Assignment No: 7

Name: Nivas Marudhavanan

UTA ID: 1002197281

INSTRUCTIONS

Every learner should submit his/her own homework solutions. However, you are allowed to discuss the homework with each other– but everyone must submit his/her own solution; you may not copy someone else's solution.

The homework consists of two parts:

1. Data from our life
2. Classification

Follow the prompts in the attached jupyter notebook. We are using a clean and modified version of the auto imports dataset (Description of the original dataset is in the cell bellow). Download the df2.csv file from Canvas and put it in your working directory. Don't forget to add libraries to use in your analysis. You can use the code as a guide that was provided in the class.

Add markdown cells to your analysis to include your solutions, comments, answers. Add as many cells as you need, for easy readability comment when possible.

Submission: Run all your code cells and export the file as HTML. Submit a zip of your .ipynb file and HTML file. Add your name and UTA ID in the markdown cell above.

Good luck!

Title: 1985 Auto Imports Database

Relevant Information: -- Description This data set consists of three types of entities: (a) the specification of an auto in terms of various characteristics, (b) its assigned insurance risk rating, (c) its normalized losses in use as compared to other cars. The second rating corresponds to the degree to which the auto is more risky than its price indicates. Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. Actuarians call this process "symboling". A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.

```
  The third factor is the relative average loss payment per insured
  vehicle year.  This value is normalized for all autos within a
  particular size classification (two-door small, station wagons,
  sports/speciality, etc...), and represents the average loss per
car
  per year.
```

-- Note: Several of the attributes in the database could be used as a "class" attribute.

1. Number of Instances: 205

2. Number of Attributes: 26 total -- 15 continuous -- 1 integer -- 10 nominal

3. Attribute Information:
   Attribute: Attribute Range: ------------------ ---
   ----
4. symboling: -3, -2, -1, 0, 1, 2, 3.
5. normalized-losses: continuous from 65 to 256.
6. make: alfa-romero, audi, bmw, chevrolet, dodge, honda,isuzu, jaguar, mazda, mercedes-benz, mercury, mitsubishi, nissan, peugot, plymouth, porsche, renault, saab, subaru, toyota,volkswagen, volvo
7. fuel-type: diesel, gas.
8. aspiration: std, turbo.
9. num-of-doors: four, two.
10. body-style: hardtop, wagon, sedan, hatchback, convertible.
11. drive-wheels: 4wd, fwd, rwd.
12. engine-location: front, rear.
13. wheel-base: continuous from 86.6 120.9.
14. length: continuous from 141.1 to 208.1.
15. width: continuous from 60.3 to 72.3.
16. height: continuous from 47.8 to 59.8.
17. curb-weight: continuous from 1488 to 4066.
18. engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
19. num-of-cylinders: eight, five, four, six, three, twelve, two.
20. engine-size: continuous from 61 to 326.
21. fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
22. bore: continuous from 2.54 to 3.94.
23. stroke: continuous from 2.07 to 4.17.
24. compression-ratio: continuous from 7 to 23.
25. horsepower: continuous from 48 to 288.
26. peak-rpm: continuous from 4150 to 6600.
27. city-mpg: continuous from 13 to 49.
28. highway-mpg: continuous from 16 to 54.
29. price: continuous from 5118 to 45400.
30. Missing Attribute Values: (denoted by "?")

1. Data from our lives:

Describe a situation or problem from your job, everyday life, current events, etc., for which a classification would be appropriate.

Title: Quiz question deduction for question difficulty classification.

Quiz questions assess learning and offer a variety of formats.

Past student performance data on these questions is highly valuable for accurately classifying difficulty.

Model learns: Question content (keywords, structure, type) and corresponding difficulty levels.

Prediction: New unseen questions are analyzed, and difficulty is predicted based on learned patterns.

Student Benefits:

Targeted practice on challenging questions. Improved learning efficiency and comprehension. Personalized learning paths based on difficulty.

Independent Variables (Features):

Question Type : (Essay,Short Answers, MCQ) Students Performance

Dependent Variable:

Question Difficulty: (Easy,Difficulty,Medium) Cateogry

2. Preprocessing

```
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import learning_curve
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import accuracy_score, precision_score,
        recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import RobustScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import r2_score
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import svm
from sklearn.svm import SVC
```

In our class we covered multiple classification methods. In this part of the home work you can compare them

Use the dataset 'df.csv' from Canvas. Follow the prompts to complete the homework.

```
df2 =pd.read_csv('df2.csv')
```

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   fuel_type    195 non-null    object
 1   wheel_base   195 non-null    float64
 2   length       195 non-null    float64
 3   width        195 non-null    float64
 4   heights      195 non-null    float64
 5   curb_weight  195 non-null    int64
 6   engine_size  195 non-null    int64
 7   bore         195 non-null    float64
 8   stroke       195 non-null    float64
 9   compression  195 non-null    float64
 10  horse_power  195 non-null    int64
 11  peak_rpm     195 non-null    int64
 12  city_mpg     195 non-null    int64
 13  highway_mpg  195 non-null    int64
 14  price        195 non-null    int64
dtypes: float64(7), int64(7), object(1)
memory usage: 23.0+ KB
```

```
df2.shape
```

```
(195, 15)
```

2.1 Replace ['gas', 'diesel'] string values to [0, 1]

```
df2['fuel_type'].unique()
```

```
array(['gas', 'diesel'], dtype=object)
```

```
data = df2.copy()
category_mapping = {'gas': 0, 'diesel': 1}
data['fuel_type'] = data['fuel_type'].map(category_mapping)
data.head()
```

	fuel_type	wheel_base	length	width	heights	curb_weight
0	0	88.6	168.8	64.1	48.8	2548
1	0	88.6	168.8	64.1	48.8	2548
2	0	94.5	171.2	65.5	52.4	2823
3	0	99.8	176.6	66.2	54.3	2337
4	0	99.4	176.6	66.4	54.3	2824

```
data.tail()
```

	fuel_type	wheel_base	length	width	heights	curb_weight
190	0	109.1	188.8	68.9	55.5	2952

191	0	109.1	188.8	68.8	55.5	3049
192	0	109.1	188.8	68.9	55.5	3012
193	1	109.1	188.8	68.9	55.5	3217
194	0	109.1	188.8	68.9	55.5	3062

2.2 : Define your X and y: your dependent variable is fuel_type, the rest of the variables are your independent variables

```
X = data.drop(columns=['fuel_type'])
y = data['fuel_type']
```

2.3 Split your data into training and testing set. Use test_size=0.3, random_state=746 !

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.3, random_state=746)
```

3. Classification

3.1 Use Logistic regression to classify your data. Print/report your confusion matrix, classification report and AUC. What do you notice?

```
log_regression = LogisticRegression()

log_regression.fit(X_train, y_train)

y_prediction = log_regression.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_prediction)
cnf_matrix
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/sit
packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver
options:
    https://scikit-
learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
array([[50,  0],
       [ 0,  9]])
```

Confusionn Matrix suggest Model has predict all True Positive and True Negative Values correctly

L1 Regularization

```
model = LogisticRegression(solver='liblinear', penalty='l1')
model.fit(X_train, y_train)
accuracy = model.score(X_test, y_test)
print(f'Test accuracy: {accuracy:.2f}')
```

```
Test accuracy: 1.00
```

```
class_names=[0, 1]
figure, axis = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu"
        ,fmt='g')
axis.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 427.9555555555555, 'Predicted label')
```

```
target_names = ['gas', 'diesel']
print(classification_report(y_test, y_prediction,
        target_names=target_names))
```

```
              precision    recall  f1-score   support

         gas       1.00      1.00      1.00        50
      diesel       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

CROSS VALIDATION

```
cv_scores = cross_val_score(model, X_train, y_train, cv=5)
test_accuracy = model.score(X_test, y_test)
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
print("Test accuracy:", test_accuracy)
```

```
Cross-validation scores: [1. 1. 1. 1. 1.]
Mean cross-validation score: 1.0
Test accuracy: 1.0
```

```
y_pred_proba = log_regression.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

AUC = 1 implies that the model correctly identifies all positive instances (True Positives) without any False Positives.

3.2 Use Naive Bayes to classify your data. Print/report your confusion matrix, classification report and AUC. What do you notice?

```
scaler = RobustScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred = gnb.predict(X_test)
y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0])
```

Test Set Accuracy

```
print('Model accuracy score: {0:0.4f}'.
        format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 1.0000
```

Train Set Accuracy

```
y_pred_train = gnb.predict(X_train)
```

```
y_pred_train
```

```
array([1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0,
       0, 0, 0, 0])
```

```
print('Training-set accuracy score: {0:0.4f}'.
        format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 1.0000
```

```
cnf_matrix_gb = metrics.confusion_matrix(y_test, y_prediction)
cnf_matrix_gb
predictions = model.predict(X_test)
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/sit
packages/sklearn/base.py:493: UserWarning: X does not have valid
feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_gb, annot=True, fmt='d', cmap='Blues',
        cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

LEARNING CURVE

```
def plot_learning_curve(estimator, X, y,
        train_sizes=np.linspace(0.1, 1.0, 5), cv=5):
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, train_sizes=train_sizes, cv=cv)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.figure(figsize=(10, 6))
    plt.fill_between(train_sizes, train_scores_mean -
        train_scores_std,
                     train_scores_mean + train_scores_std,
        alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean -
        test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
        color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.xlabel("Training examples")
    plt.ylabel("Score")
    plt.title("Learning Curve")
    plt.legend(loc="best")
    plt.grid(True)
    plt.show()

plot_learning_curve(gnb, X_train, y_train)
```

The learning curve suggests that the model is capable of learning from the data. However, the slight gap between the training and cross-validation scores indicates potential overfitting, especially with larger training sets

Cross Validation

```
cv_scores = cross_val_score(gnb, X_train, y_train, cv=5)
test_accuracy = gnb.score(X_test, y_test)
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
print("Test accuracy:", test_accuracy)
```

```
Cross-validation scores: [1. 1. 1. 1. 1.]
Mean cross-validation score: 1.0
Test accuracy: 1.0
```

```
target_names = ['gas', 'diesel']
print(classification_report(y_test, y_prediction,
        target_names=target_names))
```

```
              precision    recall  f1-score   support

         gas       1.00      1.00      1.00        50
      diesel       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

```
y_pred_probablity = gnb.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_probablity)
auc = metrics.roc_auc_score(y_test, y_pred_probablity)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

AUC = 1 and training dataset accuracy = 0.998 and test dataset accuracy =1 suggest model performs very well on both training and test dataset

3.3 Use KNN to classify your data. First find the optimal k and than run you classification. Print/report your confusion matrix, classification report and AUC. What do you notice?

Find K

```
k_values = [i for i in range (1,31)]
scores = []
scaler = StandardScaler()
X = scaler.fit_transform(X)
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    score = cross_val_score(knn, X, y, cv=5)
    scores.append(np.mean(score))
sns.lineplot(x = k_values, y = scores, marker = 'o')
plt.xlabel("K Values")
plt.ylabel("Accuracy Score")
```

```
Text(0, 0.5, 'Accuracy Score')
```

OPTIMAL VALUE OF N SHOULD BE IN THE RANGE OF 1 TO 10

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=13)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9152542372881356
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

CROSS VALIDATION

```
cv_scores = cross_val_score(knn, X_train, y_train, cv=5)
test_accuracy = knn.score(X_test, y_test)
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
print("Test accuracy:", test_accuracy)
```

```
Cross-validation scores: [1. 1. 1. 1. 1.]
Mean cross-validation score: 1.0
Test accuracy: 1.0
```

```
cnf_matrix_knn = metrics.confusion_matrix(y_test, y_prediction)
cnf_matrix_knn
```

```
array([[50,  0],
       [ 0,  9]])
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_knn, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
target_names = ['gas', 'diesel']
print(classification_report(y_test, y_prediction,
        target_names=target_names))
```

```
              precision    recall  f1-score   support

         gas       1.00      1.00      1.00        50
      diesel       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

Split the data further into training, validation to evaluate its final performance on the unseen test set

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
        test_size=0.3, random_state=42)
param_grid = {
    'n_neighbors': [3],
    'weights': ['uniform', 'distance']
}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid,
        cv=cv, scoring='accuracy')
```

```
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Test Accuracy: 1.0
```

```
y_pred_proba = knn.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

Based on these results, we can infer that your KNN model has learned excellent decision boundaries to separate the "gas" and "diesel" data points in our training set.

3.4 Choose one: SVM or Random Forest to classify your data. Print/report your confusion matrix, classification report and AUC

```
clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

```
cnf_matrix_svc = metrics.confusion_matrix(y_test, y_prediction)
cnf_matrix_svc
```

```
array([[50,  0],
       [ 0,  9]])
```

```
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, y_pred)
print("AUC:", auc)
```

```
AUC: 1.0
```

```
target_names = ['gas', 'diesel']
print(classification_report(y_test, y_prediction,
        target_names=target_names))
```

```
              precision    recall  f1-score   support

         gas       1.00      1.00      1.00        50
      diesel       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

CROSS VALIDATION

```
cv_scores = cross_val_score(clf, X_train, y_train, cv=5)
test_accuracy = clf.score(X_test, y_test)
```

```
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
print("Test accuracy:", test_accuracy)
```

```
Cross-validation scores: [1. 1. 1. 1. 1.]
Mean cross-validation score: 1.0
Test accuracy: 1.0
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/sit
packages/sklearn/model_selection/_split.py:737: UserWarning: The
least populated class in y has only 2 members, which is less than
n_splits=5.
  warnings.warn(
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_svc, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
calibrated_svm = CalibratedClassifierCV(clf)
calibrated_svm.fit(X_train, y_train)
y_pred_proba = calibrated_svm.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr, tpr, label='SVM (linear kernel), auc='+str(auc))
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve')
plt.legend(loc=4)
plt.grid(True)
plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/sit
packages/sklearn/model_selection/_split.py:737: UserWarning: The
least populated class in y has only 2 members, which is less than
n_splits=5.
  warnings.warn(
```

Results suggest that SVM model has learned effective decision boundaries for the specific data it was trained on.

3.5 Compare your results and comment on your findings. Which one(s) did the best job? What could have been the problem with the ones that did not work? etc.

All the models which are compared (Logistic Regression, Naive Bayes, KNN, and SVM) achieved perfect accuracy (1.0) on the test set, with AUC (Area Under the ROC Curve) of 1.0 as well and cross validation also gives accuracy 1 except knn. This suggests that all models performed exceptionally well on this specific dataset for the binary classification task of distinguishing between "gas" and "diesel" classes.

4. Bonus question (15 extra points)

Try to fix the inbalanced nature of the data with a tool from the lecture. Run one of the classification methods (preferable one that "failed" before) and see if you get better results.

```
class_counts = data['fuel_type'].value_counts()
print(class_counts)
X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.3, random_state=742)
minority_class = class_counts.idxmin()
sm = SMOTE(sampling_strategy=minority_class, k_neighbors=10)
X_train_resampled, y_train_resampled = sm.fit_resample(X_train,
        y_train)
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance']
}
knn = KNeighborsClassifier()
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid,
        cv=5, scoring='accuracy')
grid_search.fit(X_train_resampled, y_train_resampled)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
data['fuel_type_factorized'], _ = pd.factorize(data['fuel_type'])
plt.bar(data['fuel_type_factorized'].unique(),
        data['fuel_type'].value_counts())
plt.xlabel("Fuel Type")
plt.ylabel("Count")
plt.title("Distribution of Fuel Types (0 or 1)")
plt.xticks([0, 1])
plt.show()
print("Test Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)

print("Classification Report:\n", classification_report(y_test,
        y_pred))
```

```
0    175
1     20
Name: fuel_type, dtype: int64
```

```
Test Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        54
           1       1.00      1.00      1.00         5

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

```
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
svm = SVC(kernel='rbf', gamma=1, C=1)
svm.fit(X_train_smote, y_train_smote)
y_pred = svm.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
        y_pred))
```

```
Accuracy: 0.9322033898305084
Classification Report:
               precision    recall  f1-score   support

           0       0.93      1.00      0.96        54
           1       1.00      0.20      0.33         5

    accuracy                           0.93        59
   macro avg       0.97      0.60      0.65        59
weighted avg       0.94      0.93      0.91        59
```

The introduction of SMOTE to address class imbalance while it performs well on the majority class, it struggles to accurately identify the minority class.