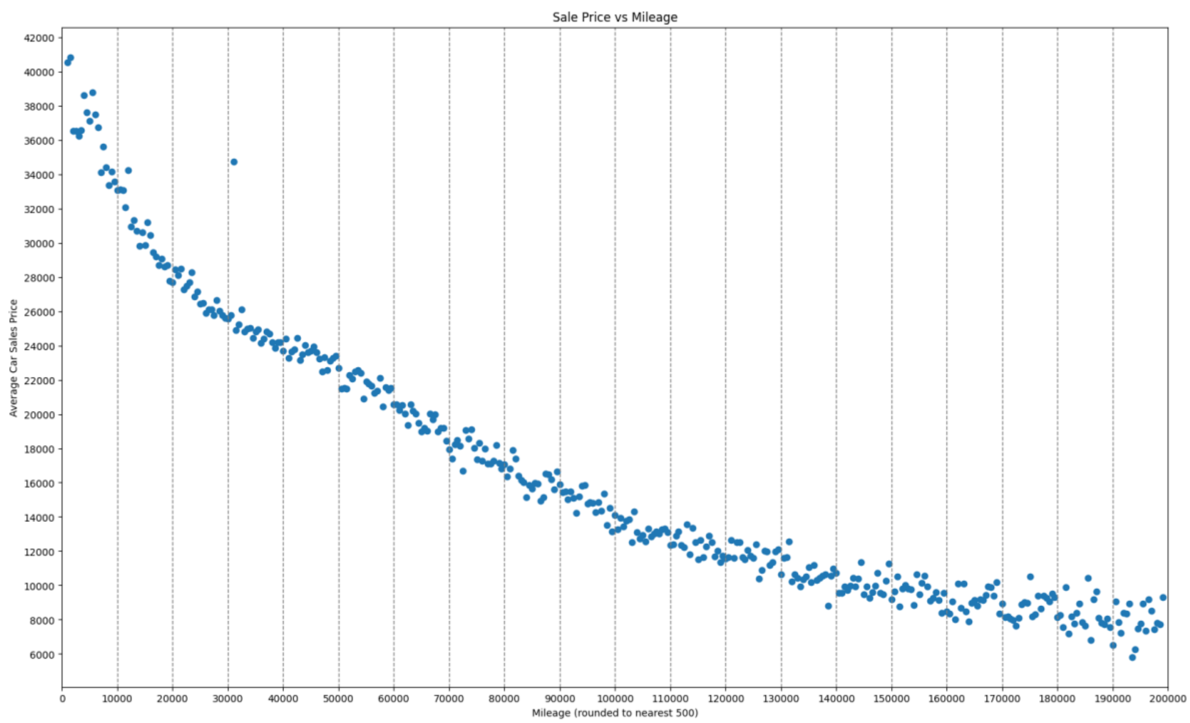
Name
TheDebugger

Mean Squared Error: 0.004400654824710213
R^2: 0.9206683031451792

Method: Bagging

I noticed a few of the top entries used this method, so I changed from bagging to XGBoost and used the large dataset this time around for training but found that I had better performance with bagging.

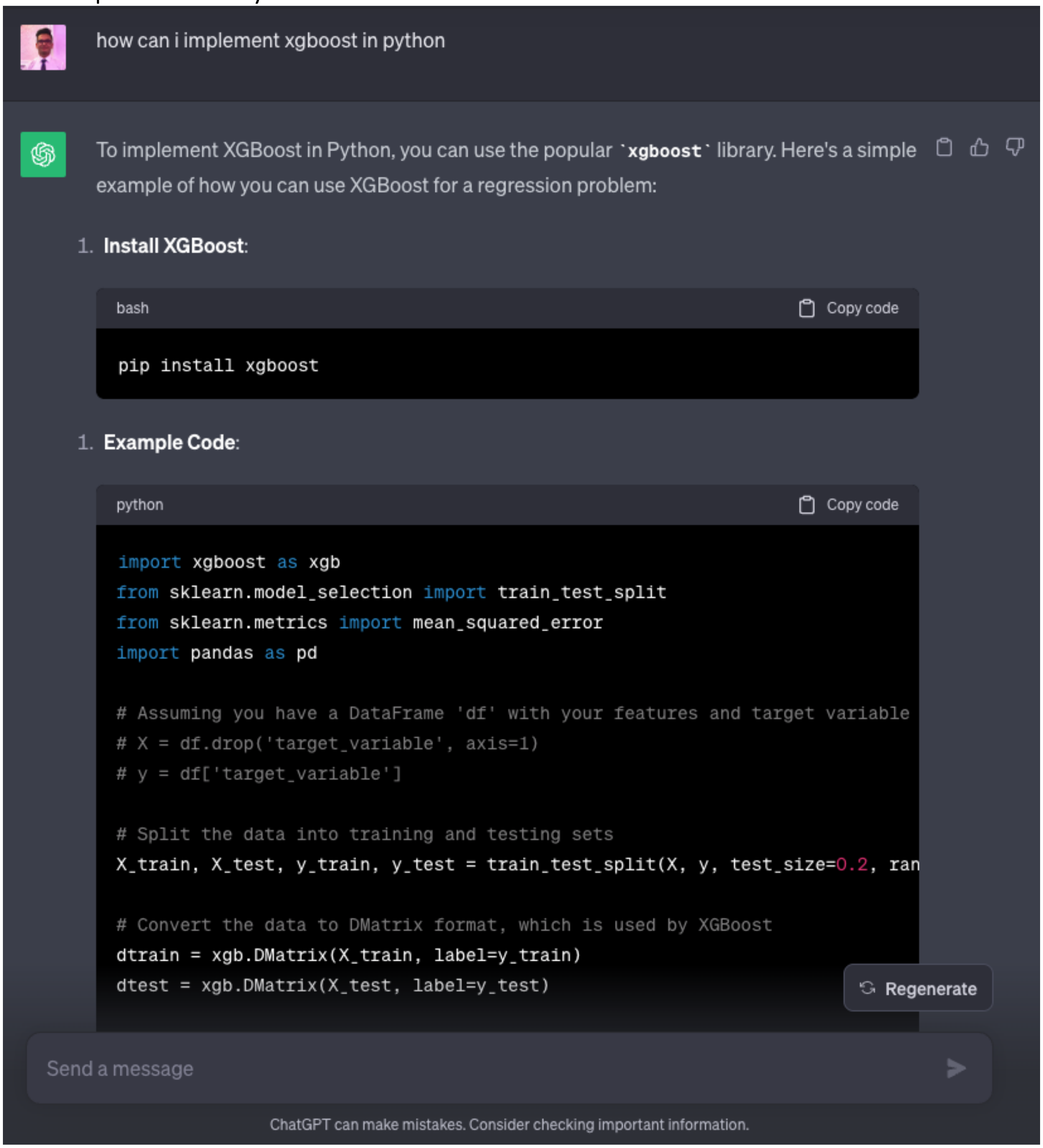
Question 2 (Plot):



It seems to be a continuous trend in our dataset. There do not seem to be any apparent discontinuities across each 10,000-mile mark. There are a few very faint discrepancies, but it is not nearly as apparent as the research study had. One of the possible reasons for this is the dataset. The dataset used in the research had 22 million transactions and contained data from 2002 to 2008. Our dataset is much more limited (less than 0.5% of the data the study used), so it might not be visible to see any discontinuities. Our sample may not represent these economic factors well. Considering the timing of 2002-2008 and the fact that the average price of cars back then was $10,000 also may play a factor.

Question 3 (ChatGPT):

For question 1, I noticed that some of the top entries used XGBoost so I asked ChatGPT how I could implement it in Python.



```python
# Convert the data to DMatrix format, which is used by XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Set XGBoost parameters
params = {
    'objective': 'reg:squarederror',  # for regression tasks
    'max_depth': 3,
    'learning_rate': 0.1,
    'n_estimators': 100
}

# Train the model
model = xgb.train(params, dtrain, num_boost_round=10)

# Make predictions on the test set
predictions = model.predict(dtest)

# Evaluate the model
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

This is a basic example, and you might need to adjust the parameters based on your specific dataset and problem. Also, you can use `XGBRegressor` from scikit-learn, which provides a similar interface.

Remember to replace `'target_variable'` with the actual name of your target variable column. Adjust the parameters in the `params` dictionary based on your specific requirements.

Regenerate

Send a message

ChatGPT can make mistakes. Consider checking important information.

For question 2, I provided a description of the plot directly from the research paper to ChatGPT and asked it to help me build the plot. This saved me a lot of time and worked out very well with few modifications required.

given a dataset of car prices and mileage, how can i make a plot to match plot 2 from the following description:
Begin the empirical analysis with a simple plot of the raw price data as a function of mileage using information on the over 22 million cars that were sold at auctions during our sample period. In Figure 2, each dot shows the average sale price for cars in a 500-mile mileage bin, starting at 1,000 miles. There is a dot for the average price of cars with 1,000 through 1,499 miles, then a dot for cars with 1,500 to 1,999 miles, and so on through 125,000 miles. The vertical lines in the graph indicate each 10,000-mile mark. As one would expect, average prices decrease with increasing mileage. Within each 10,000-mile band, average prices decline quite smoothly. There are, however, clear and sizeable discontinuities in average prices at nearly all 10,000-mile marks.

To create a similar plot in Python using a dataset with car prices and mileage, you can use a library like Matplotlib. Below is an example code snippet to get you started. This assumes you have a DataFrame named `df` with columns 'mileage' and 'price':

python Copy code

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sample DataFrame (replace this with your actual data)
data = {
    'mileage': np.random.randint(1000, 126000, size=1000),
    'price': np.random.randint(5000, 30000, size=1000),
}

df = pd.DataFrame(data)

# Define the mileage bins and calculate average prices for each bin
mileage_bins = np.arange(1000, 126000, 500)
df['mileage_bin'] = pd.cut(df['mileage'], bins=mileage_bins, right=False)
average_prices = df.groupby('mileage_bin')['price'].mean()

# Plot the data
plt.figure(figsize=(10, 6))
plt.scatter(df['mileage'], df['price'], alpha=0.3, label='Individual Cars')
plt.plot(average_prices.index.right, average_prices.values, color='red', mar

# Add vertical lines at each 10,000-mile mark
for mile_mark in range(10000, 130000, 10000):
    plt.axvline(x=mile_mark, color='gray', linestyle='--', linewidth=1)

# Set labels and title
```

Regenerate

Send a message

how can i set the x-axis scale to be in intervals of 10,000 to a maximum of what was mentioned

You can set the x-axis scale to be in intervals of 10,000 by adjusting the ticks and limits. Here's an updated version of the code:

python   Copy code

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sample DataFrame (replace this with your actual data)
data = {
    'mileage': np.random.randint(1000, 126000, size=1000),
    'price': np.random.randint(5000, 30000, size=1000),
}

df = pd.DataFrame(data)

# Define the mileage bins and calculate average prices for each bin
mileage_bins = np.arange(1000, 130000, 5000)
df['mileage_bin'] = pd.cut(df['mileage'], bins=mileage_bins, right=False)
average_prices = df.groupby('mileage_bin')['price'].mean()

# Extract midpoints of the bins
midpoints = (average_prices.index.left + average_prices.index.right) / 2

# Plot the data
plt.figure(figsize=(10, 6))
plt.scatter(df['mileage'], df['price'], alpha=0.3, label='Individual
plt.plot(midpoints, average_prices.values, color='red', marker='o', l
```



```python
# Add vertical lines at each 10,000-mile mark
for mile_mark in range(10000, 130000, 10000):
    plt.axvline(x=mile_mark, color='gray', linestyle='--', linewidth=1)

# Set x-axis ticks to 10,000-mile intervals
plt.xticks(range(0, 130000, 10000))

# Set labels and title
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.title('Average Car Price vs Mileage')
plt.legend()
plt.grid(True)
plt.show()
```

This code sets the x-axis ticks to be at intervals of 10,000 miles and limits the x-axis to a maximum of 130,000 miles. Adjust the values in `range` and `xticks` as needed for your specific data range.

Code:

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

Data Prep

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

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

In [ ]:
```python
# Repeat for large
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 [ ]:
```python
# useless categories
# fleet, is_cpo, is oemcpo major options, bed, bed height, bed length, cabin, iscab, transmission display, engine cylinders
data.drop(['is_certified','vehicle_damage_category', 'combine_fuel_economy','wheel_system_display','fleet','is_cpo', 'is_oemcpo','bed',
           'bed_height','bed_length','cabin','iscab',
           'transmission_display','engine_cylinders'], errors='ignore',
  axis='columns', inplace=True)
dataL.drop(['is_certified','vehicle_damage_category', 'combine_fuel_economy','wheel_system_display','fleet','is_cpo', 'is_oemcpo','bed',
            'bed_height','bed_length','cabin','iscab',
            'transmission_display','engine_cylinders'], errors='ignore',
  axis='columns', inplace=True)
```

In [ ]:
```python
# Remove data points under 40000
data = data[data['price'] >= 40000]
dataL = dataL[dataL['price'] >= 40000]
```

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

dataL.columns
dataL.shape
```

In [ ]:
```python
# Look at data
for col in data:
    print(col)
    print(data[col].unique())
```

In [ ]:
```python
# Preprocess data to valid format

floatCols = ["back_legroom", "front_legroom", "height", "length", "wheelbase", "width", "fuel_tank_volume"]
intCols = ["maximum_seating"]
for col in floatCols:
    # Preprocess columns in small set
    data[col] = data[col].str.split(' ').str[0]
    data[col].replace('--', np.nan , inplace=True)
    data[col] = pd.to_numeric(data[col],downcast='float')

    # Preprocess columns in large set
    dataL[col] = dataL[col].str.split(' ').str[0]
    dataL[col].replace('--', np.nan , inplace=True)
    dataL[col] = pd.to_numeric(dataL[col],downcast='float')

for col in intCols:
    data[col] = data[col].str.split(' ').str[0]
    data[col].replace('--', np.nan , inplace=True)
    data[col] = pd.to_numeric(data[col],downcast='integer')
    data[col].replace(np.nan, 5, inplace=True)

    dataL[col] = dataL[col].str.split(' ').str[0]
    dataL[col].replace('--', np.nan , inplace=True)
    dataL[col] = pd.to_numeric(dataL[col],downcast='integer')
    dataL[col].replace(np.nan, 5, inplace=True)
```

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

In [ ]:
```python
# Replace all with mean and mode
categorical_columns = ['trimid','body_type','city','dealer_zip','engine_type','exterior_color','franchise_make','fuel_type','horsepower','interior_color'
                       ,'listing_color','major_options','make_name','model_name','power','sp_name','torque','transmission','trim_name','wheel_system']
bool_columns = ['frame_damaged','franchise_dealer','has_accidents','is_new','salvage','theft_title']

for col in data.columns:
    if col in categorical_columns or col in bool_columns:
        # 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, fuel_tank_volume, height,
# highway_fuel_economy,  mileage, wheelbase, width

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

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

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

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

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

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

In [ ]:

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

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

In [ ]:

```
# Confirm that data is now valid
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 [ ]:

```
# Update the csv files to not have to preprocess data each time
# 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", float_format="%1.6f")

# Updated csv file
csv_fileL = "./updatedLarge.csv"


# Use numpy.savetxt to save the array as a CSV file
dataL.to_csv(csv_fileL,index=False, encoding="utf-8", float_format="%1.6f")
```

Apply Label Encoder and Standard Scaler

In [ ]:

```
data = pd.read_csv("./updatedSmall.csv")
dataL = pd.read_csv("./updatedLarge.csv")
```

In [ ]:

```
# setup categorical variables
# label encoder to encode the different categorical features
categorical_columns = ['trimid','body_type','city','dealer_zip','engine_type','exterior_color','franchise_make','fuel_type','horsepower','interior_color'
                       ,'listing_color','major_options','make_name','model_name','power','sp_name','torque','transmission','trim_name','wheel_system']
bool_columns = ['frame_damaged','franchise_dealer','has_accidents','is_new','salvage','theft_title']


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])

    # Do same for large data set
    dataL[category] = dataL[category].astype(str)
    print(all)

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

```
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])

    # Do same for large dataset
    dataL[category] = dataL[category].astype(str)
    print(all)

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

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

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

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

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

missingL = dataL.isna().sum()

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

In [ ]:
```
data['target'] = np.log(data['price'])
dataL['target'] = np.log(dataL['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, DecisionTreeRegressor as DTR,  plot_tree, export_text)
from sklearn.metrics import (accuracy_score , log_loss)
from sklearn.ensemble import (RandomForestRegressor as RF, GradientBoostingRegressor as GBR)

from matplotlib.pyplot import subplots
import statsmodels.api as sm
import xgboost as xgb
from sklearn.metrics import mean_squared_error

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'])

YL = dataL['target']
XL = dataL.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, random_state=42)
X_trainL, X_testL, y_trainL, y_testL = train_test_split(XL, YL, test_size=0.3, random_state=42)
```

Bagging

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

bagL = RF(max_features=X_trainL.shape[1],random_state=0)
bagL.fit(X_trainL,y_trainL)
```

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

# calculate mse
mse = mean_squared_error(y_test, y_hat_bag)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_test, y_hat_bag)
print(f'R^2: {r2}')



# Repeat for Large
axL = subplots(figsize=(8,8))[1]
y_hat_bagL = bagL.predict(X_testL)
axL.scatter(y_hat_bagL, y_testL)

# calculate mse
mse = mean_squared_error(y_testL, y_hat_bagL)
```

```
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_testL, y_hat_bagL)
print(f'R^2: {r2}')
```

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

In [ ]:
```
feature_impL = pd.DataFrame( {'importance':bagL.feature_importances_}, index=XL.columns)
feature_impL.sort_values(by='importance', ascending=False)
```

In [ ]:
```
data_bag = RF(max_features=X_train.shape[1], n_estimators=500, random_state=0).fit(X_train, y_train)
y_hat_bag_500 = data_bag.predict(X_test)


data_bagL = RF(max_features=X_trainL.shape[1], n_estimators=500, random_state=0).fit(X_trainL, y_trainL)
y_hat_bag_500L = data_bagL.predict(X_testL)
```

In [ ]:
```
mse = mean_squared_error(y_test, y_hat_bag_500)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_test, y_hat_bag_500)
print(f'R^2: {r2}')

mse = mean_squared_error(y_testL, y_hat_bag_500L)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_testL, y_hat_bag_500L)
print(f'R^2: {r2}')
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
ax.scatter(y_hat_bag_500, y_test)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Small Dataset with Bagging')
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
ax.axline((10.5,10.5), slope=1)
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
ax.scatter(y_hat_bag_500L, y_testL)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Large Dataset with Bagging')
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
ax.axline((10.5,10.5), slope=1)
```

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

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

Random Forests

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

In [ ]:
```
rfL = RF(max_features=int(np.sqrt(X_trainL.shape[1])),random_state=0)
rfL.fit(X_trainL,y_trainL)
```

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)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Small Dataset with Random Forest')
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
y_hat_rfL = rfL.predict(X_testL)
ax.scatter(y_hat_rfL, y_testL)
np.mean((y_testL - y_hat_rfL)**2)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Large Dataset with Random Forest')
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
```

In [ ]:
```
mse = mean_squared_error(y_test, y_hat_rf)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_test, y_hat_rf)
print(f'R^2: {r2}')
```

In [ ]:
```
mse = mean_squared_error(y_testL, y_hat_rfL)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_testL, y_hat_rfL)
print(f'R^2: {r2}')
```

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

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

Boosting

In [ ]:
```
data_boost = GBR(n_estimators=5000, learning_rate=0.2, max_depth=3, random_state=0)
data_boost.fit(X_train, y_train)
```

In [ ]:
```
data_boostL = GBR(n_estimators=5000, learning_rate=0.2, max_depth=3, random_state=0)
data_boostL.fit(X_trainL, y_trainL)
```

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

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
y_hat_boost = data_boost.predict(X_test)
ax.scatter(y_hat_boost, y_test)
np.mean((y_test - y_hat_boost)**2)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Small Dataset with Boosting')
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
y_hat_boostL = data_boostL.predict(X_testL)
ax.scatter(y_hat_boostL, y_testL)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Large Dataset with Boosting')
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
```

In [ ]:
```
mse = mean_squared_error(y_test, y_hat_boost)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_test, y_hat_boost)
print(f'R^2: {r2}')
```

In [ ]:
```
mse = mean_squared_error(y_testL, y_hat_boostL)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_testL, y_hat_boostL)
print(f'R^2: {r2}')
```

XGBoost

In [ ]:
```
# Convert the data to DMatrix format, which is used by XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

params = {
'objective': 'reg:squarederror',
'eval_metric': 'rmse',
'eta': 0.1, # lr
'max_depth': 9, # depth
'subsample': 0.3,
'colsample_bytree': 0.3
}
# train
num_round = 150
model = xgb.train(params, dtrain, num_round)
# predict
y_train_xgb = model.predict(dtrain)
y_hat_xgb = model.predict(dtest)


# calculate mse
mse = mean_squared_error(y_test, y_hat_xgb)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_test, y_hat_xgb)
print(f'R^2: {r2}')
```

In [ ]:
```
# Large model
dtrainL = xgb.DMatrix(X_trainL, label=y_trainL)
dtestL = xgb.DMatrix(X_testL, label=y_testL)
modelL = xgb.train(params, dtrainL, num_round)
# predict
y_train_xgbL = modelL.predict(dtrainL)
y_hat_xgbL = modelL.predict(dtestL)


# calculate mse
mse = mean_squared_error(y_testL, y_hat_xgbL)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_testL, y_hat_xgbL)
print(f'R^2: {r2}')
```

In [ ]:
```
ax = subplots(figsize=(8,8))[1]
ax.scatter(y_hat_xgb, y_test)
np.mean((y_test - y_hat_xgb)**2)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Small Dataset with XGBoost')
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
```

```
In [ ]: ax = subplots(figsize=(8,8))[1]
        ax.scatter(y_hat_xgbL, y_testL)
        np.mean((y_testL - y_hat_xgbL)**2)
        ax.title.set_text('Log Car Price (Predicted vs Actual) in Large Dataset with XGBoost')
        ax.set_xlabel("Log Observed Car Prices")
        ax.set_ylabel("Log Predicted Car Prices")
```

Create log charts for each model

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

Make Predictions

```
In [ ]: # Preprocess prediction data
        # Read in Data
        dataPred = pd.read_csv("./Econ424_F2023_PC5_test_data_without_response_var.csv")
        print(dataPred.head())

        dataPred.drop(['is_certified','vehicle_damage_category', 'combine_fuel_economy','wheel_system_display','fleet','is_cpo', 'is_oemcpo','bed','bed_height','bed_
          axis='columns', inplace=True)
        dataPred.shape

        weirdCols = ["back_legroom", "front_legroom", "height", "length", "wheelbase", "width", "maximum_seating", "fuel_tank_volume"]
        # Iterate through the columns and extract float components for matching columns
        for column in weirdCols:
            if len(dataPred[column].unique()) >= 4 and dataPred[column].dtype == object:
                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','exterior_color','franchise_make','fuel_type','horsepower','interior_color'
                               ,'listing_color','major_options','make_name','model_name','power','sp_name','torque','transmission','trim_name','wheel_system']
        bool_columns = ['frame_damaged','franchise_dealer','has_accidents','is_new','salvage','theft_title']

        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','exterior_color','franchise_make','fuel_type','horsepower','interior_color'
                               ,'listing_color','major_options','make_name','model_name','power','sp_name','torque','transmission','trim_name','wheel_system']
        bool_columns = ['frame_damaged','franchise_dealer','has_accidents','is_new','salvage','theft_title']


        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 [ ]:
```
dataPred.drop(columns=['price'],inplace=True)
```

In [ ]:
```
# apply prediction
Y_test = bestModel.predict(dataPred)
Y_testL = bestModelL.predict(dataPred)
```

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

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 [ ]:
```
# output to csv file
csv_file_outL = "./outputL.csv"

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

Plot of Sales Price vs Mileage

In [ ]:
```
data.shape
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import math

# Create a new DataFrame with average prices for each mileage bin
max_mileage = 200000
mileage_bins = range(1000, max_mileage, 500)

# Calculate bins and average prices
df['mileage_bins'] = pd.cut(df['mileage'], bins=mileage_bins, right=False)
avg_prices = df.groupby('mileage_bins')['price'].mean().reset_index()

# Set the size of the plot
plt.figure(figsize=(20, 20))  # Adjust the width and height as needed

fig, ax = plt.subplots(figsize=(20,12))

# Plot the average prices for each mileage bin
ax.scatter(mileage_bins[:-1], avg_prices['price'])

# Set the y-axis ticks to go up by 2000
ax.set_xticks(np.arange(0, max_mileage+10000, 10000))
ax.set_yticks(np.arange(6000, max(avg_prices['price']) + 2000, 2000))

# Label the axes
ax.set_xlabel('Mileage (rounded to nearest 500)')
ax.set_ylabel('Average Car Sales Price')
ax.set_title('Sale Price vs Mileage')
ax.set_xlim(0,max_mileage)

# # Add vertical lines at each 10,000-mile mark
for mile_mark in range(10000, max_mileage, 10000):
    ax.axvline(x=mile_mark, color='gray', linestyle='--', linewidth=1)
```