Algorithms Used for Classification

1. CART (Classification and Regression Trees)
2. Gaussian Naive Bayes / Naive Bayes
3. Gradient Boosting Machines (AdaBoost)
4. K-Nearest Neighbors (K-NN)
5. Logistic Regression
6. Multi-Layer Perceptron (MLP)
7. Perceptron
8. Random Forest
9. Support Vector Machines (SVM)

1. CART (Classification and Regression Trees) - DecisionTree Classifier

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size and random seed for reproducibility
test_size = 0.20
random_seed = 50  # You can change this value

# Split the dataset into training and testing sets
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=random_seed)

# Initialize and train the Decision Tree Classifier with
hyperparameters
max_depth = 5  # You can adjust this value
min_samples_split = 2  # You can adjust this value
min_samples_leaf = 1  # You can adjust this value

model = DecisionTreeClassifier(
    max_depth=max_depth,
    min_samples_split=min_samples_split,
    min_samples_leaf=min_samples_leaf,
    random_state=random_seed
)

model.fit(X_train, Y_train)

# Evaluate the accuracy
accuracy = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (accuracy * 100.0))

Accuracy: 93.353%
```

2. Gaussian Naive Bayes

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)
```

```
# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size
test_size = 0.20  # Hyperparameter: Fraction of the dataset to use for
testing
seed = 7

# Split the dataset into test and train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)

# Create a Gaussian Naive Bayes classifier
model = GaussianNB(priors=None, var_smoothing=1e-9)
# Hyperparameters:
# - priors: You can specify class prior probabilities if you have
prior knowledge.
# - var_smoothing: A smoothing parameter for avoiding zero variances.

# Train the model on the training data
model.fit(X_train, Y_train)

# Evaluate the accuracy
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result * 100.0))

Accuracy: 92.775%
```

3. Gradient Boosting Machines (AdaBoost)

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
```

```
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size
test_size = 0.20  # Hyperparameter: Fraction of the dataset to use for
testing
seed = 7

# Split the dataset into test and train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)

# Create an AdaBoost classifier
model = AdaBoostClassifier(n_estimators=50, random_state=seed)
# Hyperparameters:
# - n_estimators: The number of weak classifiers (base estimators) to
train. You can adjust this to control the complexity of the ensemble.
# - random_state: The random seed for reproducibility. You can set
this to a specific value if you want consistent results.

# Train the model on the training data
model.fit(X_train, Y_train)

# Evaluate the accuracy
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result * 100.0))

Accuracy: 95.087%
```

4. K-Nearest Neighbors (K-NN)

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
```

```
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size
test_size = 0.20  # Hyperparameter: Fraction of the dataset to use for
testing
seed = 7

# Split the dataset into test and train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)

# Create a K-Nearest Neighbors (K-NN) classifier
model = KNeighborsClassifier(n_neighbors=5, weights='uniform',
algorithm='auto')
# Hyperparameters:
# - n_neighbors: The number of nearest neighbors to consider when
making predictions. You can adjust this to control the model's
sensitivity to local patterns.
# - weights: Determines how the neighbors' contributions are weighted
(e.g., 'uniform' or 'distance'). You can choose the appropriate
weighting strategy.
# - algorithm: The algorithm used to compute the nearest neighbors
('auto', 'ball_tree', 'kd_tree', or 'brute'). You can choose the most
suitable algorithm based on your data size and structure.

# Train the model on the training data
model.fit(X_train, Y_train)

# Evaluate the accuracy
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result * 100.0))
```

```
Accuracy: 98.844%
```

5. Logistic Regression

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size
test_size = 0.20  # Hyperparameter: Fraction of the dataset to use for
testing
seed = 7

# Split the dataset into test and train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)

# Create a Logistic Regression model
model = LogisticRegression(max_iter=200, solver='lbfgs', C=1.0)
# Hyperparameters:
# - max_iter: The maximum number of iterations for the solver to
converge. You can adjust this if the model does not converge.
# - solver: The algorithm to use for optimization ('lbfgs',
'liblinear', etc.). Choose an appropriate solver for your data and
problem.
# - C: Inverse of regularization strength. Smaller values increase
regularization. You can adjust this to control the trade-off between
```

```
fitting the data and preventing overfitting.

# Train the model on the training data
model.fit(X_train, Y_train)

# Evaluate the accuracy
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result * 100.0))

Accuracy: 89.017%
```

6. Multi-Layer Perceptron (MLP)

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size
test_size = 0.20  # Hyperparameter: Fraction of the dataset to use for
testing
seed = 7

# Split the dataset into test and train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)
```

```
# Create an MLP-based model
model = MLPClassifier(hidden_layer_sizes=(65, 32), activation='relu',
solver='adam', max_iter=1000, random_state=seed)
# Hyperparameters:
# - hidden_layer_sizes: The number of neurons in each hidden layer.
You can customize the architecture by adjusting this parameter.
# - activation: The activation function used in the hidden layers
('relu', 'tanh', etc.). Choose the appropriate one for your problem.
# - solver: The algorithm for weight optimization ('adam', 'lbfgs',
etc.). Select the one that works best for your data.
# - max_iter: The maximum number of iterations for the solver to
converge. You can adjust this if the model does not converge.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.

# Train the model
model.fit(X_train, Y_train)

# Evaluate the accuracy
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result * 100.0))

Accuracy: 99.422%
```

7. Perceptron

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
```

```
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size
test_size = 0.20  # Hyperparameter: Fraction of the dataset to use for
testing
seed = 7

# Split the dataset into test and train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)

# Create a Perceptron classifier
model = Perceptron(max_iter=200, random_state=seed, eta0=1.0, tol=1e-
3)
# Hyperparameters:
# - max_iter: The maximum number of iterations for the solver to
converge. You can adjust this if the model does not converge.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.
# - eta0: The initial learning rate. You can control the step size for
weight updates by adjusting this.
# - tol: The tolerance for stopping criterion. The model will stop
training when the change in the average loss is smaller than this
value.

# Train the model
model.fit(X_train, Y_train)

# Evaluate the accuracy
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result * 100.0))
```

```
Accuracy: 83.526%
```

8. Random Forest

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
```

```
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size
test_size = 0.20  # Hyperparameter: Fraction of the dataset to use for
testing
seed = 7

# Split the dataset into test and train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)

# Create a Random Forest classifier
rfmodel = RandomForestClassifier(n_estimators=100, random_state=seed,
max_depth=None, min_samples_split=2, min_samples_leaf=1)
# Hyperparameters:
# - n_estimators: The number of decision trees in the random forest.
Adjust this to control the ensemble size.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.
# - max_depth: The maximum depth of the decision trees. You can limit
tree depth to prevent overfitting.
# - min_samples_split: The minimum number of samples required to split
a node. Adjust this to control tree node splitting.
# - min_samples_leaf: The minimum number of samples required in a leaf
node. You can adjust this to control tree leaf size.

# Train the model
rfmodel.fit(X_train, Y_train)

# Evaluate the accuracy
result = rfmodel.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result * 100.0))
```

```
Accuracy: 99.422%
```

9. Support Vector Machines (SVM)

- Sampling Technique - Train/Test Split (80:20)
- Classification Metrics - Accuracy

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
Y = dataframe['Classification']

# Set the test size
test_size = 0.20  # Hyperparameter: Fraction of the dataset to use for
testing
seed = 7

# Split the dataset into test and train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)

# Create an SVM classifier
model = SVC(kernel='linear', C=1.0, random_state=seed)
# Hyperparameters:
# - kernel: The type of kernel to use ('linear', 'poly', 'rbf', etc.).
Choose the appropriate kernel for your problem.
# - C: The regularization parameter. Smaller values increase
regularization. You can adjust this to control the trade-off between
```

```
fitting the data and preventing overfitting.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.

# Train the model
model.fit(X_train, Y_train)

# Evaluate the accuracy
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result * 100.0))

Accuracy: 89.595%
```

Algorithms Used for Regression

1. CART (Classification and Regression Trees)
2. Elastic Net
3. Gradient Boosting Machines (AdaBoost)
4. K-Nearest Neighbors (K-NN)
5. Lasso and Ridge Regression
6. Linear Regression
7. Multi-Layer Perceptron (MLP)
8. Random Forest
9. Support Vector Machines (SVM)

**When comparing models, a lower MAE is generally better.

1. CART (Classification and Regression Trees) - DecisionTree Regressor

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)
```

```
# Split the dataset into a 10-fold cross-validation
kfold = KFold(n_splits=10, random_state=None)

# Train the data on a Decision Tree Regressor
model = DecisionTreeRegressor(max_depth=None, min_samples_split=2,
min_samples_leaf=1, random_state=None)
# Hyperparameters:
# - max_depth: The maximum depth of the decision tree. You can limit
tree depth to prevent overfitting.
# - min_samples_split: The minimum number of samples required to split
an internal node. Adjust this to control node splitting.
# - min_samples_leaf: The minimum number of samples required in a leaf
node. You can adjust this to control leaf size.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.

# Calculate the mean absolute error
scoring = 'neg_mean_absolute_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MAE: %.3f (%.3f)" % (-results.mean(), results.std()))

MAE: 191257.973 (37669.408)
```

2. Elastic Net

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import ElasticNet

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Split the dataset into a 10-fold cross-validation
kfold = KFold(n_splits=10, random_state=None)

# Train the data on an Elastic Net model
model = ElasticNet(alpha=1.0, l1_ratio=0.5, max_iter=1000,
```

```
random_state=None)
# Hyperparameters:
# - alpha: The regularization parameter that controls the balance
between L1 (Lasso) and L2 (Ridge) penalties. Adjust this to control
the regularization strength.
# - l1_ratio: The mixing parameter for L1 and L2 penalties. A value of
0 corresponds to L2, 1 to L1, and values in between to combinations.
# - max_iter: The maximum number of iterations for the solver to
converge. You can adjust this if the model does not converge.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.

# Calculate the mean absolute error
scoring = 'neg_mean_absolute_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MAE: %.3f (%.3f)" % (-results.mean(), results.std()))

MAE: 220326.393 (35703.212)
```

3. Gradient Boosting Machines (AdaBoost)

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostRegressor

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Split the dataset into a 10-fold cross-validation
kfold = KFold(n_splits=10, random_state=None)

# Train the data on an AdaBoost Regressor
ada_model = AdaBoostRegressor(n_estimators=50, learning_rate=1.0,
random_state=None)
# Hyperparameters:
# - n_estimators: The number of weak regressors to combine in the
ensemble. You can adjust this to control the complexity of the
```

```
ensemble.
# - learning_rate: The contribution of each weak regressor to the
final prediction. You can adjust this to control the impact of
individual estimators.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.

# Calculate the mean absolute error with AdaBoost
scoring = 'neg_mean_absolute_error'
ada_results = cross_val_score(ada_model, X, Y, cv=kfold,
scoring=scoring)
print("AdaBoost MAE: %.3f (%.3f)" % (-ada_results.mean(),
ada_results.std()))

AdaBoost MAE: 348184.901 (29348.249)
```

4. K-Nearest Neighbors (K-NN)

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Split the dataset into a 10-fold cross-validation
kfold = KFold(n_splits=10, random_state=None)

# Train the data on a K-Nearest Neighbors Regressor
knn_model = KNeighborsRegressor(n_neighbors=5, weights='uniform',
algorithm='auto')
# Hyperparameters:
# - n_neighbors: The number of nearest neighbors to consider when
making predictions. You can adjust this to control the model's
sensitivity to local patterns.
# - weights: Determines how the neighbors' contributions are weighted
(e.g., 'uniform' or 'distance'). You can choose the appropriate
```

```
weighting strategy.
# - algorithm: The algorithm used to compute the nearest neighbors
('auto', 'ball_tree', 'kd_tree', or 'brute'). You can choose the most
suitable algorithm based on your data size and structure.

# Calculate the mean absolute error with K-NN
scoring = 'neg_mean_absolute_error'
knn_results = cross_val_score(knn_model, X, Y, cv=kfold,
scoring=scoring)
print("K-NN MAE: %.3f (%.3f)" % (-knn_results.mean(),
knn_results.std()))
```

```
K-NN MAE: 248792.473 (44887.474)
```

5. Lasso and Ridge Regression

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Split the dataset into a 10-fold cross-validation
kfold = KFold(n_splits=10, random_state=None)

# Train the data on a Lasso Regression model
lasso_model = Lasso(alpha=1.0, max_iter=1000, random_state=None)
# Hyperparameters for Lasso:
# - alpha: The regularization parameter that controls the strength of
L1 regularization. Adjust this to control the level of sparsity in the
model.
# - max_iter: The maximum number of iterations for the solver to
converge. You can adjust this if the model does not converge.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.
```

```
# Calculate the mean absolute error with Lasso
scoring = 'neg_mean_absolute_error'
lasso_results = cross_val_score(lasso_model, X, Y, cv=kfold,
scoring=scoring)
print("Lasso MAE: %.3f (%.3f)" % (-lasso_results.mean(),
lasso_results.std()))

# Train the data on a Ridge Regression model
ridge_model = Ridge(alpha=1.0, max_iter=1000, random_state=None)
# Hyperparameters for Ridge:
# - alpha: The regularization parameter that controls the strength of
L2 regularization. Adjust this to control the strength of
regularization.
# - max_iter: The maximum number of iterations for the solver to
converge. You can adjust this if the model does not converge.
# - random_state: The random seed for reproducibility. Set this to a
specific value for consistent results.

# Calculate the mean absolute error with Ridge
ridge_results = cross_val_score(ridge_model, X, Y, cv=kfold,
scoring=scoring)
print("Ridge MAE: %.3f (%.3f)" % (-ridge_results.mean(),
ridge_results.std()))
```

```
Lasso MAE: 156500.518 (34501.360)
Ridge MAE: 163854.322 (31709.921)
```

6. Linear Regression

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)
# Split the dataset into a 10-fold cross-validation
```

```
kfold = KFold(n_splits=10, random_state=None)

# Train the data on a Linear Regression model
model = LinearRegression()

# Calculate the mean absolute error
scoring = 'neg_mean_absolute_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MAE: %.3f (%.3f)" % (-results.mean(), results.std()))
```

```
MAE: 156496.403 (34504.130)
```

7. Multi-Layer Perceptron (MLP)

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPRegressor

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Split the dataset into a 10-fold cross-validation
kfold = KFold(n_splits=10, random_state=None)

# Train the data on an MLP Regressor with specified hyperparameters
mlp_model = MLPRegressor(
    hidden_layer_sizes=(100, 50),  # Hyperparameter: Adjust the
architecture as needed, specifying the number and size of hidden
layers.
    activation='relu',             # Hyperparameter: Choose an
appropriate activation function ('identity', 'logistic', 'tanh',
'relu', etc.).
    solver='adam',                # Hyperparameter: Choose an
optimization algorithm ('adam', 'lbfgs', 'sgd', etc.).
    learning_rate='constant',     # Hyperparameter: Choose a learning
rate schedule ('constant', 'invscaling', 'adaptive').
    max_iter=1000,                # Hyperparameter: Adjust the maximum
```

```
number of iterations for training.
    random_state=50               # Hyperparameter: Set a random seed
for reproducibility.
)

# Calculate the mean absolute error with MLP
scoring = 'neg_mean_absolute_error'
mlp_results = cross_val_score(mlp_model, X, Y, cv=kfold,
scoring=scoring)
print("MLP MAE: %.3f (%.3f)" % (-mlp_results.mean(),
mlp_results.std()))
```

```
MLP MAE: 254932.805 (39260.489)
```

8. Random Forest

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Split the dataset into a 10-fold cross-validation
kfold = KFold(n_splits=10, random_state=None)

# Train the data on a Random Forest Regressor with specified
hyperparameters
rf_model = RandomForestRegressor(
    n_estimators=100,       # Hyperparameter: The number of trees in
the forest. You can adjust this for ensemble size.
    max_depth=None,         # Hyperparameter: The maximum depth of
each tree. Adjust to control tree depth.
    min_samples_split=2,    # Hyperparameter: The minimum number of
samples required to split an internal node. Adjust to control node
splitting.
    min_samples_leaf=1,     # Hyperparameter: The minimum number of
```

```
samples required in a leaf node. Adjust to control leaf size.
    random_state=42         # Hyperparameter: Set a random seed for
reproducibility.
)

# Calculate the mean absolute error with Random Forest
scoring = 'neg_mean_absolute_error'
rf_results = cross_val_score(rf_model, X, Y, cv=kfold,
scoring=scoring)
print("Random Forest MAE: %.3f (%.3f)" % (-rf_results.mean(),
rf_results.std()))

Random Forest MAE: 167836.651 (30518.495)
```

9. Support Vector Machines (SVM)

- Sampling Technique = K-fold Cross Validation (k=10)
- Classification Metrics = MAE

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVR

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Split the dataset into a 10-fold cross-validation
kfold = KFold(n_splits=10, random_state=None)

# Train the data on a Support Vector Regressor (SVM) with specified
hyperparameters
svm_model = SVR(
    kernel='rbf',           # Hyperparameter: The kernel function to
use ('linear', 'poly', 'rbf', etc.).
    C=1.0,                  # Hyperparameter: The regularization
parameter. Adjust this to control the trade-off between margin width
and error.
    epsilon=0.1,            # Hyperparameter: The epsilon-tube within
which no penalty is associated with errors.
)
```

```python
# Calculate the mean absolute error with SVM
scoring = 'neg_mean_absolute_error'
svm_results = cross_val_score(svm_model, X, Y, cv=kfold,
scoring=scoring)
print("SVM MAE: %.3f (%.3f)" % (-svm_results.mean(),
svm_results.std()))
```

```
SVM MAE: 241017.200 (48803.878)
```

```python
import joblib  # Import the joblib library
#Save the model to a file
#model_filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Models/classification_random_forest_model.pkl'
#joblib.dump(rfmodel, model_filename)

import joblib

# Load the saved Random Forest model
model_filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Models/classification_random_forest_model.pkl'
loaded_model = joblib.load(model_filename)

# Define sample input data
#Buying Price    Maintenance Cost Number of Doors  Number of Persons
      Lug_Boot   Safety
sample_input = [['1', '1', '2', '1', '2', '2',]]

# Make predictions using the loaded model
predictions = loaded_model.predict(sample_input)

# Define messages based on the predicted class
if predictions[0] == 1:
    print("Accepted")
else:
    print("Unaccepted")
```

```
Unaccepted
```

```
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:420: UserWarning: X does not have valid
feature names, but RandomForestClassifier was fitted with feature
names
  warnings.warn(
```

Comparing ML Algorithms

Classification

```
#Split and Train Test
#Confusion Matrix and Classification Report

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# features (X) and target (Y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
y = dataframe['Classification']

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

# Initialize classifiers
```

```python
classifiers = {
    "CART": DecisionTreeClassifier(),
    "Naive Bayes": GaussianNB(),
    "AdaBoost": AdaBoostClassifier(),
    "K-NN": KNeighborsClassifier(),
    "Logistic Regression": LogisticRegression(),
    "MLP": MLPClassifier(),
    "Perceptron": Perceptron(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
}

# Iterate through classifiers and evaluate
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    # Evaluate and print confusion matrix and classification report
    print(f"Classifier: {name}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("\n")
```

```
Classifier: CART
Confusion Matrix:
[[234   1]
 [  2 109]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       235
           1       0.99      0.98      0.99       111

    accuracy                           0.99       346
   macro avg       0.99      0.99      0.99       346
weighted avg       0.99      0.99      0.99       346



Classifier: Naive Bayes
Confusion Matrix:
[[220  15]
 [ 19  92]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       235
           1       0.86      0.83      0.84       111
```

```
    accuracy                           0.90       346
   macro avg       0.89      0.88      0.89       346
weighted avg       0.90      0.90      0.90       346



Classifier: AdaBoost
Confusion Matrix:
[[228   7]
 [ 11 100]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       235
           1       0.93      0.90      0.92       111

    accuracy                           0.95       346
   macro avg       0.94      0.94      0.94       346
weighted avg       0.95      0.95      0.95       346



Classifier: K-NN
Confusion Matrix:
[[235   0]
 [  6 105]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       235
           1       1.00      0.95      0.97       111

    accuracy                           0.98       346
   macro avg       0.99      0.97      0.98       346
weighted avg       0.98      0.98      0.98       346



Classifier: Logistic Regression
Confusion Matrix:
[[220  15]
 [ 26  85]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.94      0.91       235
           1       0.85      0.77      0.81       111

    accuracy                           0.88       346
   macro avg       0.87      0.85      0.86       346
```

```
weighted avg       0.88      0.88      0.88       346



C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(

Classifier: MLP
Confusion Matrix:
[[231   4]
 [ 11 100]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       235
           1       0.96      0.90      0.93       111

    accuracy                           0.96       346
   macro avg       0.96      0.94      0.95       346
weighted avg       0.96      0.96      0.96       346



Classifier: Perceptron
Confusion Matrix:
[[190  45]
 [  5 106]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.81      0.88       235
           1       0.70      0.95      0.81       111

    accuracy                           0.86       346
   macro avg       0.84      0.88      0.85       346
weighted avg       0.89      0.86      0.86       346



Classifier: Random Forest
Confusion Matrix:
[[235   0]
 [  2 109]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       235
```

```
           1       1.00      0.98      0.99       111

    accuracy                           0.99       346
   macro avg       1.00      0.99      0.99       346
weighted avg       0.99      0.99      0.99       346



Classifier: SVM
Confusion Matrix:
[[233   2]
 [ 10 101]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       235
           1       0.98      0.91      0.94       111

    accuracy                           0.97       346
   macro avg       0.97      0.95      0.96       346
weighted avg       0.97      0.97      0.96       346




#K-Fold Cross Variation
#Classification Accuracy

# Import necessary libraries
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}
```

```
# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# Define features (X) and target (y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
y = dataframe['Classification']

# Define a list of machine learning algorithms
algorithms = [
    ("CART", DecisionTreeClassifier()),
    ("Naive Bayes", GaussianNB()),
    ("AdaBoost", AdaBoostClassifier()),
    ("K-NN", KNeighborsClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("MLP", MLPClassifier()),
    ("Perceptron", Perceptron()),
    ("Random Forest", RandomForestClassifier()),
    ("SVM", SVC())
]

# Iterate through the algorithms and perform K-fold Cross Validation
for name, model in algorithms:
    scores = cross_val_score(model, X, y, cv=10, scoring='accuracy')
# K-fold Cross Validation with k=10
    print(f"{name}:")
    print(f"Mean Accuracy: {scores.mean()}")
    print(f"Standard Deviation: {scores.std()}\n")
```

```
CART:
Mean Accuracy: 0.8473753192633419
Standard Deviation: 0.11940050981103918

Naive Bayes:
Mean Accuracy: 0.8993816373168437
Standard Deviation: 0.06628282497867485

AdaBoost:
Mean Accuracy: 0.917898911143971
Standard Deviation: 0.06281304813811818

K-NN:
Mean Accuracy: 0.8941457185105526
Standard Deviation: 0.0398902191529915
```

```
Logistic Regression:
Mean Accuracy: 0.8709067078908456
Standard Deviation: 0.06944590627065844


C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(

MLP:
Mean Accuracy: 0.9375352870009408
Standard Deviation: 0.044445170561649996

Perceptron:
Mean Accuracy: 0.8270668100551151
Standard Deviation: 0.1008261198126333

Random Forest:
Mean Accuracy: 0.9063314961688398
Standard Deviation: 0.08069074088344796

SVM:
Mean Accuracy: 0.9288546847694583
Standard Deviation: 0.04578712545956918
```

```python
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import log_loss
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Define your dataset and features (X) and target (y) here
# Load and preprocess your data

# Define resampling technique: Repeated Random Train-Test splits
n_splits = 5  # Number of splits
n_repeats = 10  # Number of repetitions
rkf = RepeatedKFold(n_splits=n_splits, n_repeats=n_repeats,
random_state=42)

# Define classification algorithms (excluding Perceptron)
classifiers = {
    'CART': DecisionTreeClassifier(),
    'Gaussian NB': GaussianNB(),
    'AdaBoost': AdaBoostClassifier(),
    'K-NN': KNeighborsClassifier(),
```

```
    'Logistic Regression': LogisticRegression(),
    'MLP': MLPClassifier(),  # Adjust max_iter if needed
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(probability=True)  # Use probability=True for log loss
}

# Evaluate each classifier using log loss
for name, clf in classifiers.items():
    try:
        logloss_scores = -cross_val_score(clf, X, y, cv=rkf,
scoring='neg_log_loss')
        print(f'{name} Log Loss: {logloss_scores.mean():.4f} (±
{logloss_scores.std():.4f})')
    except Exception as e:
        print(f'{name} Log Loss: N/A (Exception: {str(e)})')
```

```
CART Log Loss: 0.3400 (±0.1841)
Gaussian NB Log Loss: 0.2055 (±0.0157)
AdaBoost Log Loss: 0.4945 (±0.0083)
K-NN Log Loss: 0.1165 (±0.0218)
Logistic Regression Log Loss: 0.2463 (±0.0234)
```

```
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000)
reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000)
reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
MLP Log Loss: 0.0633 (±0.0165)
Random Forest Log Loss: 0.0623 (±0.0070)
SVM Log Loss: 0.0904 (±0.0171)
```

```
#Random Forest Hypertuning

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/car_evaluation_classification.csv'
dataframe = read_csv(filename)
# Custom mapping for each feature
buying_price_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
maintenance_cost_mapping = {'vhigh': 3, 'high': 2, 'med': 1, 'low': 0}
```

```
lug_boot_mapping = {'big': 2, 'med': 1, 'small': 0}
safety_mapping = {'high': 2, 'med': 1, 'low': 0}

# Apply the custom mapping to each column
dataframe['Buying Price'] = dataframe['Buying
Price'].map(buying_price_mapping)
dataframe['Maintenance Cost'] = dataframe['Maintenance
Cost'].map(maintenance_cost_mapping)
dataframe['Lug_Boot'] = dataframe['Lug_Boot'].map(lug_boot_mapping)
dataframe['Safety'] = dataframe['Safety'].map(safety_mapping)

# Define features (X) and target (y)
X = dataframe[['Buying Price', 'Maintenance Cost', 'Number of Doors',
'Number of Persons', 'Lug_Boot', 'Safety']]
y = dataframe['Classification']

# Split the dataset into training and testing sets (80% training, 20%
testing)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define a list of hyperparameters to tune
n_estimators = [50, 100, 200]  # Number of trees in the forest
max_depth = [None, 10, 20]  # Maximum depth of each tree
min_samples_split = [2, 5, 10]  # Minimum number of samples required
to split an internal node

best_accuracy = 0.0
best_model = None

# Perform hyperparameter tuning
for n in n_estimators:
    for depth in max_depth:
        for min_samples in min_samples_split:
            # Create a Random Forest classifier with the current
hyperparameters
            rf = RandomForestClassifier(n_estimators=n,
max_depth=depth, min_samples_split=min_samples, random_state=42)

            # Train the model on the training data
            rf.fit(X_train, y_train)

            # Make predictions on the test data
            y_pred = rf.predict(X_test)

            # Calculate accuracy
            accuracy = accuracy_score(y_test, y_pred)

            # Check if this model achieved the best accuracy
            if accuracy > best_accuracy:
```

```
                best_accuracy = accuracy
                best_model = rf

            print(f"n_estimators: {n}, max_depth: {depth},
min_samples_split: {min_samples}, Accuracy: {accuracy:.3f}")

# Print the best hyperparameters and accuracy
print("\nBest Hyperparameters:")
print(f"n_estimators: {best_model.n_estimators}, max_depth:
{best_model.max_depth}, "
      f"min_samples_split: {best_model.min_samples_split}, Accuracy:
{best_accuracy:.3f}")
```

```
n_estimators: 50, max_depth: None, min_samples_split: 2, Accuracy:
0.997
n_estimators: 50, max_depth: None, min_samples_split: 5, Accuracy:
0.994
n_estimators: 50, max_depth: None, min_samples_split: 10, Accuracy:
0.991
n_estimators: 50, max_depth: 10, min_samples_split: 2, Accuracy: 0.997
n_estimators: 50, max_depth: 10, min_samples_split: 5, Accuracy: 0.997
n_estimators: 50, max_depth: 10, min_samples_split: 10, Accuracy:
0.991
n_estimators: 50, max_depth: 20, min_samples_split: 2, Accuracy: 0.997
n_estimators: 50, max_depth: 20, min_samples_split: 5, Accuracy: 0.994
n_estimators: 50, max_depth: 20, min_samples_split: 10, Accuracy:
0.991
n_estimators: 100, max_depth: None, min_samples_split: 2, Accuracy:
0.997
n_estimators: 100, max_depth: None, min_samples_split: 5, Accuracy:
0.997
n_estimators: 100, max_depth: None, min_samples_split: 10, Accuracy:
0.994
n_estimators: 100, max_depth: 10, min_samples_split: 2, Accuracy:
0.997
n_estimators: 100, max_depth: 10, min_samples_split: 5, Accuracy:
0.997
n_estimators: 100, max_depth: 10, min_samples_split: 10, Accuracy:
0.994
n_estimators: 100, max_depth: 20, min_samples_split: 2, Accuracy:
0.997
n_estimators: 100, max_depth: 20, min_samples_split: 5, Accuracy:
0.997
n_estimators: 100, max_depth: 20, min_samples_split: 10, Accuracy:
0.994
n_estimators: 200, max_depth: None, min_samples_split: 2, Accuracy:
0.997
n_estimators: 200, max_depth: None, min_samples_split: 5, Accuracy:
0.997
n_estimators: 200, max_depth: None, min_samples_split: 10, Accuracy:
```

```
0.994
n_estimators: 200, max_depth: 10, min_samples_split: 2, Accuracy:
0.997
n_estimators: 200, max_depth: 10, min_samples_split: 5, Accuracy:
0.994
n_estimators: 200, max_depth: 10, min_samples_split: 10, Accuracy:
0.994
n_estimators: 200, max_depth: 20, min_samples_split: 2, Accuracy:
0.997
n_estimators: 200, max_depth: 20, min_samples_split: 5, Accuracy:
0.997
n_estimators: 200, max_depth: 20, min_samples_split: 10, Accuracy:
0.994

Best Hyperparameters:
n_estimators: 50, max_depth: None, min_samples_split: 2, Accuracy:
0.997
```

Regression

```
#Split into Train and Test Sets
#Mean Absolute Error (MAE)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import ElasticNet, Lasso, Ridge,
LinearRegression
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor,
GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# 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=42)

# Initialize a dictionary to store the results
results = {}

# Define and evaluate each regression algorithm
regressors = {
    'CART': DecisionTreeRegressor(),
    'Elastic Net': ElasticNet(),
    'AdaBoost': AdaBoostRegressor(),
    'K-NN': KNeighborsRegressor(),
    'Lasso Regression': Lasso(),
    'Ridge Regression': Ridge(),
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'MLP': MLPRegressor()
}

for name, regressor in regressors.items():
    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    results[name] = mae

# Print the MAE for each algorithm
for name, mae in results.items():
    print(f'{name}: Mean Absolute Error = {mae:.2f}')
```

```
CART: Mean Absolute Error = 196970.82
Elastic Net: Mean Absolute Error = 265543.15
AdaBoost: Mean Absolute Error = 306335.08
K-NN: Mean Absolute Error = 280099.82
Lasso Regression: Mean Absolute Error = 199079.56
Ridge Regression: Mean Absolute Error = 209465.26
Linear Regression: Mean Absolute Error = 199070.78
Random Forest: Mean Absolute Error = 180124.27
Gradient Boosting: Mean Absolute Error = 205418.92
MLP: Mean Absolute Error = 360855.99
```

```
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
#K-Fold Cross Variation
#R^2
import pandas as pd
```

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import ElasticNet, Lasso, Ridge,
LinearRegression
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor,
GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
df = pd.read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Initialize a dictionary to store the results
results = {}

# Define and evaluate each regression algorithm using K-fold Cross
Validation
regressors = {
    'CART': DecisionTreeRegressor(),
    'Elastic Net': ElasticNet(),
    'AdaBoost': AdaBoostRegressor(),
    'K-NN': KNeighborsRegressor(),
    'Lasso Regression': Lasso(),
    'Ridge Regression': Ridge(),
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'MLP': MLPRegressor()
}

k_fold = KFold(n_splits=5, shuffle=True, random_state=42)

for name, regressor in regressors.items():
    scores = cross_val_score(regressor, X, y, cv=k_fold, scoring='r2')
    mean_r2 = scores.mean()
    results[name] = mean_r2

# Print the R-squared (R^2) for each algorithm
```

```
for name, r2 in results.items():
    print(f'{name}: Mean R-squared (R^2) = {r2:.2f}')
```

```
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
CART: Mean R-squared (R^2) = -0.53
Elastic Net: Mean R-squared (R^2) = 0.15
AdaBoost: Mean R-squared (R^2) = -0.60
K-NN: Mean R-squared (R^2) = -0.08
Lasso Regression: Mean R-squared (R^2) = 0.53
Ridge Regression: Mean R-squared (R^2) = 0.53
Linear Regression: Mean R-squared (R^2) = 0.53
Random Forest: Mean R-squared (R^2) = 0.32
Gradient Boosting: Mean R-squared (R^2) = 0.46
MLP: Mean R-squared (R^2) = -0.38
```

```
C:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
#Random Forest MAE Hypertuning
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Load the dataset
filename = 'C:/Users/user/Desktop/ITD105
Files/CaseStudy1/Datasets/used_car_price.csv'
```

```python
df = pd.read_csv(filename)

# Features and target variable
X = df[['brand', 'year', 'mileage', 'fuel_type', 'transmission']]
Y = df['price']

# One-hot encoding for categorical variables (brand and fuel_type)
X = pd.get_dummies(X, columns=['brand', 'fuel_type', 'transmission'],
drop_first=True)

# Define the hyperparameters to be tuned
n_estimators_list = [50, 100, 200]
max_depth_list = [None, 10, 20]
min_samples_split_list = [2, 5, 10]

# Initialize a dictionary to store the results
results = {}

# Loop through different hyperparameters
for n_estimators in n_estimators_list:
    for max_depth in max_depth_list:
        for min_samples_split in min_samples_split_list:
            # 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=42)

            # Initialize and train the Random Forest model with the
current hyperparameters
            model = RandomForestRegressor(n_estimators=n_estimators,
max_depth=max_depth, min_samples_split=min_samples_split)
            model.fit(X_train, y_train)

            # Make predictions
            y_pred = model.predict(X_test)

            # Calculate MAE
            mae = mean_absolute_error(y_test, y_pred)

            # Store the MAE in the results dictionary
            results[f'{n_estimators} {max_depth} {min_samples_split}']
= mae

# Print the MAE for each set of hyperparameters
for params, mae in results.items():
    print(f'Hyperparameters: {params}, MAE = {mae:.2f}')
```

```
Hyperparameters: 50 None 2, MAE = 182228.24
Hyperparameters: 50 None 5, MAE = 183944.92
Hyperparameters: 50 None 10, MAE = 191451.29
Hyperparameters: 50 10 2, MAE = 203781.52
```

```
Hyperparameters: 50 10 5, MAE = 205453.82
Hyperparameters: 50 10 10, MAE = 208326.99
Hyperparameters: 50 20 2, MAE = 184014.74
Hyperparameters: 50 20 5, MAE = 187392.50
Hyperparameters: 50 20 10, MAE = 190737.77
Hyperparameters: 100 None 2, MAE = 181267.83
Hyperparameters: 100 None 5, MAE = 185898.92
Hyperparameters: 100 None 10, MAE = 189268.17
Hyperparameters: 100 10 2, MAE = 202844.31
Hyperparameters: 100 10 5, MAE = 204037.01
Hyperparameters: 100 10 10, MAE = 207474.02
Hyperparameters: 100 20 2, MAE = 184056.43
Hyperparameters: 100 20 5, MAE = 185613.13
Hyperparameters: 100 20 10, MAE = 190086.56
Hyperparameters: 200 None 2, MAE = 182771.66
Hyperparameters: 200 None 5, MAE = 184627.20
Hyperparameters: 200 None 10, MAE = 189835.21
Hyperparameters: 200 10 2, MAE = 203040.85
Hyperparameters: 200 10 5, MAE = 204136.29
Hyperparameters: 200 10 10, MAE = 205841.15
Hyperparameters: 200 20 2, MAE = 180738.81
Hyperparameters: 200 20 5, MAE = 185088.30
Hyperparameters: 200 20 10, MAE = 189119.05
```