

```
In [1]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import math

df = pd.read_csv('Car_sales.csv')
df.head()
```

```
Out[1]:
```

	Manufacturer	Model	Sales_in_thousands	__year_resale_value	Vehicle_type	Price_in_tho
0	Acura	Integra	16.919	16.360	Passenger	
1	Acura	TL	39.384	19.875	Passenger	
2	Acura	CL	14.114	18.225	Passenger	
3	Acura	RL	8.588	29.725	Passenger	
4	Audi	A4	20.397	22.255	Passenger	

```
In [2]: df.isnull().sum()
```

```
Out[2]: Manufacturer      0
Model                    0
Sales_in_thousands      0
__year_resale_value     36
Vehicle_type            0
Price_in_thousands      2
Engine_size             1
Horsepower              1
Wheelbase               1
Width                   1
Length                  1
Curb_weight             2
Fuel_capacity           1
Fuel_efficiency         3
Latest_Launch           0
Power_perf_factor       2
dtype: int64
```

```
In [3]: len(df)
```

```
Out[3]: 157
```

```
In [4]: missing_percentage = df.isnull().mean() * 100
columns_with_missing_30 = missing_percentage[missing_percentage > 30]
print("Columns with more than 30% missing values:")
print(columns_with_missing_30)
```

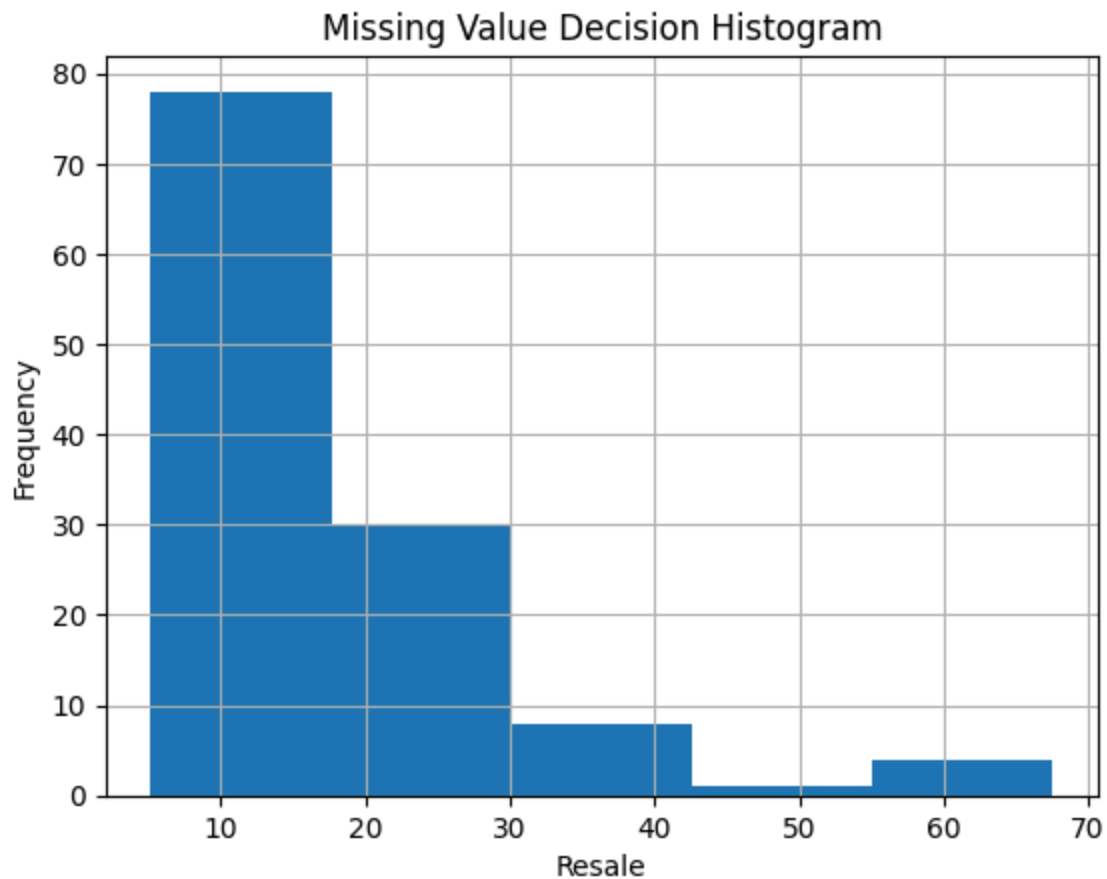
```
Columns with more than 30% missing values:
Series([], dtype: float64)
```

```
In [5]: print(df['__year_resale_value'].median())  
print(df['__year_resale_value'].mean())
```

14.18

18.07297520661157

```
In [6]: df['__year_resale_value'].hist(bins=5)  
plt.title('Missing Value Decision Histogram')  
plt.xlabel('Resale')  
plt.ylabel('Frequency')  
plt.show()
```



```
In [7]: df.fillna({'__year_resale_value': df['__year_resale_value'].median()}, inplace=True)
```

```
In [8]: df.isnull().sum()
```

```
Out[8]: Manufacturer      0
      Model                0
      Sales_in_thousands  0
      __year_resale_value  0
      Vehicle_type        0
      Price_in_thousands  2
      Engine_size         1
      Horsepower          1
      Wheelbase           1
      Width               1
      Length              1
      Curb_weight         2
      Fuel_capacity       1
      Fuel_efficiency     3
      Latest_Launch      0
      Power_perf_factor   2
      dtype: int64
```

```
In [9]: numerical_df = df.select_dtypes(include=['number'])

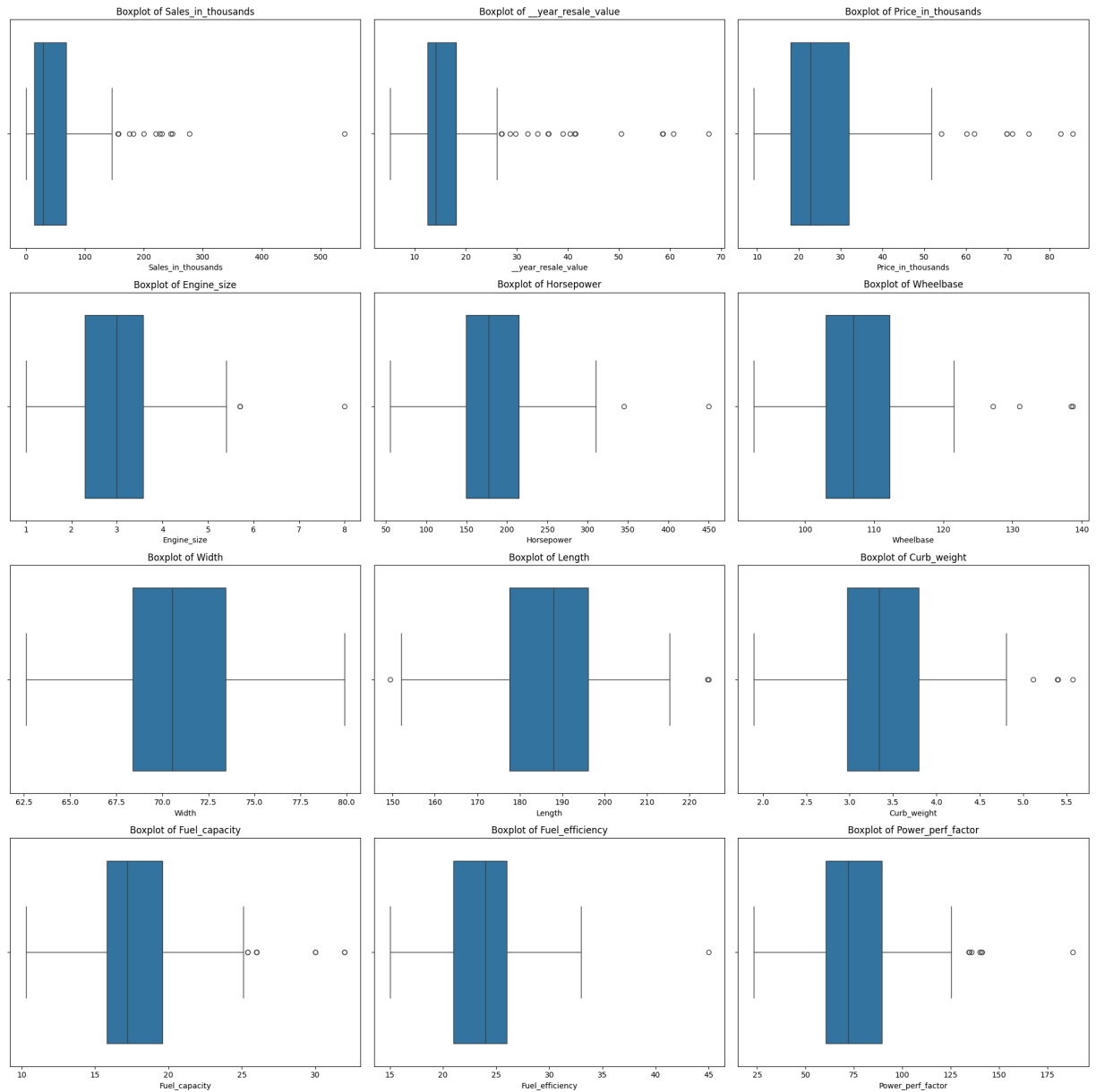
num_cols = numerical_df.columns
n = len(num_cols)

rows = math.ceil(n / 3) # Number of rows needed

plt.figure(figsize=(20, 5 * rows))

for i, col in enumerate(num_cols):
    plt.subplot(rows, 3, i + 1) # 3 plots per row
    sns.boxplot(x=numerical_df[col])
    plt.title(f"Boxplot of {col}")

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



```
In [10]: data = df.copy()

def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = (column < lower_bound) | (column > upper_bound)
    return outliers

# Numeric columns only
numeric_cols = data.select_dtypes(include='number').columns

# Check each column for outliers
outlier_flag = {}
for col in numeric_cols:
    mask = detect_outliers_iqr(data[col])
    outlier_flag[col] = "Yes" if mask.any() else "No"
```

```
# Convert to DataFrame for nice display
outlier_df = pd.DataFrame(list(outlier_flag.items()), columns=['Column', 'Has Outli
print(outlier_df)
```

	Column	Has Outliers
0	Sales_in_thousands	Yes
1	__year_resale_value	Yes
2	Price_in_thousands	Yes
3	Engine_size	Yes
4	Horsepower	Yes
5	Wheelbase	Yes
6	Width	No
7	Length	Yes
8	Curb_weight	Yes
9	Fuel_capacity	Yes
10	Fuel_efficiency	Yes
11	Power_perf_factor	Yes

```
In [11]: data = df.copy()

def detect_outliers_zscore(column, threshold=3):
    mean = np.mean(column)
    std_dev = np.std(column)

    # Avoid division by zero
    if std_dev == 0:
        return np.zeros(len(column), dtype=bool)

    z_scores = (column - mean) / std_dev
    return np.abs(z_scores) > threshold

# Only evaluate numeric columns
numeric_cols = data.select_dtypes(include='number').columns

for col in numeric_cols:
    data[f'{col}_Outlier'] = detect_outliers_zscore(data[col])

# Filter rows where ANY outlier flag is True
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]

outliers_only
```

Out[11]:

	Manufacturer	Model	Sales_in_thousands	_year_resale_value	Vehicle_type	Price_ir
18	Cadillac	Escalade	14.785	14.180	Car	
26	Chevrolet	Metro	21.855	5.160	Passenger	
39	Dodge	Viper	0.916	58.470	Passenger	
40	Dodge	Ram Pickup	227.061	15.060	Car	
41	Dodge	Ram Wagon	16.767	15.510	Car	
42	Dodge	Ram Van	31.038	13.425	Car	
43	Dodge	Dakota	111.313	11.260	Car	
52	Ford	Explorer	276.747	16.640	Car	
56	Ford	F-Series	540.561	15.075	Car	
74	Lexus	LX470	9.126	14.180	Car	
78	Lincoln	Navigator	22.925	14.180	Car	
94	Mercedes-B	S-Class	16.774	50.375	Passenger	
95	Mercedes-B	SL-Class	3.311	58.600	Passenger	
99	Mercedes-B	CL500	0.954	14.180	Passenger	
125	Porsche	Carrera Coupe	1.280	60.625	Passenger	
126	Porsche	Carrera Cabrio	1.866	67.550	Passenger	

16 rows × 28 columns



In [12]:

```
data = df.copy()

# Function to replace outliers with NaN using IQR
def replace_outliers_with_nan(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Replace outliers with NaN
    return column.where((column >= lower_bound) & (column <= upper_bound), np.nan)

# Select numeric columns
numeric_cols = data.select_dtypes(include='number').columns
```

```

# Apply to each numeric column
for col in numeric_cols:
    data[col] = replace_outliers_with_nan(data[col])

# Count null values after replacing outliers
null_counts = data[numeric_cols].isnull().sum()

print("Sum of null values in each numerical column:")
print(null_counts)

```

Sum of null values in each numerical column:

```

Sales_in_thousands    12
__year_resale_value    17
Price_in_thousands    11
Engine_size            4
Horsepower             3
Wheelbase              5
Width                  1
Length                 4
Curb_weight            6
Fuel_capacity          10
Fuel_efficiency         4
Power_perf_factor      9
dtype: int64

```

In [13]: `from sklearn.impute import KNNImputer`

```

# Separate numeric and non-numeric columns
numeric_cols = data.select_dtypes(include='number').columns
non_numeric_cols = data.select_dtypes(exclude='number').columns

# Apply KNN imputer only on numeric columns
imputer = KNNImputer(n_neighbors=2)
numeric_imputed = pd.DataFrame(
    imputer.fit_transform(data[numeric_cols]),
    columns=numeric_cols
)

# Recombine numeric + non-numeric columns
data_imputed_knn = pd.concat([numeric_imputed, data[non_numeric_cols]], axis=1)

# Check missing values
data_imputed_knn.isnull().sum()

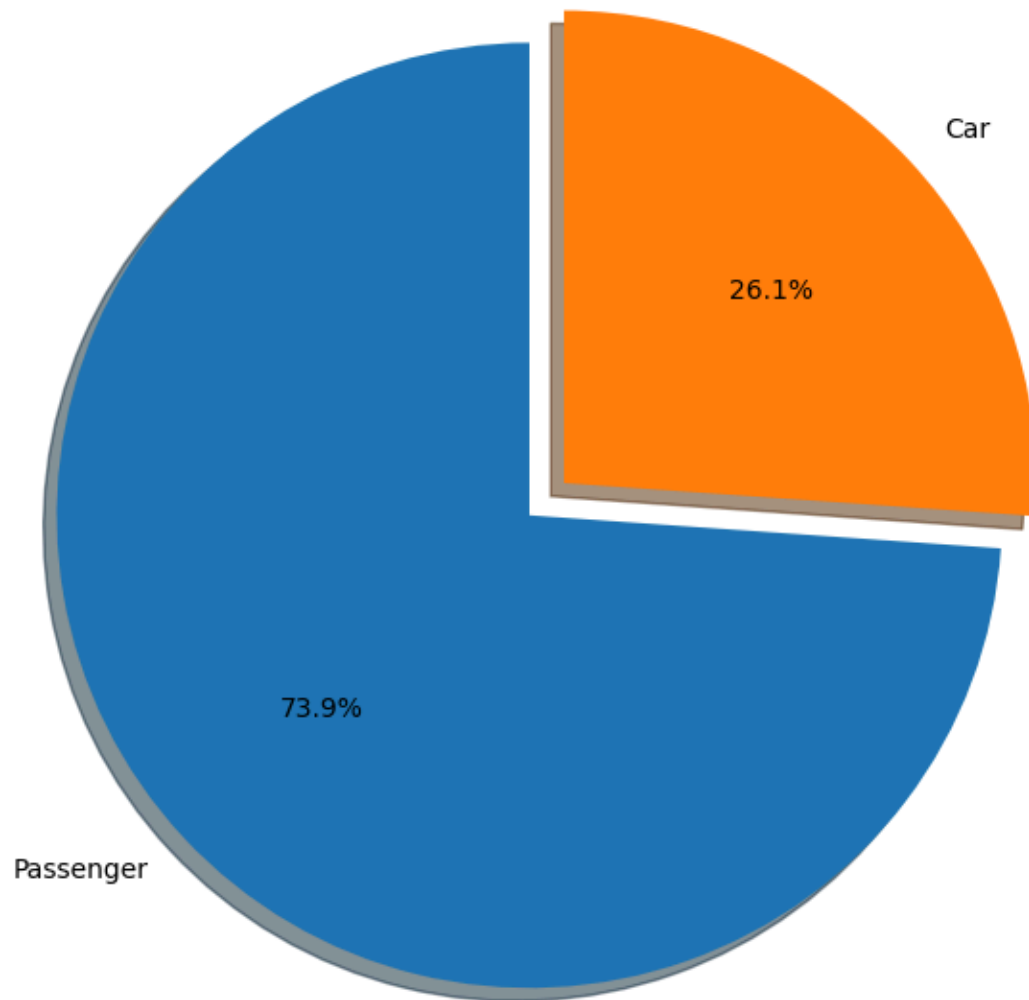
```

```
Out[13]: Sales_in_thousands      0
         __year_resale_value      0
         Price_in_thousands      0
         Engine_size              0
         Horsepower               0
         Wheelbase                0
         Width                    0
         Length                   0
         Curb_weight              0
         Fuel_capacity             0
         Fuel_efficiency           0
         Power_perf_factor         0
         Manufacturer             0
         Model                    0
         Vehicle_type             0
         Latest_Launch            0
         dtype: int64
```

```
In [14]: category_counts = df['Vehicle_type'].value_counts()
         explode = [0.1,0]

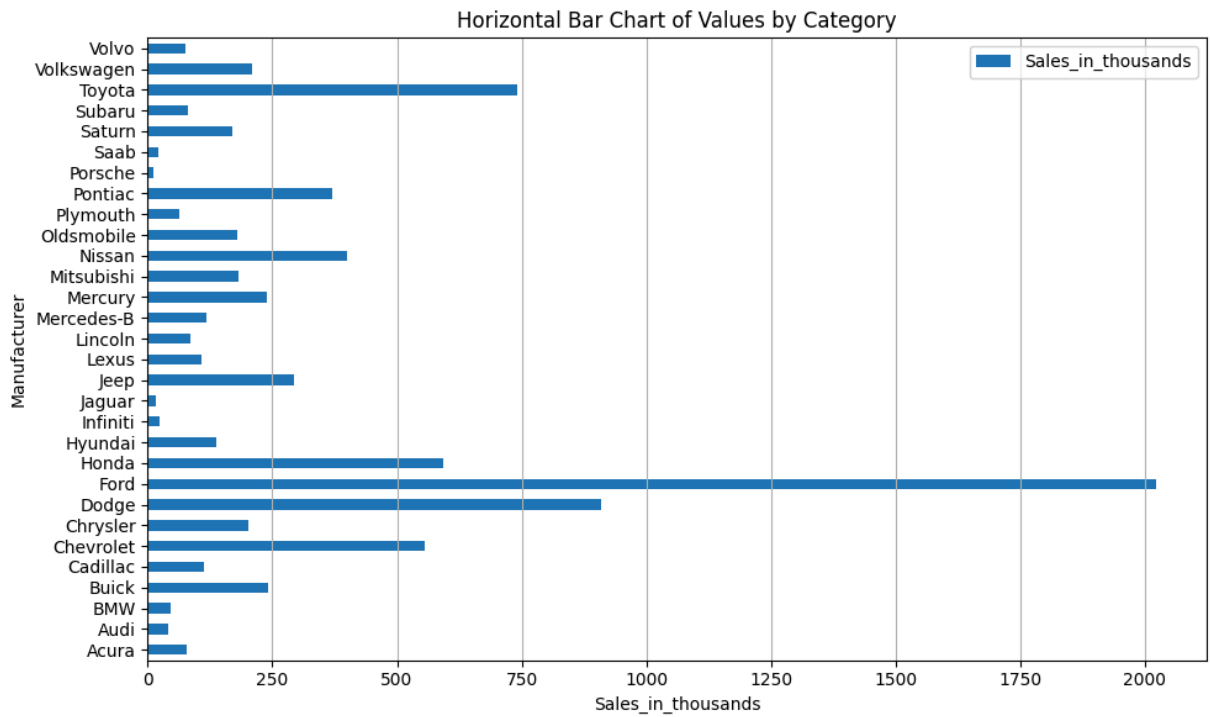
         # Create the pie chart
         plt.figure(figsize=(8, 8)) # Optional: Adjust figure size
         plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', explode=explode)
         plt.title('Distribution of Vehicle_type')
         plt.show()
```

Distribution of Vehicle_type



```
In [15]: grouped_df = df.groupby('Manufacturer')['Sales_in_thousands'].sum().reset_index()

grouped_df.plot.barh(y='Sales_in_thousands', x='Manufacturer', figsize=(10, 6), title='Sales by Manufacturer')
plt.xlabel('Sales_in_thousands')
plt.ylabel('Manufacturer')
plt.tight_layout() # Adjust layout to prevent labels from overlapping
plt.grid(axis='x')
plt.show()
```



```
In [16]: plt.hist(data['Engine_size'], bins=5, color='skyblue', edgecolor='black')
plt.title('Histogram of Engine_size')
plt.xlabel('Engine_size')
plt.ylabel('Frequency')
plt.show()
```

