salary	Categorical variable. <=50K or >50K

Exploratory Data Analysis

Headtail of Dataset 1

	age	workclass	fnlwgt	education	edu_num	martial_status
0	39	State-gov	77516	Bachelors	13	Never-married
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse
2	38	Private	215646	HS-grad	9	Divorced
…				…		…
48839	38	Private	374983	Bachelors	13	Married-civ-spouse
48840	44	Private	83891	Bachelors	13	Divorced
48841	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse

Headtail of Dataset 2

	occupation	relationship	race	sex
0	Adm-clerical	Not-in-family	white	male
1	Exec-managerial	Husband	white	male
2	Handlers-cleaners	Not-in-family	white	male
…		…		…
48839	Prof-specialty	Husband	white	male
48840	Adm-clerical	Own-child	Asian-Pac_Islander	male
48841	Exec-managerial	Husband	White	male

Headtail of Dataset 3

	capital_gain	capital_loss	hours_per_week	native_country	salary
0	2174	0	40	United-States	<=50K
1	0	0	13	United-States	<=50K
2	0	0	40	United-States	<=50K
…		…		…	
48839	0	0	50	United-States	<=50K
48840	5455	0	40	United-States	<=50K
48841	0	0	60	United-States	>50K

The dataset is US census data containing biographical profile (bio) and income. There are 15 columns and 48,841 rows. This data consists of 5 numeric variables and 10 objects, and the objects are in categorical form. Since I will be investigating income, the most important variable is salary. It is most important to understand how other variables affect salary. Among them, fnlwgt is said to be ignored as meaningless data in the description of dataset. Let's look at all of this data except fnlwgt.

Descriptive Analysis of Dataset

	age	edu_num	capital_gain	capital_loss	hours_per_week
count	48842	48842	48842	48842	48842
mean	38.64	10.08	1079.08	87.50	40.42
std	13.71	2.57	7452.02	403.00	12.39
min	17.00	1.00	0.00	0.00	1.00
25%	28.00	9.00	0.00	0.00	40.00
50%	37.00	10.00	0.00	0.00	40.00
75%	48.00	12.00	0.00	0.00	45.00
max	90.00	16.00	99999.00	4356.00	99.00

1. The data was first checked to see if it contained columns that could be removed. I should check how skewed capital_gain and capital_loss are through histogram.
2. Other numeric variables are evenly distributed, so let's do visualization again and look at the relationship with salary.

information of Dataset

#	Column	Non-Null Count	Dtype
0	age	48842	float64
1	workclass	48842	object
2	fnlwgt	48842	int64
3	education	48842	object
4	education_num	48842	int64
5	martial_status	48842	object
6	occupation	48842	object
7	relationship	48842	object
8	race	48842	object
9	sex	48842	object
10	capital_gain	48842	int64
11	capital_loss	48842	int64
12	hours_per_week	48842	int64
13	native_country	48842	object
14	salary	48842	object

Data Cleansing

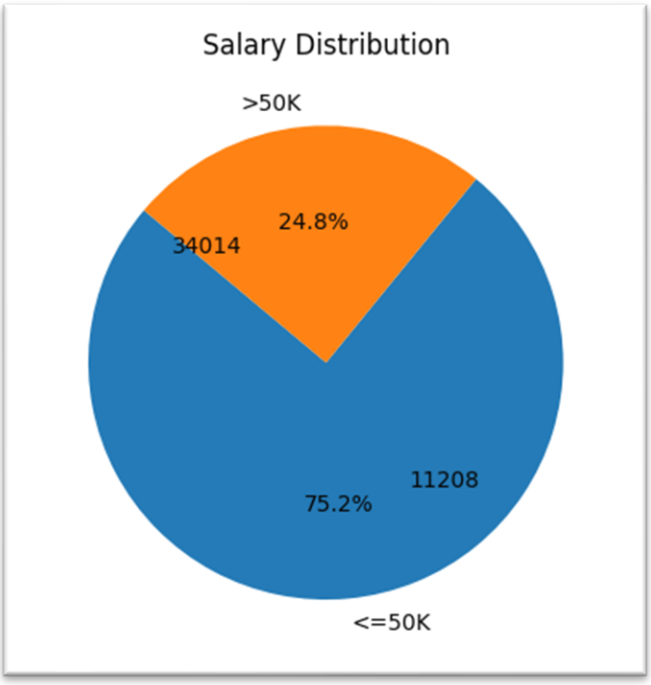
1. Drop columns ‘fnlwgt’ (Meaningless), ‘capital_loss’ (in descriptive analysis, too skewed and after checking correlation matrix with ‘salary’), and ‘workclass’ (chi-square p-value > 0.05).
2. Delete rows containing ‘?’ in ‘workclass’, ‘occupation’, and ‘native_country’. In this module, I learned that the last option to consider when cleansing data is remove, but for rows containing '?' data, it was difficult to determine how reliable the information in other columns was. The most important point is that there were 3,619 pieces of data containing '?', accounting for only 7.4%, less than 10% of the total.
3. Bucketing categorical variables (Below table). I think that too much categorical data in each column can create noise in the analysis, so I organize the data that is less than 0.05%, or in the case of education, I organize the data with data such as education level lower than high school, college and associate education, and Bachelor's or higher. The meaning was analyzed again and grouped together. I re-understood the meaning of cases such as marital_status and occupation and grouped them into categories located in relatively similar places.
4. Making dummy variables with categorical columns and drop original columns. I used the drop_first=True option when extracting the dummy variable. Through this, we wanted to reduce computing energy. Drop_first=True is important to use, as it helps in reducing the extra column created during dummy variable creation. Hence it reduces the correlations created among dummy variables (Soumya, 2020).
5. After cleansing the dataset, 26 columns and 45,222 rows

Bucketing Categorical Variables

Categorical_ Column	Before bucketing	After bucketing
workclass	8 unique values	Private 0.736725 others 0.111116 Self-emp-not-inc 0.083540 Local-gov 0.068618
education	16 unique values	HS_below 0.452081 Col_nAssoc 0.295542 Bach_above 0.252377
martial_status	7 unique values	Married 0.478506 Never-married 0.322807 Divor_Sep_Wid 0.198686
occupation	14 unique values	Technical_manual_labor 0.386405 Professional_managerial 0.265181 Sales_service 0.225908 Clerical_administrative 0.122507
relationship	5 unique values	5 unique values
race	5 unique values	White 0.860267 Black 0.093494 others 0.046239
sex	2 unique values	2 unique values
native_country	40+ unique values	United_States 0.913095 others 0.066936 Mexico 0.019968

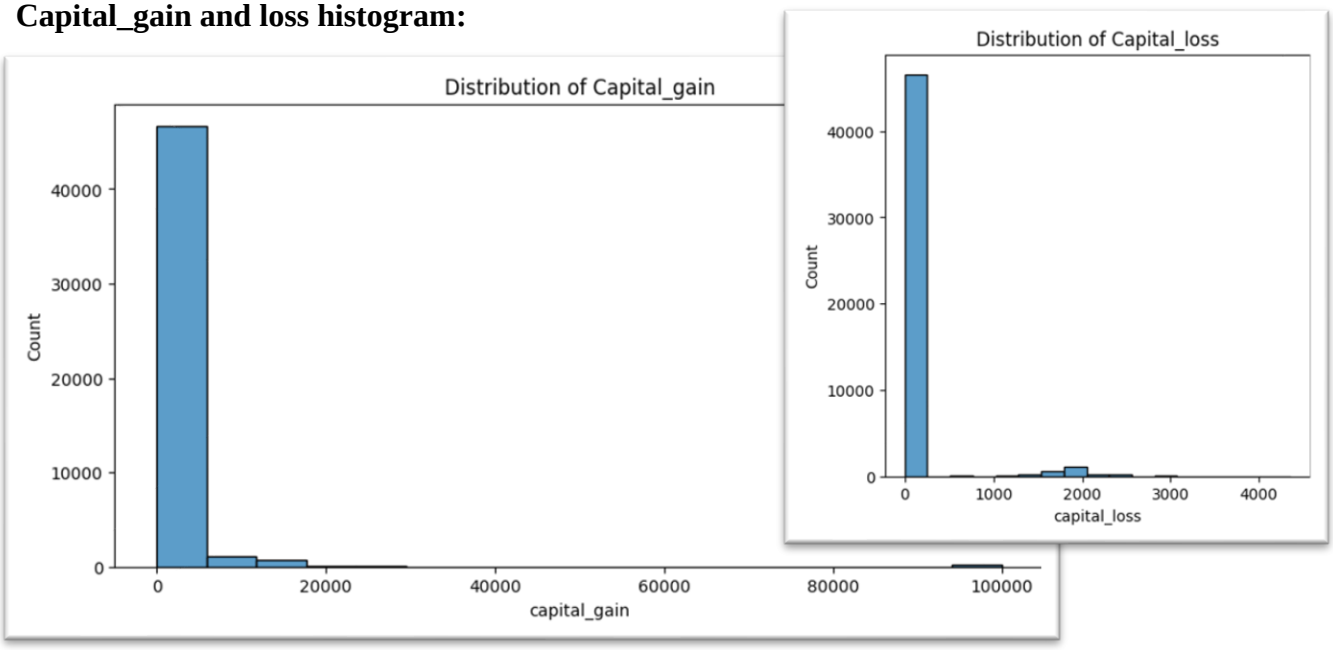
Data Visualizations

Salary Distribution:



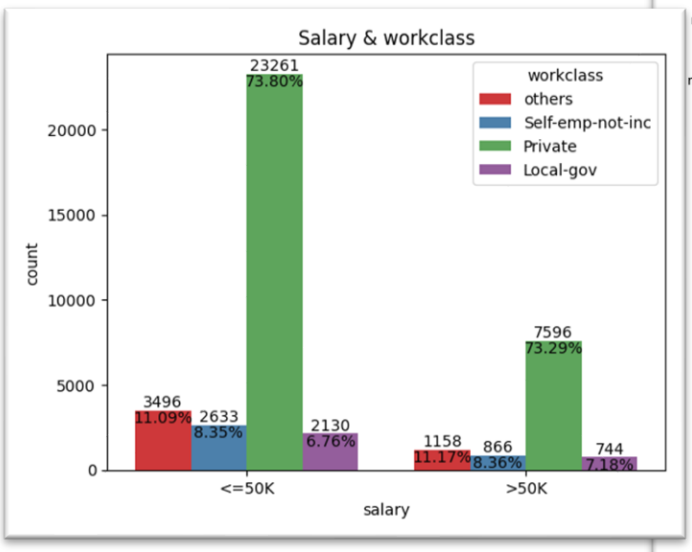
After cleansing (especially, deleting every '?' rows), salary over 50K is 24.8% and 34,014 counts, which is around one-fourth of whole data. Salary less or equal 50K is 75.2% and 11,208 counts, which is around three-fourth of whole data. We can say that this is imbalanced dataset.

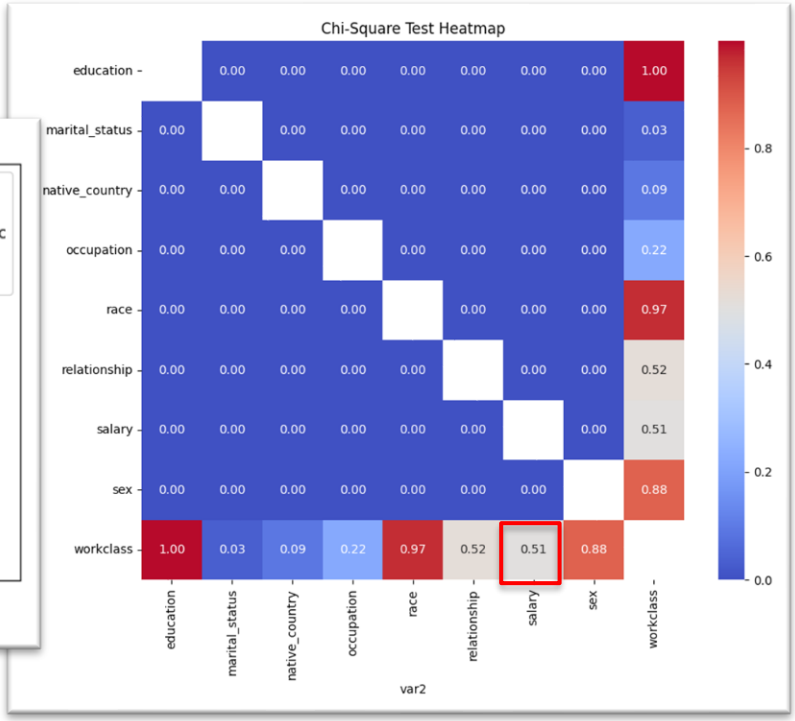
Capital_gain and loss histogram:



If you look at Capital_gain and Capital_loss, you can see that many values are concentrated around 0. Capital_gain has 91.7% of data gathered at 0, and capital_loss has an even larger 95.3% data at 0. Therefore, we checked again in the future to see if there were any columns to be removed from the KNN analysis among this data, and decided to remove Capital_loss from the correlation matrix.

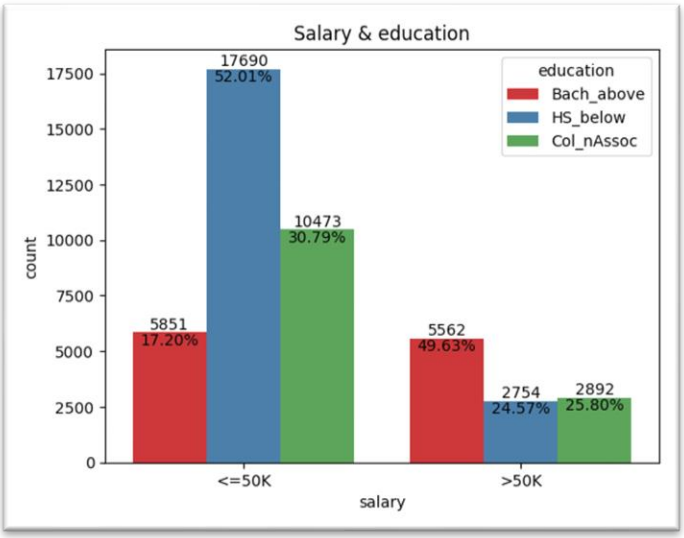
Salary & workclass:





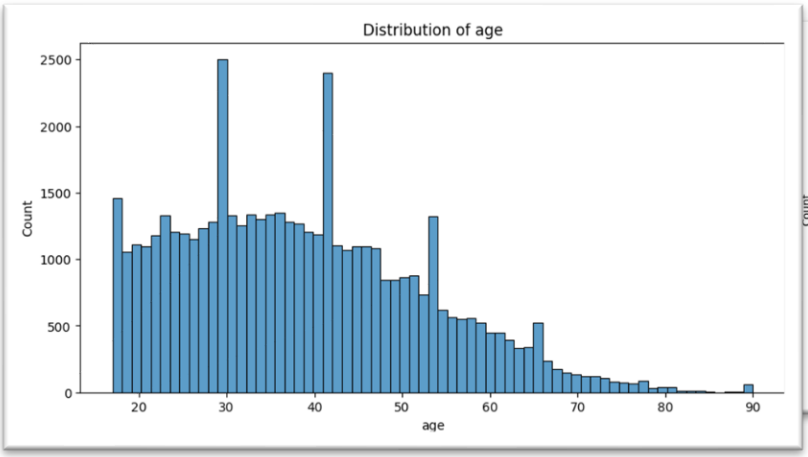
Looking at the results of bucketing the workclass, I left most of the workclass categories as is and included only those with a low percentage as others. Interestingly, when dividing salary <=50K and >50K, the ratio of workclasses was almost the same. In other words, it is difficult to divide salary groups according to workclass. This can be confirmed later in the chi-square results.

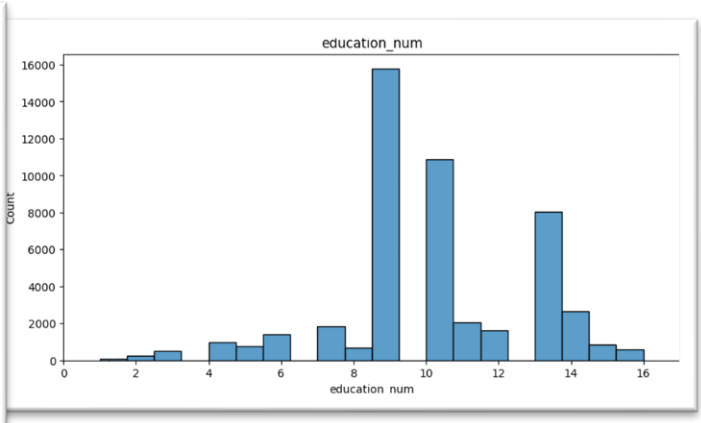
Salary & Education (Salary & others in Appendix):



In the case of education, approximately 50% of students had a Bachelor's degree or higher and were in the >50K group. The proportion of high school below <=50K was more than half of the total. It can be assumed that education played a large role in dividing salary groups.

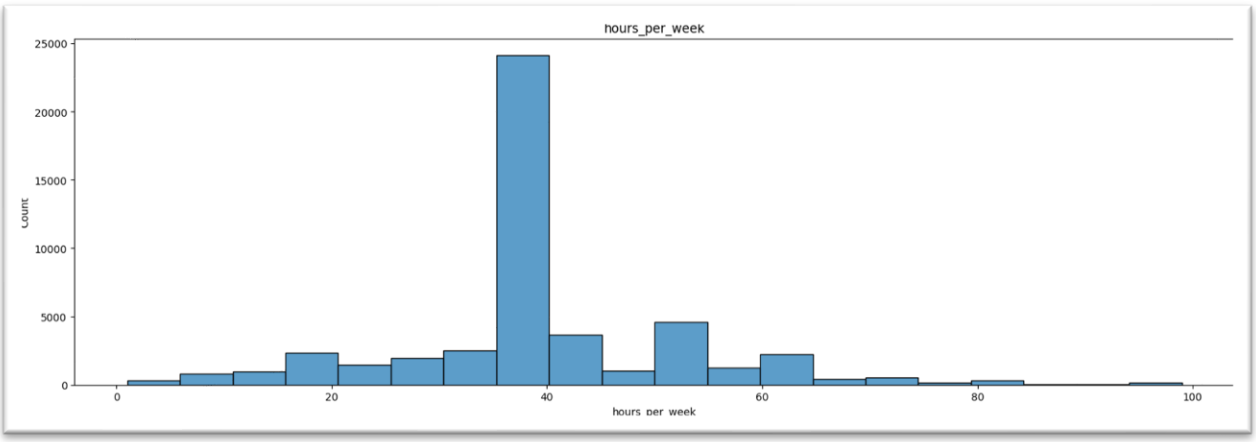
Age & education_num histogram:





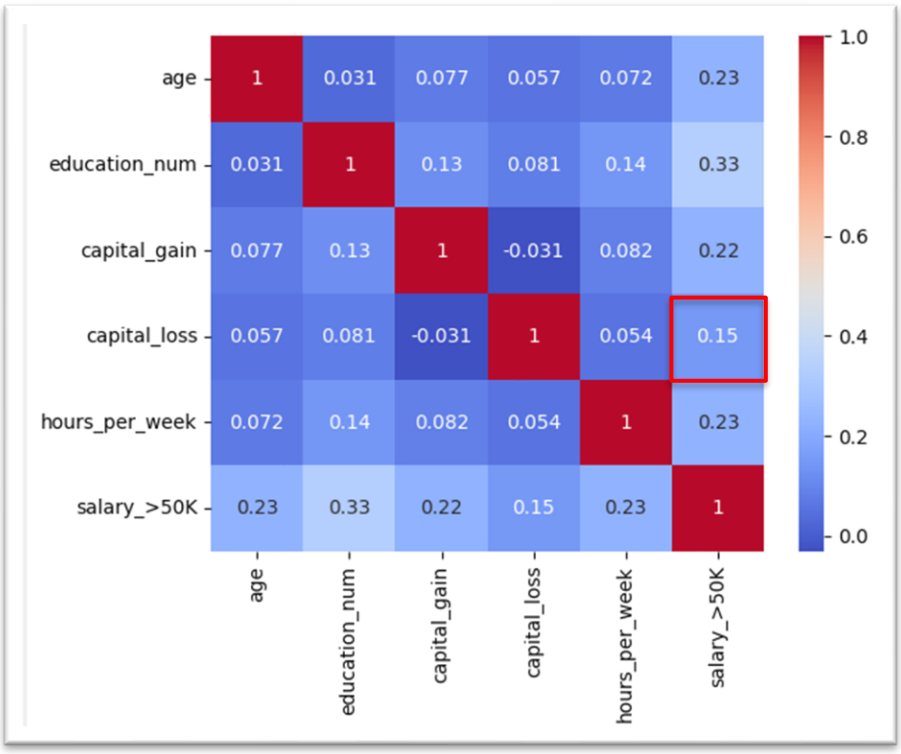
In the case of education, there are clustered around 9, which is close to the median, and relatively more people with a long period of education are included in the data.

hours_per_week histogram:



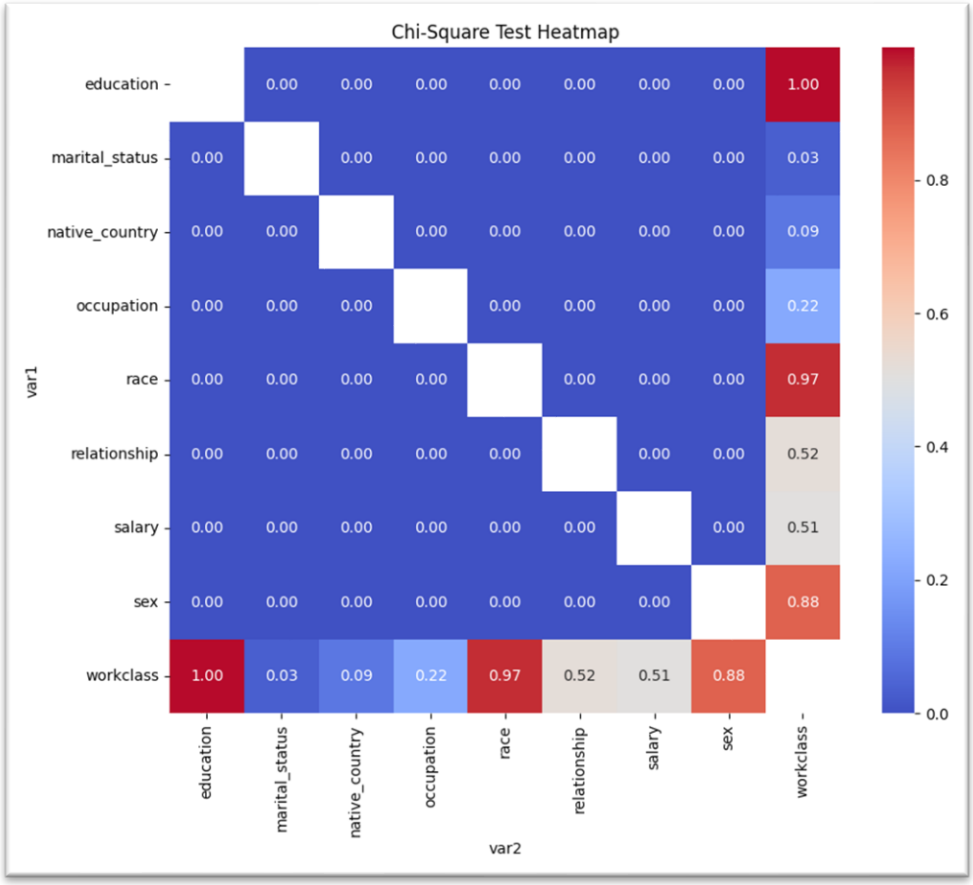
'Hours_per_week' was concentrated around 40 hours, and data was concentrated in mode. Except for the mode data, it was evenly distributed overall.

Heatmap of correlation matrix:



This is the correlation matrix between the numeric variable and salary dummy. ‘capital_loss’ was decided to be removed based on this data, the fact that the percentage of 0 exceeds 95%, and the final results of KNN including and without capital_loss. There was no significant difference between the results without and including capital_loss.

Chi-squared test with categorical variables



1. The following is a heatmap from the chi-square test. If you only look at the salary section, you can see that most of the data is lower than the p-value of 0.05.
2. However, as confirmed in the visualization earlier, there was almost no difference in categorical variables between the two workclass and salary groups, which was reflected in this chi-square heatmap, showing a p-value of 0.51.
3. Therefore, I decided to also remove the workclass before KNN analysis.

Result of KNN & Interpretation

Split data to train and test, and why?

You should test your model to measure the performance of your model. For this task, you have to separate a set of data from the same data distribution and keep it separately. You cannot touch this dataset or do any parameter tuning with this dataset in the model training process (Malintha, 2018). KNN basically does not need to divide data to perform analysis, but we can examine the process of dividing data and comparing test data and predicted data to measure performance.

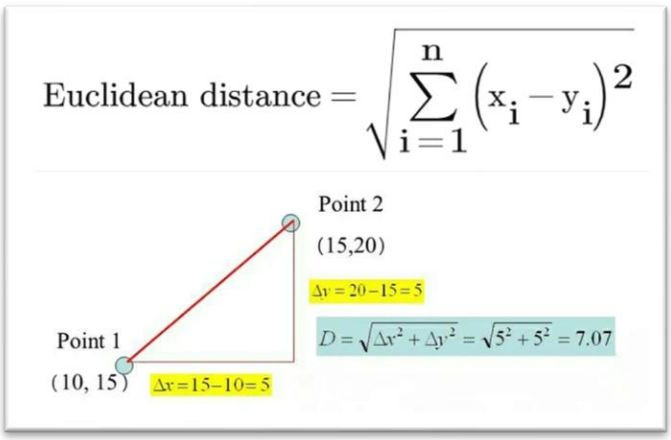
In the table below, I will compare performance by number k. I will include the workclass table that I removed earlier via chi-square in the process, and see for myself how not including it affects performance.

Model Performance by k

k		With all	Drop 'capital_loss'	Drop 'workclass'
3	Accuracy	0.773	0.773 (-)	0.773 (-)
	Precision	0.786	0.786 (-)	0.786 (-)
	Recall	0.959	0.959 (-)	0.959 (-)
	F1 Score	0.864	0.864 (-)	0.864 (-)
5	Accuracy	0.770	0.770 (-)	0.770 (-)
	Precision	0.786	0.786 (-)	0.786 (-)
	Recall	0.954	0.954 (-)	0.954 (-)
	F1 Score	0.862	0.862 (-)	0.862 (-)
7	Accuracy	0.770	0.770 (-)	0.771 (-)
	Precision	0.786	0.786 (-)	0.786 (-)
	Recall	0.954	0.954 (-)	0.955 (-)
	F1 Score	0.862	0.862 (-)	0.862 (-)
9	Accuracy	0.766	0.762 (-0.04)	0.763 (+0.01)
	Precision	0.785	0.785 (-)	0.785 (-)
	Recall	0.949	0.943 (+0.05)	0.944 (+0.01)
	F1 Score	0.859	0.856 (-0.04)	0.857 (+0.01)
11	Accuracy	0.759	0.755 (-0.04)	0.756 (+0.01)
	Precision	0.784	0.784 (-)	0.784 (-)
	Recall	0.939	0.931 (-0.08)	0.932 (+0.01)
	F1 Score	0.854	0.851 (-0.03)	0.852 (+0.01)
20	Accuracy	0.775	0.773 (-0.02)	0.773 (-)
	Precision	0.803	0.786 (-0.17)	0.786 (-)
	Recall	0.930	0.959 (+0.29)	0.959 (-)
	F1 Score	0.862	0.864 (+0.02)	0.864 (-)

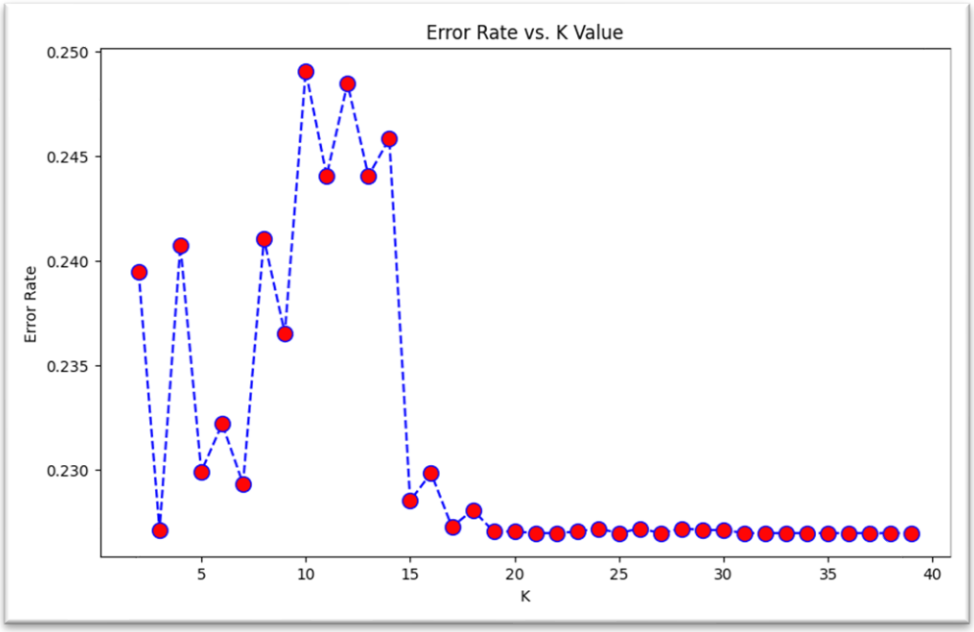
To think about how to find the value of k, I ran knn with six different k values and compared the performance. The one on the left is the performance when KNN was run using all data except fnlwgt. The middle is performance when capital_loss is excluded. The last column is when workclasses that did not show significant differences in chi-square were excluded.

How to choose k



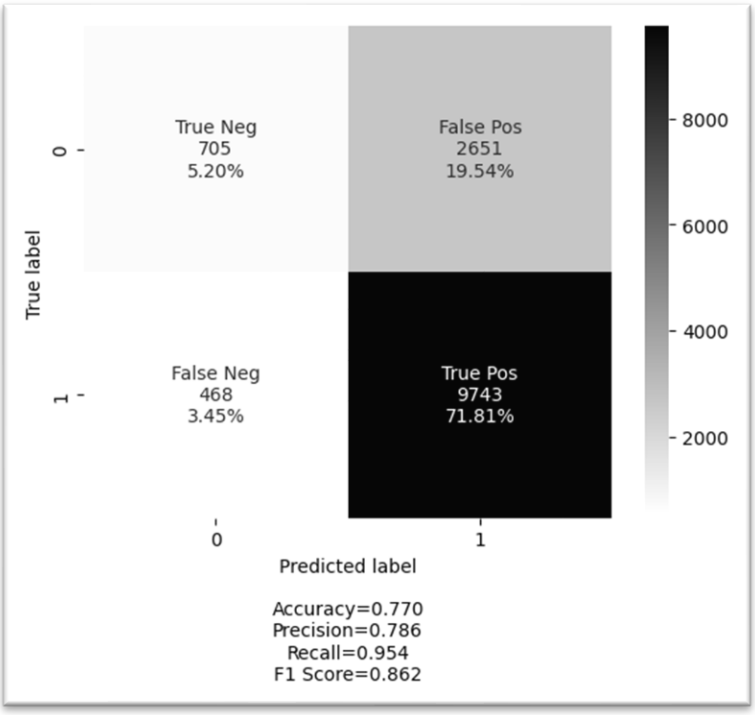
The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. Overall, it is recommended to have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset (IBM, 2023). There is no method to determine k. Therefore, we must choose which k based on performance and understanding of the data. We can choose k in KNN algorithm while comparing the distance and accuracy with k by k. However, as mentioned earlier, this requires understanding the data first. But the principle is to use odd numbers. If we use even

numbers, we end up with a tie vote and cannot predict properly.



The way KNN works is by taking votes from adjacent points. If your data is well separated then as you increase K then the model won't be as confused. In some cases, the performance would decrease; this is when K is so big that it's more than the number of samples inside a class. And with algorithms like KNN, it's straight-forward to understand the reason why it made the prediction with simple plotting. Also check you test data is big enough and there's no data leakage between the training and test (Mohamed, 2022). According to this, it can be said that the skewed data we saw in descriptive analysis and data visualization showed this phenomenon.

Final Result with k = 19



Among the various k, I ultimately decided to use k, which is 3. This means that I vote by looking at the three points closest to the salary I want to estimate. As mentioned earlier, to avoid ties, odd numbers, such as 20, are generally not used. I believe that using more k should result in higher performance, but it is difficult to clearly explain why this result was obtained. However, I think that since we have 21

variables including dummy, if we do a KNN that reflects all of them, we could get good results without setting the number of k large.

Interpretation

1. Accruacy was recorded at 77.0%, precision was at 78.6%, Recall was at 95.4%, and F1 Score was at 86.2%. This means that the accuracy of classifying salary categories is 77.0%. I set '<=50K' to 1 and >50K to 0. However, as we saw in the visualization, this is imbalanced data.
2. You can see that the analysis for '<=50K' is much more accurate because there are many more values. There are a total of 1,173 (705+468) predicted as 0 (>50K) in the confusion matrix. I can see that 695 of these are actually (>50K), and 60.1% are predicted correctly.
3. I set <=50K to 1 because I thought distinguishing <=50K was more important for this analysis. In the case of Type 1 error, there are people who were expected to be '>50K' but are <=50K'. I thought reducing this number would make it easier to create policies to close the wage gap.

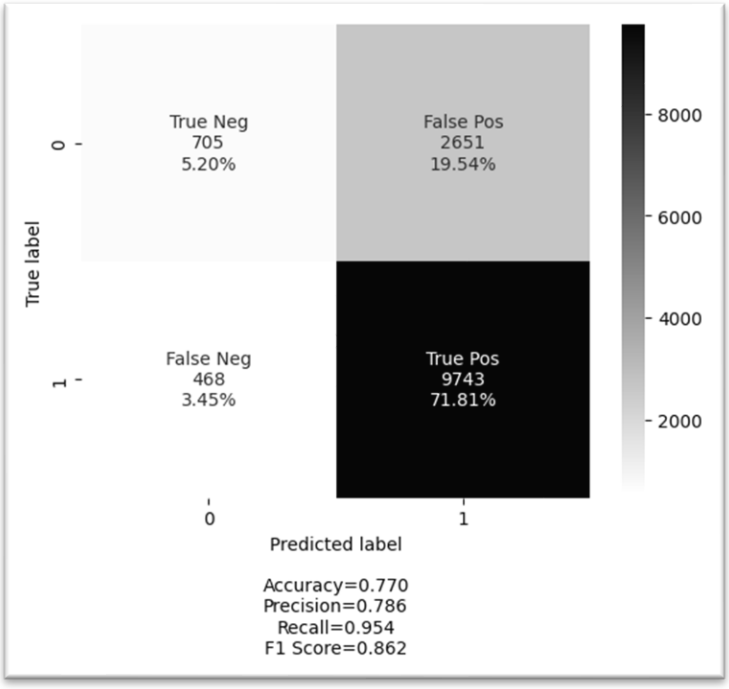
Answering Questions

1. Data Cleansing process & Explanation?

1. Drop column ‘fnlwgt’ (meaningless, mentioned as ‘ignore’ in data description).
2. Finally, I excluded 'capital_loss', one of the numeric variables, and 'workclass', one of the categorical variables, from the KNN analysis.
3. Capital_loss was too skewed data as it contained more than 95% of the '0' value. For workclass, the chi-square value was higher than 0.05 in determining salary. Therefore there is no evidence that there are correlation between salary and workclass.
4. I learned that the last option to consider when cleansing data is remove, but for rows containing '?' data, it was difficult to determine how reliable the information in other columns was. The most important point is that there were 3,619 pieces of data containing '?', accounting for only 7.4%, less than 10% of the total.
5. Bucketing categorical variables (Below table). I think that too much categorical data in each column can create noise in the analysis, so I organize the data that is less than 0.05%, or in the case of education, I organize the data with data such as education level lower than high school, college and associate education, and Bachelor's or higher. The meaning was analyzed again and grouped together. I re-understood the meaning of cases such as marital_status and occupation and grouped them into categories located in relatively similar places.
6. Making dummy variables with categorical columns and drop original columns. After cleansing the dataset, 26 columns and 45,222 rows
7. Before executing KNN, using StandardScaler, to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1.

2. A nearest neighbors model with the given data, interpret the results

Final Result with k = 19



1. Accruacy was recorded at 77.0%, precision was at 78.6%, Recall was at 95.4%, and F1 Score was at 86.2%. This means that the accuracy of classifying salary categories is 77.0%. I set '<=50K' to 1 and >50K to 0. However, as we saw in the visualization, this is imbalanced data.
2. You can see that the analysis for '<=50K' is much more accurate because there are many more values. There are a total of 1,173 (705+468) predicted as 0 (>50K) in the confusion matrix. I can see that 695 of these are actually (>50K), and 60.1% are predicted correctly.
3. I set <=50K to 1 because I thought distinguishing <=50K was more important for this analysis. In the case of Type 1 error, there are people who were expected to be '>50K' but are <=50K'. I thought reducing this number would make it easier to create policies to close the wage gap.
4. I find that the value of K=3 shows similar or better accuracy, but since it is limited to local information, I choose k as 19.

3. Convey those results to stakeholders?

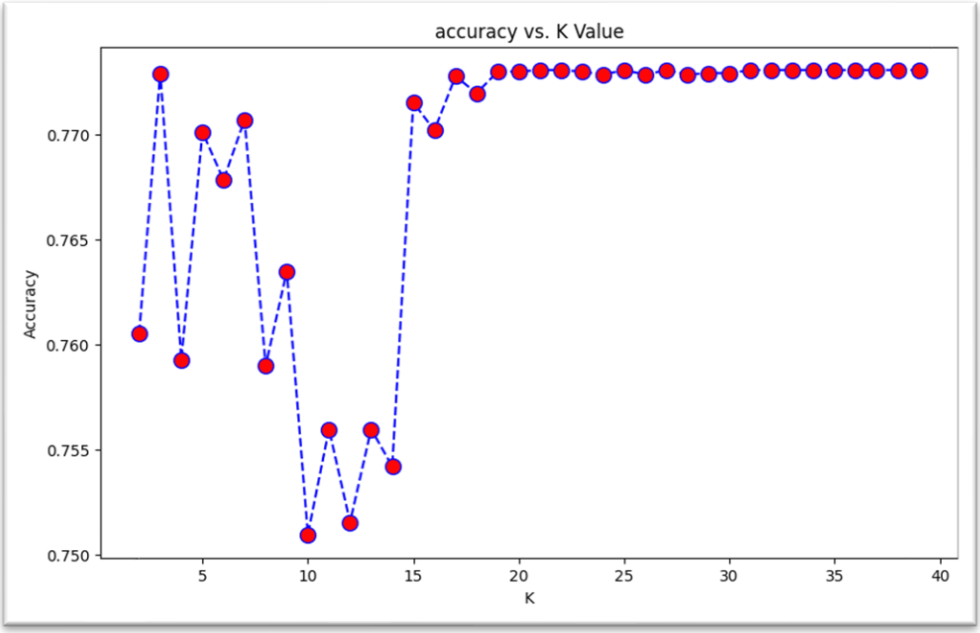
1. Through KNN analysis, we can see that this model can estimate salary groups using our other factors with an accuracy of 77.3%.
2. In terms of reducing the income gap, this means that after confirming that salary groups are divided based on other factors and checking which factors are included, insights into why the income gap occurs can be discovered through this data.
3. If the government has this data, it can be used to predict income gaps, especially the '<=50K' group, and help implement policies for them.
4. However, there appears to be a limit to predicting '>50K' due to imbalanced data. Therefore, in the future, additional data for '>50K' can be collected or methods of randomly generating additional data can be used for analysis.

4. Highlight key learning points such as feature importance of variables, how those variables explain the scenario?

1. I first divided numerical variables and categorical variables. Correlation was confirmed in numerical variables, and correlation with salary category was confirmed in categorical variables through chi-square test.

2. In the numerical variables I included in the KNN analysis, most variables showed a correlation of 0.2 or more.
3. The categorical variables I included in the KNN analysis also all recorded p-values of 0.05 or less in the chi-squared test.
4. It can be said that the numerical values are within a similar range, which means that all columns can be assumed to have values that are meaningful in determining the nearest point of salary.

5. How you determined K, why you choose that final value of K, and the overall accuracy of your model and accompanying models



1. K shows the best performance at 3, and because the data is not evenly distributed, data leakage occurred, or the test dataset is insufficient, you can see that k increases as shown above, but accuracy decreases.
2. Then, when k becomes 19, the accuracy is restored again, and after that, the accuracy is relatively steady.
3. I decided that k=19 was the turning point and that k was a value that could predict data without being limited to the region, so I chose 19 rather than 3.
4. This means that the data we have often contains values from many columns, and especially in the case of categorical variables, the data is often more similar. Since the target variables were also binary, it is assumed that the similarity between the data would have been more noticeable.
5. I will later look into where the problem occurred through visualization of KNN.

Conclusion

KNN is the most basic machine learning model. A machine learning model is a program that can find patterns or make decisions from a previously unseen dataset (databricks, 2023).

In this module, I directly converted k and looked directly at how the result of knn changes depending on data cleansing based on my understanding of columns. knn is the most basic machine learning model and has the advantage of being able to make predictions that other machine learning models cannot, such as when it is difficult to obtain a lot of data. However, it was also confirmed that as simple as the mechanism is, it only works well if the data is evenly distributed. I thought that I could first look at the data through KNN and use it as a step to apply other analysis methods or prepare for other analyzes based on this.

Reference

Amey, Band. (2020, May 23). How to find the optimal value of K in KNN?. Medium. Retrieved from Retrieved from https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb

IBM. (2023). K-Nearest neighbors algorithm. Retrieved from https://www.ibm.com/topics/knn#:~:text=The%20choice%20of%20k%20will,optimal%20k%20for%20your%20dataset.

Soumya. (2020, October 30). drop_First=true during dummy variable creation in pandas. stackoverflow. Retrieved from https://stackoverflow.com/questions/63661560/drop-first-true-during-dummy-variable-creation-in-pandas

Malintha. (2018, June 30). Do I need data separation in KNN?. StackExchange. Retrieved from https://stats.stackexchange.com/questions/353904/do-i-need-data-separation-in-knn

Saturn Cloud. (2023, June 19). How to give column names when reading CSV files using pandas. Retrieved from https://saturncloud.io/blog/how-to-give-column-names-when-reading-csv-files-using-pandas/#:~:text=To%20give%20column%20names%20when%20reading%20a%20CSV%20file%20using,which%20is%20the%20default%20value).

Niharika Aitam. (2023, August 2). How do we adjust the size of the plot in Seaborn?. tutorialspoint. Retrieved from https://www.tutorialspoint.com/how-do-we-adjust-the-size-of-the-plot-in-seaborn

Ritesh, Jain. (2020, May 31). Correlation between categorical variables. Medium. Retrieved from https://medium.com/@ritesh.110587/correlation-between-categorical-variables-63f6bd9bf2f7

Farukh, Hashmi. (2022). How to measure the correlation between two categorical variables in python. thinkingneuron. Retrieved from https://thinkingneuron.com/how-to-measure-the-correlation-between-two-categorical-variables-in-python/

Chet, Lemon. (n.d.). Predicting if income exceeds $50,000 per year based on 1994 US Census Data with Simple Classification Techniques. ucsd. Retrieved from https://cseweb.ucsd.edu/classes/sp15/cse190-c/reports/sp15/048.pdf

ARINDAM GHOSH03. (2021). KNN - 84% accuracy(census data). Kaggle. Retrieved from https://www.kaggle.com/code/arindamghosh03/knn-84-accuracy-census-data/notebook#Train-Test-Split

Malli. (2023). Pandas get unique values in column. SparkBy{Examples}. Retrieved from https://sparkbyexamples.com/pandas/pandas-find-unique-values-from-columns/#:~:text=You%20can%20get%20unique%20values,to%20get%20from%20multiple%20columns.

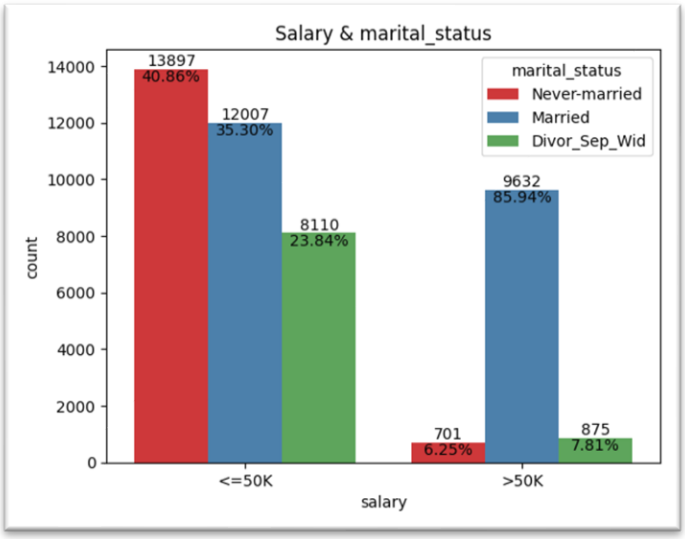
jezrael. (2017, December 20). pandas value_counts( ) not in descending order. stackoverflow. Retrieved from https://stackoverflow.com/questions/47899830/pandas-value-counts-not-in-descending-order

Mohamed Nabil. (2022, November 29). Accuracy score of my KNN model is constant as k increases?. StackExchange. Retrieved from https://stats.stackexchange.com/questions/597306/accuracy-score-of-my-knn-model-is-constant-as-k-increases#:~:text=The%20way%20KNN%20works%20is,of%20samples%20inside%20a%20class.

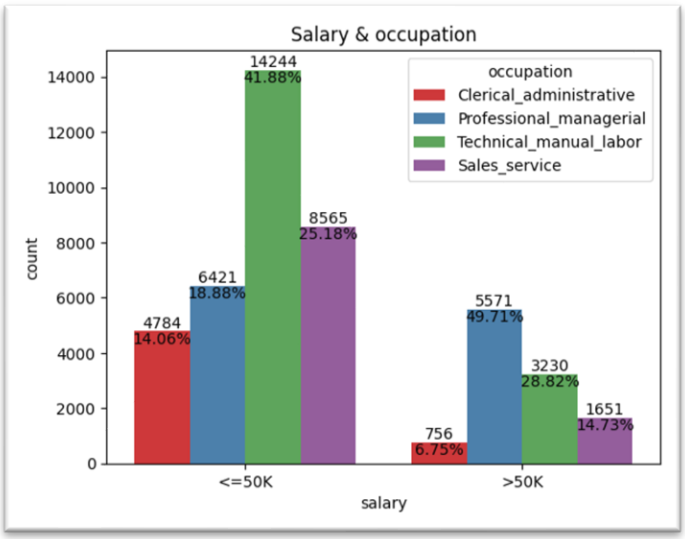
databricks. (2023). Machine Learning Models. Retrieved from https://www.databricks.com/glossary/machine-learning-models

Appendix (Contents):

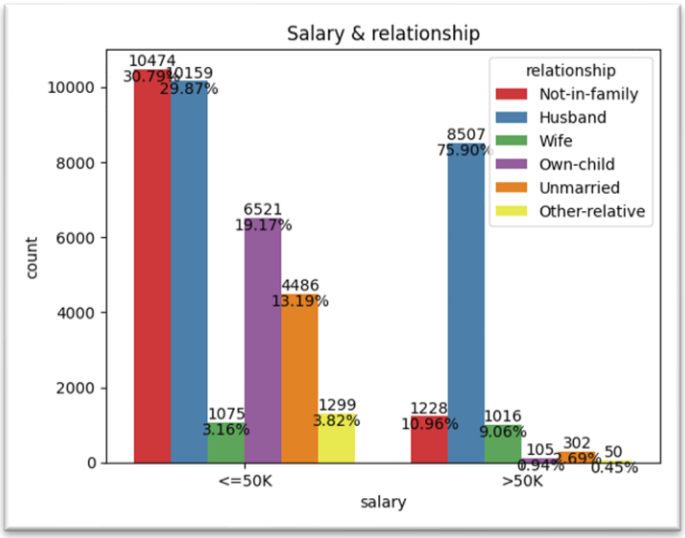
Salary & marital_status:



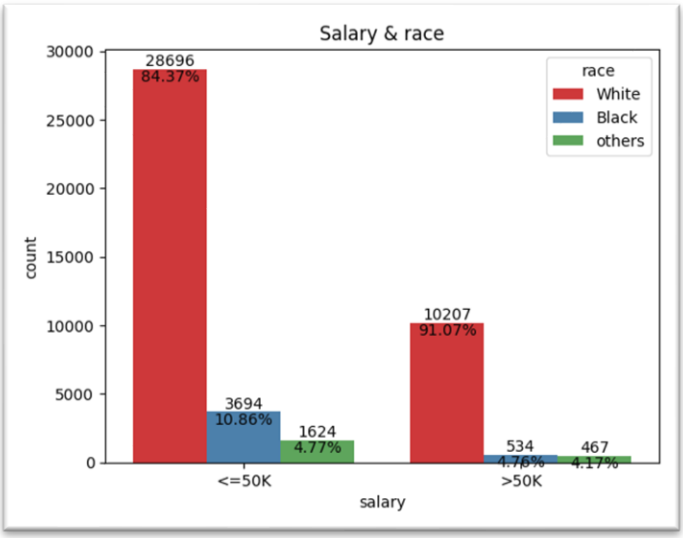
Salary & Occupation:



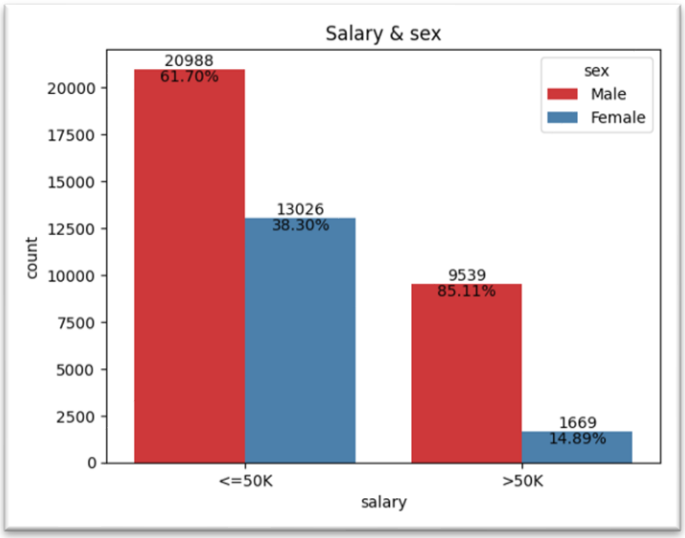
Salary & relationship:



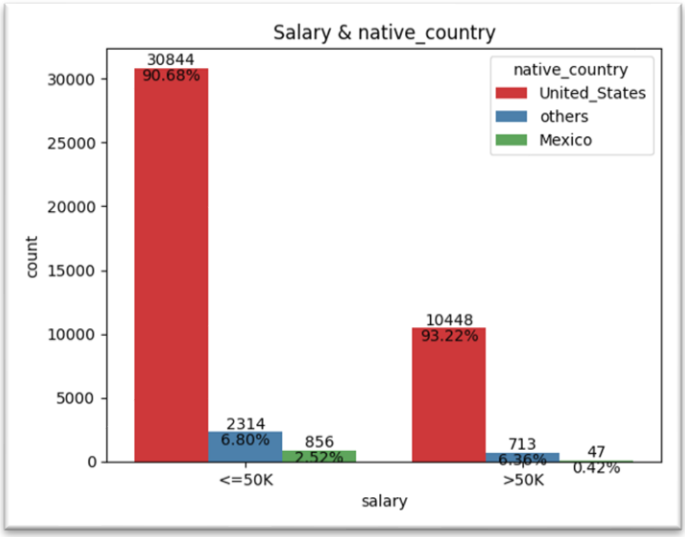
Salary & race:



Salary & sex:



Salary & native_country



Appendix (Python):

```
# In[3]:
import pandas as pd
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from collections import Counter
# ### DATA Import # In[4]:
df = pd.read_csv('adult-all.csv', header=None, names=['age', 'workclass', 'fnlwgt',
'education', 'education_num','marital_status', 'occupation', 'relationship', 'race',
'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'salary'])
df.head()
df_copy=df.copy()
# In[5]:
df.head()
# In[6]:
df.tail()
# ## Visualization & Understanding Dataset # In[7]:
def missing_values(df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent =
(df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1,
keys=['Missing_Number', 'Missing_Percent'])
    return missing_values[missing_values['Missing_Number']>0] def first_looking(df):
    print(colored("Shape:", attrs=['bold']), df.shape,'\n',
          colored('-'*79, 'red', attrs=['bold']),
          colored("\nInfo:\n", attrs=['bold']), sep='')
    print(df.info(), '\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Number of Uniques:\n", attrs=['bold']), df.nunique(),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Missing Values:\n", attrs=['bold']), missing_values(df),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("All Columns:", attrs=['bold']), list(df.columns),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')     df.columns=
df.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')
print(colored("Columns after rename:", attrs=['bold']), list(df.columns),'\n',
              colored('-'*79, 'red', attrs=['bold']), sep='')
# In[8]:
get_ipython().system('pip install colorama')
# In[9]:
import colorama
from colorama import Fore, Style  # maakes strings colored
from termcolor import colored
# In[10]:
missing_values(df)
# In[11]:
first_looking(df)
# In[12]:
```

```
df.describe()
# In[13]:
df=df.drop(['fnlwgt'], axis=1)
# In[14]:
# df.drop('capital_loss',1,inplace=True)
# df.drop('capital_gain',1,inplace=True)
# In[15]:
df.describe()
# In[16]:
pip install dython
# In[17]:
df.head()
# In[18]:
pip install pandas_profiling
# In[19]:
import pandas_profiling
# In[20]:
df1=df.copy()
# In[21]:
df1.profile_report()
# In[22]:
df1['salary'] = df1['salary'].replace('<=50K', 1)
# In[23]:
df1['salary'] = df1['salary'].replace('>50K', 0)
# In[24]:
df1.tail()
# In[25]:
import seaborn as sns
# In[26]:
#distribution of capital_gain
plt.figure(figsize=(10, 5))
sns.histplot(x=df1['capital_gain'])
plt.title("Distribution of Capital_gain")
# In[27]:
#distribution of capital_loss
plt.figure(figsize=(5, 5))
sns.histplot(x=df1['capital_loss'])
plt.title("Distribution of Capital_loss")
# In[28]:
#distribution of age
plt.figure(figsize=(10, 5))
sns.histplot(x=df1['age'])
plt.title("Distribution of age")
# In[29]:
#distribution of education_num
plt.figure(figsize=(10, 5))
ax=sns.histplot(x=df1['education_num'], bins=20)
ax.set_xlim(0, 17) plt.title("education_num")
# In[30]:
#distribution of hours_per_week
plt.figure(figsize=(20, 6))
sns.histplot(x=df1['hours_per_week'], bins=20)
ax.set_xlim(38, 100) plt.title("hours_per_week")
```

```
# #### Correlation Matrix # In[31]:
# num_vars without capital_loss
num_vars2 = ['age', 'education_num', 'capital_gain', 'hours_per_week', 'salary']
df2 = df_copy[num_vars2]
# In[32]:
df2
# In[33]:
sal = pd.get_dummies(df2[["salary"]],drop_first = True)
df2 = pd.concat([df2,sal],axis=1)
# In[34]:
df2
# In[35]:
df2 = df2.drop(['salary'], axis=1)
# In[36]:
df2.info()
# In[37]:
df2['salary_>50K']=df2['salary_>50K'].astype(int)
# In[38]:
census_corr_matrix=df2.corr()
# In[39]:
sns.heatmap(census_corr_matrix, cmap='coolwarm', annot=True)
# In[40]:
# num_vars without capital_loss
num_vars2 = ['age', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week',
'salary']
df2 = df_copy[num_vars2]
# In[41]:
df2
# In[42]:
sal = pd.get_dummies(df2[["salary"]],drop_first = True)
df2 = pd.concat([df2,sal],axis=1)
# In[43]:
df2
# In[44]:
df2 = df2.drop(['salary'], axis=1)
# In[45]:
df2.info()
# In[46]:
df2['salary_>50K']=df2['salary_>50K'].astype(int)
# In[47]:
census_corr_matrix2 = df2.corr()
# In[48]:
sns.heatmap(census_corr_matrix2, cmap='coolwarm', annot=True)
# ### Data Cleansing # In[49]:
cat_vars = [x for x in df.columns if df[x].dtype =="object"]
cat_vars
# In[50]:
num_vars = [x for x in df.columns if x not in cat_vars]
num_vars
# In[51]:
for i in cat_vars:
    print("====="+i+"=====")
    print(df[i].value_counts(normalize=True).sort_values(ascending=False))
```

```
    print("\n")
# #### Histograms # In[52]:
for i in cat_vars:
    plt.title(i)
    sns.countplot(x = df[i])     plt.xticks(rotation=90)
    plt.show()
# In[53]:
# to do iteration of
# df_cat = df_cat[~(df_cat['workclass'] == '?')]
# df_cat = df_cat[~(df_cat['occupation'] == '?')]
# df_cat = df_cat[~(df_cat['native_country'] == '?')] for i in cat_vars:
    df=df[~(df[i] == '?')]
# In[54]:
for i in cat_vars:
    plt.title(i)
    sns.countplot(x = df[i])     plt.xticks(rotation=90)
    plt.show()
# #### Bucketing Criteria = 0.05 # In[55]:
# workclass bucketing
def assign(x):
    less_005 = ['State-gov','Self-emp-inc','Federal-gov','Without-pay']
    if x in less_005:
        return "others"
    else:
        return x df['workclass']=pd.DataFrame(map(assign,df['workclass']))
df["workclass"].value_counts(normalize=True).sort_values(ascending = False)
# In[56]:
# education bucketing
def categorize_education(x):
    HS_below = ['Preschool', '1st-4th', '5th-6th', '7th-8th', '9th', '10th', '11th',
'12th', 'HS-grad']
    Col_nAssoc = ['Some-college', 'Assoc-voc', 'Assoc-acdm']
    Bach_above = ['Bachelors', 'Masters', 'Prof-school', 'Doctorate']
    if x in HS_below:
        return "HS_below"
    elif x in Col_nAssoc:
        return "Col_nAssoc"
    else:
        return "Bach_above" df['education'] =
df['education'].apply(categorize_education)
df['education'].value_counts(normalize=True).sort_values(ascending=False)
# In[57]:
# marital_status bucketing
def categorize_marital(x):
    Married = ['Married-civ-spouse', 'Married-spouse-absent', 'Married-AF-spouse']
    Divor_Sep_Wid = ['Divorced', 'Separated', 'Widowed']
    if x in Married:
        return "Married"
    elif x in Divor_Sep_Wid:
        return "Divor_Sep_Wid"
    else:
        return x df['marital_status'] = df['marital_status'].apply(categorize_marital)
df['marital_status'].value_counts(normalize=True).sort_values(ascending=False)
```

```
# In[58]:
def categorize_occupation(occupation):
    professional_managerial = ['Prof-specialty', 'Exec-managerial']
    clerical_administrative = ['Adm-clerical']
    sales_service = ['Sales', 'Other-service']
    technical_manual = ['Craft-repair', 'Machine-op-inspct', 'Transport-moving',
'Handlers-cleaners', 'Farming-fishing', 'Tech-support', 'Protective-serv', 'Priv-
house-serv', 'Armed-Forces']     if occupation in professional_managerial:
        return 'Professional_managerial'
    elif occupation in clerical_administrative:
        return 'Clerical_administrative'
    elif occupation in sales_service:
        return 'Sales_service'
    elif occupation in technical_manual:
        return 'Technical_manual_labor'
    else:
        return 'Others' df['occupation'] =
df['occupation'].apply(categorize_occupation)
df['occupation'].value_counts(normalize=True).sort_values(ascending=False)
# In[59]:
# race bucketing
def assign(x):
    check = ['Amer-Indian-Eskimo','Other','Asian-Pac-Islander']
    if x in check:
        return "others"
    else:
        return x df["race"] = df['race'].apply(assign)
df['race'].value_counts(normalize=True).sort_values(ascending=False)
# In[60]:
#native.country
def assign(x):
    United_States = ['United-States']
    Mexico = ['Mexico']
    if x in United_States:
        return "United_States"
    elif x in Mexico:
        return "Mexico"
    else:
        return "others" df["native_country"] = df['native_country'].apply(assign)
df['native_country'].value_counts(normalize=True).sort_values(ascending=False)
# In[61]:
df_buck=df.copy()
# In[62]:
df_buck
# In[63]:
for i in cat_vars:
    plt.title(i)
    sns.countplot(x = df[i])     plt.xticks(rotation=90)
    plt.show()
# In[64]:
for i in cat_vars:
    Countplot = sns.countplot(x='salary', hue=i, data=df, palette='Set1')
    Countplot.set_title(f"Salary & {i}")
```

```
    plt.show()
# In[65]:
for i in cat_vars:
    # Create a bar plot
    Countplot = sns.countplot(x='salary', hue=i, data=df, palette='Set1')
    Countplot.set_title(f"Salary & {i}")     # Calculate the total count for each hue
category
    total_counts = [sum(Countplot.patches[j].get_height() for j in range(i,
len(Countplot.patches), len(df['salary'].unique()))) for i in
range(len(df['salary'].unique()))]     # Annotate the bars with count and percentage
within each hue category
    for j, p in enumerate(Countplot.patches):
        height = p.get_height()
        hue_index = j % len(df['salary'].unique())
        Countplot.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),
ha='center', va='bottom')
        percentage = (height / total_counts[hue_index]) * 100
        Countplot.annotate(f'{percentage:.2f}%', (p.get_x() + p.get_width() / 2.,
height), ha='center', va='top')     plt.show()
# ### Creating dummy variables # In[66]:
#### 'workclass'
wc = pd.get_dummies(df[["workclass"]],drop_first = True)
df = pd.concat([df,wc],axis=1)
### 'education'
edu = pd.get_dummies(df[["education"]],drop_first = True)
df = pd.concat([df,edu],axis=1)
### 'marital.status'
ms = pd.get_dummies(df[["marital_status"]],drop_first = True)
df = pd.concat([df,ms],axis=1)
### 'occupation'
occu =pd.get_dummies(df[["occupation"]],drop_first = True)
df = pd.concat([df,occu],axis=1)
### 'relationship'
rel =pd.get_dummies(df[["relationship"]],drop_first = True)
df = pd.concat([df,rel],axis=1)
### 'race'
race = pd.get_dummies(df[["race"]],drop_first = True)
df = pd.concat([df,race],axis=1)
### 'native.country'
nc = pd.get_dummies(df[["native_country"]],drop_first = True)
df = pd.concat([df,nc],axis=1)
# In[67]:
df.head()
# In[68]:
###Creating a function for converting binary variable
def binary(x):
    if x == "Male":
        return 1
    else:
        return 0 df["sex"] = df["sex"].apply(binary)
# In[69]:
###dropping the columns as dummy has been created
df.drop(['workclass','education','marital_status','occupation','relationship','race','
```

```
native_country'],axis=1,inplace=True)
# In[70]:
df.head()
# In[71]:
df['salary'] = df['salary'].apply(lambda x:1 if x == "<=50K" else 0)
# In[72]:
df['salary'].value_counts()
# In[73]:
df_knn=df.copy()
# In[74]:
counts = df['salary'].value_counts()
labels = ['<=50K', '>50K'] # Create a pie chart
fig, ax = plt.subplots()
ax.pie(counts, labels=labels, autopct='%1.1f%%', startangle=140) # Add count labels to
the pie chart slices
for i, count in enumerate(counts):
    angle = (sum(counts[:i]) + count / 2) / sum(counts) * 360
    x = 0.7 * np.cos(np.deg2rad(angle))
    y = 0.7 * np.sin(np.deg2rad(angle))
    ax.text(x, y, f'{count}', ha='center', va='center') plt.title('Salary
Distribution')
plt.show()
# In[74]:# In[75]:
df_knn
df_knn2=df_knn.copy()
# In[76]:
df_knn2
# In[77]:
df_knn.drop('capital_loss',1,inplace=True)
# In[78]:
df_knn
# In[79]:
X = df_knn.drop("salary",axis=1)
X.head()
# In[80]:
y = df_knn["salary"]
y.head()
# ### Train-Test Split # In[81]:
from sklearn.model_selection import train_test_split
# In[82]:
###Splitting the data into train-test set
X_train,X_test,y_train,y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=100)
# ### Feature Scaling # In[83]:
from sklearn.preprocessing import StandardScaler
# In[84]:
sc = StandardScaler()
# In[85]:
num_vars = ['age', 'education_num', 'capital_gain', 'hours_per_week']
# In[86]:
X_train[num_vars] = sc.fit_transform(X_train[num_vars])
# In[87]:
X_train[num_vars]
```

```
# In[88]:
X_train
# ## KNN modeling # In[89]:
from sklearn.neighbors import KNeighborsClassifier
# In[90]:
from sklearn.metrics import classification_report,confusion_matrix
# In[91]:
X = df_knn.drop("salary",axis=1)
X.head()
# In[92]:
y = df_knn["salary"]
y.head()
# In[93]:
###Splitting the data into train-test set
X_train,X_test,y_train,y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=100)
# In[94]:
X_train[num_vars] = sc.fit_transform(X_train[num_vars])
# In[95]:
knn = KNeighborsClassifier(n_neighbors=7)
knn = knn.fit(X_train,y_train)
# In[96]:
y_pred = knn.predict(X_test)
# In[97]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[98]:
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns def make_confusion_matrix(cf,
                        group_names=None,
                        categories='auto',
                        count=True,
                        percent=True,
                        cbar=True,
                        xyticks=True,
                        xyplotlabels=True,
                        sum_stats=True,
                        figsize=None,
                        cmap='Blues',
                        title=None):
    '''
    This function will make a pretty plot of an sklearn Confusion Matrix cm using a
Seaborn heatmap visualization.     Arguments
    ---------
    cf:            confusion matrix to be passed in     group_names:   List of strings
that represent the labels row by row to be shown in each square.     categories:
List of strings containing the categories to be displayed on the x,y axis. Default is
'auto'     count:         If True, show the raw number in the confusion matrix.
Default is True.     normalize:     If True, show the proportions for each category.
Default is True.     cbar:          If True, show the color bar. The cbar values are
based off the values in the confusion matrix.
                  Default is True.     xyticks:       If True, show x and y ticks.
```

```
Default is True.     xyplotlabels:  If True, show 'True Label' and 'Predicted Label'
on the figure. Default is True.     sum_stats:     If True, display summary statistics
below the figure. Default is True.     figsize:       Tuple representing the figure
size. Default will be the matplotlib rcParams value.     cmap:          Colormap of
the values displayed from matplotlib.pyplot.cm. Default is 'Blues'
                  See http://matplotlib.org/examples/color/colormaps_reference.html
title:         Title for the heatmap. Default is None.     '''
    # CODE TO GENERATE TEXT INSIDE EACH SQUARE
    blanks = ['' for i in range(cf.size)]     if group_names and
len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
    else:
        group_labels = blanks     if count:
        group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
    else:
        group_counts = blanks     if percent:
        group_percentages = ["{0:.2%}".format(value) for value in
cf.flatten()/np.sum(cf)]
    else:
        group_percentages = blanks     box_labels = [f"{v1}{v2}{v3}".strip() for v1,
v2, v3 in zip(group_labels,group_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])
    # CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
    if sum_stats:
        #Accuracy is sum of diagonal divided by total observations
        accuracy  = np.trace(cf) / float(np.sum(cf))         #if it is a binary
confusion matrix, show some more stats
        if len(cf)==2:
            #Metrics for Binary Confusion Matrices
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.3f}\nF1
Score={:0.3f}".format(
                accuracy,precision,recall,f1_score)
        else:
            stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    else:
        stats_text = ""
    # SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
    if figsize==None:
        #Get default figure size if not set
        figsize = plt.rcParams.get('figure.figsize')     if xyticks==False:
        #Do not show categories if xyticks is False
        categories=False
    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)

sns.heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=categories,ytic
klabels=categories)     if xyplotlabels:
        plt.ylabel('True label')
        plt.xlabel('Predicted label' + stats_text)
    else:
```

```
        plt.xlabel(stats_text)    if title:
        plt.title(title)
# In[99]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[100]:
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
# In[101]:
print('Accuracy: %.3f' % accuracy_score(y_test,y_pred))
print('Precision: %.3f' % precision_score(y_test,y_pred))
print('Recall: %.3f' % recall_score(y_test,y_pred))
print('F1 Score: %.3f' % f1_score(y_test,y_pred))
# ### knn with 3 # In[102]:
knn3 = KNeighborsClassifier(n_neighbors=3)
knn3 = knn3.fit(X_train,y_train)
# In[103]:
y_pred = knn3.predict(X_test)
# In[104]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[105]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[106]:
print('Accuracy: %.3f' % accuracy_score(y_test,y_pred))
print('Precision: %.3f' % precision_score(y_test,y_pred))
print('Recall: %.3f' % recall_score(y_test,y_pred))
print('F1 Score: %.3f' % f1_score(y_test,y_pred))
# ### knn with 11
#  # In[107]:
knn11 = KNeighborsClassifier(n_neighbors=11)
knn11 = knn11.fit(X_train,y_train)
# In[108]:
y_pred = knn11.predict(X_test)
# In[109]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[110]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[111]:
```

```
print('Accuracy: %.3f' % accuracy_score(y_test,y_pred))
print('Precision: %.3f' % precision_score(y_test,y_pred))
print('Recall: %.3f' % recall_score(y_test,y_pred))
print('F1 Score: %.3f' % f1_score(y_test,y_pred))
# ### knn with 9
#  # In[112]:
knn9 = KNeighborsClassifier(n_neighbors=9)
knn9 = knn9.fit(X_train,y_train)
# In[113]:
y_pred = knn9.predict(X_test)
# In[114]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[115]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[116]:
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
# ### knn with 20
#  # In[117]:
knn20 = KNeighborsClassifier(n_neighbors=20)
knn20 = knn20.fit(X_train,y_train)
# In[118]:
y_pred = knn20.predict(X_test)
# In[119]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[120]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[121]:
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
# ### knn with 5
#  # In[122]:
knn5 = KNeighborsClassifier(n_neighbors=5)
knn5 = knn5.fit(X_train,y_train)
# In[123]:
y_pred = knn5.predict(X_test)
# In[124]:
```

```
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[125]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[126]:
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
# In[126]:# In[126]:# In[126]:# ### Chi-square # In[127]:
import os as os
from itertools import product
import scipy.stats as ss
# In[128]:
df_buck.info()
# In[129]:
df_cat = pd.DataFrame(data = df_buck.dtypes, columns =
                                          ['a']).reset_index()
cat_var = list(df_cat['index'].loc[df_cat['a'] == 'object'])
cat_var
# In[130]:
df_cat = df_buck[cat_var]
df_cat
# In[131]:
df_cat['salary'] = df_cat['salary'].replace('<=50K', 1)
df_cat['salary'] = df_cat['salary'].replace('>50K', 0)
# In[132]:
df_cat_v1 = df_cat.dropna()
df_cat_v1.shape
# In[133]:
cat_var1 = ('workclass', 'education', 'marital_status', 'occupation', 'relationship',
'race', 'sex', 'native_country', 'salary')
# In[134]:
cat_var2 = ('workclass', 'education', 'marital_status', 'occupation', 'relationship',
'race', 'sex', 'native_country', 'salary')
# In[135]:
cat_var_prod = list(product(cat_var1,cat_var2, repeat = 1))
# In[136]:
result = [] for i in cat_var_prod:
    if i[0] != i[1]:
        result.append((i[0], i[1], list(ss.chi2_contingency(pd.crosstab(
                            df_cat_v1[i[0]], df_cat_v1[i[1]])))[1]))
# In[137]:
chi_test_output = pd.DataFrame(result, columns = ['var1', 'var2', 'coeff'])
# In[138]:
chi_test_output.pivot(index='var1', columns='var2', values='coeff')
# In[139]:
heatmap_data = chi_test_output.pivot(index='var1', columns='var2', values='coeff') #
```

```
Create a heatmap using Seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_data, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Chi-Square Test Heatmap')
plt.show()
# In[140]:
df=df_buck.copy()
# In[141]:
df.drop('workclass',1,inplace=True)
# In[142]:
df.drop('capital_loss',1,inplace=True)
# In[143]:
df
# ### Creating dummy variables # In[144]:
### 'education'
edu = pd.get_dummies(df[["education"]],drop_first = True)
df = pd.concat([df,edu],axis=1)
### 'marital.status'
ms = pd.get_dummies(df[["marital_status"]],drop_first = True)
df = pd.concat([df,ms],axis=1)
### 'occupation'
occu =pd.get_dummies(df[["occupation"]],drop_first = True)
df = pd.concat([df,occu],axis=1)
### 'relationship'
rel =pd.get_dummies(df[["relationship"]],drop_first = True)
df = pd.concat([df,rel],axis=1)
### 'race'
race = pd.get_dummies(df[["race"]],drop_first = True)
df = pd.concat([df,race],axis=1)
### 'native.country'
nc = pd.get_dummies(df[["native_country"]],drop_first = True)
df = pd.concat([df,nc],axis=1)
# In[145]:
df.head()
# In[146]:
###Creating a function for converting binary variable
def binary(x):
    if x == "Male":
        return 1
    else:
        return 0 df["sex"] = df["sex"].apply(binary)
# In[147]:
###dropping the columns as dummy has been created
df.drop(['education','marital_status','occupation','relationship','race','native_count
ry'],axis=1,inplace=True)
# In[148]:
df.head()
# In[149]:
df['salary'] = df['salary'].apply(lambda x:1 if x == "<=50K" else 0)
# In[150]:
df['salary'].value_counts()
# In[151]:
df_knn=df.copy()
```

```
# In[152]:
counts = df['salary'].value_counts()
labels = ['<=50K', '>50K'] # Create a pie chart
fig, ax = plt.subplots()
ax.pie(counts, labels=labels, autopct='%1.1f%%', startangle=140) # Add count labels to
the pie chart slices
for i, count in enumerate(counts):
    angle = (sum(counts[:i]) + count / 2) / sum(counts) * 360
    x = 0.7 * np.cos(np.deg2rad(angle))
    y = 0.7 * np.sin(np.deg2rad(angle))
    ax.text(x, y, f'{count}', ha='center', va='center') plt.title('Salary
Distribution')
plt.show()
# In[153]:
df_knn
df_knn2=df_knn.copy()
# In[154]:
X = df_knn.drop("salary",axis=1)
X.head()
# In[155]:
y = df_knn["salary"]
y.head()
# ### Train-Test Split # In[156]:
from sklearn.model_selection import train_test_split
# In[157]:
###Splitting the data into train-test set
X_train,X_test,y_train,y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=100)
# ### Feature Scaling # In[158]:
from sklearn.preprocessing import StandardScaler
# In[159]:
sc = StandardScaler()
# In[160]:
X_train[num_vars] = sc.fit_transform(X_train[num_vars])
# In[161]:
X_train[num_vars]
# In[162]:
X_train
# ## KNN modeling # In[163]:
from sklearn.neighbors import KNeighborsClassifier
# In[164]:
from sklearn.metrics import classification_report,confusion_matrix
# In[165]:
X = df_knn.drop("salary",axis=1)
X.head()
# In[166]:
y = df_knn["salary"]
y.head()
# In[167]:
###Splitting the data into train-test set
X_train,X_test,y_train,y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=100)
# In[168]:
```

```
X_train[num_vars] = sc.fit_transform(X_train[num_vars])
# In[169]:
num_vars
# In[170]:
knn = KNeighborsClassifier(n_neighbors=7)
knn = knn.fit(X_train,y_train)
# In[171]:
y_pred = knn.predict(X_test)
# In[172]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[173]:
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns def make_confusion_matrix(cf,
                          group_names=None,
                          categories='auto',
                          count=True,
                          percent=True,
                          cbar=True,
                          xyticks=True,
                          xyplotlabels=True,
                          sum_stats=True,
                          figsize=None,
                          cmap='Blues',
                          title=None):
    '''
    This function will make a pretty plot of an sklearn Confusion Matrix cm using a
Seaborn heatmap visualization.     Arguments
    ---------
    cf:            confusion matrix to be passed in     group_names:   List of strings
that represent the labels row by row to be shown in each square.     categories:
List of strings containing the categories to be displayed on the x,y axis. Default is
'auto'     count:         If True, show the raw number in the confusion matrix.
Default is True.     normalize:     If True, show the proportions for each category.
Default is True.     cbar:          If True, show the color bar. The cbar values are
based off the values in the confusion matrix.
                  Default is True.     xyticks:       If True, show x and y ticks.
Default is True.     xyplotlabels:  If True, show 'True Label' and 'Predicted Label'
on the figure. Default is True.     sum_stats:     If True, display summary statistics
below the figure. Default is True.     figsize:       Tuple representing the figure
size. Default will be the matplotlib rcParams value.     cmap:          Colormap of
the values displayed from matplotlib.pyplot.cm. Default is 'Blues'
                  See http://matplotlib.org/examples/color/colormaps_reference.html
title:         Title for the heatmap. Default is None.     '''
    # CODE TO GENERATE TEXT INSIDE EACH SQUARE
    blanks = ['' for i in range(cf.size)]     if group_names and
len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
    else:
        group_labels = blanks     if count:
        group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
    else:
```

```
        group_counts = blanks     if percent:
        group_percentages = ["{0:.2%}".format(value) for value in
cf.flatten()/np.sum(cf)]
    else:
        group_percentages = blanks     box_labels = [f"{v1}{v2}{v3}".strip() for v1,
v2, v3 in zip(group_labels,group_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])
    # CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
    if sum_stats:
        #Accuracy is sum of diagonal divided by total observations
        accuracy  = np.trace(cf) / float(np.sum(cf))         #if it is a binary
confusion matrix, show some more stats
        if len(cf)==2:
            #Metrics for Binary Confusion Matrices
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.3f}\nF1
Score={:0.3f}".format(
                accuracy,precision,recall,f1_score)
        else:
            stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    else:
        stats_text = ""
    # SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
    if figsize==None:
        #Get default figure size if not set
        figsize = plt.rcParams.get('figure.figsize')     if xyticks==False:
        #Do not show categories if xyticks is False
        categories=False
    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)

sns.heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=categories,ytic
klabels=categories)     if xyplotlabels:
        plt.ylabel('True label')
        plt.xlabel('Predicted label' + stats_text)
    else:
        plt.xlabel(stats_text)     if title:
        plt.title(title)
# In[174]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[175]:
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
# In[176]:
print('Accuracy: %.3f' % accuracy_score(y_test,y_pred))
print('Precision: %.3f' % precision_score(y_test,y_pred))
print('Recall: %.3f' % recall_score(y_test,y_pred))
```

```
print('F1 Score: %.3f' % f1_score(y_test,y_pred))
# ### knn with 3 # In[177]:
knn3 = KNeighborsClassifier(n_neighbors=3)
knn3 = knn3.fit(X_train,y_train)
# In[178]:
y_pred = knn3.predict(X_test)
# In[179]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[180]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[181]:
print('Accuracy: %.3f' % accuracy_score(y_test,y_pred))
print('Precision: %.3f' % precision_score(y_test,y_pred))
print('Recall: %.3f' % recall_score(y_test,y_pred))
print('F1 Score: %.3f' % f1_score(y_test,y_pred))
# ### knn with 11
#  # In[182]:
knn11 = KNeighborsClassifier(n_neighbors=11)
knn11 = knn11.fit(X_train,y_train)
# In[183]:
y_pred = knn11.predict(X_test)
# In[184]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[185]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[186]:
print('Accuracy: %.3f' % accuracy_score(y_test,y_pred))
print('Precision: %.3f' % precision_score(y_test,y_pred))
print('Recall: %.3f' % recall_score(y_test,y_pred))
print('F1 Score: %.3f' % f1_score(y_test,y_pred))
# ### knn with 9
#  # In[187]:
knn9 = KNeighborsClassifier(n_neighbors=9)
knn9 = knn9.fit(X_train,y_train)
# In[188]:
y_pred = knn9.predict(X_test)
# In[189]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[190]:
labels = ['True Neg','False Pos','False Neg','True Pos']
```

```
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[191]:
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
# ### knn with 20
#  # In[192]:
knn20 = KNeighborsClassifier(n_neighbors=20)
knn20 = knn20.fit(X_train,y_train)
# In[193]:
y_pred = knn20.predict(X_test)
# In[194]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[195]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[196]:
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
# ### knn with 5
#  # In[197]:
knn5 = KNeighborsClassifier(n_neighbors=5)
knn5 = knn5.fit(X_train,y_train)
# In[198]:
y_pred = knn5.predict(X_test)
# In[199]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[200]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[201]:
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
# ## How to choose k # In[205]:
```

```
error_rate = []
for i in range(2,40):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test)) plt.figure(figsize=(10,6))
plt.plot(range(2,40),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(error_rate)))
# In[206]:
acc = []
# Will take some time
from sklearn import metrics
for i in range(2,40):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(X_train,y_train)
    yhat = neigh.predict(X_test)
    acc.append(metrics.accuracy_score(y_test, yhat)) plt.figure(figsize=(10,6))
plt.plot(range(2,40),acc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-",max(acc),"at K =",acc.index(max(acc)))
# ### knn with 19 # In[207]:
knn19 = KNeighborsClassifier(n_neighbors=19)
knn19 = knn19.fit(X_train,y_train)
# In[208]:
y_pred = knn5.predict(X_test)
# In[209]:
cf_matrix=confusion_matrix(y_test,y_pred)
cf_matrix
# In[210]:
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
# In[211]:
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
```