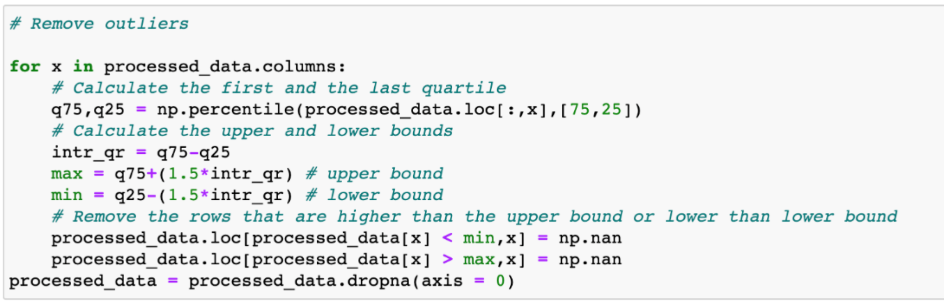
Introduction

The following report's objective is to study the relation between the emission of nitrous oxide (NOX) and the measurement of nine sensors planted in a gas station. In this task, I will build a model to predict the emissions of nitrous oxide (NOX) given the data generated by the other nine sensors. This is considered a regression problem because we want to predict the values of a continuous dependent variable (NOX) based on the other nine independent values (sensors) hence using a regression algorithm is the appropriate method to solve this problem. Overall, I will consider building a Linear Regression model as the baseline model. I am going to use Mean Squared Error (MSE) as the performance metric for this problem because it measures the average squared errors difference between the predicted values and the actual values. Hence, evaluate the robustness to choose the best model in terms of generalising to unseen data.

Getting the data ready to be used with the machine learning model

Before building the models, I will proceed with a data preprocessing, to ensure the quality of our data, this include the following steps:

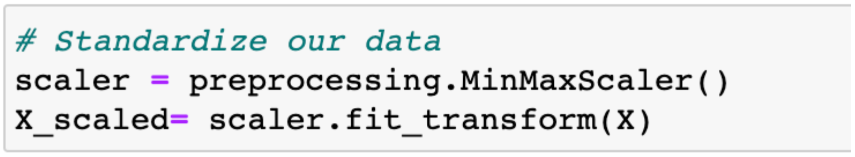
- First, I will remove the outliers, with the calculation of the upper and lower bound for each feature, to reduce the variance hence improve the model prediction accuracy.



```
# Remove outliers

for x in processed_data.columns:
    # Calculate the first and the last quartile
    q75,q25 = np.percentile(processed_data.loc[:,x],[75,25])
    # Calculate the upper and lower bounds
    intr_qr = q75-q25
    max = q75+(1.5*intr_qr) # upper bound
    min = q25-(1.5*intr_qr) # lower bound
    # Remove the rows that are higher than the upper bound or lower than lower bound
    processed_data.loc[processed_data[x] < min,x] = np.nan
    processed_data.loc[processed_data[x] > max,x] = np.nan
processed_data = processed_data.dropna(axis = 0)
```

Figure 1: Removing Outliers

- Second, I will standardize the independent variables (sensors) to remove the effect of the units using the MinMaxScaler() function:



```
# Standardize our data
scaler = preprocessing.MinMaxScaler()
X_scaled= scaler.fit_transform(X)
```

Figure 2: Standardizing Independent Variables

Methods

As shown in figure 3 below, we can notice from the correlation matrix that the correlation between every two variables is weak hence there is no linear relationship between the independent variables (sensors) and dependent variable (NOX) In spite of the low correlation between (NOX) and the other nine features as shown in the figure below, I will proceed with a multivariate regression model since we have a multivariate problem with nine explicative variables.

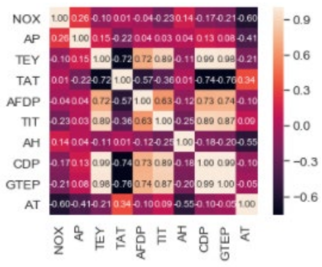


Figure 3: Correlation Matrix

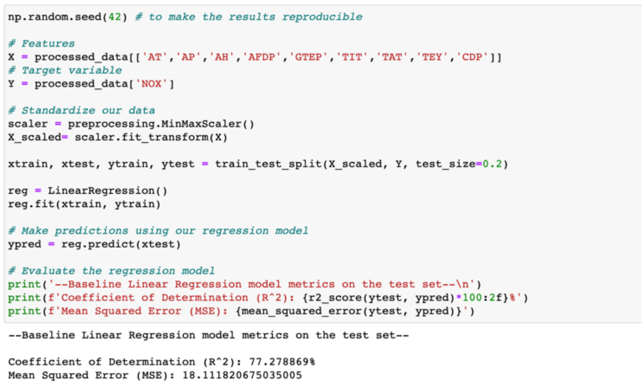
To increase the robustness of the model, I chose to build a decision tree models that evaluate the model by averaging their performance across multiple runs or repeats of cross-validation. When fitting a final model, it may be desirable to either increase the number of trees or the max depth until the variance of the model is reduced across repeated evaluations, or to fit multiple final models and average their predictions. Scikit-learn library provides plenty of regression algorithms. In our case, I chose to use the three following regression algorithms:

- Linear Regression (Baseline Algorithm)
- Random Forest Regressor
- Gradient Tree Boosting Regressor

I chose Linear Regression model as a baseline model since it has a simple and decent performance for our problem. The point here it to try to improve the baseline model and measure its performance against Random Forest and Gradient Tree Boosting regressors models. I tried the following performance metrics:

1- **R Squared Error (R^2):** because it describes the proportion of variance explained by our model hence it can be used to know how well a regression model performs in general.
2- Mean Squared Error (MSE): because it tells us how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line which represents the errors and squaring them. The squaring is necessary to remove any negative signs. The lower the value the better.

- **First Model: Linear Regression Model to Compare with Other Regressors**

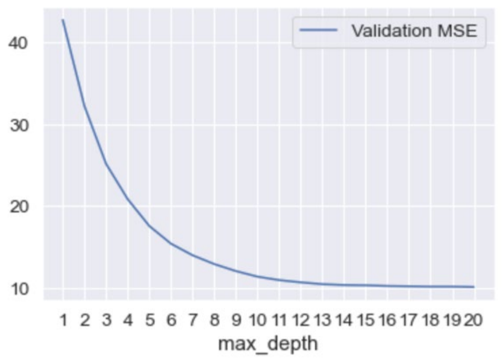


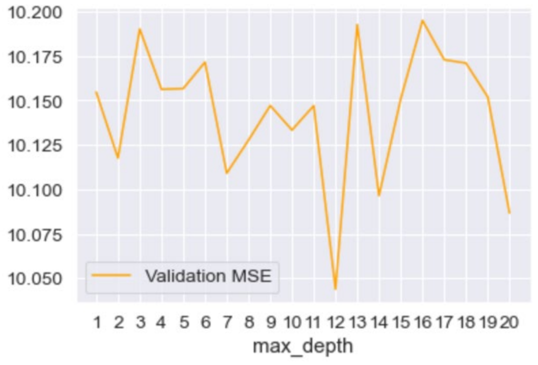
```
np.random.seed(42) # to make the results reproducible

# Features
X = processed_data[['AT','AP','AH','AFDP','GTEP','TIT','TAT','TEY','CDP']]
# Target variable
Y = processed_data['NOX']

# Standardize our data
scaler = preprocessing.MinMaxScaler()
X_scaled= scaler.fit_transform(X)

xtrain, xtest, ytrain, ytest = train_test_split(X_scaled, Y, test_size=0.2)

reg = LinearRegression()
reg.fit(xtrain, ytrain)

# Make predictions using our regression model
ypred = reg.predict(xtest)

# Evaluate the regression model
print('--Baseline Linear Regression model metrics on the test set--\n')
print(f'Coefficient of Determination (R^2): {r2_score(ytest, ypred)*100:2f}%')
print(f'Mean Squared Error (MSE): {mean_squared_error(ytest, ypred)}')
```

```
--Baseline Linear Regression model metrics on the test set--

Coefficient of Determination (R^2): 77.278869%
Mean Squared Error (MSE): 18.111820675035005
```

Figure 4: Linear Regression Model (Baseline Model)

Before implementing the baseline model, I have splitted the dataset into training and testing sets to calculate the accuracy of our model. Next, I Instantiated the model and fit it to the training dataset. Finally, I made predictions on the testing dataset and calculated model performance. We notice a decent R^2 value (77.27%) and a bit high mean squared error (18.11) that indicates adequate performance of the baseline model.





Random Forest Regressor *Gradient Boosting Regressor*

Results

Below are the testing results of the three models that show the performance of the models to generalise to unseen data.

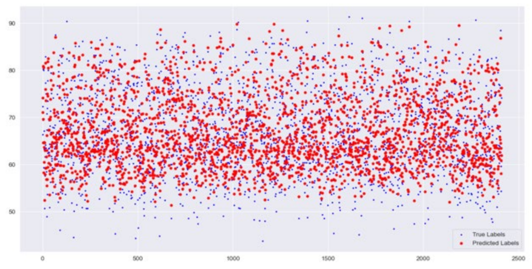


Figure 7: Baselline Model Results

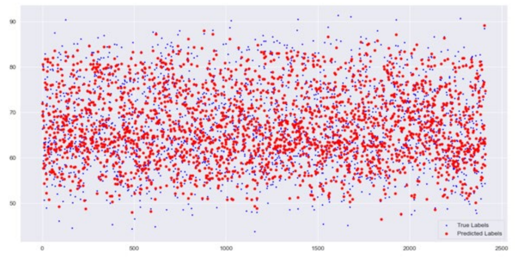


Figure 8: Tuned Random Forest Regressor Model Results (max_depth = 20)

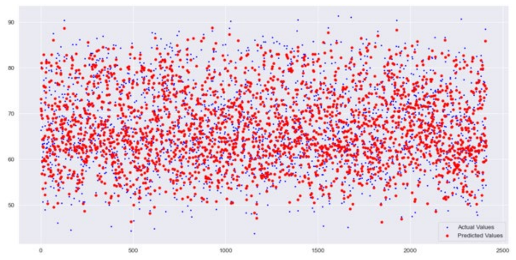
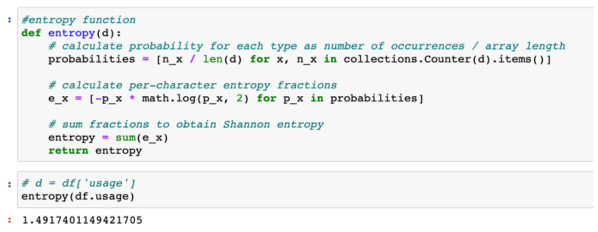


Figure 9: Tuned Gradient Boosting Regressor Model Results (max_depth=12)

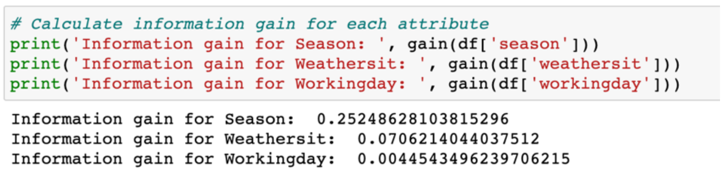
From the figures above (Blue – Ground Truth, Red – Predicted Values), we notice that the baseline model mispredict a lot of nitrous oxide values in the bottom. Other tuned models appear to perform better and predict correctly some of these values in the bottom. Overall, tuned random forest regressor performs the best out of all of the three models.

- Baseline Model: MSE (18.11)
- Tuned Random Forest Model: MSE (10.60)
- Tuned Gradient Descent model: MSE (10.61)



```
: #entropy function
  def entropy(d):
      # calculate probability for each type as number of occurrences / array length
      probabilities = [n_x / len(d) for x, n_x in collections.Counter(d).items()]

      # calculate per-character entropy fractions
      e_x = [-p_x * math.log(p_x, 2) for p_x in probabilities]

      # sum fractions to obtain Shannon entropy
      entropy = sum(e_x)
      return entropy

: # d = df['usage']
  entropy(df.usage)

: 1.4917401149421705
```

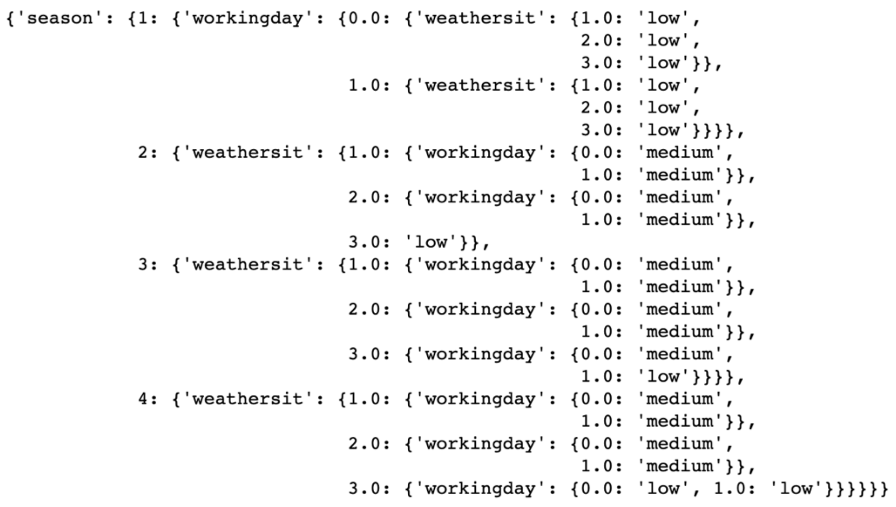
Entropy function



```
# Calculate information gain for each attribute
print('Information gain for Season: ', gain(df['season']))
print('Information gain for Weathersit: ', gain(df['weathersit']))
print('Information gain for Workingday: ', gain(df['workingday']))

Information gain for Season:  0.25248628103815296
Information gain for Weathersit:  0.0706214044037512
Information gain for Workingday:  0.0044543496239706215
```

Information Gain values for the three attributes ("Season", "Workingday", and "Weathersit")

Decision Tree Algorithm



```
{'season': {1: {'workingday': {0.0: {'weathersit': {1.0: 'low',
                                                    2.0: 'low',
                                                    3.0: 'low'}},
                               1.0: {'weathersit': {1.0: 'low',
                                                    2.0: 'low',
                                                    3.0: 'low'}}}},
            2: {'weathersit': {1.0: {'workingday': {0.0: 'medium',
                                                    1.0: 'medium'}},
                               2.0: {'workingday': {0.0: 'medium',
                                                    1.0: 'medium'}},
                               3.0: 'low'}},
            3: {'weathersit': {1.0: {'workingday': {0.0: 'medium',
                                                    1.0: 'medium'}},
                               2.0: {'workingday': {0.0: 'medium',
                                                    1.0: 'medium'}},
                               3.0: {'workingday': {0.0: 'medium',
                                                    1.0: 'low'}}}},
            4: {'weathersit': {1.0: {'workingday': {0.0: 'medium',
                                                    1.0: 'medium'}},
                               2.0: {'workingday': {0.0: 'medium',
                                                    1.0: 'medium'}},
                               3.0: {'workingday': {0.0: 'low', 1.0: 'low'}}}}}}
```

A Decision Tree built using the three attributes ("Season", "Workingday", and "Weathersit")

Overall, the decision tree structure has some redundant nodes. This can be improved by introducing a pruning technique to the tree. Pruning will eliminate unnecessary splits in the tree, keeping only the nodes that generalise better. Hence, reducing the tree's complexity and improving its accuracy (avoids miss-classification).

Adding a new attribute (tempbins) to the Decision Tree

This "tempbins" is calculated by binning the attribute temp into three values, i.e., high, medium and low. I've noticed that the attribute has a higher IG than other features in the dataset (0.2888); hence, it will alter the structure of the tree and generalises better than other features. Introducing a pruning technique can produce a better tree structure with higher classification accuracy.

I. Motivation

1. For what purpose was the dataset created?

The WikiArt Emotions dataset was created to study art and its impact on human's perception hence there is a need to improve the search for art with different features primarily the emotions they evoked on the observer.

2. Who created the dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organisation)?

The dataset was created by Dr Saif M. Mohammad and Dr Svetlana Kiritchenko on behalf of the National Research Council (NRC) in Canada. Their research interests primarily in Computational Linguistics and Natural Language Processing (NLP).

3. Who funded the creation of the dataset?

The creation of the WikiArt Emotions dataset was conducted at the National Research Council (NRC) which is an organisation funded by the Canadian government.

II. Composition

1. What do the instances that compromise the dataset represent (e.g., documents, photos, people, countries)?

The WikiArt Emotions dataset consists of various pieces of art mostly paintings along with their crowdsourced annotations for emotions evoked in the observer. The actual pieces of art are selected from WikiArt.org's collection for four western styles which are Renaissance Art, Post-Renaissance Art, Modern Art, and Contemporary Art.

2. How many instances are there in total (of each type, if appropriate)?

There were 308 people that annotated between 20 and 1,525 pieces of art, where a total of 41,985 sets of responses were obtained.
There are 151,151 pieces of art in total available on the WikiArt.org website. These pieces of art are corresponding to ten main art styles and 168 style categories. The researchers have been able to annotate 4,105 pieces of art out of twenty-two categories on the WikiArt.org website.
Below is the list of the 22 categories with the number of art pieces annotated from each category.

- **Contemporary Art**

Minimalism	200
Modern Art	200
Abstract Art	200
Abstract Expressionism	200
Art Informel	20