In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
```

In [2]:
```
df = pd.read_csv('used_vehicle.csv')
df.head()
```

Out[2]:

	Unnamed: 0	Year	Make	Model	Kilometres	Body Type	Engine	Transmission	Drivetrain	Exterior Colour	Interior Colour	Passengers	Doors	Fuel Type	C
0	0	2019	Acura	MDX	53052 km	SUV	V6 Cylinder Engine	9 Speed Automatic	AWD	Majestic Black Pearl	Red	NaN	NaN	Gas	12.2L/100
1	1	2018	Acura	MDX	77127 km	SUV	V6 Cylinder Engine	9 Speed Automatic	AWD	Modern Steel Metallic	Black	NaN	NaN	Gas	12.6L/100
2	2	2019	Acura	RDX	33032 km	SUV	2.0L 4cyl	10 Speed Automatic	AWD	White Diamond Pearl	Black	5.0	4	Premium Unleaded	11.0L/100
3	3	2020	Acura	RDX	50702 km	SUV	4 Cylinder Engine	NaN	AWD	Platinum White Pearl	Black	NaN	NaN	Gas	11.0L/100
4	4	2021	Acura	RDX	67950 km	SUV	4 Cylinder Engine	NaN	AWD	Apex Blue Pearl	Red	NaN	NaN	Gas	11.3L/100

Data Preprocessing Part 1

In [3]:
```
df.drop(columns=['Unnamed: 0'], inplace=True)
df.shape
```

Out[3]:
```
(24198, 16)
```

In [4]:
```
df.dtypes
```

Out[4]:
```
Year                int64
Make               object
Model              object
Kilometres         object
Body Type          object
 Engine            object
 Transmission      object
 Drivetrain        object
 Exterior Colour   object
 Interior Colour   object
 Passengers       float64
 Doors             object
 Fuel Type         object
 City              object
 Highway           object
Price               int64
dtype: object
```

In [5]:
```
# Remove rows with missing values in 'City' and 'Highway' columns in the original dataframe
df.dropna(subset=[' City', ' Highway'], inplace=True)
# Extract all the characters before the first 'L' only if 'L' is present
mask = df[' City'].str.contains('L')
df.loc[mask, ' City'] = df.loc[mask, ' City'].str.split('L', n=1, expand=True)[0]

mask = df[' Highway'].str.contains('L')
df.loc[mask, ' Highway'] = df.loc[mask, ' Highway'].str.split('L', n=1, expand=True)[0]

df.head()
```

Out[5]:

	Year	Make	Model	Kilometres	Body Type	Engine	Transmission	Drivetrain	Exterior Colour	Interior Colour	Passengers	Doors	Fuel Type	City	Highway	Price
0	2019	Acura	MDX	53052 km	SUV	V6 Cylinder Engine	9 Speed Automatic	AWD	Majestic Black Pearl	Red	NaN	NaN	Gas	12.2	9.0	4388
1	2018	Acura	MDX	77127 km	SUV	V6 Cylinder Engine	9 Speed Automatic	AWD	Modern Steel Metallic	Black	NaN	NaN	Gas	12.6	9.0	3648
2	2019	Acura	RDX	33032 km	SUV	2.0L 4cyl	10 Speed Automatic	AWD	White Diamond Pearl	Black	5.0	4	Premium Unleaded	11.0	8.6	4088
3	2020	Acura	RDX	50702 km	SUV	4 Cylinder Engine	NaN	AWD	Platinum White Pearl	Black	NaN	NaN	Gas	11.0	8.6	4459
4	2021	Acura	RDX	67950 km	SUV	4 Cylinder Engine	NaN	AWD	Apex Blue Pearl	Red	NaN	NaN	Gas	11.3	9.1	4698

In [6]:
```
# Remove non-numeric characters from the 'City' and 'Highway' columns
df[' City'] = df[' City'].str.replace('[^0-9]', '').astype(int)
df[' Highway'] = df[' Highway'].str.replace('[^0-9]', '').astype(int)

print(df.dtypes)
```

```
Year                int64
Make               object
Model              object
Kilometres         object
Body Type          object
 Engine            object
 Transmission      object
 Drivetrain        object
 Exterior Colour   object
 Interior Colour   object
 Passengers       float64
 Doors              object
 Fuel Type          object
 City                int32
 Highway             int32
Price                int64
dtype: object

C:\Users\Michael\AppData\Local\Temp\ipykernel_6156\4270810385.py:2: FutureWarning: The default value of regex will ch
ange from True to False in a future version.
  df[' City'] = df[' City'].str.replace('[^0-9]', '').astype(int)
C:\Users\Michael\AppData\Local\Temp\ipykernel_6156\4270810385.py:3: FutureWarning: The default value of regex will ch
ange from True to False in a future version.
  df[' Highway'] = df[' Highway'].str.replace('[^0-9]', '').astype(int)
```

In [7]:
```
df.shape
```

Out[7]: (17835, 16)

In [8]:
```
df.dropna(subset=['Kilometres'], inplace=True)
df['Kilometres'] = df['Kilometres'].str.replace('km', '').astype(int)
df.dtypes
```

Out[8]:
```
Year                int64
Make               object
Model              object
Kilometres          int32
Body Type          object
 Engine            object
 Transmission      object
 Drivetrain        object
 Exterior Colour   object
 Interior Colour   object
 Passengers       float64
 Doors             object
 Fuel Type         object
 City               int32
 Highway            int32
Price               int64
dtype: object
```

In [9]:
```
df.shape
```

Out[9]:
```
(17662, 16)
```

In [10]:
```
df.head()
```

Out[10]:

	Year	Make	Model	Kilometres	Body Type	Engine	Transmission	Drivetrain	Exterior Colour	Interior Colour	Passengers	Doors	Fuel Type	City	Highway	Price
0	2019	Acura	MDX	53052	SUV	V6 Cylinder Engine	9 Speed Automatic	AWD	Majestic Black Pearl	Red	NaN	NaN	Gas	122	90	4388(
1	2018	Acura	MDX	77127	SUV	V6 Cylinder Engine	9 Speed Automatic	AWD	Modern Steel Metallic	Black	NaN	NaN	Gas	126	90	3648(
2	2019	Acura	RDX	33032	SUV	2.0L 4cyl	10 Speed Automatic	AWD	White Diamond Pearl	Black	5.0	4	Premium Unleaded	110	86	4088{
3	2020	Acura	RDX	50702	SUV	4 Cylinder Engine	NaN	AWD	Platinum White Pearl	Black	NaN	NaN	Gas	110	86	4459{
4	2021	Acura	RDX	67950	SUV	4 Cylinder Engine	NaN	AWD	Apex Blue Pearl	Red	NaN	NaN	Gas	113	91	4698{

In [11]:
```
# Add decimal point to Highway and City columns
df[' Highway'] = df[' Highway'] / 10
df[' City'] = df[' City'] / 10
df.head()
```

Out[11]:

	Year	Make	Model	Kilometres	Body Type	Engine	Transmission	Drivetrain	Exterior Colour	Interior Colour	Passengers	Doors	Fuel Type	City	Highway	Price
0	2019	Acura	MDX	53052	SUV	V6 Cylinder Engine	9 Speed Automatic	AWD	Majestic Black Pearl	Red	NaN	NaN	Gas	12.2	9.0	4388(
1	2018	Acura	MDX	77127	SUV	V6 Cylinder Engine	9 Speed Automatic	AWD	Modern Steel Metallic	Black	NaN	NaN	Gas	12.6	9.0	3648(
2	2019	Acura	RDX	33032	SUV	2.0L 4cyl	10 Speed Automatic	AWD	White Diamond Pearl	Black	5.0	4	Premium Unleaded	11.0	8.6	4088{
3	2020	Acura	RDX	50702	SUV	4 Cylinder Engine	NaN	AWD	Platinum White Pearl	Black	NaN	NaN	Gas	11.0	8.6	4459{
4	2021	Acura	RDX	67950	SUV	4 Cylinder Engine	NaN	AWD	Apex Blue Pearl	Red	NaN	NaN	Gas	11.3	9.1	4698{

In [12]:
```
# iterate over columns with object datatype
for col in df.select_dtypes(include='object'):
    # count the unique number of values
    unique_count = df[col].nunique()
    # print the result
    print(f'The number of unique values in the "{col}" column is: {unique_count}')
```

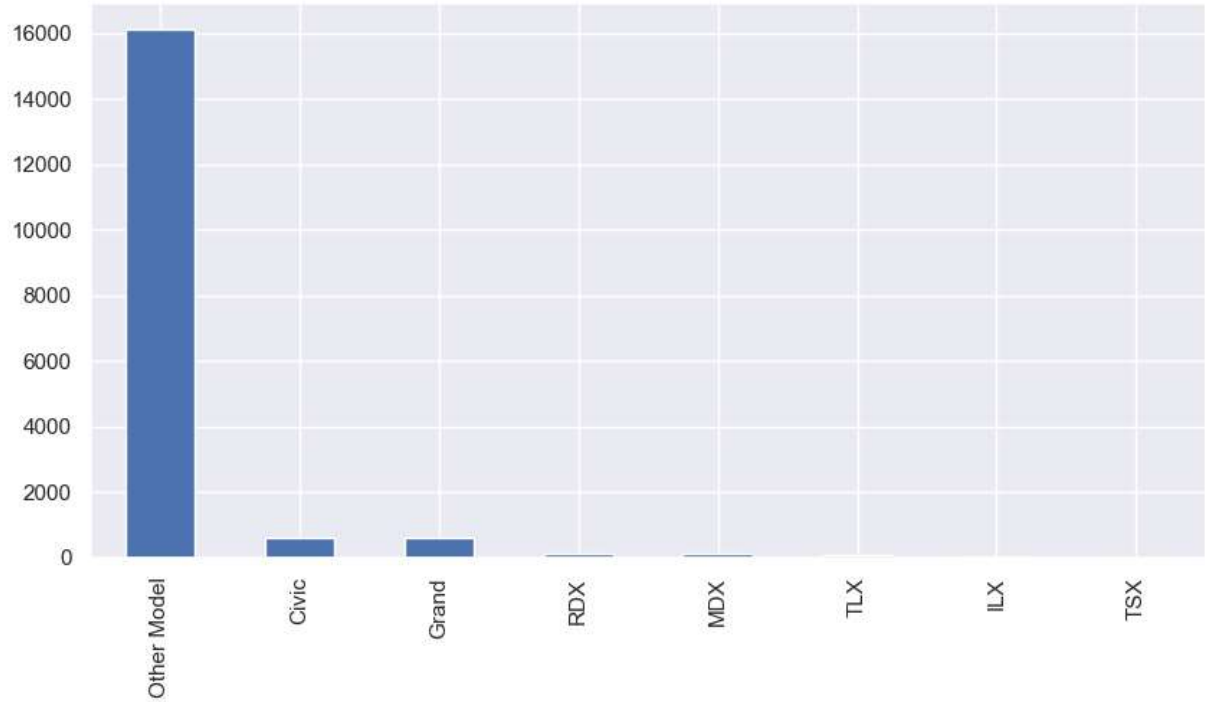
```
The number of unique values in the "Make" column is: 39
The number of unique values in the "Model" column is: 489
The number of unique values in the "Body Type" column is: 26
The number of unique values in the " Engine" column is: 957
The number of unique values in the " Transmission" column is: 20
The number of unique values in the " Drivetrain" column is: 6
The number of unique values in the " Exterior Colour" column is: 1258
The number of unique values in the " Interior Colour" column is: 15
The number of unique values in the " Doors" column is: 4
The number of unique values in the " Fuel Type" column is: 11
```

In [13]:
```
def fetch_model(text):
    if 'MDX' in text:
        return 'MDX'
    elif 'TSX' in text:
        return 'TSX'
    elif 'Grand' in text:
        return 'Grand'
    elif 'Civic' in text:
        return 'Civic'
    elif 'RDX' in text:
        return 'RDX'
    elif 'ILX' in text:
        return 'ILX'
    elif 'TLX' in text:
        return 'TLX'
    else:
        return 'Other Model'
```

In [14]:
```
df['Model'] = df['Model'].apply(fetch_model)
```

In [15]:
```
plt.figure(figsize=(10,5))
df['Model'].value_counts().plot(kind='bar')
```

Out[15]: <AxesSubplot:>



In [16]:
```
df.drop(columns=[' Exterior Colour', ' Interior Colour'], inplace=True)
df.shape
```

Out[16]:
```
(17662, 14)
```

In [17]:
```
df.Make.unique()
```

Out[17]:
```
array(['Acura', 'Alfa Romeo', 'Audi', 'Bentley', 'Volvo', 'BMW',
       'Aston Martin', 'Buick', 'Cadillac', 'Chevrolet', 'Chrysler',
       'Maserati', 'Porsche', 'Mazda', 'McLaren', 'Mercedes-Benz', 'MINI',
       'Mitsubishi', 'Scion', 'Subaru', 'Dodge', 'Ferrari', 'Fiat',
       'Ford', 'Genesis', 'GMC', 'Honda', 'Hummer', 'Hyundai', 'Infiniti',
       'Jaguar', 'Jeep', 'Kia', 'Lamborghini', 'Nissan', 'Polestar',
       'Rolls-Royce', 'Suzuki', 'Volkswagen'], dtype=object)
```

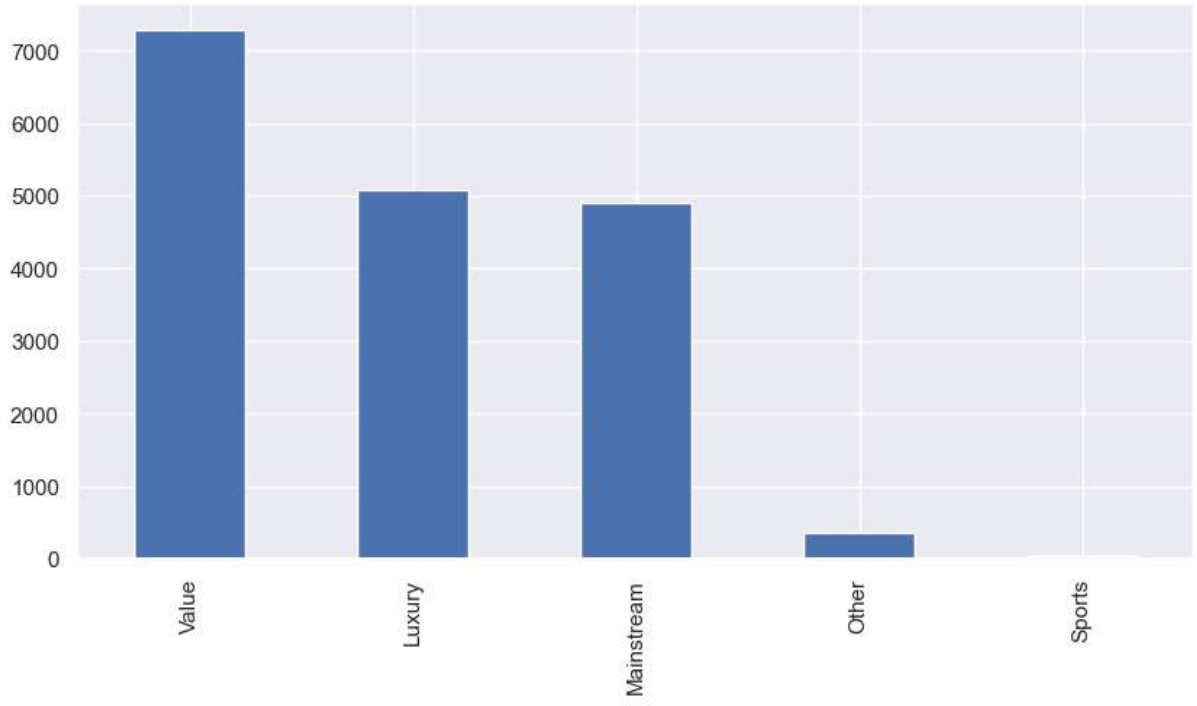
Make the segmentation for car brands

In [18]:
```
def segment_make(make):
    if make in ['Acura', 'Alfa Romeo', 'Audi', 'Bentley', 'BMW', 'Cadillac', 'Genesis', 'Infiniti', 'Jaguar', 'Lamborg
        return 'Luxury'
    elif make in ['Buick', 'Chevrolet', 'Chrysler', 'Dodge', 'Ford', 'GMC', 'Jeep', 'Ram']:
        return 'Mainstream'
    elif make in ['Ferrari', 'Lotus']:
        return 'Sports'
    elif make in ['Honda', 'Hyundai', 'Kia', 'Mazda', 'Mitsubishi', 'Nissan', 'Subaru', 'Toyota', 'Volkswagen']:
        return 'Value'
    else:
        return 'Other'
```

In [19]:
```
df['Make'] = df['Make'].apply(segment_make)
```

In [20]:
```
plt.figure(figsize=(10,5))
df['Make'].value_counts().plot(kind='bar')
```

Out[20]:
```
<AxesSubplot:>
```



Segment body type

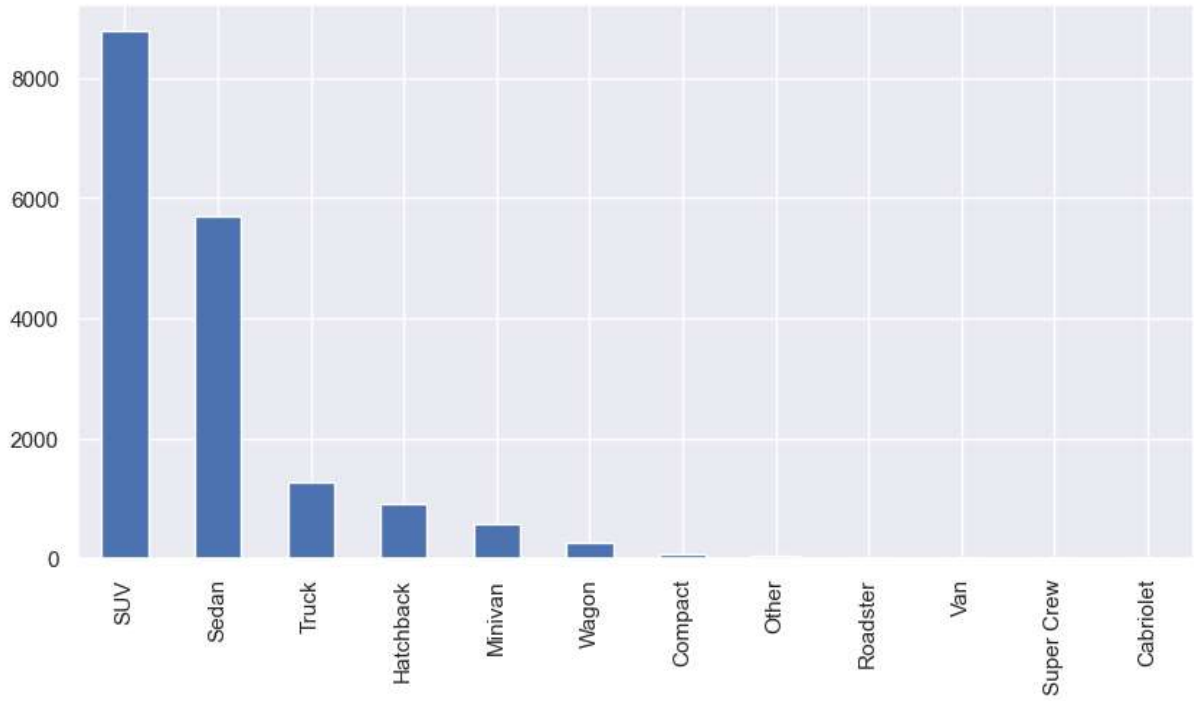
In [21]:
```
df['Body Type'].unique()
```

Out[21]:
```
array(['SUV', 'Sedan', 'Coupe', 'Convertible', 'Hatchback', nan, 'Wagon',
       'Roadster', 'Station Wagon', 'Truck', 'Truck Extended Cab',
       'Extended Cab', 'Crew Cab', 'Regular Cab', 'Compact',
       'Truck Crew Cab', 'Super Cab', 'Minivan', 'Cabriolet',
       'Van Regular', 'Super Crew', 'Quad Cab', 'Truck Super Cab',
       'Van Extended', 'Truck Double Cab', 'Truck King Cab',
       'Truck Long Crew Cab'], dtype=object)
```

In [22]:
```
# Define the body type segments
suv = ['SUV']
sedan = ['Sedan', 'Coupe', 'Convertible']
hatchback = ['Hatchback']
wagon = ['Wagon', 'Station Wagon']
truck = ['Truck', 'Truck Extended Cab', 'Extended Cab', 'Crew Cab',
         'Regular Cab', 'Truck Crew Cab', 'Super Cab', 'Quad Cab',
         'Truck Super Cab', 'Truck Double Cab', 'Truck King Cab',
         'Truck Long Crew Cab']
van = ['Van Regular', 'Van Extended']
minivan = ['Minivan']
roadster = ['Roadster']
cabriolet = ['Cabriolet']
super_crew = ['Super Crew']
compact = ['Compact']

# Create a dictionary to map each body type to its corresponding segment
body_type_segments = {}
for body_type in df['Body Type'].unique():
    if body_type in suv:
        body_type_segments[body_type] = 'SUV'
    elif body_type in sedan:
        body_type_segments[body_type] = 'Sedan'
    elif body_type in hatchback:
        body_type_segments[body_type] = 'Hatchback'
    elif body_type in wagon:
        body_type_segments[body_type] = 'Wagon'
    elif body_type in truck:
        body_type_segments[body_type] = 'Truck'
    elif body_type in van:
        body_type_segments[body_type] = 'Van'
    elif body_type in minivan:
        body_type_segments[body_type] = 'Minivan'
    elif body_type in roadster:
        body_type_segments[body_type] = 'Roadster'
    elif body_type in cabriolet:
        body_type_segments[body_type] = 'Cabriolet'
    elif body_type in super_crew:
        body_type_segments[body_type] = 'Super Crew'
    elif body_type in compact:
        body_type_segments[body_type] = 'Compact'
    else:
        body_type_segments[body_type] = 'Other'

# Map the body type segments to the dataframe
df['Body Type'] = df['Body Type'].map(body_type_segments)
```

In [23]:
```
plt.figure(figsize=(10,5))
df['Body Type'].value_counts().plot(kind='bar')
```

Out[23]: <AxesSubplot:>



Segment Transmission

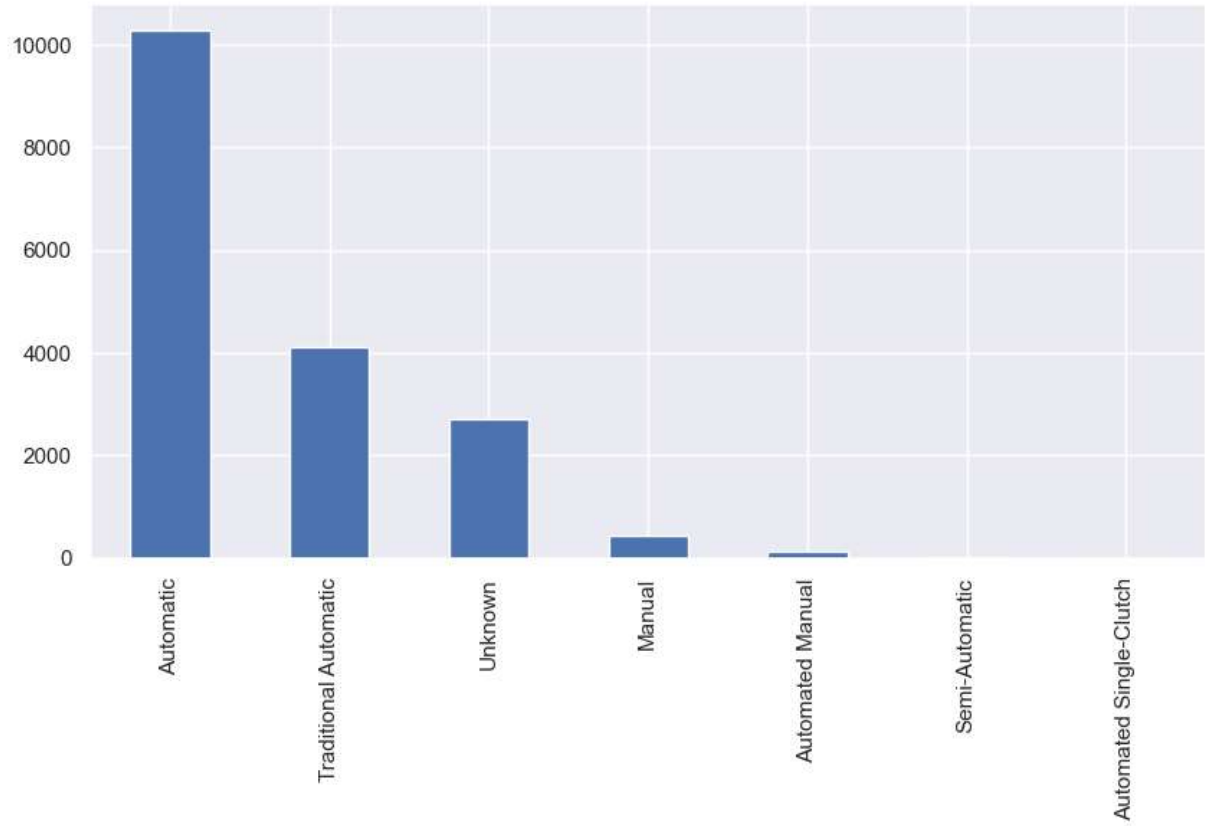
In [24]:
```
df[' Transmission'].unique()
```

Out[24]:
```
array(['9 Speed Automatic', '10 Speed Automatic', nan, 'Automatic',
       '6 Speed Manual', '8 Speed Automatic', '6 Speed Automatic',
       '8 Speed Automatic with auto-shift', '5 Speed Automatic',
       '5 Speed Manual', '4 Speed Automatic', 'Manual',
       '7 Speed Automatic with auto-shift',
       '6 Speed Automatic with auto-shift', 'Sequential',
       '7 Speed Automatic', 'CVT', '1 Speed Automatic', '7 Speed Manual',
       '5 Speed Automatic with auto-shift', 'F1 Transmission'],
      dtype=object)
```

In [25]:
```
def segment_transmission(transmission):
    if transmission in ['Automatic', 'CVT', '1 Speed Automatic']:
        return 'Automatic'
    elif transmission in ['6 Speed Manual', '5 Speed Manual', '7 Speed Manual']:
        return 'Manual'
    elif transmission in ['9 Speed Automatic', '10 Speed Automatic', '8 Speed Automatic', '7 Speed Automatic', '5 Spee
        return 'Traditional Automatic'
    elif transmission in ['8 Speed Automatic with auto-shift', '6 Speed Automatic with auto-shift', '7 Speed Automatic
        return 'Automated Manual'
    elif transmission == 'Sequential':
        return 'Semi-Automatic'
    elif transmission == 'F1 Transmission':
        return 'Automated Single-Clutch'
    else:
        return 'Unknown'

df['Transmission'] = df[' Transmission'].apply(segment_transmission)
```

In [26]:
```
plt.figure(figsize=(10,5))
df['Transmission'].value_counts().plot(kind='bar')
```

Out[26]: <AxesSubplot:>



In [27]:
```
df.drop(columns=[' Transmission'], inplace=True)
df.shape
```

Out[27]: (17662, 14)

Drop Engine column

In [28]:
```
df.drop(columns=[' Engine'], inplace=True)
df.shape
```

Out[28]: (17662, 13)

In [29]:
```
df.head()
```

Out[29]:

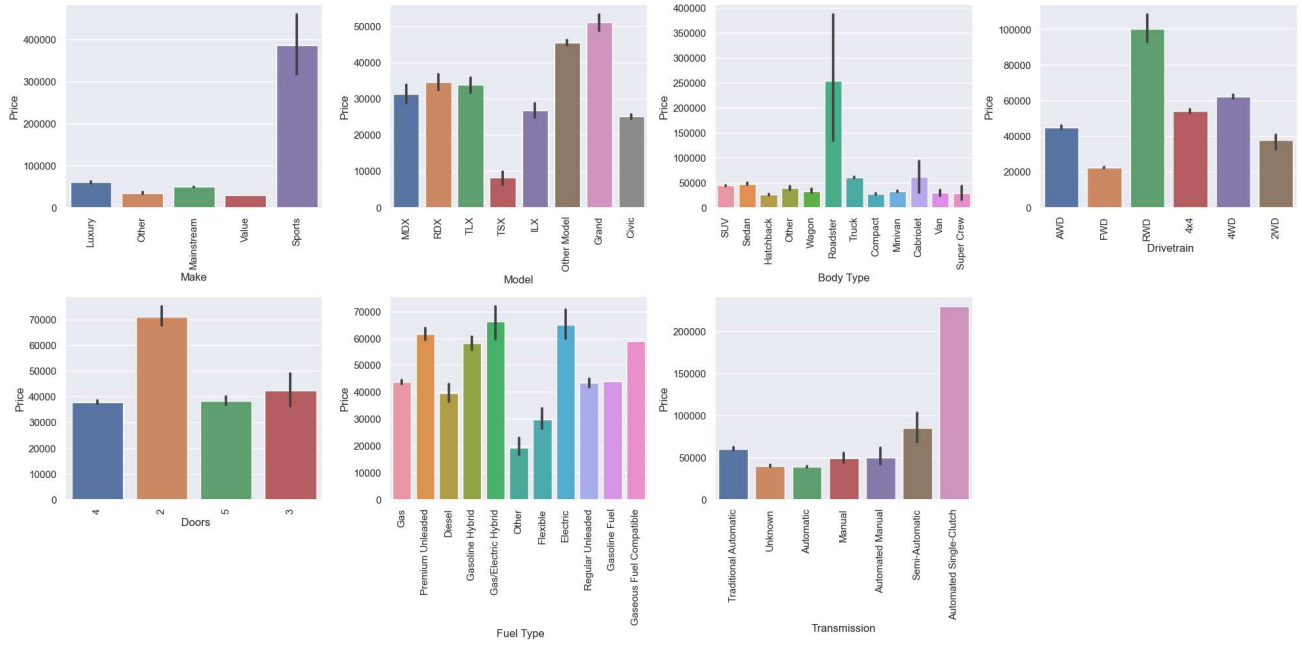
	Year	Make	Model	Kilometres	Body Type	Drivetrain	Passengers	Doors	Fuel Type	City	Highway	Price	Transmission
0	2019	Luxury	MDX	53052	SUV	AWD	NaN	NaN	Gas	12.2	9.0	43880	Traditional Automatic
1	2018	Luxury	MDX	77127	SUV	AWD	NaN	NaN	Gas	12.6	9.0	36486	Traditional Automatic
2	2019	Luxury	RDX	33032	SUV	AWD	5.0	4	Premium Unleaded	11.0	8.6	40888	Traditional Automatic
3	2020	Luxury	RDX	50702	SUV	AWD	NaN	NaN	Gas	11.0	8.6	44599	Unknown
4	2021	Luxury	RDX	67950	SUV	AWD	NaN	NaN	Gas	11.3	9.1	46989	Unknown

Exploratory Data Analysis

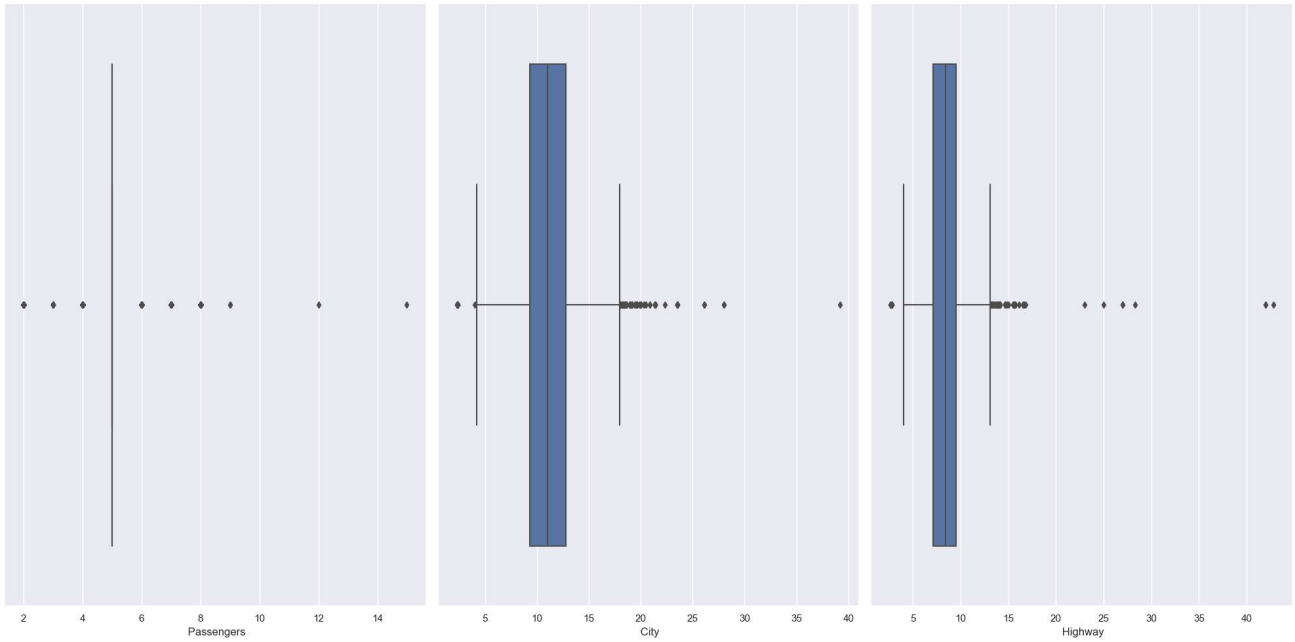
In [30]:
```
# iterate over columns with object datatype
for col in df.select_dtypes(include='object'):
    # count the unique number of values
    unique_count = df[col].nunique()
    # print the result
    print(f'The number of unique values in the "{col}" column is: {unique_count}')
```

```
The number of unique values in the "Make" column is: 5
The number of unique values in the "Model" column is: 8
The number of unique values in the "Body Type" column is: 12
The number of unique values in the " Drivetrain" column is: 6
The number of unique values in the " Doors" column is: 4
The number of unique values in the " Fuel Type" column is: 11
The number of unique values in the "Transmission" column is: 7
```

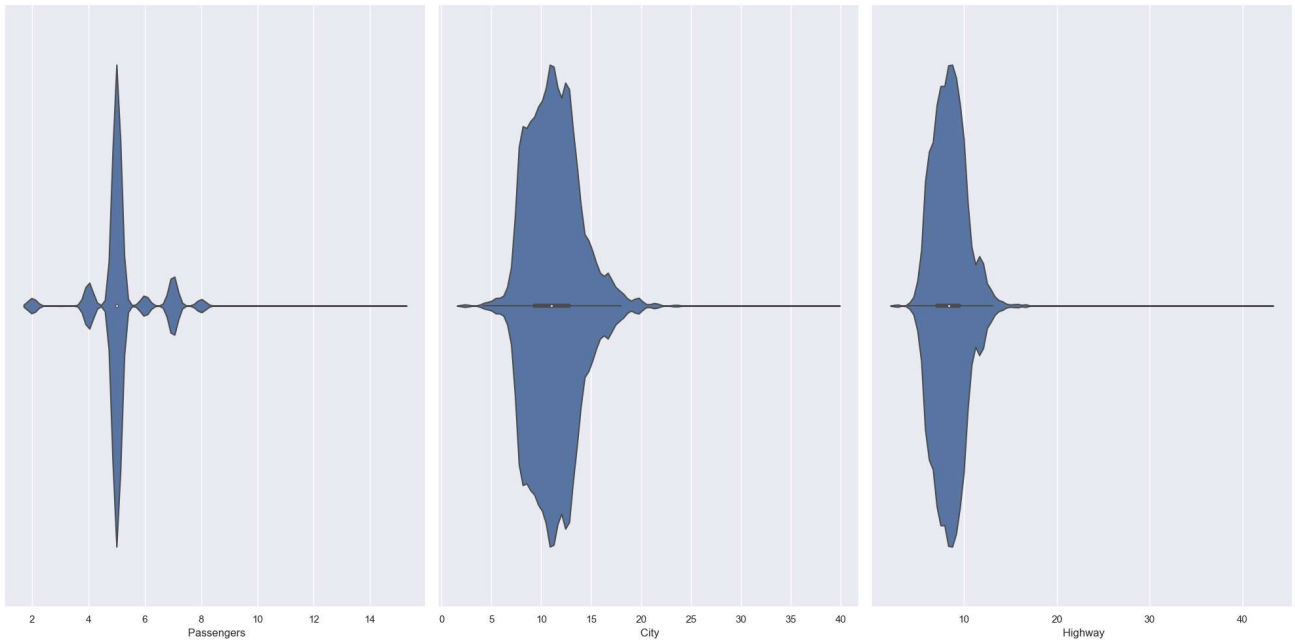
In [31]:
```
# list of categorical variables to plot
cat_vars = ['Make', 'Model', 'Body Type', ' Drivetrain', ' Doors', ' Fuel Type', 'Transmission']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))
axs = axs.ravel()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.barplot(x=var, y='Price', data=df, ax=axs[i], estimator=np.mean)
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# remove the eighth subplot
fig.delaxes(axs[7])

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



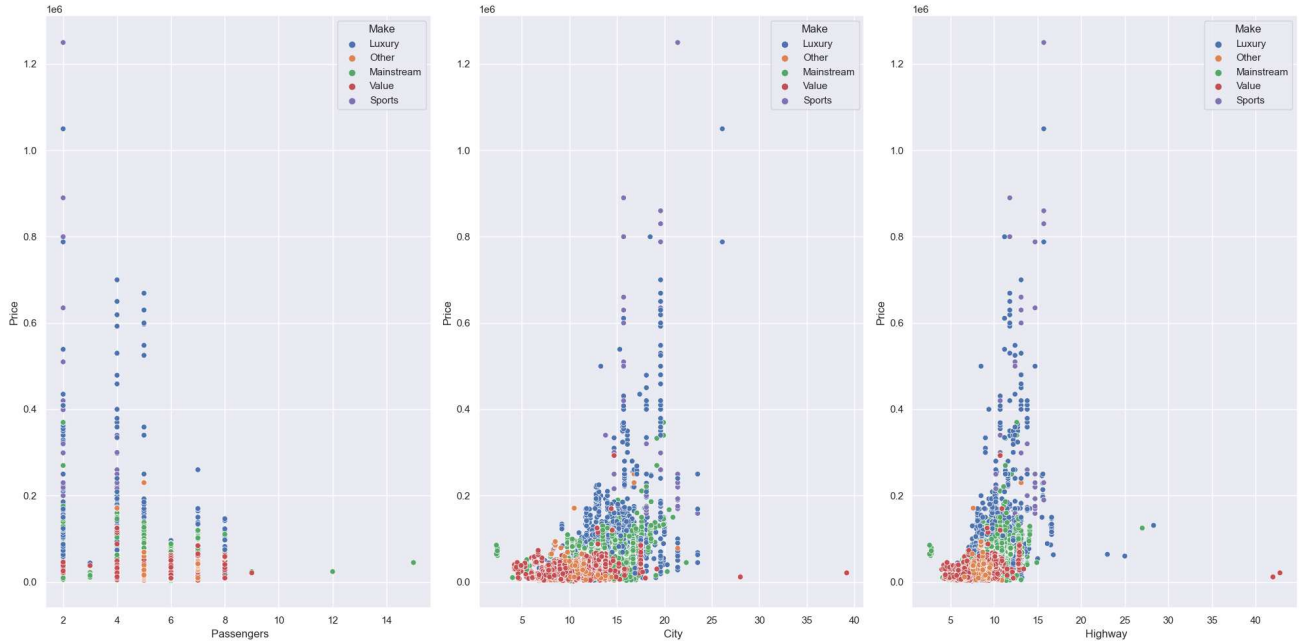
In [32]:
```
num_vars = [' Passengers', ' City', ' Highway']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



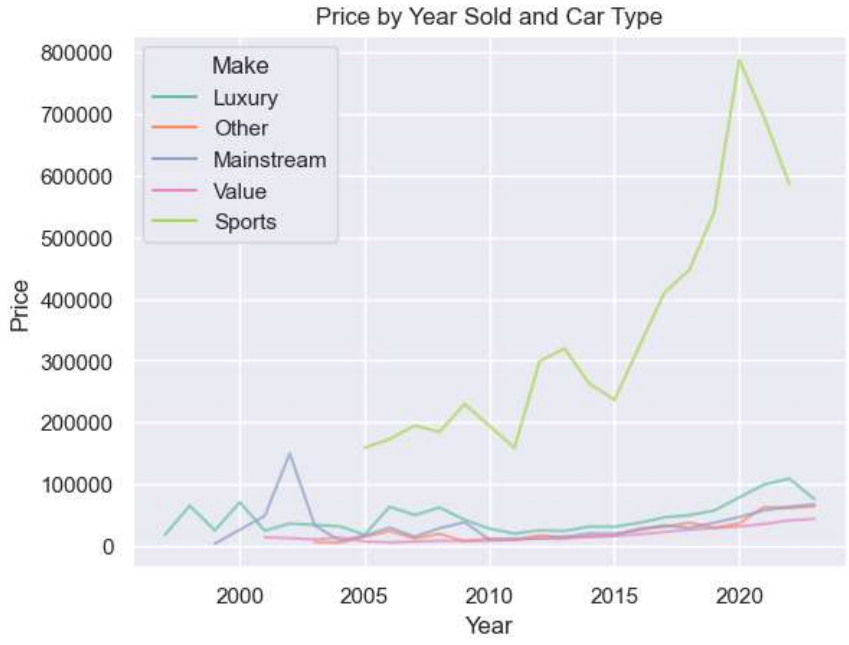
In [33]:
```
num_vars = [' Passengers', ' City', ' Highway']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [34]:
```
num_vars = [' Passengers', ' City', ' Highway']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.scatterplot(x=var, y='Price', hue='Make', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [35]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Year', y='Price', hue='Make', data=df, ci=None, estimator='mean', alpha=0.7)

plt.title("Price by Year Sold and Car Type")
plt.xlabel("Year")
plt.ylabel("Price")

plt.show()
```



Data Preprocessing Part 2

In [36]:
```
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[36]:
```
 Passengers    46.342430
 Doors         19.805232
 Drivetrain     2.276073
dtype: float64
```

In [37]:
```
df.drop(columns=[' Passengers'], inplace=True)
df[' Doors'] = df[' Doors'].fillna(df[' Doors'].median())
df.dropna(subset=[' Drivetrain'], inplace=True)
df.shape
```

Out[37]:
```
(17260, 12)
```

Label Encoding for Object datatype

In [38]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
Make: ['Luxury' 'Other' 'Mainstream' 'Value' 'Sports']
Model: ['MDX' 'RDX' 'TLX' 'TSX' 'ILX' 'Other Model' 'Grand' 'Civic']
Body Type: ['SUV' 'Sedan' 'Hatchback' 'Other' 'Wagon' 'Roadster' 'Truck' 'Compact'
 'Minivan' 'Cabriolet' 'Van' 'Super Crew']
 Drivetrain: ['AWD' 'FWD' 'RWD' '4x4' '4WD' '2WD']
 Doors: [4.0 '4' '2' '5' '3']
 Fuel Type: ['Gas' 'Premium Unleaded' 'Diesel' 'Gasoline Hybrid' 'Gas/Electric Hybrid'
 'Other' 'Flexible' 'Electric' 'Regular Unleaded' 'Gasoline Fuel'
 'Gaseous Fuel Compatible']
Transmission: ['Traditional Automatic' 'Unknown' 'Automatic' 'Manual' 'Automated Manual'
 'Semi-Automatic' 'Automated Single-Clutch']
```

In [40]:
```
df[' Doors'] = df[' Doors'].astype(float)
```

In [41]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Make: [0 2 1 4 3]
Model: [3 5 6 7 2 4 1 0]
Body Type: [ 6  7  2  4 11  5  9  1  3  0 10  8]
 Drivetrain: [3 4 5 2 1 0]
 Fuel Type: [ 3  9  0  7  4  8  2  1 10  6  5]
Transmission: [5 6 2 3 0 4 1]
```

In [42]:
```
df.dtypes
```

Out[42]:
```
Year              int64
Make              int32
Model             int32
Kilometres        int32
Body Type         int32
 Drivetrain       int32
 Doors          float64
 Fuel Type        int32
 City           float64
 Highway        float64
Price             int64
Transmission      int32
dtype: object
```

Remove Outlier using Z-Score

In [43]:
```
from scipy import stats

# define a function to remove outliers using z-score for only selected numerical columns
def remove_outliers(df, cols, threshold=3):
    # loop over each selected column
    for col in cols:
        # calculate z-score for each data point in selected column
        z = np.abs(stats.zscore(df[col]))
        # remove rows with z-score greater than threshold in selected column
        df = df[(z < threshold) | (df[col].isnull())]
    return df
```

In [44]:
```
selected_cols = [' City', ' Highway']
df_clean = remove_outliers(df, selected_cols)
df_clean.shape
```

Out[44]:
```
(17027, 12)
```

In [45]:
```
#dataframe before the outlier removed
df.shape
```

Out[45]:
```
(17260, 12)
```

In [46]:
```
#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df_clean.corr(), fmt='.2g', annot=True)
```

Out[46]: <AxesSubplot:>

Machine Learning Model Building

In [47]:
```
X = df_clean.drop('Price', axis=1)
y = df_clean['Price']
```

In [48]:
```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

Decision Tree Regressor

In [49]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_boston


# Create a DecisionTreeRegressor object
dtree = DecisionTreeRegressor()

# Define the hyperparameters to tune and their values
param_grid = {
    'max_depth': [2, 4, 6, 8],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 3, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='neg_mean_squared_error')

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

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 4}
```

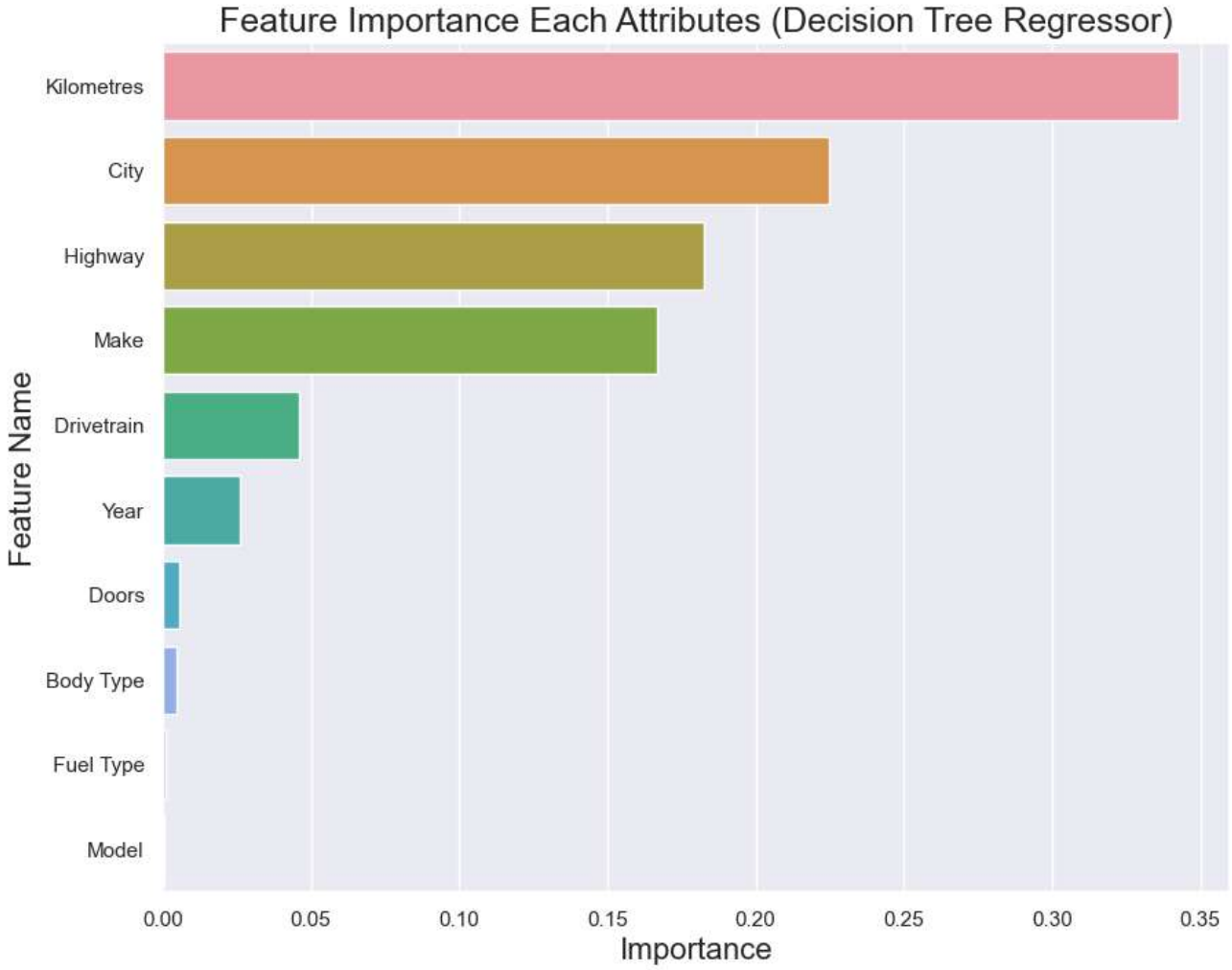
In [50]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=8, max_features='auto', min_samples_leaf=4, min_samples_split=
dtree.fit(X_train, y_train)
```

Out[50]:
```
DecisionTreeRegressor(max_depth=8, max_features='auto', min_samples_leaf=4,
                      min_samples_split=8, random_state=0)
```

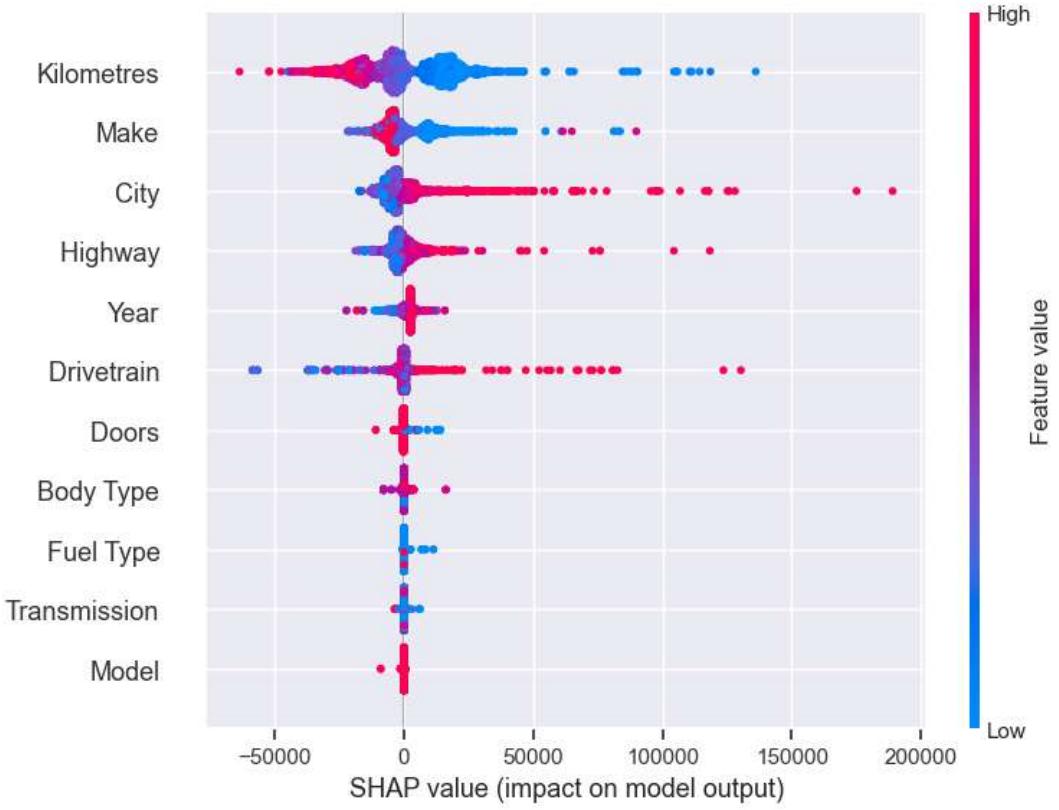
In [51]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 7059.377754152276
MAPE is 0.17389139445970517
MSE is 327116827.39038324
R2 score is 0.769459884693911
RMSE score is 18086.371316280754
```

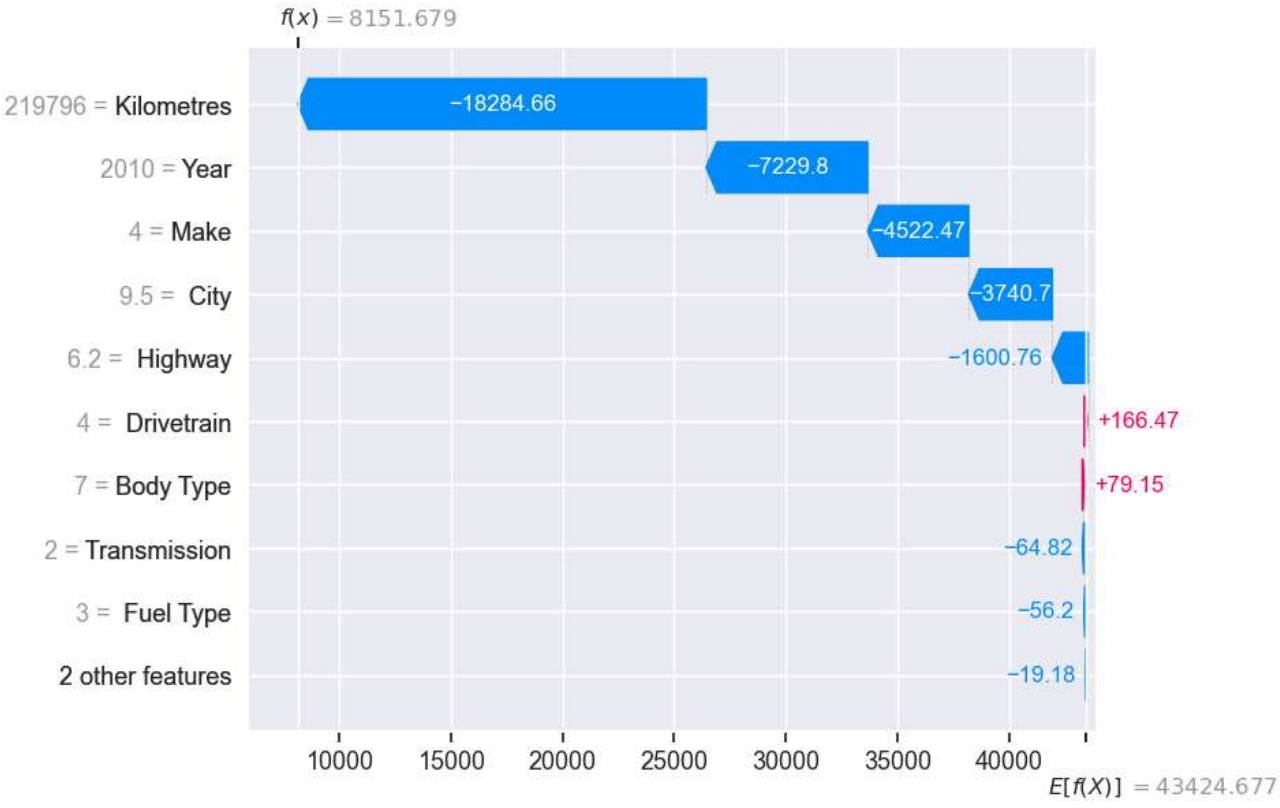
In [52]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Decision Tree Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [53]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [54]:
```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```



Random Forest Regressor

In [55]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a Random Forest Regressor object
rf = RandomForestRegressor()

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='r2')

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

# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'max_depth': 9, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2}
```

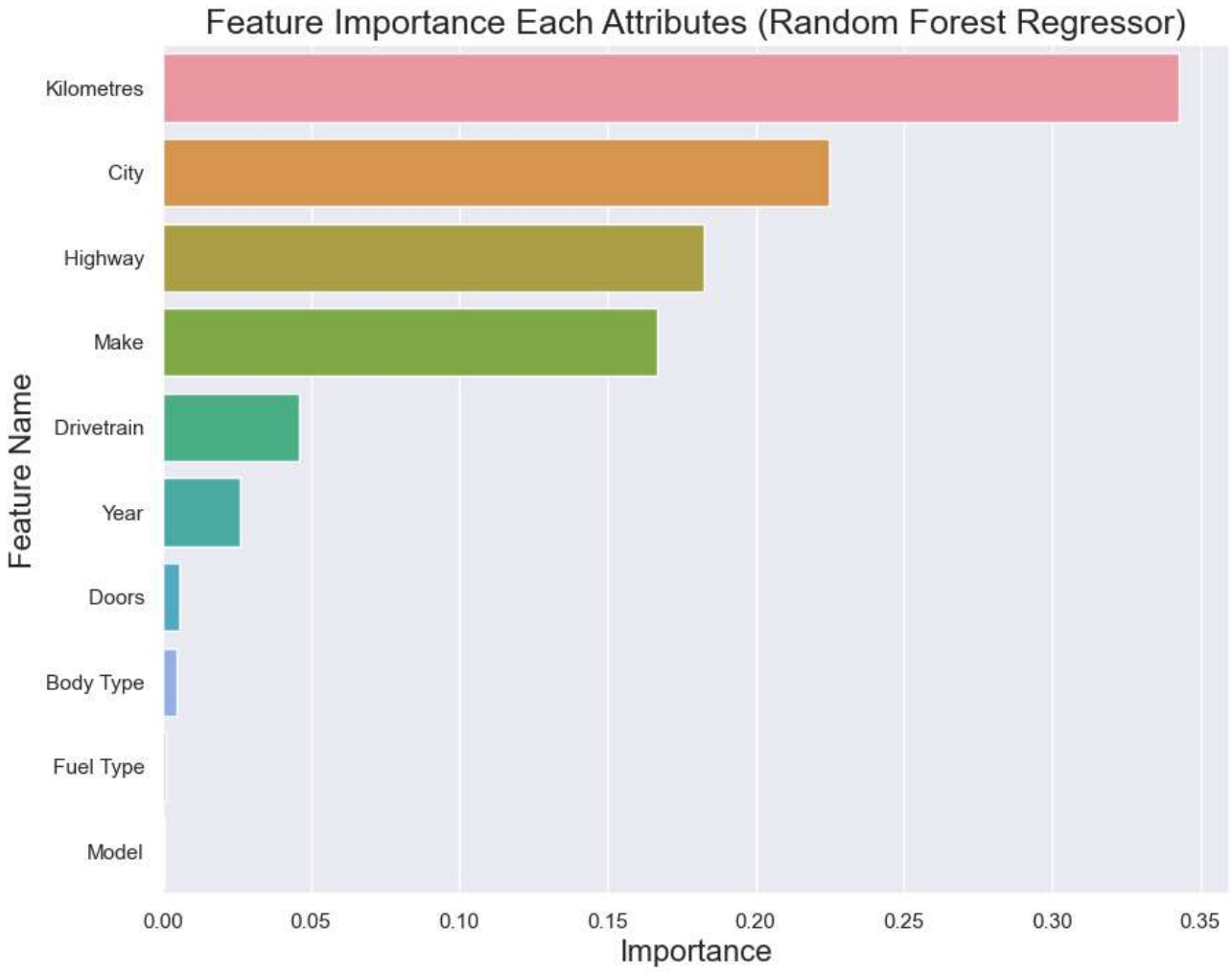
In [56]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=0, max_depth=9, min_samples_split=2, min_samples_leaf=1,
                           max_features='sqrt')
rf.fit(X_train, y_train)
```

Out[56]:
```
RandomForestRegressor(max_depth=9, max_features='sqrt', random_state=0)
```

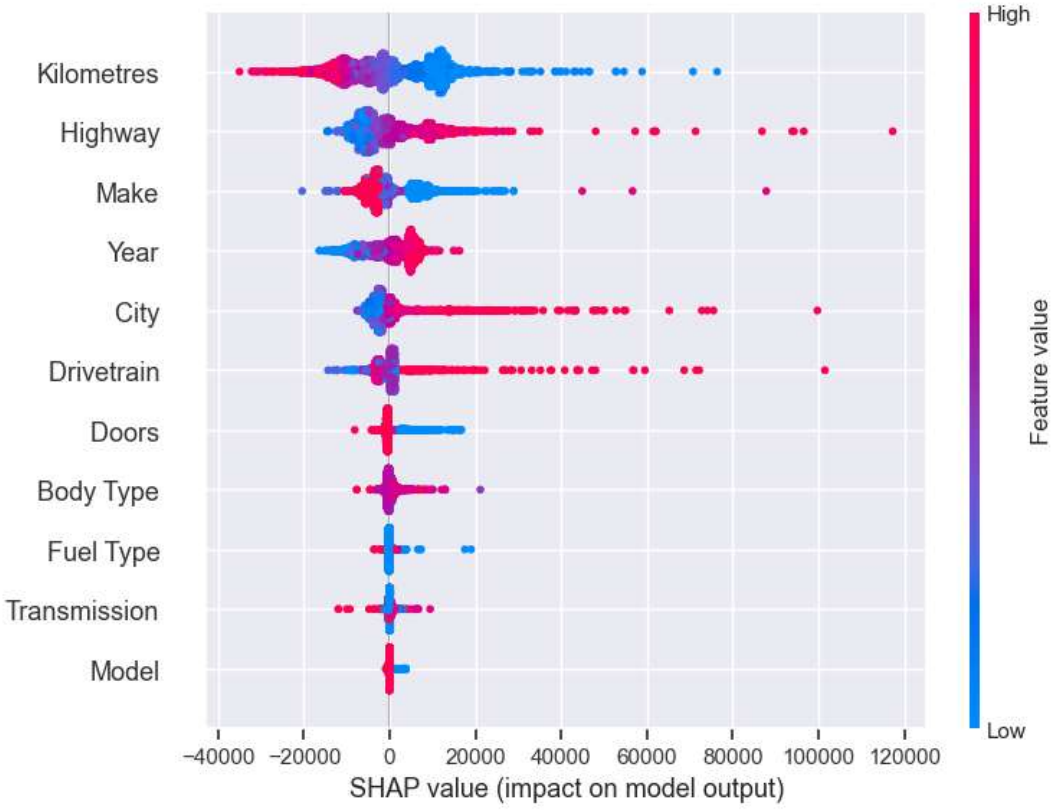
In [57]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 5978.515668161472
MAPE is 0.16069794103678922
MSE is 233754623.5904292
R2 score is 0.8352581910695883
RMSE score is 15289.036058248708
```

In [58]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Random Forest Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```

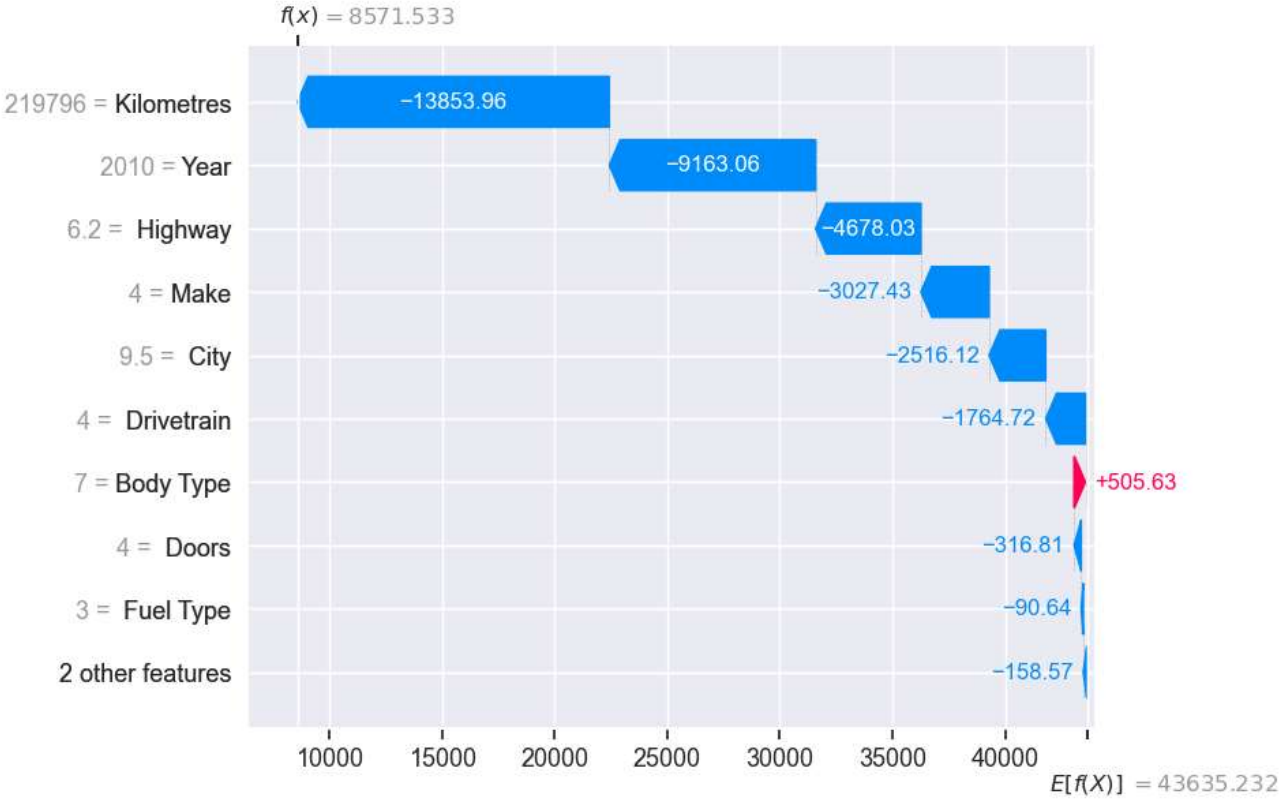


In [59]:
```
import shap
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [61]:
```
explainer = shap.Explainer(rf, X_test, check_additivity=False)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```

```
98%|==================| 3341/3406 [00:40<00:00]
```



AdaBoost Regressor

In [62]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import GridSearchCV

# Define AdaBoostRegressor model
abr = AdaBoostRegressor()

# Define hyperparameters and possible values
params = {'n_estimators': [50, 100, 150],
          'learning_rate': [0.01, 0.1, 1, 10]}

# Perform GridSearchCV with 5-fold cross validation
grid_search = GridSearchCV(abr, param_grid=params, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Print best hyperparameters and corresponding score
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'learning_rate': 0.1, 'n_estimators': 50}
```

In [63]:
```
from sklearn.ensemble import RandomForestRegressor
abr = AdaBoostRegressor(random_state=0, learning_rate=0.1, n_estimators=50)
abr.fit(X_train, y_train)
```

Out[63]:
```
AdaBoostRegressor(learning_rate=0.1, random_state=0)
```

In [64]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = abr.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 13233.929296884458
MAPE is 0.4019031020406141
MSE is 591868330.2418412
R2 score is 0.5828725957373728
RMSE score is 24328.344173861096
```

In [66]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": abr.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (AdaBoost Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```