Milestone #3 Report

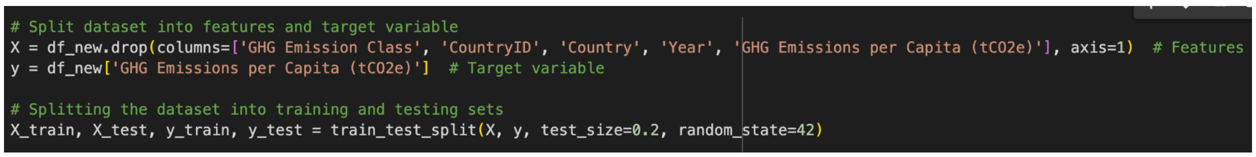
Sai Abhishek Gangineni, Robin Godinho, Marta Gonzalez
INST 737

Updates to Dataset

During Milestone 3, the dataset remained largely unchanged from the state it was in during Milestone 2. This decision was deliberate and stemmed from the thoroughness of the data exploration and preprocessing conducted in earlier stages. By maintaining the dataset's stability and consistency, we were able to preserve the integrity of the analyses and models developed in Milestone 2. Instead, any changes that were made were smaller adjustments focused on exploring the new machine learning techniques expected for this milestone, and optimizing predictive performance. These adjustments included fine-tuning feature selection, addressing any null values or anomalies identified, and optimizing data preprocessing steps to enhance model performance. By refining the dataset in this targeted manner, we were able to ensure that it remained well-suited for the specific methodologies and algorithms employed in Milestone 3.

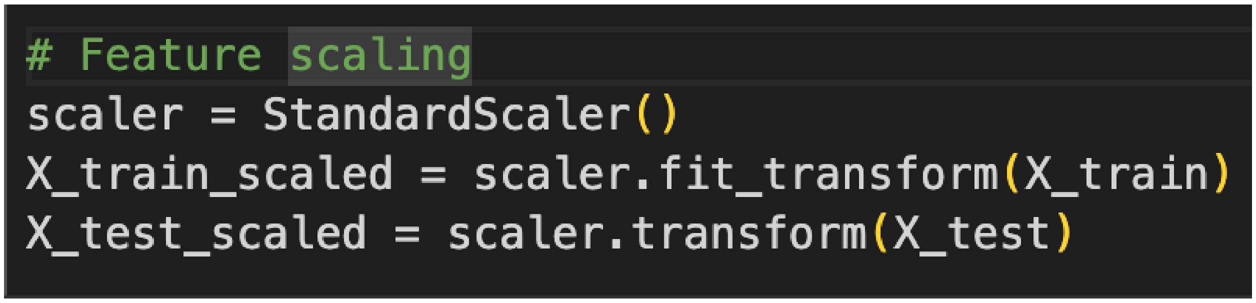
Support Vector Regressions

The first model that we explored in this milestone was the Support Vector Machine (SVM) to see how it applies to our research question, which was to predict greenhouse gas (GHG) emissions per capita for a country in a year based on various socioeconomic indicators. Given that our dependent variable, GHG Emissions per Capita, is a continuous feature, we chose to frame our approach for this model as a regression, rather than a classification. Beginning with categorizing the dataset into the independent and dependent features, we then split the dataset into training and testing sets.



```
# Split dataset into features and target variable
X = df_new.drop(columns=['GHG Emission Class', 'CountryID', 'Country', 'Year', 'GHG Emissions per Capita (tCO2e)'], axis=1)  # Features
y = df_new['GHG Emissions per Capita (tCO2e)']  # Target variable

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

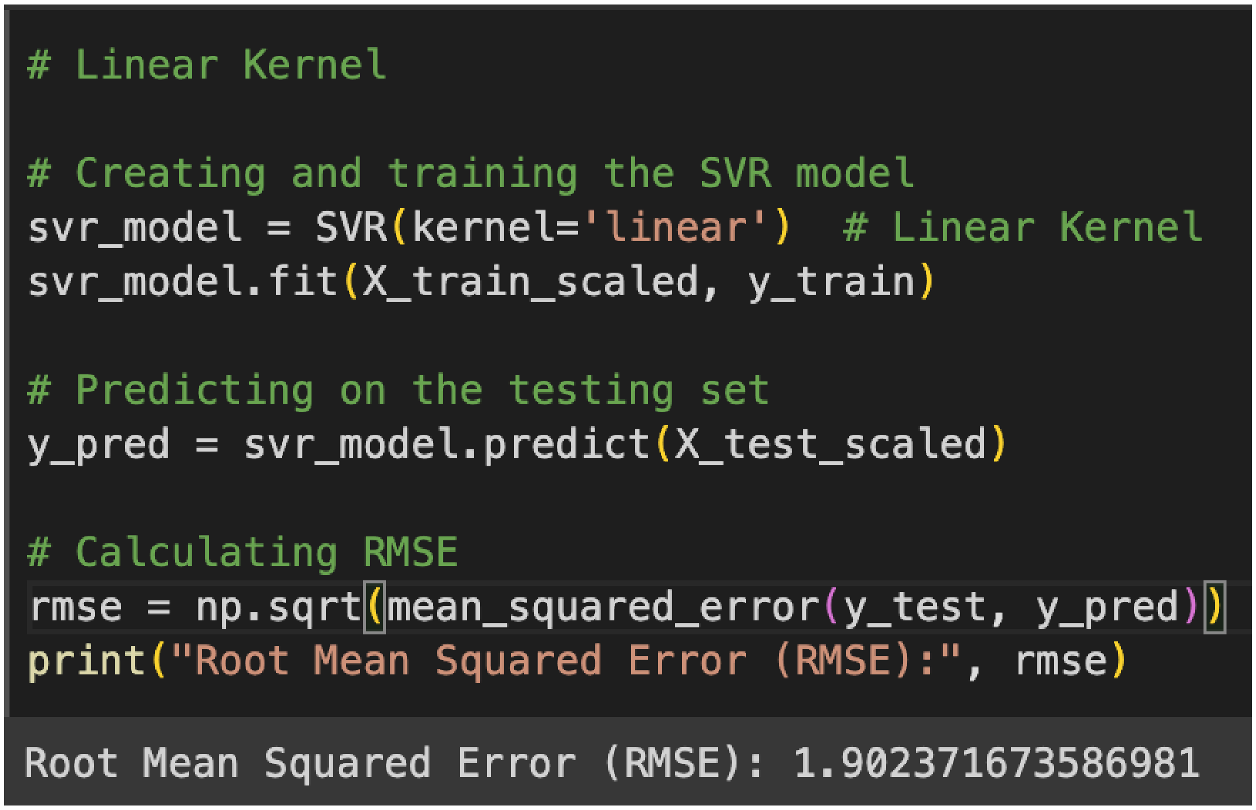
Initial implementation of the Python sklearn Support Vector Regression (SVR) model was a failure in that the code was continuously running for over 20 minutes without any results before being forcefully stopped. Research then found that the best way to speed up the training of SVR models is to scale the input features before training the model. Scaling can help improve the convergence speed of the optimization algorithm used by the SVR model. By scaling the input features, the optimization process during training is likely to converge faster, leading to reduced training times. This can be especially beneficial for datasets with many features or large ranges of feature values.



```
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Linear Kernel

After feature scaling the dataset, the SVR model ran smoothly and quickly. The first model that we created and trained was a linear kernel. After predicting the results of the model on the testing set, we then calculated the Root Mean Squared Error (RMSE), a value to determine the predictive accuracy of the model with lower values corresponding to higher accuracy, which came out to a result of **1.902371673586981**.

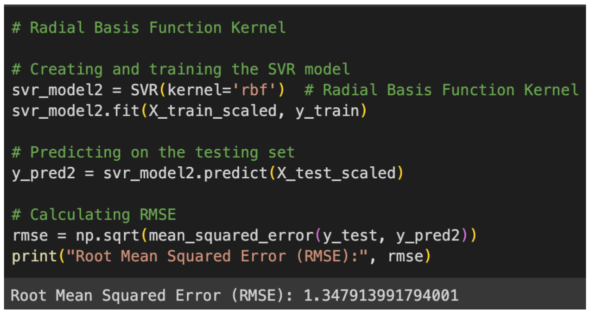


```
# Linear Kernel

# Creating and training the SVR model
svr_model = SVR(kernel='linear')  # Linear Kernel
svr_model.fit(X_train_scaled, y_train)

# Predicting on the testing set
y_pred = svr_model.predict(X_test_scaled)

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Root Mean Squared Error (RMSE): 1.902371673586981
```

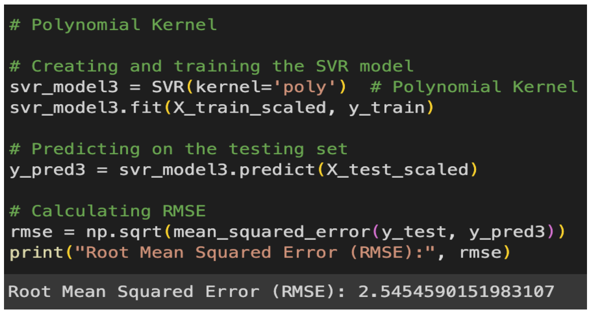
Non Linear Kernels

This same SVR analysis was then repeated multiple times with different non-linear kernel options offered by the Python sklearn SVR package. The resulting models were created, trained, and then used to predict on the testing set before a similar RMSE score was calculated to compare the results between different kernels.

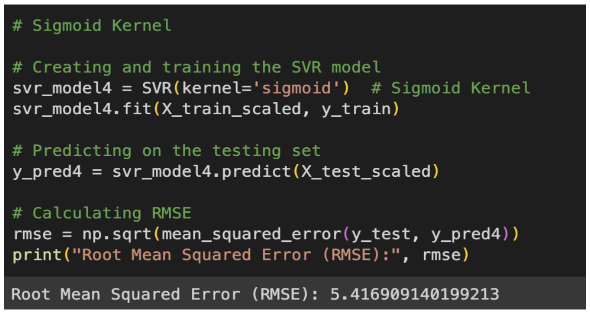


```
# Radial Basis Function Kernel

# Creating and training the SVR model
svr_model2 = SVR(kernel='rbf')  # Radial Basis Function Kernel
svr_model2.fit(X_train_scaled, y_train)

# Predicting on the testing set
y_pred2 = svr_model2.predict(X_test_scaled)

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred2))
print("Root Mean Squared Error (RMSE):", rmse)

Root Mean Squared Error (RMSE): 1.347913991794001
```

The three kernels tested were the Radial Basis Function, Polynomial, and Sigmoid Kernels. The best results were from the **Radial Basis Function** Kernel which had a RMSE rate as low as **1.347913991794001**, better than the linear kernel. The remaining two kernels, Polynomial and Sigmoid, however returned results that were much worse than the linear one with RMSE rates as high as 2.55 and 5.42 respectively.



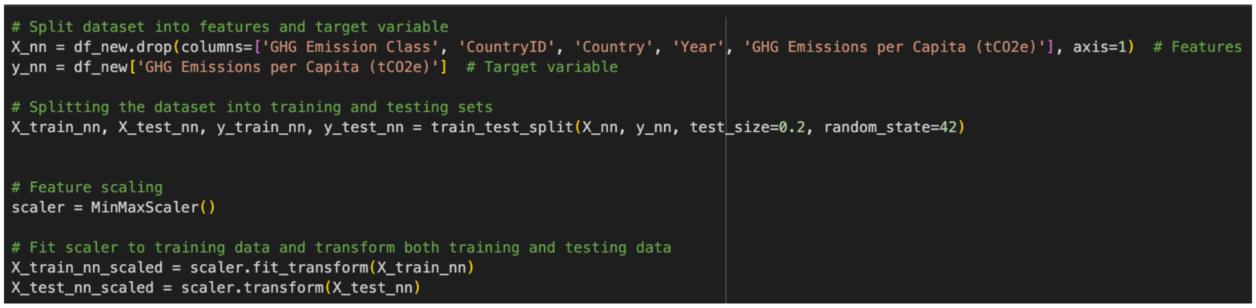
```
# Polynomial Kernel

# Creating and training the SVR model
svr_model3 = SVR(kernel='poly')  # Polynomial Kernel
svr_model3.fit(X_train_scaled, y_train)

# Predicting on the testing set
y_pred3 = svr_model3.predict(X_test_scaled)

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred3))
print("Root Mean Squared Error (RMSE):", rmse)

Root Mean Squared Error (RMSE): 2.5454590151983107
```



```
# Sigmoid Kernel

# Creating and training the SVR model
svr_model4 = SVR(kernel='sigmoid')  # Sigmoid Kernel
svr_model4.fit(X_train_scaled, y_train)

# Predicting on the testing set
y_pred4 = svr_model4.predict(X_test_scaled)

# Calculating RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred4))
print("Root Mean Squared Error (RMSE):", rmse)

Root Mean Squared Error (RMSE): 5.416909140199213
```

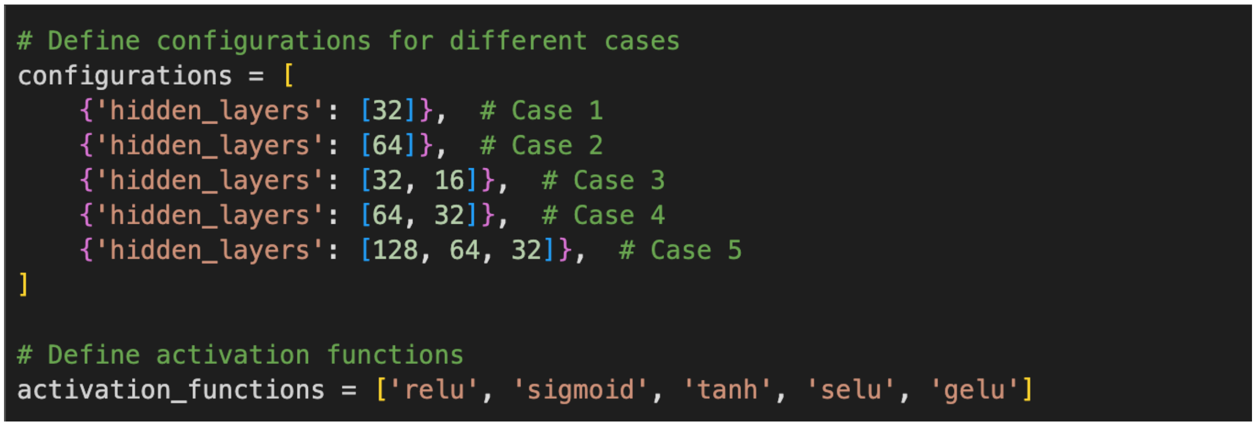
Neural Networks

The next type of machine learning model that we explored was Neural Networks. Using the TensorFlow Keras Python API, we created and trained a Neural Network. The first step was to normalize the input variables to the [0-1] range which we did through the MinMaxScaler after we had split the dataset into the training and testing sets.



```
# Split dataset into features and target variable
X_nn = df_new.drop(columns=['GHG Emission Class', 'CountryID', 'Country', 'Year', 'GHG Emissions per Capita (tCO2e)'], axis=1)  # Features
y_nn = df_new['GHG Emissions per Capita (tCO2e)']  # Target variable

# Splitting the dataset into training and testing sets
X_train_nn, X_test_nn, y_train_nn, y_test_nn = train_test_split(X_nn, y_nn, test_size=0.2, random_state=42)


# Feature scaling
scaler = MinMaxScaler()

# Fit scaler to training data and transform both training and testing data
X_train_nn_scaled = scaler.fit_transform(X_train_nn)
X_test_nn_scaled = scaler.transform(X_test_nn)
```

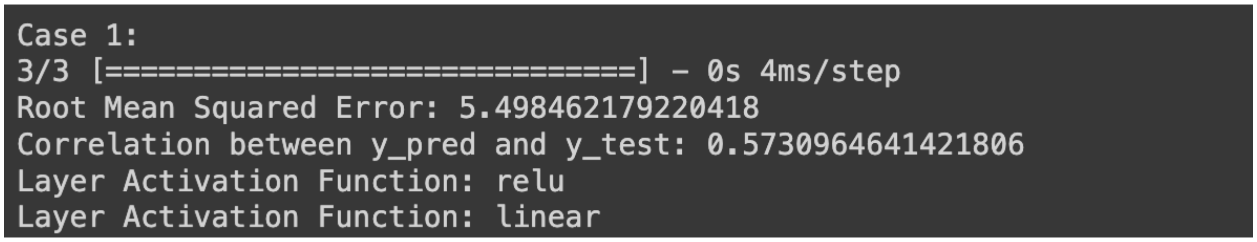
Then, we created multiple configurations corresponding to different neural network architectures with varying numbers of hidden layers and numbers of neurons per layer. We also created a list of different activation functions. Using a double for loop, we iterated through each type of network architecture and had it run for each type of activation function for a total of 25 trained neural network models. This allowed us to test various types of combinations and use their performance results to determine what type of architecture and activation function would yield the best predictive accuracy.



```
# Define configurations for different cases
configurations = [
    {'hidden_layers': [32]},  # Case 1
    {'hidden_layers': [64]},  # Case 2
    {'hidden_layers': [32, 16]},  # Case 3
    {'hidden_layers': [64, 32]},  # Case 4
    {'hidden_layers': [128, 64, 32]},  # Case 5
]

# Define activation functions
activation_functions = ['relu', 'sigmoid', 'tanh', 'selu', 'gelu']
```

Results

For each neural network that we ran, we predicted it on the testing set and collected both the Root Mean Squared Error rate along with the Correlation Coefficient between the prediction and test values. The lower the RMSE value and the higher the correlation coefficient, the better the model was at predicting the target variable, GHG Emissions per Capita.



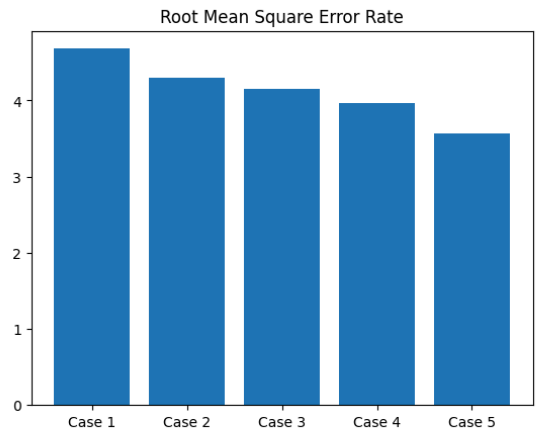
```
Case 1:
3/3 [==============================] - 0s 4ms/step
Root Mean Squared Error: 5.498462179220418
Correlation between y_pred and y_test: 0.5730964641421806
Layer Activation Function: relu
Layer Activation Function: linear
```

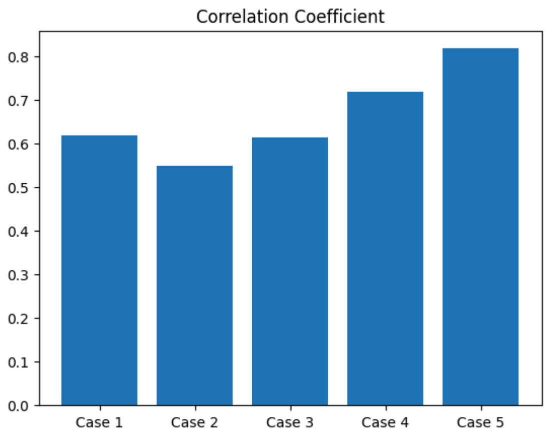
Case Number	**# Hidden Layers**	**# Neurons per Layer**	**Root Mean Squared Error**	**Correlation Coefficient**	**Activation Function**
Case 1	1	32	5.437090013389804	0.616462383592115	Rectified Linear Unit
Case 2	1	32	7.547931977829418	0.5246104380817211	Sigmoid
Case 3	1	32	5.998071336581122	0.4661836646097	Hyperbolic Tangent

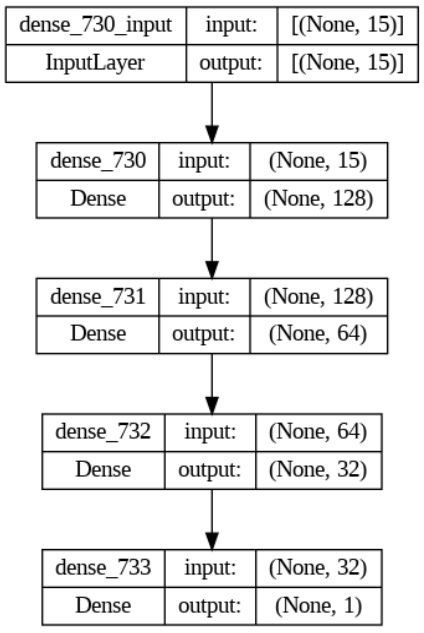
Case 4	1	32	4.7452426981919995	0.4377057435843911	Scaled Exponential Linear Unit
Case 5	1	32	6.957810755471686	0.43120005538098233	Gaussian error linear unit
Case 6	1	64	4.777165251360016	0.546226118419154	Rectified Linear Unit
Case 7	1	64	6.044517113490647	0.5765505707755293	Sigmoid
Case 8	1	64	4.432613845218229	0.5956509752221344	Hyperbolic Tangent
Case 9	1	64	4.346202796882004	0.5897385911937996	Scaled Exponential Linear Unit
Case 10	1	64	4.365786723500212	0.48037597557181677	Gaussian error linear unit
Case 11	2	32, 16	4.6223195101149175	0.4781117892841893	Rectified Linear Unit
Case 12	2	32, 16	8.533886245584421	0.4924747076714421	Sigmoid
Case 13	2	32, 16	5.66004473288448	0.4362142708133205	Hyperbolic Tangent
Case 14	2	32, 16	4.37399911277423	0.6067081752808078	Scaled Exponential Linear Unit
Case 15	2	32, 16	4.195841228372153	0.6402514090579895	Gaussian error linear unit
Case 16	2	64, 32	4.261668662724245	0.5802107835662128	Rectified Linear Unit
Case 17	2	64, 32	6.035051099833939	0.5683815191744203	Sigmoid
Case 18	2	64, 32	4.7145617807216125	0.5144135876941834	Hyperbolic Tangent
Case 19	2	64, 32	3.8636174565134165	0.8275142442665293	Scaled Exponential Linear Unit
Case 20	2	64, 32	4.170198120279107	0.6387950393817324	Gaussian error linear unit
Case 21	3	128, 64, 32	3.8350113760668587	0.7894371783504112	Rectified Linear Unit
Case 22	3	128, 64, 32	5.2894751345620845	0.6015013433323	Sigmoid

Case 23	3	128, 64, 32	4.768513983600344	0.3864433956965477	Hyperbolic Tangent
Case 24	**3**	**128, 64, 32**	**2.535051578268540 8**	**0.8964832604538904**	**Scaled Exponential Linear Unit**
Case 25	3	128, 64, 32	3.573342516404833	0.8105418317117417	Gaussian error linear unit

Based on the results we received, as the number of hidden layers and number of neurons per layer increased, there were consistent increases in the predictive accuracy of the model. That trend can be witnessed in these graphs as well:







The change in accuracy based on activation function was a less evident trend as different architectures yielded the best results for different activation functions. But, the overall best neural network model was the architecture with **3 Hidden Layers, with 128, 64, and 32 Neurons** for each of them and used a **Scaled Exponential Linear Unit (SELU)** activation function. This model yielded a RMSE rate as low as **2.5350515782685408** and a Correlation Coefficient as high as **89.65%**. The plot for this model can be seen to the right where there are 15 input variables, followed by the 3 hidden layers, and finally the single output layer. This was the most accurate model that we built showcasing how an increase in layers and neurons can improve upon the performance of the model.

Clustering

For the following section, we decide to analyze the relationship between the population total of each country and their respective GHG emission levels, to observe if there is any correlation between these two variables. Below is a scatter plot, displaying the relationship between the two variables. There are some visualization representations of clusters forming based on the given population, which indicates that this is a dataset to explore in this section.

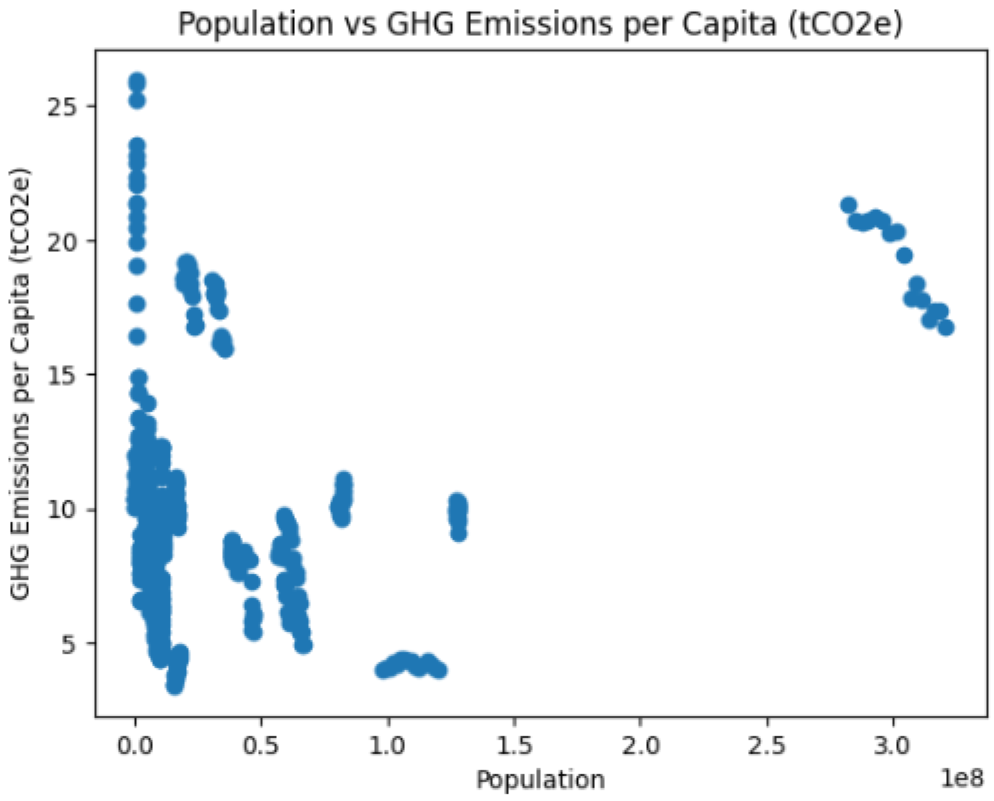


Figure: Scatter plot of population vs GHG emissions per capita

K-Means Clustering

The first clustering method that was explored was the partitional clustering by utilizing the K-means algorithm. The algorithm begins by selecting a preferred value for k, which will be the set number of clusters in which the data will be partitioned.

After repeating the k-means algorithm process a 4 times, and making updates to the centroids it was determined that the best value for the clustering is k=3. This means that the algorithm converges, and there is no significant difference in the centroids as the number of k increases beyond k=3.

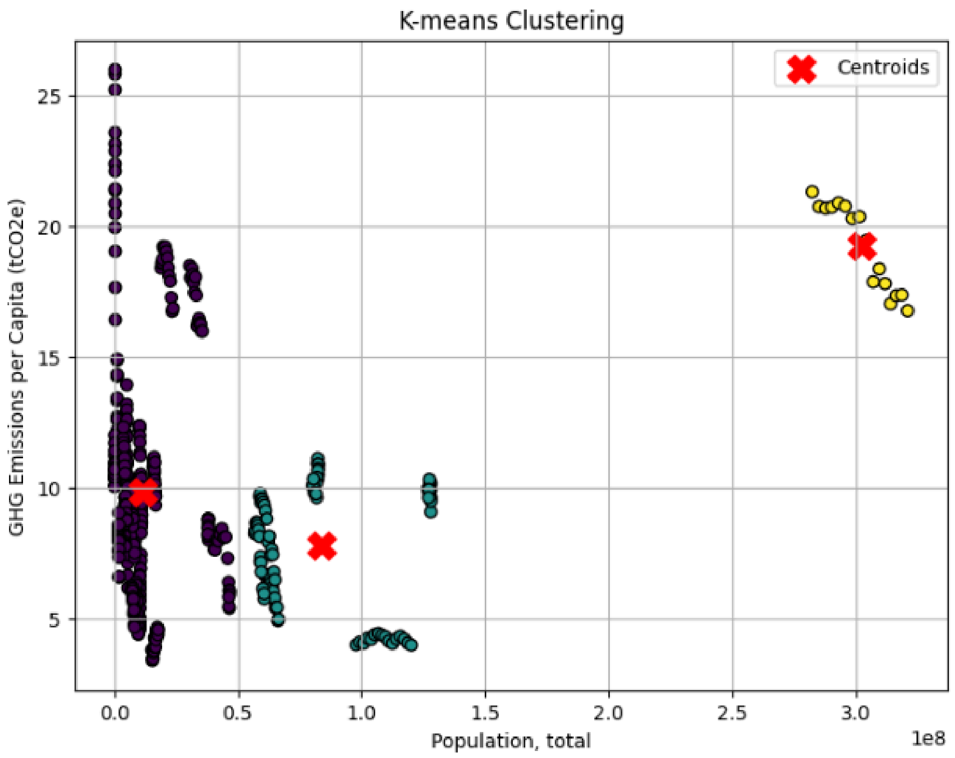


Figure: Three distinct clusters using K-Means Algorithm

Another method to determine the correct value for k is by plotting the Elbow curve. This is done by plotting the number of clusters (K) on the x-axis and the corresponding SSE on the y-axis. The SSE is the sum of squared errors. As the number of clusters increases, the SSE tends to decrease because smaller clusters can better fit the data. However, this decrease will reach a point of diminishing returns.

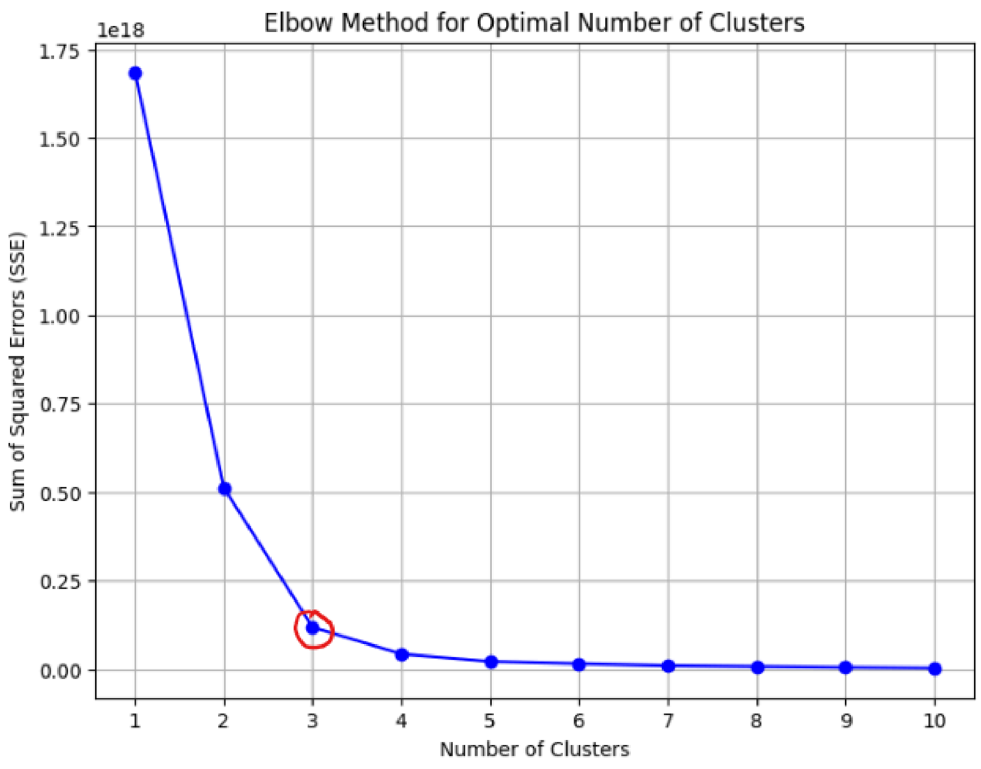


Figure: Elbow curve for optimal number of clusters

Above is the Elbow curve for the increased number of clusters. According to the curve, the 'elbow' can be found when k=3, which is the number that was chosen for the K-Means cluster.

Hierarchical Clustering

Hierarchical clusters organize the data into a tree-like structure (dendrogram), where the leaves of the tree represent individual data points, and the branches represent clusters of varying sizes. As clusters are merged, a dendrogram is constructed to visualize the hierarchical relationships between data points and clusters. The height at which two clusters are merged in the dendrogram represents the distance (or dissimilarity) at which they were merged.

In order to decide the correct number of clusters it was important to cut the dendrogram at a certain height to obtain a particular number of clusters. After testing various values for the dendrogram, it was determined that the best number of clusters is 3. Each cluster had 16, 320 and 144 sample elements respectively as shown in the dendrogram figure below.

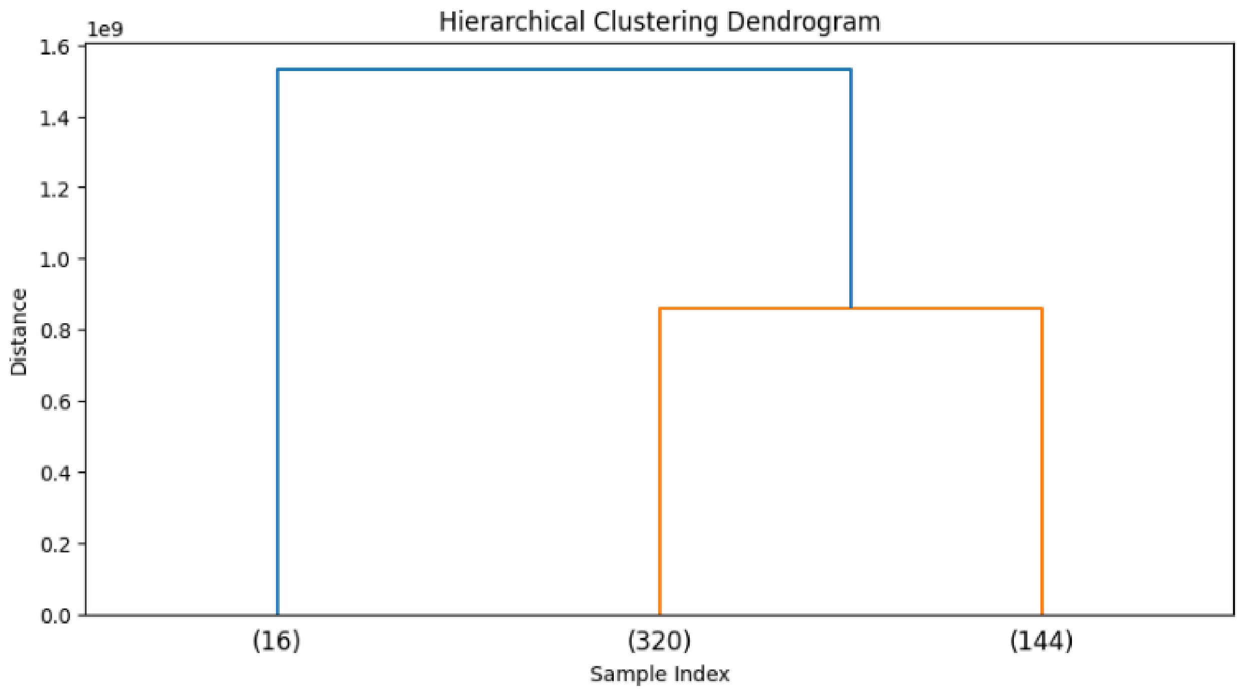


Figure: Hierarchical Clustering of Population vs GHG emissions per capita

Density-based Clustering

Density-based clustering is a clustering technique that identifies clusters based on the density of data points in the feature space. Unlike partitioning or hierarchical clustering, density-based clustering does not require specifying the number of clusters beforehand and can discover clusters of arbitrary shapes and sizes.

For this density-based clustering we used the BDSCAN algorithm. The BDSCAN uses two parameters, epsilon (ε) which is a distance threshold that defines the neighborhood of a data point, and MinPts which is the minimum number of points required to form a dense region. For our given dataset we chose to have epsilon at 0.5 and our dense regions to have at least 5 sample points. The following cluster was generated using 1000 sample tries with 42 random states.

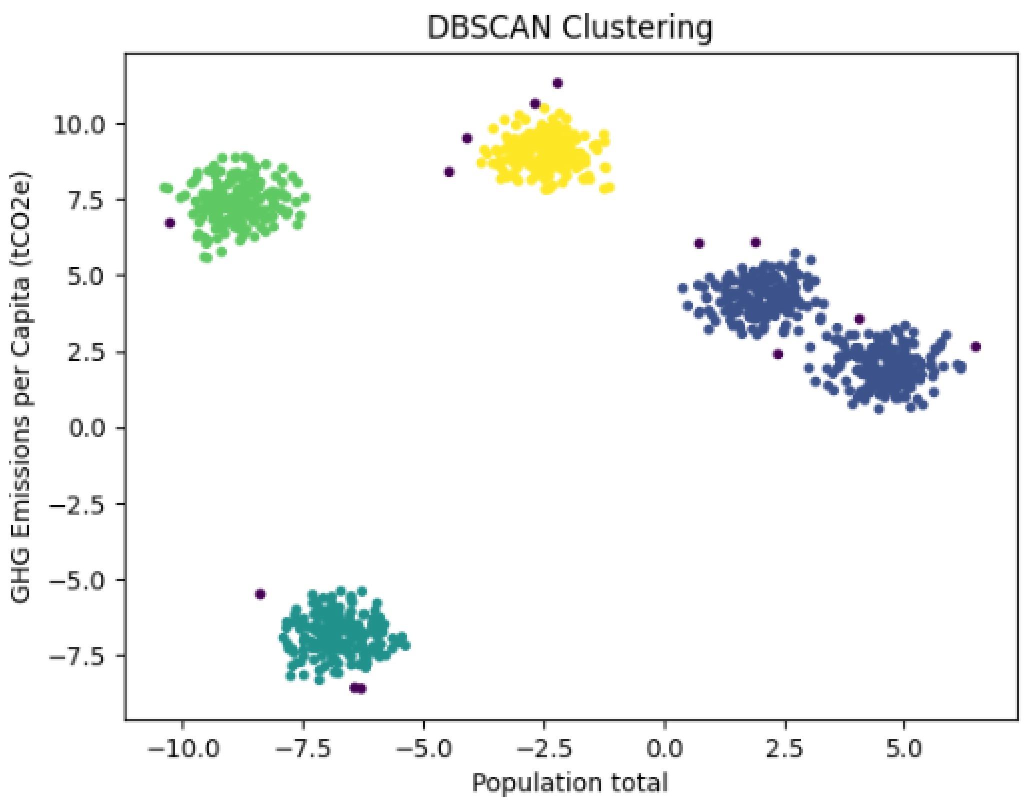


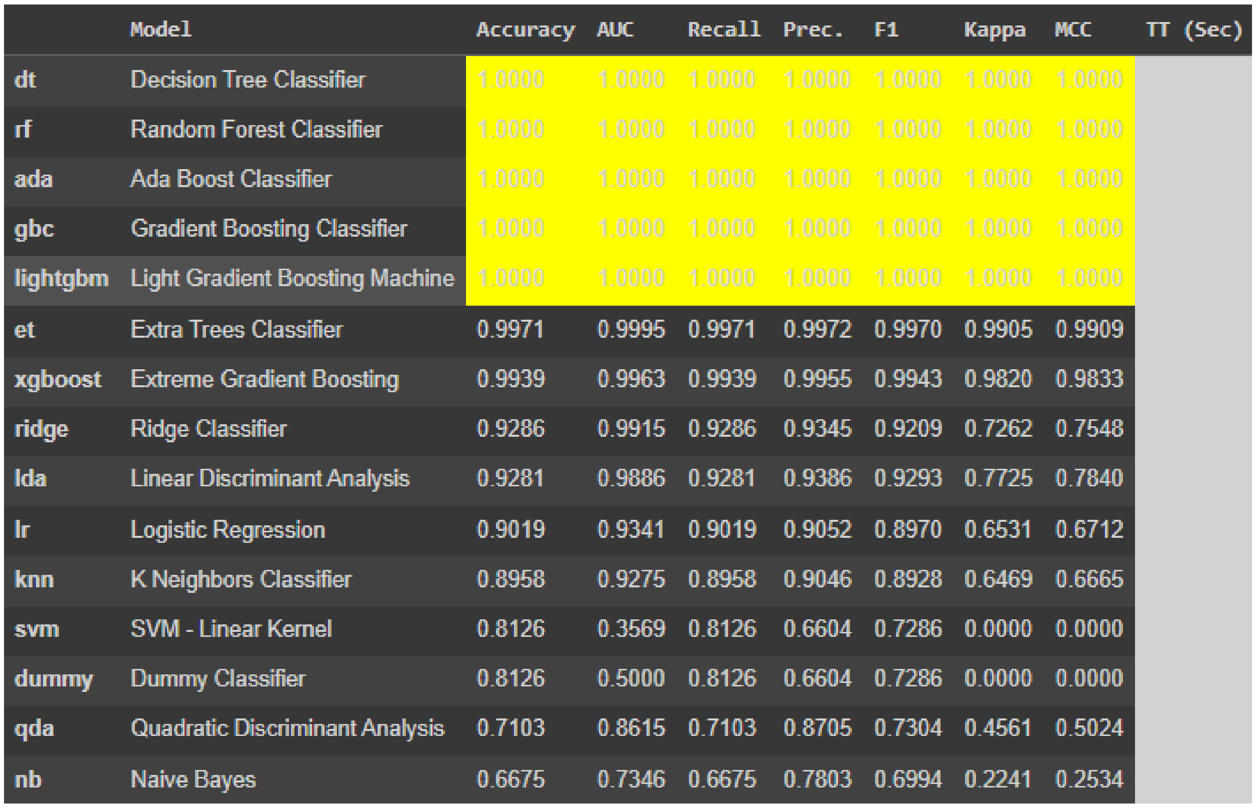
Figure: DBSCAN clustering

According to the density-based clustering figure above, the algorithm was able to generate 4 distinct clusters for the dataset. The model appears to have performed really well given the epsilon and distance threshold. The purple points are the core points. If the number of neighbors is greater than or equal to MinPts, the point is marked as a core point.

Comparative Analysis

The classification task method that we have decided to use for this comparative analysis is the decision tree model that was implemented in milestone 2. After running the Caret package in python, it was possible to obtain a list of all of the different models, and how well they performed in comparison to one another.

In question 1 of this milestone 3 we made use of the SVM model with a linear kernel, which performed relatively well with prediction 81 percent. In question 2 of milestone 3, we implemented neural networks, which performed significantly better with a prediction accuracy of 89 percent. Lastly, we compared it to question 3 from milestone 2, which made use of decision trees and random forest models. These two models performed the best as classification methods with an accuracy of 100 percent as shown in results from the figure below.



	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
dt	Decision Tree Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
rf	Random Forest Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
ada	Ada Boost Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
gbc	Gradient Boosting Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
lightgbm	Light Gradient Boosting Machine	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
et	Extra Trees Classifier	0.9971	0.9995	0.9971	0.9972	0.9970	0.9905	0.9909	
xgboost	Extreme Gradient Boosting	0.9939	0.9963	0.9939	0.9955	0.9943	0.9820	0.9833	
ridge	Ridge Classifier	0.9286	0.9915	0.9286	0.9345	0.9209	0.7262	0.7548	
lda	Linear Discriminant Analysis	0.9281	0.9886	0.9281	0.9386	0.9293	0.7725	0.7840	
lr	Logistic Regression	0.9019	0.9341	0.9019	0.9052	0.8970	0.6531	0.6712	
knn	K Neighbors Classifier	0.8958	0.9275	0.8958	0.9046	0.8928	0.6469	0.6665	
svm	SVM - Linear Kernel	0.8126	0.3569	0.8126	0.6604	0.7286	0.0000	0.0000	
dummy	Dummy Classifier	0.8126	0.5000	0.8126	0.6604	0.7286	0.0000	0.0000	
qda	Quadratic Discriminant Analysis	0.7103	0.8615	0.7103	0.8705	0.7304	0.4561	0.5024	
nb	Naive Bayes	0.6675	0.7346	0.6675	0.7803	0.6994	0.2241	0.2534	

Figure: Caret package with accuracy of different ML models

Cross Validation

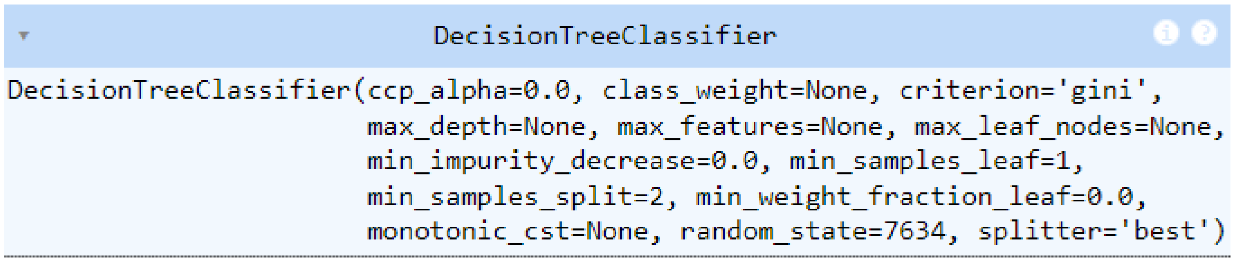
For the cross validation techniques, we decided to compute k-fold cross-validation with k=5 and k=10, and the LOOCV (leave-One-Out Cross Validation). These three cross validation techniques proved to work the best for our target feature which was the 'GHG emission Class.'

5-fold Cross Validation
In 5-fold cross-validation, the model is trained and evaluated 5 times. Each time, a different fold is held out for validation while the remaining 4 folds are used for training. The 5-fold CV took the least computational time since it required less training and evaluation of the models.

10-fold Cross Validation
In 10-fold cross-validation, the model is trained and evaluated 10 times. Each time, a different fold is held out for validation while the remaining 9 folds are used for training. Generally, 10-fold CV provides a more statistically stable estimate of model performance compared to 5-fold CV.

LOOCV
For each training iteration, a model is trained using all data points except the one being left out. The model is then evaluated on the single data point that was left out. This gives an estimate of how well the model generalizes to unseen data points.

For three of the cross validation techniques, the decision tree classifier came out to be the best model for our selected feature of predicting the GHG emission class of low, mid or high.

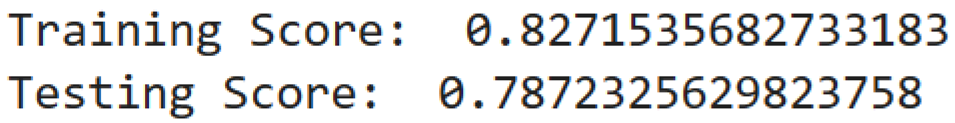


```
DecisionTreeClassifier

DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       monotonic_cst=None, random_state=7634, splitter='best')
```

Feature Selection

Filter Method - Linear Regression

The first feature selection method applied was the filter method and it was applied to the linear regression model we had created in Milestone 2. We first began by fitting the original model of features ['Fossil fuel energy consumption (% of total)', 'GDP per capita, PPP (current international $)','Agricultural land (% of land area)', 'Forest area (% of land area)', 'Urban population (% of total population)', 'Access to Clean Fuels and Technologies for cooking (% of total population)', 'Access to electricity (% of rural population with access)', 'Access to electricity (% of total population)', 'Access to electricity (% of urban population with access)', 'Energy intensity level of primary energy (MJ/PPP)', 'Renewable electricity share of total electricity output (%)', 'Renewable energy share of TFEC (%)', 'Population, total', 'Total electricity output per capita (kWh)', 'Total final energy consumption per Capita (TFEC) (MJ)'] and the target variable ['GHG Emissions per Capita (tCO2e)'] and producing the training and testing model accuracy scores.



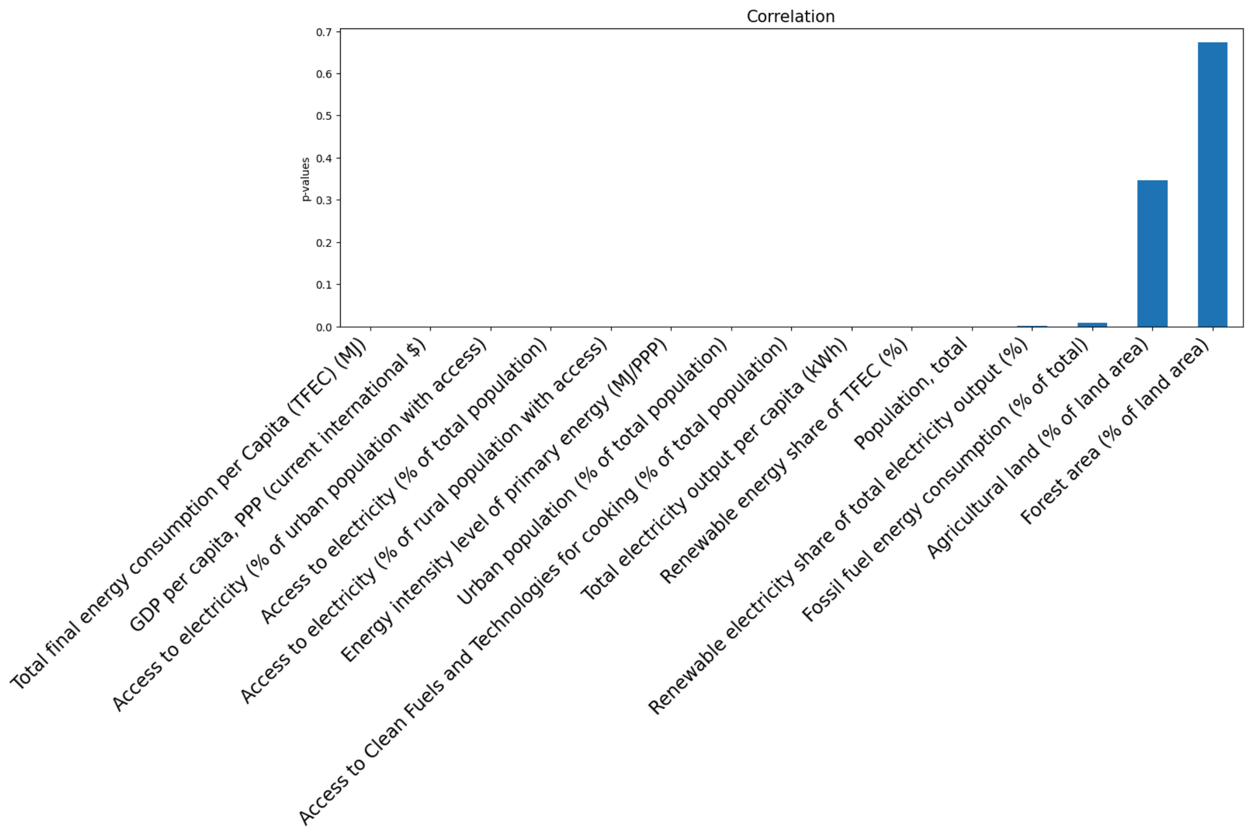
```
Training Score:  0.8271535682733183
Testing Score:  0.7872325629823758
```

The next step was finding the original correlation coefficient of that model to be able to compare to the new model later on. The original correlation coefficient was .894 and the original p-value was 5.127346688538688e-43.

```
r = np.corrcoef(y_test, y_pred)
print(r)
stats.pearsonr(y_test, y_pred)
```

```
[[1.         0.89426818]
 [0.89426818 1.        ]]
PearsonRResult(statistic=0.894268179300889, pvalue=5.127346688538688e-43)
```

We plotted, in ascending order, all of the p-values produced from the application of 'f_regression' on the training set. From this, we could clearly see, in order, which of our features have the most significance in predicting our target variable GHG emissions.



We obtained a list of the top 30% of features that were significant in predicting GHG emissions per capita:

```
sel = SelectPercentile(f_regression, percentile=30).fit(X_train, y_train)
sel.get_feature_names_out()
```

```
array(['GDP per capita, PPP (current international $)',
       'Access to electricity (% of rural population with access)',
       'Access to electricity (% of total population)',
       'Access to electricity (% of urban population with access)',
       'Total final energy consumption per Capita (TFEC) (MJ)'],
      dtype=object)
```

We obtained the new training and testing model accuracy scores after implementing the new list of selected features:

```
reg = linreg.fit(X_train_t, y_train)

print('Training Score: ', reg.score(X_train_t, y_train))
print('Testing Score: ', reg.score(X_test_t, y_test))
```

```
Training Score:  0.5290398650081647
Testing Score:  0.413288003027406
```

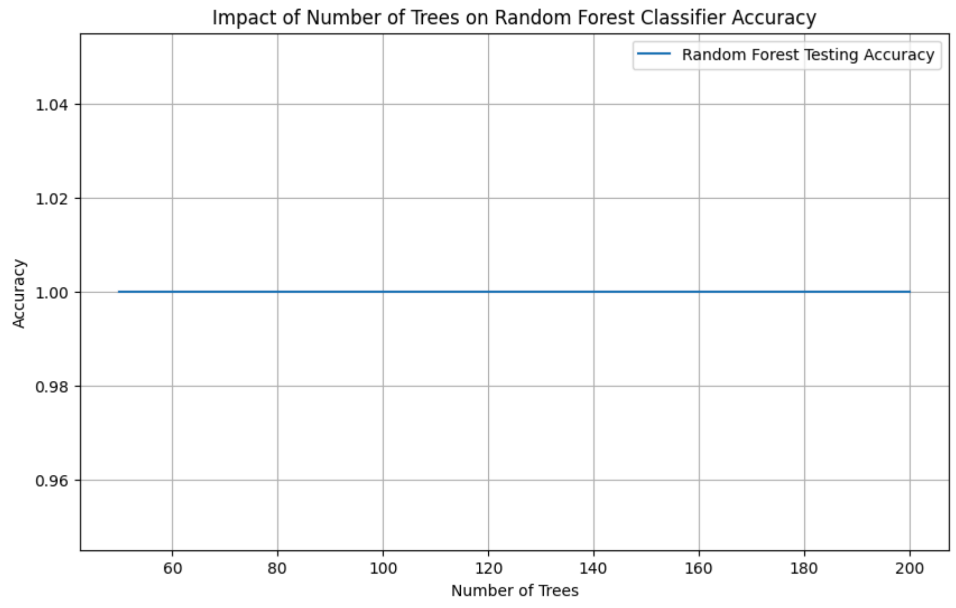
We found that both scores had decreased. We also obtained the new correlation coefficient and the p-value and found that the coefficient had decreased and the p-value had increased.

```
y_pred = reg.predict(X_test_t)
r = np.corrcoef(y_test, y_pred)
print(r)
stats.pearsonr(y_test, y_pred)
```

```
[[1.         0.64750742]
 [0.64750742 1.        ]]
PearsonRResult(statistic=0.647507422167676, pvalue=1.3292904983606864e-15)
```

Wrapper Method - Random Forest Classifier

Our next model was the random forest classifier model. We implemented a sequential feature selection wrapper method on this model and found that the accuracy of the model remained the same as it had previously.



We also found that our features of importance changes slightly from the first model, by eliminating the GHG emissions per capita feature as well as changes the importance scores. All of the new feature scores increased from the original model.

```
                                        Feature  Importance
3  Total final energy consumption per Capita (TFE...    0.378617
2          Total electricity output per capita (kWh)    0.375643
1                                  Population, total    0.155196
0                Agricultural land (% of land area)    0.090544
```

Embedded Method - Logistic Regression

Our next model was our previously created Logistic Regression model. We used this model to implement an embedded method using Lasso regression. We found two features of importance using this method:

```
Selected Features: ['Energy intensity level of primary energy (MJ/PPP)', 'Renewable energy share of TFEC (%)']
```

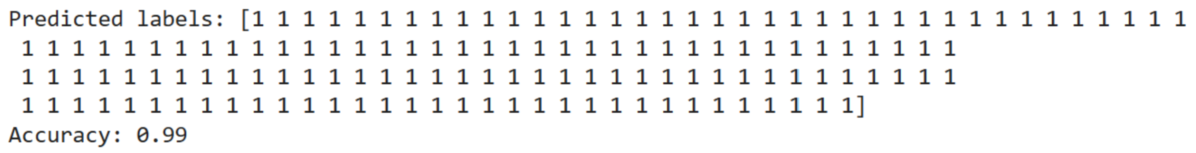
```
              precision    recall  f1-score   support

           0       0.03      1.00      0.05         2
           1       1.00      0.79      0.88       334

    accuracy                           0.79       336
   macro avg       0.51      0.89      0.47       336
weighted avg       0.99      0.79      0.88       336
```

```
Intercept: 5.171728610637628
Coefficients: [ 0.24918203 -0.05103496]
```

```
Independent Variable: Energy intensity level of primary energy (MJ/PPP)
Log-Odds for a Unit Increase: 0.24918203322387494
Odds Ratio for a Unit Increase: 1.2829755559913112
Independent Variable: Renewable energy share of TFEC (%)
Log-Odds for a Unit Increase: -0.05103495673632645
Odds Ratio for a Unit Increase: 0.9502454524721053
```

```
Most Predictive Features:
Energy intensity level of primary energy (MJ/PPP): 0.24918203322387494
Renewable energy share of TFEC (%): 0.05103495673632645
```

We found our model accuracy score to remain the same with the new model implementation using only the selected features.



```
Predicted labels: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Accuracy: 0.99
```

Ethical Issues

We found that our original dataset contained features that could not be directly compared between countries since they were dependent on the population of that country. It would not have been accurate to include a value for GHG emissions for China that looked representatively larger when, in fact, China also has a much larger population than many of the countries in our dataset, which would be directly influencing their emissions values. To solve this initial problem, we had to discard our original dataset and find a new one that contained the population totals for each country as well as our initial variables of interest. To make each value truly representative of each country, we normalized the values of our dependent variables by dividing them by the population totals. We then renamed the features with a ‘per capita’ identifier.

Our data is also very broad in scope because this type of data was some of the easiest to find. However, we are limited in producing more specific, detailed insights on the predictors of GHG emissions. Our data does not reflect the practices of each individual city or state in each country, nor does it allow us to identify specific industries or sectors that influence each of our features the most.

After feature selection method applications, some of our models were left with a limited range of features. While this can have benefits for the accuracy of predictions, it also meant that our models lacked certain context that other features provided. For example, our logistic regression model ended up with the features ‘Energy intensity level of primary energy (MJ/PPP)’ and ‘Renewable energy share of TFEC (%)’ but was no longer taking into account the access to electricity values that would provide these features with added context.

Contributions

Sai Abhishek Gangineni - Questions 1 and 2 of the code (Support Vector Regressions and Neural Networks), Contributed equally to the report and presentation.

Robin Godinho - Questions 3 and 4 of the code (Clustering and Comparative Analysis), Contributed equally to the report and presentation

Marta Gonzalez - Question 5 of the code (Feature Selection) and Question 6, Contributed equally to the report and presentation.