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

In [2]:
```
df = pd.read_csv('pakwheels-11Jul2020.csv')
df.head()
```

Out[2]:

| | Ad No | Name | Price | Model Year | Location | Mileage | Registered City | Engine Type | Engine Capacity | Transmission |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4096758 | Toyota Vitz F 1.0 2017 | 2385000.0 | 2017 | G- 8, Islamabad Islamabad | 9869 | Un-Registered | Petrol | 1000 cc | Automatic |
| 1 | 4168305 | Toyota Corolla GLi Automatic 1.3 VVTi 2019 | 111000.00000000001 | 2019 | Peshawar KPK | 11111 | Islamabad | Petrol | 1300 cc | Automatic |
| 2 | 4168298 | Suzuki Alto VXL 2019 | 1530000.0 | 2019 | Akora Khattak, Nowshera KPK | 17500 | Un-Registered | Petrol | 660 cc | Automatic |
| 3 | 4168307 | Suzuki Alto VXR 2019 | 1650000.0 | 2019 | Abdullahpur, Faisalabad Punjab | 9600 | Lahore | Petrol | 660 cc | Manual |
| 4 | 4168306 | Toyota Corolla XLi VVTi 2010 | 1435000.0 | 2010 | 9th Avenue, Islamabad Islamabad | 120000 | Islamabad | Petrol | 1300 cc | Manual |

# Data Preprocessing Part 1

In [3]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[3]:
```
Name                 7328
Price                1511
Location             2143
Registered City       136
Engine Type             3
Engine Capacity       118
Transmission            2
Color                  24
Assembly                2
Body Type              18
Features             4940
Last Updated          661
URL                 56186
dtype: int64
```

In [4]:
```
# Remove categorial column that have huge unique value
df.drop(columns=['Ad No', 'Name', 'Location', 'Features', 'URL', 'Last Updated'], inplace=True
df.head()
```

Out[4]:

| | Price | Model Year | Mileage | Registered City | Engine Type | Engine Capacity | Transmission | Color | Assembly | Body Type |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2385000.0 | 2017 | 9869 | Un-Registered | Petrol | 1000 cc | Automatic | Silver | Imported | Hatchback |
| **1** | 111000.00000000001 | 2019 | 11111 | Islamabad | Petrol | 1300 cc | Automatic | White | Local | Sedan |
| **2** | 1530000.0 | 2019 | 17500 | Un-Registered | Petrol | 660 cc | Automatic | White | Local | Hatchback |
| **3** | 1650000.0 | 2019 | 9600 | Lahore | Petrol | 660 cc | Manual | White | Local | Hatchback |
| **4** | 1435000.0 | 2010 | 120000 | Islamabad | Petrol | 1300 cc | Manual | Black | Local | Sedan |

In [5]:
```
# Remove 'cc' suffix
df['Engine Capacity'] = df['Engine Capacity'].str.replace('cc', '')
df.head()
```

Out[5]:

| | Price | Model Year | Mileage | Registered City | Engine Type | Engine Capacity | Transmission | Color | Assembly | Body Type |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2385000.0 | 2017 | 9869 | Un-Registered | Petrol | 1000 | Automatic | Silver | Imported | Hatchback |
| **1** | 111000.00000000001 | 2019 | 11111 | Islamabad | Petrol | 1300 | Automatic | White | Local | Sedan |
| **2** | 1530000.0 | 2019 | 17500 | Un-Registered | Petrol | 660 | Automatic | White | Local | Hatchback |
| **3** | 1650000.0 | 2019 | 9600 | Lahore | Petrol | 660 | Manual | White | Local | Hatchback |
| **4** | 1435000.0 | 2010 | 120000 | Islamabad | Petrol | 1300 | Manual | Black | Local | Sedan |

In [6]:
```
# Convert 'Engine Capacity' column to integer
df['Engine Capacity'] = df['Engine Capacity'].astype(int)

# Convert 'Price' column to integer and remove 'Call for Price'
df['Price'] = pd.to_numeric(df['Price'], errors='coerce')
df['Price'] = df['Price'].astype(float)
df.dtypes
```

Out[6]:
```
Price              float64
Model Year           int64
Mileage              int64
Registered City     object
Engine Type         object
Engine Capacity      int32
Transmission        object
Color               object
Assembly            object
Body Type           object
dtype: object
```

In [7]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[7]:
```
Registered City    136
Engine Type          3
Transmission         2
Color               24
Assembly             2
Body Type           18
dtype: int64
```

# Segment Registered City into smaller unique value

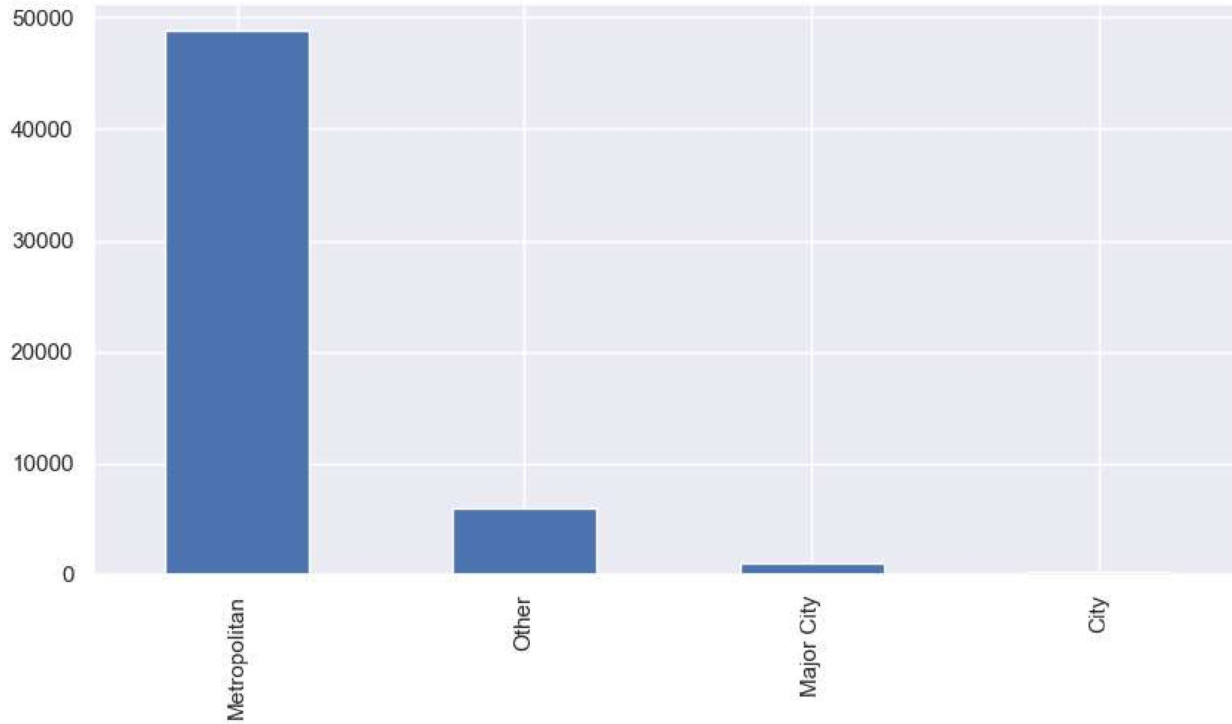
In [8]:
```
df['Registered City'].unique()
```

Out[8]:
```
array(['Un-Registered', 'Islamabad', 'Lahore', 'Rawalpindi', 'Karachi',
       'Multan', 'Faisalabad', 'Sialkot', 'Peshawar', 'Gujranwala',
       'Sargodha', 'Attock', 'Bahawalpur', 'Lower Dir', 'Chakwal',
       'Jehlum', 'Kahuta', 'Abottabad', 'Shiekhopura', 'Shikar pur',
       'Shaikhupura', 'Nowshera', 'Jhang', 'Kohat', 'Lodhran',
       'Khair Pur Mirs', 'Khushab', 'Hyderabad', 'Mirpur A.K.', 'Vehari',
       'Narowal', 'Mansahra', 'Muzaffar Gargh', 'Dera ismail khan',
       'Dadu', 'Toba Tek Singh', 'Swat', 'Nawabshah', 'Okara',
       'Mirpur khas', 'Rahim Yar Khan', 'Quetta', 'Gujrat',
       'Pak pattan sharif', 'Sanghar', 'Sahiwal', 'Hunza', 'Larkana',
       'Mardan', 'Hari pur', 'Nankana sahib', 'Gilgit', 'Chichawatni',
       'Mandi bahauddin', 'Wah cantt', 'Liaqat Pur', 'Hub-Balochistan',
       'Mian Wali', 'Khanewal', 'Karore lalisan', 'Gujar Khan',
       'Sadiqabad', 'Layyah', 'Karak', 'Bahawal Nagar', 'D.G.Khan',
       'Sukkur', 'Ahmed Pur East', 'Bhakkar', 'Sawabi', 'Kashmore',
       'Charsadda', 'Qazi ahmed', 'Dera Allah Yar', 'Thatta',
       'Muzaffarabad', 'Badin', 'Pindi gheb', 'Iskandarabad', 'Murree',
       'Muridkay', 'Nowshera cantt', 'Abdul Hakeem', 'Dir', 'Fort Abbass',
       'Moro', 'Kazi ahmed', 'Havali Lakhan', 'Tarbela', 'Khushal kot',
       'Tando Allah Yar', 'Daska', 'Wazirabad', 'Malakand Agency',
       'Hafizabad', 'Haroonabad', 'Digri', 'Kashmir', 'Daharki', 'Bannu',
       'Sara-E-Alamgir', 'Bhimber', 'Rajanpur', 'Shadiwal',
       'Ahmedpur Lamma', 'Melsi', 'Talagang', 'Lasbella', 'Mingora',
       'Kamra', 'Burewala', 'Taxila', 'Kasur', 'Kot Momin', 'Jand',
       'Khairpur', 'Sangla Hills', 'Kandh kot', 'Depal pur', 'Chiniot',
       'Akhora khattak', 'Sajawal', 'Ghotki', 'Mian Channu', 'Dijkot',
       'Rahwali', 'Sambrial', 'Jamshoro', 'Arifwala', 'Dadyal Ak',
       'Kharian', 'Pasroor', 'Kotly Ak', 'Hayatabad', 'Bhai pheru',
       'Rawala kot'], dtype=object)
```

In [9]:
```
# Define the function to segment the cities
def segment_city(city):
    if city in ['Islamabad', 'Lahore', 'Rawalpindi', 'Karachi', 'Multan', 'Faisalabad']:
        return 'Metropolitan'
    elif city in ['Peshawar', 'Gujranwala', 'Sialkot']:
        return 'Major City'
    elif city in ['Sargodha', 'Hyderabad', 'Quetta']:
        return 'City'
    else:
        return 'Other'

df['Registered City'] = df['Registered City'].apply(segment_city)
```

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

Out[10]: <AxesSubplot:>




# Cleaned dataset part 1

In [11]:
```
df.dtypes
```

Out[11]:
```
Price              float64
Model Year           int64
Mileage              int64
Registered City     object
Engine Type         object
Engine Capacity      int32
Transmission        object
Color               object
Assembly            object
Body Type           object
dtype: object
```

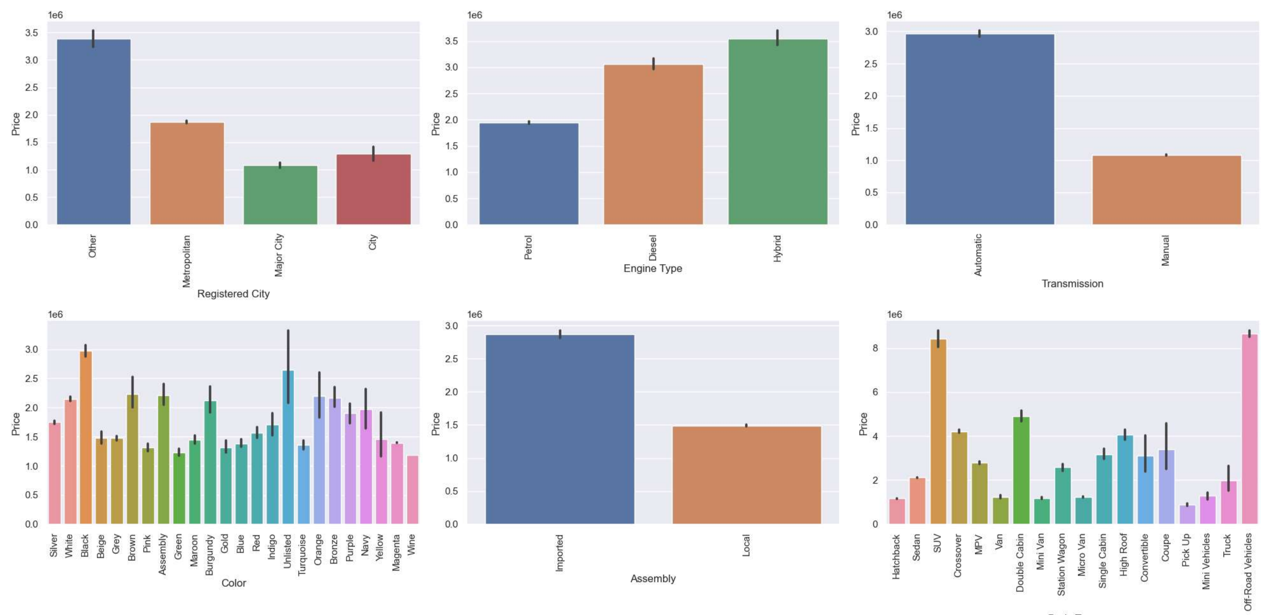
In [12]:
```
df.head()
```

Out[12]:

| | Price | Model Year | Mileage | Registered City | Engine Type | Engine Capacity | Transmission | Color | Assembly | Body Type |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2385000.0 | 2017 | 9869 | Other | Petrol | 1000 | Automatic | Silver | Imported | Hatchback |
| 1 | 111000.0 | 2019 | 11111 | Metropolitan | Petrol | 1300 | Automatic | White | Local | Sedan |
| 2 | 1530000.0 | 2019 | 17500 | Other | Petrol | 660 | Automatic | White | Local | Hatchback |
| 3 | 1650000.0 | 2019 | 9600 | Metropolitan | Petrol | 660 | Manual | White | Local | Hatchback |
| 4 | 1435000.0 | 2010 | 120000 | Metropolitan | Petrol | 1300 | Manual | Black | Local | Sedan |

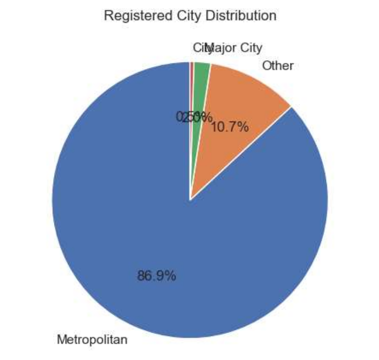
# Exploratory Data Analysis

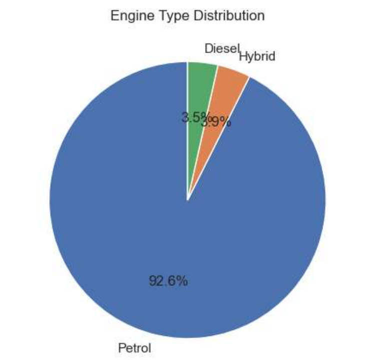
In [13]:
```
# list of categorical variables to plot
cat_vars = ['Registered City', 'Engine Type', 'Transmission', 'Color', 'Assembly', 'Body Type'

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(20, 10))
axs = axs.flatten()

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

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```

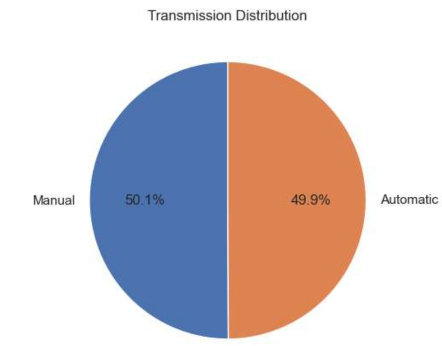


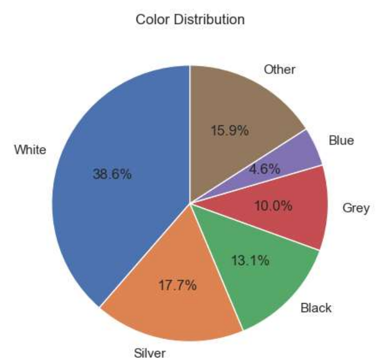
In [14]:
```
# Specify the maximum number of categories to show individually
max_categories = 5

cat_vars = ['Registered City', 'Engine Type', 'Transmission', 'Color', 'Assembly', 'Body Type'

# Create a figure and axes
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 15))

# Create a pie chart for each categorical variable
for i, var in enumerate(cat_vars):
    if i < len(axs.flat):
        # Count the number of occurrences for each category
        cat_counts = df[var].value_counts()

        # Group categories beyond the top max_categories as 'Other'
        if len(cat_counts) > max_categories:
            cat_counts_top = cat_counts[:max_categories]
            cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), index=['Other'])
            cat_counts = cat_counts_top.append(cat_counts_other)

        # Create a pie chart
        axs.flat[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)

        # Set a title for each subplot
        axs.flat[i].set_title(f'{var} Distribution')

# Adjust spacing between subplots
fig.tight_layout()

# Show the plot
plt.show()
```

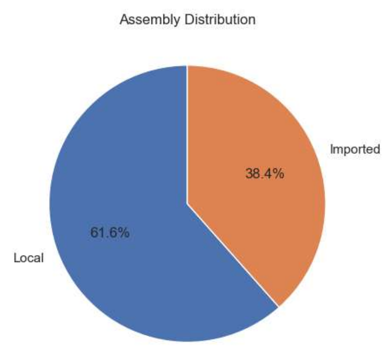
```
C:\Users\Michael\AppData\Local\Temp\ipykernel_25620\2115016549.py:19: FutureWarning: The seri
es.append method is deprecated and will be removed from pandas in a future version. Use panda
s.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_25620\2115016549.py:19: FutureWarning: The seri
es.append method is deprecated and will be removed from pandas in a future version. Use panda
s.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
```

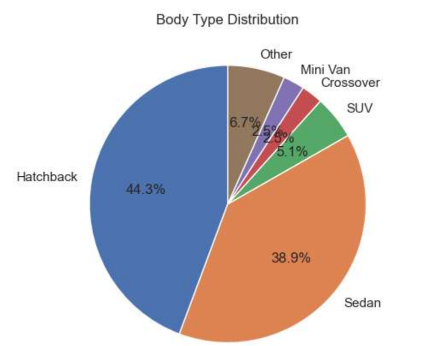

Registered City Distribution
City
Major City
Other
10.7%
86.9%
Metropolitan


Engine Type Distribution
Diesel
Hybrid
3.5%
3.9%
92.6%
Petrol

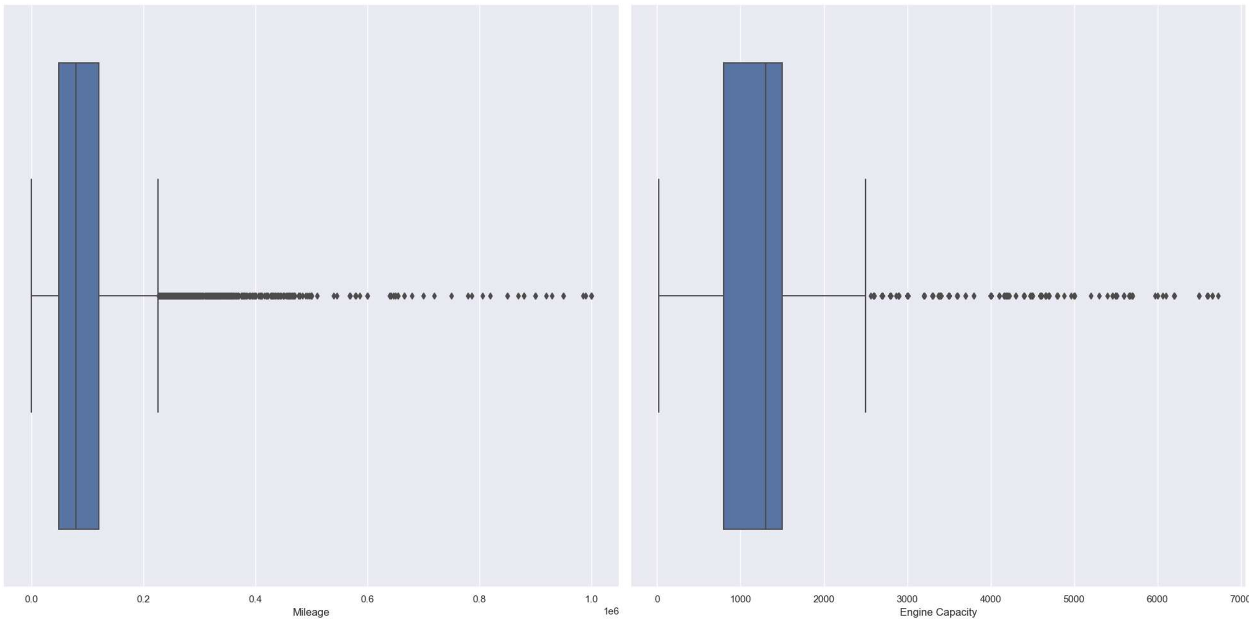

Transmission Distribution
Manual
50.1%
49.9%
Automatic


Color Distribution
Other
15.9%
Blue
4.6%
White
38.6%
10.0%
Grey
13.1%
17.7%
Black
Silver

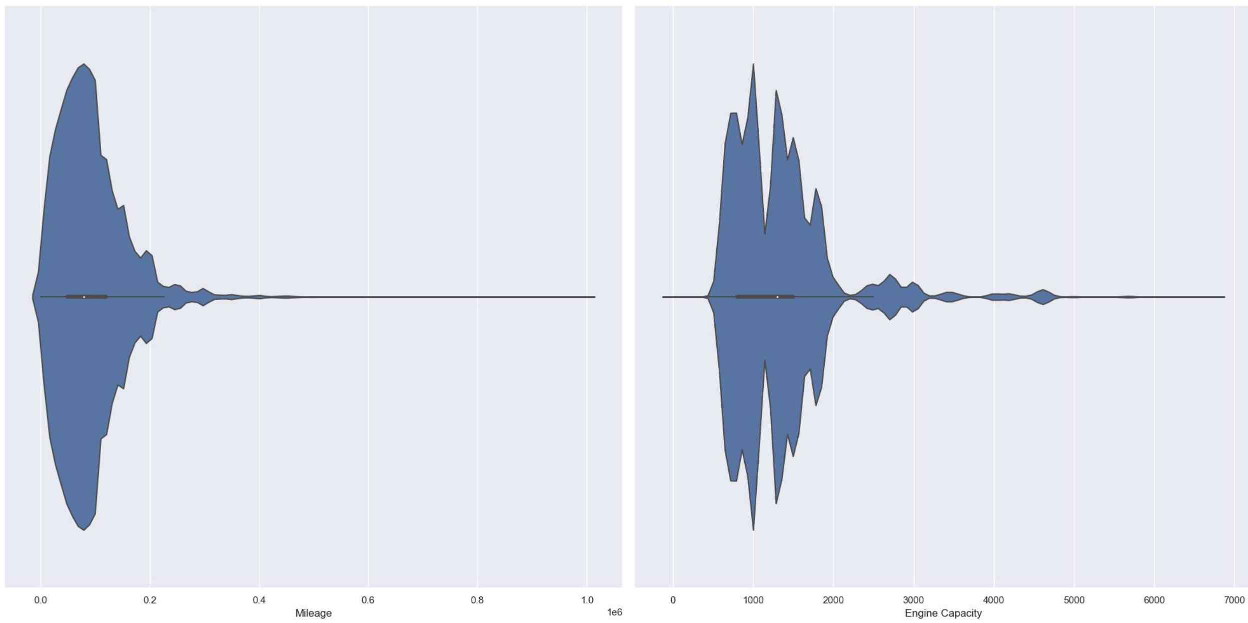

Assembly Distribution
Imported
38.4%
61.6%
Local


Body Type Distribution
Other
Mini Van
Crossover
SUV
6.7%
2.5%
5.1%
Hatchback
44.3%
38.9%
Sedan

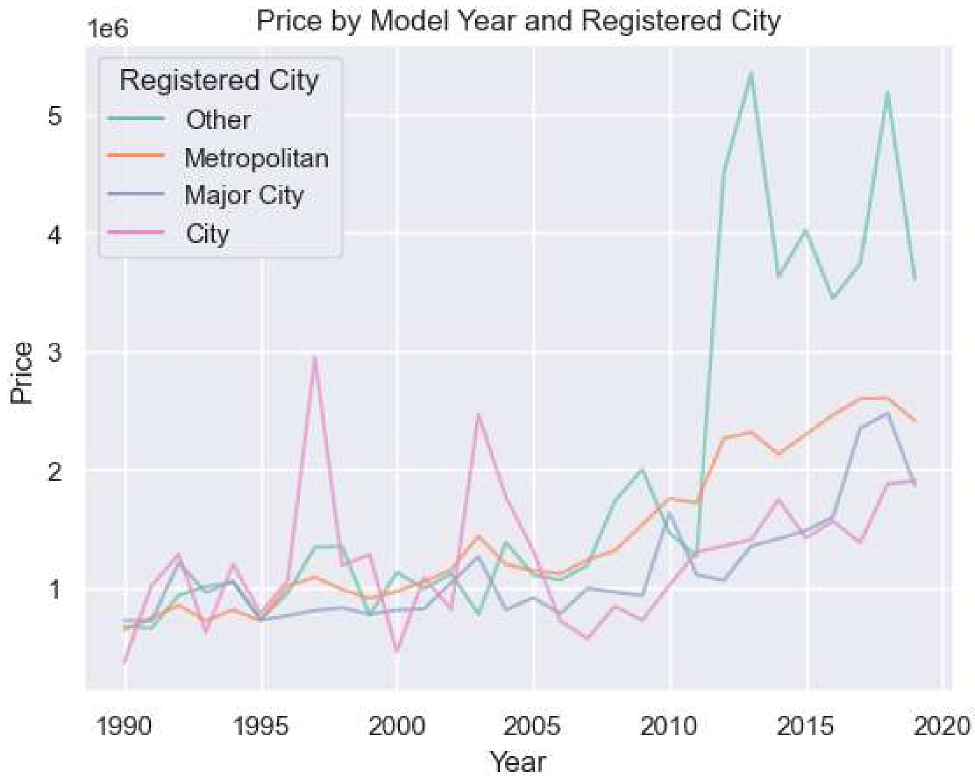
In [15]:
```
num_vars = ['Mileage', 'Engine Capacity']

fig, axs = plt.subplots(nrows=1, ncols=2, 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()
```



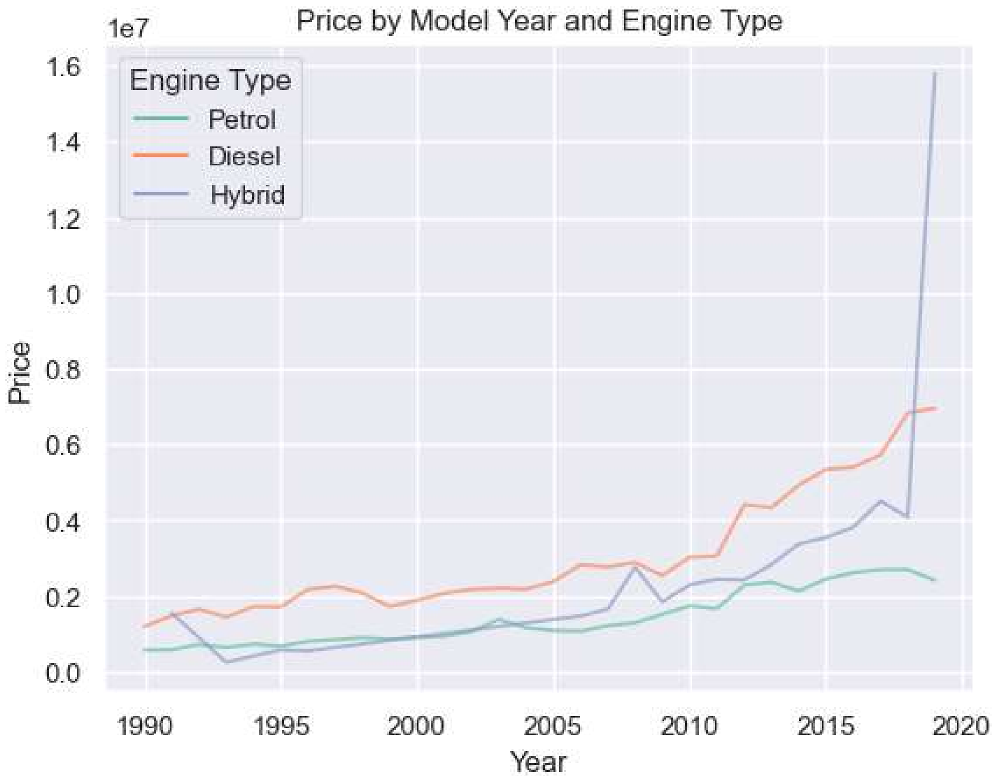
In [16]:
```
num_vars = ['Mileage', 'Engine Capacity']

fig, axs = plt.subplots(nrows=1, ncols=2, 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()
```



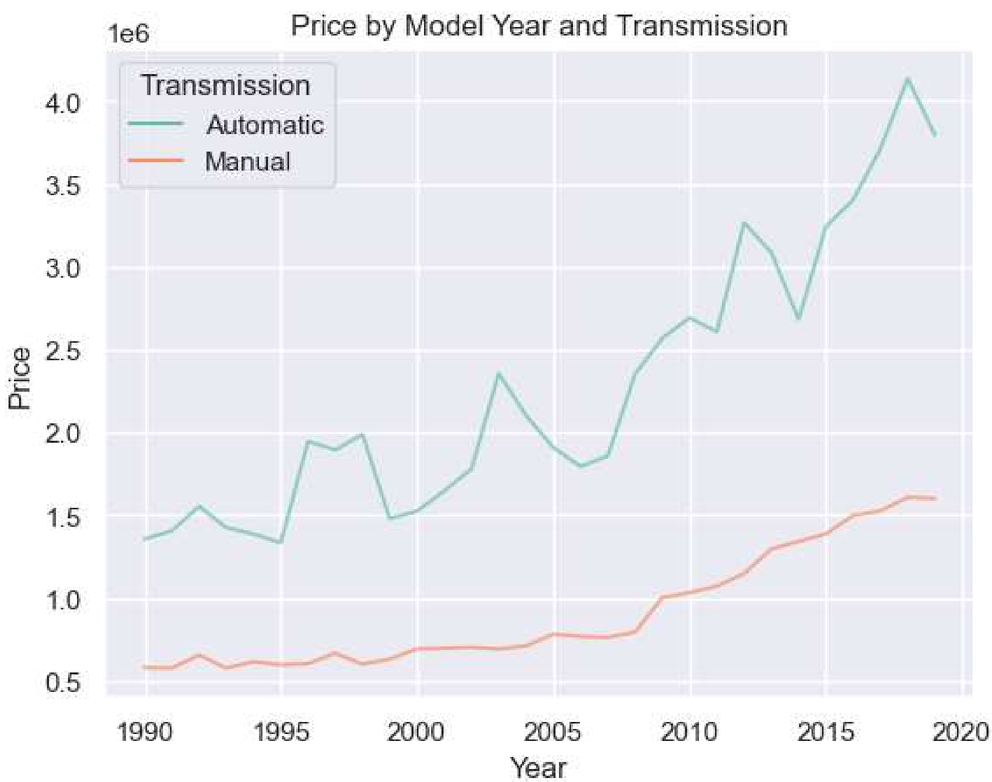
In [17]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Model Year', y='Price', hue='Registered City', data=df, ci=None, estimator='me

plt.title("Price by Model Year and Registered City")
plt.xlabel("Year")
plt.ylabel("Price")

plt.show()
```



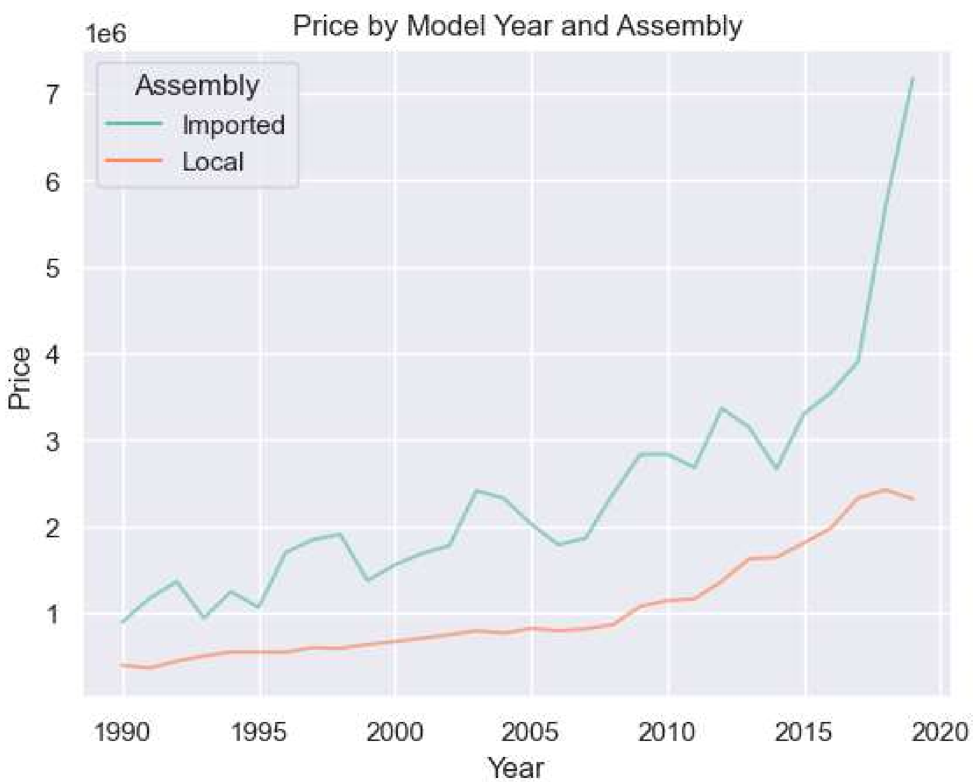
In [18]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Model Year', y='Price', hue='Engine Type', data=df, ci=None, estimator='mean',

plt.title("Price by Model Year and Engine Type")
plt.xlabel("Year")
plt.ylabel("Price")

plt.show()
```



In [19]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Model Year', y='Price', hue='Transmission', data=df, ci=None, estimator='mean'

plt.title("Price by Model Year and Transmission")
plt.xlabel("Year")
plt.ylabel("Price")

plt.show()
```

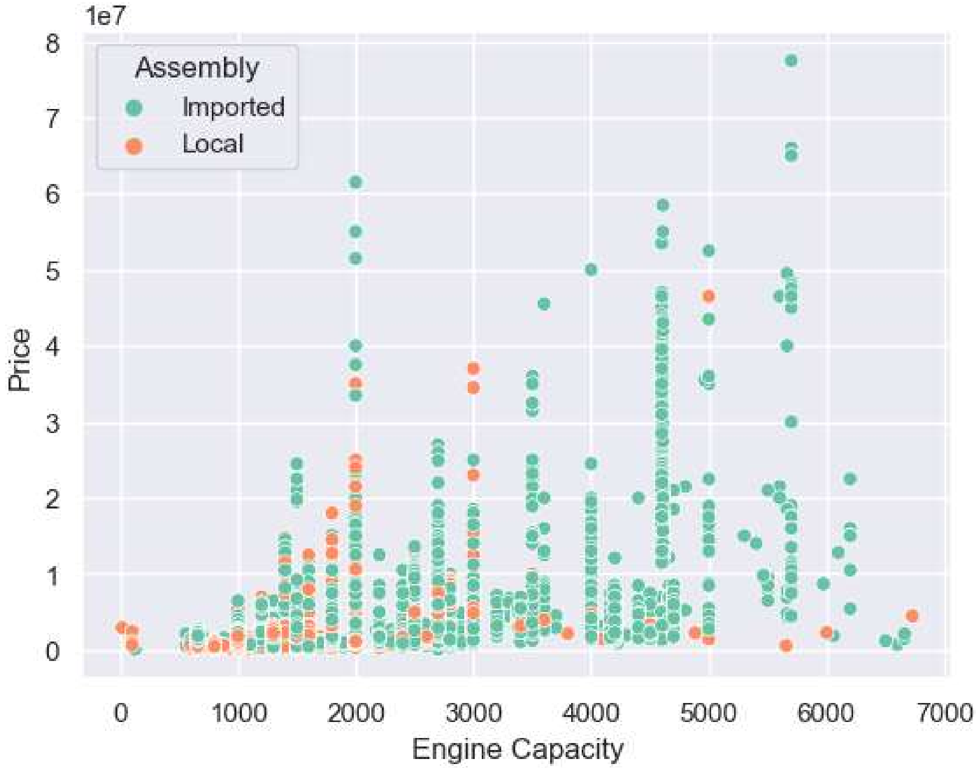


In [20]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Model Year', y='Price', hue='Assembly', data=df, ci=None, estimator='mean', al

plt.title("Price by Model Year and Assembly")
plt.xlabel("Year")
plt.ylabel("Price")

plt.show()
```



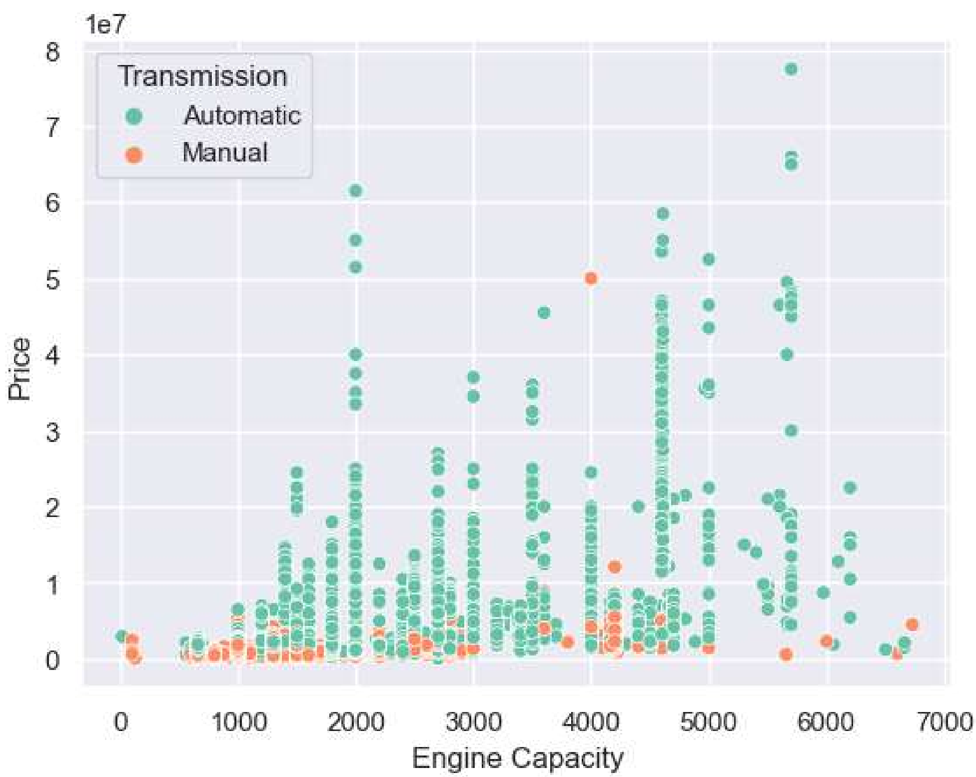
In [22]:
```
sns.scatterplot(x='Engine Capacity', y='Price', hue='Assembly', data=df)
```

Out[22]: <AxesSubplot:xlabel='Engine Capacity', ylabel='Price'>



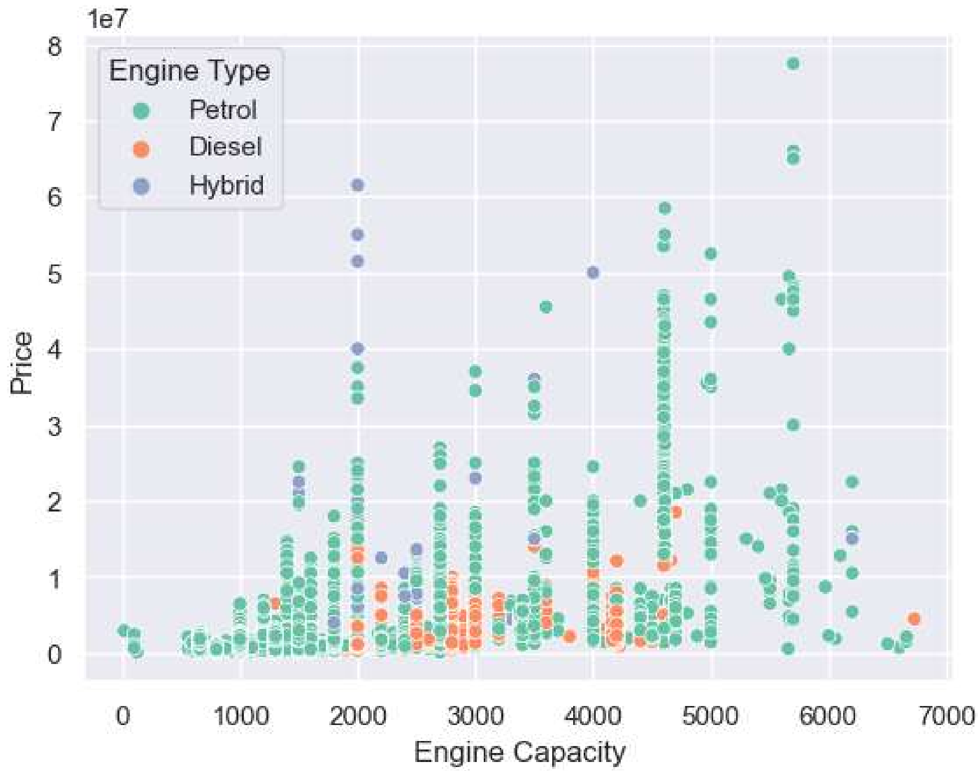

In [23]:
```
sns.scatterplot(x='Engine Capacity', y='Price', hue='Transmission', data=df)
```

Out[23]: <AxesSubplot:xlabel='Engine Capacity', ylabel='Price'>



In [24]:
```
sns.scatterplot(x='Engine Capacity', y='Price', hue='Engine Type', data=df)
```

Out[24]: <AxesSubplot:xlabel='Engine Capacity', ylabel='Price'>




In [25]:
```
sns.scatterplot(x='Engine Capacity', y='Price', hue='Registered City', data=df)
```

Out[25]: <AxesSubplot:xlabel='Engine Capacity', ylabel='Price'>

# Data Preprocessing Part 2

In [26]:
```
# Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[26]:
```
Body Type      11.513544
Engine Type     2.281707
Price           2.265689
dtype: float64
```

In [27]:
```
df.shape
```

Out[27]:
```
(56186, 10)
```

In [28]:
```
# Remove the null value because its very low
df.dropna(inplace=True)
df.shape
```

Out[28]:
```
(47564, 10)
```

# Label Encoding for each object dataype

In [29]:
```
# 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()}")
```

```
Registered City: ['Other' 'Metropolitan' 'Major City' 'City']
Engine Type: ['Petrol' 'Diesel' 'Hybrid']
Transmission: ['Automatic' 'Manual']
Color: ['Silver' 'White' 'Black' 'Beige' 'Grey' 'Brown' 'Pink' 'Assembly'
 'Maroon' 'Burgundy' 'Gold' 'Blue' 'Red' 'Indigo' 'Unlisted' 'Green'
 'Turquoise' 'Orange' 'Bronze' 'Purple' 'Yellow' 'Navy' 'Magenta' 'Wine']
Assembly: ['Imported' 'Local']
Body Type: ['Hatchback' 'Sedan' 'SUV' 'Crossover' 'MPV' 'Van' 'Double Cabin'
 'Mini Van' 'Station Wagon' 'Micro Van' 'Single Cabin' 'High Roof'
 'Convertible' 'Coupe' 'Pick Up' 'Mini Vehicles' 'Truck'
 'Off-Road Vehicles']
```

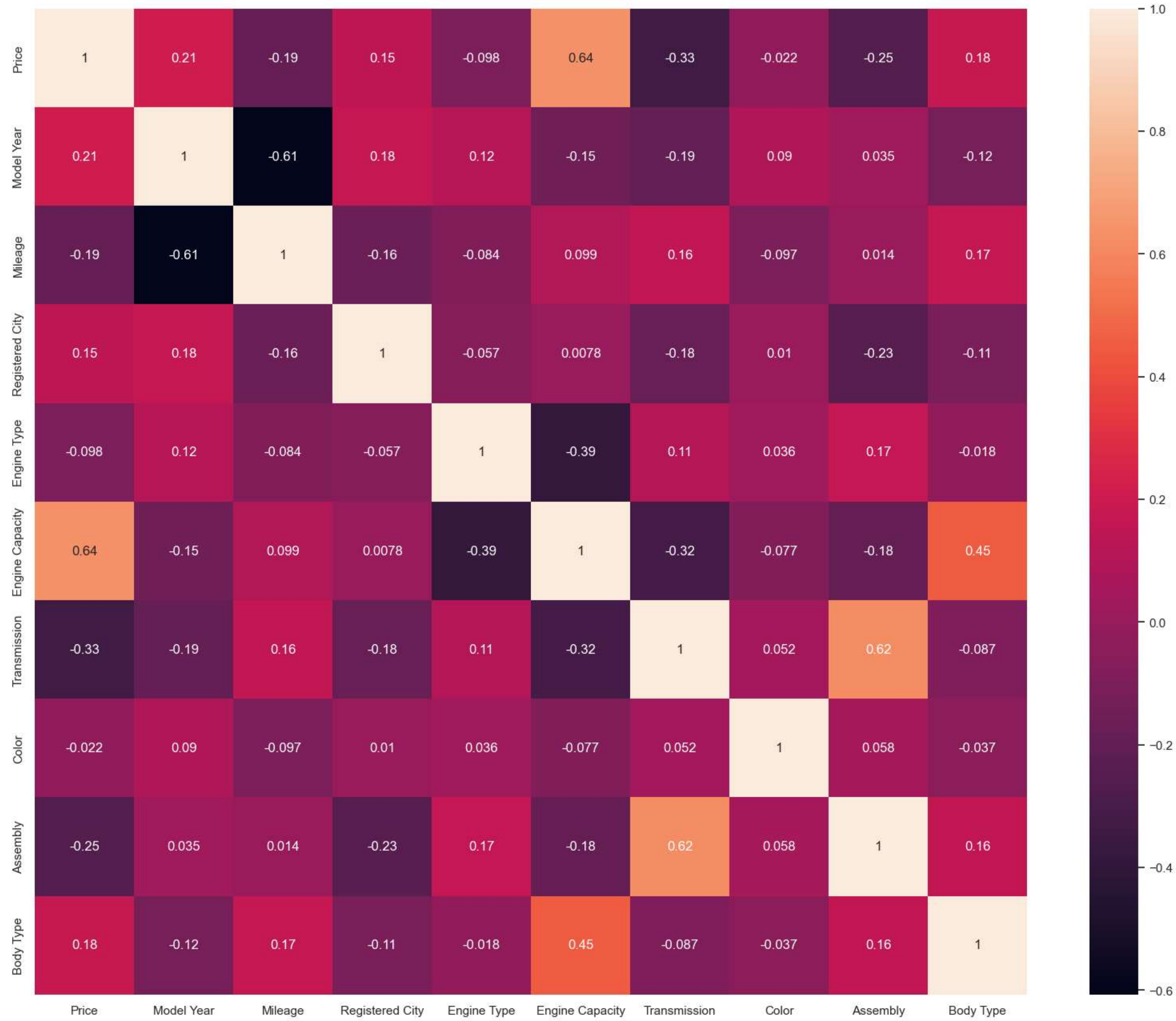
In [30]:
```
from sklearn import preprocessing

# 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()}")
```

```
Registered City: [3 2 1 0]
Engine Type: [2 0 1]
Transmission: [0 1]
Color: [18 21  2  1  9  5 15  0 12  6  7  3 17 10 20  8 19 14  4 16 23 13 11 22]
Assembly: [0 1]
Body Type: [ 4 13 12  2  6 17  3  8 15  7 14  5  0  1 11  9 16 10]
```

# Correlation Heatmap

In [31]:
```
#Correlation Heatmap (print the correlation score each variables)
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[31]: <AxesSubplot:>




# Train Test Split

In [32]:
```
from sklearn.model_selection import train_test_split
# Select the features (X) and the target variable (y)
X = df.drop('Price', axis=1)
y = df['Price']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

# Remove the Outlier from train data using Z-Score

In [33]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['Mileage', 'Engine Capacity']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

# Decision Tree Regressor

In [34]:
```
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'],
    'random_state': [0, 42]
}

# 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': 3, 'min_samples_split': 8, 'rand
om_state': 0}
```

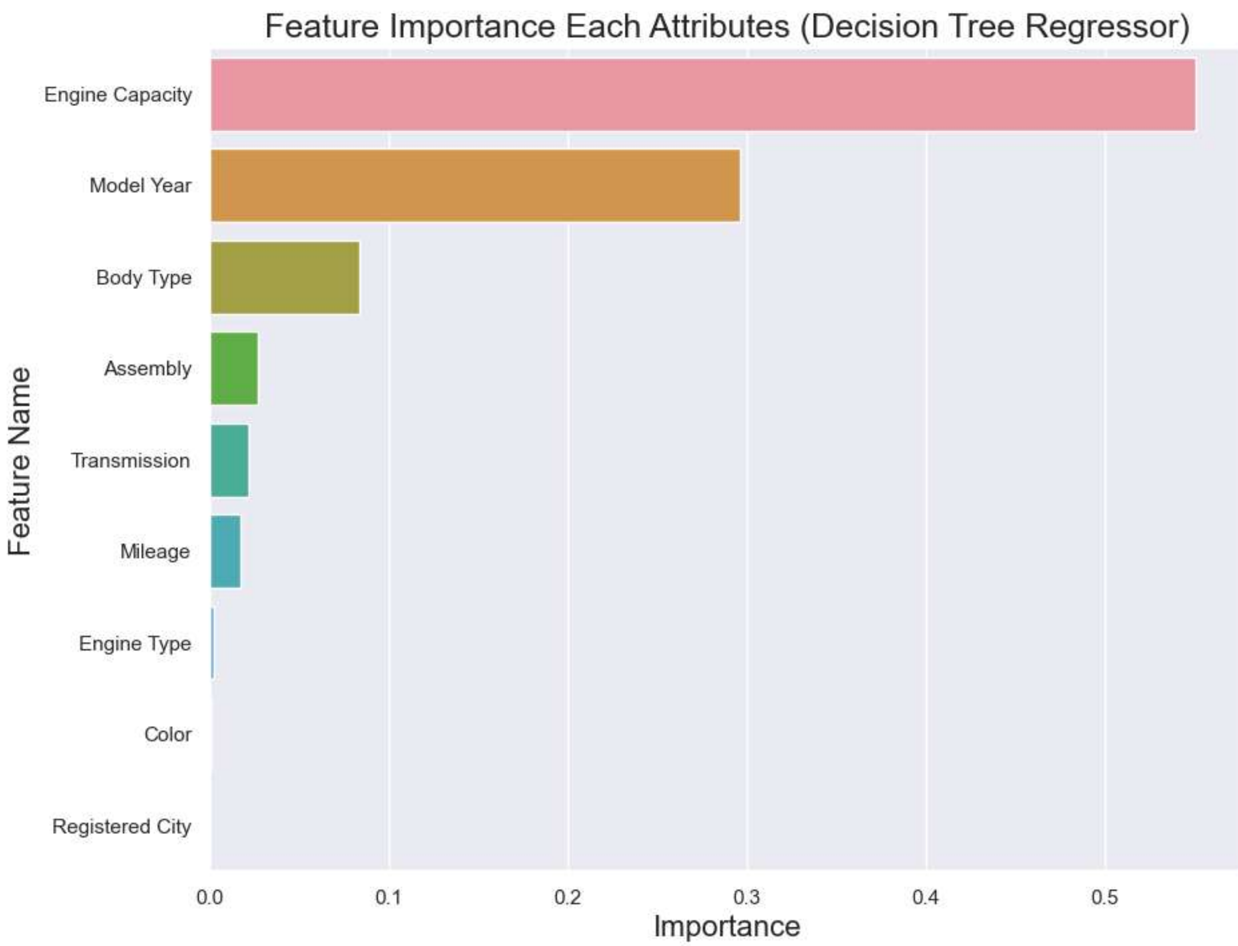
In [35]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=8, max_features='auto', min_samples_lea
dtree.fit(X_train, y_train)
```

Out[35]:
```
DecisionTreeRegressor(max_depth=8, max_features='auto', min_samples_leaf=3,
                      min_samples_split=8, random_state=0)
```

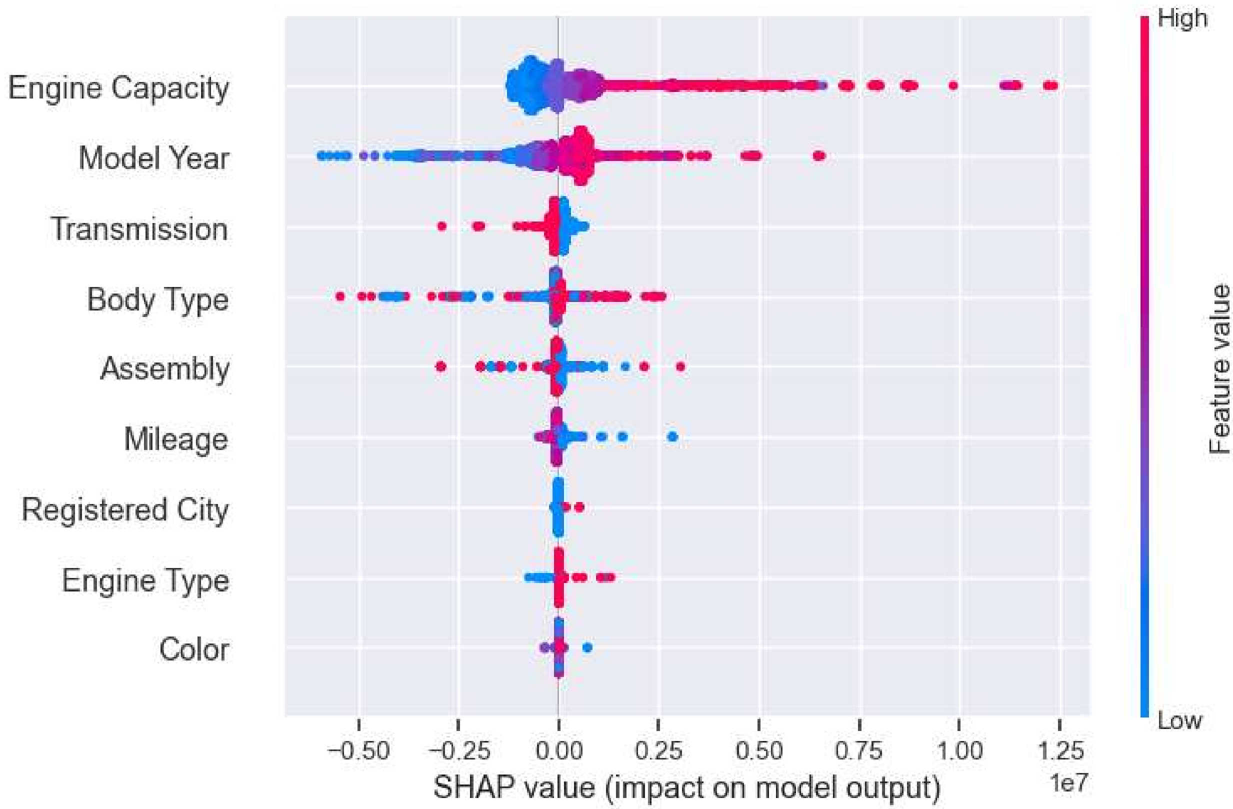
In [36]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = 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 354766.0090956561
MAPE is 0.13888668625590878
MSE is 2385067100555.926
R2 score is 0.7005650417501269
RMSE score is 1544366.245602359
```

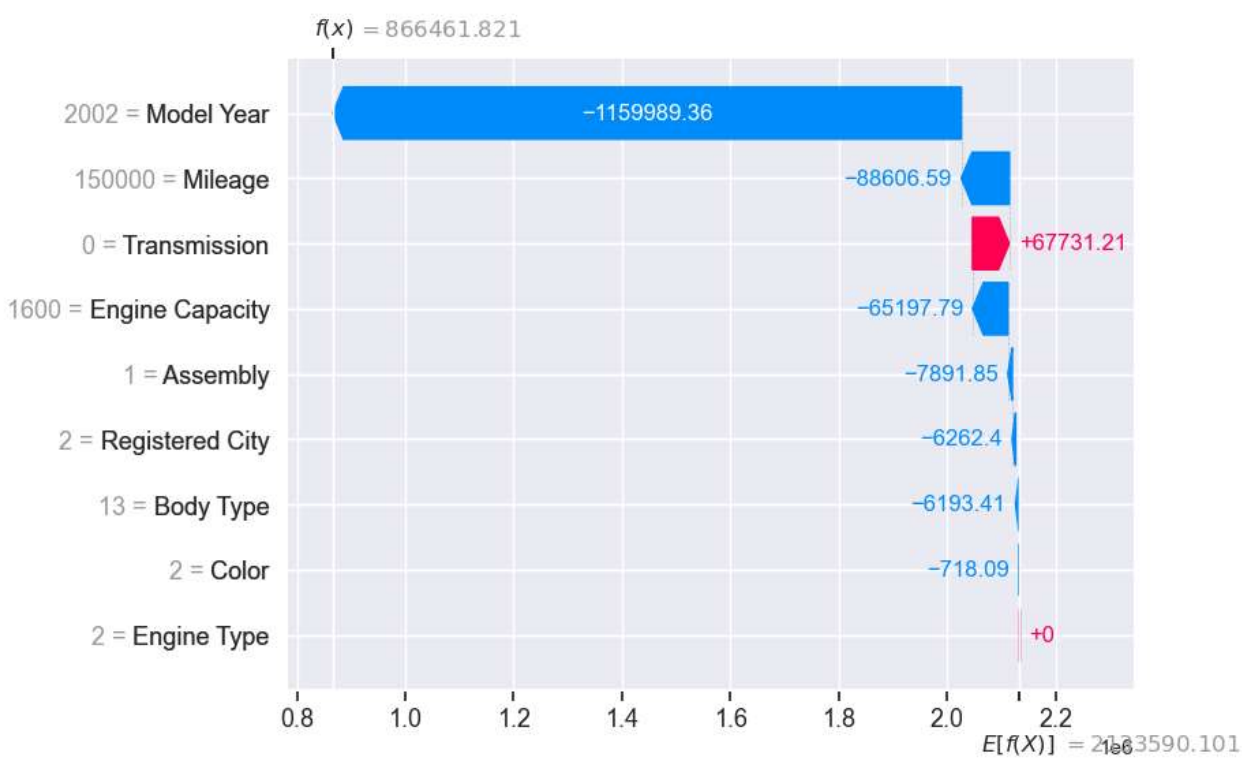
In [37]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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 [38]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```




In [39]:
```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```



# # AdaBoost Regressor

In [41]:
```
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],
          'random_state': [0, 42]}

# 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': 0.01, 'n_estimators': 150, 'random_state': 0}
```

In [42]:
```
from sklearn.ensemble import RandomForestRegressor
abr = AdaBoostRegressor(random_state=0, learning_rate=0.01, n_estimators=150)
abr.fit(X_train, y_train)
```

Out[42]:
```
AdaBoostRegressor(learning_rate=0.01, n_estimators=150, random_state=0)
```

In [43]:
```
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 709114.1171410958
MAPE is 0.4185430836642879
MSE is 3812927029936.1406
R2 score is 0.5213033437287274
RMSE score is 1952671.7670761107
```

In [44]:
```
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()
```