CSE 578: DATA VISUALIZATION

Project Final Report

SRIKAR ANUDEEP REMANI(1225320899)

Team Goals and Business Objective:

The project's goal is to find patterns in the dataset through the creation of visualizations, show them to UVW executives, and determine the elements that influence a person's compensation. After conducting preliminary study, we intend to create machine learning models that can accurately estimate each person's income. The outcomes of the analysis and the application would then be used by the UVW marketing team to customize how it spoke to the target audience in its marketing campaigns.

Assumptions:

1. While I was receiving the dataset, I began to think that it was precise and correct. It cannot be incorrect and must accurately represent the desired information without being misleading. If it is not apparent how the data will be used, it may be off-target or cost more than is necessary to achieve accuracy and precision.

2. **Validity and Legitimacy:** I presume the dataset to be genuine and lawful. For instance, the gender-related questions in the dataset only allow for a restricted number of possibilities and do not allow for open-ended responses. Any responses other than these would not be deemed as authentic or actual according to the dataset's criterion.

3. **Relevance and Timeliness:** I assumed that the dataset had been put together on schedule. The erroneous conclusions may be drawn as a result of inaccurate information that was gathered either too early or too late. If the salaries in the dataset were gathered during a recession, for example, the dataset would not be justified and the findings would not be reliable.

4. **Completeness and Accuracy:** Inaccurate and inaccurate information can both be detrimental. I began the analysis by assuming that the dataset is complete and that there are no gaps in the data collection process in order to prevent only having a partial understanding of the entire situation that will be demonstrated. If there isn't a clear picture of how operations are going, decisions won't be made with knowledge. To determine whether or not the needs are being satisfied, it is essential to know each requirement that goes into a thorough data collection.

5. **Feature Selection:** We think that the features that are best at class prediction will provide patterns that are simpler to comprehend. The great majority of analyses and visualizations are consequently produced using the selected unbiased features.

User Stories:

- **Data Cleaning:** To prepare the dataset for running the models, preprocessing and data cleaning were performed. There are some erroneous or missing pieces of data ("?").
- **Attribute Analysis:** Individual attribute analysis was carried out using the 14 features of the dataset, which were done by me. I selected the initial set of features that has to be considered for additional analysis utilizing individual attribute analysis.
- **Visualizations:** For the factors discovered throughout the feature engineering process, I worked on multivariate analysis and graphs.
- **Machine Learning:** I focused on developing a machine learning-based model that can estimate a person's salary.
- **Executive and Systems Report :** A report on executive and systems will be prepared.

Visualizations:

In order to look for trends, each attribute was initially analyzed independently using data exploration techniques. As data visualization techniques, pie charts, bar charts, histograms, and box plots were used to achieve this. Other statistical techniques, such as the mean, median, and standard deviation, were used when needed to better comprehend the distribution that lay beneath. Later, using methods like mosaic plots, parallel coordinate plots, and scatter plots, we used multivariate feature analysis to support our initial research. This allowed us to pinpoint a number of important characteristics.

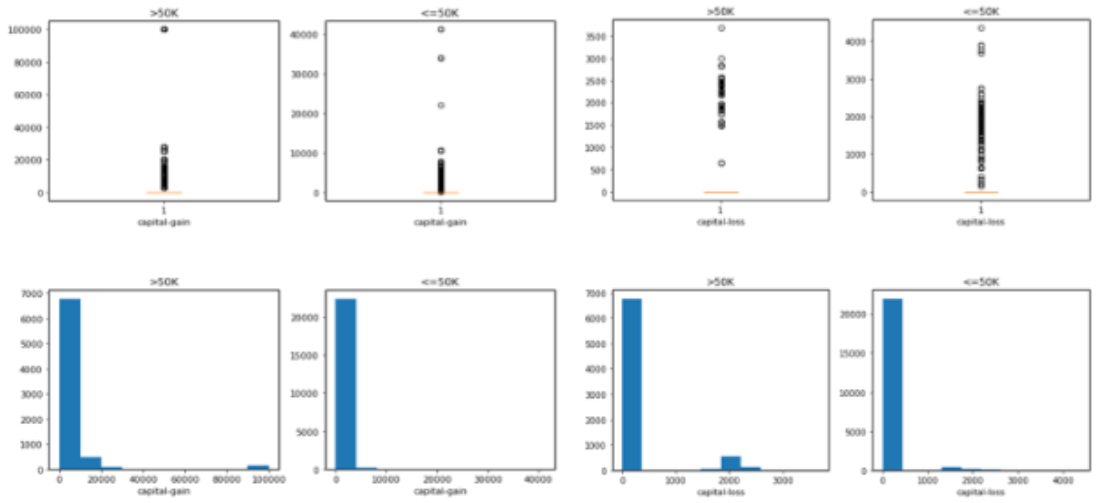
On top of that, the important components' feature engineering for the machine learning analysis that was developed. The next step entailed instructing a few machine learning models on how to determine the individual's wage. This helped us comprehend the significance of the traits and verified that our earlier study of the features was accurate

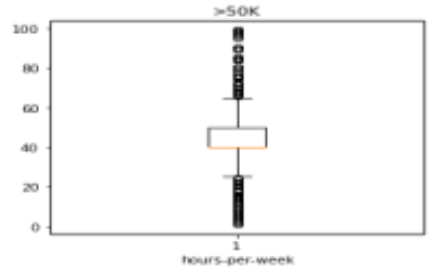
Initial analysis of individual features:

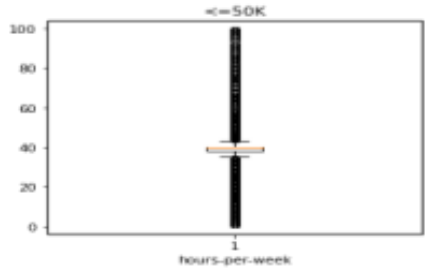
1. **Age:** Your compensation substantially decreases as you get older. Those who make more than $50,000 a year are most likely to be between the ages of 30 and 55.
2. **Work class:** This element shouldn't be taken into account because the data are comparable for both classes. How many government employees make more than $50,000 a year is an interesting statistic.
3. **Fnlwgt:** This property has a salary barrier of $50,000 for values over $45,000.
4. **Education:** It is clear from the graphs that people who have completed their bachelor's degree or above are more likely to earn at least $50,000 annually.
5. **Education-num:** This variable and the education attribute are identical.
6. **Marital status:** It's interesting to note that 85% of those who earn more than $50,000 annually describe themselves as married or in a civil union. Additionally, it demonstrates how much less singles earn.
7. **Occupation:** It can be assumed that more than 50% of people with yearly wages of above $50,000 hold managerial, administrative, or professorial specialties. There is an equal distribution of people who work in crafts and repairs between the two classes.
8. **Relationship:** As you can see, 75% of all people who make more than $50,000 fall within the relationship category "Husband," which has a greater effect on wages. Even the "Marital-status" column reveals that those who are single or unmarried frequently earn less than $50,000.
9. **Race:** The "Race" feature hardly or never affects a person's wage range. Both classrooms' majority of students self-identify as "White."
10. **Sex:** Among all groups, men are more prevalent than women, albeit this trend becomes more pronounced at higher pay levels.
11. **Capital gain:** If a person has a significant capital gain, they are more likely to make over $50,000 annually.
12. **Capital loss:** For all different types of people, the capital loss is typically the same. Thus using this to classify the data might not be a good idea.
13. **Hours per week:** Those who put in more than 40 hours a week are usually considered to earn ">50K."
14. **Native Country :** The following nationalities are more likely to have an annual income of less than $50,000:.

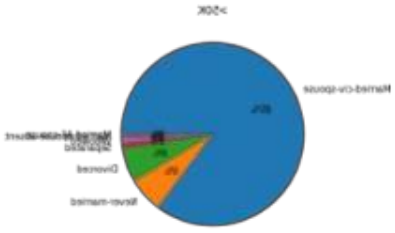
UNIVARIATE ANALYSIS :

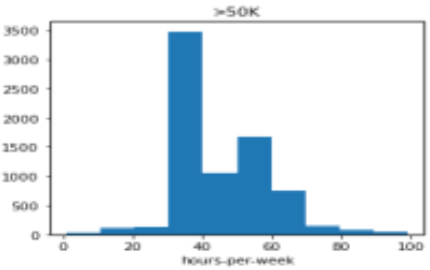
Data Analysis/Visualization of each feature:

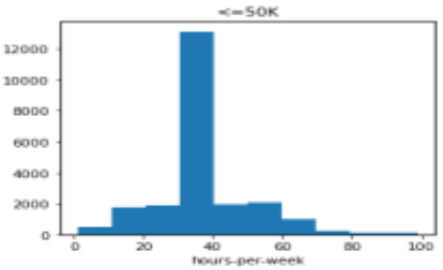


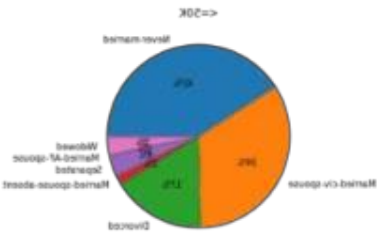

>50K
hours-per-week


<=50K
hours-per-week

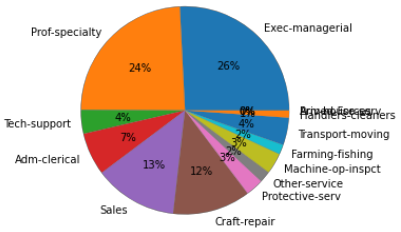


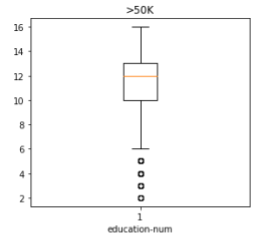

>50K
hours-per-week

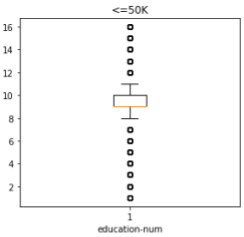

<=50K
hours-per-week



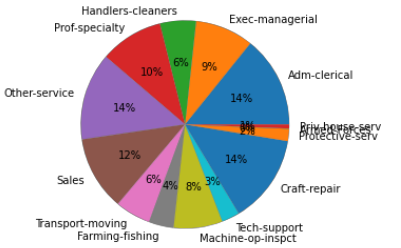
>50K

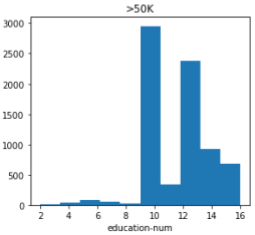

Exec-managerial
Prof-specialty
Handlers-cleaners
Transport-moving
Farming-fishing
Machine-op-inspct
Other-service
Protective-serv
Craft-repair
Sales
Adm-clerical
Tech-support

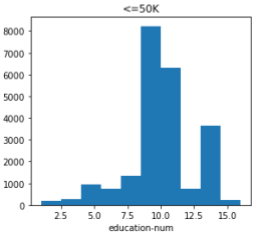

>50K
education-num

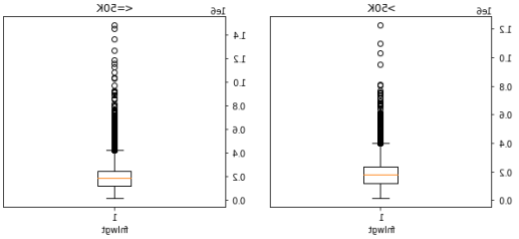

<=50K
education-num

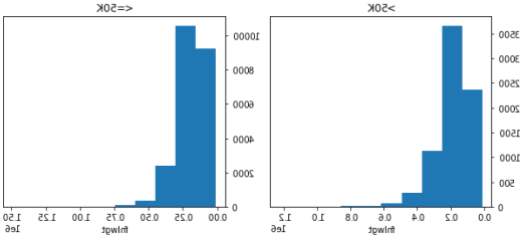
<=50K


Handlers-cleaners
Prof-specialty
Exec-managerial
Adm-clerical
Other-service
Priv-house-serv
Protective-serv
Sales
Craft-repair
Transport-moving
Farming-fishing
Machine-op-inspct
Tech-support


>50K
education-num


<=50K
education-num

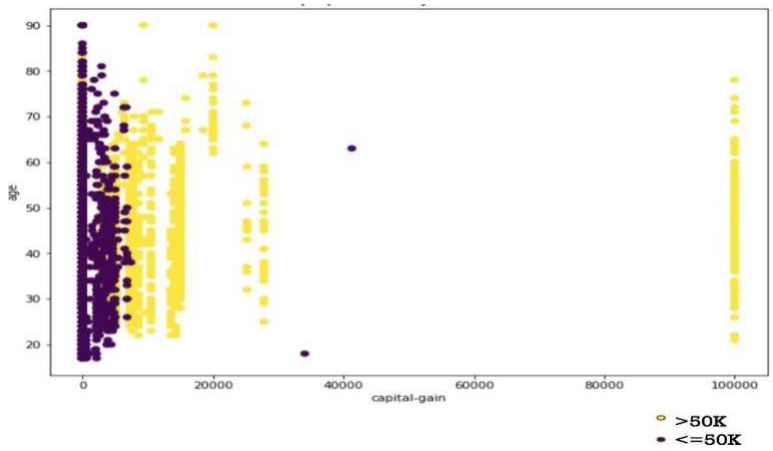

>50K
<=50K
fnlwgt


>50K
<=50K
fnlwgt

MULTIVARIATE ANALYSIS:

The following notable graphs were useful in identifying the main characteristics of the dataset:

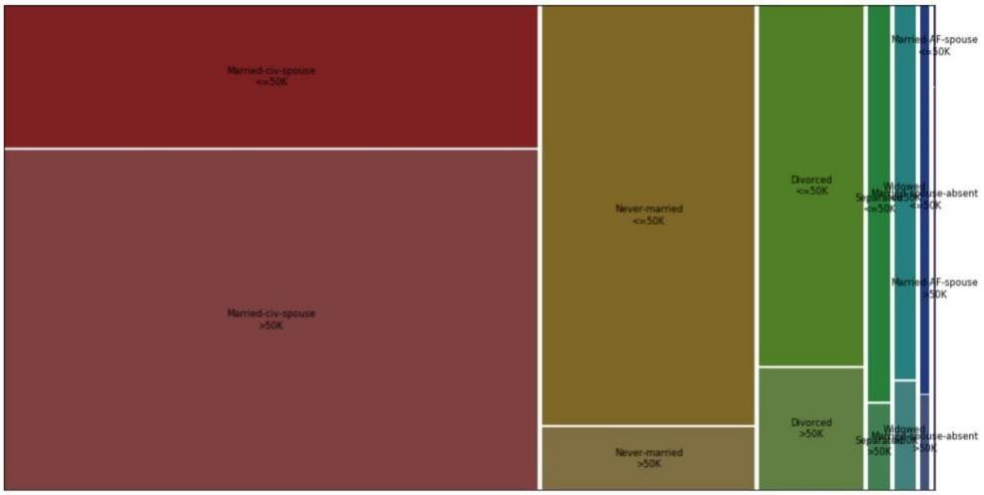
Scatter plot: Capital-gain and Age:



Inferences:

1. There appears to be a separation between the two classes of data, save a few outliers.
2. Those who have made significant capital gains are more likely to earn more than $50,000 yearly.

Mosaic plot: Marital status and Salary:



Inferences:

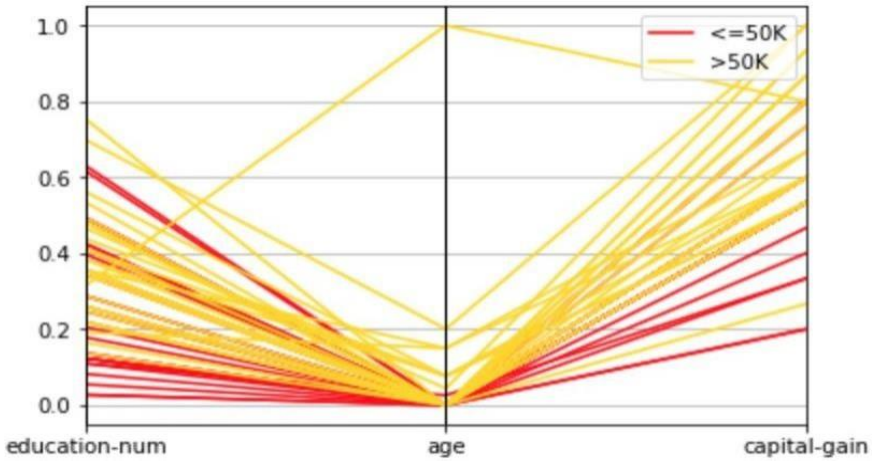
1. The distribution of the two classes for the bulk of categorical data is severely skewed, indicating that this characteristic can be utilized to differentiate between the two classes.
2. "Married-civ-spouses" are more likely to earn more than $50,000 per year than other groups of people.
3. Those who identify as "never married" are more likely to make under $50,000 a year.

Mosaic plot: Occupation and salary:

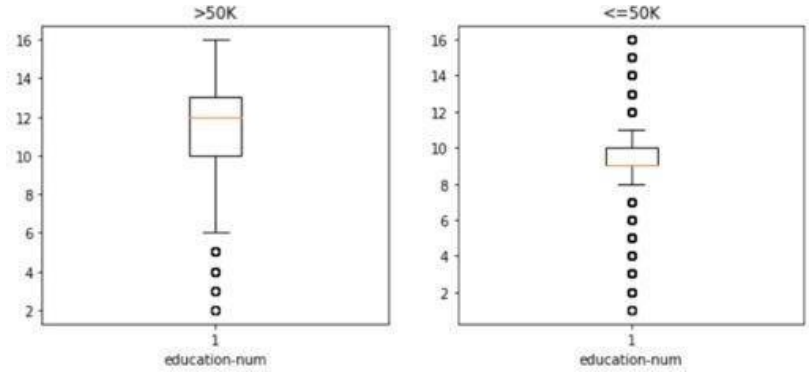
Inferences:

1. The distribution of the two classes for the bulk of categorical data is severely skewed, indicating that this characteristic can be utilized to differentiate between the two classes.
2. Those with "Exec-managerial" and "Prof-speciality" jobs are more likely to make above $50,000 annually.

Parallel Coordinate Plot: Capital-gain, Education-num and Age



Box plot: education-num



Inferences:

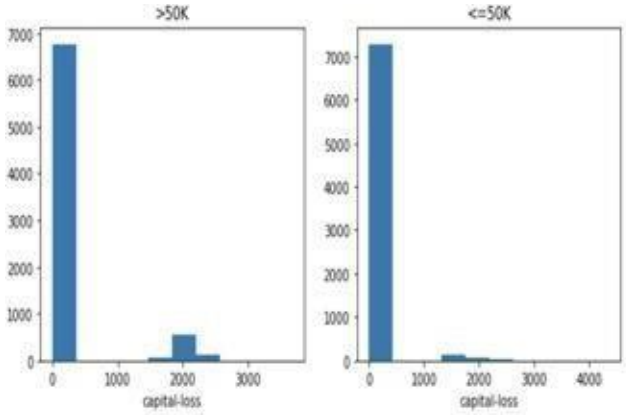
1. The parallel coordinate plot demonstrates how these three elements combine to distinguish between theyellow and red lines.

2. The box plot shows that there is a considerable difference in the distribution of education between the twoclasses of data.

3. More education increases the likelihood of earning above $50,000 annually.

4. Elderly are more likely to have higher wages than younger persons.

Histogram: capital-loss



Inferences:

1. As can be observed from the graphic, the distributions for the two classes of data are identical, suggesting that this property may not be useful in differentiating between the two classes of data.

MACHINE LEARNING ANALYSIS:

What must be done:
1. **Feature engineering:** There was no feature engineering done to the numerical features. According to our preliminary data exploration research, each item of categorical data is given a number value based on the characteristic that makes up that category.
2. **Data Normalization:** Data was standardized by scaling each feature to a value between 0 and 1. This ensures that each attribute receives the same weight from the ML algorithms.
3. **Data Distribution:** The distribution of the data was 80:20, with 80% of the data being used for training and 20% being utilized for testing.
4. **ML Model Training:** A few examples of the techniques used to train the ML models include Gradient Boosting, Random Forest, Support Vector Machine, Neural Network, Bagging, and Logistic Regression.
5. **Hyperparameter tuning:** To get the most accurate results from each of these algorithms, the hyperparameters were adjusted.
6. **Evaluating Performance:** Metrics such as F1 score, recall, precision, and accuracy were used to assess how well the trained models performed on the test dataset. The performance review's findings are as follows:

Acc	F1	Prec	Recall	Model
0.8118	0.8193	0.7902	0.8506	BaggingClassifier()
0.7799	0.7946	0.7469	0.8488	MultinomialNB()
0.8044	0.8136	0.7793	0.8510	LogisticRegression()
0.8080	0.8122	0.7971	0.8279	AdaBoostClassifier()
0.8189	0.8292	0.7868	0.8764	RandomForestClassifier()
0.8218	0.8297	0.7966	0.8657	GradientBoostingClassifier()
0.7944	0.8025	0.7743	0.8328	DecisionTreeClassifier()
0.8120	0.8205	0.7872	0.8568	MLPClassifier()
0.8093	0.8230	0.7699	0.8839	SVC()
0.7870	0.7922	0.7758	0.8092	KNeighborsClassifier()

QUESTIONS:

- **The dataset is highly skewed because of the substantially asymmetric (2:1) class distribution of the training data. The topic at hand is how this data might be applied to infer and evaluate the visualization.**

  We used the k-fold cross validation of data approaches to address this problem. Machine learning models are evaluated on a short data sample using a resampling technique known as k-fold cross validation. We only used a tiny sample from the dataset in order to make sure that the class distribution is balanced and to evaluate how the model/analysis-algorithm is expected to perform generally.

  We repeated this procedure while keeping the same class distribution, using random sampling from the data distribution.

- **What features should be used, and how should the weights for each feature in the dataset be determined?**

  The 14 features in this dataset are composed of six continuous features and the remaining ten categorical traits. Not every feature helps with class prediction. Using data visualization tools, we evaluated the distribution and importance of each attribute separately. A higher score was given to the attributes that had more pronounced differences between the patterns for the two groups of data.

- **What is the process for converting category data into numerical data?**

  In order for machine learning models to function, they cannot be trained on categorical data. As a result, we used one-hot encoding, which allots a number to each distinct category value, to transform the categorical data into numerical data. Categories that make it clear which class each item belongs to received higher scores.

- **Which visualizations should be used for data analysis?**

  Because they accurately depict categorical data, pie charts and mosaic plots (multivariate) were utilized to represent categorical data. Visualizations such as the histogram, box plots, scatter plots, and parallel coordinate plots were employed with continuous data.

- **What methodology should be used for machine learning analysis?**

  The dataset was divided into training and testing sets after categorical data had been transformed into continuous data. It was decided to divide the data in two, with 20% being used for testing and 80% being used for training. After that, we developed a range of algorithms with various underlying categorization theories. We modified the algorithms' hyperparameters to maximize the performance of each strategy.Finally, the test dataset was used to evaluate the effectiveness of each approach.

NOT DOING :

One of our future objectives is to create visual recommenders that employ a query builder to extract the features of a fresh sample or user data before suggesting the class label based on the features. By anticipating class labels based on range set data, class labels can be determined based on range queries. Consider calculating the salary for a person in their mid- to late-35s who puts in 20 to 30 hours per week.

APPENDIX:

UNIVARIATE ANALYSIS PYTHON CODE:

```
import pandas as pd
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv("readonly/adult.txt", header=None, sep=", ")
df.columns = ["age", "workclass", "fnlwgt", "education", "education-num", "marital-status",
"occupation", "relationship", "race", "sex", "capital-gain", "capital-loss", "hours-per-week", "native-
country", "class"]
df = df[df["workclass"] != '?']
df = df[df["education"] != '?']
df = df[df["marital-status"] != '?']
df = df[df["occupation"] != '?']
df = df[df["relationship"] != '?']
df = df[df["race"] != '?']
df = df[df["sex"] != '?']
df = df[df["native-country"] != '?']

below = df[df["class"] == "<=50K"]
above = df[df["class"] == ">50K"]
print("Count(Above 50K) = " + str(len(above.index)))
print("Count(Below 50K) = " + str(len(below.index)))

df.head()

def analyze_numerical_data(column):
    above_50k = list(above[column])
    below_50k = list(below[column])

    print(column)
    print()
    print("Mean")
    print("Above 50K = " + str(np.mean(above_50k)))
    print("Below 50K = " + str(np.mean(below_50k)))
    print()
    print("Median")
    print("Above 50K = " + str(np.median(above_50k)))
    print("Below 50K = " + str(np.median(below_50k)))
    print()
    print("Standard Deviation")
```

```
    print("Above 50K = " + str(np.std(above_50k)))
    print("Below 50K = " + str(np.std(below_50k)))

    plt.close()
    fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(10,10))
    fig.subplots_adjust(hspace=.5)

    axes[0, 0].boxplot(above_50k)
    axes[0, 0].set_title(">50K")
    axes[0, 0].set_xlabel(column)

    axes[0, 1].boxplot(below_50k)
    axes[0, 1].set_title("<=50K")
    axes[0, 1].set_xlabel(column)

    axes[1, 0].hist(above_50k)
    axes[1, 0].set_title(">50K")
    axes[1, 0].set_xlabel(column)

    axes[1, 1].hist(below_50k)
    axes[1, 1].set_title("<=50K")
    axes[1, 1].set_xlabel(column)

    plt.show()


def analyze_categorical_data(column):
    above_50k = Counter(above[column])
    below_50k = Counter(below[column])

    print(column)
    print()
    plt.close()
    fig, axes = plt.subplots(ncols=1, nrows=2, figsize=(5,10))
    axes[0].pie(above_50k.values(), labels=above_50k.keys(), autopct='%1.0f%%')
    axes[0].set_title(">50K")
    axes[1].pie(below_50k.values(), labels=below_50k.keys(), autopct='%1.0f%%')
    axes[1].set_title("<=50K")
    plt.show()


def analyze_per_unique_value(column):
    unique_values = df[column].unique()
    plt.close()
```

```
    fig, axes = plt.subplots(ncols=1, nrows=len(unique_values), figsize=(5,5 * len(unique_values)))

    for i, val in enumerate(unique_values):
        val_df = df[df[column] == val]
        above_50k = val_df[val_df["class"] == ">50K"]
        below_50k = val_df[val_df["class"] == "<=50K"]
        axes[i].pie([len(below_50k.index), len(above_50k.index)], labels=["<=50K (Count-" +
str(len(below_50k.index)) + ")", ">50K (Count-" + str(len(above_50k.index)) + ")"],
autopct='%1.0f%%')
        axes[i].set_title(val)

    plt.show()

analyze_numerical_data("capital-gain")
analyze_numerical_data("capital-loss")
analyze_numerical_data("hours-per-week")
analyze_categorical_data("native-country")
print()
print()
analyze_per_unique_value("native-country")
analyze_categorical_data("sex")
print()
print()
analyze_per_unique_value("sex")
analyze_categorical_data("race")
print()
print()
analyze_per_unique_value("race")
analyze_categorical_data("relationship")
print()
print()
analyze_per_unique_value("relationship")
analyze_categorical_data("occupation")
print()
print()
analyze_per_unique_value("occupation")
analyze_categorical_data("marital-status")
print()
print()
analyze_per_unique_value("marital-status")
analyze_numerical_data("education-num")
analyze_categorical_data("education")
print()
print()
```

```
analyze_per_unique_value("education")
analyze_numerical_data("fnlwgt")
analyze_numerical_data("age")
```

MULTIVARIATE ANALYSIS PYTHON CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy
from statsmodels.graphics.mosaicplot import mosaic
from sklearn.preprocessing import MinMaxScaler
df = pd.read_csv('./dataset/adult.data.txt', sep=", ", header=None)
df.columns = ["age", "workclass", "fnlwgt", "education", "education-num", "marital-status",
"occupation", "relationship", \
        "race", "sex", "capital-gain", "capital-loss", "hours-per-week", "native-country", "salary-
range"]
df = df[df["workclass"] != '?']
df = df[df["education"] != '?']
df = df[df["marital-status"] != '?']
df = df[df["occupation"] != '?']
df = df[df["relationship"] != '?']
df = df[df["race"] != '?']
df = df[df["sex"] != '?']
df = df[df["native-country"] != '?']
below_50K = df[df["salary-range"] == "<=50K"].sample(n=7841)
above_50K = df[df["salary-range"] == ">50K"]

df = pd.concat([above_50K, below_50K])
df['class'] = (df["salary-range"] == ">50K")*1

def plot_scatter_plot(column1, column2, column3):
    plt.close()
    fig, axes = plt.subplots(ncols=2, nrows=3, figsize=(10,10))
    fig.subplots_adjust(hspace=.5)

    x = below_50K[column1]
    y = below_50K[column2]
    axes[0, 0].scatter(x,y)
    axes[0, 0].set_title("<=50K")
    axes[0, 0].set_xlabel(column1)
    axes[0, 0].set_ylabel(column2)

    x = above_50K[column1]
    y = above_50K[column2]
```

```
    axes[0, 1].scatter(x,y)
    axes[0, 1].set_title(">50K")
    axes[0, 1].set_xlabel(column1)
    axes[0, 1].set_ylabel(column2)

    x = below_50K[column2]
    y = below_50K[column3]
    axes[1, 0].scatter(x,y)
    axes[1, 0].set_title("<=50K")
    axes[1, 0].set_xlabel(column2)
    axes[1, 0].set_ylabel(column3)

    x = above_50K[column2]
    y = above_50K[column3]
    axes[1, 1].scatter(x,y)
    axes[1, 1].set_title(">50K")
    axes[1, 1].set_xlabel(column2)
    axes[1, 1].set_ylabel(column3)

    x = below_50K[column3]
    y = below_50K[column1]
    axes[2, 0].scatter(x,y)
    axes[2, 0].set_title("<=50K")
    axes[2, 0].set_xlabel(column3)
    axes[2, 0].set_ylabel(column1)

    x = above_50K[column3]
    y = above_50K[column1]
    axes[2, 1].scatter(x,y)
    axes[2, 1].set_title(">50K")
    axes[2, 1].set_xlabel(column3)
    axes[2, 1].set_ylabel(column1)

    plt.show()


plot_scatter_plot('capital-gain', 'age', 'hours-per-week')

def plot_scatter_plot_diff(column1, column2, column3):
    plt.close()
    fig, axes = plt.subplots(ncols=1, nrows=3, figsize=(10,30))
    fig.subplots_adjust(hspace=.5)

    colors = df['class']
```

```
    x = df[column1]
    y = df[column2]
    axes[0].scatter(x,y,c=colors)
    axes[0].set_title("purple <=50K, yellow >50K")
    axes[0].set_xlabel(column1)
    axes[0].set_ylabel(column2)

    x = df[column2]
    y = df[column3]
    axes[1].scatter(x,y,c=colors)
    axes[1].set_title("purple <=50K, yellow >50K")
    axes[1].set_xlabel(column2)
    axes[1].set_ylabel(column3)

    x = df[column3]
    y = df[column1]
    axes[2].scatter(x,y,c=colors)
    axes[2].set_title("purple <=50K, yellow >50K")
    axes[2].set_xlabel(column3)
    axes[2].set_ylabel(column1)

    plt.show()

plot_scatter_plot_diff('capital-gain', 'age', 'hours-per-week')
def plot_scatter_matrix_below50K(column1, column2, column3):
    plt.close()
    fig, axes = plt.subplots(ncols=1, nrows=1, figsize=(15,11))
    fig.subplots_adjust(hspace=.5)
    df_below_sm = below_50K[[column1, column2, column3]]
    print("Salary <=50K")
    pd.plotting.scatter_matrix(df_below_sm, ax=axes)
    plt.show()

plot_scatter_matrix_below50K('capital-gain', 'age', 'hours-per-week')
def plot_scatter_matrix_above50K(column1, column2, column3):
    plt.close()
    fig, axes = plt.subplots(ncols=1, nrows=1, figsize=(15,11))
    fig.subplots_adjust(hspace=.5)
    df_above_sm = above_50K[[column1, column2, column3]]
    print("Salary >50K")
    pd.plotting.scatter_matrix(df_above_sm, ax=axes)
    plt.show()
plot_scatter_matrix_above50K('capital-gain', 'age', 'hours-per-week')
plt.close()
```

```
fig, axes = plt.subplots(ncols=1, nrows=1, figsize=(15,8))
fig.subplots_adjust(hspace=.5)
mosaic(df, ['occupation', 'salary-range'], ax=axes, axes_label=False)
plt.show()
plt.close()
fig, axes = plt.subplots(ncols=1, nrows=1, figsize=(15,8))
fig.subplots_adjust(hspace=.5)
mosaic(df, ['marital-status', 'salary-range'], ax=axes, axes_label=False)
plt.show()
frame_pc = df[['education-num', 'age', 'capital-gain', 'class']].copy()
frame_np_array = MinMaxScaler().fit_transform(frame_pc.values)
frame_pc = pd.DataFrame(frame_np_array)
df.index = frame_pc.index
frame_pc['salary-range'] = df['salary-range']
frame_pc.columns = ['capital-gain', 'education-num', 'age', 'class', 'salary-range']
frame_pc_below_50K = frame_pc[frame_pc["class"] == 0.0].sample(n=30)
frame_pc_above_50K = frame_pc[frame_pc["class"] == 1.0].sample(n=30)
frame_pc = pd.concat([frame_pc_below_50K, frame_pc_above_50K])
pd.plotting.parallel_coordinates(frame_pc, 'salary-range', cols=['education-num', 'age', 'capital-
gain'], color=('#FF0000',
                    '#FFD700'))
plt.show()
def plot_mosaic_class(column1, column2, column3):
    plt.close()
    cols = [column1, column2, column3]
    for i in range(3):
        for j in range(i+1, 3):
            print("Salary <=50K")
            mosaic(below_50K, [cols[i], cols[j]])
            plt.show()
            print("Salary >50K")
            mosaic(above_50K, [cols[i], cols[j]])
            plt.show()
    for i in range(3):
        print("Salary <=50K")
        mosaic(below_50K, [cols[i]])
        plt.show()
        print("Salary >50K")
        mosaic(above_50K, [cols[i]])
        plt.show()
    for i in range(3):
        mosaic(df, [cols[i], 'salary-range'])
        plt.show()
```

```
plot_mosaic_class('education', 'marital-status', 'occupation')
```

MACHINE LEARNING ANALYSIS CODE:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
AdaBoostClassifier, BaggingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import warnings

warnings.filterwarnings("ignore") # to ignore warnings being printed on the console


def get_workclass_value(x):
    if x == "Without-pay":
        return 7
    elif x == "Private":
        return 6
    elif x == "State-gov":
        return 5
    elif x == "Self-emp-not-inc":
        return 4
    elif x == "Local-gov":
        return 3
    elif x == "Federal-gov":
        return 2
    else:
        return 1


def get_marital_status_value(x):
    if x == "Never-married":
        return 7
    elif x == "Separated":
        return 6
```

```
    elif x == "Married-spouse-absent":
        return 5
    elif x == "Widowed":
        return 4
    elif x == "Divorced":
        return 3
    elif x == "Married-civ-spouse":
        return 2
    else:
        return 1


def get_occupation_value(x):
    if x in ["Exec-managerial", "Prof-specialty", "Protective-serv"]:
        return 1
    elif x in ["Sales", "Transport-moving", "Tech-support", "Craft-repair"]:
        return 2
    else:
        return 3


def get_relationship_value(x):
    if x == "Own-child":
        return 6
    elif x == "Other-relative":
        return 5
    elif x == "Unmarried":
        return 4
    elif x == "Not-in-family":
        return 3
    elif x == "Husband":
        return 2
    else:
        return 1


def get_race_value(x):
    if x == "Other":
        return 5
    elif x == "Amer-Indian-Eskimo":
        return 4
    elif x == "Black":
        return 3
    elif x == "White":
```

```
        return 2
    else:
        return 1


def get_sex_value(x):
    if x == "Male":
        return 2
    else:
        return 1


def get_native_country_value(x):
    if x in ["United-States", "Cuba", "Poland", "Thailand", "Ecuador", "China", "South", "Scotland",
"Greece", "Ireland", "Hungary"]:
        return 2
    elif x in ["India", "England", "Canada", "Germany", "Iran", "Philippines", "Cambodia", "Taiwan",
"France", "Italy", "Japan", "Yugoslavia", "Hong"]:
        return 1
    else:
        return 3


def get_class_value(x):
    if x == ">50K":
        return 1
    else:
        return 0


temp_df1 = pd.read_csv("readonly/adult.txt", header=None, sep=", ")
temp_df2 = pd.read_csv("readonly/testdata.txt", header=None, sep=", ")
df = pd.concat([temp_df1, temp_df2]).sample(frac=1)
df.columns = ["age", "workclass", "fnlwgt", "education", "education-num", "marital-status",
"occupation", "relationship", "race", "sex", "capital-gain", "capital-loss", "hours-per-week", "native-
country", "class"]
df["class"] = df["class"].apply(lambda x: x.replace(".", ""))
df = df[df["workclass"] != '?']
df = df[df["education"] != '?']
df = df[df["marital-status"] != '?']
df = df[df["occupation"] != '?']
df = df[df["relationship"] != '?']
df = df[df["race"] != '?']
df = df[df["sex"] != '?']
```

```
df = df[df["native-country"] != '?']

below = df[df["class"] == "<=50K"].sample(n=11208)
above = df[df["class"] == ">50K"]

train_data = pd.concat([above, below]).sample(frac=1)
train_data = train_data.drop("capital-loss", axis=1)
train_data = train_data.drop("education", axis=1)
train_data = train_data.drop("fnlwgt", axis=1)
train_data['workclass'] = train_data['workclass'].apply(get_workclass_value)
train_data['marital-status'] = train_data['marital-status'].apply(get_marital_status_value)
train_data['occupation'] = train_data['occupation'].apply(get_occupation_value)
train_data['relationship'] = train_data['relationship'].apply(get_relationship_value)
train_data['race'] = train_data['race'].apply(get_race_value)
train_data['sex'] = train_data['sex'].apply(get_sex_value)
train_data['native-country'] = train_data['native-country'].apply(get_native_country_value)
train_data['class'] = train_data['class'].apply(get_class_value)
train_data.head()

feature_matrix_df = train_data.iloc[:, :-1]
labels_df = train_data.iloc[:, -1]
feature_matrix = feature_matrix_df.values
labels = labels_df.values

train_data, test_data, train_labels, test_labels = train_test_split(feature_matrix, labels,
test_size=0.2, random_state=42)

transformed_train_data = MinMaxScaler().fit_transform(train_data)
transformed_test_data = MinMaxScaler().fit_transform(test_data)

transformed_train_data[:5,:]

def train_model(model_name, Model, tuned_parameters, X_train, y_train, fold_num=10,
scoring_function="f1_macro",
                useRandomizedSearch=True):
    if useRandomizedSearch == False:
        clf = GridSearchCV(Model, tuned_parameters, cv=fold_num, scoring=scoring_function)
    else:
        clf = RandomizedSearchCV(Model, tuned_parameters, cv=fold_num,
scoring=scoring_function)
    clf.fit(X_train, y_train)

    print("Best parameters set found:")
    print(clf.best_params_)
```

```
    print("Best score: %0.3f" % clf.best_score_)
    print("Scores on training set:")
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    meanMap = {}
    stdMap = {}
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        meanMap[str(params)] = mean
        stdMap[str(params)] = std
    for params in sorted(meanMap, key=meanMap.get, reverse=True):
        print("%0.3f (+/-%0.03f) for %r" % (meanMap[params], stdMap[params] * 2, params))

    return clf.best_params_

models_dict = {}
print("\n\nTRAINING DATA RESULTS:")
print("\n\nBaggingClassifier Classifier")
tuned_parameters = {'random_state': [10],
                    'n_estimators': np.arange(10, 41, 4),
                    'max_samples': [0.7, 0.8],
                    'max_features': [0.7, 0.8],
                    'oob_score': [True, False]}
models_dict["BaggingClassifier()"] = train_model("BaggingClassifier", BaggingClassifier(),
                                    tuned_parameters, transformed_train_data, train_labels,
                                    useRandomizedSearch=True)

print("\n\nMultinomialNB Classifier")
tuned_parameters = {'fit_prior': [True, False],
                    'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 2]}
models_dict["MultinomialNB()"] = train_model("MultinomialNB", MultinomialNB(),
tuned_parameters,
                                   transformed_train_data, train_labels, useRandomizedSearch=True)

print("\n\nLogisticRegression Classifier")
tuned_parameters = {'random_state': [10],
                    'C': [0.01, 0.1, 1, 0.001],
                    'fit_intercept': [True, False],
                    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
                    'warm_start': [True, False]}
models_dict["LogisticRegression()"] = train_model("LogisticRegression", LogisticRegression(),
tuned_parameters,
                                     transformed_train_data, train_labels, useRandomizedSearch=True)

print("\n\nAdaBoostClassifier Classifier")
```

```
tuned_parameters = {'random_state': [10],
                    'n_estimators': range(10, 41, 4),
                    'learning_rate': [0.001, 0.01, 0.1, 1],
                    'algorithm': ['SAMME.R', 'SAMME']}
models_dict["AdaBoostClassifier()"] = train_model("AdaBoostClassifier", AdaBoostClassifier(),
tuned_parameters,
                                  transformed_train_data, train_labels, useRandomizedSearch=True)

print("\n\nRandomForestClassifier Classifier")
tuned_parameters = {'random_state': [10],
                    'n_estimators': np.arange(10, 41, 4),
                    'max_depth': [10, 20, 30, 40],
                    'min_samples_split': [2, 3, 5, 10],
                    'warm_start': [True, False],
                    'min_samples_leaf': [1, 2, 4]}
models_dict["RandomForestClassifier()"] = train_model("RandomForestClassifier",
RandomForestClassifier(),
                                    tuned_parameters, transformed_train_data, train_labels,
                                    useRandomizedSearch=True)

print("\n\nGradientBoostingClassifier Classifier")
tuned_parameters = {'random_state': [10],
                    'n_estimators': range(10, 41, 4),
                    'max_depth': range(3, 8),
                    'min_samples_split': range(2, 5),
                    'min_samples_leaf': range(40, 60, 4),
                    'max_features': range(5, 10),
                    'subsample': [0.6, 0.7, 0.75, 0.8, 0.85, 0.9]}
models_dict["GradientBoostingClassifier()"] = train_model("GradientBoostingClassifier",
                                      GradientBoostingClassifier(), tuned_parameters,
                                      transformed_train_data, train_labels,
                                      useRandomizedSearch=True)
print("\n\nDecisionTreeClassifier Classifier")
tuned_parameters = {'random_state': [10],
                    'max_depth': range(10, 30, 3),
                    'min_samples_split': [2, 5, 10],
                    'min_samples_leaf': [1, 2, 3, 4]}
models_dict["DecisionTreeClassifier()"] = train_model("DecisionTreeClassifier",
DecisionTreeClassifier(),
                                    tuned_parameters, transformed_train_data, train_labels,
                                    useRandomizedSearch=True)

print("\n\nMLPClassifier Classifier")
tuned_parameters = {'random_state': [10],
```

```
            'max_iter': range(200, 500, 100),
            'hidden_layer_sizes': [(40, 40, 40), (50, 50, ), (70, 50, 30,), (70, 30,), (30,)],
            'activation': ['relu', 'identity', 'logistic', 'tanh'],
            'solver': ['lbfgs', 'sgd', 'adam'],
            'shuffle': [True, False],
            'learning_rate_init': [0.001, 0.01, 0.1],
            'learning_rate': ['constant', 'invscaling', 'adaptive']}
models_dict["MLPClassifier()"] = train_model("MLPClassifier", MLPClassifier(), tuned_parameters,
                             transformed_train_data, train_labels, useRandomizedSearch=True)

print("\n\nSVC Classifier")
tuned_parameters = {'random_state': [10],
            'kernel': ['rbf', 'linear', 'poly', 'sigmoid'],
            'degree': np.arange(2, 20),
            'C': np.arange(1, 10)}
models_dict["SVC()"] = train_model("SVC", SVC(), tuned_parameters, transformed_train_data,
train_labels,
                       useRandomizedSearch=True)

print("\n\nKNeighborsClassifier Classifier")
tuned_parameters = {'n_neighbors': np.arange(1, 31),
            'weights': ["uniform", "distance"],
            'p': np.arange(1, 6),
            'metric': ["minkowski", "braycurtis", "canberra", "matching", "dice", "kulsinski",
                   "rogerstanimoto", "russellrao", "sokalmichener", "sokalsneath"]}
models_dict["KNeighborsClassifier()"] = train_model("KNeighborsClassifier", KNeighborsClassifier(),
                                tuned_parameters, transformed_train_data, train_labels,
                                useRandomizedSearch=True)


def run_model(name, model, X_train, X_test, y_train, y_test):
   model.fit(X_train, y_train)
   y_predict = model.predict(X_test)
#    print()
#    try:
#        print("Name: " + name + "\nFeature Importance:")
#        print(model.feature_importances_)
#    except:
#        print("Feature Importance missing.")
   return accuracy_score(y_test, y_predict), f1_score(y_test, y_predict), precision_score(y_test,
                                                        y_predict), recall_score(
       y_test, y_predict)
```

```
def run_test(models_dict, X_train, X_test, y_train, y_test):
    print("Acc\tF1\tPrec\tRecall\tModel")
    for name in models_dict.keys():
        model = eval(name)
        model.set_params(**models_dict[name])
        acc, f1, prec, rec = run_model(name, model, X_train, X_test, y_train, y_test)
        print("%.4f\t%.4f\t%.4f\t%.4f\t%s" % (acc, f1, prec, rec, name))

print("\n\n\nTESTING DATA RESULTS:\n\n")
run_test(models_dict, transformed_train_data, transformed_test_data, train_labels, test_labels)
```