Fuel Consumption Ratings (2023)

EECS 3401 Group 2
Geon Kim (218831214)
Kevin Dao (218118190)
Rahil Rajbhandari (218512533)
Justin Cho (218332759)

1. Framing the Problem and Looking at the Big Picture.

Supervised, Unsupervised, or Reinforcement Learning?
Supervised: Columns are labelled to train algorithms to predict outcomes accurately

Classification task, Regression task, or something else?
Regression: Predicts a numerical value

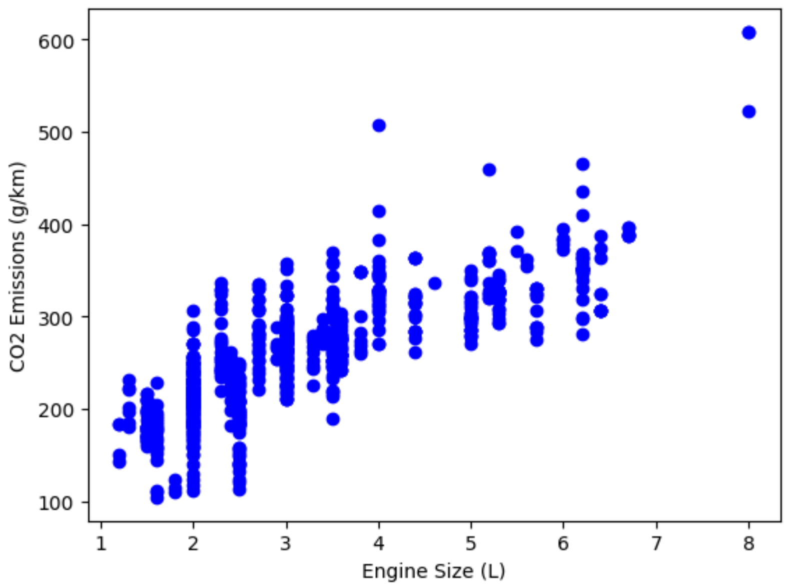
Batch learning or Online learning techniques?
Batch learning: We are using a small dataset that is trained only once, due to not being given extra data

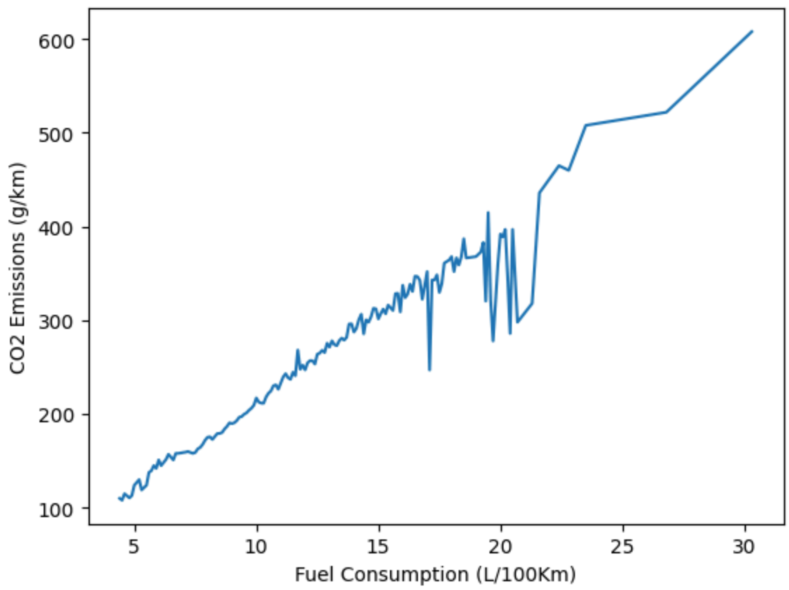
Instance-Based or Model-Based Learning
Model-based learning: The algorithm detects patterns to build a model to make a prediction.

Looking at the big picture
This will determine the estimated CO2 emissions for a vehicle given its engine specifications.
Using this information, one can determine if it is worth purchasing a vehicle or not by considering the impact of its CO2 emissions on their decision-making process.

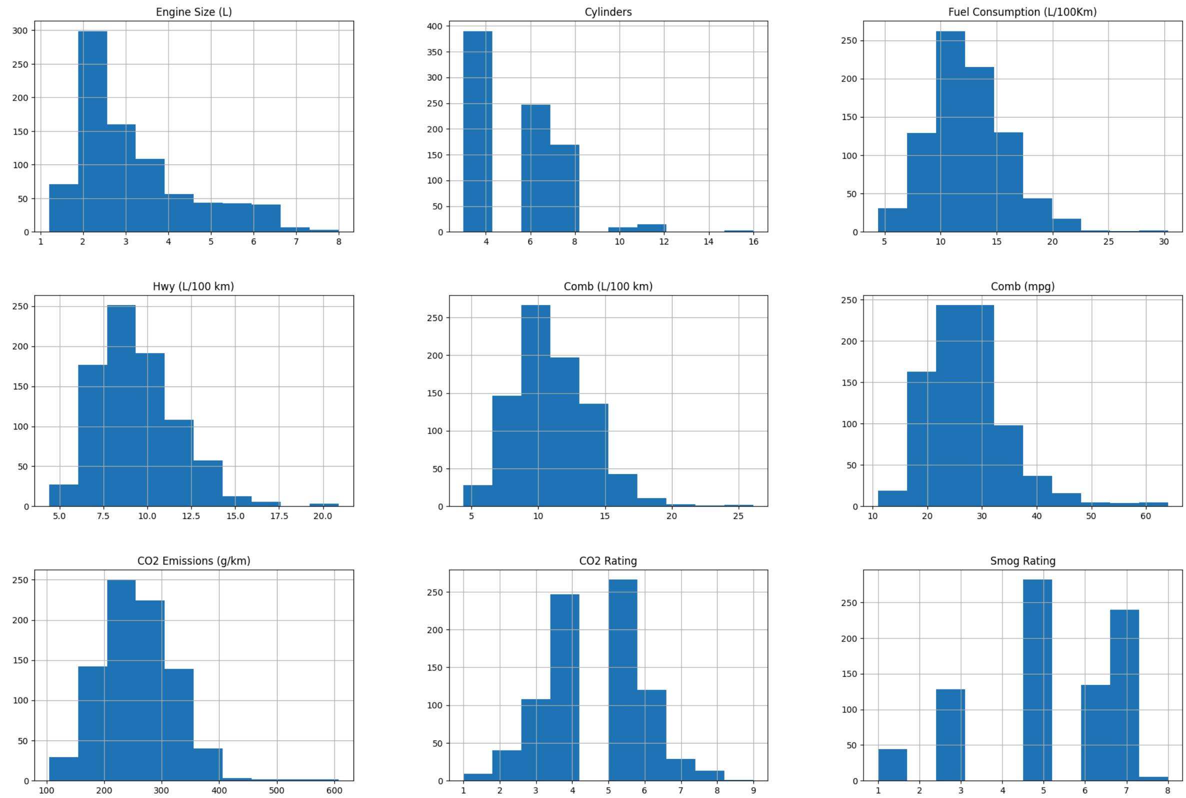
2. A Description of the Dataset and Graphs of EDA.

This dataset contains information on vehicles. This includes:
- Year - Direct Year the vehicle released
- Make - Manufacturer
- Model - Specific version of a car
- Vehicle Class - Type of vehicle such SUV, full-sized etc
- Engine Size (L)
- Cylinders - number of cylinders in the engine
- Transmission - type of transmission(gearbox)
- Fuel Type - gasoline, diesel etc
- Fuel Consumption (L/100km) - fuel consumption in cities
- Hwy Consumption (L/100km) - fuel consumption on highways
- Comb (L/100km) (mpg) - a combination of city fuel consumption and highway (55% city, 45% highway)
- CO2 Emissions - (g/km), target column to predict
- CO2 Rating (1-10) - tailpipe emission of carbon dioxide rated from 1 (worst) to 10 (best)
- Smog Rating (1-10) - tailpipe emission of smog-forming pollutants rated from 1 (worst) to 10 (best)





Engine Size and Fuel Consumption are the columns considered they have the most impact on the target column. Hence, those two graphs are presented to visualize the relationship between CO2 emission and two features each.

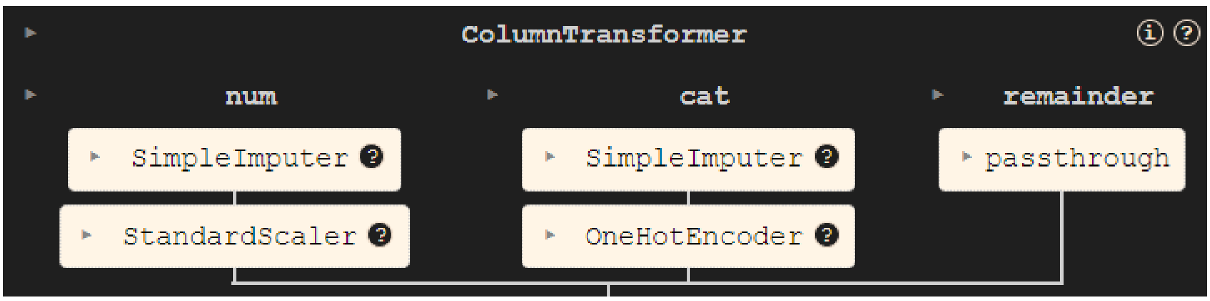


3. Data Cleaning and Preprocessing

Rows Dropped: rows 835 to 857 because they were explanation rows for some of the columns in the dataset. Dropped any instances where there were null or dummy values in addition.
Columns Dropped: [Transmission, Make, Year, Vehicle Class, Model, Comb (mpg)]
Reasoning:

1. Transmission, Make, Year, Vehicle Class, and Model do not impact CO2 emissions.
2. Comb (mpg) - Comb (L/100 km) already exists and is just another way to display the fuel consumption but in miles per gallon.

We split the dataset into 80% training and 20% testing.



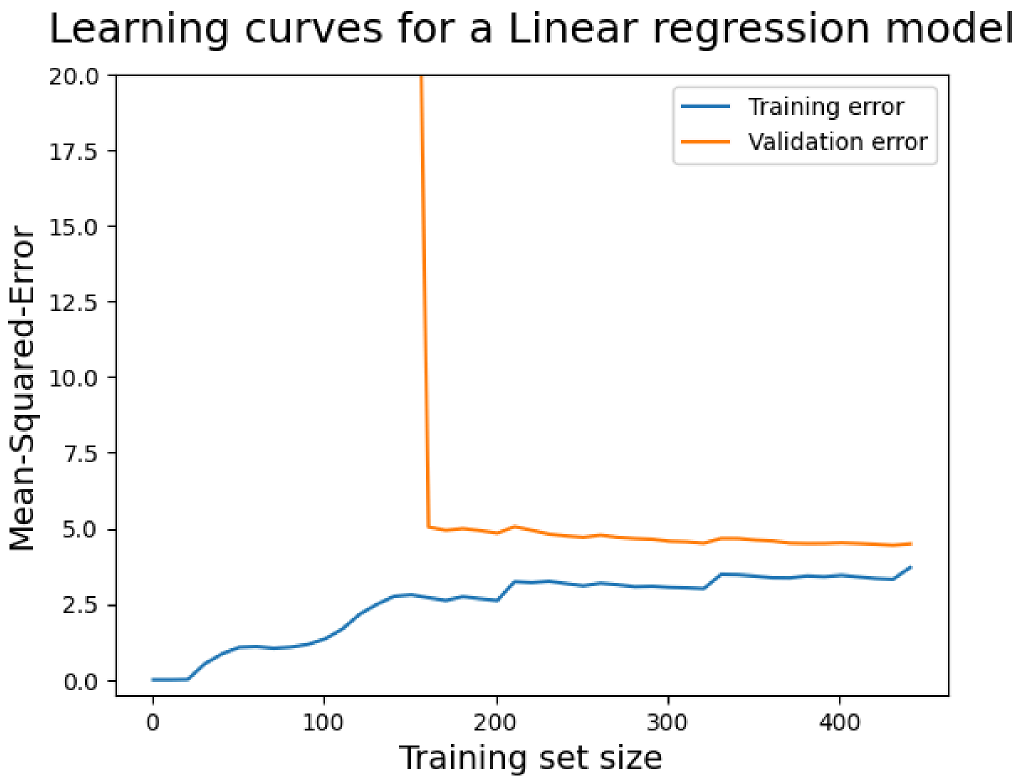
4. Training and Evaluation of Three Machine Learning Algorithms

1. Linear Regression Model
2. Elastic Net Model
3. Decision Tree Regressor Model

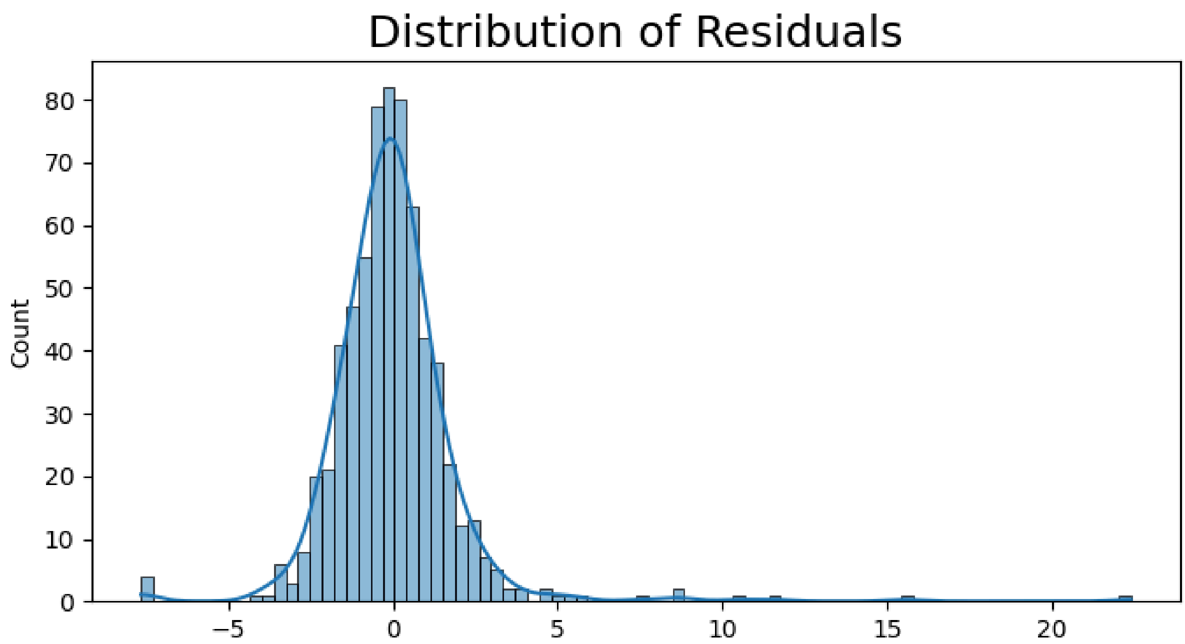
	fit_time	score_time	test_neg_mean_absolute_error	test_neg_mean_squared_error	test_neg_root_mean_squared_error	test_r2
Linear regression	0.001334	0.002002	-1.286814	-4.521086	-2.106325	0.998902
Elastic net	0.002669	0.001335	-3.848991	-64.569211	-7.788080	0.985757
Decision tree regressor	0.012678	0.002002	-2.690691	-91.546547	-8.189840	0.981902

The best performance from the three models was Linear Regression due to mean squared error results. The R^2 corresponds to the accuracy which indicates how accurately the model predicts the target value.

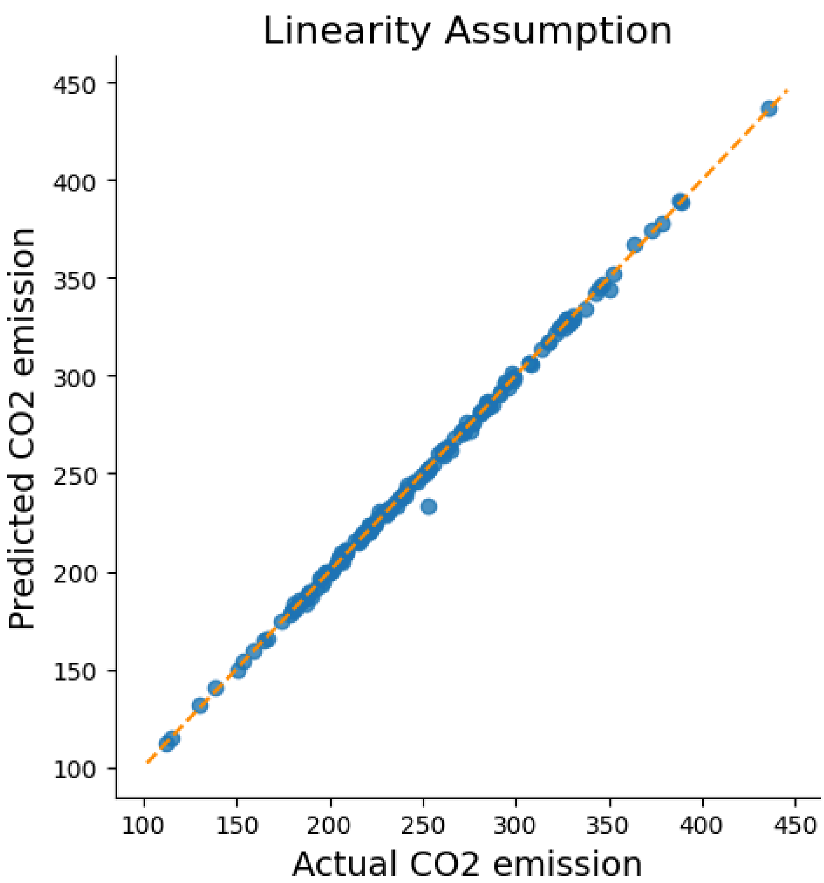
5. Graphs for the Best-Performing Algorithm.



As shown in the graph above, after the point that the training set size reaches about 150, training and validation errors tend to converge to 4.5 mean-squared error. This implies that the model's performance is steady enough while having a low error rate.



Generally, it is known that the model performs well if its residual distribution follows the shape of normal distribution. P-value, which is an indicator of how the graph is close to the shape of normal distribution curve, is given 0 for this model. This is below the threshold of 0.05 and meaning that the model does not perform well perhaps. However, the next graph shows the reason.



Since the prediction is very accurate, the residual is almost 0 for most cases(except outliers). This causes the residual distribution to skew to 0 value and its vicinity. Hence, the model's residual distribution cannot follow a normal distribution.

6. Any limitations you have run into.

Dataset values do not really deviate from each other, causing the dataset to have high linearity barring the outliers, as a result, the model is too accurate than initially predicted.

7. Next steps

A next step for this project could involve obtaining 2024 vehicle data and applying the algorithms to that dataset. By comparing the CO_2 emissions between the 2023 and 2024 years, we can analyze any observed differences between them and gain insight into any trends over time.

8. Appendix 1

Dataset URL:
https://www.kaggle.com/datasets/imtkaggleteam/fuel-concumption-ratings-2023
https://github.com/jewbe22/eecs3401-group2-groupProject/blob/main/src/CO2_emission.ipynb

GitHub URL:
https://github.com/jewbe22/eecs3401-group2-groupProject/blob/main/src/CO2_emission.ipynb

9. Appendix 2

```
import pandas as pd
import requests as rq
from io import StringIO

url =
'https://raw.githubusercontent.com/jewbe22/eecs3401-group2-groupProject/main/data/Fuel_Consumption_Ratings_2023.csv'
download = rq.get(url).content
co2_emission = pd.read_csv(StringIO(download.decode(errors='ignore')), sep=',',
engine='python')
co2_emission

co2_emission.columns

# todo
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#rows 0-833 only contain the instances, the rest contain description of the features
co2_emission.isnull().sum()

#removing instances with null or dummy values
co2_emission.dropna(inplace=True)

co2_emission.describe()

#use this to analyze some trends i.e 75% of the average vehicle have < 3.6 engine
sizes
co2_emission.info()
corr_matrix = co2_emission.corr(numeric_only=True)
```

```
corr_matrix
#finding correlations
corr_matrix["Engine Size (L)"].sort_values(ascending=False)

#correlation with respect to engine size
corr_matrix["Fuel Consumption (L/100Km)"].sort_values(ascending=False)

#correlation with respect to fuel consumption
g1 = sns.lineplot(x="Engine Size (L)", y="CO2 Emissions (g/km)",
data=co2_emission, errorbar=None)
plt.show()
g2 = sns.lineplot(x="Fuel Consumption (L/100Km)", y="CO2 Emissions (g/km)",
data=co2_emission, errorbar=None)
plt.show()

# Scatterplot of Engine Size vs CO2 Emissions
X1 = co2_emission["Engine Size (L)"]
y1 = co2_emission["CO2 Emissions (g/km)"]
# Plot points
fig, pl = plt.subplots()
pl.scatter(X1, y1, color = 'b')
plt.xlabel("Engine Size (L)")
plt.ylabel("CO2 Emissions (g/km)")

# Scatterplot of Fuel Consumption vs CO2 Emissions
X = co2_emission["Fuel Consumption (L/100Km)"]
y = co2_emission["CO2 Emissions (g/km)"]
# Plot points
fig, pl = plt.subplots()
pl.scatter(X, y, color = 'b')
plt.xlabel("Fuel Consumption (L/100Km)")
plt.ylabel("CO2 Emissions (g/km)")
co2_emission.hist(figsize=(24, 16))
plt.show()

# todo

#removing irrelevant columns from the dataset
co2_emission=co2_emission.drop(['Transmission','Make','Year','Vehicle
Class','Model','Comb (mpg)'],axis=1)

co2_emission.head()

from sklearn.compose import ColumnTransformer
```

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

num_cols = ['Engine Size (L)','Fuel Consumption (L/100Km)', 'Hwy (L/100
km)','Comb (L/100 km)']
# dtype of elements in cat_cols are numeric but actually they are categorical, and
thus, they are subjects of oneHotEncoder()
cat_cols = ['Cylinders', 'Fuel Type', 'CO2 Rating', 'Smog Rating']

#create pipelines for numeric and categorical columns
num_pipeline = make_pipeline(SimpleImputer(strategy='mean'), StandardScaler())
cat_pipeline = make_pipeline(SimpleImputer(strategy='most_frequent'),
OneHotEncoder())

#use ColumnTransformer to set the estimators and transformations

preprocessing = ColumnTransformer([('num', num_pipeline, num_cols),
                                   ('cat', cat_pipeline, cat_cols)],
                                    remainder='passthrough' )

preprocessing

# Apply the preprocessing pipeline on the dataset

dataset_prepared = preprocessing.fit_transform(co2_emission)
feature_names=preprocessing.get_feature_names_out()
dataset_prepared

try:
    dataset_prepared = pd.DataFrame(data=dataset_prepared,
columns=feature_names)
except:
    dataset_prepared = pd.DataFrame.sparse.from_spmatrix(data=dataset_prepared,
columns=feature_names)

from sklearn.model_selection import train_test_split

X = dataset_prepared.drop(["remainder__CO2 Emissions (g/km)"], axis=1)
y = dataset_prepared["remainder__CO2 Emissions (g/km)"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV, cross_validate, cross_val_score

RANDOM_STATE = 42

linear_model = LinearRegression()
linear_params = {
    'fit_intercept':[True, False],
    'copy_X':[True],
    'n_jobs':[None, 1, 2, 5, 10]
}

tree_model = DecisionTreeRegressor()
tree_params = {
    'criterion':['squared_error', 'friedman_mse', 'absolute_error', 'poisson'],
    'splitter':['best','random'],
    'random_state':[RANDOM_STATE]
}

elastic_model = ElasticNet()
elastic_params = {
    'alpha':[0.1, 0.3, 0.5, 1, 1.5],
    'l1_ratio':[0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],
    'max_iter':[ 500, 1000, 2000, 4000],
    'copy_X':[True],
    'positive':[True, False],
    'random_state':[RANDOM_STATE]
}

scoring_methods = ['neg_mean_absolute_error', 'neg_mean_squared_error',
'neg_root_mean_squared_error', 'r2']
K_FOLD = 3
SCORING_WITH = 'neg_mean_squared_error'

linear_best = GridSearchCV(linear_model, linear_params,
scoring=scoring_methods, refit=SCORING_WITH, cv=K_FOLD)
tree_best = GridSearchCV(tree_model, tree_params, scoring=scoring_methods,
refit=SCORING_WITH, cv=K_FOLD)
elastic_best = GridSearchCV(elastic_model, elastic_params,
scoring=scoring_methods, refit=SCORING_WITH, cv=K_FOLD)
models = [('Linear regression',linear_best),
```

```
        ('Decision tree regressor',tree_best),
        ('Elastic net',elastic_best)]

cv_result = {}
temp = []
for (name, model) in models:
   model.fit(X_train, y_train)
   cv_result[name] = pd.DataFrame(cross_validate(model.best_estimator_, X_train,
y_train, scoring=scoring_methods ,cv=K_FOLD))
   temp.append((name, model.best_estimator_))

# models becomes the list of algorithms with their best parameter set
models = temp

stat_columns = cv_result[models[0][0]].mean().index
statistic = pd.DataFrame(columns=stat_columns)
model_names = [models[i][0] for i in range(len(models))]
for model in model_names:
   statistic.loc[-1] = cv_result[model].mean()
   statistic.index = statistic.index + 1

statistic.index = model_names

# neg_mean_squared_error is used since it has a property such that the model is
intuitively better when the score is greater
# unlike mean_squared_error
statistic.sort_values(by='test_' + SCORING_WITH, inplace=True, ascending=False)
statistic

#source : https://www.dataquest.io/blog/learning-curves-machine-learning/
from sklearn.model_selection import learning_curve
train_size = np.arange(start=1, stop=int(X_train.shape[0] * float(1 - 1/K_FOLD)),
step=10)

#finding the best estimator
best_estimator = None
for pair in models:
   if pair[0] == statistic.head(1).index:
      best_estimator = (pair[0], pair[1])
      break

# best_estimator == (name, model_object)
train_sizes, train_scores, validation_scores =
learning_curve(estimator=best_estimator[1],
```

```
                                       X=X_train, y=y_train, train_sizes=train_size,
                                       cv=K_FOLD, scoring=SCORING_WITH)

# reverting the sign of the score into positive since neg_mean_squared_error is used
for scoring
train_scores_mean = -train_scores.mean(axis = 1)
validation_scores_mean = -validation_scores.mean(axis = 1)

plt.plot(train_sizes, train_scores_mean, label = 'Training error')
plt.plot(train_sizes, validation_scores_mean, label = 'Validation error')
plt.ylabel('Mean-Squared-Error', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title(f'Learning curves for a {best_estimator[0]} model', fontsize = 18, y = 1.03)
plt.legend()
plt.ylim(-0.5, 20)

#source:https://medium.com/swlh/multi-linear-regression-using-python-44bd0d10082
d
from statsmodels.stats.diagnostic import normal_ad
import statsmodels.api as sm

# OLS is a type of linear regression which has some useful extra attributes such as
resid(==residual)
X_train_ols = sm.add_constant(X_train)
X_test_ols = sm.add_constant(X_test)
olsmod = sm.OLS(y_train, X_train_ols).fit()
y_pred = olsmod.predict(X_test_ols)
residual = olsmod.resid

# Performing the test on the residuals
p_value = normal_ad(residual)[1]
print(f'p-value from the test Anderson-Darling test below 0.05 generally means
non-normal: {p_value}')

# Plotting the residuals distribution
plt.subplots(figsize=(8, 4))
plt.title('Distribution of Residuals', fontsize=18)
sns.histplot(data=residual, kde=True)
plt.show()

# Reporting the normality of the residuals
if p_value < 0.05:
   print('Residuals are not normally distributed')
else:
```

```
    print('Residuals are normally distributed')

plot_columns = ['actual_CO2_emission','predicted_emission']
# to match the shape of y_test and y_pred
Y = pd.DataFrame(data=np.concatenate((np.reshape(a=y_test,
newshape=(y_test.shape[0],1)),
                                    np.reshape(a=y_pred, newshape=(y_pred.shape[0],1))),
axis=1),columns=plot_columns)

sns.lmplot(x=plot_columns[0], y=plot_columns[1], data=Y ,fit_reg=False)

# Plotting the diagonal line
line_coords = np.arange(start=Y[plot_columns].min().min()-10,
                        stop=Y[plot_columns].max().max()+10)
plt.plot(line_coords, line_coords,  # X and y points
         color='darkorange', linestyle='--')

plt.ylabel('Predicted CO2 emission', fontsize=14)
plt.xlabel('Actual CO2 emission', fontsize=14)
plt.title('Linearity Assumption', fontsize=16)

plt.show()
```