**CO2 Emission Predictor**

**A Project Report**

Submitted in partial fulfillment of the

Requirements for the award of the Degree of

**BACHELOR OF SCIENCE**

**(COMPUTER SCIENCE)**

By

**Satyam Sharma**

**Seat Number: 444**

**Under the esteemed guidance of**

**Prof. Madhav Mishra**

**Assistant Professor**



**DEPARTMENT OF COMPUTER SCIENCE**

**GURU NANAK KHALSA COLLEGE**

**OF**

**ARTS, SCIENCE & COMMERCE**

**(*Autonomous)***

**MATUNGA, MUMBAI - 400 019**

**AY 2023-2024**

**GURU NANAK KHALSA COLLEGE**

**OF**

**ARTS, SCIENCE & COMMERCE**

**(*Autonomous)***

**MATUNGA, MUMBAI, MAHARASHTRA – 400 019**

**DEPARTMENT OF COMPUTER SCIENCE**



**CERTIFICATE**

This is to certify that the entitled, **“CO2 Emission Predictor,”** is bonafied work of **Satyam Sharma** bearing Seat No. **444** submitted in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE in COMPUTER SCIENCE** from University of Mumbai.

**Internal Guide Coordinator**

**External Examiner**

**Date: College Seal**

APPROVAL OF PROJECT PROPOSAL

###### **PRN No: 2021016401287432 Roll No: 444**

1. **Name of the Student**

**Satyam Sharma**

1. **Title of the Project**

**CO2 Emission Predictor**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Name of the Project Guide**

**Prof. Madhav Mishra**

**Signature of the Student Signature of the Guide**

**Date: Date:**

**Signature of the Head of Dept. Signature of the Examiner**

**Date: Date:**

**ACKNOWLEDGEMENT**

I would like to express my thanks to the people who have helped me most throughout my project.

I am grateful to my **Prof. Madhav Mishra** for nonstop support for the project. I cannot say thank you enough for him tremendous support and help.

I owe my deep gratitude to our HOD of Computer Science Department **Mrs. Jasmeet Kaur Ghai** who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

At last, but not the least I want to thank all my friends who helped/treasured me out in completing the project, where they all exchanged their own interesting ideas, thoughts and made this possible to complete my project with all accurate information. I wish to thank my parents for their personal support or attention who inspired/encouraged me to go my own way.

|  |
| --- |
|  |
|  |  |

**DECLARATION**

I hereby declare that the project entitled, “**CO2 Emission Predictor**” done at **Guru Nanak Khalsa College**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE (Computer Science)** to be submitted as final semester project as part of our curriculum.

Satyam Sharma

|  |
| --- |
|  |
|  |  |

# **Table of Contents**

* **Title**

* **Problem Statement**

* **Abstract**

* **Objectives**

* **Data Description**
* **Proposed Library**

* **UML Diagrams**

* **Model Description**

* **Code**

* **Evaluation**

* **Future Development**

# Title

# **CO2 Emission Prediction**

# Problem Statement

The increase in carbon dioxide (CO2) emissions is a critical contributor to global climate change. Predicting future CO2 emissions is essential for governments, organizations, and individuals to develop effective strategies for reducing greenhouse gas emissions and mitigating the adverse impacts of climate change. This project aims to create a robust and accurate predictive model for CO2 emission. The primary objective of this project is to develop a data-driven model for accurately predicting future carbon dioxide (CO2) emissions.

The goal of this project is to develop a machine learning model that can predict carbon dioxide (CO2) emissions for a given set of input parameters. As the world continues to grapple with climate change and the environmental impact of CO2 emissions, accurate predictions of CO2 emissions are crucial for both individuals and organizations to make informed decisions and take steps towards reducing their carbon footprint.

The project aims to create a predictive model that considers various factors such as energy consumption, transportation methods, industrial activities, and other relevant variables to estimate CO2 emissions. The model should be accurate, robust, and adaptable to different geographical regions and sectors.

# Abstract

This project focuses on developing a machine learning model to predict C02 emission with the goal of aiding environmental sustainability efforts. The increasing concern over climate change and its impacts on the planet has underscored the importance of accurately estimating and monitoring carbon dioxide (CO2) emissions. This machine learning model aims to provide accurate predictions of CO2 emission depending on various features. By offering timely and precise CO2 emissions forecasts. Project also aims to encourage the adoption of eco-friendly practices. The model's user-friendliness makes it a valuable resource for promoting environmental awareness for more sustainable future.

# 

# 

# Objective

## Data Collection and Integration:

Gather and integrate historical data on CO2 emissions, including data from various sources such as government agencies, research institutions, and industry records.

## Data Preprocessing:

Clean and preprocess the collected data to address issues like missing values, outliers, and data inconsistencies. Normalize and standardize the data to make it suitable for modeling.

## Feature Engineering:

Identify and create relevant features that may impact CO2 emissions, considering factors like Engine Size, Cylinders, Fuel Type and Fuel Consumption Combined.

## Model Development:

Utilize appropriate machine learning and statistical modeling techniques to build a predictive model for CO2 emissions.

Evaluate and compare various modeling approaches, including regression models, and advanced methods like ensemble models.

## Model Training and Validation:

Train the predictive model on historical data and validate its performance using appropriate metrics such as Mean Absolute Error (MAE) or Root Mean Square Error (RMSE). Implement cross-validation and time-series-specific validation techniques to ensure the model's robustness.

## Visualization and Interpretation:

Develop data visualization tools to communicate the predicted CO2 emission effectively.

Provide an interpretable analysis of the model’s findings, highlighting the most influential factors affecting the CO2 emissions findings.

## Application and policy support:

Make the predictive model accessible for policymakers, organizations and the public to inform decision-making.

Assist in developing emission reduction strategies and climate policies based on the model’s forecasts.

## Educational Outreach:

Contribute to public awareness and understanding of the factors influencing CO2 emissions and the importance of reducing them for climate change mitigation.

## Challenge and Future Directions:

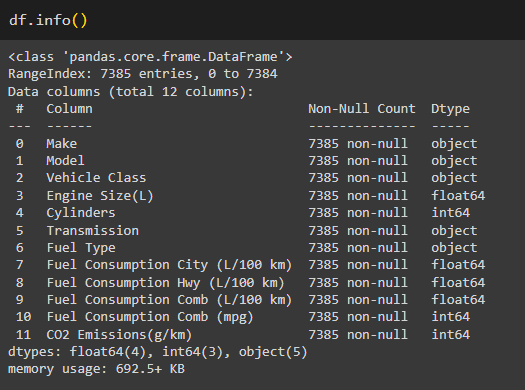
Identify and address challenges in CO2 emission predictions, such as data quality, model complexity, and the dynamic nature of emissions

Explore opportunities for future research, including real-time data integration and improved remote sensing technologies.

By addressing these challenges and objectives, this project aims to provide a valuable tool for predicting CO2 emissions, supporting climate change mitigation efforts, and contributing to a sustainable and environmentally responsible future.

# Data Description

### df.info()



### df.head()

## Column Description

Make: This feature tells us the Company Name of the vehicle.

Model: This column describes which model the car is for the specific company as shown in the ‘Make’ Column.

* AWD: All-Wheel Drive
* FFV: Flexible-fuel vehicle
* SWB: short wheelbase
* LWB: Long Wheelbase
* EWB: Extended wheelbase

Vehicle Class: Class of the vehicle depending on their utility, capacity, and weight.

Engine Size(L): This feature tells us the size of an engine used in Liter.

Cylinders: This feature shows the numbers of cylinders the engine consists of in a vehicle.

Transmission: These features tell us the transmission types with numbers of gears.

* A: Automatic
* AM: Automated manual
* AS: Automated with select shift
* AV: Continuously variable
* M: Manual
* 3-10: Number of gears

Fuel Type: This Feature shows us the type of Fuel used in the engine.

* X: Regular gasoline
* Z: Premium gasoline
* D: Diesel
* E: Ethanol(E85)
* N: Natural gas

Fuel Consumption City (L/100km): Fuel Consumption on City roads(L/100km)

Fuel Consumption Hwy (L/100km): Fuel consumption on Highway roads (L/100km).

Fuel Consumption Comb (L/100km): The combined fuel consumption (55% city, 45% highway) is shown in L/100km.

Fuel Consumption Comb (mpg): The combined fuel consumption in both city and highway is shown in mile per gallon(mpg).

CO2 Emission(g/km): The tailpipe emissions of carbon dioxide (in grams per kilometer) for combined city and highway driving. This is the column we are looking to predict on a test dataset.

The columns of the dataset have only float, object and integer values.

This dataset captures the details of how CO2 emissions by a vehicle can vary with the different features. The dataset has been taken from Canada Government official open data website. This is a compiled version. This contains data over a period of 7 years.

There are a total of 7358 rows and 12 columns. There are few abbreviations that have been used to describe the features. The same can be found in the Data Description sheet.

The data has been taken and compiled from the below Canada Government official link

[Fuel consumption ratings - Open Government Portal](https://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64#wb-auto-6)

# Proposed Libraries

## Pandas:

It offers data structure and function for easy data manipulation, analysis and cleaning. It simplifies tasks like reading/writing data from various sources, handling missing data, and performing statistical computation.

## NumPy:

Numerical Python is a fundamental library in python for numerical and mathematical operations. It is used for numerical computing, providing support for large, multi-dimensional arrays and matrices. It offers a wide range of mathematical functions to perform operations.

## Seaborn:

Seaborn is a python data visualization library built on top of Matplotlib. A visualization library based on matplotlib that provides a high-level interface for creating attractive statistical graphics. It simplifies the process of creating informative and visually appealing plots.

## sklearn.neighbors:

A module within the scikit-learn (sklearn) library, which is a popular machine learning library in Python. This module provides a range of tools and algorithms for performing tasks related to nearest neighbors, including both supervised and unsupervised machine learning techniques. The primary focus of sklearn.neighbors is on nearest neighbor classification and regression. For regression tasks we can use the ‘KNeighborsRegressor’, and for classification we can use the ‘KNeighborsClassifier’.

## sklearn.ensemble:

A submodule within the scikit-learn (sklearn) library, a popular machine learning library in Python. The ensemble module in scikit-learn focuses on ensemble methods, which are techniques for combining multiple machine learning models to improve predictive performance and reduce overfitting. ‘RandomForestClassifier’ and ‘RandomForestRegressor’ classes create an ensemble of decision tree. ‘AdaBoostClassifier’ and ‘AdaBoostRegressor’ classes implement adaptive boosting.

## sklearn.tree:

A module within the scikit-learn (sklearn) library, a popular machine learning library in Python. The tree module in scikit-learn focuses on decision tree-based algorithms and methods for classification and regression tasks. ‘DecisionTreeClassifier’ a class for classification tasks and ‘DecisionTreeRegressor’ a class for regression tasks in sklearn.tree.

## sklearn.svm:

A module within the scikit-learn (sklearn) library, a popular machine learning library in Python. The svm module in scikit-learn focuses on Support Vector Machines (SVMs), which are supervised machine learning algorithms used for classification and regression tasks. Support Vector Classification(SVC) for classification tasks and Support Vector Regression(SVR) for Regression tasks.

## sklearn.linear\_model:

A module within the scikit-learn (sklearn) library, a widely-used machine learning library in Python. The linear\_model module in scikit-learn focuses on linear models for various machine learning tasks, including regression, classification, and dimensionality reduction. ‘LinearRegression’ is a class for building linear regression models to predict a continuous target variable from one or more input features. ‘Ridge’ is a class for building Ridge regression models (L1 regularization) to Linear Regression. ‘Lasso’ is a class for building lasso regression models((L2 regularization) to Linear Regression.

## sklearn.model\_selection:

A module within the scikit-learn (sklearn) library, a widely-used machine learning library in Python. The model\_selection module in scikit-learn provides a range of tools and utilities for splitting datasets, cross-validation, hyperparameter tuning, and performance evaluation.

## sklearn.metrics:

a module within the scikit-learn (sklearn) library, a popular machine learning library in Python. The metrics module in scikit-learn provides a wide range of functions and classes for evaluating the performance of machine learning models. These metrics help assess how well a model is doing on various tasks such as classification, regression, clustering, and more.

## sklearn.preprocessing:

a module within the scikit-learn (sklearn) library, a popular machine learning library in Python. The preprocessing module in scikit-learn provides various tools for data preprocessing, transformation, and feature engineering. ‘StandardScaler’ for standardization and ‘MinMaxScaler’ for Normalization.

## warnings:

The import warnings statement in Python is used to control how warning messages are displayed or handled in your code. By using the warnings module, you can control whether to ignore, display, or handle these warning messages.

## plotly:

Plotly is a Python library for creating interactive and visually appealing data visualizations. It is known for its flexibility and wide range of supported chart types, including line charts, bar charts, scatter plots, heatmaps, and more. Plotly allows you to create interactive web-based visualizations that can be embedded in web applications or shared online.

## matplotlib.pyplot:

A sub-library of Matplotlib, one of the most widely used Python libraries for creating static, animated, and interactive visualizations in Python. matplotlib.pyplot is commonly used for creating 2D plots and graphs, including line charts, bar charts, scatter plots, histograms, and more.

# UML Diagrams

Unified Modeling Language (UML) diagrams are a standardized way to visualize, document, and communicate the different aspects of a software system or a complex process. UML diagrams are widely used in software engineering and system design to represent various elements and relationships within a system. There are several types of UML diagrams, each serving a specific purpose. Here are some UML diagrams:

* Use Case Diagram:

Represents the functional requirements of a system.

Shows how users (actors) interact with the system through various use cases (functionalities).

* Class Diagram:

Represents the static structure of a system.

Shows classes, their attributes, methods, and the relationships between classes.

* Activity Diagram:

Represents the workflow or business process of a system.

Shows the flow of activities and decisions in a process or use case.

UML diagrams are a powerful tool for system design, analysis, and documentation.

The choice of which type of UML diagram to use depends on the specific aspect of the system or process you want to model or communicate.

Depending on the complexity of your project, you may use one or more of these diagram types to represent different aspects of your system.

## **Class Diagram**

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

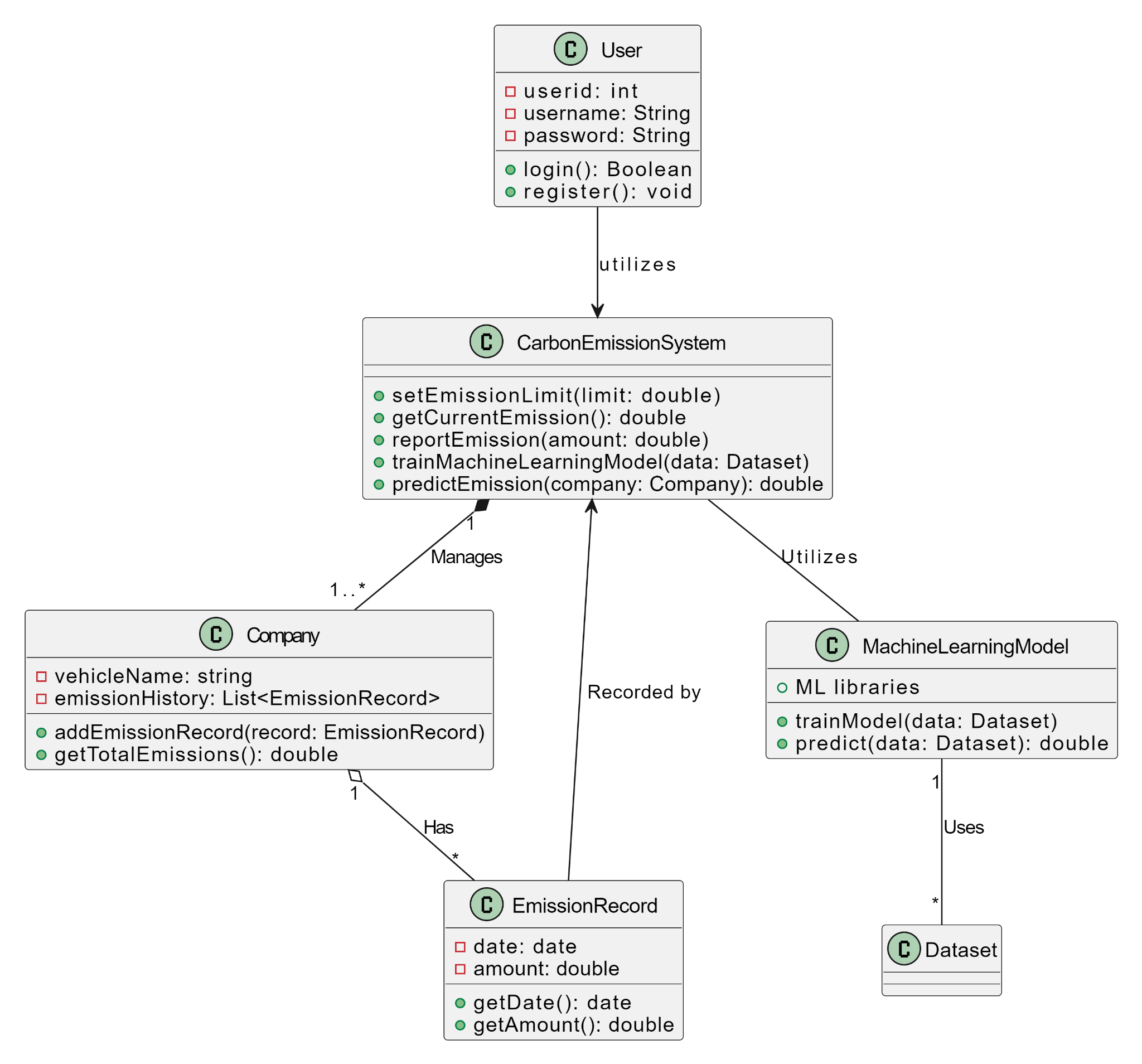
Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

In this class diagram, we have a ‘Company’ class representing information about the vehicles. Each vehicle is identified by a ‘vehicleName’ and has attributes such as ‘emissionHistory’ and additional unspecified attributes.

The ‘EmissionRecord’ class defines the vehicle emission records such as amount of CO2 emissions on a particular date, Emission Record can perform actions like ‘getDate()’ and ‘getAmount()’

The ‘User’ class defines the user’s login credentials with a ‘userid’, ‘username’ and ‘password’. Users can perform actions like ‘login()’ and ‘register()’.

Within the ‘CarbonEmissionSystem’ a ‘MachineLearningModel’ is responsible for predictive tasks. It utilizes machine learnings models such as ‘LinearRegression’, ‘SupportVectorRegression’, ‘RandomForestRegression’, ‘DecisionTreeRegression’, ‘AdaBoostRegression’ and ‘KNNRegression’. These models likely use machine learning libraries to perform their tasks, though the specific details are not provided.



## **Activity Diagram**

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc

We use Activity Diagrams to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram.

An activity diagram focuses on the condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.

An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modeling where their primary use is to depict the dynamic aspects of a system.

A decision point arises, whether the user is registered and has successfully logged in. If the answer is ‘yes’ , users can enter the vehicle data and submit the data for further prediction to the machine learning model once the data is verified by the system. The user can proceed to generate and ‘Display Prediction’.

If the answer is ‘no’, it means an unregistered or unsuccessful login attempt. The system will show ‘Authentication error’ the user is directed back to the login/register process, allowing them the opportunity to retry the registration or login process.

Activity Diagram


## Use-Case Diagram

A use case diagram is a type of UML (Unified Modeling Language) diagram that represents the functional requirements of a system from the perspective of external actors (users or systems). Use case diagrams illustrate how users interact with a system and define the various use cases (functionalities) that the system provides.

In this use case diagram:

* The system is represented at the top as a rectangle labeled "Carbon Dioxide Prediction by a Vehicle."
* The various use cases (functionalities) that the system provides are listed within the system boundary. These include "Collect Vehicle Data," "Predict CO2 Emissions," "Display CO2 Prediction," "Configure Prediction Settings," "Update Data," and "View Historical Data."

Actors:

* Two external actors are depicted in the diagram: "Driver" and "System User." These represent the individuals or entities interacting with the system.

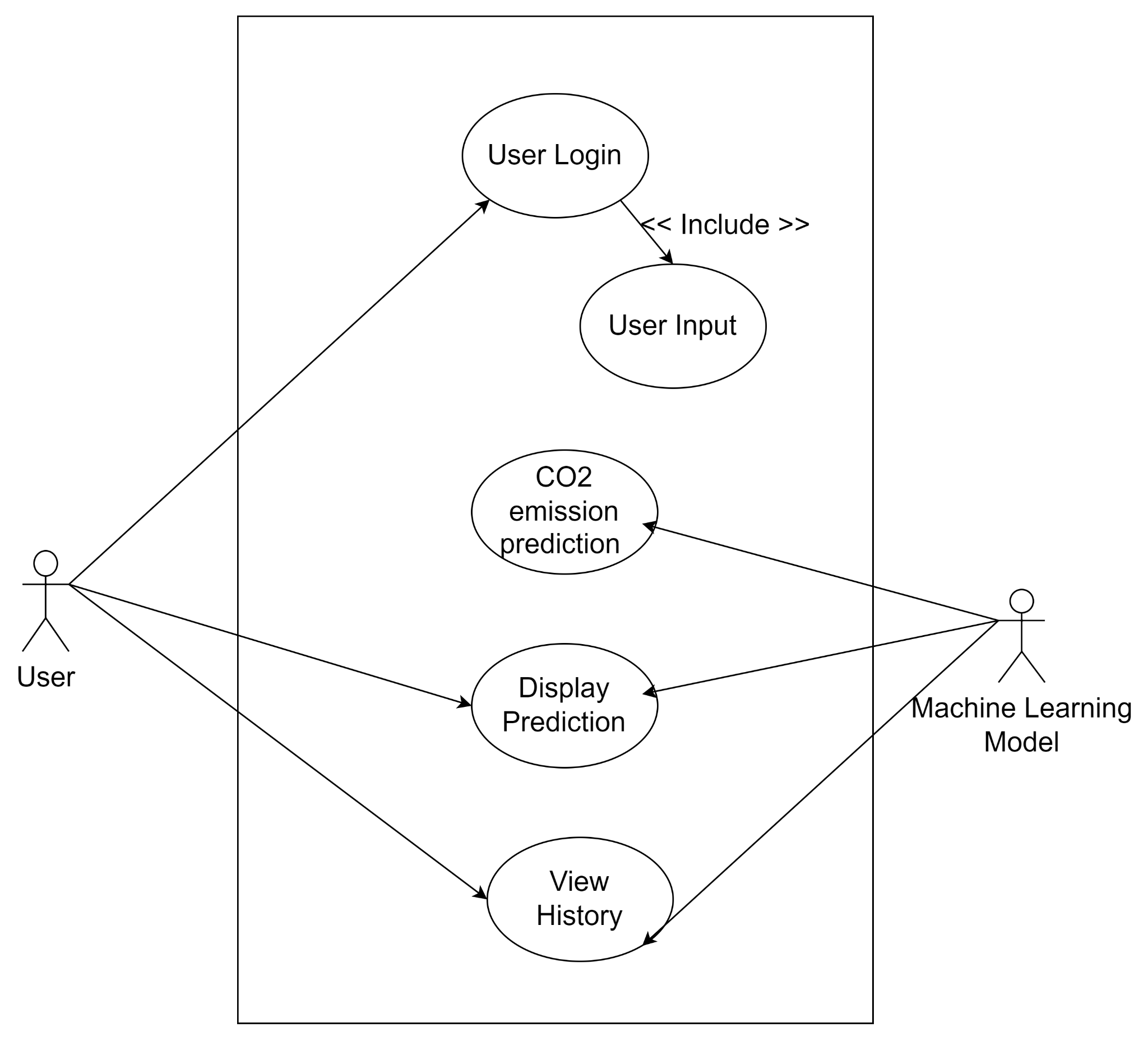
Associations:

* The associations between the actors and the use cases illustrate how the external actors interact with the system's functionalities. For example, the "Driver" interacts with "Collect Vehicle Data," "Predict CO2 Emissions," and "Display CO2 Prediction."

Use Cases:

* Each use case is a rectangular shape with the name of the functionality (e.g., "Collect Vehicle Data").
* Use cases describe the actions that the system performs in response to interactions from external actors (e.g., the "Driver" collects vehicle data).

Use case diagrams help to provide an overview of the system's functionality and how users or external systems interact with it. It serves as a useful tool for initial requirements analysis and can be a foundation for more detailed system design and development.



# Model Description

## **Linear Regression**

Linear regression is a type of statistical analysis used to predict the relationship between two variables. It assumes a linear relationship between the independent variable and the dependent variable, and aims to find the best-fitting line that describes the relationship. The line is determined by minimizing the sum of the squared differences between the predicted values and the actual values.

Multiple linear regression is a technique to understand the relationship between a single dependent variable and multiple independent variables.

The formulation for multiple linear regression is also similar to simple linear regression with

the small change that instead of having one beta variable, you will now have betas for all the variables used. The formula is given as:

Y = B0 + B1X1 + B2X2 + … + BpXp + ε

## **Support Vector Regression**

Support Vector Regression (SVR) is a type of machine learning algorithm used for regression analysis. The goal of SVR is to find a function that approximates the relationship between the input variables and a continuous target variable, while minimizing the prediction error.

SVR seeks to find a hyperplane that best fits the data points in a continuous space. This is achieved by mapping the input variables to a high-dimensional feature space and finding the hyperplane that maximizes the margin (distance) between the hyperplane and the closest data points, while also minimizing the prediction error.

SVR can handle non-linear relationships between the input variables and the target variable by using a kernel function to map the data to a higher-dimensional space. This makes it a powerful tool for regression tasks where there may be complex relationships between the input variables and the target variable.

## **Random Forest Regression**

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated

**Decision Tree Regression**

A decision tree is one of the most frequently used Machine Learning algorithms for solving regression as well as classification problems. As the name suggests, the algorithm uses a tree-like model of decisions to either predict the target value (regression) or predict the target class (classification).

The process of splitting starts at the root node and is followed by a branched tree that finally leads to a leaf node (terminal node) that contains the prediction or the final outcome of the algorithm. Construction of decision trees usually works top-down, by choosing a variable at each step that best splits the set of items. Each subtree of the decision tree model can be represented as a binary tree where a decision node splits into two nodes based on the conditions.

Decision trees where the target variable or the terminal node can take continuous values (typically real numbers) are called regression trees. If the target variable can take a discrete set of values these trees are called classification trees.

**KNeighbors regression**

In statistics, the k-nearest neighbors algorithm (k-NN) is a non-parametric supervised learning method. It is used for classification and regression. In both cases, the input consists of the k closest training examples in a data set. The output depends on whether k-NN is used for classification or regression:

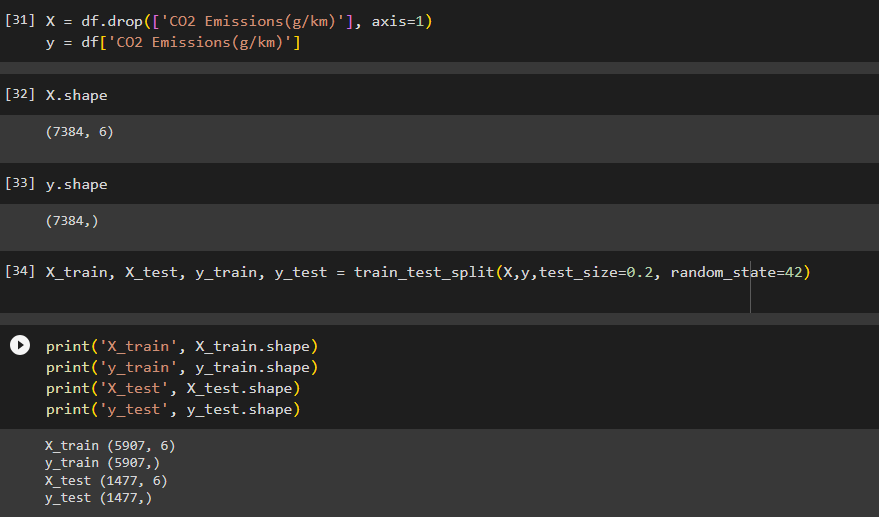
* In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
* In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors. If k = 1, then the output is simply assigned to the value of that single nearest neighbor.

k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

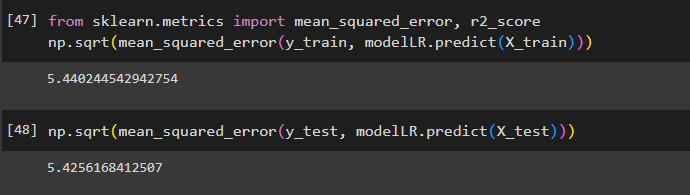
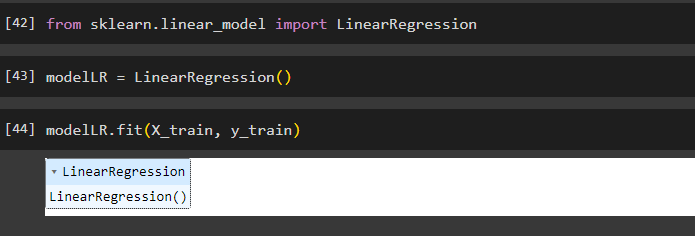
# Model Evaluation

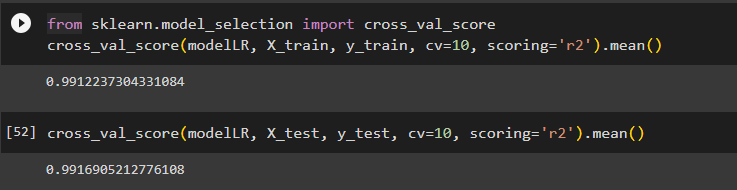
We will be evaluating the model on the basis of three different test\_size{0.2, 0.25, 0.3} and train\_size{0.8,0.75,0.70}.

### **test\_size=0.2 and train\_size=0.8**

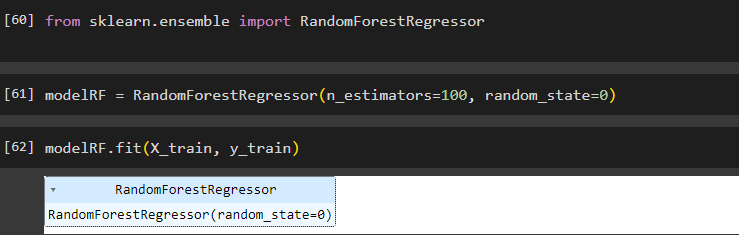


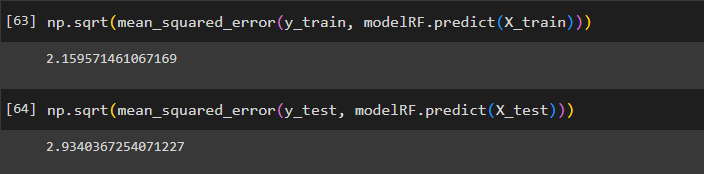
Linear Regression:

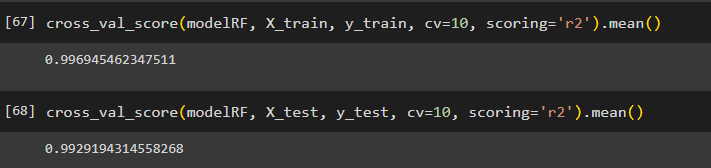




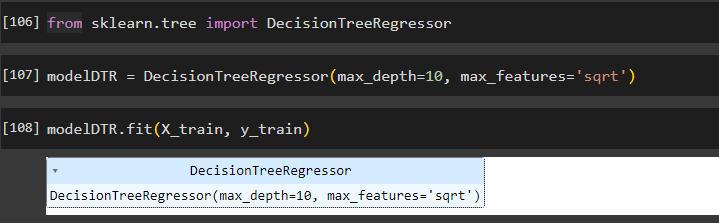
Random Forest Regression

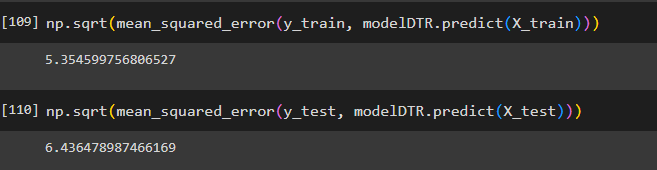


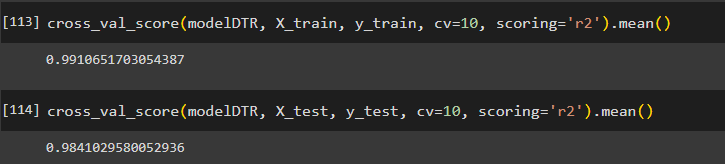




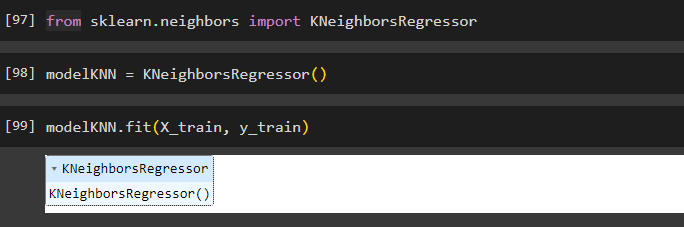
Decision Tree Regression

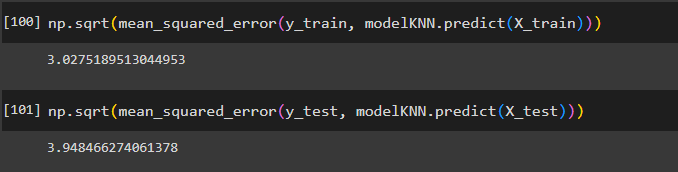


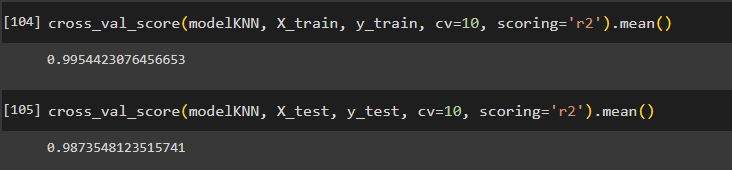




KNeighbors Regression

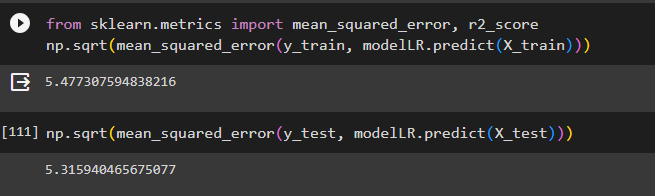


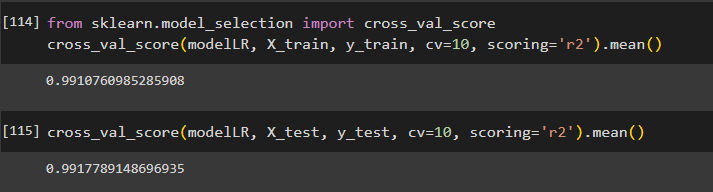




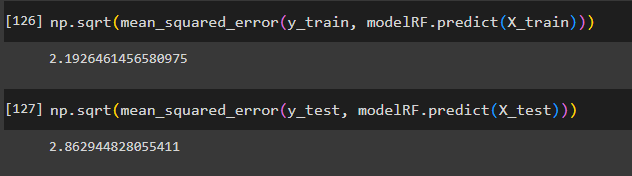
**test\_size=0.25 and train\_size=0.75**

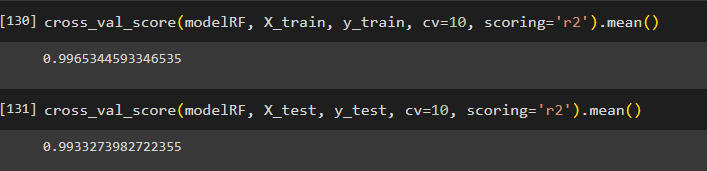
Linear Regression



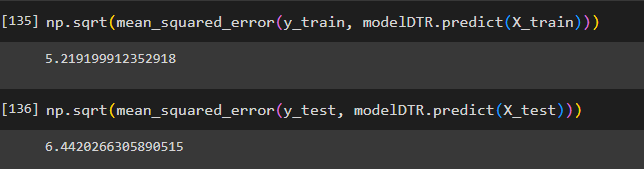
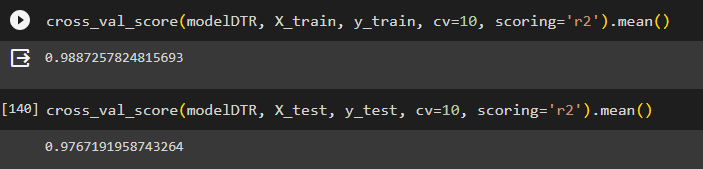


Random Forest Regression

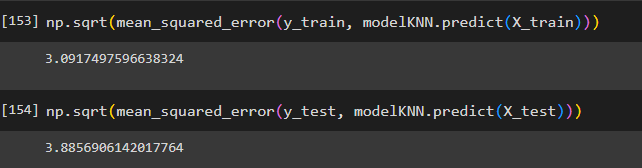
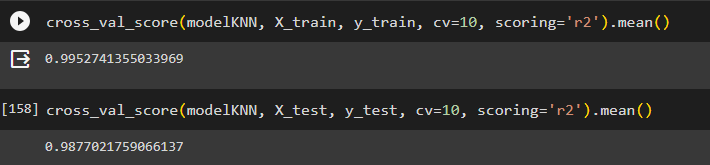




Decision Tree Regression

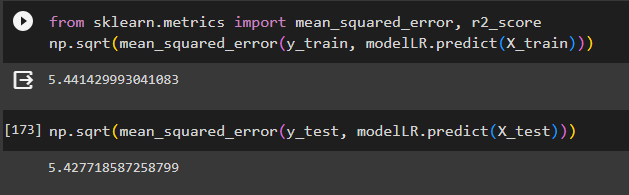
 

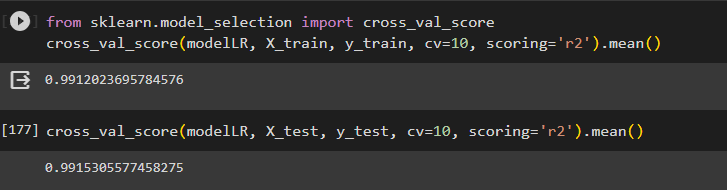
KNeighbors Regression

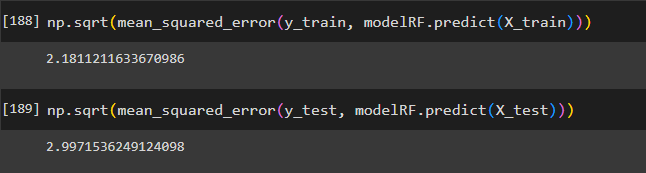
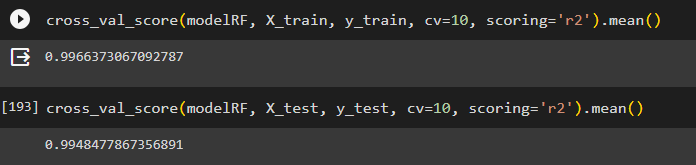
**test\_size=0.30 and train\_size=0.70**

Linear Regression

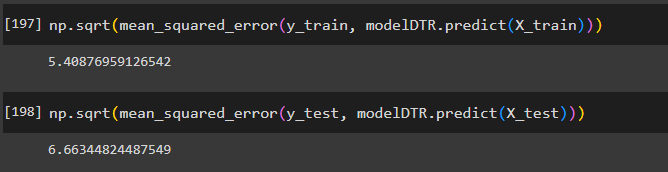
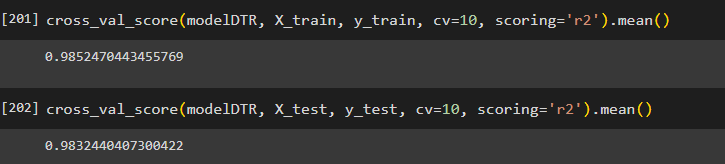




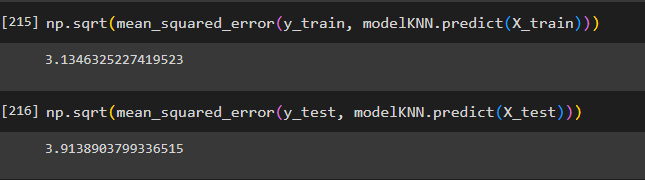
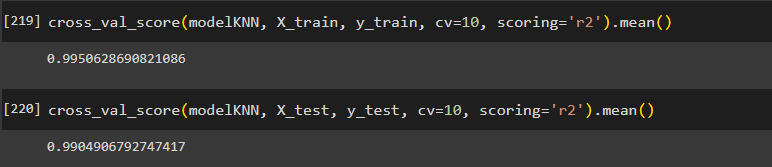
Random Forest Regression

Decision Tree Regression

KNeighbors Regression

From model evaluation of all the five regression models we can clarify that the ‘RandomForestRegressor’ and ‘KNeighborsRegressor’ performed well in predictive tasks.

Random Forest Regressor achieved the lowest Mean squared error on the test set [2.93] with the cross-validation score of [0.99] followed by the KNeighborsRegressor Mean squared error is [3.94] and cross validation score of [0.98]

The ensemble learning technique and hyper parameter tuning results in better performance of Random Forest Regressor. Random Forest Regressor is the most accurate prediction model for this given dataset.

# Code

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# plt.rcParmas["figure.figsize"]=(10,6)

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

import plotly.express as px

import plotly.graph\_objects as go

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler

from plotly.subplots import make\_subplots

# %matplotlib inline

import warnings

warnings.filterwarnings("ignore")

df = pd.read\_csv('/content/sample\_data/CO2 Emissions\_Canada.csv')

df.head()

df.tail(5)

df.sample(5)

df.info()

df.isnull().values.any()

df.isnull().sum()

def missing\_data(data):

total = data.isnull().sum().sort\_values(ascending = False)

percent = (data.isnull().sum()/data.isnull().count()\*100).sort\_values(ascending = False)

return pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

missing\_data(df)

df\_duplicated = df[df.duplicated() == True]

df\_duplicated

df.describe().T

#Shape

df.shape

#Columns name

df.columns

def get\_unique\_values(df):

output\_data = []

for col in df.columns:

# If the number of unique values in the column is less than or equal to 10

if df.loc[:, col].nunique() <= 10:

# Get the unique values in the column

unique\_values = df.loc[:, col].unique()

# Append the column name, number of unique values, unique values, and data type to the output data

output\_data.append([col, df.loc[:, col].nunique(), unique\_values, df.loc[:, col].dtype])

else:

# Otherwise, append only the column name, number of unique values, and data type to the output data

output\_data.append([col, df.loc[:, col].nunique(),"-", df.loc[:, col].dtype])

output\_df = pd.DataFrame(output\_data, columns=['Column Name', 'Number of Unique Values', ' Unique Values ', 'Data Type'])

return output\_df

get\_unique\_values(df)

df.corr()

print("Correlation Matrix")

plt.rcParams['figure.figsize']=(8,6)

sns.heatmap(df.corr(), cmap='coolwarm', linewidths=.5, fmt='.2f', annot=True)

index = 0

for feature in df.select\_dtypes('number').columns:

index+=1

plt.figure(figsize=(30,30))

plt.subplot((len(df.columns)), 2, index)

sns.boxplot(x=feature, data = df, whis = 3)

plt.tight\_layout()

plt.show()

for column in ['Make', 'Model', 'Vehicle Class', 'Transmission', 'Fuel Type']:

df\_column = df[column].value\_counts().reset\_index().rename(columns={'index':column, column:'Count'})[0:25]

df\_column = df\_column.sort\_values(by='Count', ascending=False)

fig = go.Figure(go.Bar(x=df\_column[column], y=df\_column['Count'],

marker = {'color':df\_column['Count'], 'colorscale':'Viridis'},

text = df\_column['Count'], textposition="outside"))

fig.update\_layout(title\_text = f'Top 25 {column}', xaxis\_title=column, yaxis\_title='Number of vehicles', title\_x=0.5, width=700, height=500)

fig.show()

def explore\_cat\_feature(feature):

group = df.groupby(feature).mean()

plt.figure(figsize=[15,5])

plots = group['CO2 Emissions(g/km)'].sort\_values().plot(kind='bar',fontsize=15)

plt.xlabel(feature, fontsize=15);

plt.ylabel("Mean CO2 Emission", fontsize=15);

plt.title("Mean CO2 Emission according to {} feature\n".format(feature), fontsize=20)

for feature in ['Make', 'Vehicle Class', 'Engine Size(L)', 'Cylinders', 'Transmission', 'Fuel Type']:

explore\_cat\_feature(feature)

sns.scatterplot(x='Engine Size(L)', y='CO2 Emissions(g/km)', data=df, hue='Fuel Type')

df.drop(['Make', 'Model', 'Vehicle Class', 'Fuel Consumption City (L/100 km)', 'Fuel Consumption Hwy (L/100 km)', 'Transmission', 'Fuel Consumption Comb (mpg)'], inplace=True, axis=1)

df.head()

sns.scatterplot(x='Fuel Type', y='CO2 Emissions(g/km)', data=df, hue='Fuel Type')

df\_N = df[df['Fuel Type']=='N']

index = df\_N.index

df\_N

for i in index:

df.drop(i, axis=0, inplace=True)

df[df['Fuel Type']=='N']

dums = pd.get\_dummies(df['Fuel Type'], prefix='Fuel\_Type', drop\_first=True)

dums[0:15]

frames = [df, dums]

df = pd.concat(frames, axis=1)

df

df.drop(['Fuel Type'], inplace=True, axis=1)

df.head()

X = df.drop(['CO2 Emissions(g/km)'], axis=1)

y = df['CO2 Emissions(g/km)']

X.shape

y.shape

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3, random\_state=42)

print('X\_train', X\_train.shape)

print('y\_train', y\_train.shape)

print('X\_test', X\_test.shape)

print('y\_test', y\_test.shape)

from sklearn.linear\_model import LinearRegression

modelLR = LinearRegression()

modelLR.fit(X\_train, y\_train)

modelLR.intercept\_

modelLR.coef\_

from sklearn.metrics import mean\_squared\_error, r2\_score

np.sqrt(mean\_squared\_error(y\_train, modelLR.predict(X\_train)))

np.sqrt(mean\_squared\_error(y\_test, modelLR.predict(X\_test)))

modelLR.score(X\_train, y\_train)

modelLR.score(X\_test, y\_test)

from sklearn.model\_selection import cross\_val\_score

cross\_val\_score(modelLR, X\_train, y\_train, cv=10, scoring='r2').mean()

cross\_val\_score(modelLR, X\_test, y\_test, cv=10, scoring='r2').mean()

from sklearn.ensemble import RandomForestRegressor

modelRF = RandomForestRegressor(n\_estimators=100, random\_state=0)

modelRF.fit(X\_train, y\_train)

np.sqrt(mean\_squared\_error(y\_train, modelRF.predict(X\_train)))

np.sqrt(mean\_squared\_error(y\_test, modelRF.predict(X\_test)))

modelRF.score(X\_train, y\_train)

modelRF.score(X\_test, y\_test)

cross\_val\_score(modelRF, X\_train, y\_train, cv=10, scoring='r2').mean()

cross\_val\_score(modelRF, X\_test, y\_test, cv=10, scoring='r2').mean()

from sklearn.tree import DecisionTreeRegressor

modelDTR = DecisionTreeRegressor(max\_depth=10, max\_features='sqrt')

modelDTR.fit(X\_train, y\_train)

np.sqrt(mean\_squared\_error(y\_train, modelDTR.predict(X\_train)))

np.sqrt(mean\_squared\_error(y\_test, modelDTR.predict(X\_test)))

modelDTR.score(X\_train, y\_train)

modelDTR.score(X\_test, y\_test)

cross\_val\_score(modelDTR, X\_train, y\_train, cv=10, scoring='r2').mean()

cross\_val\_score(modelDTR, X\_test, y\_test, cv=10, scoring='r2').mean()

from sklearn.neighbors import KNeighborsRegressor

modelKNN = KNeighborsRegressor()

modelKNN.fit(X\_train, y\_train)

np.sqrt(mean\_squared\_error(y\_train, modelKNN.predict(X\_train)))

np.sqrt(mean\_squared\_error(y\_test, modelKNN.predict(X\_test)))

modelKNN.score(X\_train, y\_train)

modelKNN.score(X\_test, y\_test)

cross\_val\_score(modelKNN, X\_train, y\_train, cv=10, scoring='r2').mean()

cross\_val\_score(modelKNN, X\_test, y\_test, cv=10, scoring='r2').mean()

# Future Enhancement

As we can see our models are overfeeding. Learning is less. We can create new data columns. We can drop the columns that are not important. We need to do feature engineering.

Feature Engineering:

Explore additional features or engineered features that can better capture the factors affecting carbon emissions, such as road conditions, traffic patterns, and driving behavior.

Advanced Algorithms:

Experiment with more advanced machine learning algorithms and techniques, such as deep learning, ensemble methods, or time series forecasting.

Online Learning:

Implement online learning techniques to adapt to changing conditions and continuously update the model as new data becomes available.

Hybrid Models:

Combine multiple machine learning models to create an ensemble model that leverages the strengths of each component model.

Explainable AI:

Use techniques for model interpretability and explainability to help users understand why specific emissions predictions are made.

Anomaly Detection:

Add anomaly detection capabilities to identify unusual driving patterns or emission levels that may indicate a vehicle issue or environmental anomaly.

Data Augmentation:

Augment training data with synthetic or augmented data to improve the model's robustness.

Cross-Validation Strategies:

Implement advanced cross-validation techniques to assess the model's generalization performance and reduce overfitting.

Hyperparameter Optimization:

Use automated hyperparameter tuning methods to find the best combination of hyperparameters for the model.

Model Ensemble:

Create an ensemble of multiple models, each trained on different subsets of data or with different feature subsets.

Continuous Monitoring:

Develop a system for continuous monitoring of the model's performance and retraining the model when necessary.

User Feedback Integration:

Integrate user feedback on prediction accuracy to fine-tune the model and address user-specific patterns.

Enhancements to the machine learning model should align with the specific goals and requirements of the Carbon Dioxide Emission Prediction system and consider the availability of data and computational resources. Regular model updates and retraining will be essential to ensure the system remains effective and reliable over time.