CA5_2

April 29, 2024

0.1 Importing

[230]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler, OneHotEncoder,␣
↪PolynomialFeatures
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC, SVR
from sklearn.linear_model import LogisticRegression, LinearRegression,␣
↪ElasticNet, RANSACRegressor, BayesianRidge, Ridge
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,␣
↪GradientBoostingClassifier, BaggingClassifier, ExtraTreesClassifier,␣
↪RandomForestRegressor, GradientBoostingRegressor, AdaBoostClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import  mean_absolute_error
from xgboost import XGBClassifier



# To control warning messages. I should have done this last time to make your␣
↪life easier!
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore', "ConvergenceWarning")
```

1 Reading dataset

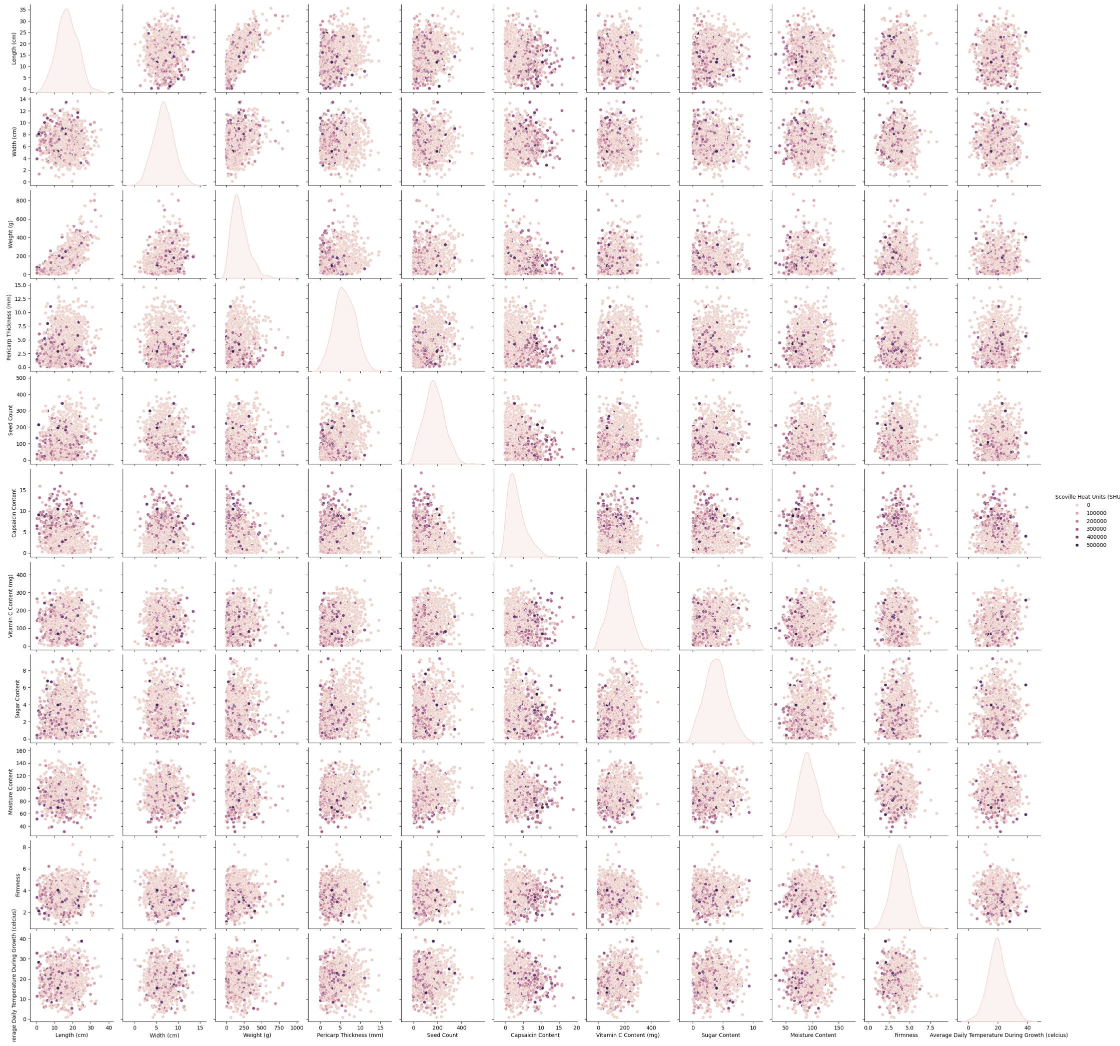
[231]:
```
# Load data
df = pd.read_csv("train.csv")
```

[231]:
```
     Length (cm)  Width (cm)  Weight (g)  Pericarp Thickness (mm)  Seed Count  \
0          17.37        5.42       94.30                     4.90      193.93
1          27.78        4.75      262.71                     6.56      186.29
2           6.17        3.51       66.72                     7.96      298.81
3           6.12        6.07       51.24                     4.57       39.36
4          28.58        4.84      166.51                     3.07      194.07
..           …           …           …                       …          …
995         8.67        6.51       19.00                     2.87        1.53
996        17.17        9.25      150.86                     1.41      386.87
997        14.16        6.87      124.72                     1.97      202.83
998         3.71        7.12       29.53                     1.05      115.61
999        14.33        8.99      179.04                     4.19      344.16

     Capsaicin Content  Vitamin C Content (mg)  Sugar Content  \
0                 3.21                  173.59           6.15
1                 8.19                  100.41           2.36
2                 4.69                  125.91           6.75
3                 2.76                  143.54           5.93
4                 7.01                  193.76           2.85
..                 …                       …              …
995               0.63                    9.02           0.63
996               2.27                  268.93           2.21
997               3.31                  203.84           2.90
998               9.80                   45.95           2.39
999               2.65                  164.35           1.11

     Moisture Content  Firmness   color Harvest Time  \
0               88.59      3.40     red       Midday
1              111.20      5.45   green       Midday
2               72.98      2.77     red       Midday
3               63.93      1.62  yellow       Midday
4               88.19      3.99     red       Midday
..                …         …      …           …
995             95.54      4.86  yellow      Evening
996            131.71      2.59  yellow      Morning
997            114.42      3.17  yellow      Evening
998             97.70      4.01   green      Evening
999             80.82      2.95   green      Evening

     Average Daily Temperature During Growth (celcius)  \
0                                                 8.68
1                                                22.44
```

```
2                                                  24.99
3                                                  13.05
4                                                  27.08
..                                                   …
995                                                16.57
996                                                22.39
997                                                15.84
998                                                16.05
999                                                22.88

    Average Temperature During Storage (celcius)  Scoville Heat Units (SHU)
0                                            5-6                       0.00
1                                            NaN                       0.00
2                                            NaN                  455995.06
3                                            NaN                       0.00
4                                            NaN                       0.00
..                                             …                          …
995                                          NaN                   88266.90
996                                          NaN                       0.00
997                                          7-8                       0.00
998                                          NaN                  188390.86
999                                          NaN                  409383.92

[1000 rows x 15 columns]
```

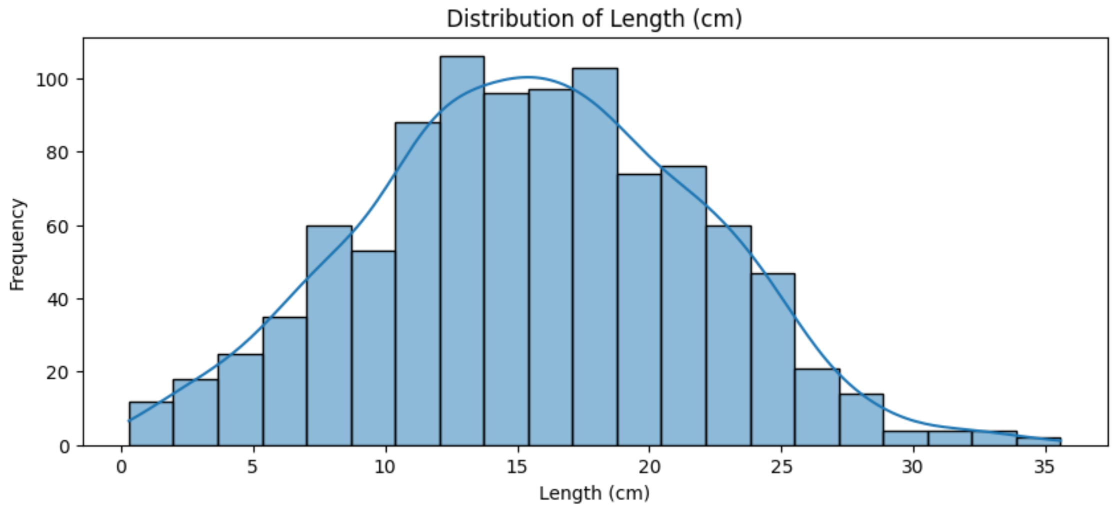
1.1 Visualisation and data exploring

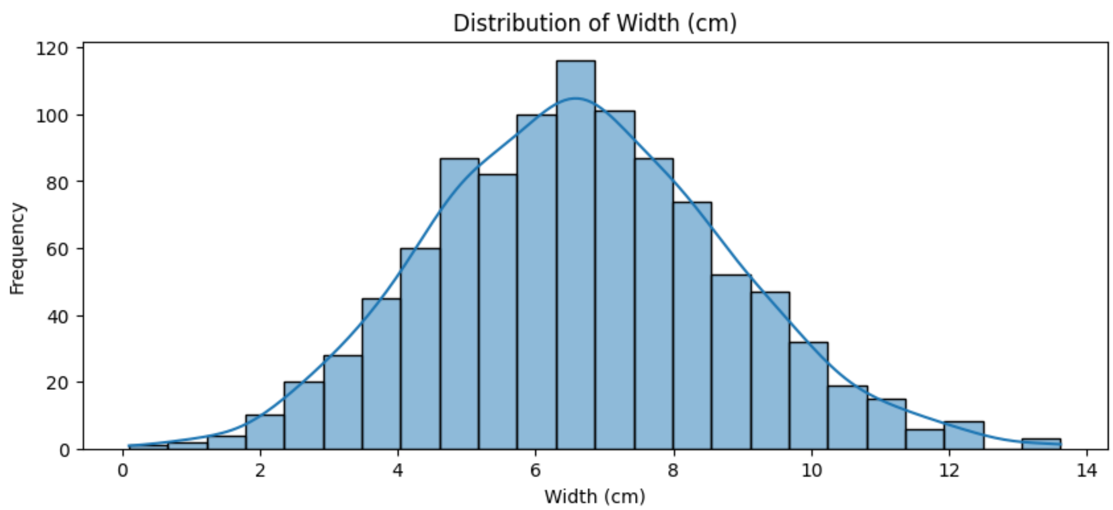
[232]:
```
# for comparing different features to each other from the dataset, with hue as
 ↪'Scoville Heat Units (SHU)'

sns.pairplot(df, hue='Scoville Heat Units (SHU)')
plt.legend
plt.show()
```

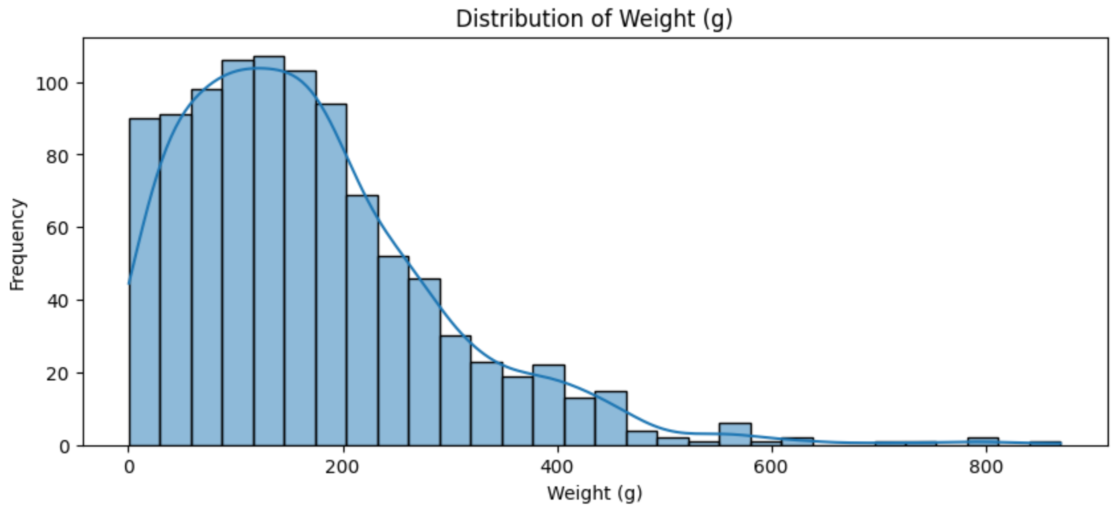


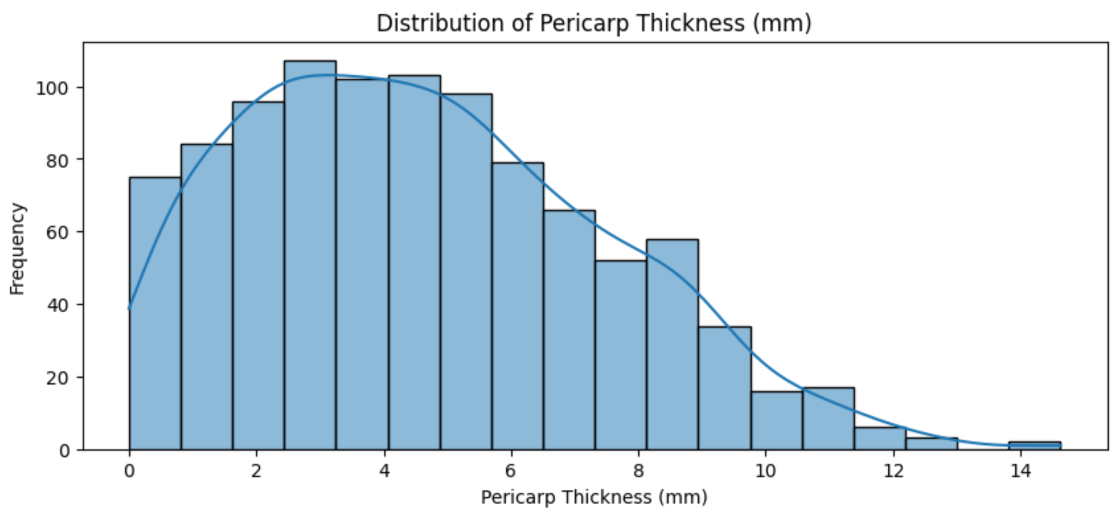
[233]:

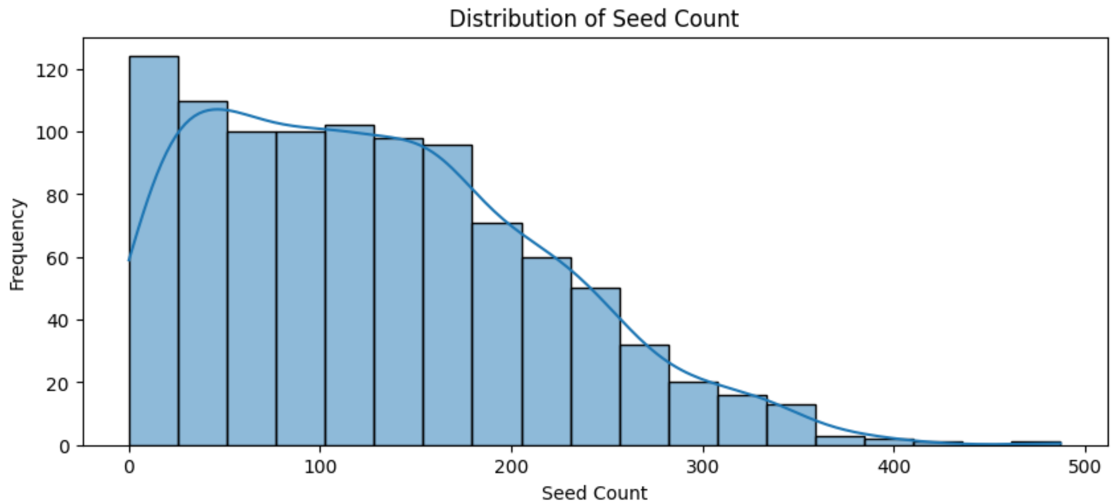
```python
#Checking the distorbution of each feature

for column in df.columns:
    plt.figure(figsize=(10, 4))
    sns.histplot(df[column], kde=True)  # kde (Kernel Density Estimate) adds a density curve
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

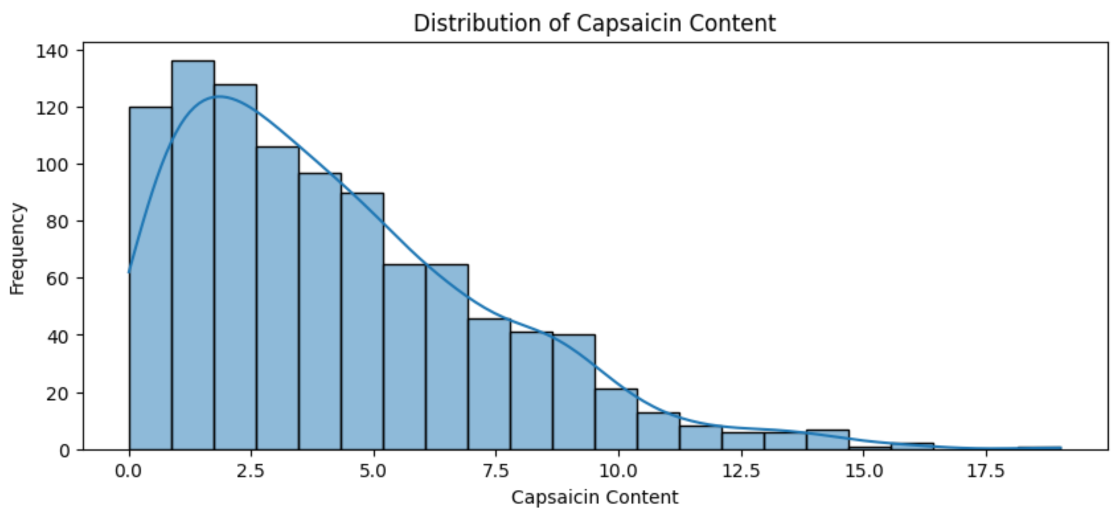

Distribution of Length (cm)
Frequency
Length (cm)
0
20
40
60
80
100
0
5
10
15
20
25
30
35

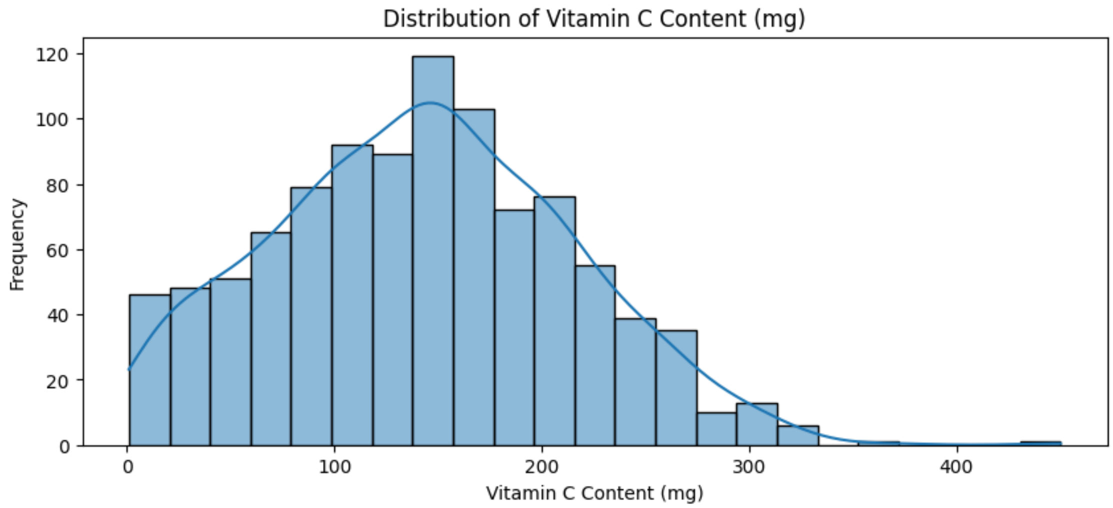

Distribution of Width (cm)
Frequency
Width (cm)
0
20
40
60
80
100
120
0
2
4
6
8
10
12
14

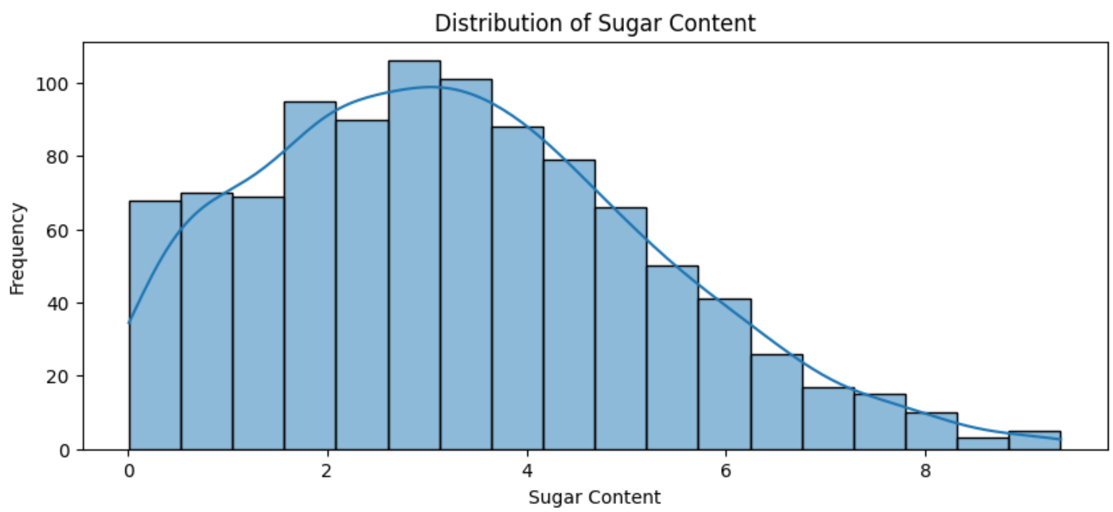

Distribution of Weight (g)
Frequency
0
20
40
60
80
100
0
200
400
600
800
Weight (g)

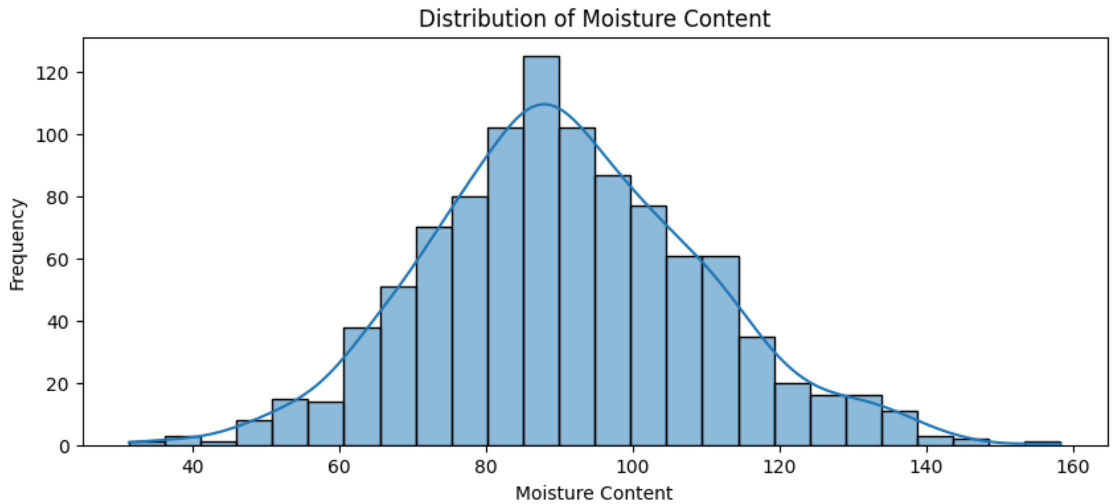

Distribution of Pericarp Thickness (mm)
Frequency
0
20
40
60
80
100
0
2
4
6
8
10
12
14
Pericarp Thickness (mm)

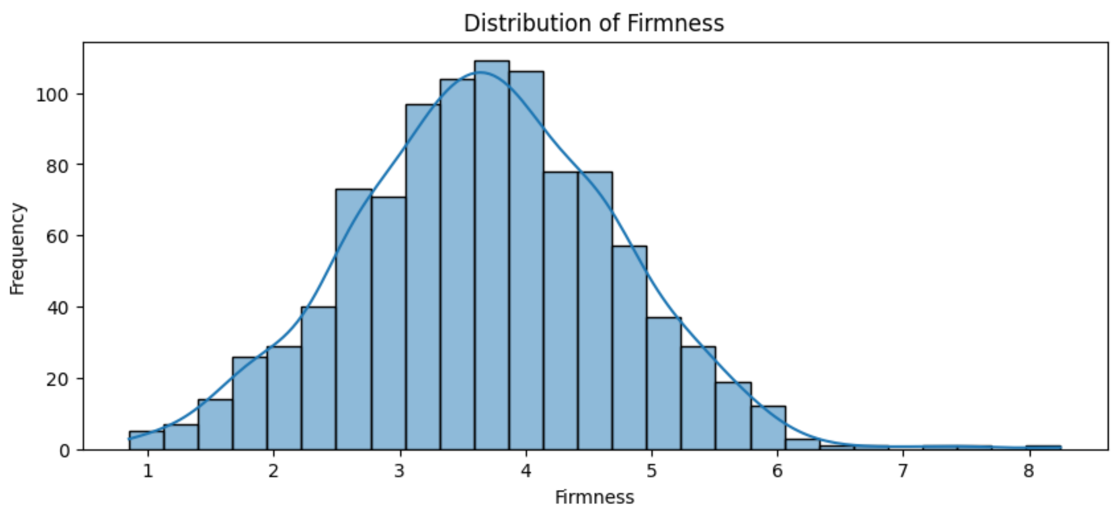

Distribution of Seed Count
Frequency
Seed Count
0
20
40
60
80
100
120
0
100
200
300
400
500

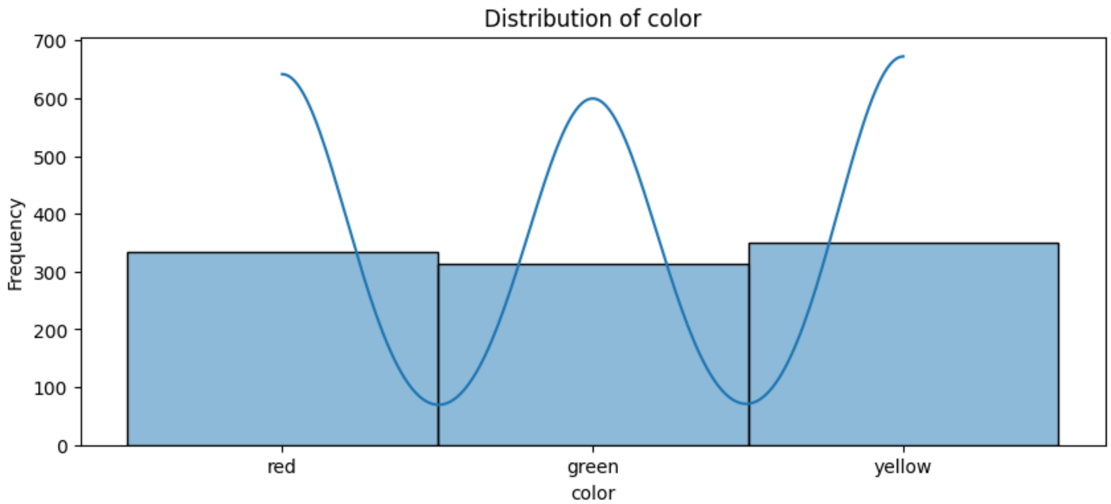

Distribution of Capsaicin Content
Frequency
Capsaicin Content
0
20
40
60
80
100
120
140
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

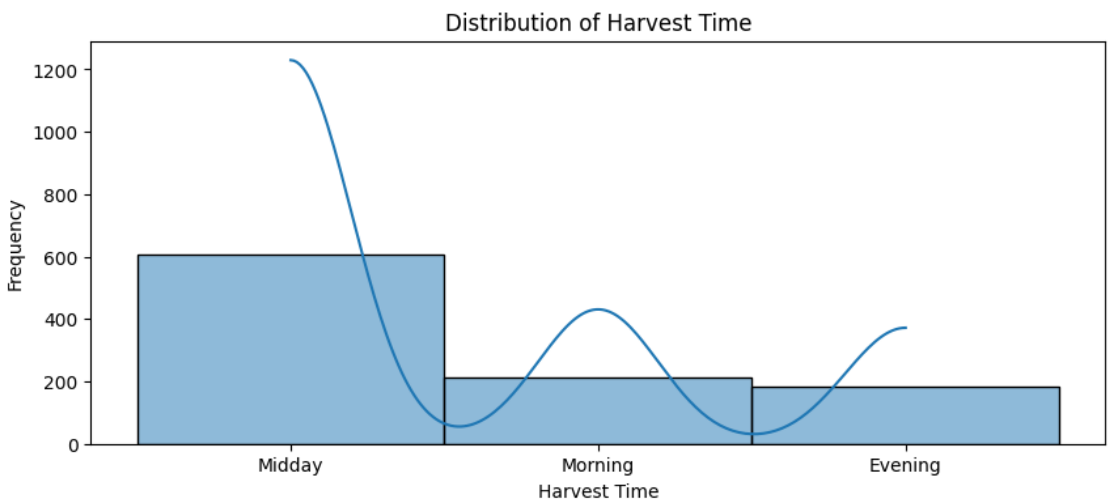

Distribution of Vitamin C Content (mg)
Frequency
Vitamin C Content (mg)
0
20
40
60
80
100
120
0
100
200
300
400

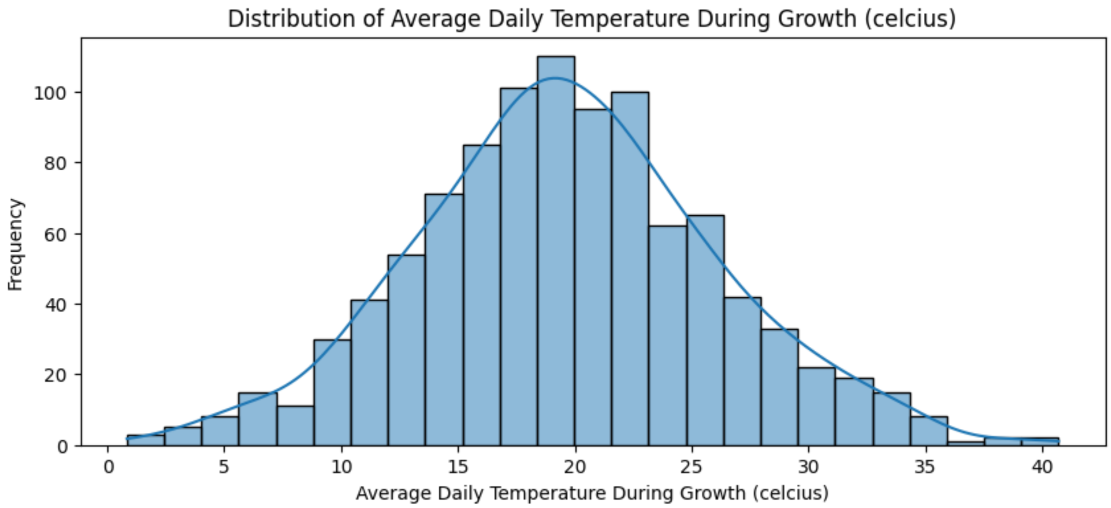

Distribution of Sugar Content
Frequency
Sugar Content
0
20
40
60
80
100
0
2
4
6
8

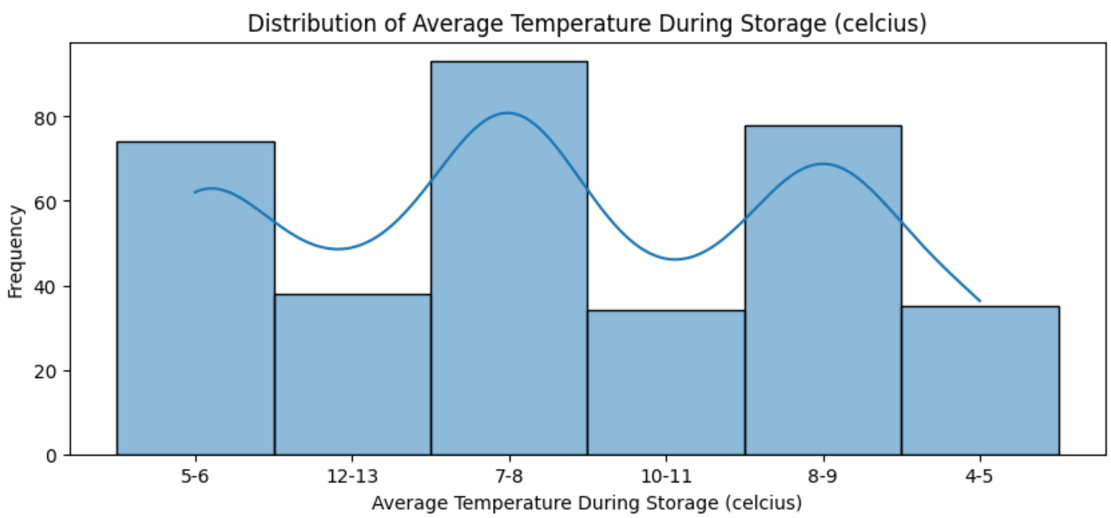

Distribution of Moisture Content
Frequency
Moisture Content
0
20
40
60
80
100
120
40
60
80
100
120
140
160

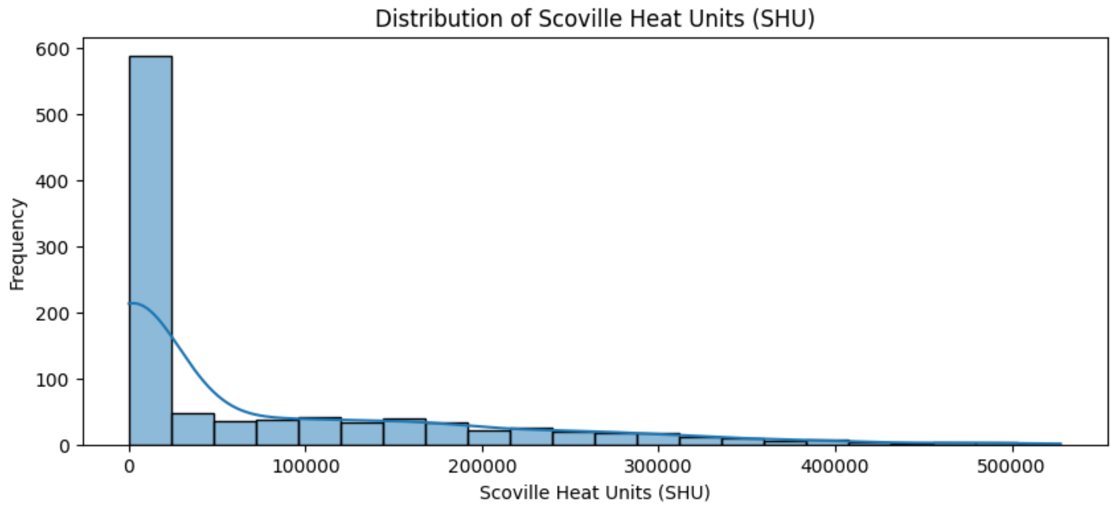

Distribution of Firmness
Frequency
Firmness
0
20
40
60
80
100
1
2
3
4
5
6
7
8


Distribution of color
Frequency
0
100
200
300
400
500
600
700
red
green
yellow
color


Distribution of Harvest Time
Frequency
0
200
400
600
800
1000
1200
Midday
Morning
Evening
Harvest Time


Distribution of Average Daily Temperature During Growth (celcius)
Frequency
0
20
40
60
80
100
0
5
10
15
20
25
30
35
40
Average Daily Temperature During Growth (celcius)


Distribution of Average Temperature During Storage (celcius)
Frequency
0
20
40
60
80
5-6
12-13
7-8
10-11
8-9
4-5
Average Temperature During Storage (celcius)



[234]:
```
df.describe()
```

[234]:
```
       Length (cm)  Width (cm)  Weight (g)  Pericarp Thickness (mm)  \
count   999.000000  999.000000  999.000000               998.000000
mean     15.574675    6.641572  169.346406                 4.619499
std       6.267303    2.139023  123.779026                 2.829503
min       0.300000    0.100000    0.560000                 0.000000
25%      11.290000    5.140000   79.020000                 2.400000
50%      15.520000    6.600000  147.230000                 4.280000
75%      19.900000    8.045000  227.625000                 6.560000
max      35.570000   13.620000  869.970000                14.630000

       Seed Count  Capsaicin Content  Vitamin C Content (mg)  Sugar Content  \
count  999.000000         999.000000             1000.000000     999.000000
mean   128.731301           4.215385              142.035180       3.283534
std     87.270366           3.163125               72.246142       1.938264
min      0.040000           0.010000                0.950000       0.010000
25%     55.390000           1.710000               92.290000       1.865000
50%    119.490000           3.590000              141.730000       3.140000
75%    186.845000           6.115000              192.720000       4.555000
max    487.260000          19.020000              450.290000       9.360000

       Moisture Content    Firmness  \
count       1000.000000  999.000000
mean          90.878380    3.679179
std           18.724314    1.034726
min           31.400000    0.850000
25%           78.585000    2.980000
50%           89.690000    3.660000
```

```
75%         103.200000    4.375000
max         158.300000    8.250000

       Average Daily Temperature During Growth (celcius)  \
count                                         1000.000000
mean                                            19.641960
std                                              6.436255
min                                              0.840000
25%                                             15.397500
50%                                             19.495000
75%                                             23.530000
max                                             40.700000

       Scoville Heat Units (SHU)
count                1000.000000
mean                70941.260020
std                108149.917069
min                     0.000000
25%                     0.000000
50%                     0.000000
75%                121349.617500
max                527639.860000
```

[235]:

```
#finding missing values in each feature
nan_per_column = df.isna().sum()
#finding totaol missing values in dataset
total_nan = df.isna().sum().sum()

# To display the number of NaNs per column
print(nan_per_column)
# To display the total number of NaNs in the DataFrame
print(f"Total number of NaN values in the DataFrame: {total_nan}")
```

```
Length (cm)                                            1
Width (cm)                                             1
Weight (g)                                             1
Pericarp Thickness (mm)                                2
Seed Count                                             1
Capsaicin Content                                      1
Vitamin C Content (mg)                                 0
Sugar Content                                          1
Moisture Content                                       0
Firmness                                               1
color                                                  1
Harvest Time                                           0
Average Daily Temperature During Growth (celcius)      0
Average Temperature During Storage (celcius)         648
Scoville Heat Units (SHU)                              0
```

```
dtype: int64
Total number of NaN values in the DataFrame: 658
```

2 !!!Comments on visualisation of data!!!

There was a lot of missing values on the feature “Average Temperature During Storage (celcius)”, so i’m removing it. Im replacing the rest of the missing values in the dataset with the median in the preprocesser

It looks like there wasn’t much outliers in the dataset. Maybe som big values in ‘Average Temperature During Storage (celcius)’, so i set a threshold there for max values. It worked well on the models i used at least.

There is some categorical data that need transformation.

I tried to put together features like Weight and lenght into one feature, but it didn’t make much difference, so i didn’t stick with it.

3 Data cleaning and preprocessing

[236]:
```
#Dropping the dataset with a lot if missing values
# I'm using SimpleImputer in the preprocesser that will replace missing values.
df.drop('Average Temperature During Storage (celcius)', axis=1, inplace=True)
# Setting a threshold for 'Vitamin C Content (mg)' at 400.
df.drop(df[df['Vitamin C Content (mg)'] > 400].index, inplace=True)
```

[237]:
```
# Separate features and target
X = df.drop('Scoville Heat Units (SHU)', axis=1)
y = df['Scoville Heat Units (SHU)']

# Identify column types
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = X.select_dtypes(include=['object']).columns

#making the preprocesser, that handles both numerical and categorical data
preprocessor = ColumnTransformer([
    ('num', Pipeline([ #for numerical data
        ('imputer', SimpleImputer(strategy='median')), #replacing missing␣
 ↪values with median of the feature
    ]), numerical_cols),
    ('cat', Pipeline([ #for categorical data
        ('encoder', OneHotEncoder(handle_unknown='ignore')) #For transforming␣
 ↪categorical data into numeric values
    ]), categorical_cols)
])

# Split data into train and test sets
```

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

#selecting y/y_train/y_test_binary as values bigger than 1 for the ensemble␣
    ↪classifier
y_train_binary = (y_train > 0).astype(int)
y_test_binary = (y_test > 0).astype(int)
y_binary= (y > 0).astype(int)
```

3.1 Model selecting

Here i select the models that works best for the dataset! I want to make a regression pipeline (A) and two seqential pipelines, one ensemble classifier and one regression classifier (C). I only included the model selector for (C), because it's literally the same procedure as (A), with only difference that i choose a combination of models with best MAE-score

[248]:
```
# Base estimators for stacking and voting
base_estimators = [
    ('lr', Pipeline([
    ('scaler', StandardScaler()),
    ('logistic', LogisticRegression(random_state=42, max_iter=1000)) ])),
    ('svc', SVC(probability=True, random_state=42)),
    ('dt', DecisionTreeClassifier(random_state=42)),
    ('rf', RandomForestClassifier(random_state=42)),
    ('knn', KNeighborsClassifier())
]

# Create a decision tree classifier with max_depth=2
base_estimator = DecisionTreeClassifier(max_depth=2)


# Lists of ensemble models to try
classifiers = [
    XGBClassifier(),
    RandomForestClassifier(random_state=42),
    GradientBoostingClassifier(random_state=42),
    #ExtraTreesClassifier(random_state=42), # Not sure if allowed
    BaggingClassifier(random_state=42),
]

#List of regressors to try
regressors = [
    RANSACRegressor(random_state=42),
    LinearRegression(),
    Ridge(random_state=42),
    ElasticNet(random_state=42),
    DecisionTreeRegressor(random_state=42),
```

```
    RandomForestRegressor(random_state=42),
    GradientBoostingRegressor(random_state=42),
    BayesianRidge(),
    SVR(),
    Pipeline([
        ('polynomial_features', PolynomialFeatures()),
        ('linear_regression', LinearRegression())
    ]),
    Pipeline([
        ('pca', PCA()),  # Adjust n_components as needed
        ('linear_regression', LinearRegression())
    ])
]
```

[249]:
```
# For scoring best score and info
best_score = np.inf
best_combo = None

#Kfold cross validation with 10 folds
kf = KFold(n_splits=10, shuffle=True, random_state=42)

#iterating over diferent classifiers in list
for clf in classifiers:
    # Create a unique name for the classifier pipeline for easy identification
    pipeline_clf_name = f'{clf.__class__.__name__}_pipeline'
    #define pipeline for ensemble classifier
    pipeline_clf = Pipeline([
        ('preprocessing', preprocessor),
        ('classifier', clf)
    ])

    #iterating over different classifiers in list
    for reg in regressors:
         # Create a unique name for the classifier pipeline for easy␣
 ↪identification
        pipeline_reg_name = f'{reg.__class__.__name__}_pipeline'
        # Create pipeline for the regressor with a name
        pipeline_reg = Pipeline([
            ('preprocessing', preprocessor),
            ('scaling', StandardScaler()),
            ('regressor', reg)
        ])

        # Perform cross-validation

         # List to store the Mean Absolute Error for each fold
        scores = []
```

```
        for train_index, test_index in kf.split(X):
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train, y_test = y.iloc[train_index], y.iloc[test_index]

            # Convert the target variable to binary (1 for spicy, 0 for not␣
 ↪spicy)
            y_train_binary = (y_train > 0).astype(int)

            # Fit the classifier pipeline on the training data
            pipeline_clf.fit(X_train, y_train_binary)
            # Identify indices of spicy samples to train the regressor only on␣
 ↪these
            spicy_peppers_indices = (pipeline_clf.predict(X_train) == 1)

             # Fit the regressor pipeline only on spicy samples
            pipeline_reg.fit(X_train[spicy_peppers_indices],␣
 ↪y_train[spicy_peppers_indices])

             # Get predictions for spicy/not spicy from the classifier
            binary_predictions = pipeline_clf.predict(X_test)

             # Get regression predictions for the test set
            regression_predictions = pipeline_reg.predict(X_test)

             # Combine predictions: Only apply regression predictions where the␣
 ↪classifier predicts spicy
            combined_predictions = regression_predictions * binary_predictions

            # Calculate Mean Absolute Error for the combined predictions
            mae = mean_absolute_error(y_test, combined_predictions)
            scores.append(mae)

         # Calculate the average MAE across all folds
        average_mae = np.mean(scores)
        # Update the best score and store the best classifier/regressor␣
 ↪combination if this combo is better
        if average_mae < best_score:
            best_score = average_mae
            best_combo = (clf, reg)
            best_combo_names = (pipeline_clf_name, pipeline_reg_name)

print(f'Best combination: {best_combo_names[0]} + {best_combo_names[1]} with␣
 ↪average MAE {best_score}')
```

```
Best combination: GradientBoostingClassifier_pipeline + ElasticNet_pipeline with
average MAE 56668.42040089873
```

3.2 Pipeline (A)

[250]:
```
# Making a pipeline with LinearRegression()
regression_pipeline = Pipeline([
    ('preprocessor', preprocessor),  # Preprocessing
    ('scaling', StandardScaler()),  # Scaling features
    ('regression', LinearRegression())  # Linear regression model
])

# Fit the model
regression_pipeline.fit(X_train, y_train)

# Predict using the model
regression_predictions = regression_pipeline.predict(X_test)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, regression_predictions)

print(f"Mean Absolute Error (MAE) on Test Data: {mae}")
```

```
Mean Absolute Error (MAE) on Test Data: 56121.327718945155
```

3.3 Pipelines for (C) (Please read under)

So i used ExtraTreesClassifier and Ridge classifiers for (C). Then i realized that we havent learning about ExtraTreesClassifier in this course, so if it isn't allowed then my best score on kaggle shouldn't count. Ive made pipelines with Gradientboosting and ElasticNet if ExtraTreeClassifier isn't allowed.

3.4 My best combination of models

[251]:
```
# Define the pipeline for binary classifier
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier',␣
 ↪ExtraTreesClassifier(random_state=42))])

# Define the hyperparameters grid to be tuned
params = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [None, 10, 20, 30],
}

# Setup the GridSearchCV object
binary_classifier_pipeline = GridSearchCV(estimator=pipeline,
                                          param_grid=params,
                                          cv=5,
                                          scoring='neg_mean_absolute_error',
                                          verbose=1,
                                          n_jobs=-1)
```

[252]:
```python
# # Define the pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                            ('scaling', StandardScaler()),
                           ('regressor', Ridge(random_state=42))])

# Define the hyperparameters grid to be tuned
params = {
    'regressor__alpha': [0.1, 1.0, 10.0, 100.0],
    # Add other parameters you want to tune
}

# Setup the GridSearchCV object
regression_pipeline = GridSearchCV(estimator=pipeline,
                           param_grid=params,
                           cv=5,
                           scoring='neg_mean_absolute_error',
                           verbose=1,
                           n_jobs=-1)
```

3.5 Alternative Models

[261]:
```python
# This is the binary classifier pipeline im using if ExtraTreeClassifier isn't␣
↪allowed.


pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', GradientBoostingClassifier(random_state=42))
])

# Define the hyperparameters grid to be tuned for the GradientBoostingClassifier
params = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__learning_rate': [0.01, 0.1, 0.2],  # Learning rate to tune
    'classifier__max_depth': [3, 5, 7],  # Depths to tune
    # You can add other parameters here to tune
}

# Setup the GridSearchCV object for the binary classifier
binary_classifier_pipeline = GridSearchCV(estimator=pipeline,
                                          param_grid=params,
                                          cv=5,
                                          scoring='neg_mean_absolute_error',
                                          verbose=1,
                                          n_jobs=-1)
```

[264]:
```
#This is the regressor pipeline im using if ExtraTreeClassifier isn't allowed

# Define the pipeline with an Elastic Net regressor
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('scaling', StandardScaler()),
    ('regressor', ElasticNet(random_state=42))
])

# Define the hyperparameters grid to be tuned for the Elastic Net
params = {
    'regressor__alpha': [0.1, 1.0, 10.0, 100.0],  # Regularization strength
    'regressor__l1_ratio': [0.1, 0.5, 0.9],       # Mix ratio of L1 and L2␣
 ↪regularization
}

# Setup the GridSearchCV object for the regression pipeline
regression_pipeline = GridSearchCV(estimator=pipeline,
                                   param_grid=params,
                                   cv=5,
                                   scoring='neg_mean_absolute_error',
                                   verbose=1,
                                   n_jobs=-1)
```

3.6 Training and evaluation of models

[265]:
```
# Fitting binary classifier with trainig sets.
binary_classifier_pipeline.fit(X_train, y_train_binary)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

[265]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
                                        ColumnTransformer(transformers=[('num',
Pipeline(steps=[('imputer',
          SimpleImputer(strategy='median'))]),
Index(['Length (cm)', 'Width (cm)', 'Weight (g)', 'Pericarp Thickness (mm)',
       'Seed Count', 'Capsaicin Content', 'Vitamin C Content (mg)',
       'Sugar Content', 'Moisture Content', 'Firmness',
       'Average Daily Temperatur…
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
Index(['color', 'Harvest Time'], dtype='object'))])),
                                       ('classifier',
GradientBoostingClassifier(random_state=42))]),
             n_jobs=-1,
             param_grid={'classifier__learning_rate': [0.01, 0.1, 0.2],
```

```
                        'classifier__max_depth': [3, 5, 7],
                        'classifier__n_estimators': [100, 200, 300]},
             scoring='neg_mean_absolute_error', verbose=1)
```

[266]:
```
# To estimate the scoville score (SHU) of those samples that the binary␣
 ↪classifier identifies as spicy peppers.
spicy_peppers_indices = (binary_classifier_pipeline.predict(X_train) == 1)

# Training the model
regression_pipeline.fit(X_train[spicy_peppers_indices],␣
 ↪y_train[spicy_peppers_indices])

# Combine predictions of both models into a single prediction vector
binary_predictions = binary_classifier_pipeline.predict(X_test)
regression_predictions = regression_pipeline.predict(X_test)

# Combine predictions for spicy peppers (SHU > 0)
combined_predictions = regression_predictions * binary_predictions

# Evaluate the combined model
mae_combined = mean_absolute_error(y_test, combined_predictions)
print("Mean Absolute Error (Combined Model):", mae_combined)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Mean Absolute Error (Combined Model): 48329.081649784996
```

[267]:
```
# Fit all the data for the binary classifier
binary_classifier_pipeline.fit(X, y_binary)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

[267]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
                                        ColumnTransformer(transformers=[('num',
Pipeline(steps=[('imputer',
          SimpleImputer(strategy='median'))]),
Index(['Length (cm)', 'Width (cm)', 'Weight (g)', 'Pericarp Thickness (mm)',
       'Seed Count', 'Capsaicin Content', 'Vitamin C Content (mg)',
       'Sugar Content', 'Moisture Content', 'Firmness',
       'Average Daily Temperatur…
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
Index(['color', 'Harvest Time'], dtype='object'))])),
                                       ('classifier',
GradientBoostingClassifier(random_state=42))]),
             n_jobs=-1,
             param_grid={'classifier__learning_rate': [0.01, 0.1, 0.2],
                        'classifier__max_depth': [3, 5, 7],
```

```
                             'classifier__n_estimators': [100, 200, 300]},
             scoring='neg_mean_absolute_error', verbose=1)
```

[268]:
```
spicy_peppers_indices = (binary_classifier_pipeline.predict(X) == 1)
#Fit all the data for the regresssion classifier
regression_pipeline.fit(X[spicy_peppers_indices], y[spicy_peppers_indices])
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

[268]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
                                        ColumnTransformer(transformers=[('num',
Pipeline(steps=[('imputer',
          SimpleImputer(strategy='median'))]),
Index(['Length (cm)', 'Width (cm)', 'Weight (g)', 'Pericarp Thickness (mm)',
       'Seed Count', 'Capsaicin Content', 'Vitamin C Content (mg)',
       'Sugar Content', 'Moisture Content', 'Firmness',
       'Average Daily Temperatur…
      dtype='object')),
                                                                        ('cat',
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
Index(['color', 'Harvest Time'], dtype='object'))])),
                                       ('scaling', StandardScaler()),
                                       ('regressor',
                                        ElasticNet(random_state=42))]),
             n_jobs=-1,
             param_grid={'regressor__alpha': [0.1, 1.0, 10.0, 100.0],
                         'regressor__l1_ratio': [0.1, 0.5, 0.9]},
             scoring='neg_mean_absolute_error', verbose=1)
```

3.7 Kaggle Submission

[269]:
```
#Load test set
X_test = pd.read_csv("test.csv")

#Use the trained pipelines to make predictions on the test set
binary_predictions = binary_classifier_pipeline.predict(X_test)
regression_predictions = regression_pipeline.predict(X_test)

#Combine predictions of both models into a single prediction vector
combined_predictions = regression_predictions * binary_predictions

# Create a DataFrame with the index and predicted SHU values
results_df = pd.DataFrame({'index': range(len(X_test)), 'Scoville Heat Units␣
 ↪(SHU)': combined_predictions})

# Save the DataFrame to a CSV file with the specified format
```

```python
results_df.to_csv('predicted_shu.csv', index=False)

# Display the first few rows of the generated file
print(results_df.head())
```

```
   index  Scoville Heat Units (SHU)
0      0              115594.011090
1      1               83444.914943
2      2                   0.000000
3      3                   0.000000
4      4              241134.189192
```