MACHINE LEARNING PROJECT

Data Visualization, Linear Regression, and Classification

This project explores key concepts in machine learning, including data visualization for insightful patterns, linear regression for predictive modeling, and classification techniques for decision-making. Through practical implementations, it demonstrates the application of these methods in solving real-world problems.

Prepared by:

Farah AL-Fuqaha'a

Dana Salman

Farah AL-Hessa

Jana AL-Araishy

Raghad AL-Tamary

Instructor: Dr. Esra'a AL-Shdaifat

Introduction

This project explores the application of machine learning techniques on two distinct datasets: a CO2 emissions dataset and a loans dataset. Our goal is to analyze, visualize, and build predictive models to gain insights and draw meaningful conclusions.

For the CO2 emissions dataset, we focused on understanding the relationship between different variables and CO2 emissions. We applied data visualization techniques to uncover patterns and trends. Subsequently, we implemented linear regression to predict CO2 emissions based on selected features. To enhance model performance, we rebuilt the regression model using forward selection for feature optimization.

For the loans dataset, the focus was on classifying loan applications into different risk categories. We employed a range of classification algorithms, including decision trees, support vector machines (SVM), logistic regression, ensemble methods, and Naive Bayes. The performance of these models was evaluated to determine the most effective approach for accurate classification.

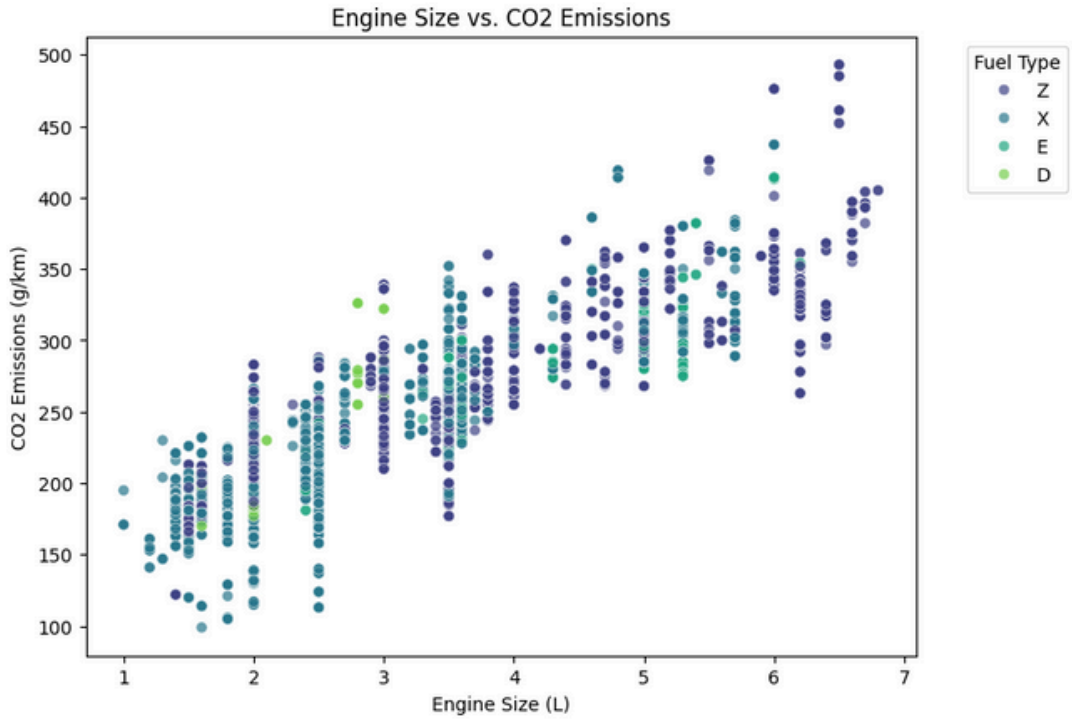
This document details the methodology, results, and insights from our work, highlighting the potential of machine learning in solving real-world problems in environmental and financial domains.

Task 1: Visualization

We started with importing the import matplotlib and seaborn libraries along with pandas.

1.1

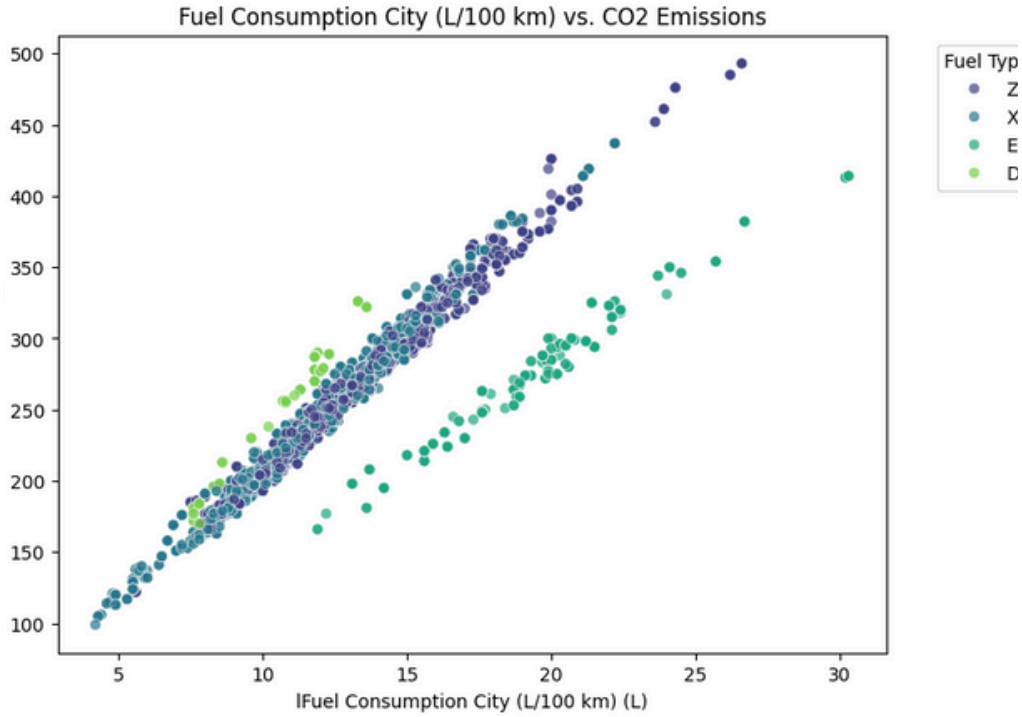
visualizing the relationship between **Engine Size** and **co2 Emission** with a scatter plot.
the x-axis represents engine size, and the y-axis represents CO_2 emissions. Each data point is color-coded based on fuel type (e.g., gasoline, diesel). The data shows positive correlation between the features that when the size of engine increases the co2 emission increases as well.



Task 1: Visualization

1.2

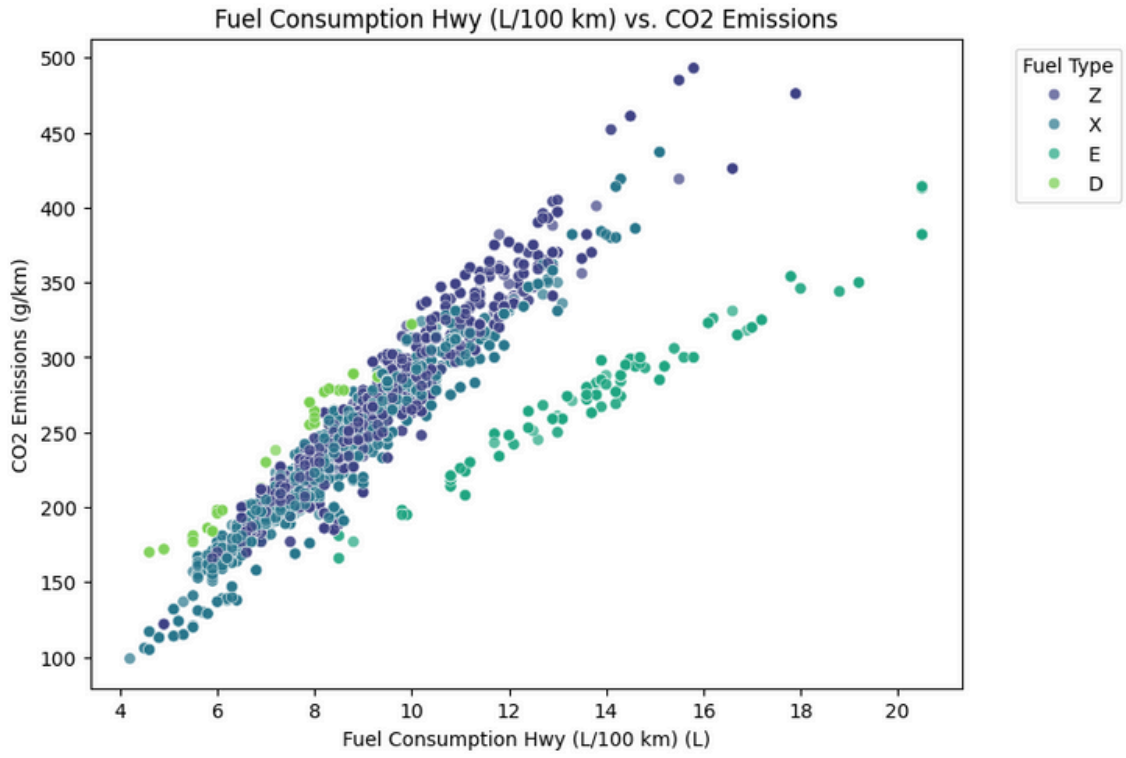
Visualizing the relationship between **Fuel Consumption in city** and **CO2 Emission** with a scatter plot. The x-axis represents fuel consumption and y-axis represents CO2 emission. Each data point is color-coded based on fuel type (e.g., gasoline, diesel). The scatter plot reveals a strong positive correlation between fuel consumption and CO_2 emissions, with the data points forming a near-linear trend. This indicates that as city fuel consumption increases, CO_2 emissions also increase.



Task 1: Visualization

1.3

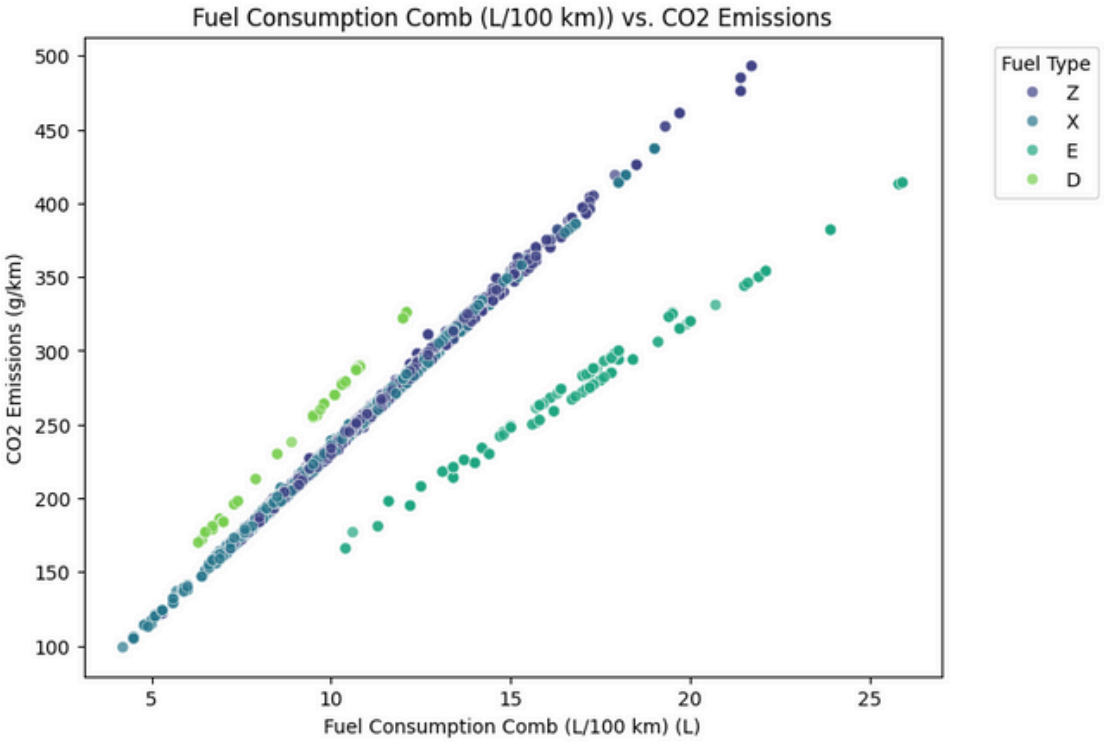
Visualizing the relationship between **Fuel Consumption in Highways** and **CO2 Emission** with a scatter plot. The x-axis represents fuel consumption and y-axis represents CO2 emission. Each data point is color-coded based on fuel type (e.g., gasoline, diesel). The scatter plot reveals a strong positive correlation between fuel consumption and CO_2 emissions. This indicates that as Highways fuel consumption increases, CO_2 emissions also increase.



Task 1: Visualization

1.4

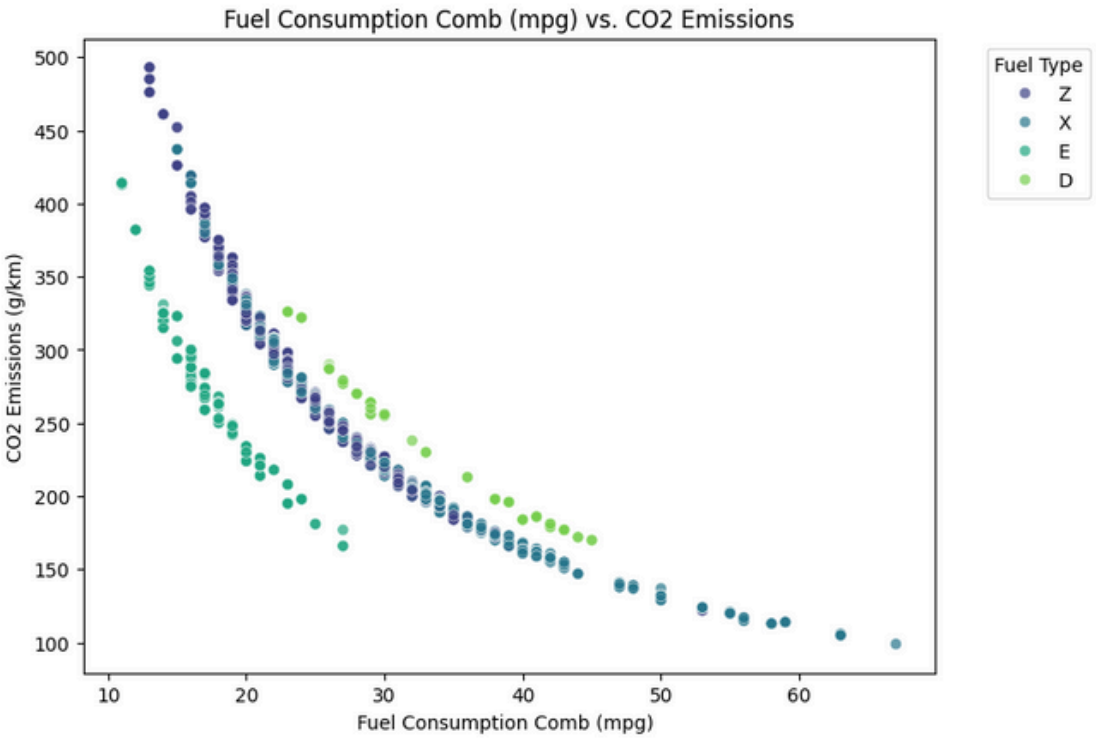
Visualizing the relationship between **Fuel Consumption combined (city and highway)** and **CO2 Emission** with a scatter plot. The x-axis represents fuel consumption and y-axis represents CO2 emission. Each data point is color-coded based on fuel type (e.g., gasoline, diesel). The scatter plot reveals a strong positive correlation between fuel consumption and CO_2 emissions, with dots creating near-linear trend, This indicates that as Highways and city fuel consumption increases at both city and highway, CO_2 emissions also increase.



Task 1: Visualization

1.5

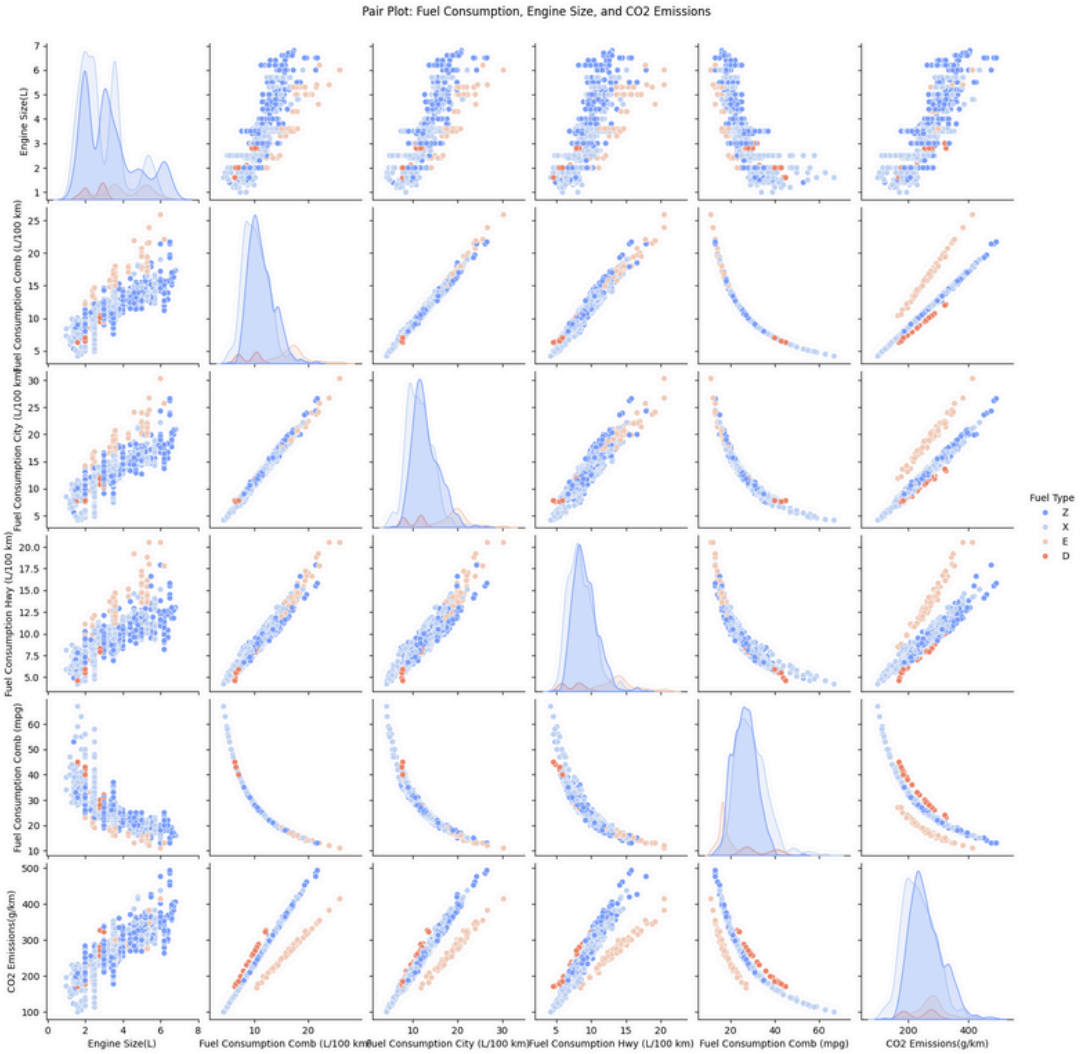
Visualizing the relationship between **Fuel Consumption of miles per imperial gallon** and **CO2 Emission** with a scatter plot. The x-axis represents fuel consumption (miles per imperial gallon), and the y-axis represents CO_2 emissions (g/km). Each data point is color-coded based on fuel type (e.g., gasoline, diesel), The scatter plot reveals a negative correlation: as fuel consumption of miles per imperial gallon increases (indicating greater fuel efficiency), CO_2 emissions decrease.



Task 1: Visualization

1.6

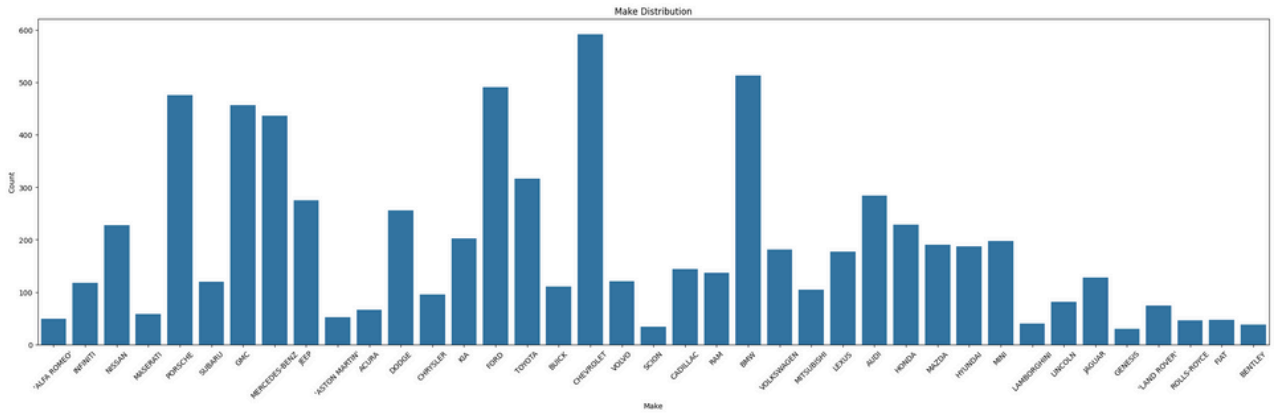
This pair plot visualizes the relationships between **engine size**, **various fuel consumption metrics (city, highway, combined in both L/100 km and mpg)**, and **CO_2 emissions**. The data points are color-coded by fuel type, enabling the comparison of patterns across categories like gasoline and diesel vehicles. Diagonal KDE plots show the distribution of individual variables, while the scatter plots highlight pairwise relationships, such as the strong correlation between fuel consumption and CO_2 emissions.



Task 1: Visualization

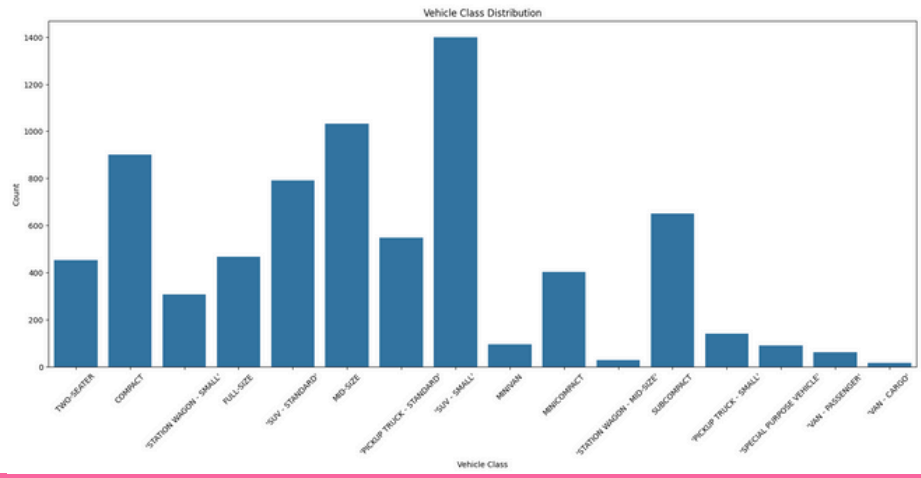
2.1

Visualize the distribution of the **Make feature**, which represents the brands of the vehicles with count plot . The count plot shows the number of vehicles for each brand in the dataset. It highlights that chevrloet vehicles have the highest representation, with almost 600 entries.



2.2

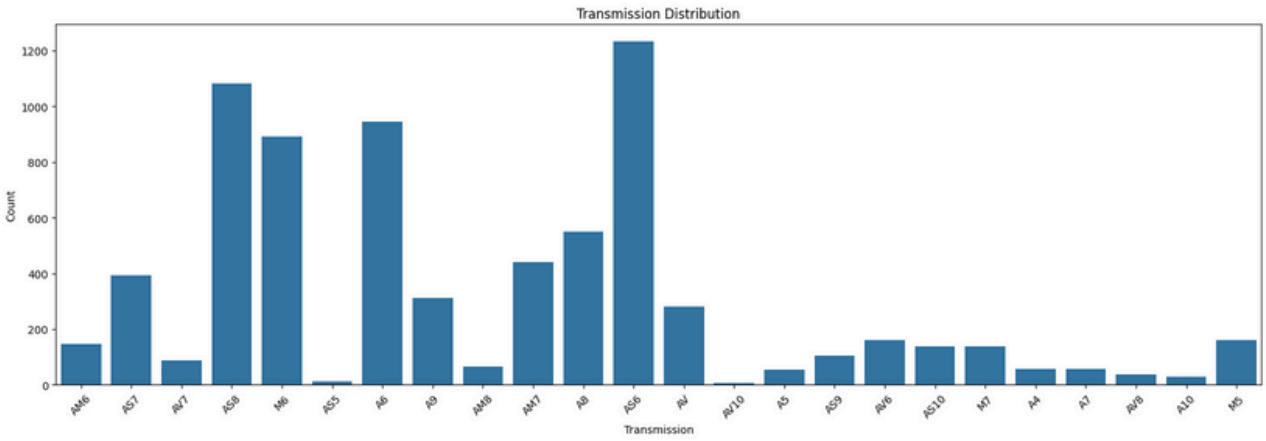
Visualize the distribution of the **Vehicle class feature**, which represents category of vehicles based on their size with count plot. The count plot shows the frequency of each category in the dataset. It highlights that suv-small vehicles have the highest representation, with almost 1400 entries.



Task 1: Visualization

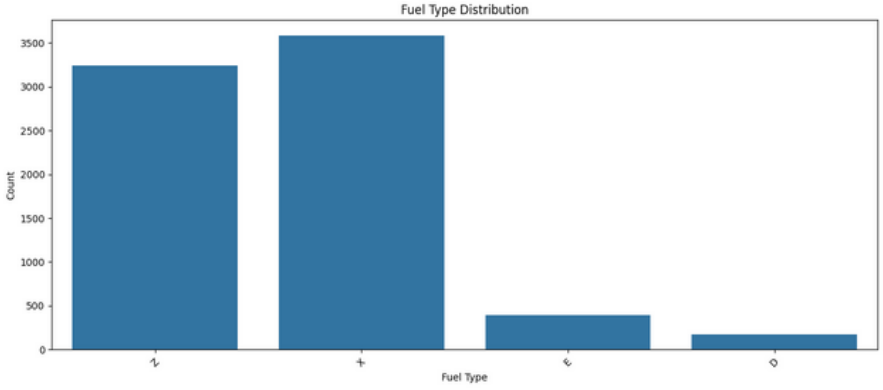
2.3

Visualize the distribution of the **Transmission feature**, which represents transmission system used in the vehicle with count plot. The count plot shows the frequency of each transmission system in the dataset. It highlights that As6 system have the highest representation, with almost 1200 entries.



2.4

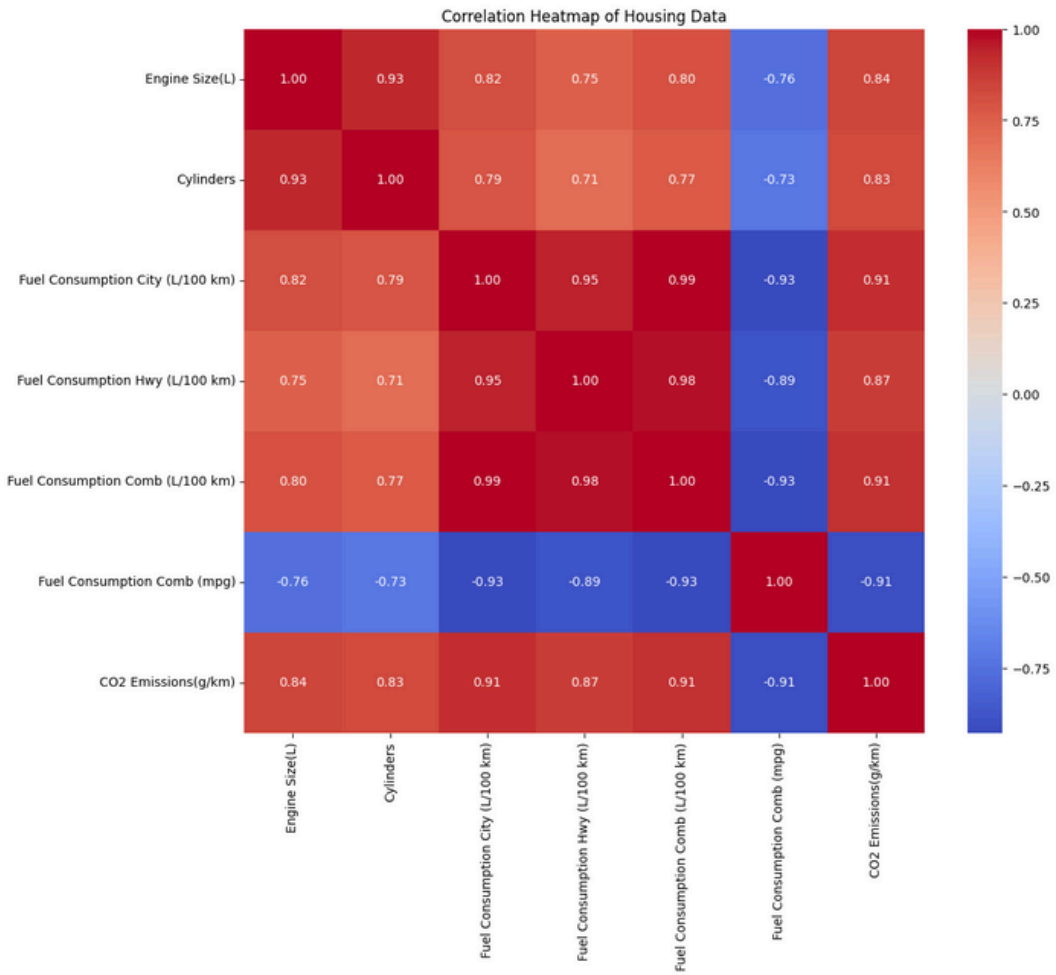
Visualize the distribution of the **Fuel Type feature**, which represents the types of fuel with count plot. The count plot shows the frequency of each type of fuel in the dataset. It highlights that Regular gasoline (X) have the highest representation, with almost over 3500 entries.



Task 1: Visualization

3

Visualize the correlation between **numerical features** with a heatmap. The **heatmap** displays the correlation coefficients between each pair of features, with numerical values in each cell. A positive correlation will be indicated by higher values, while negative correlations will be represented by lower values (below zero). The heatmap provides a clear representation of how the numerical features are related, helping to identify both strong and weak correlations, it shows that there is a remarkable negative correlations between Fuel Consumption per imperial gallon and the other features and a high positive correlation between Fuel Consumption in city and Fuel Consumption highway and Fuel Consumption combined.



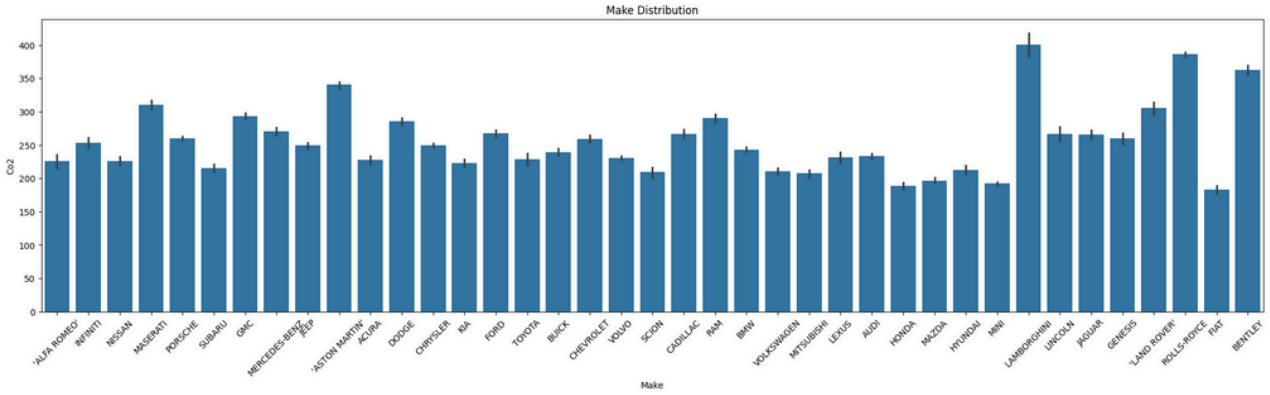
Task 1: Visualization

4

Identify key features contributing to CO_2 emission depending on Bar plot for categorical data and Heatmap for numerical data

4.1

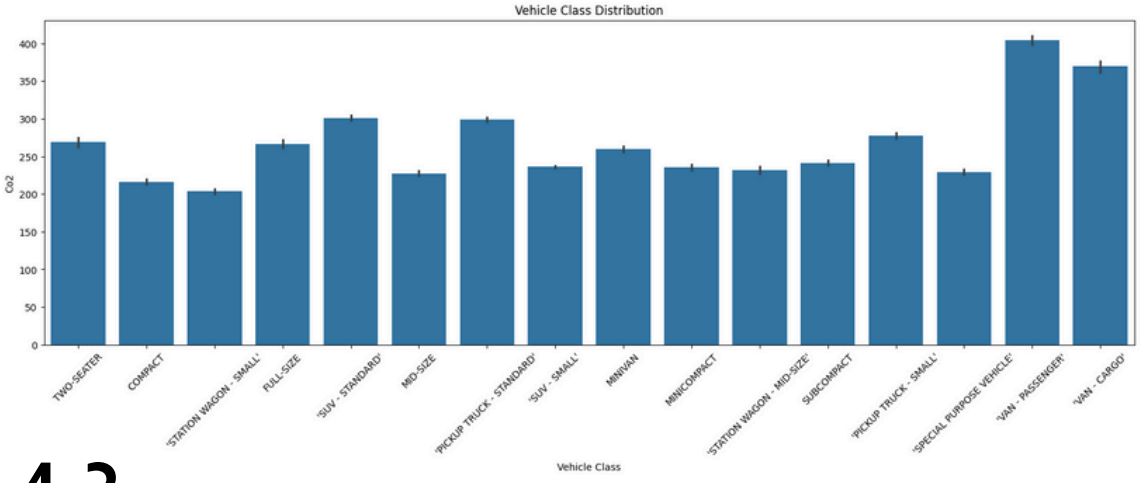
Visualizing the brands frequency of **vehicles**, the x-axis represents the brands and the y-axis represents co2 emission, the graph shows that the most frequent brand is Lamborgeni which makes the Lamborgeni Brand the most contributive brand in co2 emission.



Task 1: Visualization

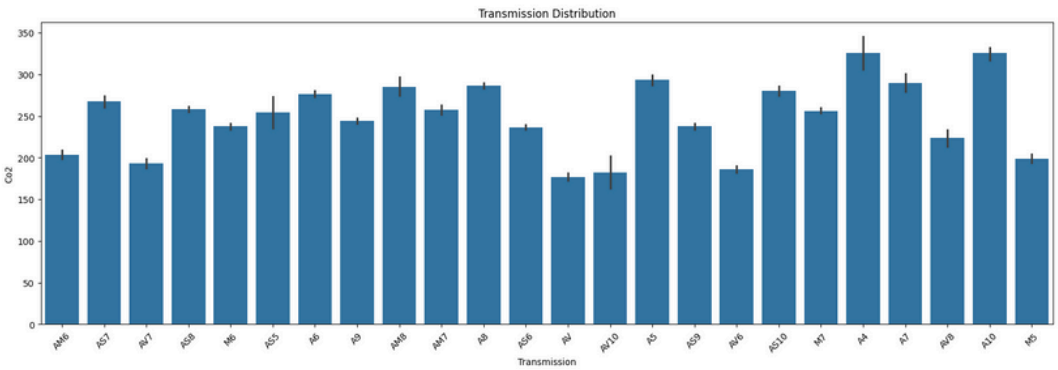
4.2

visualizing the frequency of **vehicles classes** ,the x-axis represents the classes and the y-axis represents co2 emission, the graph shows that the most frequent class is Van-passenger vehicle which makes the Van-passenger vehicle class the most contributive class in co2 emission.



4.3

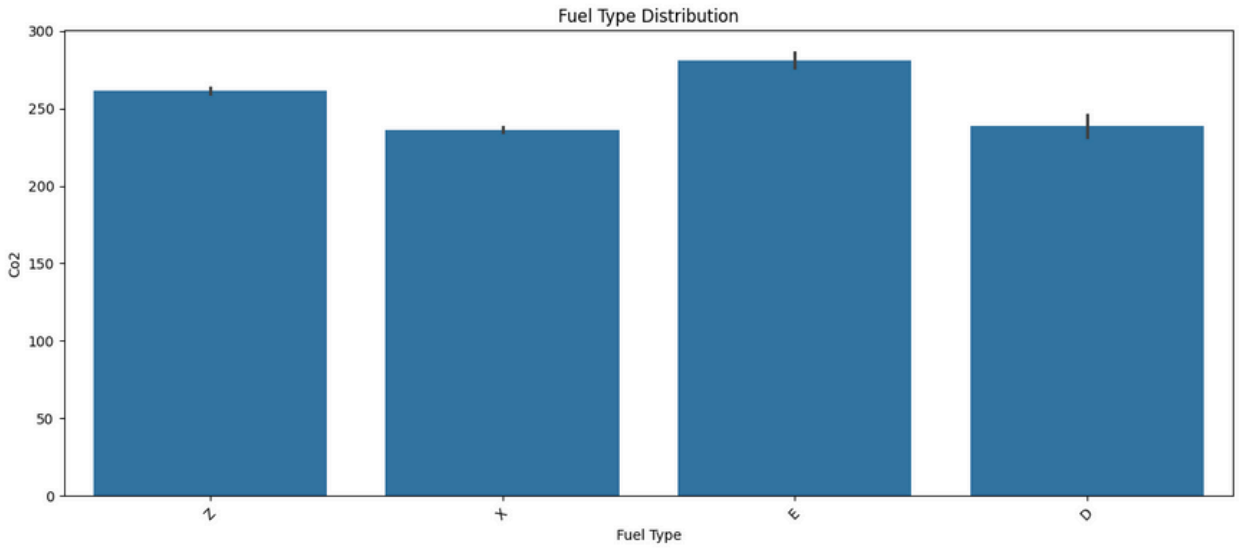
visualizing the frequency of **Transmission system** ,the x-axis represents the Transmission systems and the y-axis represents co2 emission, the graph shows that the most frequent system is A4 which makes the A4 system the most contributive system in co2 emission.



Task 1: Visualization

4.4

visualizing the frequency of Fuel types, the x-axis represents the fuel types and the y-axis represents co2 emission, the graph shows that the most frequent type is E which makes the E type the most contributive system in co2 emission.



Task 2: Linear Regression

Data Preprocessing and Exploration

1. Column Name Cleaning:
Remove leading/trailing spaces and quotes from column names

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)
0	'ALFA ROMEO'	'4C Coupe'	TWO-SEATER	1.8	4	AM6	Z	9.7	6.9	8.4	34	197
1	INFINITI	'Q50 HYBRID AWD'	COMPACT	3.5	6	AS7	Z	9.1	7.7	8.5	33	200
2	NISSAN	'JUKE AWD'	'STATION WAGON - SMALL'	1.6	4	AV7	Z	8.8	7.5	8.2	34	194
3	MASERATI	'QUATTROPORTE GTS'	FULL-SIZE	3.8	8	AS8	Z	17.6	10.7	14.5	19	334
4	PORSCHE	CAYENNE	'SUV - STANDARD'	3.6	6	M6	Z	15.8	10.9	13.6	21	313
...	...	...	...	...	...	...	...	...	...	...	...	...
7380	RAM	'1500 4X4 FFV'	'PICKUP TRUCK - STANDARD'	3.6	6	A8	E	20.7	14.7	18.0	16	300
7381	PORSCHE	'Panamera GTS ST'	FULL-SIZE	4.0	8	AM8	Z	15.7	10.5	13.4	21	313

✓ 0s completed at 01:09

2.Data Types Overview:
Display the data types of all columns to understand the structure and identify numerical and categorical features.

```
Make                              object
Model                             object
Vehicle Class                     object
Engine Size(L)                   float64
Cylinders                          int64
Transmission                      object
Fuel Type                         object
Fuel Consumption City (L/100 km)  float64
Fuel Consumption Hwy (L/100 km)   float64
Fuel Consumption Comb (L/100 km)  float64
Fuel Consumption Comb (mpg)        int64
CO2 Emissions(g/km)                int64
dtype: object
```

Task 2: Linear Regression

3.Dataset Information:

Provide an overview of the dataset, including column names, non-null counts, and memory usage.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7385 entries, 0 to 7384
Data columns (total 12 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Make                              7385 non-null   object
 1   Model                             7385 non-null   object
 2   Vehicle Class                     7385 non-null   object
 3   Engine Size(L)                    7385 non-null   float64
 4   Cylinders                         7385 non-null   int64
 5   Transmission                      7385 non-null   object
 6   Fuel Type                         7385 non-null   object
 7   Fuel Consumption City (L/100 km)  7385 non-null   float64
 8   Fuel Consumption Hwy (L/100 km)   7385 non-null   float64
 9   Fuel Consumption Comb (L/100 km)  7385 non-null   float64
 10  Fuel Consumption Comb (mpg)       7385 non-null   int64
 11  CO2 Emissions(g/km)               7385 non-null   int64
dtypes: float64(4), int64(3), object(5)
memory usage: 692.5+ KB
```

4.Missing Values Analysis:

Identify missing values in the dataset for further handling and imputation.

```
        Make  Model  Vehicle Class  Engine Size(L)  Cylinders  Transmission  \
0      False  False          False           False      False         False
1      False  False          False           False      False         False
2      False  False          False           False      False         False
3      False  False          False           False      False         False
4      False  False          False           False      False         False
...      ...    ...            ...             ...        ...           ...
7380   False  False          False           False      False         False
7381   False  False          False           False      False         False
7382   False  False          False           False      False         False
7383   False  False          False           False      False         False
7384   False  False          False           False      False         False

      Fuel Type  Fuel Consumption City (L/100 km)  \
0         False                             False
1         False                             False
2         False                             False
3         False                             False
4         False                             False
...         ...                               ...
7380      False                             False
7381      False                             False
7382      False                             False
7383      False                             False
7384      False                             False
```

Task 2: Linear Regression

5.Statistical Summary:
Generate descriptive statistics for numerical columns, including mean, standard deviation, minimum, and maximum values.

	Engine Size(L)	Cylinders	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)
count	7385.000000	7385.000000	7385.000000	7385.000000	7385.000000	7385.000000	7385.000000
mean	3.154746	5.573053	12.527163	9.053297	10.963643	27.402573	249.912525
std	1.326801	1.787038	3.470961	2.234606	2.879662	6.918796	56.405149
min	1.000000	3.000000	4.200000	4.200000	4.200000	11.000000	99.000000
25%	2.000000	4.000000	10.100000	7.600000	9.000000	23.000000	209.000000
50%	3.000000	6.000000	12.000000	8.700000	10.500000	27.000000	244.000000
75%	3.600000	6.000000	14.400000	10.200000	12.500000	31.000000	285.000000
max	6.800000	12.000000	30.300000	20.500000	25.900000	67.000000	493.000000

6.Describe Categorical Data:
Display unique, top and aggregate the highest value.

	Make	Model	Vehicle Class	Transmission	Fuel Type
count	7385	7385	7385	7385	7385
unique	39	997	16	24	4
top	CHEVROLET	'JOURNEY FFV'	'SUV - SMALL'	AS6	X
freq	592	42	1399	1232	3582

```
print(data['Fuel Type'].unique())
```

```
['Z' 'X' 'E' 'D']
```

Task 2: Linear Regression

2.1 Model Construction:

2.1.1 Split Dataset

The dataset was divided into training (70%) and testing (30%) subsets using train_test_split function with a random_state value of 42 to ensure consistency. Features were then standardized using StandardScaler to achieve a mean of 0 and a standard deviation of 1, ensuring consistent feature scaling for better model performance

```
[ ] x
```

	Engine Size(L)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	Cylinders
0	1.8	8.4	34	4
1	3.5	8.5	33	6
2	1.6	8.2	34	4
3	3.8	14.5	19	8
4	3.6	13.6	21	6
...	...	...	...	...
7380	3.6	18.0	16	6
7381	4.0	13.4	21	8
7382	2.0	9.5	30	4
7383	2.0	9.1	31	4
7384	6.0	15.7	18	12

```
[ ] y

0       197
1       200
2       194
3       334
4       313
       ...
7380    300
7381    313
7382    223
7383    209
7384    364
Name: CO2 Emissions(g/km), l
```

Task 2: Linear Regression

2.1.2 Build a Multiple Linear Regression model

First, we used the LinearRegression class from sklearn.linear_model to create and train the model with our training data (x_train and y_train). Once trained, we used the model to predict outcomes on the testing dataset (x_test), and the predicted values were stored in y_predict.

2.2 Feature Selection:

2.2.1 Forward technique

Second model we tried to use all numerical features inoreder to see the difference between the 2 models MSE ans R2:

```
Selected Features: Index(['Fuel Consumption Comb (mpg)', 'Cylinders'], dtype='object')
```

2.2.2 Rebuild the model using Selected Features

We rebuild a linear regression model using only the features selected in the previous step to predict CO2 emissions. First, we filter the training and testing data to include only the selected features. Then, a LinearRegression model is trained on the filtered training data, and is used to predict CO2 emissions on the filtered testing data

Task 2: Linear Regression

2.3 Evaluation Metrics:

The values of MSE and R^2 for the First Model are:

```
Mean Squared Error=  344.34694966115006
R^2=  0.8955082394447692
```

The values of MSE and R^2 for the Model using Selected Features are:

```
R2 Score: 0.8852430219675873
MSE: 378.1754190743682
```

The First model performs slightly better because it has a lower MSE and a higher R^2 score.

Task 3: Classification

3.1 Data Preprocessing and Exploration

3.1.1 Column Name Cleaning:

Remove leading/trailing spaces and quotes from column names

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value
0	54	1	' Not Graduate'	' Yes'	8400000	23000000	12	657	15400000	14800000
1	2588	1	' Graduate'	' Yes'	1200000	4700000	2	667	1100000	300000
2	4074	3	' Graduate'	' No'	900000	2800000	10	560	2700000	300000
3	2990	5	' Not Graduate'	' Yes'	6500000	13800000	10	636	13000000	5500000
4	3827	1	' Graduate'	' Yes'	3400000	12300000	8	523	6100000	4700000
...	...	...	...	...	...	...	...	...	...	...
4264	94	1	' Graduate'	' No'	1700000	6400000	10	452	900000	1100000
4265	2448	2	' Graduate'	' Yes'	4800000	17500000	20	381	400000	6900000
4266	1925	0	' Graduate'	' Yes'	5000000	19900000	6	748	8300000	8500000
4267	4187	3	' Not Graduate'	' Yes'	4000000	8600000	4	667	11500000	6300000
4268	933	0	' Not Graduate'	' Yes'	7900000	24000000	20	728	18600000	15600000

3.1.2 Dataset Information:

Provide an overview of the dataset, including column names, non-null counts, and memory usage.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   loan_id                   4269 non-null   int64 
 1   no_of_dependents          4269 non-null   int64 
 2   education                 4269 non-null   object
 3   self_employed             4269 non-null   object
 4   income_annum              4269 non-null   int64 
 5   loan_amount               4269 non-null   int64 
 6   loan_term                 4269 non-null   int64 
 7   cibil_score               4269 non-null   int64 
 8   residential_assets_value  4269 non-null   int64 
 9   commercial_assets_value   4269 non-null   int64 
 10  luxury_assets_value       4269 non-null   int64 
 11  bank_asset_value          4269 non-null   int64 
 12  loan_status               4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
None
```

Task 3: Classification

3.1.3 Missing Values Analysis:

Identify missing values in the dataset for further handling and imputation.

```
      loan_id  no_of_dependents  education  self_employed  income_annum  \
0       False             False      False          False         False
1       False             False      False          False         False
2       False             False      False          False         False
3       False             False      False          False         False
4       False             False      False          False         False
...       ...               ...        ...            ...           ...
4264    False             False      False          False         False
4265    False             False      False          False         False
4266    False             False      False          False         False
4267    False             False      False          False         False
4268    False             False      False          False         False

      loan_amount  loan_term  cibil_score  residential_assets_value  \
0           False      False        False                     False
1           False      False        False                     False
2           False      False        False                     False
3           False      False        False                     False
4           False      False        False                     False
...           ...        ...          ...                       ...
4264        False      False        False                     False
4265        False      False        False                     False
4266        False      False        False                     False
4267        False      False        False                     False
4268        False      False        False                     False
```

3.1.4 Statistical Summary:

Generate descriptive statistics for numerical columns, including mean, standard deviation, minimum, and maximum values.

```
            loan_id  no_of_dependents  income_annum   loan_amount    loan_term  \
count  4269.000000       4269.000000  4.269000e+03  4.269000e+03  4269.000000
mean   2084.600843          2.486531  5.132888e+06  1.542111e+07    10.801124
std    1220.135062          1.706462  2.833325e+06  9.199375e+06     5.708889
min       2.000000          0.000000  2.000000e+05  3.000000e+05     2.000000
25%    1022.000000          1.000000  2.700000e+06  7.700000e+06     6.000000
50%    2098.000000          2.000000  5.200000e+06  1.500000e+07    10.000000
75%    3126.000000          4.000000  7.700000e+06  2.200000e+07    16.000000
max    4269.000000          5.000000  9.900000e+06  3.870000e+07    20.000000

       cibil_score  residential_assets_value  commercial_assets_value  \
count  4269.000000              4.269000e+03             4.269000e+03
mean    599.351370              7.567159e+06             5.069501e+06
std     172.820747              6.662199e+06             4.441275e+06
min     300.000000             -1.000000e+05             0.000000e+00
25%     451.000000              2.100000e+06             1.300000e+06
50%     593.000000              5.500000e+06             3.800000e+06
75%     749.000000              1.170000e+07             7.900000e+06
max     900.000000              2.870000e+07             1.940000e+07

       luxury_assets_value  bank_asset_value
count         4.269000e+03      4.269000e+03
mean          1.542614e+07      5.052659e+06
std           9.271711e+06      3.323036e+06
min           3.000000e+05      0.000000e+00
25%           7.500000e+06      2.300000e+06
50%           1.510000e+07      4.600000e+06
```

✓ 0s completed at 00:46

Task 3: Classification

3.1.5 Describe Categorical Data:
Display unique, top and aggregate the highest value.

	education	self_employed	loan_status
count	4269	4269	4269
unique	2	2	2
top	' Graduate'	' No'	' Approved'
freq	2215	2207	2649

```
print(data['education'].unique())
print(data['self_employed'].unique())
print(data['loan_status'].unique())
```

```
[" Not Graduate" " Graduate"]
[" Yes" " No"]
[" Approved" " Rejected"]
```

3.1.6 Encode Categorical Variables:
Converting categorical features values(education, self_employed and loan_status) which are binary features into zero and one values.

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value
0	54	1	1	1	8400000	23000000	12	657	15400000	14800000
1	2588	1	0	1	1200000	4700000	2	667	1100000	300000
2	4074	3	0	0	900000	2800000	10	560	2700000	300000
3	2990	5	1	1	6500000	13800000	10	636	13000000	5500000
4	3827	1	0	1	3400000	12300000	8	523	6100000	4700000
...	...	...	...	...	...	...	...	...	...	...
4264	94	1	0	0	1700000	6400000	10	452	900000	1100000
4265	2448	2	0	1	4800000	17500000	20	381	400000	6900000
4266	1925	0	0	1	5000000	19900000	6	748	8300000	8500000
4267	4187	3	1	1	4000000	8600000	4	667	11500000	6300000
4268	933	0	1	1	7900000	24000000	20	728	18600000	15600000

4269 rows × 13 columns

Task 3: Classification

3.1.7 Separate features and class label:

Separate the data into X and , then split it so the dataset was divided into training (70%) and testing (30%) subsets using train_test_split function with a random_state value of 42 to ensure consistency. Features were then standardized using StandardScaler to achieve a mean of 0 and a standard deviation of 1, ensuring consistent feature scaling for better model performance

x

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value
0	54	1	1	1	8400000	23000000	12	657	15400000	14800000
1	2588	1	0	1	1200000	4700000	2	667	1100000	300000
2	4074	3	0	0	900000	2800000	10	560	2700000	300000
3	2990	5	1	1	6500000	13800000	10	636	13000000	5500000
4	3827	1	0	1	3400000	12300000	8	523	6100000	4700000
...	...	...	...	...	...	...	...	...	...	...
4264	94	1	0	0	1700000	6400000	10	452	900000	1100000
4265	2448	2	0	1	4800000	17500000	20	381	400000	6900000
4266	1925	0	0	1	5000000	19900000	6	748	8300000	8500000
4267	4187	3	1	1	4000000	8600000	4	667	11500000	6300000
4268	933	0	1	1	7900000	24000000	20	728	18600000	15600000

4269 rows × 13 columns

✓ 0s completed at 00:46

[9] y

	loan_status
0	0
1	0
2	0
3	0
4	1
...	...
4264	1
4265	1
4266	0
4267	0
4268	0

4269 rows × 1 columns

Task 3: Classification

3.2 Classification Algorithms

3.2.1 Decision Tree Algorithm:

Implementing a Decision Tree algorithm using sklearn.tree to train a model for classification (or regression). We used the trained model to make predictions and evaluate its performance on the dataset.

	precision	recall	f1-score	support
0	0.99	0.99	0.99	833
1	0.98	0.98	0.98	448
accuracy			0.99	1281
macro avg	0.98	0.99	0.98	1281
weighted avg	0.99	0.99	0.99	1281

```
print(confusion_matrix(y_test, y_pred_DecisionTree))
print("accuracy_DecisionTree=", accuracy_score(y_test, y_pred_DecisionTree))
auc = roc_auc_score(y_test, y_pred_DecisionTree)
print(f"AUC_DecisionTree=: {auc}")
```

```
[[823  10]
 [  8 440]]
accuracy_DecisionTree= 0.9859484777517564
AUC_DecisionTree=: 0.9850690276110444
```

```
#Holdout_accuracy of DecisionTree
from sklearn.model_selection import train_test_split,cross_val_score
holdout_accuracy = accuracy_score(y_test, y_pred_DecisionTree)
print(f"Hold-out validation accuracy DT: {holdout_accuracy}")
```

```
Hold-out validation accuracy DT: 0.9859484777517564
```

Task 3: Classification

3.2.2 Logistic Regression Algorithm:

Implementing a Logistic Regression using sklearn.linear_model. The algorithm was used to model the relationship between the input features and the target variable by estimating the probabilities using the sigmoid function. We trained the model, made predictions, and evaluated its performance on the dataset.

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94       833
           1       0.88      0.93      0.90       448

    accuracy                           0.93      1281
   macro avg       0.92      0.93      0.92      1281
weighted avg       0.93      0.93      0.93      1281
```

```
print(confusion_matrix(y_test, y_pred_logisticRegression))
auc = roc_auc_score(y_test, y_pred_logisticRegression)
print(f"AUC_LogisticRegression: {auc}")
print("accuracy_LogisticRegression=", accuracy_score(y_test, y_pred_logisticRegression))
```

```
[[775  58]
 [ 33 415]]
AUC_LogisticRegression: 0.9283557172869147
accuracy_LogisticRegression= 0.9289617486338798
```

```
from sklearn.model_selection import train_test_split,cross_val_score
holdout_accuracy = accuracy_score(y_test, y_pred_logisticRegression)
print(f"Hold-out validation accuracy LR: {holdout_accuracy}")
```

```
Hold-out validation accuracy LR: 0.9289617486338798
```

```
10-fold cross-validation accuracyLR: 0.7723169618805731
```

Task 3: Classification

3.2.3 Support Vector Machine (SVM) Algorithm:

Implementing a Support Vector Machine (SVM) using sklearn.svm. The algorithm was used to classify data by finding the optimal hyperplane that maximizes the margin between classes. We trained the model, made predictions, and evaluated its performance on the dataset.

```
                precision    recall  f1-score   support

           0       0.97      0.93      0.95       833
           1       0.88      0.94      0.91       448

    accuracy                           0.94      1281
   macro avg       0.92      0.94      0.93      1281
weighted avg       0.94      0.94      0.94      1281
```

```
print(confusion_matrix(y_test, y_pred_SVM))
auc = roc_auc_score(y_test, y_pred_SVM)
print(f"AUC_SVM: {auc}")
print("accuracy_SVM", accuracy_score(y_test,y_pred_SVM))
```

```
[[775  58]
 [ 25 423]]
AUC_SVM: 0.9372842887154862
accuracy_SVM 0.9352068696330992
```

```
from sklearn.model_selection import train_test_split,cross_val_score
holdout_accuracy = accuracy_score(y_test, y_pred_SVM)
print(f"Hold-out validation accuracy SVM: {holdout_accuracy}")
```

```
Hold-out validation accuracy SVM: 0.9352068696330992
```

Task 3: Classification

3.2.4 Naïve Bayes Algorithm:

Implementing a Naïve Bayes algorithm using sklearn.naive_bayes. This probabilistic classifier was based on Bayes' theorem, assuming independence between features. We trained the model, made predictions, and evaluated its performance on the dataset.

	precision	recall	f1-score	support
0	0.97	0.93	0.95	833
1	0.88	0.95	0.91	448
accuracy			0.94	1281
macro avg	0.93	0.94	0.93	1281
weighted avg	0.94	0.94	0.94	1281

```
print(confusion_matrix(y_test, y_pred_NB))
print("Accuracy_NB=", accuracy_score(y_test, y_pred_NB))
auc = roc_auc_score(y_test, y_pred_NB)
print(f"AUC_NB: {auc}")
```

```
[[774  59]
 [ 21 427]]
Accuracy_NB= 0.9375487900078064
AUC_NB: 0.9411483343337336
```

```
from sklearn.model_selection import train_test_split,cross_val_score
holdout_accuracy = accuracy_score(y_test, y_pred_NB)
print(f"Hold-out validation accuracy NB: {holdout_accuracy}")
```

```
Hold-out validation accuracy NB: 0.9375487900078064
```

Task 3: Classification

3.2.5 Ensemble Algorithm:

Implementing an ensemble methods using sklearn.ensemble, combining multiple base models to improve overall performance. Techniques such as Bagging (e.g., Random Forest) was used to train and aggregate predictions, enhancing accuracy and robustness. The model was trained, predictions were made, and performance was evaluated on the dataset.

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       833
           1       0.99      0.99      0.99       448

    accuracy                           0.99      1281
   macro avg       0.99      0.99      0.99      1281
weighted avg       0.99      0.99      0.99      1281

[[830   3]
 [  4 444]]
Accuracy_RandomForset= 0.994535519125683
AUC_RandomForest: 0.9937349939975991
```

```
from sklearn.model_selection import train_test_split,cross_val_score
holdout_accuracy = accuracy_score(y_test, y_pred_RandomForest)
print(f"Hold-out validation accuracy RF: {holdout_accuracy}")
```

```
Hold-out validation accuracy RF: 0.994535519125683
```

```
cross_val_accuracies = cross_val_score(RandomForest_c, x, y, cv=10)
print(f"10-fold cross-validation accuracy RF: {cross_val_accuracies.mean()}")
```

```
10-fold cross-validation accuracy RF: 0.9953156095040188
```

Task 3: Classification

Tasks 3.3 and 3.4 were done in task 3.2

3.5 Identifying The Best Model:

We implemented various machine learning algorithms, including Decision Tree, Logistic Regression, Support Vector Machine, and Naïve Bayes, alongside ensemble methods like Random Forest using sklearn.ensemble. After training and evaluating all models, we found that the **ENSEMBLE METHOD** outperformed individual models in terms of accuracy and robustness, making them the best-performing approach for our dataset.

Accuracy_RandomForset= 0.994535519125683
AUC_RandomForest: 0.9937349939975991
10-fold cross-validation accuracy RF: 0.9953156095040188