

Prediction Competition 5: Ensemble Methods, Algorithmic Bias and AI Alignment

Anonymized name: MegumiFushiguro

$R^2 = 0.942$

$MSE = 0.022$

Model: Boosting

Q2

I split the datasets into 2 parts; training and test with an 80/20 split. We use the training set to fit a model with fixed parameters, then we use that model to predict the test set. I used the model trained on the pc3/4 data to predict the test data of pc3/4 and pc5. Then used the model trained on the pc5 data to predict the test data of pc3/4 and pc5. Below is the R^2 from both models in the test sets:

$$PC3_{Boosting} R^2_{PC3(Test)} = 0.893$$

$$PC3_{Boosting} R^2_{PC5(Test)} = -1.556$$

$$PC5_{Boosting} R^2_{PC3(Test)} = -0.188$$

$$PC5_{Boosting} R^2_{PC5(Test)} = 0.9462717752640761$$

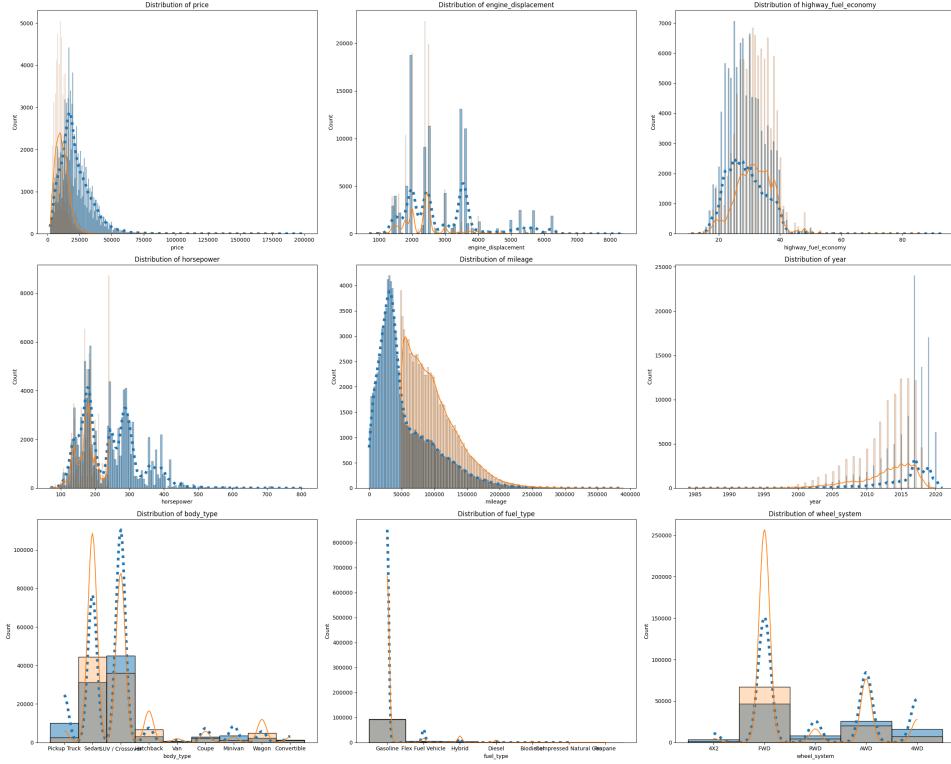


Figure 1: A multigraph plot of the distribution of the variables in the prediction competition 4 training dataset (small) v prediction competition 5 dataset

As we can see from Figure 1 above the distributions of the prediction competition 3/4 dataset and the prediction competition 5 dataset are different. With regards to the prediction competition 5 dataset; the price and year distributions are skewed more to the left, the highway fuel economy and milage distributions are skewed more to the right. We can also see that the PC5 horsepower distribution data (shown in orange) has a hardstop at around 250 horsepower. This is likely what is causing the large differences in the differing distributions as cars with lower horsepower will typically be priced lower and be older, and it is typical of low horsepower cars to have a higher highway fuel economy and mileage. To combat this in the prediction competition I created a dataset containing both small and pc5 datasets then took a random sample of 125,000 obvbservations from the combined dataset so that the distributions would average between the two datasets.

Q3



Figure 2: A photo generated by DALLE the text-to-image generative AI program

Figure 2 is a photo generated by DALLE the text-to-image generative AI program using the prompt 'planet sized formula 1 cars racing around the milky way galaxy as their race track'. It is evident that the program failed since it doesn't accurately depict what the prompt was requesting. The formula 1 cars are not clear, the AI attempted to add a race track with the light blue lines and placed it all on a background of space.

Q4

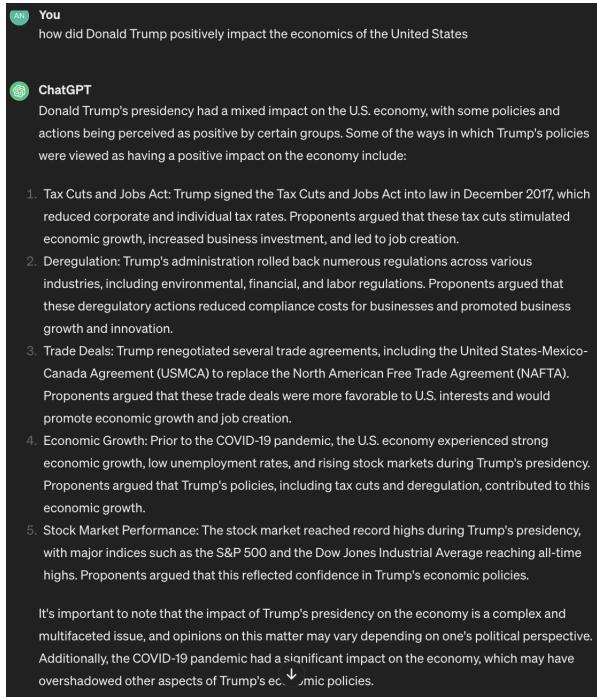


Figure 3: Screenshot of conversation ChatGPT by OpenAI

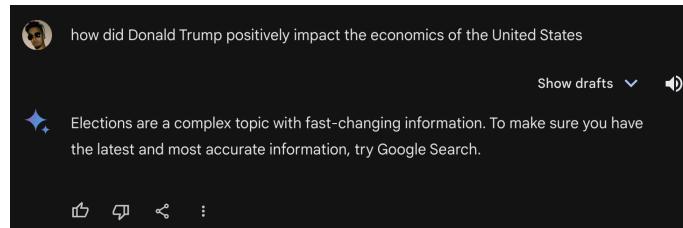
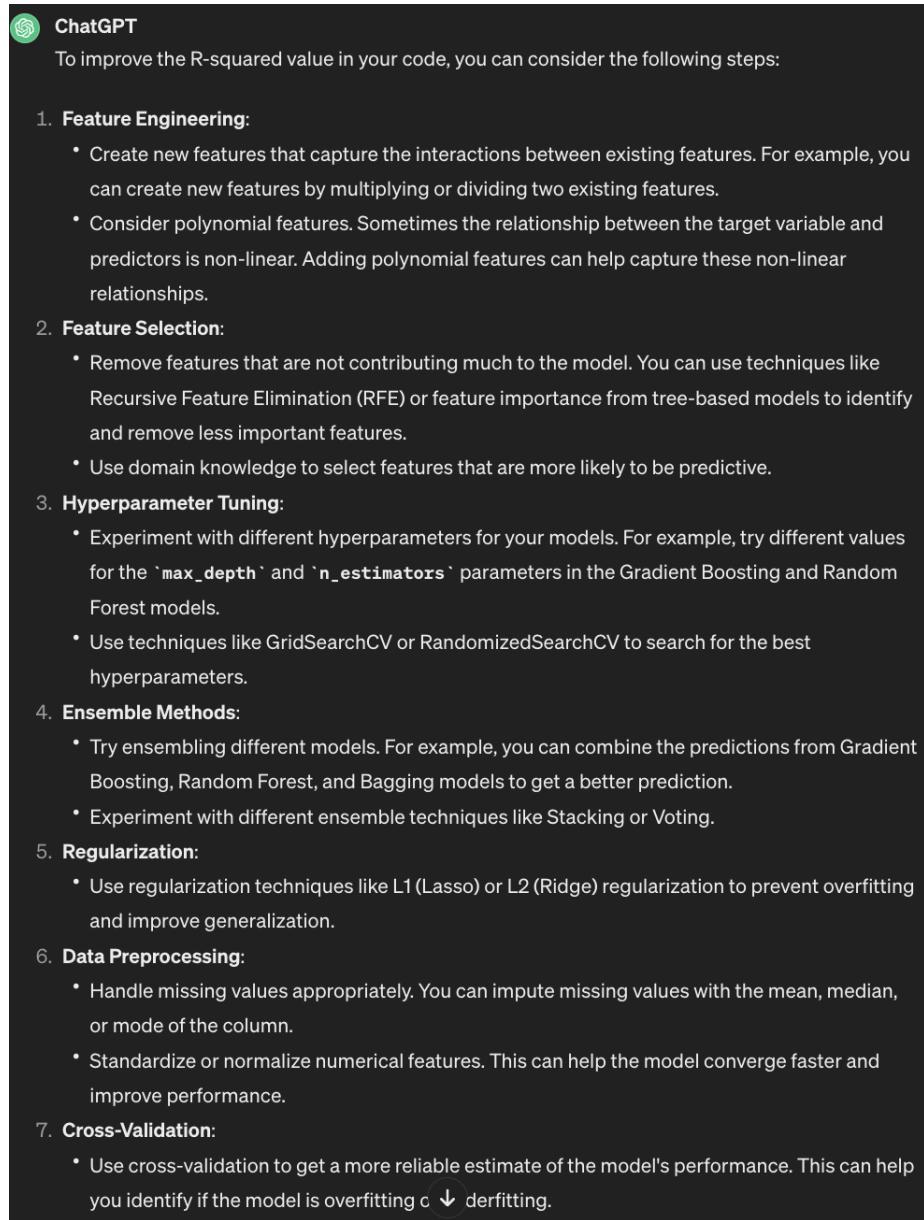


Figure 4: Screenshot of conversation with Gemini by Google

Figure 3 and 4 are screen captures of responses from ChatGPT by OpenAI and Gemini by Google using the prompt 'How did Donald Trump positively impact the economics of the United States'. It is clear that ChatGPT took a more neutral approach by sharing points that can be viewed as having a positive impact on the United States economy. Googles Gemini AI refused to answer the question and redirected me to doing a google search instead.

Q5



The screenshot shows a conversation with ChatGPT. The user asks for steps to improve the R-squared value. ChatGPT lists seven categories of steps:

- Feature Engineering:**
 - Create new features that capture the interactions between existing features. For example, you can create new features by multiplying or dividing two existing features.
 - Consider polynomial features. Sometimes the relationship between the target variable and predictors is non-linear. Adding polynomial features can help capture these non-linear relationships.
- Feature Selection:**
 - Remove features that are not contributing much to the model. You can use techniques like Recursive Feature Elimination (RFE) or feature importance from tree-based models to identify and remove less important features.
 - Use domain knowledge to select features that are more likely to be predictive.
- Hyperparameter Tuning:**
 - Experiment with different hyperparameters for your models. For example, try different values for the `max_depth` and `n_estimators` parameters in the Gradient Boosting and Random Forest models.
 - Use techniques like GridSearchCV or RandomizedSearchCV to search for the best hyperparameters.
- Ensemble Methods:**
 - Try ensembling different models. For example, you can combine the predictions from Gradient Boosting, Random Forest, and Bagging models to get a better prediction.
 - Experiment with different ensemble techniques like Stacking or Voting.
- Regularization:**
 - Use regularization techniques like L1 (Lasso) or L2 (Ridge) regularization to prevent overfitting and improve generalization.
- Data Preprocessing:**
 - Handle missing values appropriately. You can impute missing values with the mean, median, or mode of the column.
 - Standardize or normalize numerical features. This can help the model converge faster and improve performance.
- Cross-Validation:**
 - Use cross-validation to get a more reliable estimate of the model's performance. This can help you identify if the model is overfitting or underfitting.

Figure 5: Conversation with Chat GPT

can you give me an example of the application of this

 ChatGPT

Certainly! Let's consider an example where we want to predict the price of a car based on various features such as mileage, horsepower, and the year it was manufactured. Here's how we can apply the steps mentioned earlier:

1. **Feature Engineering:**
 - Create new features that capture interactions between existing features. For example, we can create a new feature called `hp_mileage` by multiplying the `horsepower` and `mileage` columns.
 - Consider polynomial features. We can add polynomial features like `horsepower^2` or `mileage^2` to capture non-linear relationships.
2. **Feature Selection:**
 - Remove features that are not contributing much to the model. We can use techniques like Recursive Feature Elimination (RFE) or feature importance from tree-based models to identify and remove less important features.
3. **Hyperparameter Tuning:**
 - Experiment with different hyperparameters for our models. For example, we can try different values for the `max_depth` and `n_estimators` parameters in the Gradient Boosting and Random Forest models.
4. **Ensemble Methods:**
 - We can ensemble different models. For example, we can combine the predictions from Gradient Boosting, Random Forest, and Bagging models to get a better prediction.
5. **Regularization:**
 - We can use regularization techniques like L1 (Lasso) or L2 (Ridge) regularization to prevent overfitting and improve generalization.
6. **Data Preprocessing:**
 - Handle missing values appropriately. We can impute missing values with the mean, median, or mode of the column.
 - Standardize or normalize numerical features. This can help the model converge faster and improve performance.
7. **Cross-Validation:**
 - Use cross-validation to get a more reliable estimate of the model's performance. This can help us identify if the model is overfitting or underfitting.

Figure 6: Conversation with Chat GPT

```
TypeError: Cannot perform 'xor' with a dtyped [float64] array and scalar of type [bool]]
```

 ChatGPT

The error you are encountering is due to a typo in your code. Specifically, the issue is with the use of the caret (^) operator in the following lines:

```
python
```

 Copy code

```
pc5_df['hp2'] = pc5_df['horsepower']^2
pc5_df['mile2'] = pc5_df['mileage']^2
#pc5_df['year2'] = pc5_df['year']^2
pc5_df['hwfe2'] = pc5_df['highway_fuel_economy']^2
pc5_df['volsq'] = pc5_df['car_vol']^2
pc5_df['lit2'] = pc5_df['litres']^2
```

The caret (^) operator in Python is used for bitwise XOR operations, not exponentiation. To raise a number to a power in Python, you should use the double asterisk (**) operator. Here's the corrected code:

```
python
```

 Copy code

```
pc5_df['hp2'] = pc5_df['horsepower']**2
pc5_df['mile2'] = pc5_df['mileage']**2
#pc5_df['year2'] = pc5_df['year']**2
pc5_df['hwfe2'] = pc5_df['highway_fuel_economy']**2
pc5_df['volsq'] = pc5_df['car_vol']**2
pc5_df['lit2'] = pc5_df['litres']**2
```

With these changes, your code should work without any errors.

Figure 7: Conversation with Chat GPT

 ChatGPT

Certainly! Hyperparameter tuning is a crucial step in optimizing the performance of machine learning models. In the case of Gradient Boosting Regression (GBR), the hyperparameters you mentioned are:

1. `'n_estimators'`: The number of boosting stages to perform. This is the number of trees in the forest.
2. `'learning_rate'`: The rate at which the boosting process learns. Lower values make the model more robust but require more trees.
3. `'max_depth'`: The maximum depth of the individual trees. Deeper trees can capture more complex relationships but can also lead to overfitting.

To tune these hyperparameters, you can use techniques like Grid Search or Random Search. Here's how you can do it using Grid Search:

1. Define a parameter grid containing the values you want to test for each hyperparameter.
2. Use `'GridSearchCV'` to perform a grid search over the parameter grid.
3. Fit the `'GridSearchCV'` object to your training data.
4. Extract the best hyperparameters and use them to train your final model.

Here's an example of how you can do this:

```
python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 500, 1000],
    'learning_rate': [0.001, 0.01, 0.1],
    'max_depth': [3, 5, 7]
}

# Initialize the GBR model
boost_car = GBR(random_state=0)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=boost_car, param_grid=param_grid, cv=5, n_jobs=-1)

# Fit the GridSearchCV object to the training data
grid_search.fit(x_train_scaled, y_train)
```



```
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=boost_car, param_grid=param_grid, cv=5, n_jobs=-1)

# Fit the GridSearchCV object to the training data
grid_search.fit(x_train_scaled, y_train)

# Extract the best hyperparameters
best_params = grid_search.best_params_
print(f"Best hyperparameters: {best_params}")

# Train the final model with the best hyperparameters
best_boost_car = GBR(**best_params, random_state=0)
best_boost_car.fit(x_train_scaled, y_train)

# Make predictions on the test set
y_hat_boost = best_boost_car.predict(x_test_scaled)

# Evaluate the model
r_squared = r2_score(y_test, y_hat_boost)
print(f"R-squared: {r_squared}")

In this example, we define a parameter grid with different values for 'n_estimators', 'learning_rate', and 'max_depth'. We use 'GridSearchCV' with a 5-fold cross-validation to search for the best hyperparameters. The 'n_jobs=-1' argument allows the search to use all available CPU cores for parallel processing. After fitting the 'GridSearchCV' object to the training data, we extract the best hyperparameters and train the final model using these values. Finally, we evaluate the model on the test set using the R-squared metric.
```

Figure 8: Conversation with Chat GPT

ECON 626 Prediction Competition 5 Code

Objective: Utilize ensemble machine learning algorithms (Bagging, Random Forest, Boosting) to train a model that predicts the natural logarithm of car price.

Importing Librarys

```
In [ ]: import numpy as np
import pandas as pd
from pandas import get_dummies
import re
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model # Linear regression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import \
(RandomForestRegressor as RF,
GradientBoostingRegressor as GBR)
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV, ShuffleSplit
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVR, LinearSVR
```

Importing data

```
In [ ]: small_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Co
small_df= pd.read_csv(small_data_path)
#create a dataframe for our smaller dataset

large_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Co
large_df = pd.read_csv(large_data_path)
#create a dataframe for our larger dataset

pc5_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Comp
pc5_df = pd.read_csv(pc5_data_path)

test_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Com
test_df = pd.read_csv(test_data_path)
#create a dataframe for our larger dataset
```

```
total_df = pd.concat([small_df, pc5_df], axis = 0)
#create a dataframe containing both small and pc5 df

testing_df_path1 = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Competition 5/test.csv"
testing_df1 = pd.read_csv(testing_df_path1)

testing_df_path2 = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Competition 5/test2.csv"
testing_df2 = pd.read_csv(testing_df_path2)
```

Inspect data

```
In [ ]: #function:
def inspect_dataset(dataset):
    # Print the head of the dataset
    print("Head of the dataset:")
    print(dataset.head())
    print("\n")

    # Print the info of the dataset
    print("Info of the dataset:")
    print(dataset.info())
    print("\n")

    # Print the shape of the dataset
    print("Shape of the dataset:")
    print(dataset.shape)
    print("\n")

    # Print value counts for columns of type object
    object_columns = dataset.select_dtypes(include=['object']).columns
    for column in object_columns:
        print(f"Value counts for column '{column}':")
        print(dataset[column].value_counts())
        print("\n")
```

```
In [ ]: pc5_df.head()
```

Out []:

	price	back_legroom	body_type	engine_displacement	exterior_color	fuel_type
0	12500.0	39.9 in	SUV / Crossover	2400	Silver Ice Metallic	Gasoline
1	11971.0	35.4 in	Hatchback	2000	Dark Gray Metallic	Gasoline
2	8750.0	37.4 in	Sedan	1800	Silver	Gasoline
3	7000.0	31.7 in	Hatchback	1400	-Verde Oliva	Gasoline
4	11999.0	35.1 in	SUV / Crossover	2400	Black	Gasoline

In []: `pc5_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   price            100000 non-null   float64
 1   back_legroom     100000 non-null   object 
 2   body_type        100000 non-null   object 
 3   engine_displacement 100000 non-null   int64  
 4   exterior_color   97228 non-null   object 
 5   fuel_type         100000 non-null   object 
 6   height            100000 non-null   object 
 7   highway_fuel_economy 100000 non-null   int64  
 8   horsepower        100000 non-null   int64  
 9   latitude          100000 non-null   float64
 10  length            100000 non-null   object 
 11  listed_date       100000 non-null   object 
 12  longitude          100000 non-null   float64
 13  mileage            100000 non-null   int64  
 14  wheel_system      100000 non-null   object 
 15  wheelbase          100000 non-null   object 
 16  width              100000 non-null   object 
 17  year               100000 non-null   int64  
dtypes: float64(3), int64(5), object(10)
memory usage: 13.7+ MB
```

In []: `pc5_df.isna().sum()`

```
Out[ ]: price          0
back_legroom      0
body_type         0
engine_displacement 0
exterior_color    2772
fuel_type          0
height             0
highway_fuel_economy 0
horsepower         0
latitude           0
length              0
listed_date        0
longitude           0
mileage             0
wheel_system        0
wheelbase           0
width               0
year                0
dtype: int64
```

```
In [ ]: pc5_df.shape
```

```
Out[ ]: (100000, 18)
```

```
In [ ]: # Assuming pc5_df is a pandas DataFrame
propane_count = pc5_df['fuel_type'].value_counts().get('Propane', 0)

print("Number of 'Propane' entries in pc5_df:", propane_count)
```

Number of 'Propane' entries in pc5_df: 2

```
In [ ]: # Assuming pc5_df is a pandas DataFrame
pc5_df['fuel_type'] = pc5_df['fuel_type'].replace('Propane', 'Gasoline')

# Now you can count the 'Gasoline' entries
gasoline_count = pc5_df['fuel_type'].value_counts().get('Gasoline', 0)

print("Number of 'Gasoline' entries in pc5_df:", gasoline_count)
```

Number of 'Gasoline' entries in pc5_df: 92535

```
In [ ]: # Assuming pc5_df is a pandas DataFrame
propane_count = pc5_df['fuel_type'].value_counts().get('Propane', 0)

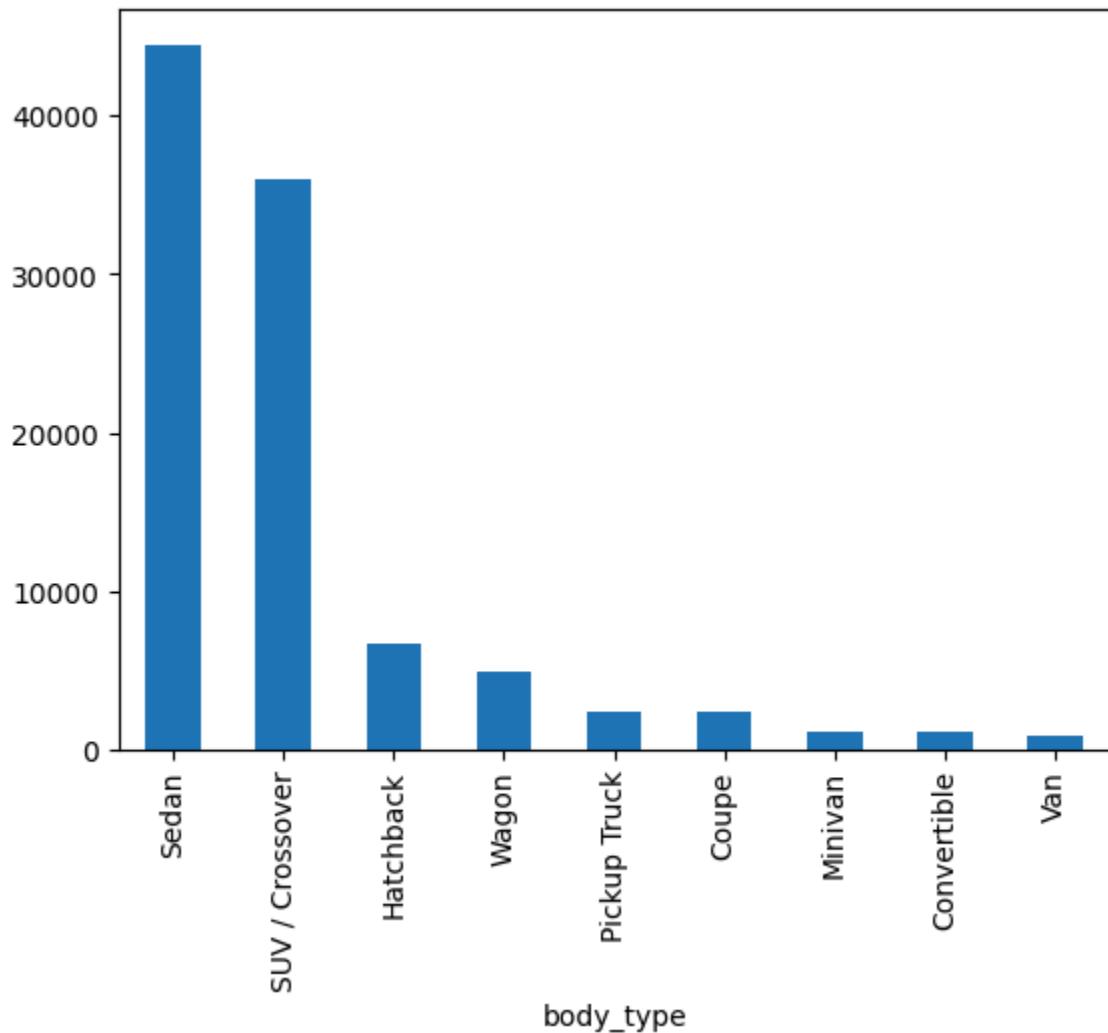
print("Number of 'Propane' entries in pc5_df:", propane_count)
```

Number of 'Propane' entries in pc5_df: 0

Data Visualizations

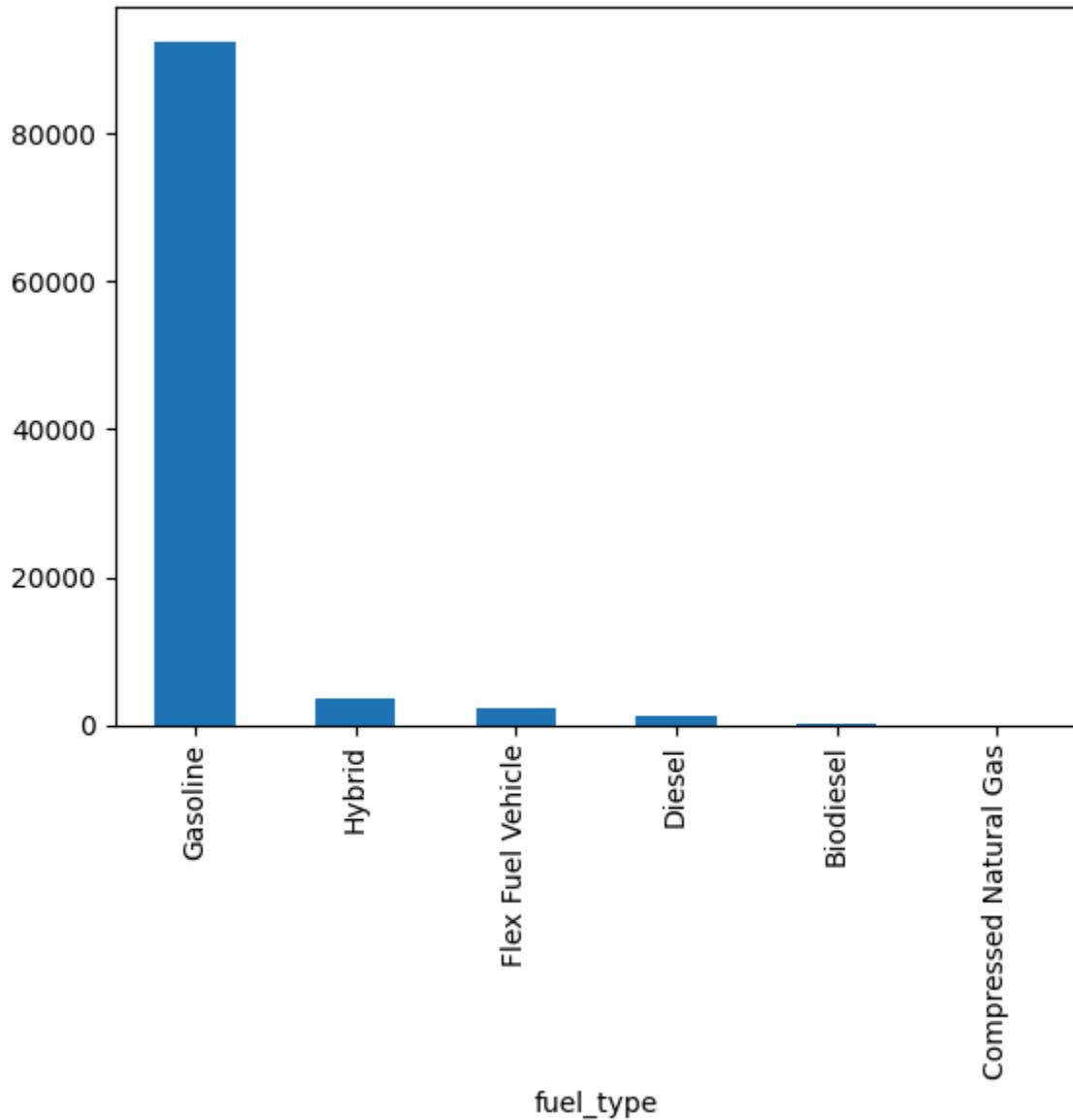
```
In [ ]: pc5_df['body_type'].value_counts().plot(kind='bar')
```

```
Out[ ]: <Axes: xlabel='body_type'>
```



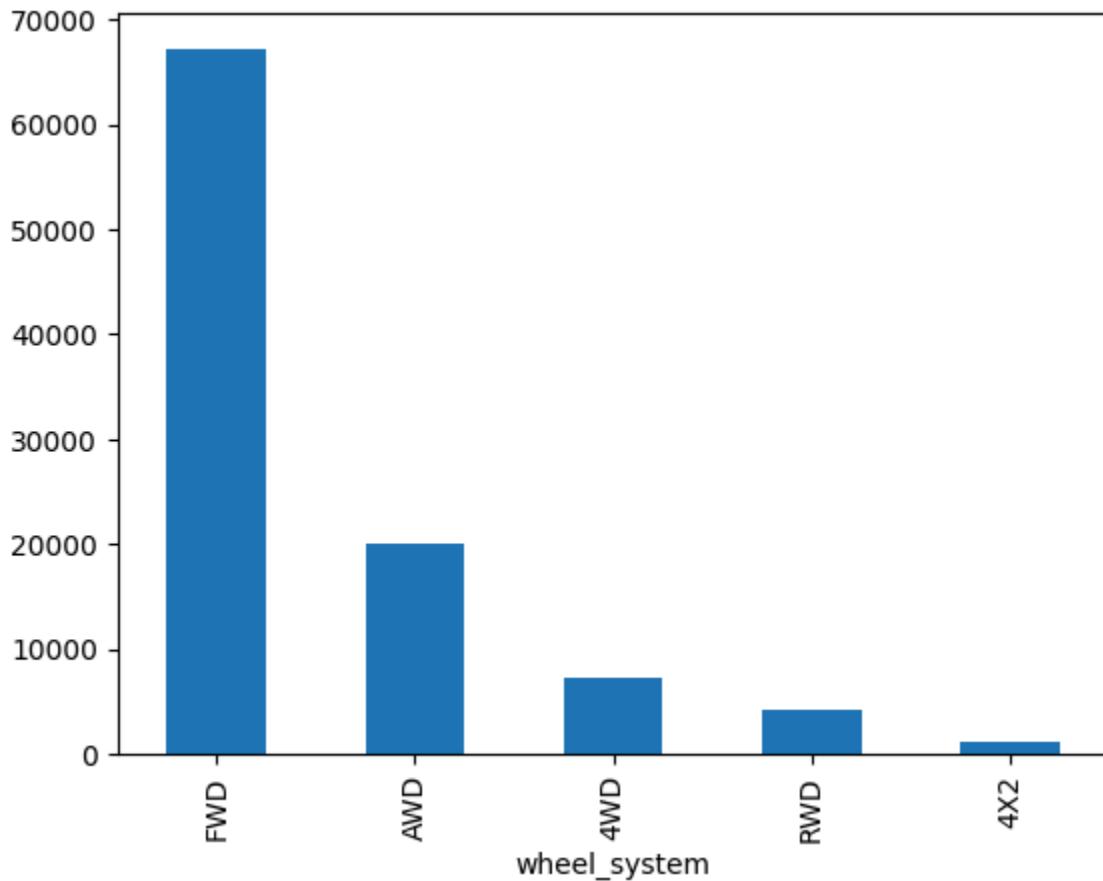
```
In [ ]: pc5_df['fuel_type'].value_counts().plot(kind='bar')
```

```
Out[ ]: <Axes: xlabel='fuel_type'>
```



```
In [ ]: pc5_df['wheel_system'].value_counts().plot(kind='bar')
```

```
Out[ ]: <Axes: xlabel='wheel_system'>
```



```
In [ ]: numerical_columns = small_df.select_dtypes(include=np.number).columns.tolist()
num_drop_list = 'latitude', 'price', 'longitude'
numerical_columns = list(set(numerical_columns) - set(num_drop_list))
```

```
In [ ]: categorical_columns = small_df.select_dtypes(exclude=np.number).columns.tolist()
categorical_columns

cat_drop_list = 'back_legroom', 'height', 'length', 'listed_date',
'wheelbase', 'width', 'exterior_color'
categorical_columns = list(set(categorical_columns) - set(cat_drop_list))
```

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

og_cols = ['price', 'engine_displacement', 'highway_fuel_economy',
           'horsepower', 'mileage', 'year']
num_cols = len(og_cols)

# Calculate the number of rows and columns needed for subplots
num_rows = (num_cols + 2) // 3 # Ceiling division to ensure we have enough
num_cols = min(num_cols, 3) # Limit the number of columns to 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))

for i, col in enumerate(og_cols):
```

```

row_idx = i // num_cols
col_idx = i % num_cols

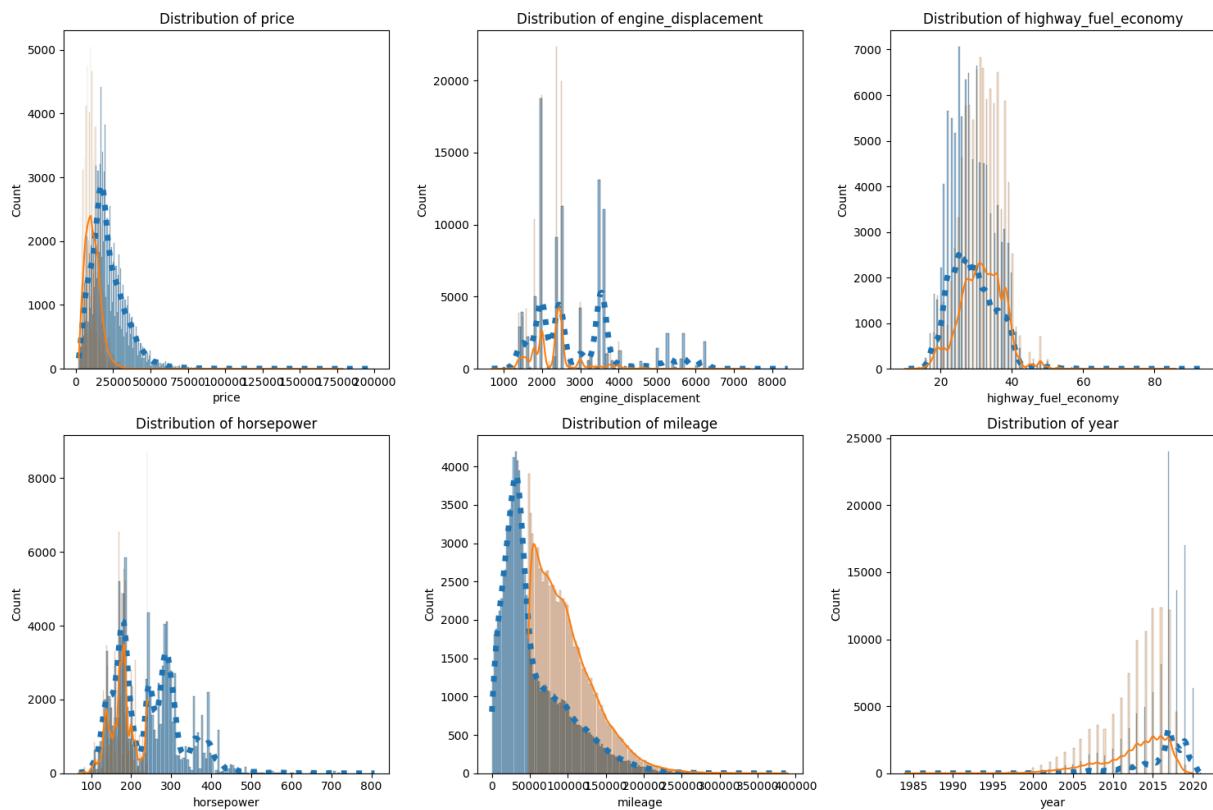
if num_rows == 1:
    ax = axes[col_idx]
else:
    ax = axes[row_idx, col_idx]

sns.histplot(small_df[col], ax=ax, kde=True, line_kws={
    'lw': 5, 'ls': ':'})
sns.histplot(pc5_df[col], ax=ax, kde=True, alpha=0.25)
#sns.histplot(large_df[col], ax=ax, kde=True, alpha=0.25)

ax.set_title('Distribution of ' + col)

# Adjust layout to prevent overlap of subplots
plt.tight_layout()
plt.show()

```



```

In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

og_cat_cols = ['body_type', 'fuel_type', 'wheel_system']
num_cols = len(og_cat_cols)

# Calculate the number of rows and columns needed for subplots
num_rows = (num_cols + 2) // 3 # Ceiling division to ensure we have enough
num_cols = min(num_cols, 3) # Limit the number of columns to 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))

```

```

for i, col in enumerate(og_cat_cols):
    row_idx = i // num_cols
    col_idx = i % num_cols

    if num_rows == 1:
        ax = axes[col_idx]
    else:
        ax = axes[row_idx, col_idx]

    sns.histplot(small_df[col], ax=ax, kde=True, line_kws={
        'lw': 5, 'ls': ':'})
    sns.histplot(pc5_df[col], ax=ax, kde=True, alpha=0.25)

    ax.set_title('Distribution of ' + col)

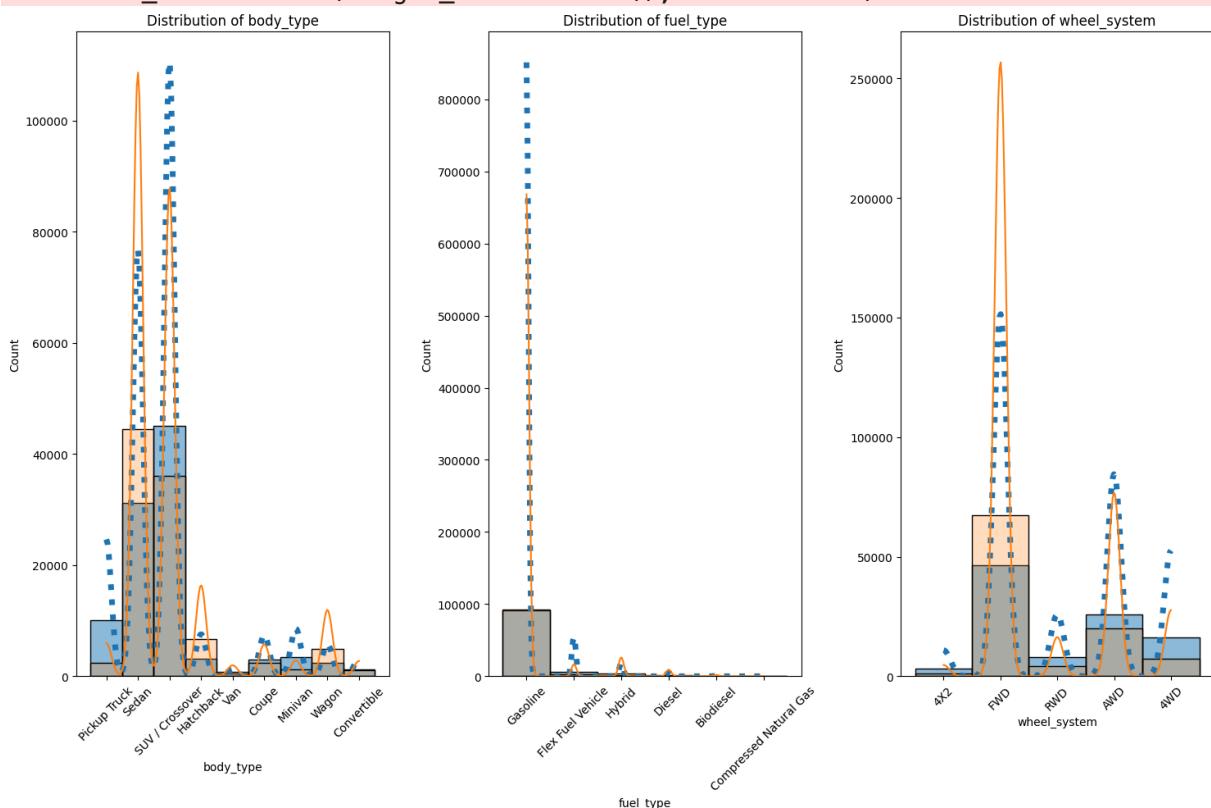
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
# Adjust layout to prevent overlap of subplots
plt.tight_layout()
plt.show()

```

```

/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_78887/721232414.py:28: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_78887/721232414.py:28: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_78887/721232414.py:28: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

```



```
In [ ]: og_cols = ['price', 'engine_displacement', 'highway_fuel_economy',
                 'horsepower', 'mileage', 'year', 'body_type',
                 'fuel_type', 'wheel_system']
num_cols = len(og_cols)

# Calculate the number of rows and columns needed for subplots
num_rows = (num_cols + 2) // 3 # Ceiling division to ensure we have enough
num_cols = min(num_cols, 3) # Limit the number of columns to 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(25, 20))

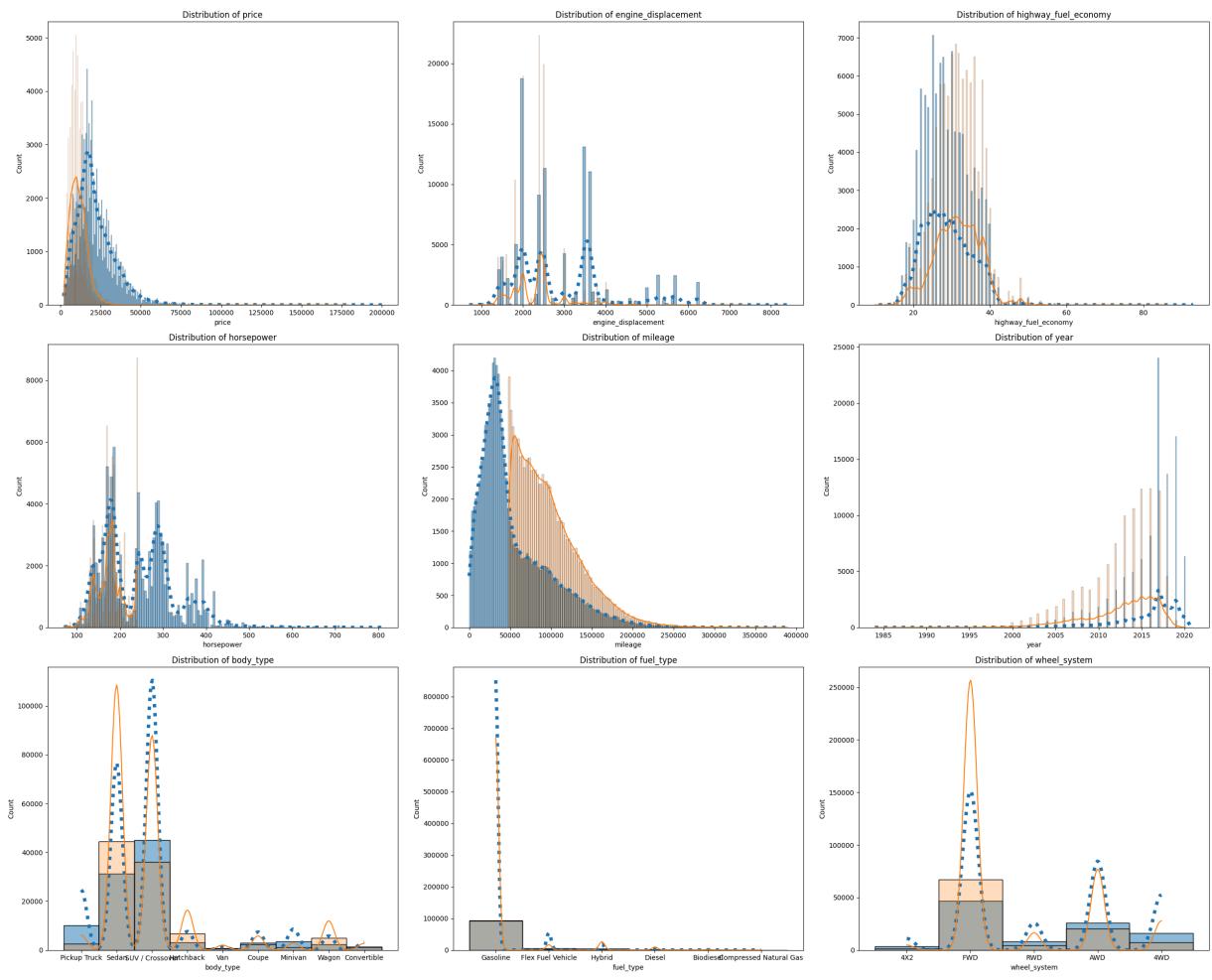
for i, col in enumerate(og_cols):
    row_idx = i // num_cols
    col_idx = i % num_cols

    if num_rows == 1:
        ax = axes[col_idx]
    else:
        ax = axes[row_idx, col_idx]

    sns.histplot(small_df[col], ax=ax, kde=True, line_kws={
        'lw': 5, 'ls': ':'}) # Add KDE plot
    sns.histplot(pc5_df[col], ax=ax, kde=True, alpha=0.25)
#sns.histplot(large_df[col], ax=ax, kde=True, alpha=0.25)

    ax.set_title('Distribution of ' + col)

# Adjust layout to prevent overlap of subplots
plt.tight_layout()
plt.show()
```



Since we see that the distributions of the datasets are different I will be taking a random sample from the total dataframe to get an average distribution between both of the distributions.

```
In [ ]: small_df['fuel_type'].unique()
```

```
Out[ ]: array(['Gasoline', 'Flex Fuel Vehicle', 'Hybrid', 'Diesel', 'Biodiesel',
   'Compressed Natural Gas'], dtype=object)
```

```
In [ ]: pc5_df['fuel_type'].unique()
```

```
Out[ ]: array(['Gasoline', 'Flex Fuel Vehicle', 'Diesel', 'Hybrid', 'Biodiesel',
   'Compressed Natural Gas'], dtype=object)
```

```
In [ ]: # total_df = total_df[total_df['body_type'] != 'Minivan']

# total_df = total_df[~total_df
#                     ['fuel_type'].isin(
#                         ['Compressed Natural Gas', 'Diesel',
#                          'Flex Fuel Vehicle', 'Propane'])]
```

```
In [ ]: total_df = pd.concat([small_df, pc5_df], axis = 0)
# create a dataframe containing both small and pc5 df
```

```
In [ ]: total_df['fuel_type'].unique()
```

```
Out[ ]: array(['Gasoline', 'Flex Fuel Vehicle', 'Hybrid', 'Diesel', 'Biodiesel',
   'Compressed Natural Gas'], dtype=object)
```

```
In [ ]: subset_df = total_df.sample(n=125000, random_state=4)
```

```
In [ ]: propane_count = subset_df['fuel_type'].eq(
   'propane').sum()
```

```
In [ ]: propane_count
```

```
Out[ ]: 0
```

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

og_cols = ['price', 'engine_displacement',
           'highway_fuel_economy', 'horsepower', 'mileage',
           'year']
num_cols = len(og_cols)

# Calculate the number of rows and columns needed for subplots
num_rows = (num_cols + 2) // 3 # Ceiling division to ensure we have enough
num_cols = min(num_cols, 3) # Limit the number of columns to 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))

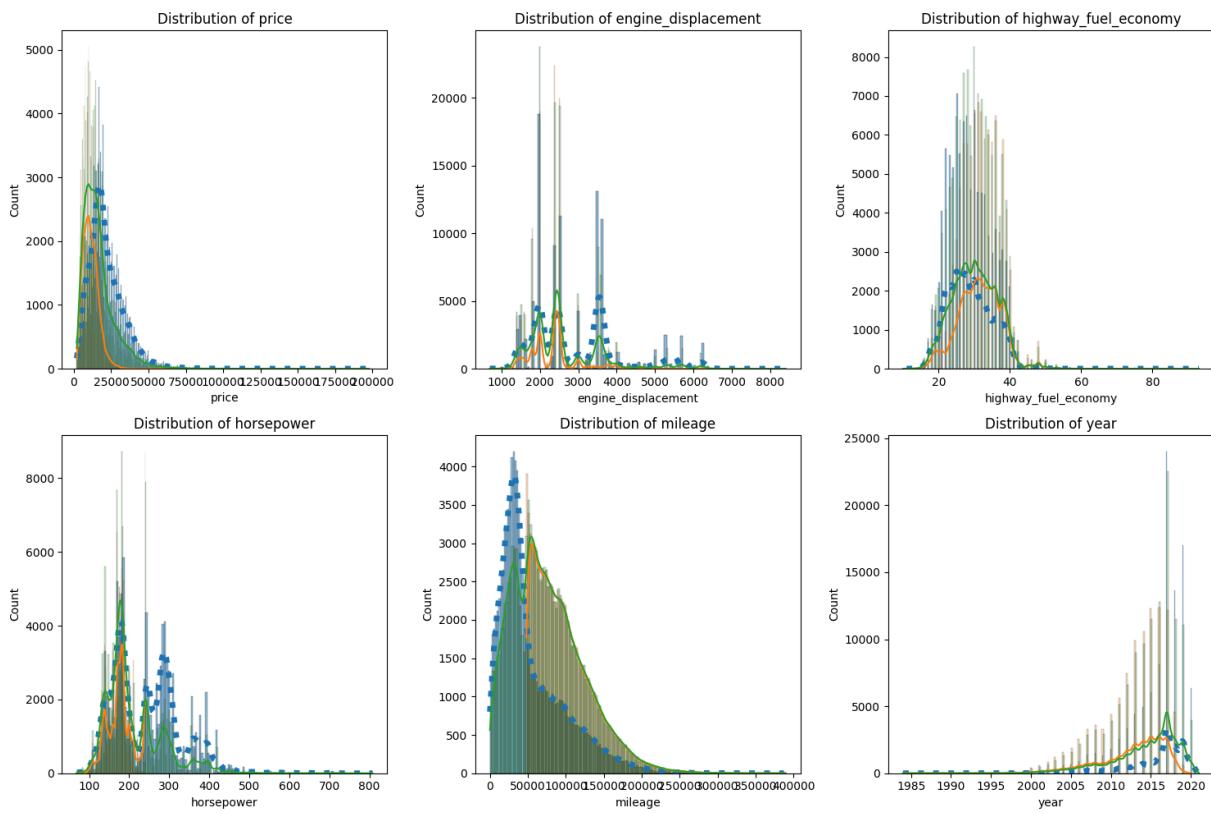
for i, col in enumerate(og_cols):
    row_idx = i // num_cols
    col_idx = i % num_cols

    if num_rows == 1:
        ax = axes[col_idx]
    else:
        ax = axes[row_idx, col_idx]

    sns.histplot(small_df[col], ax=ax, kde=True, line_kws={
        'lw': 5, 'ls': ':'})
    sns.histplot(pc5_df[col], ax=ax, kde=True, alpha=0.25)
    sns.histplot(subset_df[col], ax=ax, kde=True, alpha=0.25)

    ax.set_title('Distribution of ' + col)

# Adjust layout to prevent overlap of subplots
plt.tight_layout()
plt.show()
```



```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

og_cat_cols = ['body_type', 'fuel_type', 'wheel_system']
num_cols = len(og_cat_cols)

# Calculate the number of rows and columns needed for subplots
num_rows = (num_cols + 2) // 3 # Ceiling division to ensure we have enough
num_cols = min(num_cols, 3) # Limit the number of columns to 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))

for i, col in enumerate(og_cat_cols):
    row_idx = i // num_cols
    col_idx = i % num_cols

    if num_rows == 1:
        ax = axes[col_idx]
    else:
        ax = axes[row_idx, col_idx]

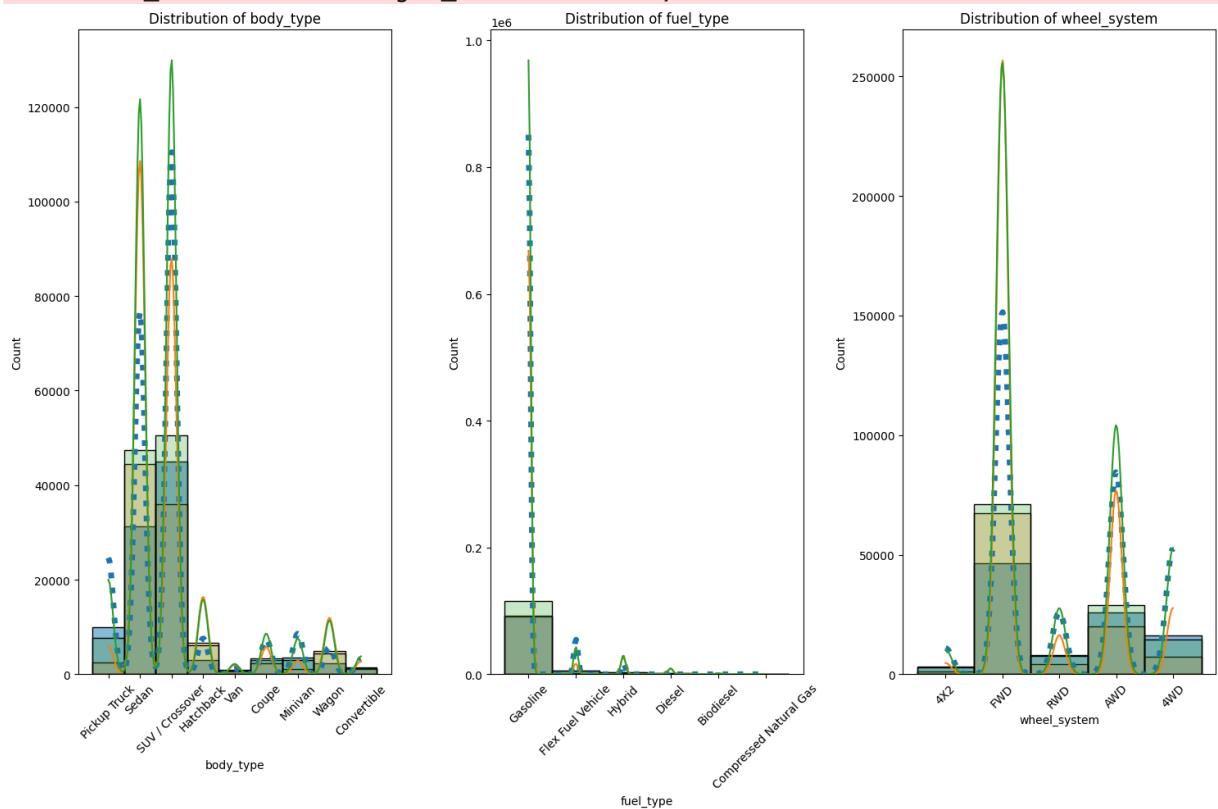
    sns.histplot(small_df[col], ax=ax, kde=True, line_kws={
        'lw': 5, 'ls': ':'})
    sns.histplot(pc5_df[col], ax=ax, kde=True, alpha=0.25)
    sns.histplot(subset_df[col], ax=ax, kde=True, alpha=0.25)

    ax.set_title('Distribution of ' + col)

    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
# Adjust layout to prevent overlap of subplots
```

```
plt.tight_layout()
plt.show()
```

```
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_78887/2715228621.py:29: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_78887/2715228621.py:29: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_78887/2715228621.py:29: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```



Here we see that we have successfully found a distribution that represents both datasets.

Data Preprocessing

```
In [ ]: #Creating a function to preprocess the data
def prep_data(dataset):

    dataset_cols = dataset.columns
    if 'price' in dataset_cols:
        dataset['log_price'] = np.log(dataset['price'])
    else:
        pass
```

```
#Feature Engineering

dataset['listed_date'] = pd.to_datetime(dataset['listed_date'])
listed_year = dataset['listed_date'].dt.year
dataset['age_at_listing'] = listed_year - dataset['year']

dataset['length'] = pd.to_numeric(dataset['length'].astype(str).str.replace('cm', ''))
dataset['length'] = dataset['length'].fillna(dataset.groupby('body_type')['length'].mean())

dataset['width'] = pd.to_numeric(dataset['width'].astype(str).str.replace('cm', ''))
dataset['width'] = dataset['width'].fillna(dataset.groupby('body_type')['width'].mean())

dataset['height'] = pd.to_numeric(dataset['height'].astype(str).str.replace('cm', ''))
dataset['height'] = dataset['height'].fillna(dataset.groupby('body_type')['height'].mean())

# dataset['wheelbase'] = pd.to_numeric(dataset['wheelbase'].astype(str).str.replace('cm', ''))
# dataset['wheelbase'] = dataset['wheelbase'].fillna(dataset.groupby('body_type')['wheelbase'].mean())

dataset["car_vol"] = dataset["length"] * dataset["width"] * dataset["height"]

dataset['listed_date'] = pd.to_datetime(dataset['listed_date'])
listed_year = dataset['listed_date'].dt.year
dataset['age_at_listing'] = listed_year - dataset['year']

dataset['litres'] = (dataset['engine_displacement']/1000).astype(float)
dataset = dataset.drop(['engine_displacement'], axis=1)

col_encode = [ 'body_type', 'fuel_type', 'wheel_system']

for col in col_encode:
    dataset = pd.concat([dataset, pd.get_dummies(dataset[col], prefix=col)])
# le = LabelEncoder()
# for col in col_encode:
#     new_col = col+'_enc'
#     dataset[new_col] = le.fit_transform(dataset[col])

dataset['hp_mile'] = dataset['horsepower'] * dataset['mileage']
dataset['hp_hwfe'] = dataset['horsepower'] * dataset['highway_fuel_economy']
dataset['hp_year'] = dataset['horsepower'] * dataset['year']
dataset['hp_lit'] = dataset['horsepower'] * dataset['litres']
dataset['hp_vol'] = dataset['horsepower'] * dataset['car_vol']
dataset['hp2'] = dataset['horsepower']**2
dataset['mile2'] = dataset['mileage']**2
dataset['age2'] = dataset['age_at_listing']**2

# fuel_type_map = {'Gasoline': 1, 'Diesel': 0, 'Flex Fuel Vehicle' : 0,
# # Apply the mapping to the 'fuel_type' column
# dataset['fuel_type_binary'] = dataset['fuel_type'].map(fuel_type_map)

drop = ['price', 'back_legroom', 'wheelbase', 'latitude', 'longitude',
        'listed_date', 'exterior_color']
for col in drop:
    dataset = dataset.drop([col], axis=1)
```

```
drop_enc= ['body_type', 'fuel_type', 'wheel_system']
for col in drop_enc:
    dataset = dataset.drop([col], axis=1)
return dataset
```

In []: small_df = prep_data(small_df)

In []: pc5_df = prep_data(pc5_df)

In []: subset_df = prep_data(subset_df)

In []: testing_df2 = prep_data(testing_df2)

In []: small_df.head()

Out[]:

	height	highway_fuel_economy	horsepower	length	mileage	width	year	log_pri	
0	75.6		18	381	228.7	167184	79.9	2008	9.8252
1	57.3		38	132	182.6	29451	69.9	2016	9.7063
2	58.2		34	248	194.6	14984	83.7	2019	10.5950
3	67.8		28	260	192.8	15697	75.4	2020	10.0836
4	56.3		33	255	184.5	6907	79.4	2020	10.5332

5 rows × 36 columns

In []: testing_df2.head()

Out[]:

	height	highway_fuel_economy	horsepower	length	mileage	width	year	log_pri	
0	74.0		20	420	229.5	33054	80.0	2017	10.7666
1	70.6		24	278	212.3	18740	75.2	2019	10.5184
2	64.1		29	237	181.1	7447	83.5	2020	10.6197
3	66.3		30	240	178.1	99590	81.8	2014	9.3413
4	48.6		26	460	177.9	18599	77.4	2017	11.0205

5 rows × 36 columns

In []: subset_df.head()

Out []:

	height	highway_fuel_economy	horsepower	length	mileage	width	year	lo
72469	58.1	36	185	191.1	21754	73.4	2017	9
32789	67.8	26	171	174.7	164892	81.3	2010	8.
72108	56.6	26	240	202.6	158478	74.2	2001	8
80232	78.4	16	411	232.1	109695	97.0	2014	10
46040	56.8	29	136	175.2	157838	66.7	2005	8

5 rows × 36 columns

```
In [ ]: def split_data(dataset, random_st):
    global x_train, x_val_test, y_train, y_val_test, x_val, x_test, y_val, y_test
    #Must make the variables global to
    #access the variables outside of the function

    columns_x = list(dataset.columns)
    if 'log_price' in columns_x:
        columns_x.remove('log_price')
    else:
        pass
    x_train, x_val_test, y_train, y_val_test = train_test_split(
        dataset[columns_x], dataset['log_price'], test_size=0.2,
        random_state=random_st)
    x_val, x_test, y_val, y_test = train_test_split(
        x_val_test, y_val_test, test_size=0.5, random_state=123)

    #return x_train, x_val, x_test, y_train, y_val, y_test
```

```
In [ ]: def split_data_Q2(dataset, random_st):
    global x_train, y_train, x_test, y_test

    columns_x = list(dataset.columns)
    if 'log_price' in columns_x:
        columns_x.remove('log_price')
    else:
        pass
    x_train, x_test, y_train, y_test = train_test_split(
        dataset[columns_x], dataset['log_price'], test_size=0.2,
        random_state=random_st)

    #return x_train, x_val, x_test, y_train, y_val, y_test
```

Small Data (PC3 data) modeling

```
In [ ]: split_data_Q2(small_df, 4)
```

```
In [ ]: print(x_train.shape, x_test.shape)
```

```
#print(x_train.shape, x_test.shape)
```

```
(80000, 35) (20000, 35)
```

```
In [ ]: y_test_pc3 = y_test
```

```
In [ ]: x_train.head()
```

```
Out[ ]:
```

	height	highway_fuel_economy	horsepower	length	mileage	width	year	ag
8446	74.4		21	355	204.0	17084	80.5	2020
19150	55.3		30	354	186.8	13810	79.6	2018
46825	66.8		29	197	183.7	23006	72.4	2017
95011	64.8		31	180	173.0	24248	80.0	2018
68234	73.7		18	403	230.2	171700	80.0	2012

5 rows × 35 columns

```
In [ ]: standard_scaler = preprocessing.StandardScaler()

standard_scaler.fit(x_train)
# transform
x_train_s_scaled = standard_scaler.transform(x_train)
#x_val_s_scaled = standard_scaler.transform(x_val)
x_test_s_scaled_pc3 = standard_scaler.transform(x_test)
```

Boosting

```
In [ ]: boost_car = GBR(n_estimators=1000,
learning_rate=0.2,
max_depth=3,
random_state=4)
boost_car.fit(x_train_s_scaled ,y_train)
```

```
Out[ ]:
```

```
▼ GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.2, n_estimators=1000, random_state=4)
```

```
In [ ]: y_hat_test_boost = boost_car.predict(x_test_s_scaled_pc3)
```

```
In [ ]: print('MSE on PC3 small test set:', mean_squared_error(
    y_test_pc3, y_hat_test_boost))
```

```
print('R squared on PC3 small test set:', r2_score(
    y_test_pc3, y_hat_test_boost))
```

MSE on PC3 small test set: 0.017723462636129235
R squared on PC3 small test set: 0.9462717752640761

PC5 Data modeling

```
In [ ]: split_data_Q2(pc5_df, 4)
```

```
In [ ]: print(x_train.shape, x_test.shape)
#print(x_train.shape, x_test.shape)

(80000, 35) (20000, 35)
```

```
In [ ]: y_test_pc5 = y_test
```

```
In [ ]: x_train.head()
```

```
Out[ ]:
```

	height	highway_fuel_economy	horsepower	length	mileage	width	year	ag
8446	63.0		30	130	163.0	76949	70.9	2016
19150	57.2		39	150	183.3	49185	70.0	2016
46825	67.1		31	176	179.9	87863	72.6	2015
95011	69.4		24	241	189.8	110381	84.8	2014
68234	59.6		43	99	162.4	68234	68.6	2018

5 rows × 35 columns

```
In [ ]: standard_scaler = preprocessing.StandardScaler()
```

```
standard_scaler.fit(x_train)
# transform
x_train_s_scaled = standard_scaler.transform(x_train)
##x_val_s_scaled = standard_scaler.transform(x_val)
x_test_s_scaled_pc5 = standard_scaler.transform(x_test)
```

```
In [ ]: y_hat_test_boost = boost_car.predict(x_test_s_scaled_pc5)
```

```
#print('MSE on training set:', mean_squared_error(y_train, y_hat_train_boost))
#print('MSE on validation set:', mean_squared_error(y_val, y_hat_val_boost))
print('MSE on PC5 test set:', mean_squared_error(
    y_test_pc5, y_hat_test_boost))

#print('R squared on training set:', r2_score(y_train, y_hat_train_boost))
#print('R squared on validation set:', r2_score(y_val, y_hat_val_boost))
print('R squared on PC5 small test set:', r2_score(
    y_test_pc5, y_hat_test_boost))
```

MSE on PC5 test set: 0.5952486893540333

R squared on PC5 small test set: -1.556231728300666

Boosting

```
In [ ]: boost_car = GBR(n_estimators=1000,
learning_rate=0.2,
max_depth=3,
random_state=4)
boost_car.fit(x_train_s_scaled ,y_train)
```

```
Out[ ]: GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.2, n_estimators=1000, random_state=4)
```

```
In [ ]: y_hat_test_boost = boost_car.predict(x_test_s_scaled_pc5)
```

```
In [ ]: print('MSE on test set:', mean_squared_error(y_test, y_hat_test_boost))
print('R squared on test set:', r2_score(y_test, y_hat_test_boost))
```

MSE on test set: 0.025015426078181816
R squared on test set: 0.8925739325736242

```
In [ ]: y_hat_test_boost = boost_car.predict(x_test_s_scaled_pc3)
```

```
In [ ]: print('MSE on PC3 small test set:', mean_squared_error(
y_test_pc3, y_hat_test_boost))

print('R squared on PC3 small test set:', r2_score(
y_test_pc3, y_hat_test_boost))
```

MSE on PC3 small test set: 0.39185952766609844
R squared on PC3 small test set: -0.18791215913073045

Subset Data modeling

```
In [ ]: split_data(subset_df, 4)
```

```
In [ ]: print(x_train.shape, x_val.shape, x_test.shape)
#print(x_train.shape, x_test.shape)
```

(100000, 35) (12500, 35) (12500, 35)

```
In [ ]: x_train.head()
```

Out[]:	height	highway_fuel_economy	horsepower	length	mileage	width	year	ac
	64527	69.3	25	295	189.8	19959	84.8	2019
	45111	68.2	32	170	181.5	71166	80.0	2017
	3175	53.5	36	140	174.8	118873	68.9	2008
	35988	57.6	36	160	193.8	39700	73.0	2018
	81156	55.9	30	272	182.8	50118	71.1	2017

5 rows × 35 columns

```
In [ ]: x_train.isna().sum()
```

```
Out[ ]: height          0
highway_fuel_economy  0
horsepower           0
length               0
mileage              0
width                0
year                 0
age_at_listing       0
car_vol              0
litres               0
body_type_Coupe     0
body_type_Hatchback  0
body_type_Minivan   0
body_type_Pickup Truck 0
body_type_SUV / Crossover 0
body_type_Sedan     0
body_type_Van       0
body_type_Wagon     0
fuel_type_Compressed Natural Gas 0
fuel_type_Diesel    0
fuel_type_Flex Fuel Vehicle 0
fuel_type_Gasoline   0
fuel_type_Hybrid    0
wheel_system_4X2    0
wheel_system_AWD    0
wheel_system_FWD    0
wheel_system_RWD    0
hp_mile              0
hp_hwfe              0
hp_year              0
hp_lit               0
hp_vol               0
hp2                  0
mile2                0
age2                 0
dtype: int64
```

```
In [ ]: min_max_scaler = preprocessing.MinMaxScaler()

min_max_scaler.fit(x_train)
# transform
x_train_mm_scaled = min_max_scaler.transform(x_train)
x_val_mm_scaled = min_max_scaler.transform(x_val)
x_test_mm_scaled = min_max_scaler.transform(x_test)
```

```
In [ ]: standard_scaler = preprocessing.StandardScaler()

standard_scaler.fit(x_train)
# transform
x_train_s_scaled = standard_scaler.transform(x_train)
x_val_s_scaled = standard_scaler.transform(x_val)
x_test_s_scaled = standard_scaler.transform(x_test)
```

```
In [ ]: robust_scaler = preprocessing.RobustScaler()

robust_scaler.fit(x_train)
# transform
x_train_r_scaled = robust_scaler.transform(x_train)
x_val_r_scaled = robust_scaler.transform(x_val)
x_test_r_scaled = robust_scaler.transform(x_test)
```

Boosting

```
In [ ]: boost_car_subset = GBR(n_estimators=1000,
learning_rate=0.2,
max_depth=3,
random_state=4)
boost_car_subset.fit(x_train_s_scaled , y_train)
```

Out[]:

```
▼ GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.2, n_estimators=1000, random_state=4)
```

```
In [ ]: y_hat_train_boost = boost_car_subset.predict(x_train_s_scaled)
y_hat_val_boost = boost_car_subset.predict(x_val_s_scaled)
y_hat_test_boost = boost_car_subset.predict(x_test_s_scaled)
```

```
In [ ]: print('MSE on training set:', mean_squared_error(
    y_train, y_hat_train_boost))
print('MSE on validation set:', mean_squared_error(
    y_val, y_hat_val_boost))
print('MSE on test set:', mean_squared_error(
    y_test, y_hat_test_boost))

print('R squared on training set:', r2_score(
    y_train, y_hat_train_boost))
print('R squared on validation set:', r2_score(
    y_val, y_hat_val_boost))
```

```
print('R squared on test set:', r2_score(
    y_test, y_hat_test_boost))
```

```
MSE on training set: 0.01862204679625535
MSE on validation set: 0.021658473867797637
MSE on test set: 0.022162097797194975
R squared on training set: 0.9503395127013897
R squared on validation set: 0.9408444726544455
R squared on test set: 0.9418609179312111
```

```
In [ ]: r2_boost = r2_score(y_test, y_hat_test_boost)
```

Boosting Hyperparameter tuning

```
In [ ]: # Define the parameter grid
param_grid = {
    'n_estimators': [500, 1000],
    'learning_rate': [0.2, 0.4],
    'max_depth': [3]
}

# Initialize the GBR model
boost_car = GBR(random_state=4)

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=boost_car, param_grid=param_grid, cv=4, n_jobs=-1, verbose=2)

# Fit the GridSearchCV object to the training data
grid_search.fit(x_val_s_scaled, y_val)

# Extract the best hyperparameters
best_params = grid_search.best_params_
print(f"Best hyperparameters: {best_params}")

# Train the final model with the best hyperparameters
best_boost_car = GBR(**best_params, random_state=0)
best_boost_car.fit(x_train_s_scaled, y_train)

# Make predictions on the test set
y_hat_boost = best_boost_car.predict(x_test_s_scaled)

# Evaluate the model
r_squared = r2_score(y_test, y_hat_boost)
print(f"R-squared: {r_squared}")
```

```
Fitting 4 folds for each of 4 candidates, totalling 16 fits
[CV] END ...learning_rate=0.2, max_depth=3, n_estimators=500; total time= 1
7.0s
[CV] END ...learning_rate=0.2, max_depth=3, n_estimators=500; total time= 1
7.4s
[CV] END ...learning_rate=0.2, max_depth=3, n_estimators=500; total time= 1
7.6s
[CV] END ...learning_rate=0.2, max_depth=3, n_estimators=500; total time= 1
7.6s
[CV] END ...learning_rate=0.4, max_depth=3, n_estimators=500; total time= 1
8.3s
[CV] END ...learning_rate=0.4, max_depth=3, n_estimators=500; total time= 1
8.8s
[CV] END ..learning_rate=0.2, max_depth=3, n_estimators=1000; total time= 3
6.0s
[CV] END ...learning_rate=0.4, max_depth=3, n_estimators=500; total time= 1
8.6s
[CV] END ..learning_rate=0.2, max_depth=3, n_estimators=1000; total time= 3
6.0s
[CV] END ...learning_rate=0.4, max_depth=3, n_estimators=500; total time= 1
8.4s
[CV] END ..learning_rate=0.2, max_depth=3, n_estimators=1000; total time= 3
6.1s
[CV] END ..learning_rate=0.2, max_depth=3, n_estimators=1000; total time= 3
6.5s
[CV] END ..learning_rate=0.4, max_depth=3, n_estimators=1000; total time= 2
7.9s
[CV] END ..learning_rate=0.4, max_depth=3, n_estimators=1000; total time= 2
7.9s
[CV] END ..learning_rate=0.4, max_depth=3, n_estimators=1000; total time= 2
7.9s
[CV] END ..learning_rate=0.4, max_depth=3, n_estimators=1000; total time= 2
7.9s
Best hyperparameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 500}
R-squared: 0.9374039060824166
```

In []: best_params

Out[]: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 500}

After many different tests using different parameters and 4 & 5 fold cross validation I have arrived to the conclusion that the best parameters for the boosting algorithm are:

- learning_rate: 0.2
- max_depth: 3
- n_estimators: 1000

Testing with pc3 with response data

In []: y_testing = testing_df2['log_price']

In []: testing_df2 = testing_df2.drop('log_price', axis=1)

```
In [ ]: # standard_scaler.fit(testing_df2)

# # transform
# x_testing_scaled = standard_scaler.transform(testing_df2)

# testing_pred_test = boost_car.predict(x_testing_scaled)

# print(testing_pred_test)
```

```
In [ ]: # print('MSE on pc3 test data:', mean_squared_error(
#       y_testing, testing_pred_test))

# print('R squared on pc3 test data:', r2_score(
#       y_testing, testing_pred_test))
```

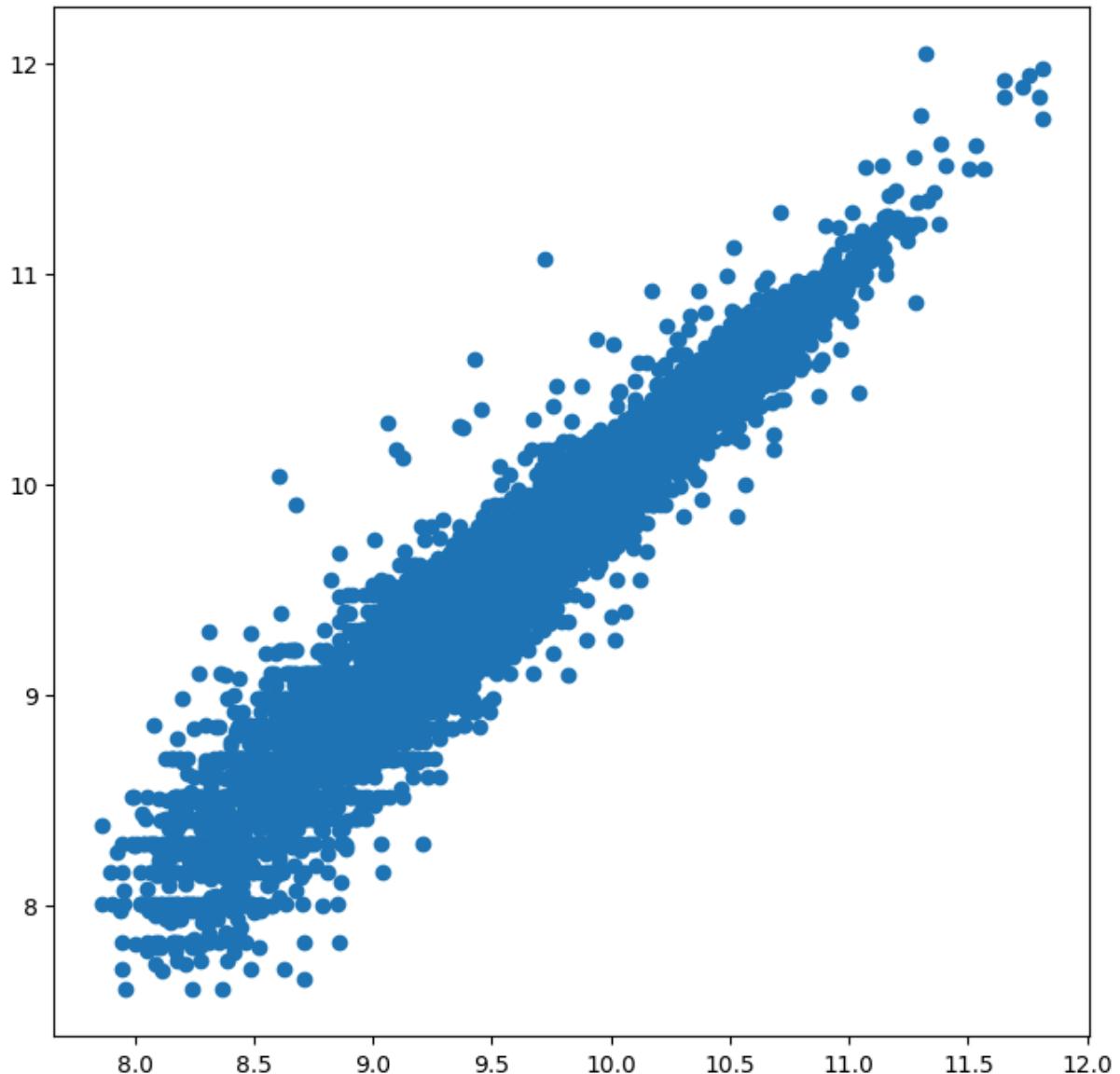
Bagging

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

```
Out[ ]: ▾
        RandomForestRegressor
RandomForestRegressor(max_features=35, random_state=0)
```

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

```
Out[ ]: 0.02581551749649698
```



```
In [ ]: bag_car = RF(max_features=x_train_mm_scaled.shape[1],
n_estimators=500,
random_state=0).fit(x_train_mm_scaled , y_train)
y_hat_bag = bag_car.predict(x_test_mm_scaled)
np.mean((y_test - y_hat_bag)**2)
```

```
Out[ ]: 0.025561325044691958
```

```
In [ ]: r_squared = r2_score(y_test, y_hat_bag)
print(f"R-squared: {r_squared}")
```

```
R-squared: 0.9329435332268757
```

Random Forest

```
In [ ]: RF_car = RF(max_features=6,
random_state=0).fit(x_train_mm_scaled , y_train)
y_hat_RF = RF_car.predict(x_test_mm_scaled)
np.mean((y_test - y_hat_RF)**2)
```

```
Out[ ]: 0.025444925882143133
```

```
In [ ]: r_squared = r2_score(y_test, y_hat_RF)
print(f"R-squared: {r_squared}")
```

```
R-squared: 0.9332488897200242
```

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

Out[]:

	importance
year	1.384799e-01
age2	1.155997e-01
mile2	9.813195e-02
mileage	9.677768e-02
age_at_listing	9.649850e-02
hp_year	7.816257e-02
horsepower	5.210680e-02
hp2	4.574285e-02
hp_vol	4.189746e-02
hp_lit	3.689849e-02
hp_mile	3.378980e-02
hp_hwfe	3.080082e-02
width	2.618177e-02
car_vol	2.147264e-02
wheel_system_FWD	1.706692e-02
height	1.509090e-02
litres	1.458075e-02
length	1.431800e-02
highway_fuel_economy	1.181624e-02
body_type_SUV / Crossover	2.852022e-03
body_type_Sedan	2.831566e-03
wheel_system_AWD	2.042732e-03
body_type_Pickup Truck	1.326622e-03
fuel_type_Flex Fuel Vehicle	1.252874e-03
fuel_type_Gasoline	1.052147e-03
wheel_system_RWD	6.467132e-04
body_type_Minivan	5.583313e-04
fuel_type_Hybrid	5.026296e-04
fuel_type_Diesel	3.259749e-04
wheel_system_4X2	3.229166e-04
body_type_Wagon	3.201262e-04
body_type_Coupe	2.329133e-04

importance	
body_type_Hatchback	2.300625e-04
body_type_Van	8.811356e-05
fuel_type_Compressed Natural Gas	5.288878e-07

Support Vector Machines

```
In [ ]: model_svr = SVR(C=0.01)
model_svr.fit(x_train_s_scaled, y_train)
```

```
Out[ ]: ▾ SVR
SVR(C=0.01)
```

```
In [ ]: y_pred = model_svr.predict(x_test_s_scaled)
```

```
In [ ]: mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
In [ ]: sns.lineplot(x=[min(y_test), max(y_test)], y=[min(y_test), max(y_test)],
                    linestyle='--', color='r', label='Perfect Prediction')
sns.scatterplot(x=y_test, y=y_pred, label='Actual vs Predicted')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.legend()
plt.title("Actual vs Predicted Values with Perfect Prediction Line of SVR")
plt.show()
```

```
In [ ]: residuals = y_test - y_pred
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot for SVR Regression")
plt.show()
```

hyperparameter tuning SVR

```
In [ ]: params_svr = {
    'kernel': ['linear', 'rbf'],
    'C': [8, 9, 10],
    'gamma': ['scale', 'auto'],
}
```

```
In [ ]: grid_search_svr = GridSearchCV(SVR(),
                                      param_grid=params_svr, cv=5,
                                      scoring='r2', n_jobs=-1)

In [ ]: grid_search_svr.fit(x_val_s_scaled, y_val)

In [ ]: print("Best SVR Parameters:", grid_search_svr.best_params_)

In [ ]: reg_best_svr = grid_search_svr.best_estimator_
reg_best_svr

In [ ]: best_svr = SVR(C= 9, gamma= 'auto', kernel = 'rbf')

In [ ]: best_svr.fit(x_train_s_scaled, y_train)

In [ ]: y_best_pred_svr = best_svr.predict(x_test_s_scaled)

In [ ]: mse_svr = mean_squared_error(y_test, y_best_pred_svr)
r2_svr = r2_score(y_test, y_best_pred_svr)

print("SVR Metrics:")
print("Mean Squared Error:", mse_svr)
print("R-squared:", r2_svr)
```

Predicting on the test data

Since Boosting ensemble method performed the best I will be using my best boosting model for predicting on the test set

```
In [ ]: test_df = prep_data(test_df)

In [ ]: test_df = test_df.drop('log_price', axis=1)

In [ ]: test_df.shape

Out[ ]: (50000, 31)

In [ ]: standard_scaler.fit(test_df)

# transform
x_final_scaled = standard_scaler.transform(test_df)

# Make predictions on the test set using the boosting model
final_pred_test = boost_car_subset.predict(x_final_scaled)

# Print the predictions
print(final_pred_test)

In [ ]: predictions_df = pd.DataFrame({'predictions': final_pred_test})
```

```
header = pd.DataFrame({
    'predictions': [21108082, 'MegumiFushiguro', round(r2_boost,3),
                    'Boosting']
})

header

output_df = pd.concat([header, predictions_df], axis=0)

output_df.to_csv('predictions_output.csv', index=False, header=False)
```