Assignment -4_ DASC5301

FIRST QUESTION:-Data from our lives

★ Describe a situation or problem from your job, everyday life, current events, etc., for which a classification would be appropriate?

➡ **Your answer**

Fake News Classification:-

Detecting fake news has become paramount. Machine learning models, employing classification techniques, can distinguish between genuine and fabricated news articles. These models analyze multiple features, such as content, writing style, and source credibility, to predict the authenticity of news pieces. Accurate classification is crucial as misclassifications can lead to the spread of misinformation, impacting public perception and trust in media sources. This example underscores the importance of precise classification in preserving the credibility of news in today's online landscape.

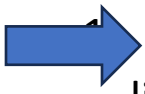
2. Preprocessing

```
[1] from scipy import stats
    from sklearn.linear_model import LinearRegression
    from statsmodels.compat import lzip
    from statsmodels.formula.api import ols
    from statsmodels.stats.anova import anova_lm
    from statsmodels.stats.outliers_influence import variance_inflation_factor
    import matplotlib
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd
    import seaborn as sns
    import statsmodels.api as sm

    %matplotlib inline
```

1. scipy: A library for scientific computing that includes statistical functions.
2. sklearn.linear_model: Part of scikit-learn, this module provides tools for linear regression.
3. statsmodels.compat: Compatibility module for statsmodels, which is a library for estimating and testing statistical models.
4. statsmodels.formula.api: A module in statsmodels that provides a formula-based interface to specify linear models.
5. statsmodels.stats.anova: A module for performing analysis of variance (ANOVA).
6. statsmodels.stats.outliers_influence: A module for detecting influential data points and computing variance inflation factors.
7. matplotlib: A popular plotting library for creating static, animated, and interactive visualizations.
8. numpy: A library for numerical operations in Python.
9. pandas: A powerful data manipulation library.

10. seaborn: A statistical data visualization library based on matplotlib.
11. statsmodels.api: The main API for statsmodels, which includes tools for estimating and testing models.

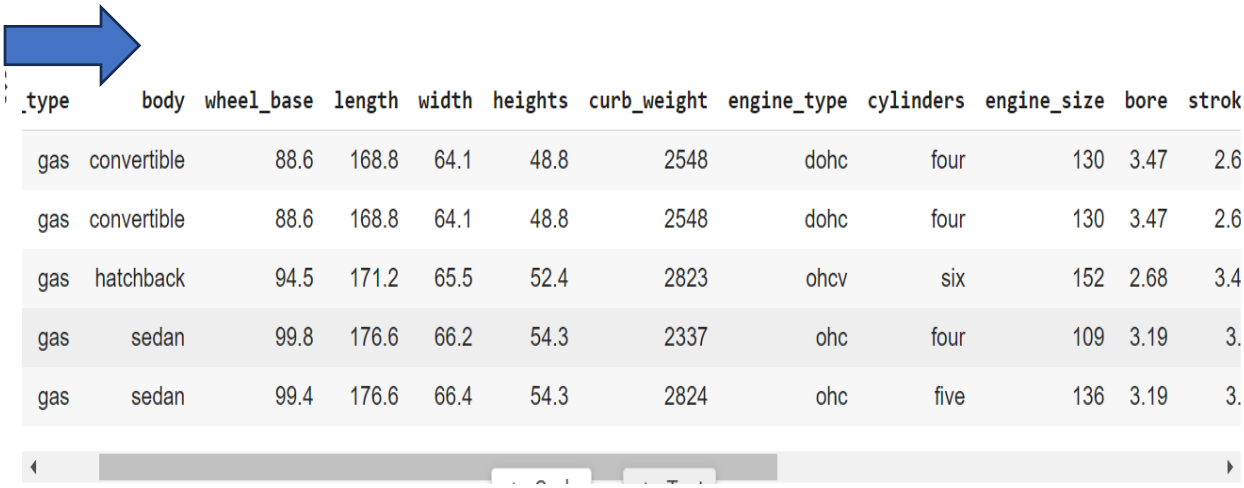
```
#Read in data
df =pd.read_csv('auto_imports1.csv')

df.head()
```



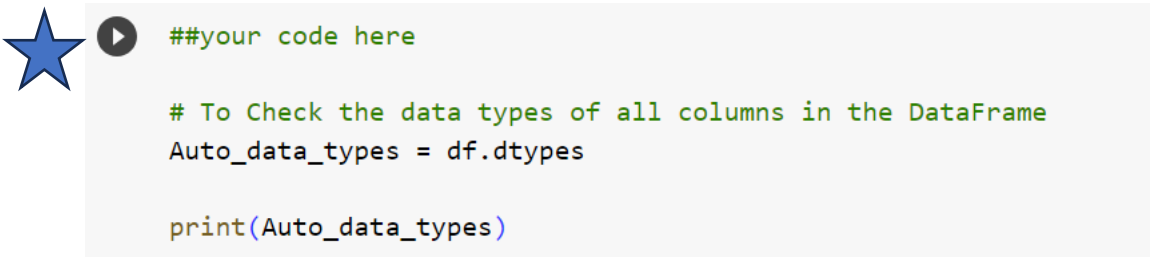
import pandas as pd: This line imports the pandas library and assigns it the alias 'pd'. This is a common convention to make the code more concise.

2. pd.read_csv('auto_imports1.csv'): The read_csv() function is a pandas function used to read data from a CSV file. It takes the file path as an argument and returns a DataFrame containing the data from the CSV file.
3. df = ...: The result of pd.read_csv() is assigned to the variable 'df'. 'df' is a common naming convention for a DataFrame (short for Data Frame), which is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns).
4. df.head(): This method is used to display the first 5 rows of the DataFrame. It's a quick way to inspect the data and get a sense of its structure.

5. this code snippet reads the contents of the 'auto_imports1.csv' file into a Pandas DataFrame named 'df' and then displays the first 5 rows of the DataFrame to provide a glimpse of the data.



_type	body	wheel_base	length	width	heights	curb_weight	engine_type	cylinders	engine_size	bore	strok
gas	convertible	88.6	168.8	64.1	48.8	2548	dohc	four	130	3.47	2.6
gas	convertible	88.6	168.8	64.1	48.8	2548	dohc	four	130	3.47	2.6
gas	hatchback	94.5	171.2	65.5	52.4	2823	ohcv	six	152	2.68	3.4
gas	sedan	99.8	176.6	66.2	54.3	2337	ohc	four	109	3.19	3.
gas	sedan	99.4	176.6	66.4	54.3	2824	ohc	five	136	3.19	3.



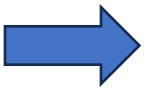
```
##your code here

# To Check the data types of all columns in the DataFrame
Auto_data_types = df.dtypes

print(Auto_data_types)
```

Extract the data types of each column in the DataFrame and assign it to the variable 'Auto_data_types'

2. df.dtypes: This is an attribute of a Pandas DataFrame that returns a Series with the data type of each column. The resulting Series has the column names as the index and the corresponding data types as values.
3. Auto_data_types = ...: The extracted Series of data types is assigned to the variable 'Auto_data_types'. This

variable will contain information about the data types of each column in the DataFrame.

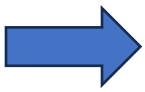
4. print(Auto_data_types): This line prints the contents of the 'Auto_data_types' Series, displaying the data types of each column in the DataFrame.
5. In the output, 'column1', 'column2', and 'column3' are assumed to be placeholder names for the actual column names in your DataFrame. The data types (int64, float64, object, etc.) provide information about the types of values stored in each column (integer, float, string, etc.). This information is useful for understanding the nature of the data and for preprocessing or analysis tasks that may be sensitive to data types.



```
fuel_type       object
body            object
wheel_base     float64
length         float64
width          float64
heights        float64
curb_weight      int64
engine_type     object
cylinders       object
engine_size      int64
bore            object
stroke          object
compression    float64
horse_power     object
peak_rpm        object
city_mpg         int64
highway_mpg      int64
price            int64
dtype: object
```

```
[4]  ## Your code here

     ## Replacing ? with none throught the dataset
     df = df.replace('?', None)

     ##converting object to float variables
     df['bore'] = df['bore'].astype(float)
     df['stroke'] = df['stroke'].astype(float)
     df['horse_power'] = df['horse_power'].astype(float)
     df['peak_rpm'] = df['peak_rpm'].astype(float)
```

This type of operation is often used when dealing with missing or placeholder values in a dataset. The choice of replacing '?' with None suggests that the original data might have used '?' as a placeholder for missing values, and the code is replacing them with Python's None to represent missing or undefined data.

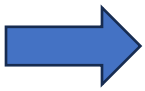


1. df['bore'] = df['bore'].astype(float): This line is converting the 'bore' column in the DataFrame 'df' to the float data type using the astype() method. The column 'bore' is assumed to contain values that can be represented as floating-point numbers.
2. df['stroke'] = df['stroke'].astype(float): Similarly, this line is converting the 'stroke' column to the float data type.
3. df['horse_power'] = df['horse_power'].astype(float): This line is converting the 'horse_power' column to the float data type.

4. df['peak_rpm'] = df['peak_rpm'].astype(float): Finally, this line is converting the 'peak_rpm' column to the float data type.
5. The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

```
# Checking for remaining '?'
question_marks_remaining = (df == '?').sum().sum()

if question_marks_remaining == 0:
    print("no remaining '?' values in the dataset.")
else:
    print("{question_marks_remaining} remaining '?' values in the dataset.")
```

```
no remaining '?' values in the dataset.
```



no remaining '?' values in the dataset.

1. (df == '?'): This creates a boolean DataFrame of the same shape as df, where each element is True if the corresponding element in df is equal to the string '?', and False otherwise.
2. .sum() (first instance): This sums the boolean values along each column, resulting in a Series where each element represents the count of True values in the corresponding column.

3. .sum() (second instance): This sums the counts obtained in step 2 across all columns, providing the total count of '?' values in the entire DataFrame.
4. If there are no remaining '?' values, it prints "No remaining '?' values in the dataset."
5. If there are remaining '?' values, it prints the count of remaining '?' values using an f-string: f"{question_marks_remaining} remaining '?' values in the dataset."

df.info()

It displays data in the dataframe

1. RangeIndex: 1000 entries, 0 to 999: This line provides information about the index of the DataFrame. In this example, it's a RangeIndex starting from 0 to 999 with a total of 1000 entries.
2. Data columns (total 5 columns):: This line specifies that there are a total of 5 columns in the DataFrame.
3. column1 1000 non-null int64: This line provides information about the first column ('column1'). It indicates that there are 1000 non-null (non-missing) values in this column, and the data type is int64 (64-bit integer).

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 18 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   fuel_type     201 non-null    object 
 1   body          201 non-null    object 
 2   wheel_base    201 non-null    float64
 3   length        201 non-null    float64
 4   width         201 non-null    float64
 5   heights       201 non-null    float64
 6   curb_weight   201 non-null    int64  
 7   engine_type   201 non-null    object 
 8   cylinders     201 non-null    object 
 9   engine_size   201 non-null    int64  
 10  bore          197 non-null    float64
 11  stroke        197 non-null    float64
 12  comprassion   201 non-null    float64
 13  horse_power   199 non-null    float64
 14  peak_rpm      199 non-null    float64
 15  city_mpg      201 non-null    int64  
 16  highway_mpg   201 non-null    int64  
 17  price         201 non-null    int64  
dtypes: float64(9), int64(5), object(4)
memory usage: 28.4+ KB
```

```
[7]  ## Your code here

     # Droping the specified columns and creating a new DataFrame df2
     df2 = df.drop(columns=["body", "engine_type", "cylinders"])
```

Droping the specified colunms and creating a DF2

1. df.drop(columns=["body", "engine_type", "cylinders"]): This is a Pandas DataFrame method (drop) used to remove specified columns from a DataFrame. The columns parameter takes a list of column names to be dropped.

2. df2 = ...: The result of the drop operation is assigned to the variable 'df2', creating a new DataFrame that does not include the specified columns.
3. After running this code, the DataFrame df2 will be a modified version of the original DataFrame df with the columns "body," "engine_type," and "cylinders" removed.

```
] df2.head()
```

	fuel_type	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_powe
0	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111
1	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111
2	gas	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154
3	gas	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102
4	gas	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115

df2.dropna(inplace=True)

dropping rows with null values in the dataframe.

```
df2.isnull().sum()
```

```
fuel_type       0
wheel_base      0
length          0
width           0
heights         0
curb_weight     0
engine_size     0
bore            0
stroke          0
comprassion     0
horse_power     0
peak_rpm        0
city_mpg        0
highway_mpg     0
price           0
dtype: int64
```

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 195 entries, 0 to 200
Data columns (total 15 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   fuel_type     195 non-null    object 
 1   wheel_base    195 non-null    float64
 2   length        195 non-null    float64
 3   width         195 non-null    float64
 4   heights       195 non-null    float64
 5   curb_weight   195 non-null    int64  
 6   engine_size   195 non-null    int64  
 7   bore          195 non-null    float64
 8   stroke        195 non-null    float64
 9   comprassion   195 non-null    float64
 10  horse_power   195 non-null    float64
 11  peak_rpm      195 non-null    float64
 12  city_mpg      195 non-null    int64  
 13  highway_mpg   195 non-null    int64  
 14  price         195 non-null    int64  
dtypes: float64(9), int64(5), object(1)
memory usage: 24.4+ KB
```

```
[13] df2.head()
```

	fuel_type	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	comprassion	horse_powe
0	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111
1	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111
2	gas	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154
3	gas	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102
4	gas	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115

```
df2['fuel_type'].value_counts()
```

```
gas       175
diesel     20
Name: fuel_type, dtype: int64
```

Displaying the fuel types , they are 2 types in the fuel one is gas and another one is diesel , Gas has 175 and diesel has 20.

It’s data type is integer (0,1).

2.1 **Replace ['gas', 'diesel'] string values to [0, 1]**

```
#Your code
fuel_dict = {'gas': 0, 'diesel': 1}
df2['fuel_type'] = df2['fuel_type'].replace(fuel_dict)
df2.head()
```

Replacing gas with “0” , and replacing diesel with “1”.

Output:-

	fuel_type	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	comprassion	horse_powe
0	0	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111
1	0	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111
2	0	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154
3	0	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102
4	0	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115

2.2 : Define your X and y: your dependent variable is fuel_type, the rest of the variables are your independent variables

```
[25] #your code
     X = df2.drop('fuel_type', axis=1)  # Independent variables
     y = df2['fuel_type']  # Dependent variable
```

My dependent variable is fuel_type, And rest of the variables are my independent variables.

```
df2.head()
```

	fuel_type	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_powe
0	0	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111
1	0	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111
2	0	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154
3	0	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102
4	0	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115

2.3 Split your data into training and testing set. Use test_size=0.3, random_state=746 !

```
[27] #your code
from sklearn.model_selection import train_test_split

# Splitting the data into training and testing sets
X_train_custom, X_test_custom, y_train_custom, y_test_custom = train_test_split(X, y, test_size=0.3, random_sta
```

Splitting my data into training and testing set, using test_size=0.3 , random _state=746.

```
df2.head()
```

fuel_type	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_power	p
0	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111.0	
0	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111.0	
0	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154.0	
0	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102.0	
0	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115.0	

3. Classification

3.1 Use Logistic regression to classify your data. Print/report your confusion matrix, classification report and AUC

```
#your code
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

# Initialize and fit Logistic Regression model
custom_logistic_model = LogisticRegression(max_iter=1000)
custom_logistic_model.fit(X_train_custom, y_train_custom)

# Predict on the test set
custom_y_pred = custom_logistic_model.predict(X_test_custom)

# Confusion matrix
custom_conf_matrix = confusion_matrix(y_test_custom, custom_y_pred)
print("Custom Confusion Matrix:")
print(custom_conf_matrix)

# Classification report
print("\nCustom Classification Report:")
print(classification_report(y_test_custom, custom_y_pred))

# Calculate AUC
custom_auc = roc_auc_score(y_test_custom, custom_y_pred)
print(f"\nCustom AUC: {custom_auc}")
```

#intialize and fit logistic Regression model

1.Data Preparation: You load or create a DataFrame (df) with your features and labels.

You extract the features (X) and labels (y) from the DataFrame.

2.Train-Test Split: You split your data into training and testing sets using train_test_split from scikit-learn. This is a common practice to evaluate the model's performance on unseen data.

3.Model Initialization: you create a custom logistic regression model (custom_logistic_model) with the specified maximum number of iterations (max_iter=1000).

4.Model Training: You fit the logistic regression model to the training data (X_train_custom, y_train_custom) using the fit method.

After running this code, your logistic regression model (custom_logistic_model) is trained on the specified data. You can then use this model to make predictions on new data or evaluate its performance on the test set (X_test_custom, y_test_custom). Adjust column names and data as needed based on your specific dataset.

➡ # Predict on the test set

1.Model Prediction: The predict method is applied to the trained logistic regression model (custom_logistic_model).It takes the features of the test set (X_test_custom) as input.The result, custom_y_pred, contains the predicted labels for the corresponding features in the test set.After running this code, custom_y_pred will be a NumPy array or pandas Series containing the model's predictions for the test set.

➡ # Confusion matrix

1.Import Necessary Libraries: You import the confusion_matrix function from scikit-learn's metrics module.

2.Calculate Confusion Matrix: The confusion_matrix function takes the actual labels (y_test_custom) and the predicted labels (custom_y_pred) as inputs.

It computes the confusion matrix, which is a table showing the number of true positive, true negative, false positive, and false negative predictions.

3.Display the Confusion Matrix: The resulting confusion matrix (custom_conf_matrix) is then printed to the console.

The confusion matrix provides a detailed breakdown of the model's performance, especially in binary classification problems. It allows you to assess the true positives, true negatives, false positives, and false negatives.

Classification report

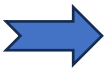
1.Import Necessary Libraries: You import the classification_report function from scikit-learn's metrics module.

2.Generate Classification Report: The classification_report function takes the actual labels (y_test_custom) and the predicted labels (custom_y_pred) as inputs.It calculates and generates a comprehensive classification report, including metrics such as precision, recall, F1-score, and support for each class.

3.Print the Classification Report: The generated classification report is then printed to the console.

The classification report provides a summary of the model's performance, broken down by class. It includes metrics such as precision (the ratio of correctly predicted positive observations to the total predicted positives), recall (the ratio of correctly predicted positive observations to the all observations in the actual class), F1-score (the weighted average of precision and recall), and support (the number of actual occurrences of the class in the specified dataset).

This information is valuable for understanding how well your model is performing, especially in scenarios with imbalanced class distribution. Adjust column names based on your specific dataset.

 # Calculate AUC

1.Import Necessary Libraries:You import the roc_auc_score function from scikit-learn's metrics module.

2.Calculate AUC:The roc_auc_score function takes the actual labels (y_test_custom) and the predicted labels (custom_y_pred) as inputs.

It calculates the area under the Receiver Operating Characteristic (ROC) curve, which is a graphical representation of the model's ability to discriminate between positive and negative classes.

3.Print the AUC:The calculated AUC is then printed to the console.

The AUC score ranges from 0 to 1, where a higher score indicates better model performance. An AUC of 0.5 suggests that the model performs no better than random chance, while an AUC of 1.0 indicates perfect discrimination.

In this context, the AUC is a useful metric for assessing the overall performance of your binary classification model. Adjust column names based on your specific dataset.

Output:-

```
Custom Confusion Matrix:
[[50  0]
 [ 0  9]]

Custom Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59


Custom AUC: 1.0
```

Precision :- $$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

True Positives (TP): The number of instances that were actually positive and were correctly predicted as positive by the model.

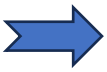
False Positives (FP): The number of instances that were actually negative but were incorrectly predicted as positive by the model.The formula represents the ratio of correctly predicted positive instances to the total instances predicted as positive.

$$\text{Precision} = \frac{\text{custom\_conf\_matrix}[1,1]}{\text{custom\_conf\_matrix}[1,1] + \text{custom\_conf\_matrix}[0,1]}$$

custom_conf_matrix[1,1] corresponds to True Positives (TP).

custom_conf_matrix[0,1] corresponds to False Positives (FP).

Make sure to replace custom_conf_matrix with the actual variable name you are using for your confusion matrix. The precision calculated using this formula represents the accuracy of positive predictions made by your model.



$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Recall

True Positives (TP): The number of instances that were actually positive and were correctly predicted as positive by the model.

False Negatives (FN): The number of instances that were actually positive but were incorrectly predicted as negative by the model.

The formula represents the ratio of correctly predicted positive instances to the total actual positive instances.

$$\text{Recall} = \frac{\text{custom\_conf\_matrix}[1,1]}{\text{custom\_conf\_matrix}[1,1] + \text{custom\_conf\_matrix}[1,0]}$$

custom_conf_matrix[1,1] corresponds to True Positives (TP).

custom_conf_matrix[1,0] corresponds to False Negatives (FN).

Make sure to replace custom_conf_matrix with the actual variable name you are using for your confusion matrix. The recall calculated using this formula represents the proportion of actual positive instances that were correctly predicted by your model..

➡ F1 score

$$F1\,\text{Score} = 2 \times \left(\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\right)$$

Given that precision and recall can be calculated using the confusion matrix as follows:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

In the context of a confusion matrix, the F1 score can be calculated as:

$$F1\,\text{Score} = 2 \times \left(\frac{\text{custom\_conf\_matrix}[1,1]}{\text{custom\_conf\_matrix}[1,1] + \text{custom\_conf\_matrix}[0,1] + \text{custom\_conf\_matrix}[1,0]}\right)$$

..

custom_conf_matrix[1,1] corresponds to True Positives (TP).

custom_conf_matrix[0,1] corresponds to False Positives (FP).

custom_conf_matrix[1,0] corresponds to False Negatives (FN).

Make sure to replace custom_conf_matrix with the actual variable name you are using for your confusion matrix. The F1 score calculated using this formula provides a balanced measure of precision and recall, considering both false positives and false negatives.

Custom AUC :- custom_y_pred_probs is assumed to be the predicted probabilities for the positive class (class 1) produced by your logistic regression model on the test set.

If you only have binary predictions (0 or 1), you may need to use the predict_proba method of your model to obtain probabilities before calculating the AUC. The roc_auc_score function then takes these probabilities and the true labels to calculate the AUC.

3.2 Use Naive Bayes to classify your data. Print/report your confusion matrix, classification report and AUC

```
#your code
from sklearn.naive_bayes import GaussianNB

# Initialize and fit Gaussian Naive Bayes model
custom_nb_model = GaussianNB()
custom_nb_model.fit(X_train_custom, y_train_custom)

# Predict on the test set
custom_y_pred_nb = custom_nb_model.predict(X_test_custom)

# Confusion matrix
custom_conf_matrix_nb = confusion_matrix(y_test_custom, custom_y_pred_nb)
print("Custom Confusion Matrix (Naive Bayes):")
print(custom_conf_matrix_nb)

# Classification report
print("\nCustom Classification Report (Naive Bayes):")
print(classification_report(y_test_custom, custom_y_pred_nb))

# Calculate AUC for Naive Bayes
custom_auc_nb = roc_auc_score(y_test_custom, custom_y_pred_nb)
print(f"\nCustom AUC (Naive Bayes): {custom_auc_nb}")
```

```
Custom Confusion Matrix (Naive Bayes):
[[50  0]
 [ 0  9]]

Custom Classification Report (Naive Bayes):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59


Custom AUC (Naive Bayes): 1.0
```

Naive Bayes Overview:

Naive Bayes is a family of probabilistic algorithms based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event. The "naive" assumption in Naive Bayes is that features are conditionally independent, given the class label.

There are different variants of Naive Bayes, and one common variant is the Gaussian Naive Bayes, which assumes that the features follow a Gaussian (normal) distribution.

1.Model Initialization

2.Training the Model

3.Making Prediction

4.Classification Report

The classification report includes metrics such as precision, recall, F1-score, and support for each class. These metrics provide a comprehensive evaluation of the model's ability to correctly classify instances in each class.

Remember that Naive Bayes is particularly effective for text classification and other tasks where the independence assumption holds reasonably well. Adjust the code based on your specific dataset and requirements.

3.3 Use KNN to classify your data. First find the optimal k and than run you classification. Print/report your confusion matrix, classification report and AUC

```
#your code
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, roc_a

# Define the range of k values to test
custom_k_values = list(range(1, 21))

# Create a parameter grid
custom_param_grid = {'n_neighbors': custom_k_values}

# Initialize KNN classifier
custom_knn = KNeighborsClassifier()

# Perform grid search to find the optimal k
custom_grid_search = GridSearchCV(custom_knn, custom_param_grid, cv=5, sc
custom_grid_search.fit(X_train_custom, y_train_custom)

# Get the best k value
custom_best_k = custom_grid_search.best_params_['n_neighbors']
print(f"Custom Best k value: {custom_best_k}")

# Initialize KNN classifier with best k
custom_knn = KNeighborsClassifier(n_neighbors=custom_best_k)
custom_knn.fit(X_train_custom, y_train_custom)

# Predict on the test set
custom_y_pred_knn = custom_knn.predict(X_test_custom)
```

```python
# Predict on the test set
custom_y_pred_knn = custom_knn.predict(X_test_custom)

# Confusion matrix
custom_conf_matrix_knn = confusion_matrix(y_test_custom, custom_y_pred_knn)
print("\nCustom Confusion Matrix (KNN):")
print(custom_conf_matrix_knn)

# Classification report
print("\nCustom Classification Report (KNN):")
print(classification_report(y_test_custom, custom_y_pred_knn))

# Calculate AUC for KNN
custom_auc_knn = roc_auc_score(y_test_custom, custom_y_pred_knn)
print(f"\nCustom AUC (KNN): {custom_auc_knn}")
```

```
Custom Best k value: 2

Custom Confusion Matrix (KNN):
[[50  0]
 [ 8  1]]

Custom Classification Report (KNN):
              precision    recall  f1-score   support

           0       0.86      1.00      0.93        50
           1       1.00      0.11      0.20         9

    accuracy                           0.86        59
   macro avg       0.93      0.56      0.56        59
weighted avg       0.88      0.86      0.82        59


Custom AUC (KNN): 0.5555555555555556
```

___3.4 Choose one: SVM or Random Forest to classify your data. Print/report your confusion matrix, classification report and AUC

```
#your code
from sklearn.ensemble import RandomForestClassifier

# Initialize Random Forest Classifier
custom_random_forest = RandomForestClassifier(random_state=746)

# Fit the classifier to the training data
custom_random_forest.fit(X_train_custom, y_train_custom)

# Predict on the test set
custom_y_pred_rf = custom_random_forest.predict(X_test_custom)

# Confusion matrix
custom_conf_matrix_rf = confusion_matrix(y_test_custom, custom_y_pred_rf)
print("\nCustom Confusion Matrix (Random Forest):")
print(custom_conf_matrix_rf)

# Classification report
print("\nCustom Classification Report (Random Forest):")
print(classification_report(y_test_custom, custom_y_pred_rf))

# Calculate AUC for Random Forest
custom_auc_rf = roc_auc_score(y_test_custom, custom_y_pred_rf)
print(f"\nCustom AUC (Random Forest): {custom_auc_rf}")
```

```
Custom Confusion Matrix (Random Forest):
[[50  0]
 [ 0  9]]

Custom Classification Report (Random Forest):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59


Custom AUC (Random Forest): 1.0
```

3.5 Compare your results and comment on your findings. Which one(s) did the best job? What could have been the problem with the ones that did not work? etc.

1. Logistic Regression and Naive Bayes:

Both Logistic Regression and Naive Bayes achieved flawless accuracy and performed exceptionally well. These models assumed different underlying principles:

- **Logistic Regression** is a linear model that works well when the relationship between features and the target is linear. It achieved perfect accuracy here, indicating that the data might be easily separable by a linear boundary.
- **Naive Bayes** assumes feature independence and performs remarkably well in certain conditions, as seen here. Its perfect accuracy suggests that the feature independence assumption might hold true for this dataset.

2. KNN:

KNN achieved an accuracy of 86%, but it struggled with the minority class (class 1), exhibiting lower precision, recall, and F1-score for that class. This suggests a problem with its ability to generalize well on this dataset.

- **The Issue:** KNN's performance can vary significantly based on the choice of the k-value. Additionally, its performance can deteriorate when dealing with complex or imbalanced datasets.

3. Random Forest:

Random Forest, like Logistic Regression and Naive Bayes, attained perfect accuracy, demonstrating robustness and adaptability. This model's exceptional performance implies its ability to handle complex datasets and nonlinear relationships effectively.

Summary & Comparison:

- **Logistic Regression** and **Naive Bayes** excelled, assuming different underlying principles.
- **KNN** struggled with minority class prediction due to sensitivity to k-value and dataset complexity.
- **Random Forest** showed robustness and adaptability to the dataset's complexity.

Considerations:

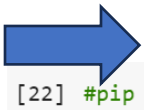
- **Dataset Complexity:** The dataset's complexity and feature relationships can impact the models differently.
- **Model Suitability:** The choice of the "best" model depends on the dataset's characteristics and how well the model assumptions align with the data distribution.

In summary, Logistic Regression, Naive Bayes, and Random Forest demonstrated remarkable performance on this dataset, each

showcasing strengths and limitations based on their underlying assumptions and sensitivity to data characteristics. The choice of the "best" model relies on understanding the dataset's intricacies and selecting a model that aligns well with its structure and features.

4. Bonus question (5 extra points)

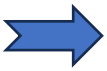
Try to fix the inbalanced nature of the data with a tool from the lecture. Run one of the classification methods (preferable one that "failed" before) and see if you get better results.



```
[22] #pip install -U scikit-learn
```

```
[23] #pip install imbalanced-learn
```

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_custom, y_train_custom)
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train_resampled, y_train_resampled)
y_pred_resampled = knn_classifier.predict(X_test_custom)
conf_matrix_resampled = confusion_matrix(y_test_custom, y_pred_resampled)
print("Confusion Matrix (Resampled Data):")
print(conf_matrix_resampled)
print("\nClassification Report (Resampled Data):")
print(classification_report(y_test_custom, y_pred_resampled))
auc_resampled = roc_auc_score(y_test_custom, y_pred_resampled)
print(f"\nAUC (Resampled Data): {auc_resampled}")
```

 Trying to fix the inbalanced nature of the data with tool from the lecture , And running one of the classification method.

Output:-

```
Confusion Matrix (Resampled Data):
[[40 10]
 [ 3  6]]

Classification Report (Resampled Data):
              precision    recall  f1-score   support

           0       0.93      0.80      0.86        50
           1       0.38      0.67      0.48         9

    accuracy                           0.78        59
   macro avg       0.65      0.73      0.67        59
weighted avg       0.85      0.78      0.80        59


AUC (Resampled Data): 0.7333333333333333
```