CA5

May 2, 2024

1 CA5

Name: Anniken Rabben
Kaggle username: Anniken01

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
```

[ ]:
```
raw_train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
raw_train.head()
```

[ ]:

	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	

	Capsaicin Content	Vitamin C Content (mg)	Sugar Content	Moisture Content	\
0	3.21	173.59	6.15	88.59	
1	8.19	100.41	2.36	111.20	

```
2              4.69                  125.91          6.75             72.98
3              2.76                  143.54          5.93             63.93
4              7.01                  193.76          2.85             88.19

   Firmness   color Harvest Time  \
0      3.40     red       Midday
1      5.45   green       Midday
2      2.77     red       Midday
3      1.62  yellow       Midday
4      3.99     red       Midday

   Average Daily Temperature During Growth (celcius)  \
0                                              8.68
1                                             22.44
2                                             24.99
3                                             13.05
4                                             27.08

  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
```

[ ]:
```
raw_train.describe()
```

[ ]:
```
       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
```

[ ]:
```
# check for missing data
print(raw_train.isnull().sum())
print()
print(test.isnull().sum())
```

```
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

Length (cm)                                            2
Width (cm)                                             0
Weight (g)                                             0
Pericarp Thickness (mm)                                0
Seed Count                                             0
Capsaicin Content                                      1
Vitamin C Content (mg)                                 0
Sugar Content                                          0
Moisture Content                                       1
Firmness                                               2
color                                                  0
Harvest Time                                           0
Average Daily Temperature During Growth (celcius)      0
Average Temperature During Storage (celcius)         522
dtype: int64
```

Note: decided to drop the column 'Average Temperature During Storage (celcius)' because it has nearly 65% missing data

[ ]:
```
# drop the column 'Average Temperature During Storage (celcius)'
train = raw_train.drop(['Average Temperature During Storage (celcius)'], axis=1)
test = test.drop(['Average Temperature During Storage (celcius)'], axis=1)

print(train.shape)
print(test.shape)
```

```
(1000, 14)
(800, 13)
```

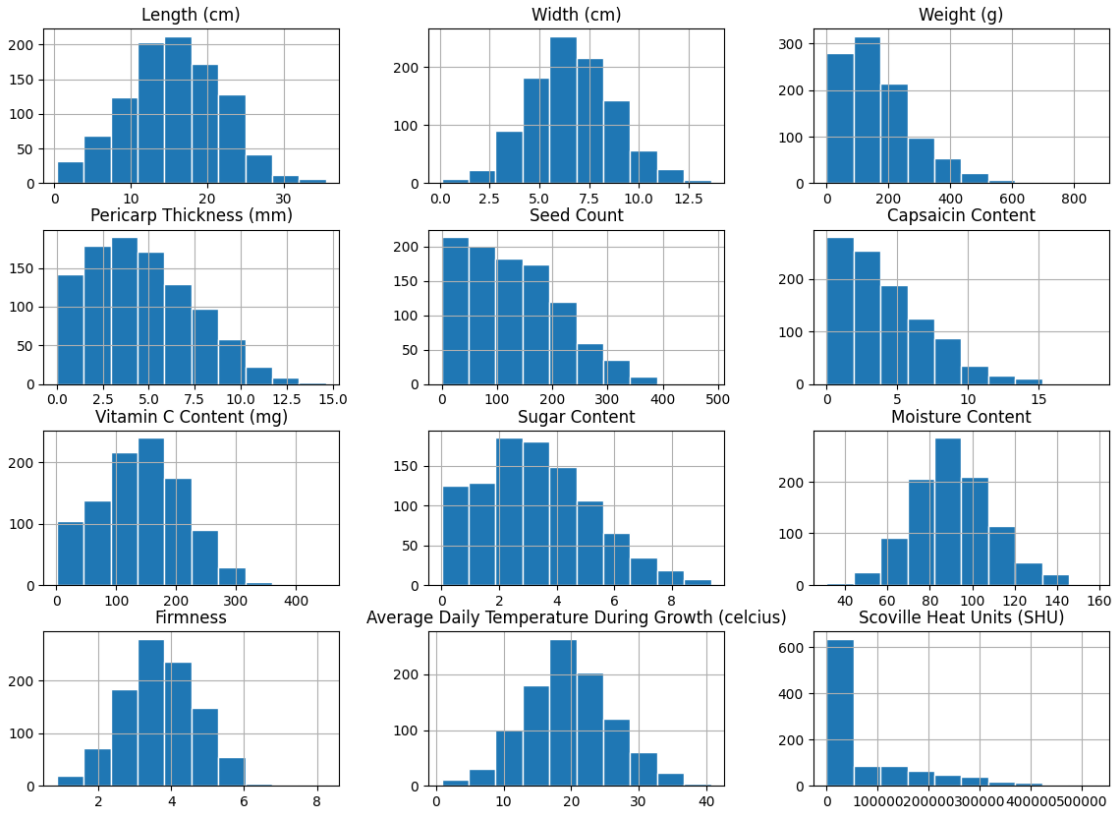
Note: decided to fill missing values in the rest of the columns that has missing values, since the number of missing values is low

[ ]:
```
# Fill inn missing values
# Combine train and test DataFrames into a single list
data = [train, test]

# List of numeric columns
numeric_cols = ['Length (cm)', 'Width (cm)', 'Weight (g)', 'Pericarp Thickness␣
 ↪(mm)', 'Seed Count', 'Capsaicin Content', 'Sugar Content','Moisture␣
 ↪Content', 'Firmness']

# Fill missing values in numeric columns with the mean
for df in data:
    for col in numeric_cols:
        df[col] = df[col].fillna(df[col].mean())
```

```python
# Fill missing values in categorical columns
categorical_cols = ['color', 'Harvest Time']
for col in categorical_cols:
    for df in data:
        df[col] = df[col].fillna(data[0][col].mode().iloc[0])

# Check for missing data
print("Missing values in train DataFrame:")
print(train.isnull().sum())

print("\nMissing values in test DataFrame:")
print(test.isnull().sum())
```

```
Missing values in train DataFrame:
Length (cm)                                    0
Width (cm)                                     0
Weight (g)                                     0
Pericarp Thickness (mm)                        0
Seed Count                                     0
Capsaicin Content                              0
Vitamin C Content (mg)                         0
Sugar Content                                  0
Moisture Content                               0
Firmness                                       0
color                                          0
Harvest Time                                   0
Average Daily Temperature During Growth (celcius)    0
Scoville Heat Units (SHU)                      0
dtype: int64

Missing values in test DataFrame:
Length (cm)                                    0
Width (cm)                                     0
Weight (g)                                     0
Pericarp Thickness (mm)                        0
Seed Count                                     0
Capsaicin Content                              0
Vitamin C Content (mg)                         0
Sugar Content                                  0
Moisture Content                               0
Firmness                                       0
color                                          0
Harvest Time                                   0
Average Daily Temperature During Growth (celcius)    0
dtype: int64
```

[ ]:
```
# Histogram for each feature
raw_train.hist(bins=10, figsize=(14,10), edgecolor='white')
plt.show()
```



[ ]:
```
# Pairplot to get an overview of the data
sns.pairplot(data=raw_train, hue = 'Scoville Heat Units (SHU)')
plt.show()
```

[ ]:
```
numeric_cols = ['Length (cm)', 'Width (cm)', 'Weight (g)', 'Pericarp Thickness␣
 ↪(mm)', 'Seed Count', 'Capsaicin Content', 'Sugar Content', 'Firmness']
test[numeric_cols] = test[numeric_cols].fillna(test[numeric_cols].mean())

# Fill missing values in categorical columns with the mode
categorical_cols = ['color', 'Harvest Time']
test[categorical_cols] = test[categorical_cols].fillna(test[categorical_cols].
 ↪mode().iloc[0])

# check for missing data
print(test.isnull().sum())
```

```
Length (cm)                                    0
Width (cm)                                     0
Weight (g)                                     0
```

```
Pericarp Thickness (mm)                                0
Seed Count                                             0
Capsaicin Content                                      0
Vitamin C Content (mg)                                 0
Sugar Content                                          0
Moisture Content                                       0
Firmness                                               0
color                                                  0
Harvest Time                                           0
Average Daily Temperature During Growth (celcius)      0
dtype: int64
```

[ ]:
```
# data types
print(train.dtypes)
```

```
Length (cm)                                          float64
Width (cm)                                           float64
Weight (g)                                           float64
Pericarp Thickness (mm)                              float64
Seed Count                                           float64
Capsaicin Content                                    float64
Vitamin C Content (mg)                               float64
Sugar Content                                        float64
Moisture Content                                     float64
Firmness                                             float64
color                                                 object
Harvest Time                                          object
Average Daily Temperature During Growth (celcius)    float64
Scoville Heat Units (SHU)                            float64
dtype: object
```

[ ]:
```
# Find the unique values in the 'color' and 'Harvest Time' columns
print(train['Harvest Time'].unique())
print(train['color'].unique())
```

```
['Midday' 'Morning' 'Evening']
['red' 'green' 'yellow']
```

[ ]:
```
# Define the columns to be one-hot encoded
categorical_cols = ['color', 'Harvest Time']

# Create a ColumnTransformer
ct = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(drop='first', dtype=int), categorical_cols)
    ],
    remainder='passthrough',  # Remainder columns will be passed through␣
 ↪without any transformations
```

```
    verbose_feature_names_out=False,
)

# Apply the ColumnTransformer to the 'train' DataFrame
train = pd.DataFrame(ct.fit_transform(train), columns=ct.
 ↪get_feature_names_out())

train.head()

# source: Lectures/Lecture_12_Preprocessing/04_categorical_data_encoding.ipynb
```

[ ]:

	color_red	color_yellow	Harvest Time_Midday	Harvest Time_Morning	\
0	1.0	0.0	1.0	0.0	
1	0.0	0.0	1.0	0.0	
2	1.0	0.0	1.0	0.0	
3	0.0	1.0	1.0	0.0	
4	1.0	0.0	1.0	0.0	

	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	

	Capsaicin Content	Vitamin C Content (mg)	Sugar Content	Moisture Content	\
0	3.21	173.59	6.15	88.59	
1	8.19	100.41	2.36	111.20	
2	4.69	125.91	6.75	72.98	
3	2.76	143.54	5.93	63.93	
4	7.01	193.76	2.85	88.19	

	Firmness	Average Daily Temperature During Growth (celcius)	\
0	3.40	8.68	
1	5.45	22.44	
2	2.77	24.99	
3	1.62	13.05	
4	3.99	27.08	

	Scoville Heat Units (SHU)
0	0.00
1	0.00
2	455995.06
3	0.00
4	0.00

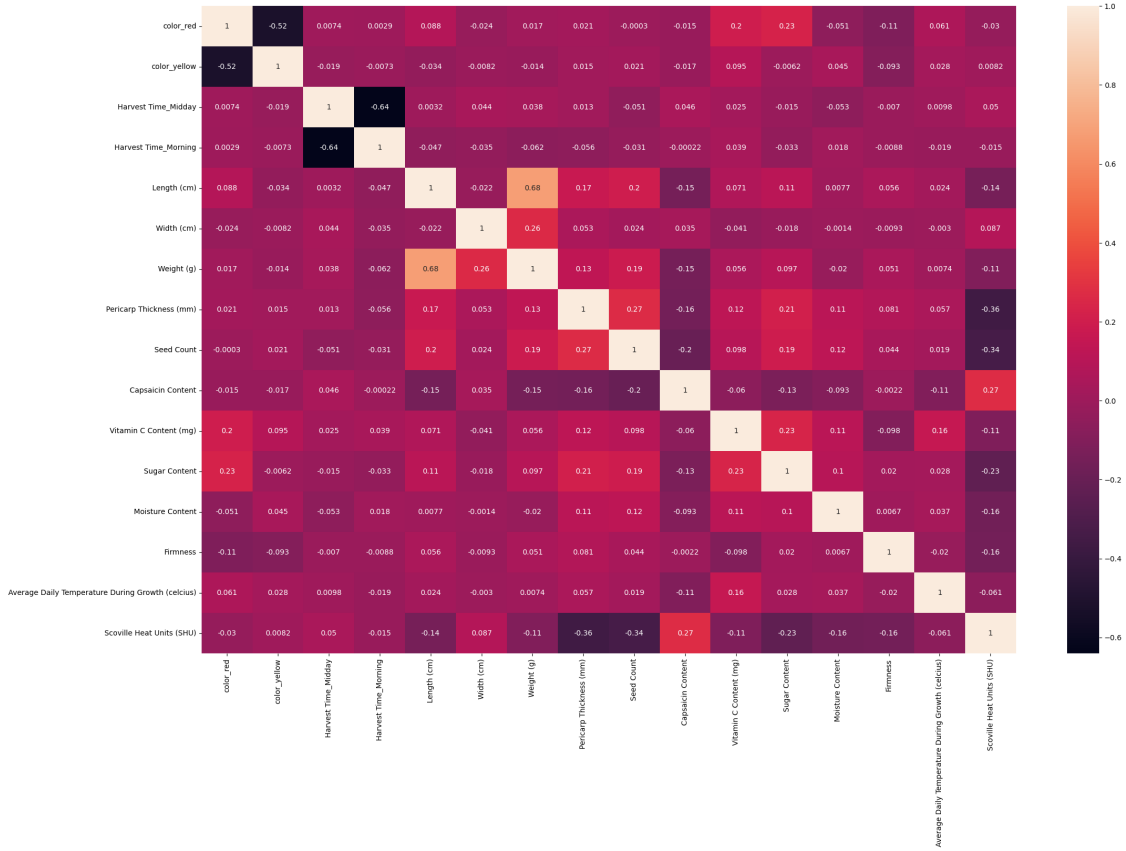
[ ]:
```
categorical_cols = ['color', 'Harvest Time']

# Create a ColumnTransformer
ct = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(drop='first', dtype=int), categorical_cols)
    ],
    remainder='passthrough',  # Remainder columns will be passed through␣
 ↪without any transformations
    verbose_feature_names_out=False,
)

# Apply the ColumnTransformer to the 'test' DataFrame
test = pd.DataFrame(ct.fit_transform(test), columns=ct.get_feature_names_out())

# source: Lectures/Lecture_12_Preprocessing/04_categorical_data_encoding.ipynb
```

[ ]:
```
# Using Z-scores to filter out the outliers
print(f'Shape of dataset before removing outliers: {train.shape}')
z_scores = stats.zscore(train)
abs_z_scores = np.abs(z_scores)
not_outliers = (abs_z_scores < 3).all(axis=1)
cleaned = train[not_outliers]

print(f'Shape of dataset after removing outliers: {cleaned.shape}')
```

```
Shape of dataset before removing outliers: (1000, 16)
Shape of dataset after removing outliers: (940, 16)
```

[ ]:
```
# Correlation matrix after cleaning
corr_matrix = cleaned.corr()
fig_corr, ax_corr = plt.subplots(figsize=(25,16))
sns.heatmap(data=corr_matrix,annot=True, ax=ax_corr)
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Note: From the correlation matrix, it might be correaltion between weight(kg) and length(cm)

[ ]:
```
# data preparation
X = cleaned.drop(columns=['Scoville Heat Units (SHU)'])
y = cleaned['Scoville Heat Units (SHU)']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=42)
```

2 A - Linear Regression Analysis

[ ]:
```
# Define the preprocessor
preprocessor = Pipeline([
    ('scaler', StandardScaler()),
    ('feature_selection', SelectKBest())
])

# Create the pipeline
pipeline = Pipeline([
```

```
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Define hyperparameters to tune
param_grid = {
    'preprocessor__feature_selection__k': [5,10, 15]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(pipeline,
                           param_grid,
                           cv=5,
                           scoring='neg_mean_absolute_error')

grid_search.fit(X, y)

# Get the best model
best_estimator = grid_search.best_estimator_

# Make predictions
y_pred = best_estimator.predict(X_test)

# Calculate MAE
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)

# print best parameters
print(grid_search.best_params_)

#source: Lectures/Lecture_15_cross_val_hyperpar_optim/scripts/
 ↪grid_random_search.ipynb
```

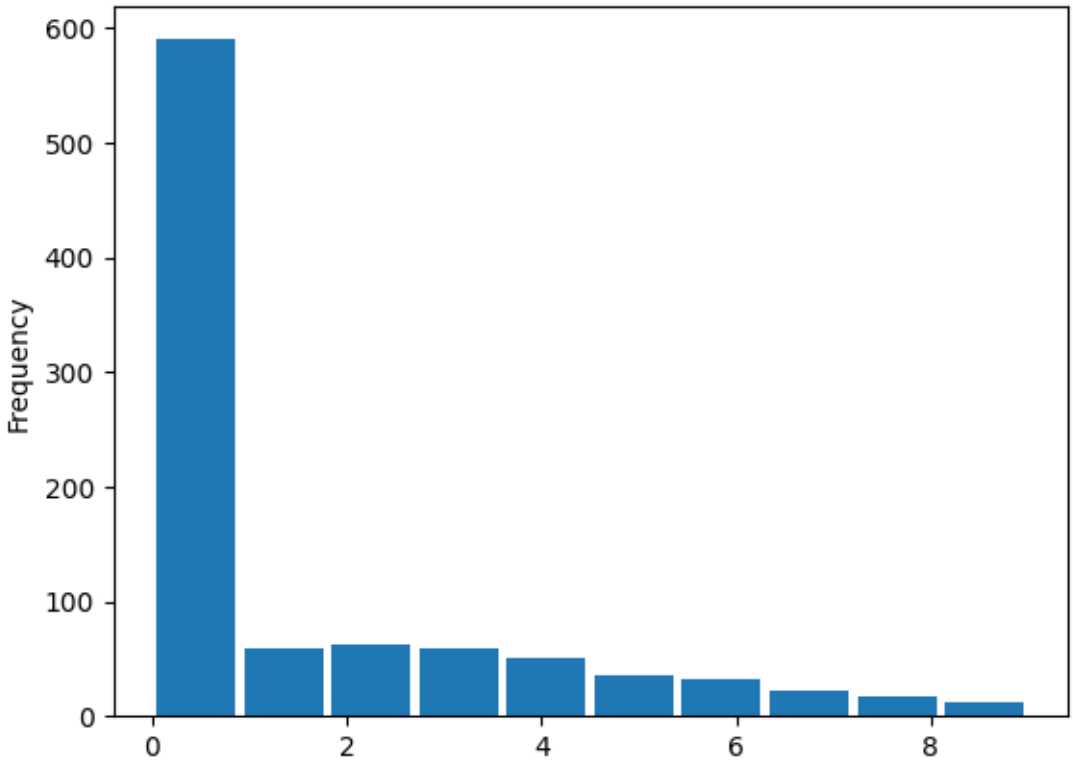
```
Mean Absolute Error (MAE): 60356.98263683694
{'preprocessor__feature_selection__k': 15}
```

[ ]:
```
# Kaggle submission
y_pred = best_estimator.predict(test)
df_submission = pd.DataFrame({'Diagnosis': y_pred})
df_submission.reset_index(level=0, inplace=True)

df_submission.columns = ['index', 'Diagnosis']
df_submission.to_csv('submission_2.csv', index=False)
```

3 B - Multi-class Classification analysis with an esembler classfier

[ ]:
```
# Binning the target variable and creating a histogram
num_bins = 10
cleaned.loc[:, 'binned_target'] = pd.cut(cleaned['Scoville Heat Units (SHU)'],␣
 ↪bins=num_bins, labels=False)

# Plot class histogram
cleaned['binned_target'].plot.hist(rwidth=0.9) # rwidth is the width of the␣
 ↪bars in the histogram plot

# source: 04_binning.ipynb
```

[ ]:
```
<Axes: ylabel='Frequency'>
```



[ ]:
```
# Splitting the data into features and target variable
X = cleaned.drop(columns=['binned_target'])
y = cleaned['binned_target']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=42)
```

```
preprocessor = Pipeline([
    ('scaler', StandardScaler()),
    ('feature_selection', SelectKBest())
])

# Creating a pipeline for preprocessing and XGBoost classifier
xgb_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('xgb', XGBClassifier(objective='multi:softmax', random_state=42))  # multi:
 ↪softmax for multiclass classification
])

# Hyperparameter search grid
param_grid = {
    'xgb__max_depth': [3, 5, 7, 9],
    'xgb__learning_rate': np.linspace(0.01, 0.5, 10),
    'xgb__subsample': [0.5, 0.75, 1],
    'xgb__n_estimators': [100, 200, 300]
}

# Hyperparameter tuning using RandomizedSearchCV
cv = 5  # Number of cross-validation folds
rscv = RandomizedSearchCV(estimator=xgb_pipeline,
                          param_distributions=param_grid,
                          cv=cv, n_iter=10,
                          scoring='accuracy',
                          random_state=42)

# Training the model
rscv.fit(X, y)

# Making predictions
y_pred = rscv.predict(X_test)

# Calculating accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# print best parameters
print(rscv.best_params_)

# source: Lectures/Lecture_17_18_Ensemble/XGboost.ipynb
```

```
Accuracy: 1.0
{'xgb__subsample': 0.5, 'xgb__n_estimators': 300, 'xgb__max_depth': 9,
'xgb__learning_rate': 0.3911111111111111}
```

note: by increasing the number og bins, the accuracy get lower. this was testeted with 10,15,20 and 25 bins

10 bins: Accuracy: 1,0
15 bins: Accuracy: 0.992
20 bins: Accuracy: 0.979
25 bins: Accuracy: 0.976

the accuracy beeing so high, might indicate there is overfitting