Code

In [ ]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import re
import numpy as np
```

Data Prep

In [ ]:
```
# Read in Data
data = pd.read_csv("./Econ424_F2023_PC4_training_data_small.csv")
print(data.head())
```

In [ ]:
```
# Find in missing values
missing = data.isna().sum()
for x in range(len(missing)):
    print(str(data.columns[x]) + ": " + str(missing[x]))
    # print(missing[x])
data.shape
data.info()
```

In [ ]:
```
# Large Data
dataL = pd.read_csv("./Econ424_F2023_PC4_training_data_large.csv")
print(dataL.head())
```

In [ ]:
```
missingL = dataL.isna().sum()
for x in range(len(missingL)):
    print(str(dataL.columns[x]) + ": " + str(missingL[x]))
```

Missing is_certified, vehicicle_damage_category, combine_fuel_economy for all of them

In [ ]:
```
# useless categories
# fleet, is_cpo, is oemcpo major options, bed, bed height, bed length, cabir
data.drop(['is_certified','vehicle_damage_category', 'combine_fuel_economy',
  axis='columns', inplace=True)
data.shape
```

In [ ]:
```
data.columns
```

In [ ]:
```
for col in data:
    print(col)
    print(data[col].unique())
```

In [ ]:
```
# Define a regular expression pattern to extract float values
# pattern = r'(\d+\.\d+) in'
pattern = r' in'
patternGal = r'(\d+\.\d+) gal'
weirdCols = ["back_legroom", "front_legroom", "height", "length", "wheelbase
# Iterate through the columns and extract float components for matching colu
for column in weirdCols:
    if len(data[column].unique()) >= 4 and data[column].dtype == object:
        print(column)
        for i in range(len(data[column])):
                if pd.isna(data[column][i]):
                    continue
                elif isinstance(data[column][i], str):
                    # print("found string")

                    if len(data[column][i]) <= 2:
                        continue
                    end = data[column][i][-3:]
                    if end == " in":
                        data[column][i] = float(data[column][i][:-3])
                        continue
                    if len(data[column][i]) <= 3:
                        continue
                    end = data[column][i][-4:]
                    if end == " gal":
                        data[column][i] = float(data[column][i][:-4])
                        continue

                    if len(data[column][i]) <= 5:
                        continue
                    end = data[column][i][-6:]
                    if end == " seats":
                        data[column][i] = int(data[column][i][:-6])

data.head
```

In [ ]:
```
for col in data.columns:
    print(col + ": " + str(data[col].unique()))
```

In [ ]:
```
# Replace all with mean and mode
categorical_columns = ['trimid','body_type','city','dealer_zip','engine_type
                       ,'listing_color','major_options','make_name','model_r
bool_columns = ['frame_damaged','franchise_dealer','has_accidents','is_new',

for col in data.columns:
    if col in categorical_columns or col in bool_columns:
        # calculate mode
        average = "-1"
        # Replace "--" with NaN
        data[col] = data[col].replace(np.nan, "--")
        data[col] = data[col].replace("--", pd.NA)
        # Calculate the mode of the valid string values
        mode_value = data[col].mode(dropna=True).iloc[0]

        # Replace NaN with the mode
```

```
        data[col].fillna(mode_value,inplace=True)

    elif col != "listed_date":
        # calculate mean
        # Convert non-numeric values ("--") to NaN
        data[col] = pd.to_numeric(data[col], errors="coerce")

        # Calculate the mean of the valid numeric values
        mean_value = data[col].dropna().mean()

        # Replace NaN and "--" with the mean
        data[col].fillna(mean_value, inplace=True)


# Mean: back_legroom, city_fuel_economy, engine_displacement, front_legroom,
# highway_fuel_economy,  mileage, wheelbase, width

# Mode: maximum seating, owner count, seller rating, trimid
```

In [ ]:
```
for col in data.columns:
    print(col + ": " + str(data[col].unique()))
```

In [ ]:
```
missing = data.isna().sum()
for x in range(len(missing)):
    print(str(data.columns[x]) + ": " + str(missing[x]))
    # print(missing[x])
data.shape
data.info()
print(data['listed_date'])
print(data['year'])
```

In [ ]:
```
# Updated csv file
csv_file = "./updatedSmall.csv"


# Use numpy.savetxt to save the array as a CSV file
data.to_csv(csv_file,index=False, encoding="utf-8")
```

Apply Label Encoder and Standard Scaler

In [ ]:
```
data = pd.read_csv("./updatedSmallFinal.csv")
```

In [ ]:
```
# setup categorical variables
# label encoder to encode the different categorical features
categorical_columns = ['trimid','body_type','city','dealer_zip','engine_type
                       ,'listing_color','major_options','make_name','model_r
bool_columns = ['frame_damaged','franchise_dealer','has_accidents','is_new',


label_encoder = LabelEncoder()
for category in categorical_columns:
    print("Doing it for category: " + category)
```

```
    data[category] = data[category].astype(str)
    print(all)

    data[category] = label_encoder.fit_transform(data[category])
# data['target'] = np.log(data['price'])
for category in bool_columns:
    print("Doing it for category: " + category)

    data[category] = data[category].astype(str)
    print(all)

    data[category] = label_encoder.fit_transform(data[category])
```

In [ ]:
```
data.head()
data.info()
```

In [ ]:
```
missing = data.isna().sum()

for x in range(len(missing)):
    print(str(data.columns[x]) + ": " + str(missing[x]))
    # print(missing[x])
```

In [ ]:
```
data['target'] = np.log(data['price'])
```

In [ ]:
```
print(data['target'])
```

Create Different Models

In [ ]:
```
# Imports
from matplotlib.pyplot import subplots
from statsmodels.datasets import get_rdataset
import sklearn.model_selection as skm
from ISLP import load_data , confusion_table
from ISLP.models import ModelSpec as MS
from sklearn.tree import (DecisionTreeClassifier as DTC, DecisionTreeRegress
from sklearn.metrics import (accuracy_score , log_loss)
from sklearn.ensemble import (RandomForestRegressor as RF, GradientBoostingR

from matplotlib.pyplot import subplots
import statsmodels.api as sm


from sklearn.model_selection import train_test_split
from matplotlib.pyplot import subplots
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score


from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
# Set up Data
Y = data['target']
```

```
X = data.drop(columns=['price','target','listed_date'])
```

In [ ]:
```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, rar
```

Bagging

In [ ]:
```
bag = RF(max_features=X_train.shape[1],random_state=0)
bag.fit(X_train,y_train)
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
y_hat_bag = bag.predict(X_test)
ax.scatter(y_hat_bag, y_test)
np.mean((y_test - y_hat_bag)**2)
```

In [ ]:
```
feature_imp = pd.DataFrame( {'importance':bag.feature_importances_}, index=X
feature_imp.sort_values(by='importance', ascending=False)
```

In [ ]:
```
data_bag = RF(max_features=X_train.shape[1], n_estimators=500, random_state=
y_hat_bag = data_bag.predict(X_test)
np.mean((y_test - y_hat_bag)**2)
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
ax.scatter(y_hat_bag, y_test)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Small Dataset with
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
ax.axline((7.5,7.5), slope=1)
```

In [ ]:
```
feature_imp = pd.DataFrame( {'importance':data_bag.feature_importances_}, ir
feature_imp.sort_values(by='importance', ascending=False)
```

In [ ]:
```
r2_score(y_test, y_hat_bag)
```

Random Forests

In [ ]:
```
rf = RF(max_features=int(np.sqrt(X_train.shape[1])),random_state=0)
rf.fit(X_train,y_train)
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
y_hat_rf = rf.predict(X_test)
ax.scatter(y_hat_rf, y_test)
np.mean((y_test - y_hat_rf)**2)
```

In [ ]:
```
r2_score(y_test, y_hat_rf)
```

In [ ]:
```
feature_imp = pd.DataFrame( {'importance':rf.feature_importances_}, index=X.
feature_imp.sort_values(by='importance', ascending=False)
```

Boosting

In [ ]:
```
data_boost = GBR(n_estimators=5000, learning_rate=0.2, max_depth=3, random_s
data_boost.fit(X_train, y_train)
```

In [ ]:
```
test_error = np.zeros_like(data_boost.train_score_)
for idx, y_ in enumerate(data_boost.staged_predict(X_test)):
    test_error[idx] = np.mean((y_test - y_)**2)
plot_idx = np.arange(data_boost.train_score_.shape[0])
ax = subplots(figsize=(8,8))[1]
ax.plot(plot_idx,data_boost.train_score_, 'b',label='Training')
ax.plot(plot_idx, test_error ,'r',label='Test')
ax.legend();
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
y_hat_boost = rf.predict(X_test)
ax.scatter(y_hat_boost, y_test)
np.mean((y_test - y_hat_boost)**2)
```

In [ ]:
```
y_hat_boost = data_boost.predict(X_test)
np.mean((y_test - y_hat_boost)**2)
```

In [ ]:
```
r2_score(y_test, y_hat_boost)
```

Create log charts for each model

In [ ]:
```
# choose best one
bestModel = data_bag
```

Make Predictions

In [ ]:
```
# Preprocess prediction data
# Read in Data
dataPred = pd.read_csv("./Econ424_F2023_PC4_test_data_without_response_var.c
print(dataPred.head())

dataPred.drop(['is_certified','vehicle_damage_category', 'combine_fuel_econc
  axis='columns', inplace=True)
dataPred.shape

weirdCols = ["back_legroom", "front_legroom", "height", "length", "wheelbase
# Iterate through the columns and extract float components for matching colu
for column in weirdCols:
    if len(dataPred[column].unique()) >= 4 and dataPred[column].dtype == obj
        print(column)
        for i in range(len(dataPred[column])):
                if pd.isna(dataPred[column][i]):
                    continue
```

```
                elif isinstance(dataPred[column][i], str):
                    # print("found string")

                    if len(dataPred[column][i]) <= 2:
                        continue
                    end = dataPred[column][i][-3:]
                    if end == " in":
                        dataPred[column][i] = float(dataPred[column][i][:-3]
                        continue
                    if len(dataPred[column][i]) <= 3:
                        continue
                    end = dataPred[column][i][-4:]
                    if end == " gal":
                        dataPred[column][i] = float(dataPred[column][i][:-4]
                        continue

                    if len(dataPred[column][i]) <= 5:
                        continue
                    end = dataPred[column][i][-6:]
                    if end == " seats":
                        dataPred[column][i] = int(dataPred[column][i][:-6])
```

In [ ]:
```
# Replace all with mean and mode
categorical_columns = ['trimid','body_type','city','dealer_zip','engine_type
                       ,'listing_color','major_options','make_name','model_n
bool_columns = ['frame_damaged','franchise_dealer','has_accidents','is_new',

for col in dataPred.columns:
    if col in categorical_columns or col in bool_columns:
        # calculate mode
        average = "-1"
        # Replace "--" with NaN
        dataPred[col] = dataPred[col].replace(np.nan, "--")
        dataPred[col] = dataPred[col].replace("--", pd.NA)
        # Calculate the mode of the valid string values
        mode_value = dataPred[col].mode(dropna=True).iloc[0]

        # Replace NaN with the mode
        dataPred[col].fillna(mode_value,inplace=True)

    elif col != "listed_date":
        # calculate mean
        # Convert non-numeric values ("--") to NaN
        dataPred[col] = pd.to_numeric(dataPred[col], errors="coerce")

        # Calculate the mean of the valid numeric values
        mean_value = dataPred[col].dropna().mean()

        # Replace NaN and "--" with the mean
        dataPred[col].fillna(mean_value, inplace=True)
```

In [ ]:
```
categorical_columns = ['trimid','body_type','city','dealer_zip','engine_type
                       ,'listing_color','major_options','make_name','model_n
bool_columns = ['frame_damaged','franchise_dealer','has_accidents','is_new',
```

```
label_encoder = LabelEncoder()
for category in categorical_columns:
    print("Doing it for category: " + category)
    dataPred[category] = dataPred[category].astype(str)
    print(all)

    dataPred[category] = label_encoder.fit_transform(dataPred[category])
for category in bool_columns:
    print("Doing it for category: " + category)
    dataPred[category] = dataPred[category].astype(str)
    print(all)

    dataPred[category] = label_encoder.fit_transform(dataPred[category])

missing = dataPred.isna().sum()

for x in range(len(missing)):
    print(str(dataPred.columns[x]) + ": " + str(missing[x]))
    # print(missing[x])
```

In [ ]:
```
# apply prediction
dataPred.drop(["price"],axis="columns",inplace=True)
Y_test = bestModel.predict(dataPred)
```

In [ ]:
```
len(Y_test)
```

In [ ]:
```
# output to csv file
csv_file_out = "./output.csv"

# Save the DataFrame to a CSV file
np.savetxt(csv_file_out, Y_test, delimiter="\n", fmt="%1.6f")
```

In [ ]:
```

```