Executive summary

This report focuses on the Glass Identification dataset, which involves classifying glass samples based on their chemical composition. The objective is to predict the type of glass, such as building windows, vehicle windows, and containers, using measurements of various chemical elements.

The solutions presented in this report address the Glass Identification as classification problem through a comprehensive process involving data retrieval and preparation, exploratory data analysis, and data modeling using Jupyter Notebook.

The findings from this research suggest that with two classification model, KNeighbors and Decision Tree, combination with Simple Hill Climping Algorithms have similar prediction accuracy which are and . The findings suggest that the two classification model without selecting proper features will return to low accuracy. Meanwhile, applying Simple Hill Climping influenced to high accuracy in classifying glass types.

Introduction

The Glass Identification dataset presents a classification problem that involves predicting the type of glass based on its chemical composition. This dataset, sourced from the UCI Machine Learning Repository, contains measurements of various chemical elements in glass samples. These measurements serve as features to train machine learning models to classify glass into categories such as building windows, vehicle windows, and containers. The accurate classification of glass types is crucial in science, manufacturing, and quality control processes. This report aims to explore the Glass Identification dataset, prepare the data for analysis, and develop models to achieve high classification accuracy using supervised learning techniques.

Methodology

The methodology for this project consists of several key steps, each critical to developing an accurate and reliable classification model. The process includes data retrieval, data preprocessing, exploratory data analysis, feature selection, model training, evaluation, and tuning.

1.Setting the research goal

The Glass Identification dataset focuses on classifying glass based on its chemical component. The objective is to identify and predict chemical component of the new sample type of glass (such as vehicle windows) based on measurement as the weight percent in corresponding oxide in the glass.

The research goal is to develop a machine learning model to accurate classify type of glass based on its chemical composition. The model can be apply to use in future scientific research and industrial.

2.Data Retrieval:

The glass identification is a dataset from UCI Machine Learning Repository. This dataset has 214 rows and 10 columns, which contains 10 attributes including Id and 7 of glass sample types. There is no missing value in this dataset.

Preparation Step: Access dataset and convert to .csv file

1. Download Glass Identification from UCI Machine Learning.
2. Open Microsoft Excel and import glass.data file as csv file.

3. Convert glass.data file to .csv by selecting delimiters using tab and comma.
4. Export to glass.csv file and save the file into JupyterNotebook workspace directory.

Step 1: Initialise JupyterNotebook environment and import necessary libraries.

```
In [1]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.base import TransformerMixin
```

Step 2: Import dataset to JupyterNotebook

```
df.head(15)
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.00	1
2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.00	1
3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.00	1
4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.00	1
5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.00	1
6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.0	0.26	1
7	1.51743	13.30	3.60	1.14	73.09	0.58	8.17	0.0	0.00	1
8	1.51756	13.15	3.61	1.05	73.24	0.57	8.24	0.0	0.00	1
9	1.51918	14.04	3.58	1.37	72.08	0.56	8.30	0.0	0.00	1
10	1.51755	13.00	3.60	1.36	72.99	0.57	8.40	0.0	0.11	1
11	1.51571	12.72	3.46	1.56	73.20	0.67	8.09	0.0	0.24	1
12	1.51763	12.80	3.66	1.27	73.01	0.60	8.56	0.0	0.00	1
13	1.51589	12.88	3.43	1.40	73.28	0.69	8.05	0.0	0.24	1
14	1.51748	12.86	3.56	1.27	73.21	0.54	8.38	0.0	0.17	1
15	1.51763	12.61	3.59	1.31	73.29	0.58	8.50	0.0	0.00	1

Step 3: Check data type and value counts

```
df.dtypes
```

```
RI                float64
Na                float64
Mg                float64
Al                float64
Si                float64
K                 float64
Ca                float64
Ba                float64
Fe                float64
Type_of_glass       int64
dtype: object
```

```
df['Type_of_glass'].value_counts()
```

```
2    76
1    70
7    29
3    17
5    13
6     9
Name: Type_of_glass, dtype: int64
```

3.Data Preparation

Remove duplications

One duplication was removed by df.drop_duplicates().

Identify missing value

There is no missing value in Glass Identification dataset as it described in dataset information.

Find and remove Outliers

There are outliers in the Glass Identification dataset. Since this dataset measures chemical components as weight percent in corresponding oxides, these chemical components are sensitive data. Additionally, most of the outliers have values of 0.0, which are meaningful in terms of chemical composition. In this report, outliers will not be removed but will be indicated and represented if they appear in the visualizations.

4.Data Exploration

Task 2.1: Explore each column

Display basic descriptive staticstics for all columns by using df.describe()

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000
mean	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.175047	0.057009	2.780374
std	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219	0.097439	2.103739
min	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000	1.000000
25%	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000	0.000000	1.000000
50%	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000	0.000000	2.000000
75%	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000	0.100000	3.000000
max	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000	7.000000

Figure 1 Basic descriptive staticstics

Based on basic descriptive staticstics we can explore each columns of Glass Identification dataset by appropriate descriptive statistics and graphs as following

RI (Refractive Index)	
Measures of central tendency	Measures of variability
Mean: 1.518365 Mode: Q1(25%): Q2(50%): Q3(75%):	Standard deviation: 0.003037 Minimum value: 1.511150 Maximum value: 1.533930

Data Exploration: Based on Mean and Range of RI, the number of ranges are very narrow and low (around 1.51-1.53). This descriptive indicates RI values clustered around Mean. Which means differences glasses type have similar Refractive Index in between 1.511150 to 1.533930.

Na (Sodium)	
Measures of central tendency	Measures of variability
Mean: 13.407850 Mode: Q1(25%): 12.907500 Q2(50%): 13.300000 Q3(75%): 13.825000	Standard deviation: 0.816604 Minimum value: 10.73 Maximum value: 17.38

Data Exploration: Based on Mean and Range of Na (Sodium), Sodium has high variability and ranges. The number of mean also indicates sodium has a significant weight percent in corresponding oxide in differences type of glass.

Mg (Magnesium)	
Measures of central tendency	Measures of variability
Mean: 2.684533 Mode: Q1(25%): 2.115000 Q2(50%): 3.480000 Q3(75%): 3.600000	Standard deviation: 1.442408 Minimum value: 0.00 Maximum value: 4.49

Data Exploration: Based on Mean and Range of Mg (Magnesium), The numbers show that some types of glass do not contains Magnesium. Mean value of Magnesium indicates glasses contain little of Magnesium.

Al (Aluminum)	
Measures of central tendency	Measures of variability
Mean: 1.444907 Mode: Q1(25%): 1.190000 Q2(50%): 1.360000 Q3(75%): 1.630000	Standard deviation: 0.499270 Minimum value: 0.29 Maximum value: 3.50

Data Exploration: Based on Mean and Range of Al (Aluminum), Aluminum has little variability and ranges. The number of mean also indicates Aluminum has a less significant weight percent in corresponding oxide in differences type of glass but all types of glass contain Aluminum.

Si(Silicon)	
Measures of central tendency	Measures of variability
Mean: 72.650935 Mode: Q1(25%): 72.280000 Q2(50%): 72.790000 Q3(75%): 73.087500	Standard deviation: 0.774546 Minimum value: 69.81 Maximum value: 75.41

Data Exploration: Based on Mean and Range of SI(Silicon), Silicon has huge variability and ranges. The number of Mean indicates that Silicon has most weight percent in corresponding oxide in all type of glasses.

K(Potassium)	
Measures of central tendency	Measures of variability
Mean: 0.497056 Mode: Q1(25%): 0.122500 Q2(50%): 0.555000 Q3(75%): 0.610000	Standard deviation: 0.652192 Minimum value: 0.00 Maximum value: 6.21

Data Exploration: Based on measures of K(Potassium). The number of minimum and maximum show that some types of glass do not contain Potassium. According to Mean 0.497056 and Q2 0.555000, Majority of glasses contain few Potassium weight percent in corresponding oxide.

Ca(Calcium)	
Measures of central tendency	Measures of variability
Mean: 8.956963 Mode: Q1(25%): 8.240000 Q2(50%): 8.600000 Q3(75%): 9.172500	Standard deviation: 1.423153 Minimum value: 5.43 Maximum value: 16.19

Data Exploration: Based on Mean and Range of Ca (Calcium), The measures show that Calcium has huge variability and ranges between 5.43-16.19, It is also indicating that all of types of glass contain Calcium as it is important component by Mean 8.956963

Ba(Barium)	
Measures of central tendency	Measures of variability
Mean: 0.175047 Mode: Q1(25%): 0.000000 Q2(50%): 0.000000 Q3(75%): 0.000000	Standard deviation: 0.497219 Minimum value: 0.00 Maximum value: 3.15

Data Exploration: Measures of Ba (Barium) show some types of glass do not contain Barium, Majority of glasses only contain few Barium (Mean 0.175047)

Fe(Iron)	
Measures of central tendency	Measures of variability
Mean: 0.057009 Mode: Q1(25%): 0.000000 Q2(50%): 0.000000 Q3(75%): 0.000000	Standard deviation: 0.097439 Minimum value: 0.00 Maximum value: 0.51

Data Exploration: Measures of Fe (Iron) indicate glasses contain very small number of weight percent in corresponding oxide of Iron and some glasses do not contain Iron

Using Seaborn histplot to show distributions of each oxides. And enable kde to show density of the distributions.

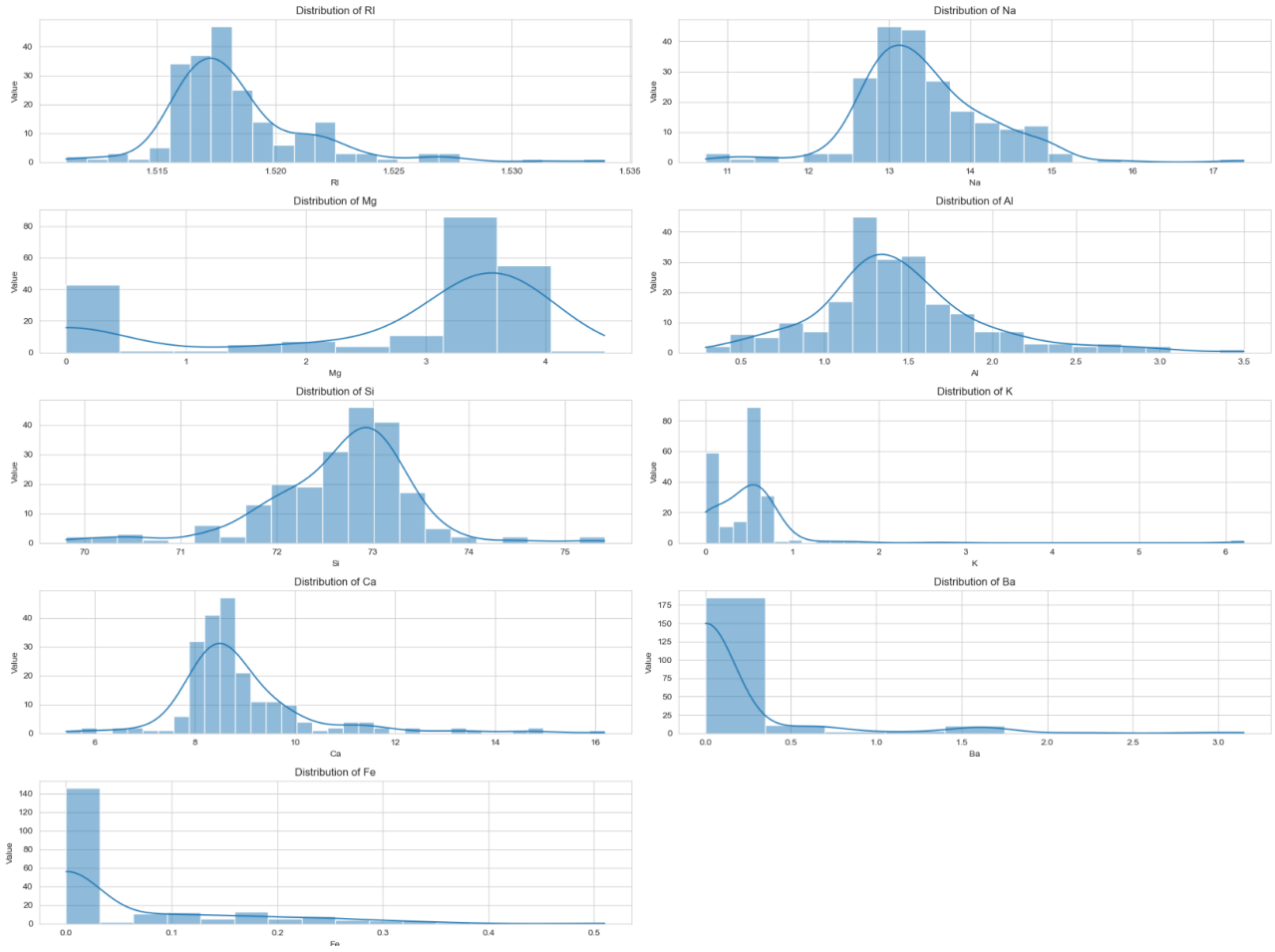


Figure 2 The histplot represents the distributions of each oxides and show density.

The histogram shows that Distribution of RI, Na, Al and Ca seem to be Poisson distribution patterns. Meanwhile, other attributes like K, Mg, Fe having very clear outlies because the characteristic of chemical components influenced to some types of glass do not contain these chemical components.

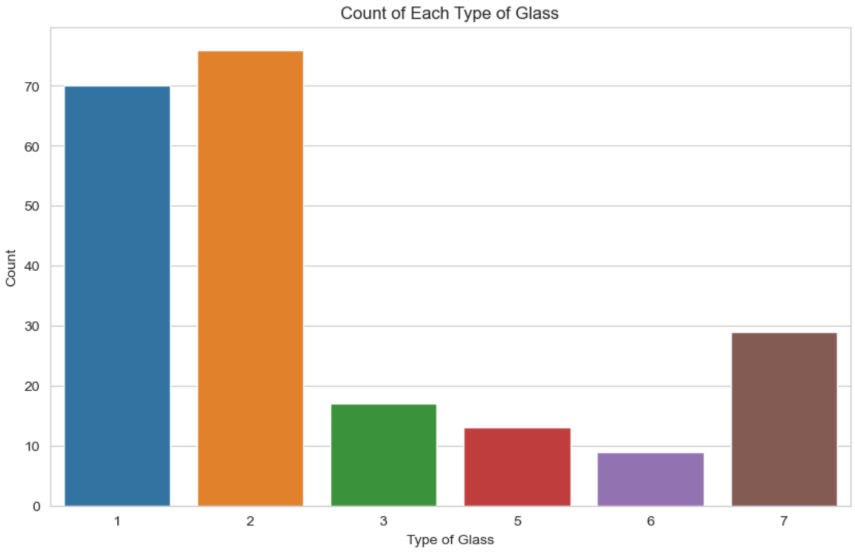


Figure 3 The graph represents the number of samples in each type of glass.

The graph represents the number of samples in each type of glass. Based on the UCI dataset, each number in the graph corresponds to a specific type of glass: 1: Building windows, float processed, 2: Building windows, non-float processed, 3: Vehicle windows, float processed, 4: Vehicle windows, non-float processed (none in this database), 5: Containers, 6: Tableware, 7: Headlamps

The graph shows that the majority of glass samples are of type 2 (76 samples), followed by type 1 (70 samples), type 7 (29 samples), type 3 (17 samples), type 5 (13 samples), and type 6 (9 samples). There are no samples of type 4.

Task 2.2: Explore the relationship between pairs of attributes

Exploration of the relationship between pairs of attributes

This report choose 10 pairs of attribute by first 4 pairs will focusing on RI (Refractive Index) by selecting RI pair with large number of weight percent in corresponding oxide, Si and Na, and pair with small number of weight percent in corresponding oxide, Al and Mg.

The 6 pairs will comparing between large number and small of weight percent in corresponding oxide.

Hypothesis about RI (Refractive Index) pair with Si, Na, Al, Mg : There will be significant relationship between these pairs of attribute. To indicates that glass chemical components affected to refractive index.

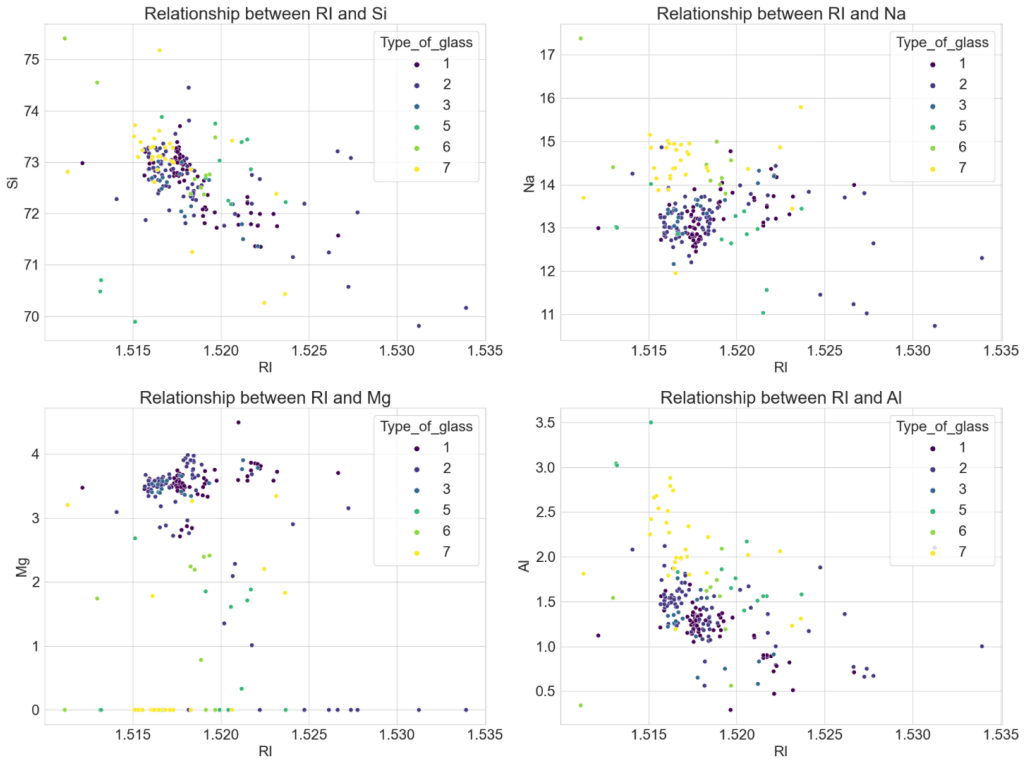


Figure 4 The scatter plots show Hypothesis about RI (Refractive Index) pair with Si, Na, Al, Mg

1. RI and Si:

In the scatter plots, the relation between RI and Si shows that there is low/weak negative Correlation. In term of observation, RI and Si are congregate nearby their mean(Si mean : 72.650935, RI mean: 1.518365). This relationship is visible and clear that Si influences RI.

2. RI and Na:

In the scatter plots, the relation between RI and Na shows that there is low/weak positive correlation. In this Scatter plot also shows value of the glass type number 7 (headlamps) trends to have more Na and less RI more than other samples. This relationship is noticeable, but it tends to be more dispersed.

3. RI and Mg:

In the scatter plots, there are amout of samples that do not have Mg(value = 0.0).

The relationship between RI and Mg seems to be unrelated to each other.

4. RI and Al:

In the scatter plots show that most of glass types have low Al, there are few outliers have high Al content.

The relationship between RI and Al are visible but weak positive correlation.

In conclusion of the hypothesis about RI :

All of the scatter plots show lack of strong linear relationship but the scatter plots show clear clusters indicates RI relationship with other attributes. The specification of difference types of glass also represent in these scatter plots. The result of this exploration data indicates chemical components and difference types of glass influence difference value of RI.

Hypothesis the relationships between pairs of attributes such as Si, Na, Ca, Mg, and Al

There will be significant patterns that help in understanding the chemical composition of different types of glass.

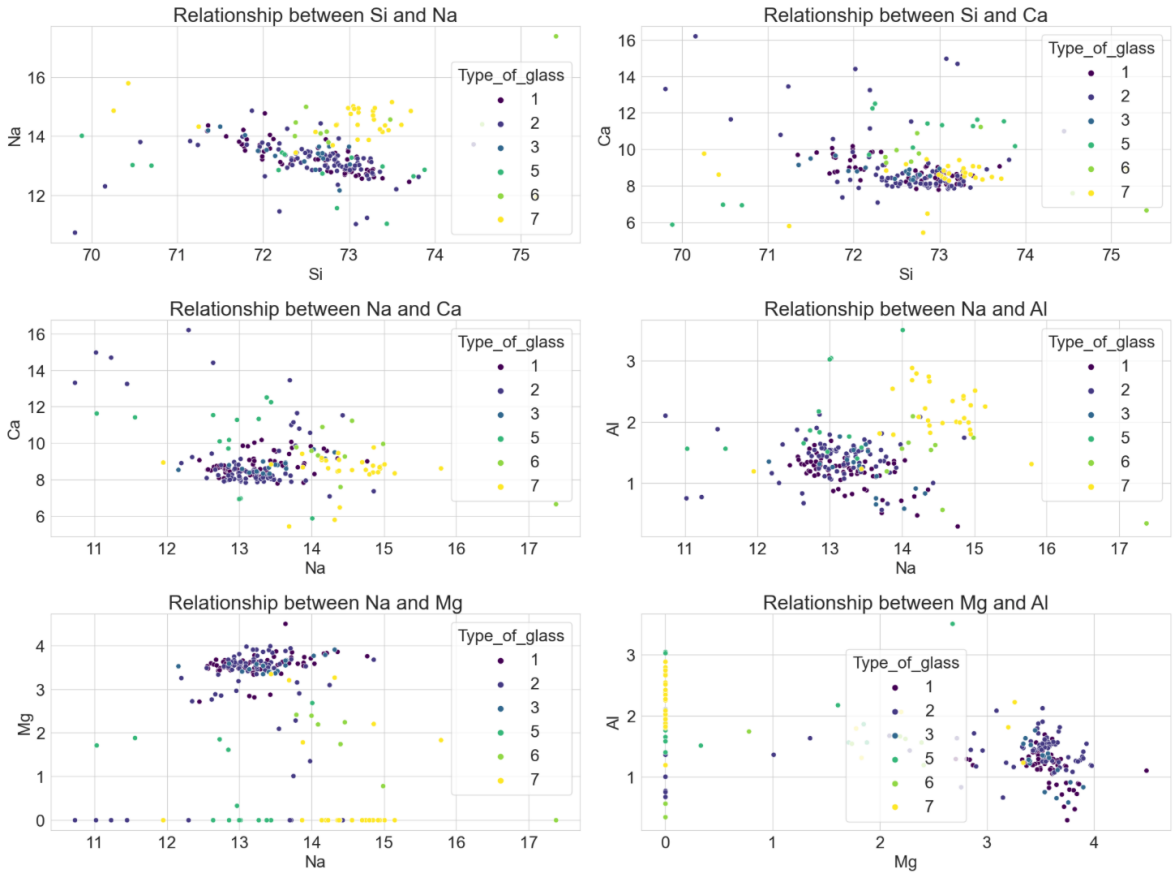


Figure 5 the scatter plots show Hypothesis the relationships between pairs of attributes such as Si, Na, Ca, Mg, and Al

1. Si and Na

The scatter plot shows relationship between Si and Na related to glass type, Different types of glass exhibit distinct distributions. There is a weak positive correlation between Si and Na. The relationship between Si and Na are visible but weak positive correlation.

2. Si and Ca

The scatter plot shows relationship between Si, Ca and glass type are related to each other, there is a cluster around their mean value. There are few outliers shown. The relationship between Si and Ca are visible but weak positive correlation.

3. Na and Ca

The scatter plot shows relationship between Na, Ca and glass type are clear. The glass type number 7 (headlamps) and other glass type are clearly separated. So Na and Ca influenced by type of glass. The relationship between Na and Ca are visible but unclear linear and correlation.

4. Na and Al

The scatter plot shows relation between Na and Al. Most of glass types have a low to moderate Al content with varying Na values. Different types of glass have distinct distributions. The relationship is noticeable, with distinct patterns for different glass types.

5. Na and Mg

The scatter plot shows relationship between Na, Mg and glass type are low/weak linear. There are outliers caused by some samples have Mg value=0.0. The relationship between Na and Mg are visible as cluster and unclear linear.

6. Mg and Al

The scatter plot shows relationship between Na, Mg and glass type are low/weak or unclear. There are outliers caused by some samples have Al value=0.0. The most of samples contain Mg value more than Al. The relationship between Mg and Al are visible as cluster and unclear linear.

In conclusion of the hypothesis about pair of attributes :

All of the scatter plots show cluster and lack of strong linear relationship. Some attributes are clearly related with each other and influenced by types of glass(Si-Na, Si-Ca, Na-Ca, Na-Al). Meanwhile, some pairs are unclear and might not related (Na-Mg, Mg-Al). The result of this exploration data can be indicate that difference types of glass contain some significant related chemical components.

5.Data Modelling

The modelling phase consists of four steps:

1. Feature engineering and model selection

2. Training the model

3. Model validation and selection

1. Feature engineering and model selection

Feature Selection: For glass Identification dataset, all features have been assigned as each chemical component. Each feature component contributes valuable information for classification. Type of glass will be target variable. In Model training section, this report will use Simple Hill Climping to select features for improving accuracy of the models.

The Label: The Label or Target of Glass Identification dataset is type of glass, which is a categorical variable.

Model Selection: This report uses two classification model since Glass Identification dataset is classification problem. The two model are including:

1. K-Nearest Neighbors, one of the simple and powerful algorithms.
2. Decision Tree, one of the classification model can work well with visualization and easy to represent comparing to the hypothesis.

2. Training the model : K-Nearest Neighbors

Model Training: K-Nearest Neighbors, Criteria: Distance

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

#X is the feature of the input data
X = df.drop('Type_of_glass', axis=1)

#y is the target value of the input data
y = df['Type_of_glass']

# This train test split select test_size 0.2, random state 42(based on Hitchhiker's Guide to the Galaxy).
# Define test_size = 0.2 means 20% of data will be assigned to be test.
# 80% of data will be assigned to be traning.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Selecting X_train and y_train by define X is the features of the dataset(e.g. Na, Si, ...), y is the target value (types_of_glass). The train test split assigns parameter: test_size=0.2 : 20% of data will be assigned to be test and 80% of data will be assigned to be training. Random_state=42 : random_state assigned based on Hitchhiker's Guide to the Galaxy. Random_state will be assigned the same value as 42 to this report.

K-Nearest Neighbors model: Findind the best accuracy n_neighbors

```
# Initialize the KNN model
clf = KNeighborsClassifier(3, weights='distance', p=1)

# Train the model
fit = clf.fit(X_train, y_train)

# Predictions
y_pre = fit.predict(X_test)
y_pre.shape
```

Clf : classifier variable with Kneighbors model and assign n_neighbors=5.

Fit: the method to training the model, fit trains the classifier clf on the training data X_train and the labels y_train.

y_pre: variable to define value that predicted.

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pre)
print(classification_report(y_test,y_pre))

print(f'Accuracy: {accuracy}')
```

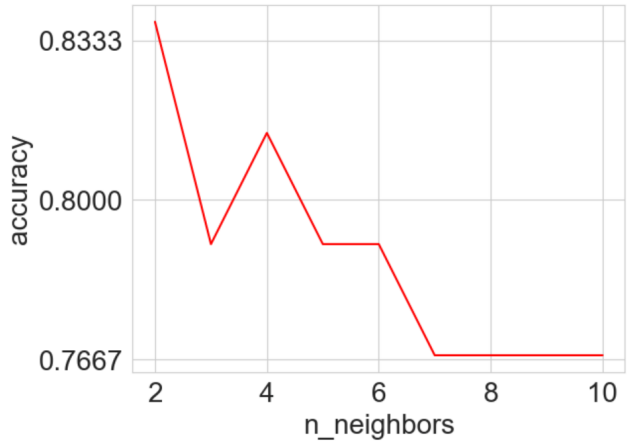
```
              precision    recall  f1-score   support

           1       0.57      0.73      0.64        11
           2       0.82      0.64      0.72        14
           3       0.50      0.67      0.57         3
           5       1.00      1.00      1.00         4
           6       0.67      0.67      0.67         3
           7       1.00      0.88      0.93         8

    accuracy                           0.74        43
   macro avg       0.76      0.76      0.76        43
weighted avg       0.77      0.74      0.75        43

Accuracy: 0.7441860465116279
```

The result of assign n_neighbors=3:

Models are evaluated based on accuracy, precision, recall, and F1-score.

The accuracy percentage shows that equal to 0.744186. So this report decided to finding the best n_neighbors by following code:

```
normal_accuracy = []  # Create an empty list of accuracy rates
k_value = range(2, 11)
for k in k_value:
    clf = KNeighborsClassifier(k, weights='distance',p=1)
    fit = clf.fit(X_train, y_train)
    y_pre = fit.predict(X_test)
    accuracy = accuracy_score(y_test, y_pre)
    normal_accuracy.append(accuracy)

plt.xlabel("n_neighbors")
plt.ylabel("accuracy")
new_ticks = np.linspace(0.6, 0.9, 10)
plt.yticks(new_ticks)
plt.plot(k_value, normal_accuracy, c='r')
plt.grid(True)
```



To accuracy the KNeighbors Model, we use the k_value for looping and identify which one is best for accuracy score.

As the graph relationship between n_neighbor and accuracy score reveals the best number of n_neighbors for training accuracy model is 2.

K-Nearest Neighbors model: After apply to Simple Hill Climbing

```
# Simple Hill Climbing Algorithm
for cur_f in col_Ind_Random:
    new_Ind.append(cur_f)
    newData = X[:, new_Ind]
    print("Shape of new data:", newData.shape)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(newData, y, test_size=0.2, random_state=42)

    # Train the KNN model
    clf = KNeighborsClassifier(n_neighbors=2, weights='distance', p=1)
    clf.fit(X_train, y_train)

    # Score the model
    cur_Score = clf.score(X_test, y_test)

    # Check the scores of features
    if cur_Score < cur_MaxScore:
        print(f"Score with {len(new_Ind)} selected features: {cur_Score}")
        new_Ind.remove(cur_f)
    else:
        cur_MaxScore = cur_Score
        print(f"Score with {len(new_Ind)} selected features: {cur_Score}")


selected_features = [df.columns[i] for i in new_Ind]
print("Selected feature indices:", selected_features)
```

```
Randomized feature indices: [8, 2, 6, 7, 1, 0, 4, 3, 5]
Shape of new data: (214, 1)
Score with 1 selected features: 0.3488372093023256
Shape of new data: (214, 2)
Score with 2 selected features: 0.32558139534883723
Shape of new data: (214, 2)
Score with 2 selected features: 0.3953488372093023
Shape of new data: (214, 3)
Score with 3 selected features: 0.5581395348837209
Shape of new data: (214, 4)
Score with 4 selected features: 0.6046511627906976
Shape of new data: (214, 5)
Score with 5 selected features: 0.6046511627906976
Shape of new data: (214, 6)
Score with 6 selected features: 0.6511627906976745
Shape of new data: (214, 7)
Score with 7 selected features: 0.7906976744186046
Shape of new data: (214, 8)
Score with 8 selected features: 0.8372093023255814
Selected feature indices: ['Fe', 'Ca', 'Ba', 'Na', 'RI', 'Si', 'Al', 'K']
```

Simple Hill Climbing is used to selecting the features which will influenced the best accuracy for the model. In the figure shows Best features are 'Fe', 'Ca', 'Ba', 'Na', 'RI', 'Si', 'Al', 'K' with 0.8372093023255814 accuracy score. The accuracy score seems satisfied abrove to 83 percentages.

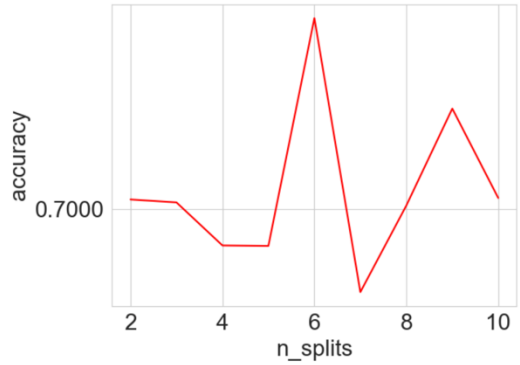
```
print("Evaluation : ")

# Evaluate the final model with the selected features
final_X = X[:, new_Ind]
X_train, X_test, y_train, y_test = train_test_split(final_X, y, test_size=0.2, random_state=42)
final_model = KNeighborsClassifier(n_neighbors=2, weights='distance', p=1)
final_model.fit(X_train, y_train)
final_y_pred = final_model.predict(X_test)

# Evaluate the final model
final_accuracy = accuracy_score(y_test, final_y_pred)
print(f"Final Model Accuracy with Selected Features: {final_accuracy}")
```

```
Evaluation :
Final Model Accuracy with Selected Features: 0.8372093023255814
```

The figure abrove is to show the result of the model that selected best features. The accuracy score of K-Nearest Neighbor is 0.8372093023255814 after applying to Simple Hill Climbing

K-Nearest Neighbors model: Apply Evaluation by K-fold

```
Cross Validation Scores:  [0.90909091 0.77272727 0.59090909 0.5        0.76190476 0.76190476
 0.71428571 0.66666667 0.76190476 0.57142857]
Average CV Score:  0.701082251082251
Number of CV Scores used in Average:  10
```



Applying the model to K-fold cross-validation to enable evaluation of model performance by applying the entire dataset. To identify imbalanced data and make modeling more reliable.

The use of loop to select the best n_splits parts for K-fold reveals that the best n_splits parts should be 6 subsets.

```
from sklearn.model_selection import KFold, cross_val_score

# Perform k-fold cross-validation
k_folds = KFold(n_splits=6, shuffle=True, random_state=42)
cv_scores = cross_val_score(final_model, final_X, y, cv=k_folds, scoring='accuracy')

# Print cross-validation results
print("Cross Validation Scores: ", cv_scores)
print("Average CV Score: ", cv_scores.mean())
print("Number of CV Scores used in Average: ", len(cv_scores))
```

```
Cross Validation Scores:  [0.86111111 0.69444444 0.63888889 0.75       0.71428571 0.65714286]
Average CV Score:  0.7193121693121695
Number of CV Scores used in Average:  6
```

The K-Fold validation after applying Simple Hill Climbing and selecting the best n_splits will have accuracy average at 0.7193121693121695. The K-Fold validation enable evalution of model performance and ability to test entire dataset. Therefore the accuracy after applying to K-Fold will be decreased but satisfied and reliable.

Training Decision Tree model:

Edit Attachments

```
# Preprocess the data
X = df.drop('Type_of_glass', axis=1)
y = df['Type_of_glass']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

clf = DecisionTreeClassifier(random_state=0, criterion="gini", max_depth=10,
                             min_samples_split=15, min_samples_leaf=5)
clf = clf.fit(X_train, y_train)
y_pre = clf.predict(X_test)
y_pre.shape
```

Decision Tree Model will use the same environment variable, train_test_split and random_state 42. And defined criteria:gini. Max_depth=10, min_samples_split 15 to prevent the overfiting.

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pre))
accuracy = accuracy_score(y_test, y_pre)
print("accuracy score : ",accuracy)
```

```
              precision    recall  f1-score   support

           1       0.71      0.91      0.80        11
           2       0.64      0.50      0.56        14
           3       0.60      1.00      0.75         3
           5       0.50      0.25      0.33         4
           6       1.00      0.67      0.80         3
           7       0.89      1.00      0.94         8

    accuracy                           0.72        43
   macro avg       0.72      0.72      0.70        43
weighted avg       0.71      0.72      0.70        43

accuracy score :  0.7209302325581395
```

The accuracy score is 0.7209302325581395 before applying to Simple Hill Climbing

Decision Tree model: After apply to Simple Hill Climbing

```
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.utils import shuffle

X = df.drop('Type_of_glass', axis=1).values
y = df['Type_of_glass'].values

# Shuffle features
col_num = X.shape[1]
col_Ind_Random = shuffle(list(range(col_num)), random_state=1)
print("Randomized feature indices:", col_Ind_Random)

# setup array
new_Ind = []
cur_MaxScore = 0.0  # Hold the value of the accuracy

# Simple Hill Climbing Algorithm
for cur_f in col_Ind_Random:
    new_Ind.append(cur_f)
    newData = X[:, new_Ind]
    print("Shape of new data:", newData.shape)

    X_train, X_test, y_train, y_test = train_test_split(newData, y, test_size=0.2,
                                                        random_state=42)
    model = DecisionTreeClassifier(random_state=0, criterion="gini",
                                   max_depth=10, min_samples_split=15,
                                   min_samples_leaf=5)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    cur_Score = accuracy_score(y_test, y_pred)

    if cur_Score < cur_MaxScore:
        print(f"Score with {len(new_Ind)} selected features: {cur_Score}")
        new_Ind.remove(cur_f)
    else:
        cur_MaxScore = cur_Score
        print(f"Score with {len(new_Ind)} selected features: {cur_Score}")


selected_features = [df.columns[i] for i in new_Ind]
print("Selected feature indices:", selected_features)
```

```
Randomized feature indices: [8, 2, 6, 7, 1, 0, 4, 3, 5]
Shape of new data: (214, 1)
Score with 1 selected features: 0.20930232558139536
Shape of new data: (214, 2)
Score with 2 selected features: 0.5348837209302325
Shape of new data: (214, 3)
Score with 3 selected features: 0.627906976744186
Shape of new data: (214, 4)
Score with 4 selected features: 0.6976744186046512
Shape of new data: (214, 5)
Score with 5 selected features: 0.7441860465116279
Shape of new data: (214, 6)
Score with 6 selected features: 0.6744186046511628
Shape of new data: (214, 6)
Score with 6 selected features: 0.7441860465116279
Shape of new data: (214, 7)
Score with 7 selected features: 0.7209302325581395
Shape of new data: (214, 7)
Score with 7 selected features: 0.6976744186046512
Selected feature indices: ['Fe', 'Mg', 'Ca', 'Ba', 'Na', 'Si']
```

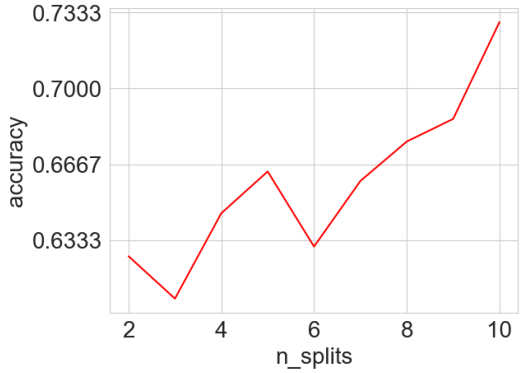
Simple Hill Climbing is used to selecting the features which will influenced the best accuracy for the model.

In the figure shows Best features are 'Fe', 'Mg', 'Ca', 'Ba', 'Na', 'Si'.

The best features are significant differences selected from KNeighbor model.

The Decision Tree model with Simple Hill Climbing provided 0.7441860465116279 accuracy score. The accuracy score seems satisfied abrove to 74 percentages.

Decision Tree model: Apply Evaluation by K-fold



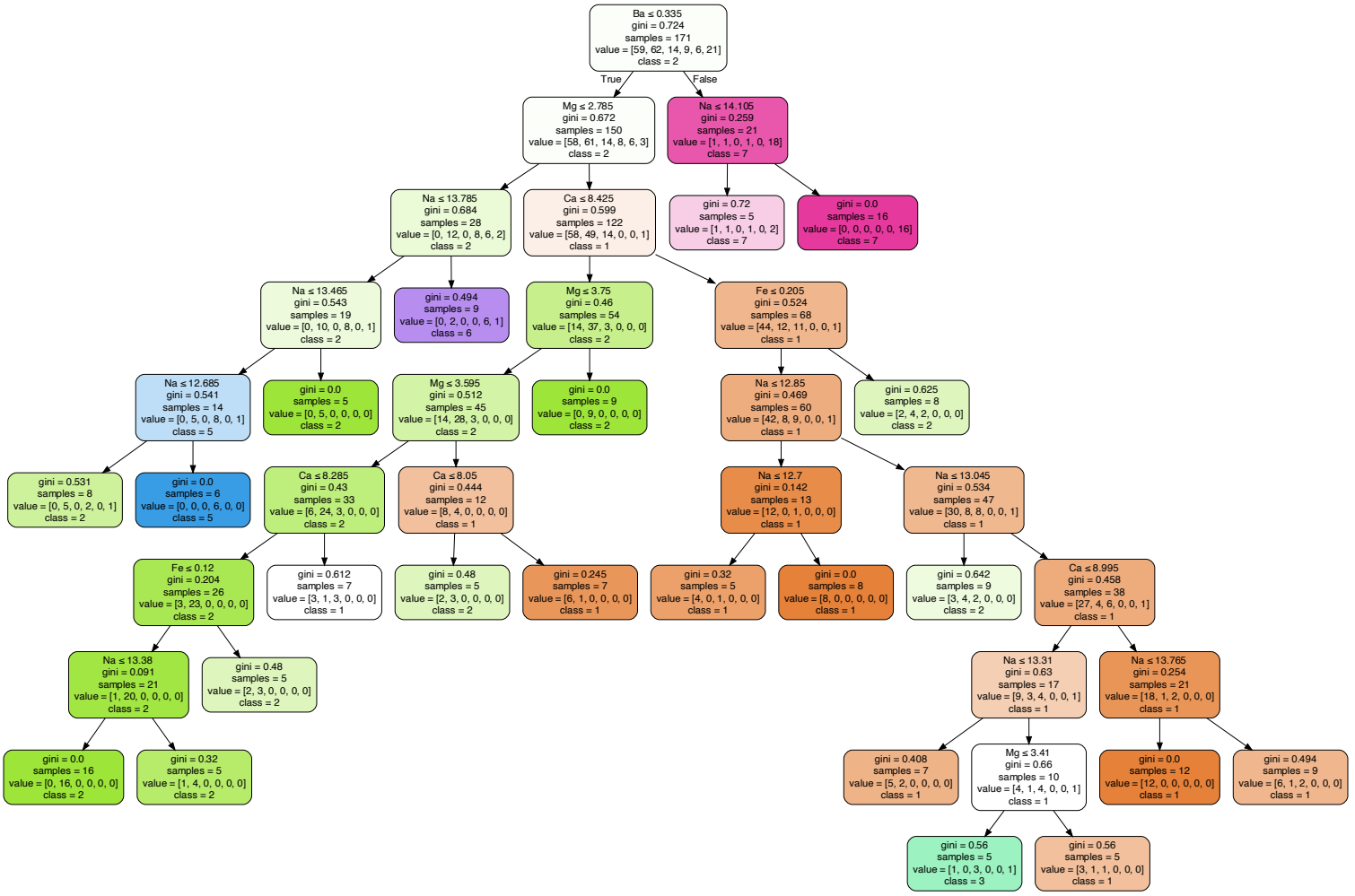
Applying the model to K-fold cross-validation to enable evaluation of model performance by applying the entire dataset. To identify imbalanced data and make modeling more reliable.

The use of loop to select the best n_splits parts for K-fold reveals that the best n_splits parts should be 10 subsets.

```
# Perform k-fold cross-validation
k_folds = KFold(n_splits=10, shuffle=True, random_state=42)
cv_scores = cross_val_score(final_model, final_X, y, cv=k_folds, scoring='accuracy')

# Print cross-validation results
print("Cross Validation Scores: ", cv_scores)
print("Average CV Score: ", cv_scores.mean())
print("Number of CV Scores used in Average: ", len(cv_scores))
```

```
Cross Validation Scores:  [0.86363636 0.72727273 0.72727273 0.54545455 0.80952381 0.76
190476
 0.80952381 0.76190476 0.66666667 0.61904762]
Average CV Score:  0.7292207792207792
Number of CV Scores used in Average:  10
```

The K-Fold validation after applying Simple Hill Climbing and selecting the best n_splits will have accuracy average at 0.7292207792207792. The K-Fold validation enable evalution of model performance and ability to test entire dataset. Therefore the accuracy after applying to K-Fold will be decreased but satisfied and reliable.



The Decision Tree Visualization: The Decision tree visualisation shows how the model classify each types of glass based on its chemical components (features). The accuracy of decision tree model seems reliable and high accuracy with 0.729. But the visualization shows that there almost be overfiting in decision tree model because in terminal nodes have very small samples.

Results

The findings from this research suggest that using two classification models, K-Nearest Neighbors (KNN) and Decision Tree, in combination with Simple Hill Climbing algorithms, results in similar prediction accuracies: 0.719 and 0.729, respectively. But after visualising of decision tree indicates that the decision tree nearly appear to be overfiting problem with have to be improve in the future. The findings indicate that these two classification models without proper feature selection tend to yield low accuracy. In contrast, applying Simple Hill Climbing significantly improves the accuracy in classifying glass types.

Discussion

Comparing the two classification models reveals differences in implementation steps and results, but both models effectively solve the classification problem with high accuracy. The Decision Tree model provides clear and interpretable visualizations that are accessible to non-programming users, making its classification method visible and understandable. On the other hand, the K-Nearest Neighbors model also achieves high accuracy, but it poses some challenges in terms of result visualization compared to the Decision Tree.

After applying Simple Hill Climbing and K-Folds cross-validation, the report found that the accuracy of both models slightly decreased but resulted in more reliable and satisfactory outcomes.

Conclusion

In conclusion, both the K-Nearest Neighbors and Decision Tree models, when combined with Simple Hill Climbing and K-Folds cross-validation, provide effective solutions for classifying glass types. Proper feature selection is crucial in achieving higher accuracy. The Decision Tree model offers better interpretability, while the K-Nearest Neighbors model remains a robust alternative despite visualization challenges. Overall, the models demonstrate strong performance in the classification task with improved reliability through the application of Simple Hill Climbing and K-Folds cross-validation.

References

Sharpsight Labs, 2023. *How to use train_test_split from scikit-learn* [online]. Available at: https://www.sharpsightlabs.com/blog/scikit-train_test_split/ [Accessed 26 May 2024].

Real Python, 2023. *Train-Test Split in Python: A Guide [online]*. Available at: https://realpython.com/train-test-split-python-data/ [Accessed 25 May 2024].