University of Salford MANCHESTER

Assessment:
Machine Learning and Data Mining
(MSc Data Science)

By:

NNAEMEKA NDUBUISI
@00738283

08/12/2023

Table of Contents

Task 1:

Predicting Income Levels Using Machine Learning models and algorithms: Exploration of Demographics and Factors Influencing Higher Income

Introduction:

Machine learning (ML) is defined as a discipline of artificial intelligence (AI) that provides machines the ability to automatically learn from data and past experiences to identify patterns and make predictions with minimal human intervention. Machine learning is a field that deals with developing computer systems capable of learning and enhancing their performance based on experiences. These systems can perform tasks or make predictions by identifying patterns and relationships in a given dataset. In this assessment, we'll utilize machine learning techniques to analyze a dataset and build predictive models that can classify individuals' income levels based on their demographic and financial attributes.

The purpose of this project is to analyze the relationship between different demographic and financial indicators and the probability of an individual earning an annual income exceeding $50,000. To achieve this, a machine learning approach has been utilized, which involves data mining techniques. This subfield of artificial intelligence involves algorithms that can learn from data without explicit programming, as explained by Jordan and Mitchell (2015).

Motivated by the need to identify factors contributing to higher income levels, this assessment is significant for individuals, policymakers, and organizations. By applying machine learning, we aim to reveal insights into the interplay between demographics, educational background, occupation, and income, which can guide career and educational decisions, as well as policy-making aimed at reducing income inequality (Atkinson, Rainwater, & Smeeding, 1995).

The methods of this process includes data preprocessing for optimal model training, exploratory data analysis and employing models like K-Nearest Neighbors (KNN) and Random Forest. KNN works by classifying new instances based on their proximity to neighbors (Cover & Hart, 1967), while Random Forest, an ensemble of decision trees, provides robustness against overfitting and handles various feature types effectively

(Breiman, 2001). The models are evaluated using metrics like accuracy, precision, and recall.

Dataset:

the dataset used for this classification analysis is the "Census Income" dataset obtained from the UCI Machine Learning Repository. It is a multivariate dataset primarily focused on social science and classification tasks. The dataset was donated by Barry Becker on April 30, 1996, and was extracted from the 1994 Census database. Becker selected a subset of reasonably clean records based on specific conditions related to age, income, final weight, and hours worked per week.

The dataset contains 48,842 instances with 14 features, spanning across 6 continuous, 8 nominal attributes. The independent variables include: age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, and native-country. The dependent variable is the income level, classified as ">50K" or "<=50K." The table below describes the data's variables, attributes, data types and a brief description.

S/N	Variable Name	Data Type	Attributes	Brief Description
1	age	Continuous	-	Age of the individual
2	workclass	Categorical	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked	Type of workclass
3	fnlwgt	Continuous	-	Final weight assigned to observations (sampling weight)
4	education	Categorical	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool	Highest level of education achieved
5	education-num	Continuous	-	Numeric representation of education level (years of education)

6	marital-status	Categorical	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse	Marital status of the individual
7	occupation	Categorical	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	Occupation of the individual
8	relationship	Categorical	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried	Relationship status of the individual
9	race	Categorical	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black	Race of the individual
10	sex	Binary	Female, Male	Gender of the individual
11	capital-gain	Continuous	-	Capital gains received by the individual
12	capital-loss	Continuous	-	Capital losses incurred by the individual
13	hours-per-week	Continuous	-	Number of hours worked per week
14	native-country	Categorical	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands	Native country of the individual
15	class	Categorical	>50K, <=50K	Income level (target variable)

This project focuses on leveraging data mining techniques to explore factors influencing higher income, revealing demographic patterns, and assessing the impacts of education

and occupation. The main objective is to predict whether an individual's annual income exceeds $50,000. The dataset includes demographic variables like age, education, gender, occupation, and financial indicators such as capital gain and loss. The project aims to experiment with K-Nearest Neighbors (KNN) and Random Forest classification models to forecast income levels, facilitating informed decision-making and predictive model development. Details on dataset demographics and indicators can be found in the readme.txt file(xxxxxxxxxxxxxcxx)

To access the dataset and learn more about its specific attributes and metadata, you can visit the following link: [Census Income - UCI Machine Learning Repository] (https://archive.ics.uci.edu/dataset/20/census+income).

Explanation and Preparation

This report will extensively cover all procedures in performing a classification algorithm using Python programming language.

Loading libraries:

The dataset for this project has been uploaded to the Jupyter working directory and will be referred to as "dataset." After importing the necessary libraries, the data can be read using the code, allowing for easy access and analysis. By consistently referring to the dataset as "dataset," it simplifies data manipulation and facilitates the application of data mining techniques, such as exploratory analysis, feature engineering, and model training.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as ask
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
#load dataset
dataset = pd.read_csv('adult.csv')
```

Using the pandas data frame, the dataset was briefly skimmed through as a usual practice, to help understanding the data set. This provided evidence to show that data contains missing values.

In [5]:
```
dataset.head()
```

Out[5]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K

In [6]:
```
dataset.shape
```

Out[6]:
```
(48842, 15)
```

In [7]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              48842 non-null  int64
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64
 11  capital-loss     48842 non-null  int64
 12  hours-per-week   48842 non-null  int64
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

Cleaning Data

While scrutinizing the dataset, it was observed that missing data were represented by the symbol "?". In lay terms, this signifies an information that is not provided. However, Python scripts does not recognize "?" as a placeholder for missing values.

To address this, it is crucial to convert "?" to a value that the programming language interprets as a missing value. Consequently, the decision is to replace "?" with "NaN" to facilitate proper handling and recognition of missing values in the dataset. The missing values attributed to nearly 7% of the entire data.

In [8]:
```
# Replacing the "?" with "Nan"
dataset.replace('?', np.nan, inplace=True)
```

In [9]:
```
# Finding out colunmns with missing values
cols_with_missing_values = dataset.columns[dataset.isnull().any()].tolist()
missing_values_count = dataset[cols_with_missing_values].isnull().sum()
print("Missing values count per column:")
print(missing_values_count)
```

```
Missing values count per column:
workclass         2799
occupation        2809
native-country     857
dtype: int64
```

To handle missing values in the dataset and remove the “NaN” values, a preprocessing step was performed by replacing the missing values with the mode, which is the most frequently occurring value in each categorical variable. This approach was chosen because the missing values in the dataset are categorical variables, and using the mean value would not be appropriate.

By replacing missing values with the mode, it ensures that the imputed values are representative of the most common category in each variable. This approach aligns with common sense reasoning and maintains the integrity of the categorical variables in the dataset.

In [10]:
```
# Replacing mising values in the 'workclass' with the mode
dataset['workclass'].fillna(dataset['workclass'].mode()[0], inplace=True)

# Replacing mising values in the 'occupation' with the mode
dataset['occupation'].fillna(dataset['occupation'].mode()[0], inplace=True)

# Replacing mising values in the 'native-country' with the mode
dataset['native-country'].fillna(dataset['native-country'].mode()[0], inplace=True)
```

Within the dataset, there are values across columns that are either synonymous or represent conjugates of each other. A recommended practice is to group such values together under a common identifier and assign a Unified name to them. This simplification process contributes to streamlining our dataset for clearer representation and analysis. The columns “workclass”, “education” and “martial-status” where consider to be of major interest for this step.

In [10]:
```
self_employed = ['Self-emp-not-inc', 'Self-emp-inc']
unemployed = ['Without-pay', 'Never-worked']
govt_employed = ['Local-gov', 'Federal-gov', 'State-gov']

dataset['workclass'].replace(to_replace = self_employed, value = 'self_employed', inplace = True)
dataset['workclass'].replace(to_replace = unemployed, value = 'unemployed', inplace = True)
dataset['workclass'].replace(to_replace = govt_employed, value = 'govt_employed', inplace = True)
dataset['workclass'].value_counts()
```

Out[10]:
```
workclass
Private          36705
govt_employed     6549
self_employed     5557
unemployed          31
Name: count, dtype: int64
```

In [12]:
```
basic_edu = ['11th', '7th-8th', '5th-6th', '9th', '10th', '12th', '1st-4th']
college_edu = ['Assoc-acdm', 'Some-college', 'Assoc-voc']
graduates = ['Bachelors', 'Masters']
higher_edu = ['Doctorate', 'Prof-school']

dataset['education'].replace(to_replace = basic_edu, value = 'basic_edu', inplace = True)
dataset['education'].replace(to_replace = college_edu, value = 'college_edu', inplace = True)
dataset['education'].replace(to_replace = graduates, value = 'graduates', inplace = True)
dataset['education'].replace(to_replace = higher_edu, value = 'higher_edu', inplace = True)
dataset['education'].value_counts()
```

Out[12]:
```
education
HS-grad        15784
college_edu    14540
graduates      10682
basic_edu       6325
higher_edu      1428
Preschool         83
Name: count, dtype: int64
```

In [11]:
```
married = ['Married-civ-spouse','Married-spouse-absent','Married-AF-spouse']
single = ['Never-married']

dataset['marital-status'].replace(to_replace = married, value = 'married', inplace = True)
dataset['marital-status'].replace(to_replace = single, value = 'single', inplace = True)
dataset['marital-status'].value_counts()
```

Out[11]:
```
marital-status
married      23044
single       16117
Divorced      6633
Separated     1530
Widowed       1518
Name: count, dtype: int64
```

Furthermore, upon close examination of the dataset, it becomes apparent that the "education-num" column represents the numeric encoding of individuals' education levels. Similarly, there seems to be a correlation between the "relationship" column and the individuals' "marital-status." Considering our project's objective, it is determined that these columns do not provide significant advantages and can be dropped from the dataset. Removing these columns streamlines the dataset and focuses on the relevant variables for predicting income levels accurately.

In [13]:
```
dataset.drop(columns=['educational-num', 'relationship'], inplace=True)
```

In [15]:
```
dataset.head(8)
```

Out[15]:

	age	workclass	fnlwgt	education	marital-status	occupation	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	basic_edu	single	Machine-op-inspct	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	married	Farming-fishing	White	Male	0	0	50	United-States	<=50K
2	28	govt_employed	336951	college_edu	married	Protective-serv	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	college_edu	married	Machine-op-inspct	Black	Male	7688	0	40	United-States	>50K
4	18	Private	103497	college_edu	single	Prof-specialty	White	Female	0	0	30	United-States	<=50K
5	34	Private	198693	basic_edu	single	Other-service	White	Male	0	0	30	United-States	<=50K
6	29	Private	227026	HS-grad	single	Prof-specialty	Black	Male	0	0	40	United-States	<=50K
7	63	self_employed	104626	higher_edu	married	Prof-specialty	White	Male	3103	0	32	United-States	>50K

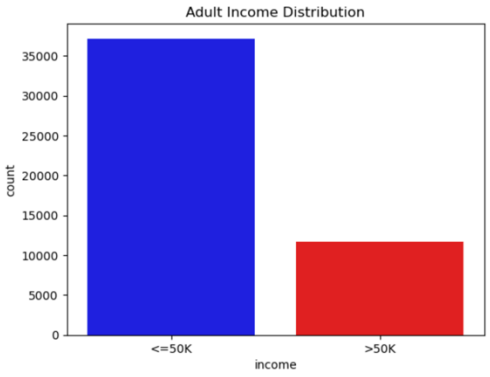
Based on the analysis of the dataset so far, it can be concluded that the data is now clean and prepared for further usage. The preprocessing steps have contributed to completing the data and making it suitable for analysis and modeling. This process ensures a more accurate representation of the data and eliminate potential biases or distortions that may arise from excluding instances with missing values. However, it's important to note that the data may still undergo additional cleaning if further observations are made during the exploratory data analysis (EDA) phase. EDA can reveal additional insights or patterns that might require additional data cleaning steps to enhance the dataset's quality and reliability.

Exploratory data analysis (EDA)

As a usual practice dataset from was subjected to exploratory data analysis (EDA) to gain insights into the data and uncover potential patterns and relationships. The EDA of the dataset revealed a significant imbalance in the income distribution. The value count

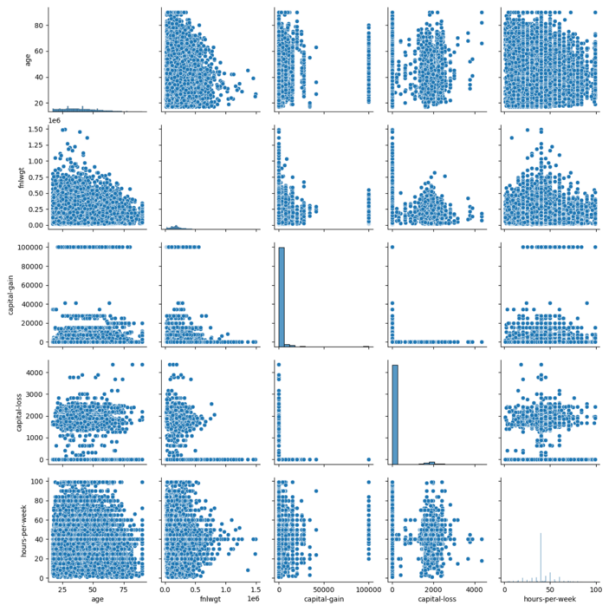
analysis showed that approximately 76% of individuals fell into the income class of "<=50K," while only 24% belonged to the ">50K" income category. This observation was effectively visualized using a countplot, which clearly displayed the disproportionate representation of the two income classes through the use of blue and red bars. The countplot emphasized the dataset's imbalanced nature, with a substantially higher number of individuals having incomes below or equal to $50K/yr compared to those earning above $50K/yr, this observation will be fixed later on within the project.

```
In [16]: dataset.income.value_counts(normalize=True)*100

Out[16]: income
         <=50K    76.071823
         >50K     23.928177
         Name: proportion, dtype: float64

In [17]: sns.countplot(data=dataset, x='income', palette=['blue', 'red'])
         plt.title('Adult Income Distribution')
         plt.show()
```

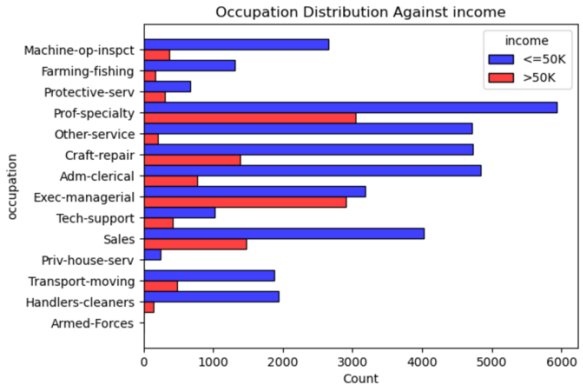


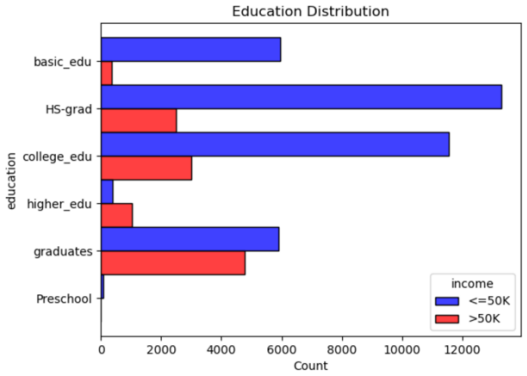
A pairplot was utilized to explore the relationships between different features in the dataset. The pairplot revealed scatter plots for each pair of numerical features, providing insights into potential correlations and trends.

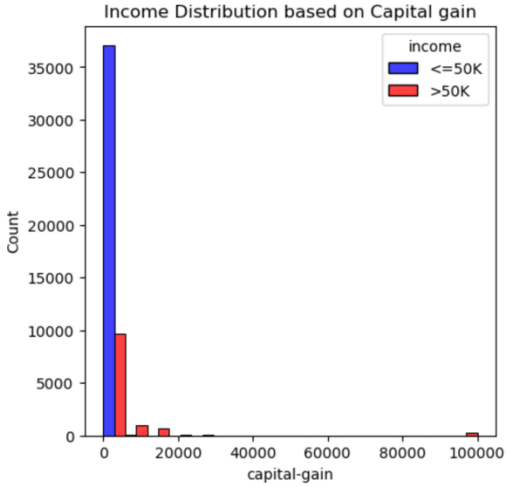
```
In [18]: sns.pairplot(dataset, palette='blue')
         plt.show()
```

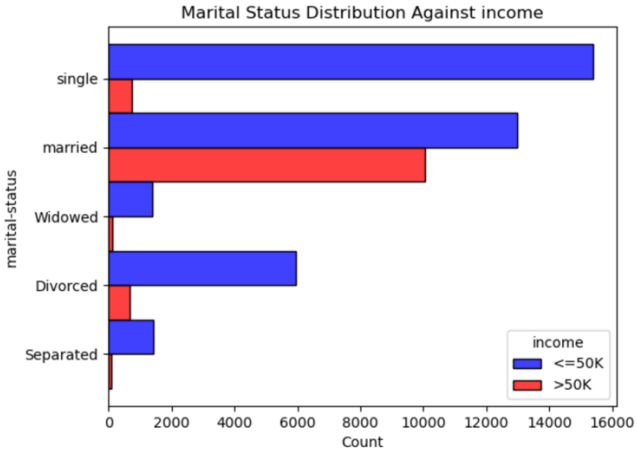


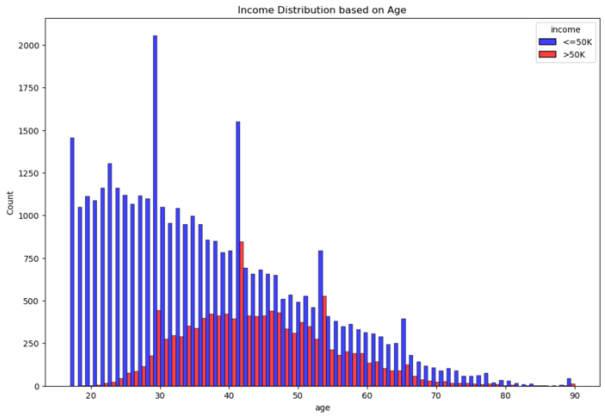
Most importantly a series of histogram including a plot for Age distribution, Income Distribution based on Age, Income Distribution based on Capital gain, Income distribution based on Education, Marital Status Distribution Against income and Occupation Distribution Against income. These illustration provided a background knowledge on the distribution of the income class as shown in the figure below.











A good glance at these the visualizations, reveals that certain variables like age, education, work class, income, and capital gain could potentially play crucial roles in the prediction process. Notably, professionals in specialist and tech-related occupations tend to have higher incomes.

To deepen into the relationships between variables, a correlation matrix was deemed essential for the EDA. However, since some variables are not numeric, a conversion step was necessary to facilitate correlation analysis. This was accomplished by employing LabelEncoder, which assigns specific numeric values to categorical attributes within the selected columns.

In [25]:
```
# Assigning numerical values to the categorical columns
columns_to_encode = ['workclass', 'education', 'marital-status', 'occupation', 'race', 'gender', 'native-country
]

le = LabelEncoder()

# Iterating over each categorical column
for column in columns_to_encode:
    dataset[column] = le.fit_transform(dataset[column])
```

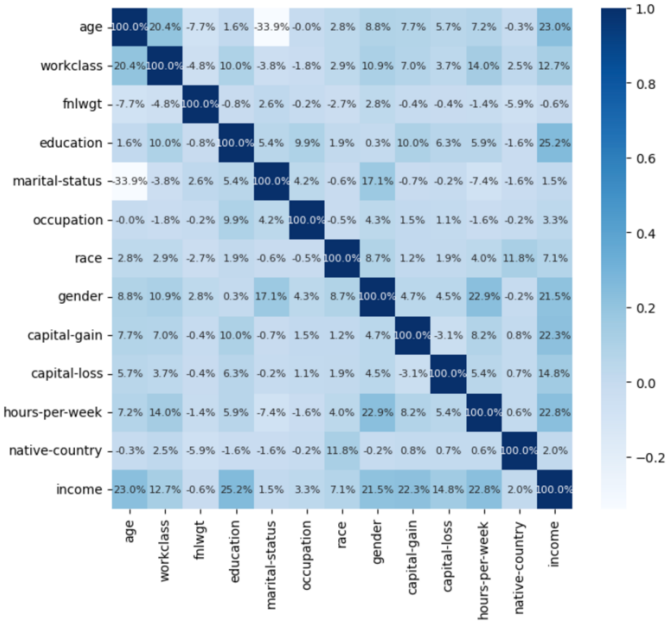
In [26]:
```
dataset.head()
```

Out[26]:

	age	workclass	fnlwgt	education	marital-status	occupation	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	0	226802	2	4	6	2	1	0	0	40	38	0
1	38	0	89814	0	3	4	4	1	0	0	50	38	0
2	28	1	336951	3	3	10	4	1	0	0	40	38	1
3	44	0	160323	3	3	6	2	1	7688	0	40	38	1
4	18	0	103497	3	4	9	4	0	0	0	30	38	0

Results from the correlation shows that education and age have the highest positive relationships with the income class. No negative correlation was observed.

In [27]:
```
plt.figure(figsize=(8,7))
correlation_matrix = dataset.corr()
annot_font_size = 8
# Create a heatmap with reduced font size
sns.heatmap(correlation_matrix, annot=True, cmap='Blues', fmt=".1%", annot_kws={"size": annot_font_size})
plt.show()
```



The correlation matrix revealed enlightening insights. Education and age exhibited the highest positive correlations with the income class. This suggests that higher levels of education and older age groups are more likely to be associated with higher incomes. Notably, no negative correlations were observed, implying that none of the examined variables have a strong negative impact on income.

The EDA findings, including the positive correlations between education and income, and the importance of age and occupation, provide a solid foundation for training and prediction models. These insights will enhance the accuracy and effectiveness of the models in predicting income levels.

Classification

Classification algorithms are widely used in machine learning to predict categorical outcomes based on input variables. As mentioned earlier two popular algorithms for classification namely K-Nearest Neighbors (KNN) and Random Forest will be used within the scope of this task.

K-Nearest Neighbors (KNN) is a simple and intuitive classification algorithm that assigns a class label to a new data point based on its nearest neighbors. It is non-parametric and requires careful selection of the K value. Random Forest, on the other hand, is an ensemble learning algorithm that combines multiple decision trees to make predictions. It handles high-dimensional data well and provides insights into feature importance. Moreover, it can be computationally expensive. The choice between KNN and Random Forest depends on factors such as dataset size, complexity, interpretability, and computational resources available.

To start this task, we will first import the named libraries from scikit-learn library for classification.

In [28]:
```
# Importing libraries set 2
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import f_classif, mutual_info_classif
from sklearn.feature_selection import RFE
from sklearn.metrics import precision_score
from sklearn.inspection import permutation_importance
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

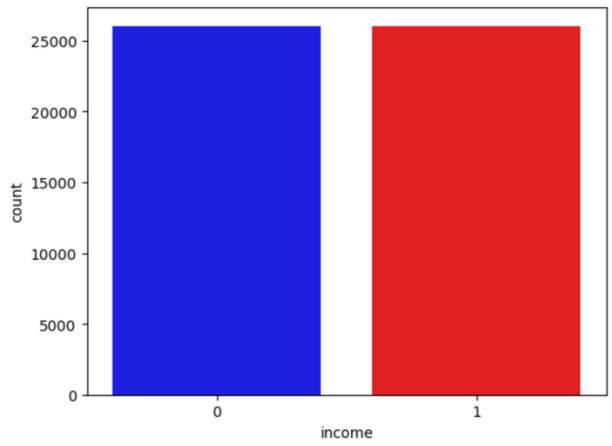
Afterwards, the dataset was initially divided into features and the target variable 'income'. The 'income' column was assigned to the variable 'y' as the target variable, while the remaining columns were assigned to the variable 'X' as the feature set.

To evaluate the model's performance, the dataset was split into training and test sets using the train_test_split function from scikit-learn. The test_size parameter was set to 0.3, allocating 30% of the data for testing and the remaining 70% for training. This division allows for assessing the model's generalization ability on unseen data.

```
In [29]:
### Fitting and Transforming the Dataset

In [30]:
X = dataset.drop('income', axis=1)  # Features
y = dataset['income']  # Target variable

In [31]:
# Splitting the dataset into the Training set and the Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=1)
```

To address the issue of class imbalance in the training data, as observed during the EDA. The SMOTE (Synthetic Minority Over-sampling Technique) algorithm for addressing class imbalance in the training data, the fit_resample() method is applied to X_train and y_train. This method oversamples the minority class by generating synthetic samples from the existing data, equalizing the representation of both classes. By oversampling the minority class, SMOTE helps to create synthetic samples, leading to a more balanced representation of the classes for improved model training and performance.

```
In [31]:
from imblearn.over_sampling import RandomOverSampler

resampler = SMOTE(random_state=0)
X_train, y_train = resampler.fit_resample(X_train,y_train)
sns.countplot(x= y_train, palette=['blue', 'red'])

Out[31]: <Axes: xlabel='income', ylabel='count'>
```



To ensure fair contribution from all features during model training and prevent dominance based on scale differences, the data was standardized. The StandardScaler from scikit-learn was utilized, transforming the features in both the training set (X_train) and test set (X_test) using the fit_transform method. This process scales the features based on their mean and standard deviation.

```
In [32]:
#scaling the data set
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.fit_transform(X_test)
```

Standardizing the data is crucial as it equalizes the importance of features in machine learning algorithms. By eliminating scale bias, the model can make accurate predictions without favoring features with larger scales. Consequently, this preprocessing step promotes fair and reliable model training and evaluation.

K-Nearest Neighbors (KNN)

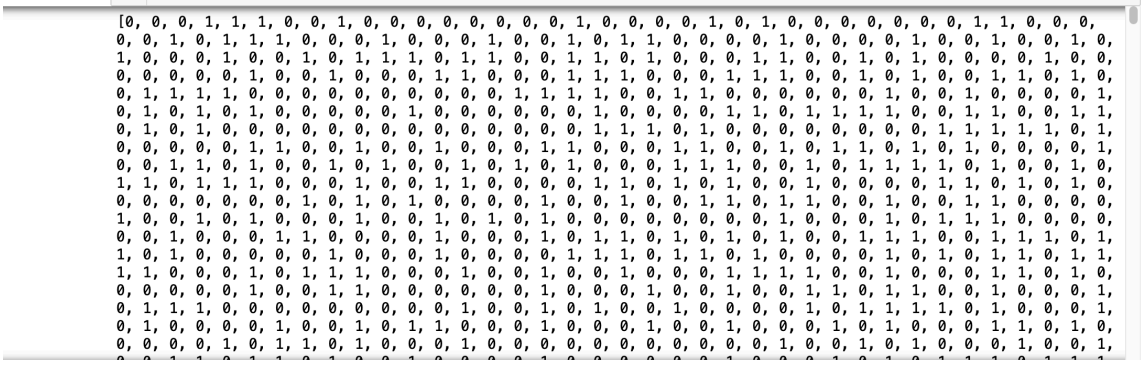
After standardizing the data, the next step is to follow through with implementation of K-Nearest Neighbors (KNN) classification using a specific set of parameters. The KNN classifier was created with five neighbors and utilized the Minkowski distance metric with p=2, representing Euclidean distance.

```
In [33]:
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train_sc, y_train)
```

Out[33]: KNeighborsClassifier
KNeighborsClassifier()

The classifier was trained on the standardized training set, X_train_sc, and the corresponding target variable, y_train, using the fit() method. This step involved finding the nearest neighbors in the feature space and assigning class labels based on majority voting. Next, the trained classifier was used to predict the target variable for the standardized test set, X_test_sc, using the predict() method. The predicted values were stored as y_pred.

```
In [34]:
# Predicting the test set results
y_pred = classifier.predict(X_test_sc)
print(y_pred.tolist())
```



Now, we have done all the necessary steps, the accuracy score will be computed by comparing the predicted values, y_pred, with the actual target values, y_test, using the metrics.accuracy_score() function. The accuracy score quantifies the overall correctness of the classification model. The metrics.classification_report() function will

presents us with the precision, recall, F1-score, and support for each class. These evaluation metrics provide a comprehensive understanding of the classification model's performance, including accuracy, confusion matrix, and detailed classification metrics.

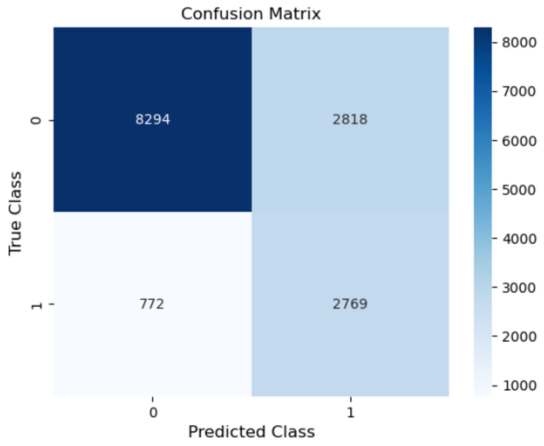
In [36]:
```
acc = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: %.2f\n\n'%(acc))
cm = metrics.confusion_matrix(y_test, y_pred)
print('confusion Matrix:')
print(cm, '\n\n')
print('-----------------------------------------------------------')
print('-----------------------------------------------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy: 0.75


confusion Matrix:
[[8294 2818]
 [ 772 2769]]


-----------------------------------------------------------
-----------------------------------------------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.91      0.75      0.82     11112
           1       0.50      0.78      0.61      3541

    accuracy                           0.75     14653
   macro avg       0.71      0.76      0.71     14653
weighted avg       0.81      0.75      0.77     14653
```

In summary, the classification model achieved an accuracy of 0.75, indicating that it correctly predicted 75% of instances in the test set. The confusion matrix revealed that 2,818 instances were falsely classified as positive, while 772 were falsely classified as negative. The results showed a precision of 0.50 and recall of 0.78 for the positive class, indicating room for improvement.

Considering these results, the model shows reasonably good performance. The accuracy score suggests a decent overall predictive capability. However, the lower precision and F1-score for the positive class indicate that the model may struggle with correctly identifying positive instances. Further evaluation and refinement could be done to improve the model's performance. Furthermore, generating the confusion matrix would visually provide details into the true positive, true negative, false positive, and false negative predictions.

In [37]:
```
ax = sns.heatmap(cm, cmap ='Blues', annot = True, fmt = 'd')

plt.xlabel('Predicted Class', fontsize=12)
plt.ylabel('True Class', fontsize=12)
plt.title('Confusion Matrix', fontsize=12)
plt.show()
```



As a additional insight and parameter to determine what factors and variables made most impact in our models prediction ability, the permutation importance function was used to permutate this results which shows that capital-gain, fnlwgt and capital-loss, age has the greatest importance in our model.

In [38]:
```
knn = KNeighborsClassifier()

# Fit the KNN model
knn.fit(X_train, y_train)

# Calculate permutation importances
result = permutation_importance(knn, X_test, y_test, n_repeats=10, random_state=42)

# DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'feature': X.columns, 'importance': result.importances_mean})
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

# Display the feature importances
print("Permutation Feature Importances:")
print(feature_importance_df)
```

```
Permutation Feature Importances:
          feature  importance
8    capital-gain    0.042885
2          fnlwgt    0.022678
9    capital-loss    0.013083
0             age    0.011772
10  hours-per-week    0.006825
5      occupation    0.000758
3       education    0.000232
6            race    0.000150
4   marital-status   -0.000014
1        workclass   -0.000027
7           gender   -0.000102
11  native-country   -0.000512
```

Moving forward we are going to employ another algorithm to train and test our dataset and compare the results of both algorithms afterwards.

Random Forest

Now, we will be employing the Random Forest algorithm for classification using a specific set of parameters. This process is quite similar to what was done earlier with

the KNN algorithm, only the model and parameter to be used will change. The Random Forest classifier was instantiated with 100 estimators and a random state of 42.

```
In [39]:
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

```
Out[39]:
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

The classifier was trained on X_train and y_train using the fit() method. Predictions for X_test were made with the predict() method, and the results were stored as y_pred.

```
In [40]:
y_pred = rf_model.predict(X_test)
```

```
In [41]:
print(y_pred.tolist())
```

```
[0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1,
 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0,
 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0,
 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1,
 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
```

After competing the training process, the results of the algorithm were computed and displayed as follows:

```
In [43]:
acc = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: %.2f\n\n'%(acc))
cm = metrics.confusion_matrix(y_test, y_pred)
print('confusion Matrix:')
print(cm, '\n\n')
print('------------------------------------------------------------')
print('------------------------------------------------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

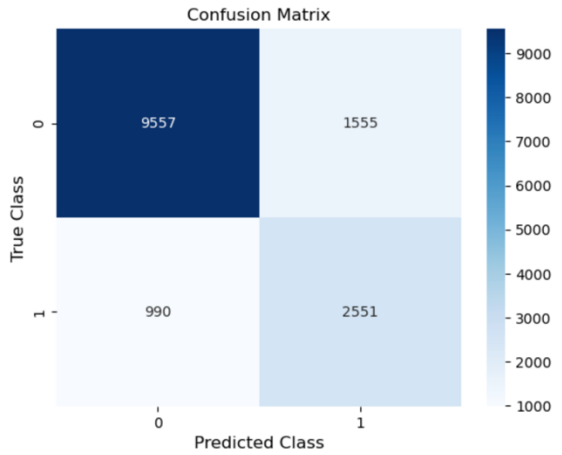
```
Accuracy: 0.83


confusion Matrix:
[[9557 1555]
 [ 990 2551]]


------------------------------------------------------------
------------------------------------------------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.91      0.86      0.88     11112
           1       0.62      0.72      0.67      3541

    accuracy                           0.83     14653
   macro avg       0.76      0.79      0.77     14653
weighted avg       0.84      0.83      0.83     14653
```

Interpretation from the results of the random forest classification algorithm, shows that the model achieved an accuracy of 0.83, correctly predicting 83% of instances. The model also shows a precision of 0.62 and recall of 0.72 for the positive class. the model

has a reasonably good performance, effectively identifies negative instances with high precision. However, it is possible to make improvements in the model to correctly classify positive instances.

The confusion matrix showed that 1,555 instances were falsely classified as positive, while 990 were falsely classified as negative.

In [44]:
```
ax = sns.heatmap(cm, cmap ='Blues', annot = True, fmt = 'd')

plt.xlabel('Predicted Class', fontsize=12)
plt.ylabel('True Class', fontsize=12)
plt.title('Confusion Matrix', fontsize=12)
plt.show()
```



Comparing the importance of the variables to the model as we have done in KNN shows that marital-status, capital-gain, education, age, occupation contributed the most to the prediction of the model.

In [45]:
```
rf = RandomForestClassifier()

# Fit the RF model
rf.fit(X_train, y_train)

# Calculate permutation importances
result = permutation_importance(rf, X_test, y_test, n_repeats=10, random_state=42)

# DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'feature': X.columns, 'importance': result.importances_mean})
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

# Display the feature importances
print("Permutation Feature Importances:")
print(feature_importance_df)
```

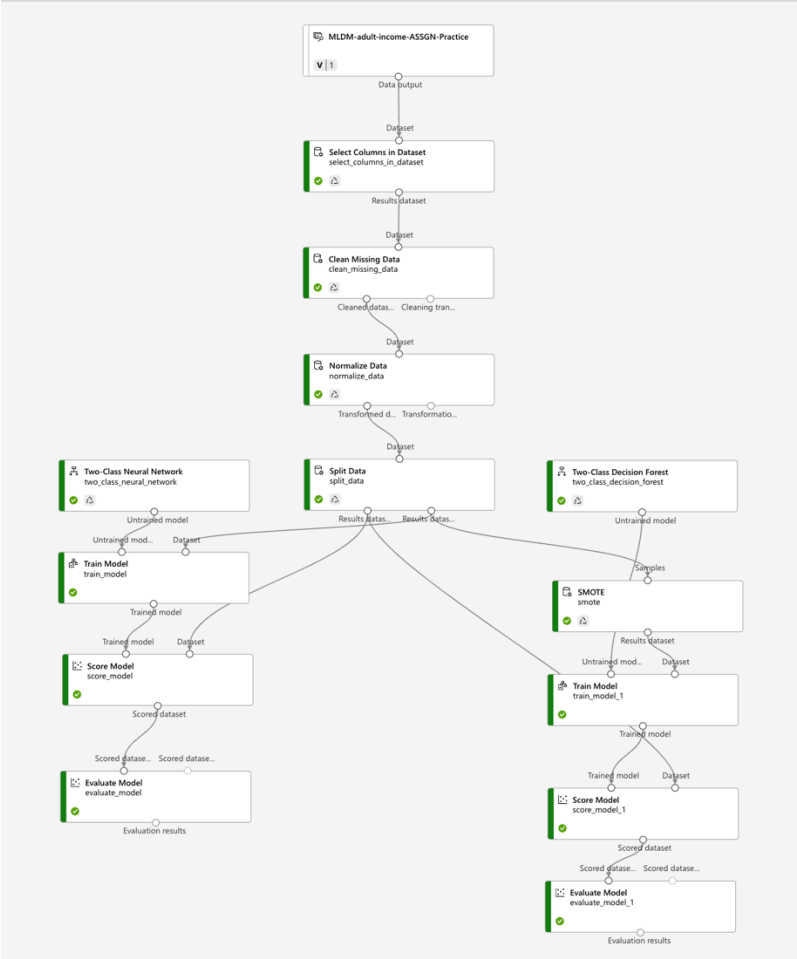
```
Permutation Feature Importances:
          feature  importance
4  marital-status    0.070306
8    capital-gain    0.043090
3       education    0.024452
0             age    0.016870
5      occupation    0.016454
10 hours-per-week    0.011417
9    capital-loss    0.009172
7          gender    0.008353
1       workclass    0.008087
11 native-country    0.002989
2          fnlwgt    0.002341
6            race    0.002320
```

In an overall comparison of both models, the Random Forest model outperforms the KNN model in terms of accuracy, precision and recall. It demonstrates better overall performance in correctly classifying instances, particularly for the positive class. Therefore, based on our dataset and task objective, the Random Forest model appears to be the better choice for performing this classification task. However, but models could be improved on for better effectiveness of predicting the positive instances mostly.

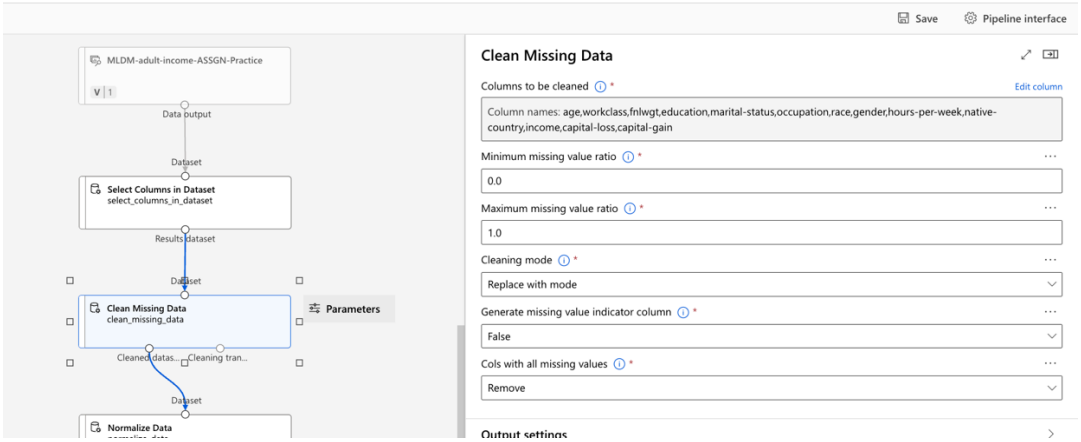
Implementation in Azure Machine Learning Designer

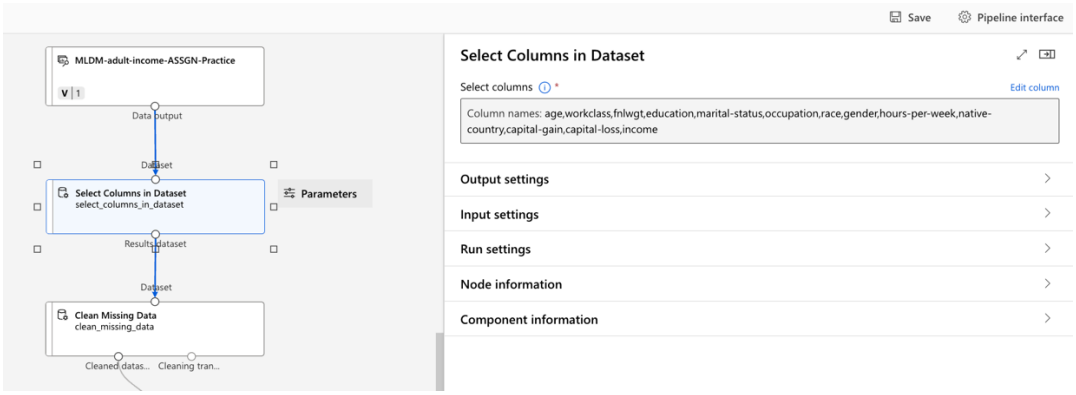
Azure Machine Learning is a powerful cloud-based service by Microsoft, offering a wide array of tools and services for building, training, and deploying machine learning models. Its integrated environment simplifies the end-to-end machine learning workflow, from data preparation to model deployment, eliminating the need for extensive coding. This automated approach drastically reduces the complexity of building machine learning models, enabling data scientists and developers to focus on the core aspects of their projects and achieve accurate classification models efficiently.

As imposed by the assessment criteria, in a classification task using Azure Machine Learning, two class neural networks and two class decision trees were employed on the given dataset. Below is an illustration of complete model design.

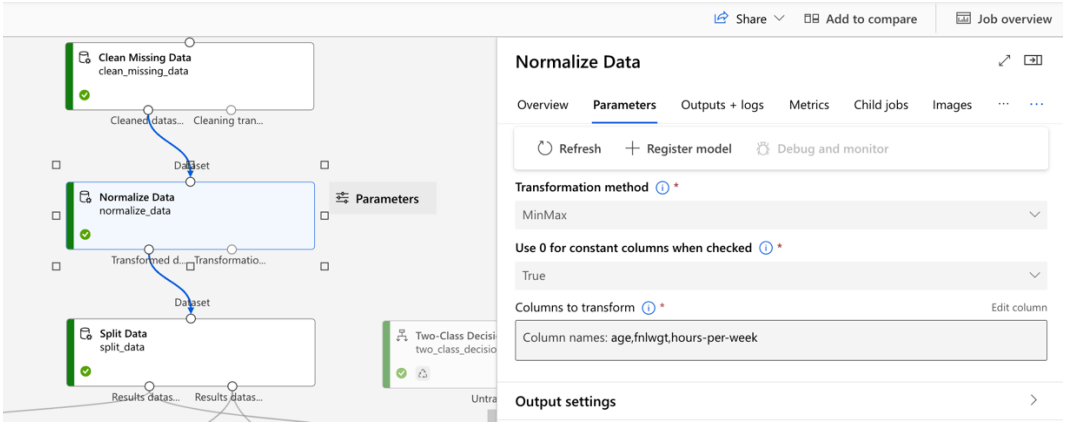


The dataset was imported into Azure from a local host and a new pipeline was created. To address missing values represented as "?" in the dataset, a preprocessing step was initially performed in Python using Jupyter Notebook, as explained in Task 1. Alternatively, Azure's “Execute Python component” could have been utilized. The dataset was further cleaned within Azure using components like "Select columns in Dataset" and "Cleaning missing Data". Unwanted columns were removed, and missing values were replaced with the mode. This approach provided flexibility and facilitated the efficient handling of missing data within the Azure environment, ensuring a clean and suitable dataset for subsequent analysis.

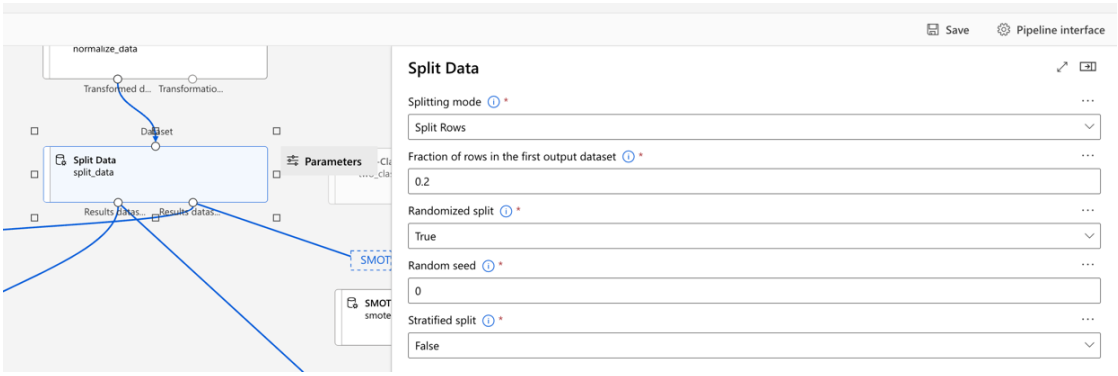




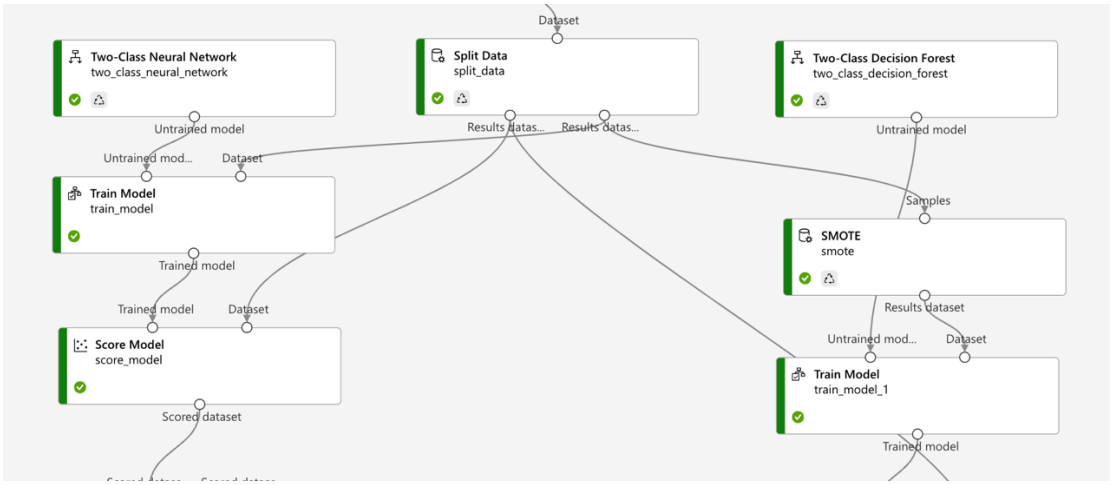
The next step involved the normalization of the cleaned data. Normalization of numerical data is a good practice that reduces the scale or range of values, ensuring that all features contribute equally to the analysis. Z score was used for the normalization process.



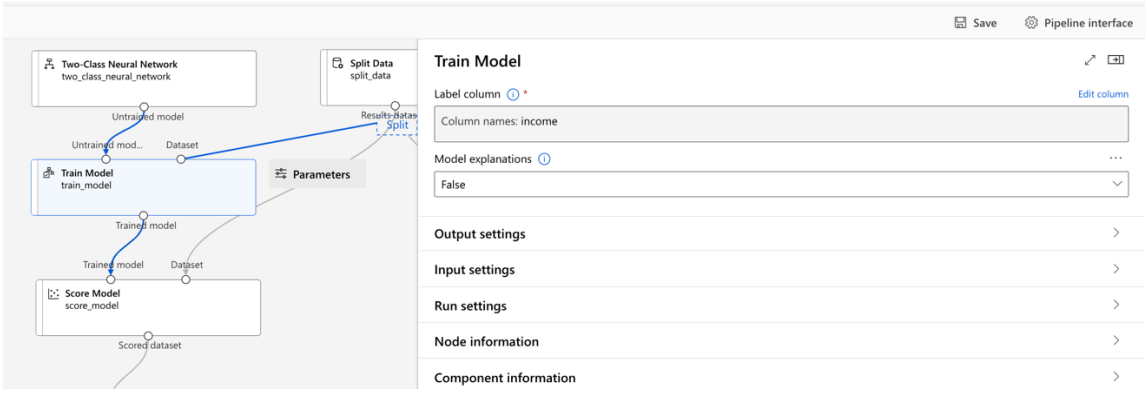
Once the above process has been achieved, it is assumed the data is deemed ready for classification. However, the problem of imbalance with the outcome variable will also be fixed afterward. Next, the clean data is split into two using “Split Data” component at a ratio of 80% for the training data and 20% for the test data.



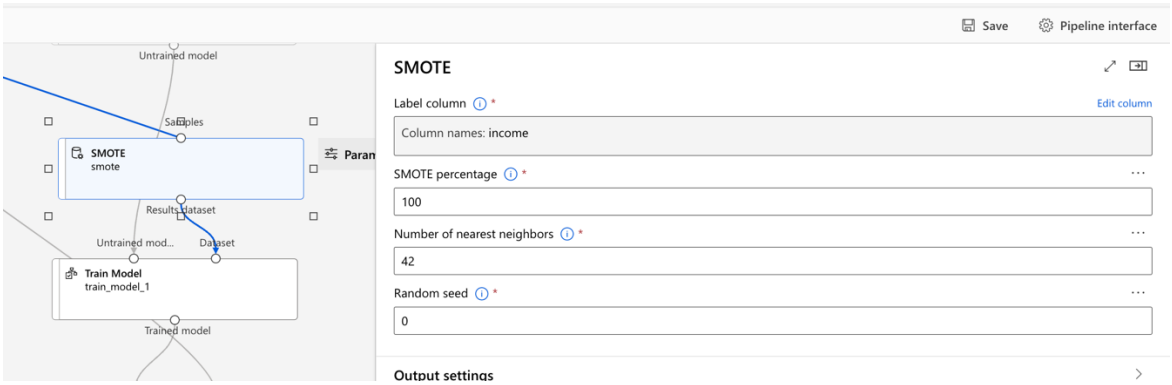
Two-class neural network models and two-class decision tree models were employed. The dataset was split into training and test data. The training data was linked into the datapoint of the training models. On the other hand, the test data was linked to the dataset for the score model, enabling the evaluation of the models' performance on unseen data.



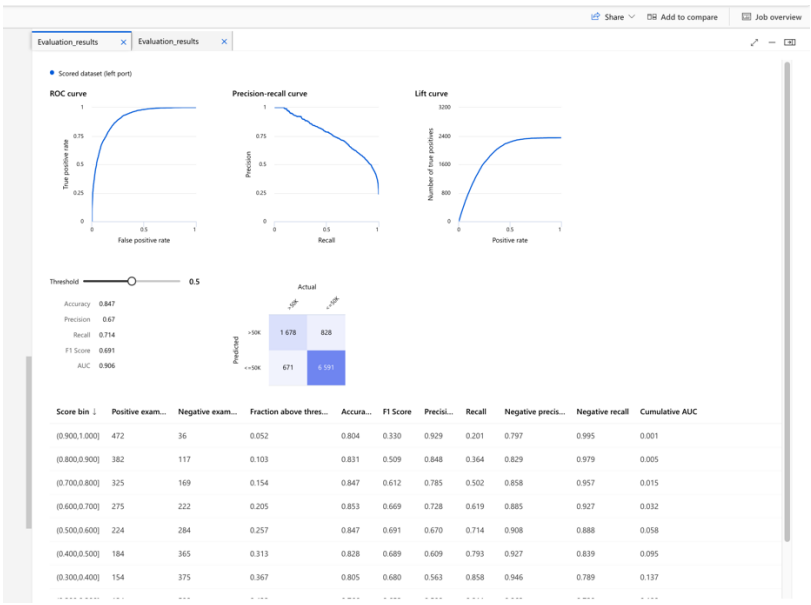
The training model parameters were configured to focus on the "income" variable, which serves as the outcome variable in this task. By setting the target variable to "income," the models were trained to learn patterns and relationships within the data that could predict the income class.



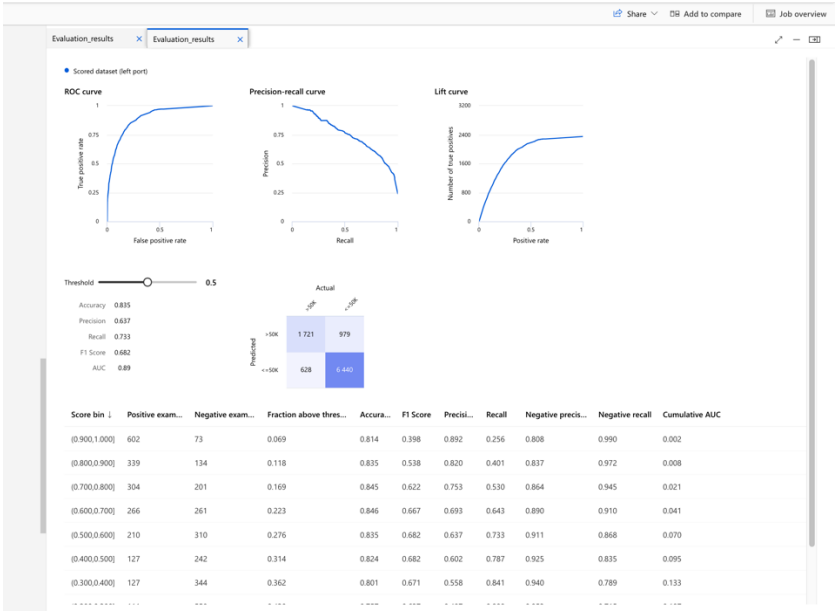
To address the issue of imbalanced data, the "smoke" component was used. It generates synthetic samples for the minority class, balancing the class distribution. This approach improves the classifier's ability to learn and make accurate predictions for both classes. This process was not done for neural network because the algorithm automatically balances imbalanced datasets.



The performance of a two-class neural network model was evaluated using various metrics, including accuracy, precision, recall, F1 score, and AUC (Area Under the Curve). The two-class neural network model achieved an accuracy of 0.847, indicating correct classification of approximately 84.7% of instances. Precision, measuring the proportion of true positives out of positive predictions, was 0.67, indicating 67% accurate positive predictions. Recall, reflecting the proportion of actual positives correctly identified, achieved a value of 0.714 (71.4%). The F1 score, combining precision and recall, was 0.691. The model exhibited good discriminatory power with an AUC of 0.906. Overall, the model demonstrated reasonably good performance.



The two-decicision tree model achieved an accuracy rate of 0.835, correctly classifying approximately 83.5% of instances. Precision, measuring the proportion of true positive predictions out of positive predictions, yielded a value of 0.637, indicating a 63.7% accuracy in identifying positive instances. The recall value, representing the proportion of actual positives correctly identified, was 0.733 (73.3%). The F1 score, which balances precision and recall, was calculated as 0.682.



In an overall comparison of all classification algorithms employed for this task, the neural network model generally outperforms all other models in terms of accuracy, precision, recall, F1 score, and AUC. It demonstrates better overall performance in

classifying instances and has a higher ability to discriminate between positive and negative instances.

Considering the ethical implications, particularly in relation to the F1 score, it is crucial to address potential biases. The model should undergo improvements to enhance fairness before deployment. This ensures a more ethical and equitable application, prioritizing responsible and unbiased predictive modeling practices.

Reference:

Atkinson, A. B., Rainwater, L., & Smeeding, T. M. (1995). Income Distribution in OECD Countries. OECD Social Policy Studies.

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32.

Cover, T., & Hart, P. (1967). Nearest Neighbor Pattern Classification. IEEE Transactions on Information Theory, 13(1), 21-27.

Jordan, M. I., & Mitchell, T. M. (2015). Machine Learning: Trends, Perspectives, and Prospects. Science, 349(6245), 255-260.

Sen, A., & Foster, J. E. (1996). On Economic Inequality. Oxford University Press.

Task 2:

"A Look at Financial Patterns: Comprehensive Analysis of Financial indicators Using K-Means and Agglomerative Clustering"

Introduction:

Clustering is a fundamental technique in data analysis that aims to group similar data points together based on their inherent patterns or similarities. It is basically an unsupervised learning method that does not rely on predefined labels or classes (Jain, A.K. et.al, 1999). Instead, clustering algorithms identify natural structures or clusters in the data, allowing for insights into the underlying relationships and similarities between data points.

Clustering can be defined as the process of partitioning a set of data points into homogeneous groups, known as clusters, where points within the same cluster are more similar to each other than to those in other clusters (Xu, R., & Wunsch II, D.C, 2005). The goal is to maximize the intra-cluster similarity and minimize the inter-cluster similarity, ensuring that data points within the same cluster are as similar as possible while being distinct from those in different clusters (Steinbach M. et.al, 2000). Clustering algorithms accomplish this by using various distance or similarity measures and optimization techniques to find the optimal grouping of data points.

For the purpose of this task, we will apply K-Means and allgromative clustering to time insights from a financial dataset

Clustering algorithms have wide-ranging applications in various domains, including customer segmentation, image and document analysis, anomaly detection, and social network analysis(Verma, P, et al. 2019).. They enable data scientists and analysts to discover hidden patterns, explore data structures, and gain valuable insights into complex datasets.

Dataset:

For this second task, the dataset includes annual salary, regular pay, incentive pay, and gross pay for employees under the County Executive and independently elected County officials for the years 2016 to the present.

The Independent and Dependent Variables:
The independent variables can be attributes such as first name, last name, department, job title, date of employment, sex, ethnicity, and pay-related information. The dependent variable could be the termination status (active or terminated) or any specific question or problem you want to solve by analyzing the data. Since we are using this dataset for a clustering task, we won't be focusing deeply on the dependent and independent variables.

The dataset contains the following data types:

- String: First name, last name, combo name, department, job title, sex, ethnicity, pay status.
- Date: Date started, orig start, date term.
- Numerical: Annual salary, regular pay, overtime pay, incentive pay, gross pay.

For further enquires and exploration, the primary source of the data is the United States official public data site. To access the dataset and learn more about its specific attributes and metadata, you can visit the following link: [Allegheny County Employee Salaries – DATA.GOV] (https://catalog.data.gov/dataset/allegheny-county-employee-salaries).

Objective

The primary objective of this project is to extract valuable insights from unstructured data, enabling stakeholders to make informed decisions. To achieve this, the project will focus on utilizing K-means clustering methods. By applying K-means clustering to the dataset, patterns and structures within the data will be uncovered. These findings will empower stakeholders with actionable knowledge, providing a deeper understanding of the dataset and facilitating well-informed decision-making. Through the use of K-means clustering, this project aims to distill meaningful insights, transforming unstructured data into valuable information in a concise and efficient manner.

Explanation and Preparation

Loading Libraries:

The project began by importing the necessary libraries, a crucial step to enable efficient data analysis, preprocessing, and the implementation of the clustering algorithms.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import warnings
from sklearn.decomposition import PCA
warnings.filterwarnings("ignore", category=FutureWarning)
```

The dataset was loaded using the pandas data frame.

In [2]:
```
dataset = pd.read_csv('redacted-2022-december-31-wprdc-new.csv')
```

In [3]:
```
dataset.head()
```

Out[3]:

	FIRST_NAME	LAST_NAME	Combo Name	DEPARTMENT	JOB_TITLE	ELECTED_OFFICIAL	DATE_STARTED	SEX	ETHNICITY	ORIG_START	DATE_
0	CATHERINE	ABALO	ABALO, CATHERINE	Kane Regional Centers	NURSING ASSISTANT	0	4/8/10	F	Black	7/6/09	
1	KEVIN J	ABBOTT	ABBOTT, KEVIN J	Emergency Management	FIRE INSTRUCTOR - PART TIME	0	6/11/18	M	White (Not of Hispanic Origin)	6/11/18	
2	JOY M	ABBOTT	ABBOTT, JOY M	Kane Regional Centers	COOK	0	2/14/99	F	White (Not of Hispanic Origin)	2/2/98	
3	ELIZABETH S	ABRAHAM	ABRAHAM, ELIZABETH S	Parks	SEASONAL AIDE	0	5/26/22	F	White (Not of Hispanic Origin)	5/26/22	
4	JASMINE	ABRAM	ABRAM, JASMINE	Emergency Management	TELECOMMUNICATION OFFICER	0	5/23/22	F	Black	5/31/16	

Initial exploration of the data lets us in on the shape, data types and missing values within the dataset. The dataset has 6280 rows and 17 columns. The columns contain various data types including 11 object, 1 int64, and 5 float64, representing different attributes of the employee salaries data.

In [4]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6280 entries, 0 to 6279
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   FIRST_NAME       6280 non-null   object
 1   LAST_NAME        6280 non-null   object
 2   Combo Name       6280 non-null   object
 3   DEPARTMENT       6280 non-null   object
 4   JOB_TITLE        6280 non-null   object
 5   ELECTED_OFFICIAL 6280 non-null   int64
 6   DATE_STARTED     6280 non-null   object
 7   SEX              6280 non-null   object
 8   ETHNICITY        6280 non-null   object
 9   ORIG_START       6280 non-null   object
 10  DATE_TERM        1263 non-null   object
 11  PAY_STATUS       6280 non-null   object
 12  ANNUAL_SALARY    6280 non-null   float64
 13  REGULAR_PAY      6275 non-null   float64
 14  OVERTIME_PAY     3999 non-null   float64
 15  INCENTIVE_PAY    3454 non-null   float64
 16  GROSS_PAY        6280 non-null   float64
dtypes: float64(5), int64(1), object(11)
memory usage: 834.2+ KB
```

In [5]:
```
dataset.shape
```

Out[5]:
```
(6280, 17)
```

In [6]:
```
dataset.isnull().sum()
```

Out[6]:
```
FIRST_NAME          0
LAST_NAME           0
Combo Name          0
DEPARTMENT          0
JOB_TITLE           0
ELECTED_OFFICIAL    0
DATE_STARTED        0
SEX                 0
ETHNICITY           0
ORIG_START          0
DATE_TERM        5017
PAY_STATUS          0
ANNUAL_SALARY       0
REGULAR_PAY         5
OVERTIME_PAY     2281
INCENTIVE_PAY    2826
GROSS_PAY           0
dtype: int64
```

Preprocessing

Before any preprocessing was done in python, the data was pre cleaned in Excel to enable easier readability of missing values in python. The initial step in python involved handling missing values in the "INCENTIVE_PAY" and "REGULAR_PAY" columns by replacing them with their respective mode values. This approach helps ensure that the most frequent values are used for imputation.

```
In [7]:
# Subtituting missing values with their modes
dataset['INCENTIVE_PAY'].fillna(dataset['INCENTIVE_PAY'].mode().iloc[0], inplace=True)
dataset['REGULAR_PAY'].fillna(dataset['REGULAR_PAY'].mode().iloc[0], inplace=True)
```

Furthermore, any rows with missing values in the "OVERTIME_PAY" column will be removed. Since the values in this column have distinct characteristics, such as being specific to each employee, replacing them with the mean or mode is not an optimal solution. Therefore, by removing rows with missing values in the "OVERTIME_PAY" column, we ensures the integrity of the analysis. It is important to note that missing values in the "DATE_TERM" column was ignored as this column will not be utilized in the analysis process.

```
In [8]:
dataset = dataset.dropna(subset=['OVERTIME_PAY'])
```

```
In [9]:
dataset.shape
```

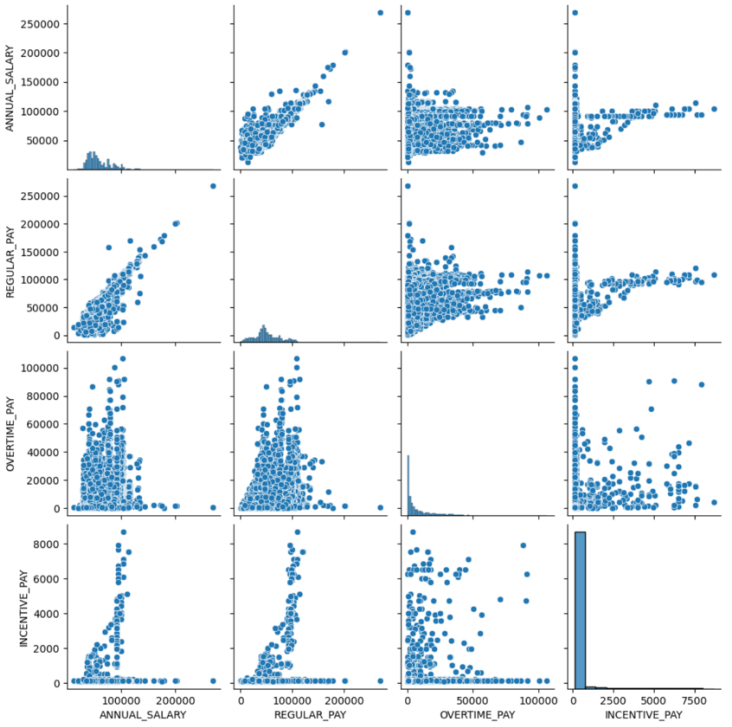
```
Out[9]:
(3999, 17)
```

```
In [10]:
dataset.isnull().sum()
```

```
Out[10]:
FIRST_NAME             0
LAST_NAME              0
Combo Name             0
DEPARTMENT             0
JOB_TITLE              0
ELECTED_OFFICIAL       0
DATE_STARTED           0
SEX                    0
ETHNICITY              0
ORIG_START             0
DATE_TERM           3534
PAY_STATUS             0
ANNUAL_SALARY          0
REGULAR_PAY            0
OVERTIME_PAY           0
INCENTIVE_PAY          0
GROSS_PAY              0
dtype: int64
```

Next, a pairplot was created using the seaborn library to visualize the pairwise relationships between annual salary, regular pay, overtime pay, and incentive pay. The plot revealed a positive linear relationship between annual salary and regular pay, while the relationships with overtime pay and incentive pay appeared more scattered.

```
In [11]: 1 X = dataset.iloc[:, 12:16]
         2 sns.pairplot(X)
Out[11]: <seaborn.axisgrid.PairGrid at 0x1384b0150>
```

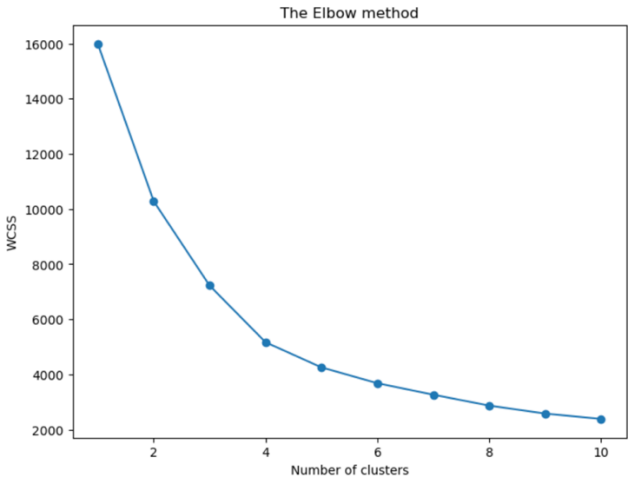


The dataset was transformed into a NumPy array by selecting columns 12 to 16 using iloc. This subset of data represents the features related to annual salary, regular pay, overtime pay, and incentive pay. The data was standardized to ensure consistent scales, the StandardScaler class from the scikit-learn library was used for this. The data was then transformed using the fit_transform method.

```
In [12]: 1 # "Standardizing the dataset to ensure that all variables are on the same scale, preventing certain features fro
         2 X = dataset.iloc[:, 12:16].values
         3 sc_X = StandardScaler()
         4 X = sc_X.fit_transform(X)
```

The Elbow method was used to determine the optimal number of clusters for the K-means clustering. The within-cluster sum of squares (WCSS) was computed for cluster numbers ranging from 1 to 10. The resulting plot displayed a gradual decrease in WCSS without a distinct elbow point, making it challenging to identify the optimal number of clusters. However, “n = 4” was observed to be the most possible number of cluster.

In [13]:
```
# using elbow method to find the optimal number of clusters
plt.figure(figsize=(8,6))
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plotting the Elbow method
plt.plot(range(1, 11), wcss, marker='o')
plt.title('The Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  # WCSS stands for "Within-Cluster Sum of Squares"
plt.show()
```



Consequently, K-means algorithm was fitted to the dataset with 4 clusters as decided after the viewing the elbow optimal number cluster. This algorithm was achieved 'k-means++' initialization method. Afterward, the resulting cluster labels were assigned to the dataset, creating a new column named "kmeans_cluster" and a count of the number of attributes are present in each cluster are displayed below:

In [14]:
```
# fitting the K-Means to the dataset
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=42)
y_kmeans = kmeans.fit_predict(X)
dataset["kmeans_cluster"] = y_kmeans
```

In [15]:
```
dataset['kmeans_cluster'].value_counts()
```

Out[15]:
```
kmeans_cluster
0    2512
2     860
1     537
3      90
Name: count, dtype: int64
```

The dataset was subjected to Principal Component Analysis (PCA) with 2 components, resulting in a reduced dimensionality. The explained variance ratios of these components were computed, indicating the proportion of the total variance explained by each component. In this case, the cumulative explained variance ratio of 0.7555 suggests that when considering both components together, they explain 75.55% of the total variance in the data.

In [16]:
```
pca=PCA(n_components=2)
X_reduced = pca.fit_transform(X)

pca.explained_variance_ratio_
```

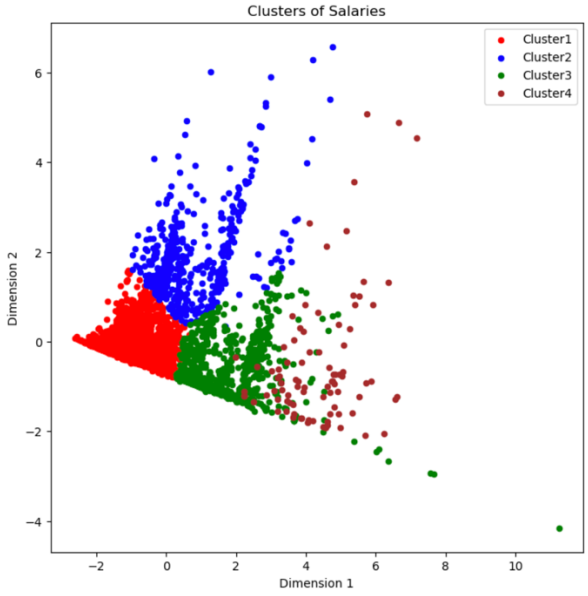
Out[16]:
```
array([0.52980021, 0.22572603])
```

In [17]:
```
sum(pca.explained_variance_ratio_)
```

Out[17]:
```
0.755526238022882
```

Judging from density and compactness of clusters suggests minimal variability within each group. Focusing on our columns of interest and the four distinct clusters used salary related insights can be deduced.

In [18]:
```
# Colour Range
colours = ['red', 'blue', 'green', 'brown', 'black']

plt.figure(figsize=(8,8))
for i in range(4):
    plt.scatter(X_reduced[y_kmeans == i, 0], X_reduced[y_kmeans == i, 1], s =20, c = colours[i], label = 'Cluste

plt.title('Clusters of Salaries')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend()
plt.show()
```



The analysis reveals key characteristics of each cluster. Cluster 0 (red) consists of individuals with lower salaries and a higher proportion of males. Cluster 1 (blue) exhibits moderate salaries with significant overtime pay. Cluster 2 (green) represents individuals with higher salaries, lower overtime pay, and a slightly higher proportion of males. Cluster 3 (brown) comprises individuals with the highest salaries, significant overtime pay, and a large proportion of males.

In [21]:
```
grouped_kmeans = new_dataset.groupby('kmeans_cluster')
cluster_kmeans = grouped_kmeans.mean()
print(cluster_kmeans)
```

```
                ELECTED_OFFICIAL  DATE_STARTED       SEX  ETHNICITY  \
kmeans_cluster
0                            0.0    749.633758  0.452229   4.303344
1                            0.0    696.121043  0.612663   4.262570
2                            0.0    736.986047  0.638372   4.659302
3                            0.0    661.044444  0.877778   4.833333

                ORIG_START   DATE_TERM  PAY_STATUS  ANNUAL_SALARY  \
kmeans_cluster
0               641.835191  211.330016    0.168392   47900.228977
1               571.603352  226.324022    0.026071   60870.256145
2               604.987209  225.189535    0.031395   88046.276826
3               567.500000  228.966667    0.011111   94383.211333

                 REGULAR_PAY  OVERTIME_PAY  INCENTIVE_PAY      GROSS_PAY
kmeans_cluster
0               38640.812771   3686.946986     143.616580   42410.767261
1               56585.934190  33652.170466     165.789348   90390.625847
2               86625.122081   6929.931674     221.180872   93726.525326
3               96620.761556  17468.652556    5018.878889  119108.293000
```

These findings provide insights for salary benchmarking, identifying gender disparities, and informing compensation strategies. Understanding the distinct salary clusters enables targeted actions to address disparities and optimize salary structures.

Agglomerative Clustering

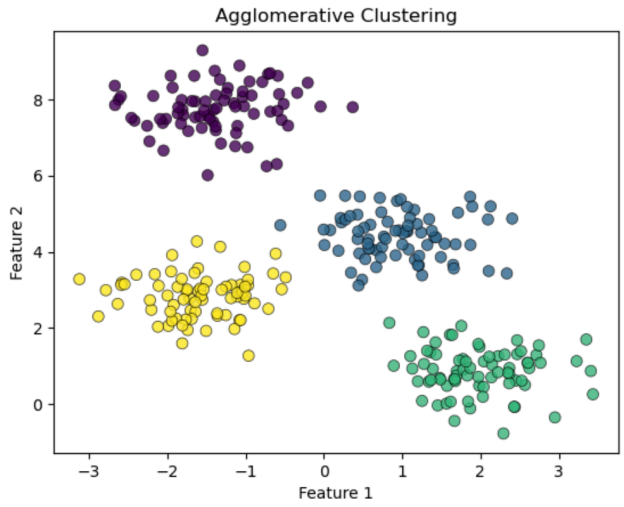
In [24]:
```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
```

In [25]:
```
# Generate some sample data
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.6, random_state=0)
```

In [27]:
```
n_clusters = 4
agg_clustering = AgglomerativeClustering(n_clusters=n_clusters)
agg_clustering.fit(X)
```

Out[27]:
```
AgglomerativeClustering(n_clusters=4)
```

In [29]:
```
# Visualize the clusters
colors = ['red', 'blue', 'green', 'brown', 'black']
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', edgecolor='k', s=50,
            linewidths=0.5, alpha=0.8, marker='o', facecolors=[colors[label] for label in labels])
plt.title('Agglomerative Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# View the updated dataset with appended clusters
print(dataset.head())
```




```
   FIRST_NAME   LAST_NAME              Combo Name              DEPARTMENT  \
0    CATHERINE       ABALO        ABALO, CATHERINE    Kane Regional Centers
2        JOY M      ABBOTT           ABBOTT, JOY M    Kane Regional Centers
4     JASMINE        ABRAM           ABRAM, JASMINE     Emergency Management
5  GWENDOLYN S  ABRAMOVITZ  ABRAMOVITZ, GWENDOLYN S  Administrative Services
6     KIARA N   ACIE-SCOTT      ACIE-SCOTT, KIARA N    Kane Regional Centers

                      JOB_TITLE  ELECTED_OFFICIAL DATE_STARTED SEX  \
0              NURSING ASSISTANT                 0       4/8/10   F
2                           COOK                 0      2/14/99   F
4       TELECOMMUNICATION OFFICER                 0      5/23/22   F
5            MANAGER REGISTRATION                 0     12/31/00   F
6  FOOD SERVICE WORKER/START 2013                 0      8/23/15   F

                      ETHNICITY ORIG_START DATE_TERM PAY_STATUS  \
```

Reference:

Jain, A.K., Murty, M.N., & Flynn, P.J. (1999). Data clustering: A review. ACM Computing Surveys, 31(3), 264-323.

Xu, R., & Wunsch II, D.C. (2005). Survey of clustering algorithms. IEEE Transactions on Neural Networks, 16(3), 645-678.

Steinbach, M., Karypis, G., & Kumar, V. (2000). A comparison of document clustering techniques. In KDD Workshop on Text Mining, 1-9.

Verma, P., & Mehta, S. (2019). Clustering techniques: A comprehensive survey. International Journal of Computer Applications, 182(8), 1-6.

Task 3:

Stars and Sentiments: A Text Mining and Sentiment Analysis Approach to Predicting Hotel Ratings

Introduction:

Text mining, also known as text analytics, is a sophisticated computational process aimed at extracting valuable insights and knowledge from unstructured textual data sources. Its applications span diverse domains, including sentiment analysis, topic modeling, text classification, entity extraction, and summarization. The primary goal of text mining is to transform large volumes of unstructured text, such as documents, social media posts, and customer reviews, into structured and actionable information.

Text mining involves several key steps. Firstly, it preprocesses raw text data, including tasks such as tokenization, stemming, and removal of stop words, to enhance computational efficiency (Manning et al., 2008). Subsequently, it employs various techniques such as natural language processing (NLP), machine learning, and statistical methods to analyze and derive meaningful patterns, sentiments, and topics from the processed text (Mikolov et al., 2013). Additionally, text mining can unveil relationships between entities and perform sentiment analysis to discern opinions and emotions expressed in the text (Pang & Lee, 2008).

The significance of text mining lies in its ability to sift through vast amounts of unstructured data, providing a structured and insightful representation that facilitates decision-making and knowledge discovery in fields ranging from marketing and customer feedback analysis to academic research and healthcare informatics.

Text mining

The object of this task is to predict if a hotel user is likely to give the establishment a 5, 4, 3, 2, 1 star based on the comments they have previously laid, and to build a model that can predict their comments are positive, negative, or neutral.

The dataset was sourced from Kaggle data resprotory , it contains information bout costumers reviews at a hotel and their ratings, the data was curated by Alam, M. H. in 2016.

Explanation and Preparation

Loading Libraries:

As usual, all needed libraries were imported, and the data set was loaded onto the Kernel using pandas data frame

In [1]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from nltk.sentiment import SentimentIntensityAnalyzer
from sklearn.metrics import accuracy_score, classification_report
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from wordcloud import WordCloud
from imblearn.over_sampling import RandomOverSampler
import re
import nltk
nltk.download(['stopwords', 'punkt', 'wordnet', 'omw-1.4', 'vader_lexicon'])
import warnings
warnings.filterwarnings("ignore")
```

The dataset is mapped to converts a numerical 'Rating' column in the dataset into descriptive labels using a mapping dictionary. Numerical values 1 and 2 are labeled as 'Negative', 3 as 'Neutral', and 4 and 5 as 'Positive'. The transformed labels are stored in a new column called “Target_Label”. This process is in a bid to enhances the understandability of the ratings.

In [3]:
```
dataset = pd.DataFrame(dataset)

# Define a mapping for replacement
rating_mapping = {
    1: 'Negative',
    2: 'Negative',
    3: 'Neutral',
    4: 'Positive',
    5: 'Positive'
}

# Replace the numerical values with labels
dataset['Target_Label'] = dataset['Rating'].replace(rating_mapping)

# Display the result
dataset.head(10)
```

Out[3]:

	Review	Rating	Target_Label
0	nice hotel expensive parking got good deal sta...	4	Positive
1	ok nothing special charge diamond member hilto...	2	Negative
2	nice rooms not 4* experience hotel monaco seat...	3	Neutral
3	unique, great stay, wonderful time hotel monac...	5	Positive
4	great stay great stay, went seahawk game aweso...	5	Positive
5	love monaco staff husband stayed hotel crazy w...	5	Positive
6	cozy stay rainy city, husband spent 7 nights m...	5	Positive
7	excellent staff, housekeeping quality hotel ch...	4	Positive
8	hotel stayed hotel monaco cruise, rooms genero...	5	Positive
9	excellent stayed hotel monaco past w/e delight...	5	Positive

The total number of outcome variable was displayed and the presence of imbalanced data in the outcome variable was acknowledged. Appropriate techniques will be implemented to address the issue and ensure reliable analysis.

In [4]:
```
# Calculate the sum of each unique value in the 'Rating' column
Target_Label_sums = dataset.groupby('Target_Label').size().reset_index(name='Count')

# Display the result
Target_Label_sums
```

Out[4]:

	Target_Label	Count
0	Negative	3214
1	Neutral	2184
2	Positive	15093

Other EDA were performed on the data to further determine its nature, no missing value was encountered.

In [7]:
```
dataset.isnull().sum()
```

Out[7]:
```
Review          0
Rating          0
Target_Label    0
dtype: int64
```

To select features, the CountVectorizer was employed to transform the text data in the “Review” column of the dataset into numerical representations. The resulting feature were assigned as follows:

X = Reviews
Y = Target_Label

In [8]:
```
# selecting features
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['Review'])
X = pd.DataFrame(X.toarray())
y = dataset['Target_Label']
X.head()
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	52913	52914	52915	52916	52917	52918	52919	52920	52921	52922
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 52923 columns

The developed preprocess_text function applies tokenization, stopword removal, lowercase conversion, and stemming to effectively preprocess text data, enhancing its quality and suitability for downstream analysis in natural language processing tasks. Afterward the dataset was split using train_test_split.

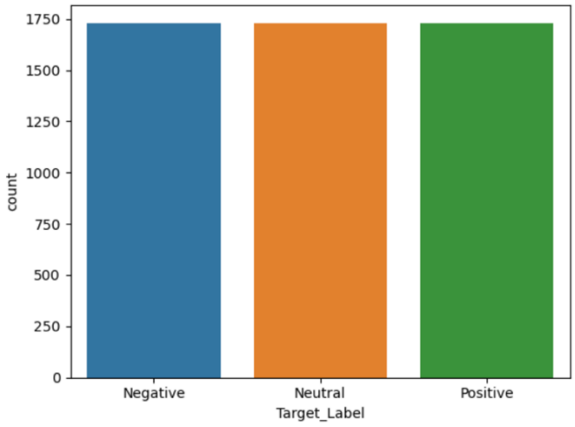
In [9]:
```
# lets now create a function to apply all our data preprocessing steps which we can then use on a corpus
def preprocess_text(text):
    tokenized_document = nltk.tokenize.RegexpTokenizer('[a-zA-Z0-9\']+').tokenize(text)  # tokenize
    cleaned_tokens = [word.lower() for word in tokenized_document if word.lower() not in stop_words] #remove
    stemmed_text = [nltk.stem.PorterStemmer().stem(word) for word in cleaned_tokens]  # stemming
    return stemmed_text
```

In [10]:
```
from sklearn.model_selection import train_test_split

y = dataset['Target_Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size = 0.2, random_state = 99)
```

To handle the imbalanced outcome variable noticed during the EDA, Undersampling using RandomUnderSampler from imblearn was utilized. This technique randomly undersamples the majority class, resulting in balanced datasets. A countplot visualization showcased the equalized distribution of classes, leading to improved model training and accurate predictions.

In [11]:
```
from imblearn.under_sampling import RandomUnderSampler

resampler = RandomUnderSampler(random_state=0)
X_train_undersampled, y_train_undersampled = resampler.fit_resample(X_train, y_train)

sns.countplot(x=y_train_undersampled)
```

Out[11]: <Axes: xlabel='Target_Label', ylabel='count'>



Multinomial Naive Bayes classifier from scikit-learn was trained on the undersampled dataset. The model learned patterns and relationships within the features to make accurate predictions. The trained Multinomial Naive Bayes model was then applied to make predictions on the X_test dataset and displayed as y_pred

In [12]:
```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(X_train_undersampled, y_train_undersampled)
```

Out[12]:
```
MultinomialNB()
```

In [13]:
```
y_pred = model.predict(X_test)
print(y_pred.tolist())
```

```
tive', 'Positive', 'Positive', 'Negative', 'Positive', 'Positive', 'Positive', 'Positive', 'Negative', 'Neutral',
'Negative', 'Neutral', 'Neutral', 'Neutral', 'Positive', 'Positive', 'Positive', 'Neutral', 'Positive', 'Positiv
e', 'Positive', 'Neutral', 'Positive', 'Negative', 'Positive', 'Neutral', 'Positive', 'Neutral', 'Positive', 'Pos
itive', 'Positive', 'Neutral', 'Positive', 'Neutral', 'Positive', 'Negative', 'Positive', 'Neutral', 'Positive',
'Negative', 'Positive', 'Neutral', 'Positive', 'Positive', 'Negative', 'Positive', 'Neutral', 'Negative', 'Neutra
l', 'Positive', 'Positive', 'Neutral', 'Neutral', 'Neutral', 'Positive', 'Positive', 'Negative', 'Positive', 'Neg
ative', 'Negative', 'Neutral', 'Neutral', 'Negative', 'Positive', 'Negative', 'Positive', 'Neutral', 'Positive',
'Neutral', 'Positive', 'Positive', 'Positive', 'Positive', 'Positive', 'Positive', 'Neutral', 'Neutral', 'Neutra
l', 'Positive', 'Neutral', 'Neutral', 'Neutral', 'Positive', 'Positive', 'Neutral', 'Negative', 'Neutral', 'Neutr
al', 'Positive', 'Positive', 'Positive', 'Positive', 'Positive', 'Negative', 'Positive', 'Negative', 'Positive',
'Positive', 'Positive', 'Negative', 'Neutral', 'Positive', 'Positive', 'Neutral', 'Positive', 'Neutral', 'Positiv
e', 'Positive', 'Positive', 'Neutral', 'Positive', 'Positive', 'Positive', 'Positive', 'Negative', 'Neutral', 'Po
sitive', 'Negative', 'Negative', 'Positive', 'Positive', 'Positive', 'Positive', 'Neutral', 'Neutral', 'Neutral',
'Neutral', 'Positive', 'Positive', 'Negative', 'Positive', 'Positive', 'Neutral', 'Neutral', 'Positive', 'Neutra
l', 'Positive', 'Positive', 'Positive', 'Neutral', 'Neutral', 'Positive', 'Neutral', 'Neutral', 'Neut
ral', 'Positive', 'Positive', 'Negative', 'Positive', 'Positive', 'Positive', 'Positive', 'Neutral', 'Neutral',
'Negative', 'Neutral', 'Neutral', 'Positive', 'Positive', 'Neutral', 'Positive', 'Neutral', 'Positive', 'Positiv
e', 'Positive', 'Positive', 'Positive', 'Neutral', 'Positive', 'Neutral', 'Neutral', 'Negative', 'Positive', 'Pos
itive', 'Positive', 'Positive', 'Positive', 'Neutral', 'Neutral', 'Positive', 'Positive', 'Negative', 'Negative',
'Neutral', 'Positive', 'Positive', 'Positive', 'Neutral', 'Positive', 'Negative', 'Positive', 'Negative', 'Positi
```

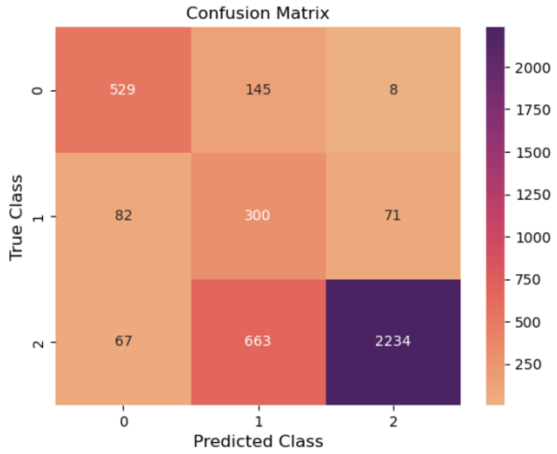
The model achieved an accuracy of 0.75 on the test dataset. The model mostly performed well in predicting the positive class with high precision (0.97) and recall (0.75), indicating a good balance between correct identification and avoiding false

positives. However, the model struggled to accurately classify neutral instances, as reflected by lower precision (0.27) and recall (0.66). Therefore, further analysis could be carried out.

In [15]:
```
#computing the accuracy and making the confusion matrix
from sklearn import metrics
acc = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: %.2f\n\n'%(acc))
cm = metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm, '\n\n')
print('-------------------------------------------------------------')
print('-------------------------------------------------------------')
print('\n\n')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy: 0.75


Confusion Matrix:
[[ 529  145    8]
 [  82  300   71]
 [  67  663 2234]]


------------------------------------------------------------
------------------------------------------------------------


Classification Report:

              precision    recall  f1-score   support

    Negative       0.78      0.78      0.78       682
     Neutral       0.27      0.66      0.38       453
    Positive       0.97      0.75      0.85      2964

    accuracy                           0.75      4099
   macro avg       0.67      0.73      0.67      4099
weighted avg       0.86      0.75      0.78      4099
```

In [16]:
```
ax = sns.heatmap(cm, cmap ='flare', annot = True, fmt = 'd')

plt.xlabel('Predicted Class', fontsize=12)
plt.ylabel('True Class', fontsize=12)
plt.title('Confusion Matrix', fontsize=12)
plt.show()
```



Sentiment analysis

Sentiment analysis, also known as opinion mining, is a natural language processing technique that aims to determine the sentiment or emotion expressed in a piece of text. It involves extracting subjective information from text data and categorizing it as positive, negative, or neutral. Sentiment analysis is widely used in various applications, such as social media monitoring, customer feedback analysis, brand reputation management, and market research.

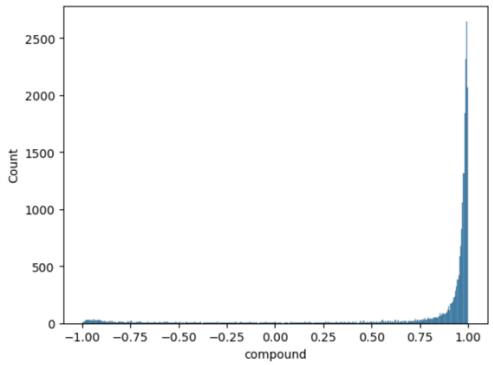
Firstly, SentimentIntensityAnalyzer was used to assigns sentiment scores to words, allowing for the assessment of overall sentiment in a given text. SentimentIntensityAnalyzer is a tool used to analyze sentiment in text. U sing the tool, the compound score of the whole sentiment analysis was assigned. Indicating negative, neutral, and positive sentiments. By incorporating these sentiment scores into the dataset, it becomes possible to capture and analyze the sentiment information associated with each review for various downstream tasks.

```
In [17]:
# Initialize the SentimentIntensityAnalyzer
sentiment = SentimentIntensityAnalyzer()
```

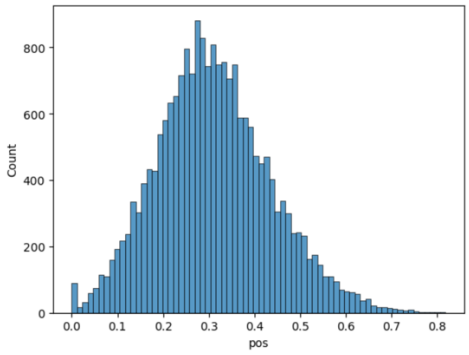
```
In [18]:
dataset['compound'] = [sentiment.polarity_scores(review)['compound'] for review in dataset['Review']]
dataset['neg'] = [sentiment.polarity_scores(review)['neg'] for review in dataset['Review']]
dataset['neu'] = [sentiment.polarity_scores(review)['neu'] for review in dataset['Review']]
dataset['pos'] = [sentiment.polarity_scores(review)['pos'] for review in dataset['Review']]
```

Form the histogram plot, it can be seen that the positive sentiments or reviews occurs most often.

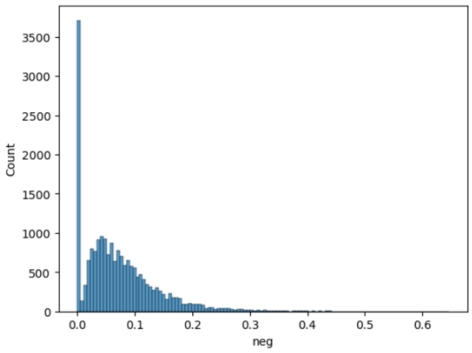
```
In [21]:
sns.histplot(dataset['compound'])
Out[21]: <Axes: xlabel='compound', ylabel='Count'>
```




```
In [22]:
sns.histplot(dataset['pos'])
Out[22]: <Axes: xlabel='pos', ylabel='Count'>
```



```
In [23]: 1 sns.histplot(dataset['neg'])
Out[23]: <Axes: xlabel='neg', ylabel='Count'>
```



A new subset was created to extract words from the dataset based on positive, neutral, and negative sentiment categories. WordClouds was then initiated to visualize the most commonly occurring words for each sentiment category. The positive_wordcloud represents positive sentiment, while the neutral_wordcloud and negative_wordcloud depict words associated with neutral and negative sentiments respectively. This approach offers a visual representation of the prevailing themes and topics within each sentiment category and providing insights into the overall sentiment expressed in the dataset. Next the text was tokenized by splitting it into individual words and removes punctuation. For each sentiment category (negative, neutral, positive), the text is converted to lowercase, and only alphabetic words are retained. The resulting tokens_negative, tokens_neutral, and tokens_positive contain the processed words from the respective sentiment categories.

In [25]:
```
# Extract words for each sentiment category
positive_reviews_words = ' '.join(dataset[dataset['Target_Label'] == 'Positive']['Review'])
neutral_reviews_words = ' '.join(dataset[dataset['Target_Label'] == 'Neutral']['Review'])
negative_reviews_words = ' '.join(dataset[dataset['Target_Label'] == 'Negative']['Review'])

# Generate WordCloud for each sentiment category
positive_wordcloud = WordCloud(width=800, height=400).generate(positive_reviews_words)
neutral_wordcloud = WordCloud(width=800, height=400).generate(neutral_reviews_words)
negative_wordcloud = WordCloud(width=800, height=400).generate(negative_reviews_words)
```

In [26]:
```
# Tokenize the text and remove punctuation
tokens_negative = [word.lower() for word in word_tokenize(negative_reviews_words) if word.isalpha()]
tokens_neutral = [word.lower() for word in word_tokenize(neutral_reviews_words) if word.isalpha()]
tokens_positive = [word.lower() for word in word_tokenize(positive_reviews_words) if word.isalpha()]
```

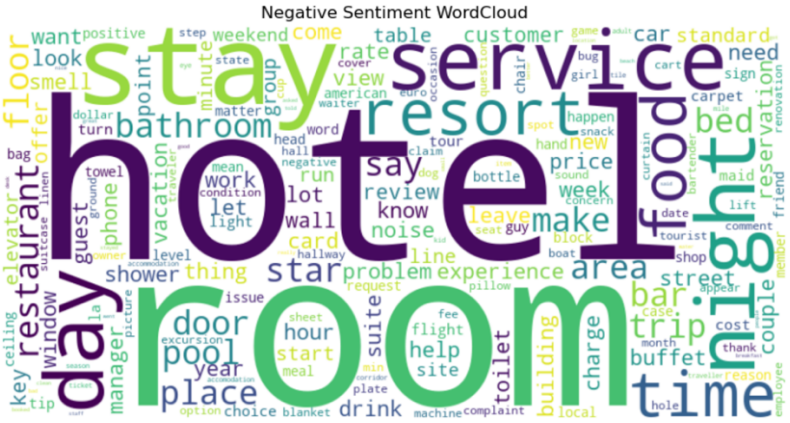
The function "generate_wordcloud" was used to simplify the visualization of frequent words for each sentiment category by allowing easy customization of dimensions.

In [28]:
```
# Generating WordCloud for each sentiment category
def generate_wordcloud(words, title):
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(words)
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(title)
    plt.show()
```

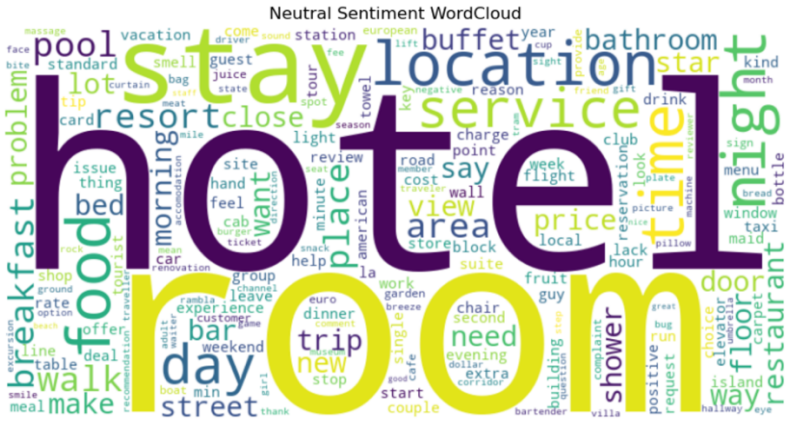
In [29]:
```
# Displaying the most common words for each sentiment
def display_most_common(fdist, title):
    print(f"\n{title} Sentiment:")
    common_words = fdist.most_common(2000)
    if not common_words:
        print(f"No words found for {title} sentiment.")
    else:
        generate_wordcloud(' '.join([word[0] for word in common_words]), f'{title} Sentiment WordCloud')

# WordCloud of positive sentiments
display_most_common(fdist_positive, 'Positive')
```

Additionally, a histogram plot is applied to display the most common words in 2000 instances. The resulting visuals provide insights into the prevalent words associated with each sentiment. However, as the actual visuals cannot be displayed here, please refer to the generated histogram plot for a comprehensive view of the most frequent words in each sentiment category.

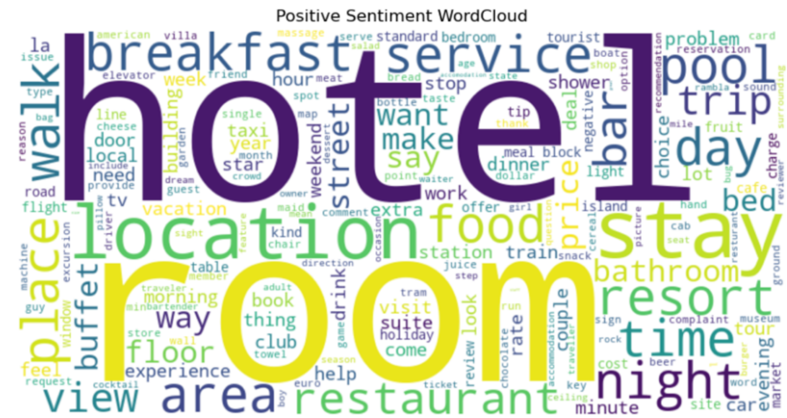
Negative Sentiment:



Neutral Sentiment:



```
Positive Sentiment:
```



And accuracy of 73% suggest that this model has performed well, however, there can be room for improvement.

Refrences:

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.

Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. Foundations and Trends® in Information Retrieval, 2(1–2), 1–135.

Alam, M. H., Ryu, W.-J., Lee, S., 2016. Joint multi-grain topic sentiment: modeling semantic aspects for online reviews. Information Sciences 339, 206–223.