```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
car_details = pd.read_csv('car_details.csv')
```

```
list(car_details.columns)
```

```
['name',
 'year',
 'selling_price',
 'km_driven',
 'fuel',
 'seller_type',
 'transmission',
 'owner',
 'mileage',
 'engine',
 'max_power',
 'torque',
 'seats']
```

```
car_details
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.5 kmpl	1197 CC	82.85 bhp	113.7Nm@ 4000rpm	5.0
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.8 kmpl	1493 CC	110 bhp	24@ 1,900-2,750(kgm@ rpm)	5.0
	Maruti Swift							First	19.3	1248		190Nm@	

```
car_details.head()
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0

```
car_details.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           8128 non-null   object
 1   year           8128 non-null   int64
 2   selling_price  8128 non-null   int64
 3   km_driven      8128 non-null   int64
 4   fuel           8128 non-null   object
 5   seller_type    8128 non-null   object
 6   transmission   8128 non-null   object
 7   owner          8128 non-null   object
 8   mileage        7907 non-null   object
 9   engine         7907 non-null   object
 10  max_power      7913 non-null   object
 11  torque         7906 non-null   object
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
```

```
car_details.describe()
```

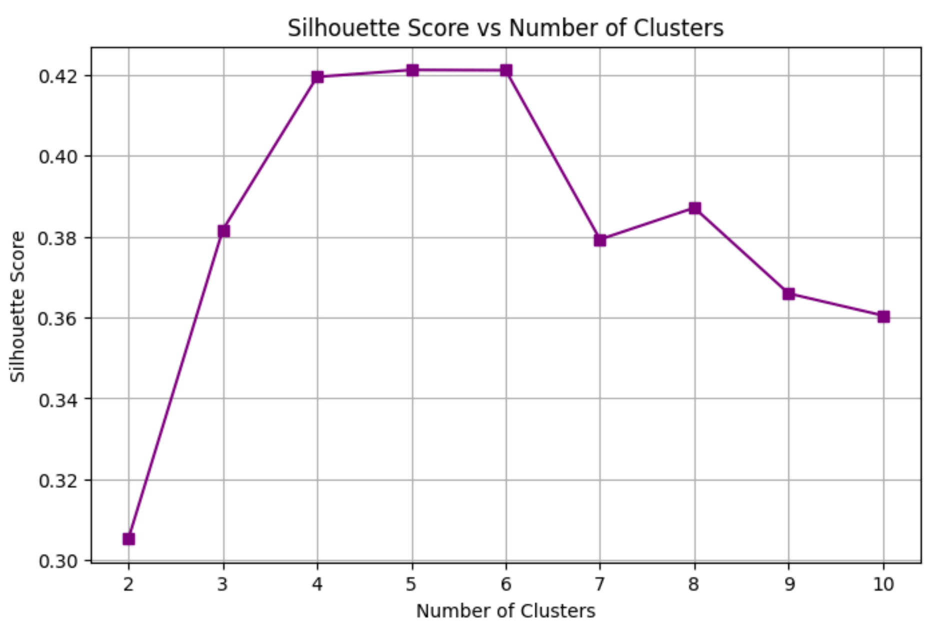
	year	selling_price	km_driven	seats
count	8128.000000	8.128000e+03	8.128000e+03	7907.000000
mean	2013.804011	6.382718e+05	6.981951e+04	5.416719
std	4.044249	8.062534e+05	5.655055e+04	0.959588
min	1983.000000	2.999900e+04	1.000000e+00	2.000000
25%	2011.000000	2.549990e+05	3.500000e+04	5.000000
50%	2015.000000	4.500000e+05	6.000000e+04	5.000000
75%	2017.000000	6.750000e+05	9.800000e+04	5.000000
max	2020.000000	1.000000e+07	2.360457e+06	14.000000

```
# Check for missing values
print(car_details.isnull().sum())

# Drop or fill missing values if necessary
car_details.dropna(inplace=True)
```

```
name               0
year               0
selling_price      0
km_driven          0
fuel               0
seller_type        0
transmission       0
owner              0
mileage          221
engine           221
max_power        215
```

```
torque          222
seats           221
dtype: int64
```

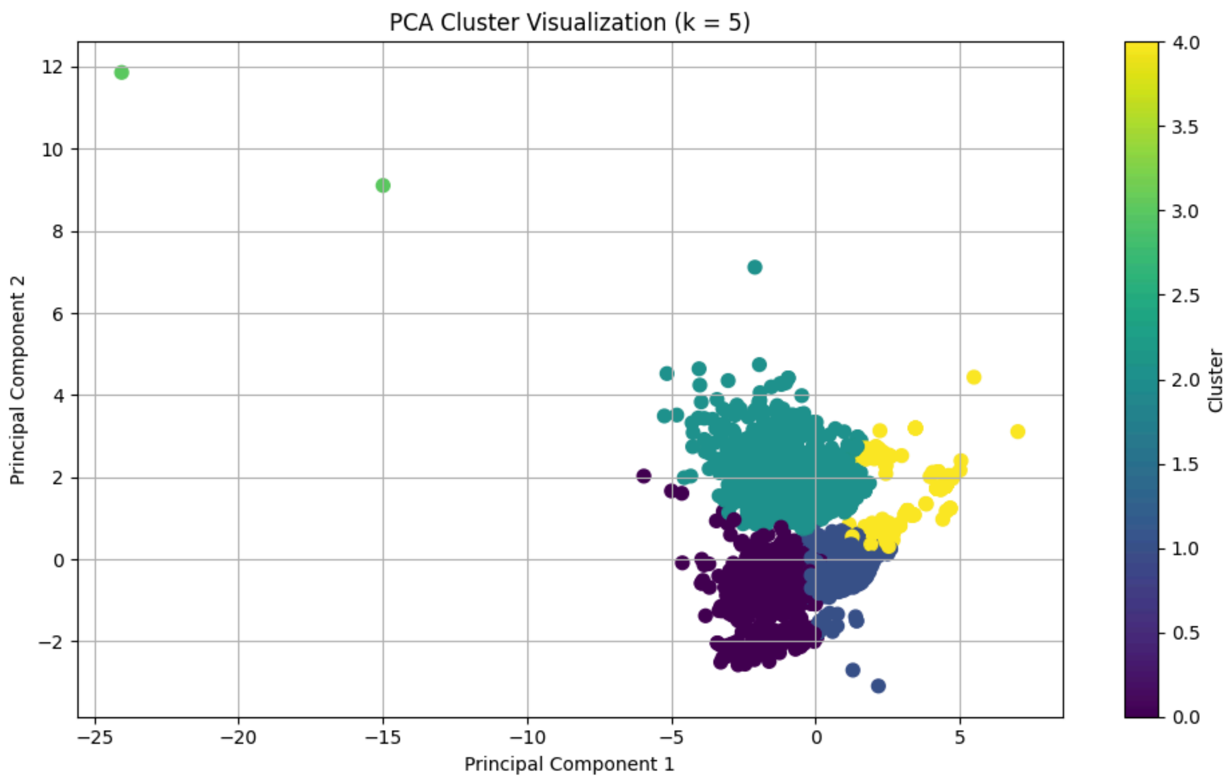
```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Select numeric features
features = ['year', 'selling_price', 'km_driven', 'seats']
car_details_clean = car_details[features].dropna()

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(car_details_clean)

silhouette_scores = []

for k in range(2, 11):  # silhouette_score not defined for k=1
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    score = silhouette_score(X_scaled, kmeans.labels_)
    silhouette_scores.append(score)

# Plot Silhouette Score
plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), silhouette_scores, marker='s', color='purple')
plt.title('Silhouette Score vs Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()
```




```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA



# Select numeric features
features = ['year', 'selling_price', 'km_driven', 'seats']
car_details_clean = car_details[features].dropna()

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(car_details_clean)

# Silhouette scores
```

```
silhouette_scores = []
k_values = list(range(2, 11))

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X_scaled)
    score = silhouette_score(X_scaled, labels)
    silhouette_scores.append(score)

# Get best k
best_k = k_values[silhouette_scores.index(max(silhouette_scores))]
print(f"Best number of clusters: {best_k}")

# Fit final KMeans
kmeans = KMeans(n_clusters=best_k, random_state=42, n_init=10)
labels = kmeans.fit_predict(X_scaled)

# PCA for 2D visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Plot clusters
plt.figure(figsize=(10, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis', s=50)
plt.title(f"PCA Cluster Visualization (k = {best_k})")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label="Cluster")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Best number of clusters: 5
```

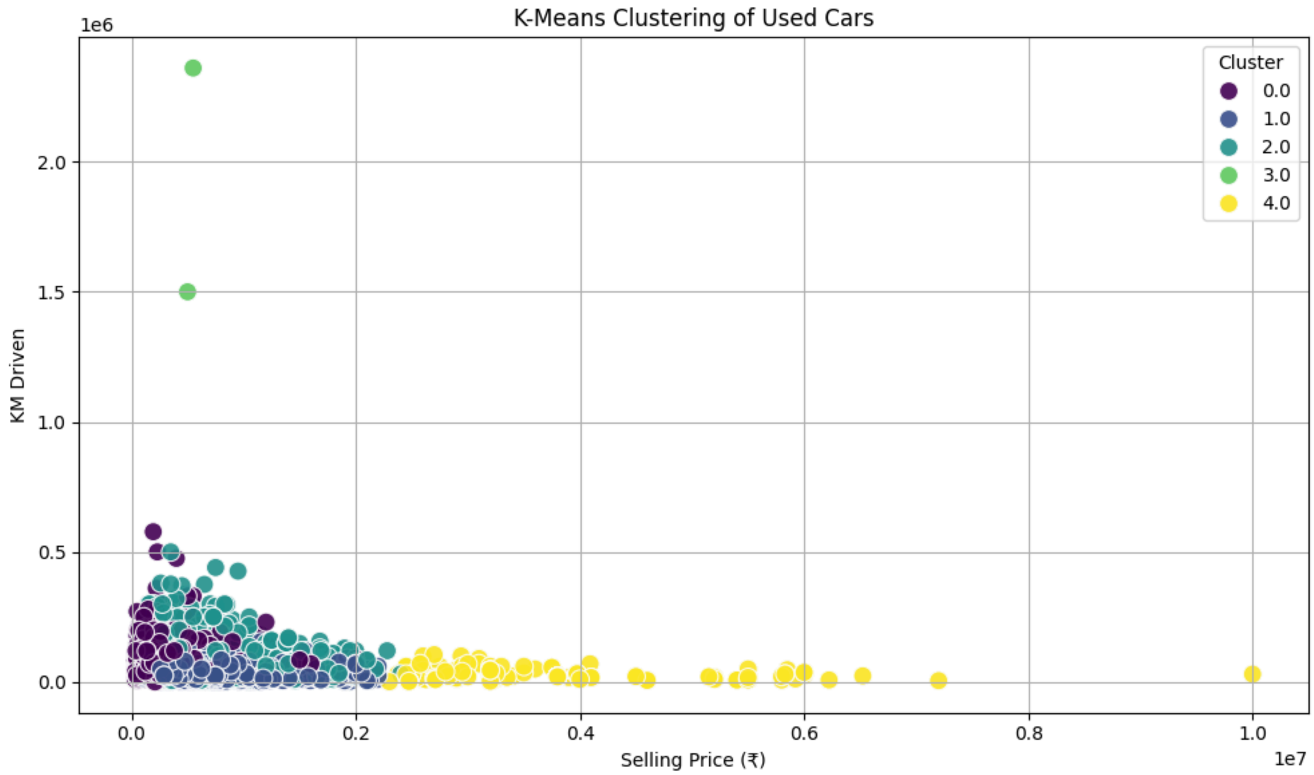



```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Select numeric features
features = ['year', 'selling_price', 'km_driven', 'seats']
car_details_clean = car_details[features].dropna()

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(car_details_clean)
```

```
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
car_details_clean['Cluster'] = kmeans.fit_predict(X_scaled)

# Merge cluster assignments back to the original DataFrame
car_details = car_details.merge(car_details_clean[['Cluster']], left_index=True, right_index=True, how='left')

# Display cluster assignments with basic details
print(car_details[['name', 'fuel', 'seller_type', 'Cluster']].head(10))
```

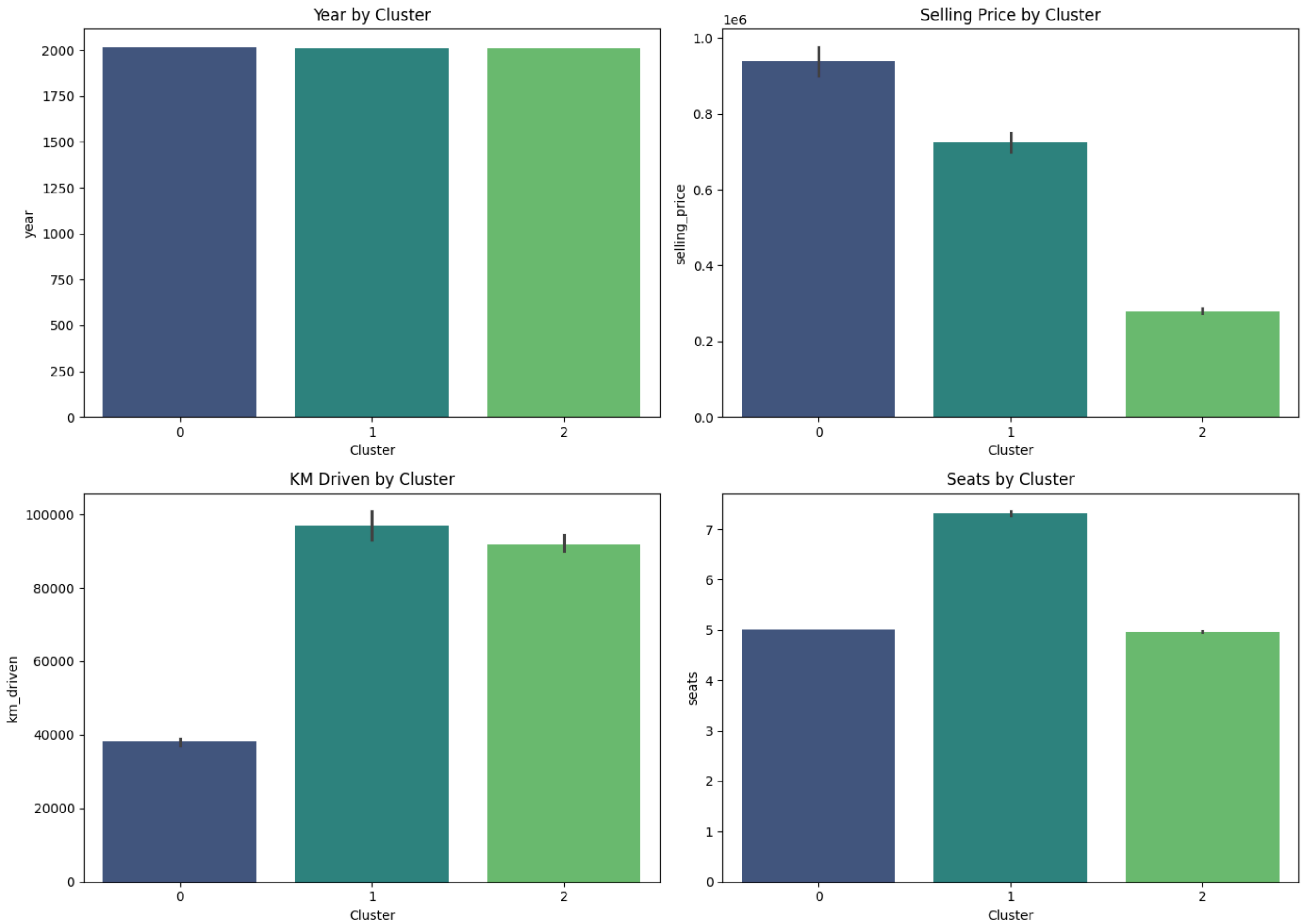
```
                                 name    fuel seller_type  Cluster
0              Maruti Swift Dzire VDI  Diesel  Individual        2
1        Skoda Rapid 1.5 TDI Ambition  Diesel  Individual        2
2            Honda City 2017-2020 EXi  Petrol  Individual        2
3           Hyundai i20 Sportz Diesel  Diesel  Individual        2
4              Maruti Swift VXI BSIII  Petrol  Individual        2
5       Hyundai Xcent 1.2 VTVT E Plus  Petrol  Individual        0
6         Maruti Wagon R LXI DUO BSIII     LPG  Individual        2
7                 Maruti 800 DX BSII  Petrol  Individual        2
8                   Toyota Etios VXD  Diesel  Individual        2
9  Ford Figo Diesel Celebration Edition  Diesel  Individual        2
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Select numeric features
features = ['year', 'selling_price', 'km_driven', 'seats']
car_details_clean = car_details[features].dropna()

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(car_details_clean)

# Assume optimal_k is determined from previous analysis (e.g., silhouette score)
# For this example, I'll use best_k from the previous cell's output (which was 5)
optimal_k = 5
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
car_details_clean['Cluster'] = kmeans.fit_predict(X_scaled)

# Merge cluster assignments back to the original DataFrame
car_details = car_details.merge(car_details_clean[['Cluster']], left_index=True, right_index=True, how='left')

plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=car_details["selling_price"],
    y=car_details["km_driven"],
    hue=car_details["Cluster"],
    palette="viridis",
    s=100,
    alpha=0.9
)

plt.xlabel("Selling Price (₹)")
plt.ylabel("KM Driven")
plt.title("K-Means Clustering of Used Cars")
plt.legend(title="Cluster")
plt.grid(True)
plt.tight_layout()
plt.show()
```



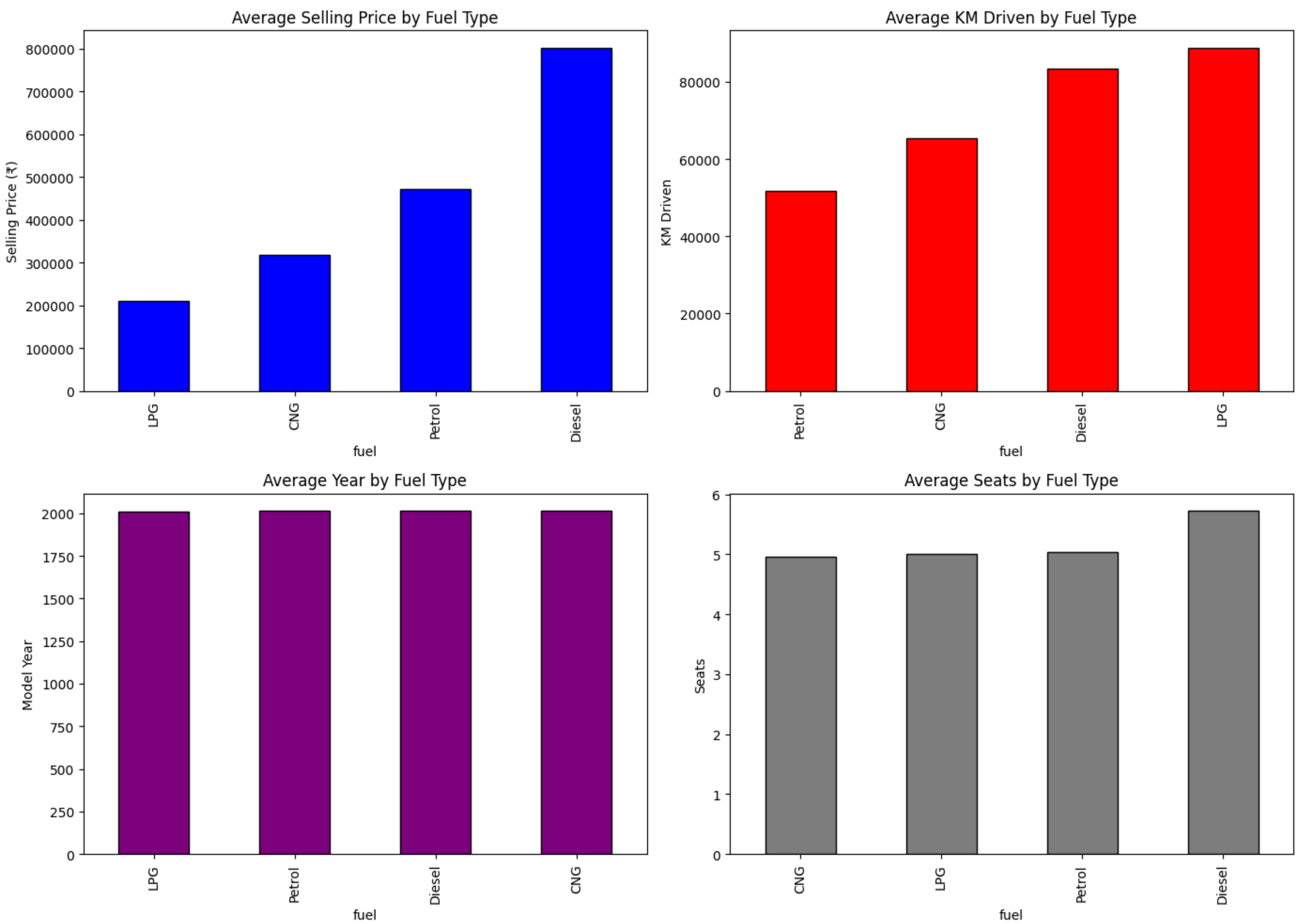

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assume 'df' is your DataFrame with a 'Cluster' column from KMeans
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Year by Cluster
sns.barplot(x=car_details["Cluster"], y=car_details["year"], palette="viridis", ax=axes[0, 0])
axes[0, 0].set_title("Year by Cluster")

# Selling Price by Cluster
sns.barplot(x=car_details["Cluster"], y=car_details["selling_price"], palette="viridis", ax=axes[0, 1])
axes[0, 1].set_title("Selling Price by Cluster")

# KM Driven by Cluster
sns.barplot(x=car_details["Cluster"], y=car_details["km_driven"], palette="viridis", ax=axes[1, 0])
axes[1, 0].set_title("KM Driven by Cluster")

# Seats by Cluster
sns.barplot(x=car_details["Cluster"], y=car_details["seats"], palette="viridis", ax=axes[1, 1])
axes[1, 1].set_title("Seats by Cluster")

# Final layout
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-20-4183022068.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.barplot(x=car_details["Cluster"], y=car_details["year"], palette="viridis", ax=axes[0, 0])
/tmp/ipython-input-20-4183022068.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.barplot(x=car_details["Cluster"], y=car_details["selling_price"], palette="viridis", ax=axes[0, 1])
/tmp/ipython-input-20-4183022068.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.barplot(x=car_details["Cluster"], y=car_details["km_driven"], palette="viridis", ax=axes[1, 0])
/tmp/ipython-input-20-4183022068.py:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.barplot(x=car_details["Cluster"], y=car_details["seats"], palette="viridis", ax=axes[1, 1])
```



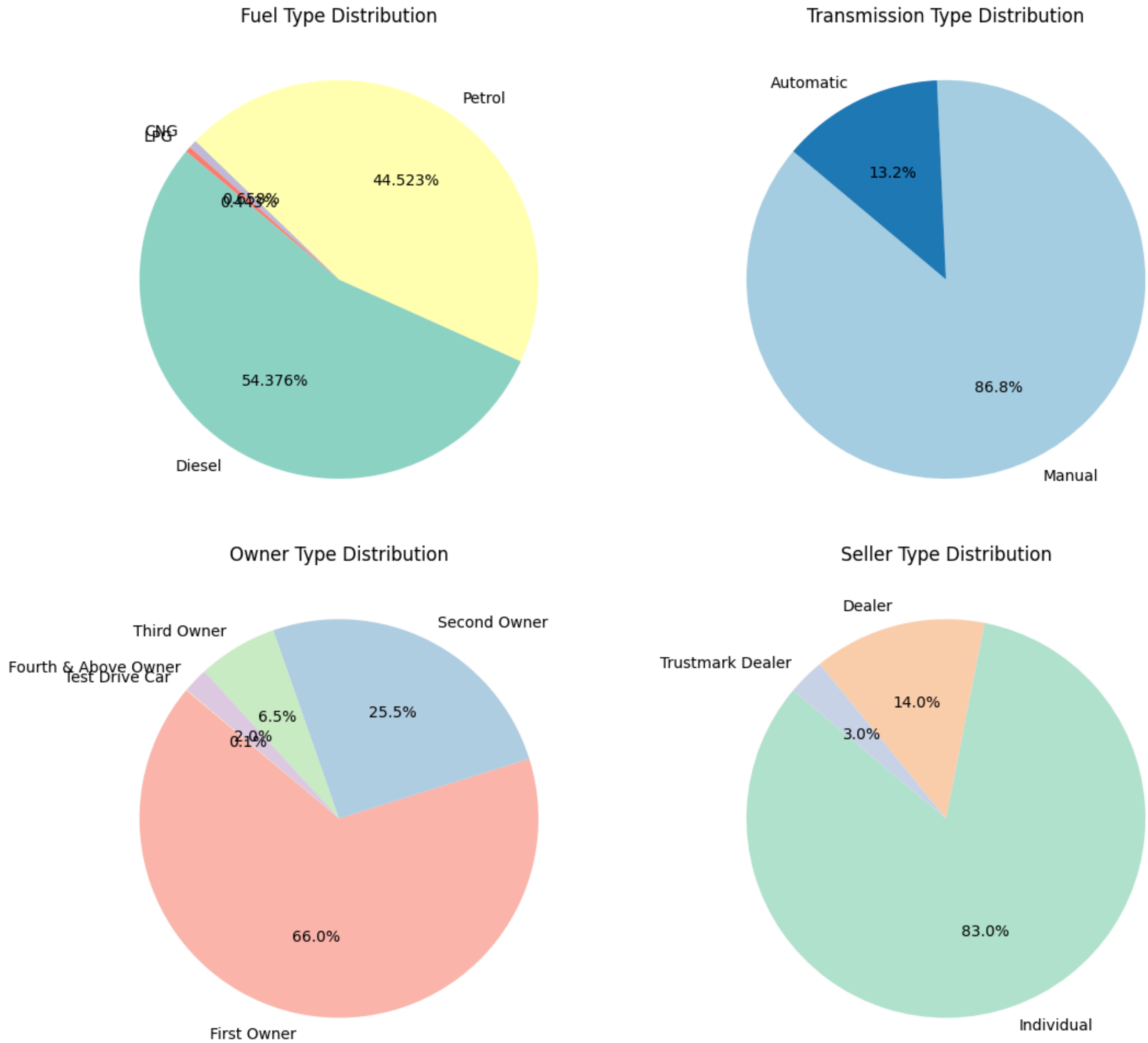

```
# Grouping by fuel type
fuel_grouped = car_details.groupby("fuel")[["selling_price", "km_driven", "year", "seats"]].mean()

# Plot bar charts
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

fuel_grouped["selling_price"].sort_values().plot(kind="bar", ax=axes[0, 0], color="blue", edgecolor="black")
axes[0, 0].set_title("Average Selling Price by Fuel Type")
axes[0, 0].set_ylabel("Selling Price (₹)")

fuel_grouped["km_driven"].sort_values().plot(kind="bar", ax=axes[0, 1], color="red", edgecolor="black")
axes[0, 1].set_title("Average KM Driven by Fuel Type")
axes[0, 1].set_ylabel("KM Driven")
```

```
fuel_grouped["year"].sort_values().plot(kind="bar", ax=axes[1, 0], color="purple", edgecolor="black")
axes[1, 0].set_title("Average Year by Fuel Type")
axes[1, 0].set_ylabel("Model Year")

fuel_grouped["seats"].sort_values().plot(kind="bar", ax=axes[1, 1], color="grey", edgecolor="black")
axes[1, 1].set_title("Average Seats by Fuel Type")
axes[1, 1].set_ylabel("Seats")

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




```
# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# 1. Fuel Type
fuel_counts = car_details['fuel'].value_counts()
axes[0, 0].pie(fuel_counts, labels=fuel_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Set3.colors)
axes[0, 0].set_title('Fuel Type Distribution')

# 2. Transmission Type
transmission_counts = car_details['transmission'].value_counts()
axes[0, 1].pie(transmission_counts, labels=transmission_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
axes[0, 1].set_title('Transmission Type Distribution')

# 3. Owner Type
owner_counts = car_details['owner'].value_counts()
axes[1, 0].pie(owner_counts, labels=owner_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Pastel1.colors)
axes[1, 0].set_title('Owner Type Distribution')

# 4. Seller Type
seller_counts = car_details['seller_type'].value_counts()
axes[1, 1].pie(seller_counts, labels=seller_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Pastel2.colors)
axes[1, 1].set_title('Seller Type Distribution')
```

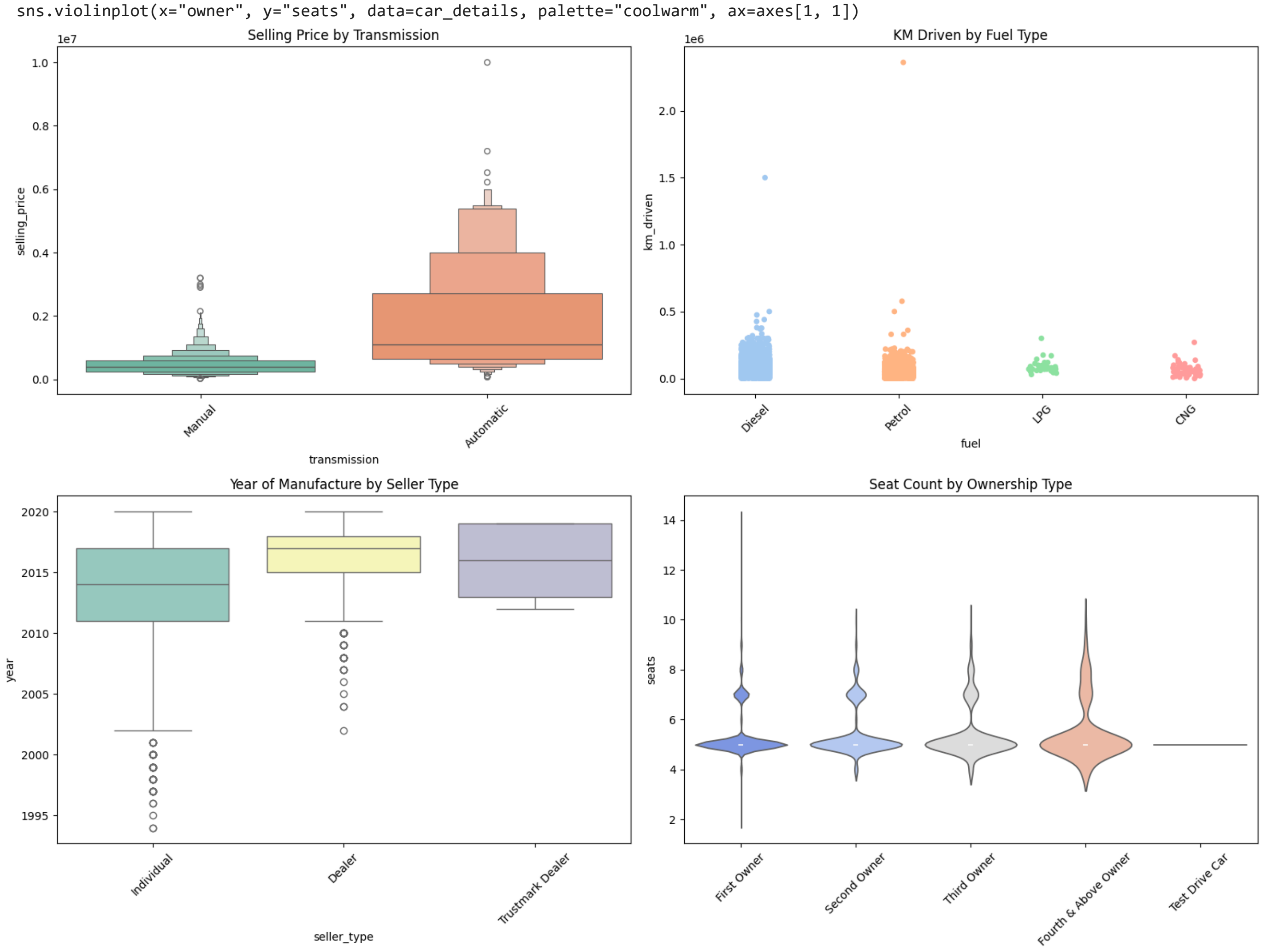
```
# Final layout
plt.tight_layout()
plt.show()
```



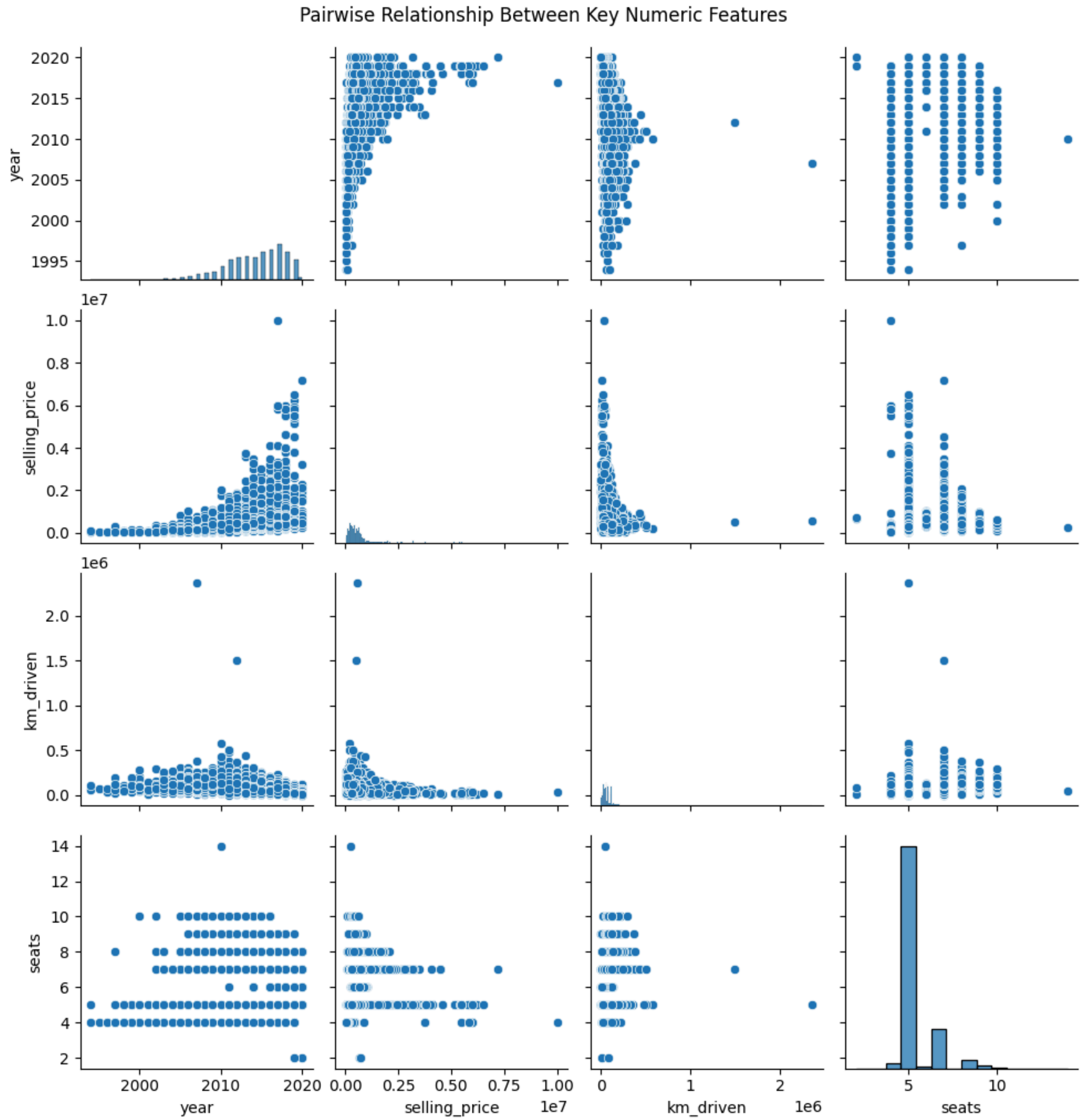

```
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# 1. Selling Price by Transmission (Boxen Plot)
sns.boxenplot(x="transmission", y="selling_price", data=car_details, palette="Set2", ax=axes[0, 0])
axes[0, 0].set_title("Selling Price by Transmission")

# 2. KM Driven by Fuel (Swarm Plot)
sns.stripplot(x="fuel", y="km_driven", data=car_details, jitter=True, palette="pastel", ax=axes[0, 1])
axes[0, 1].set_title("KM Driven by Fuel Type")

# 3. Year of Manufacture by Seller Type (Box Plot)
sns.boxplot(x="seller_type", y="year", data=car_details, palette="Set3", ax=axes[1, 0])
axes[1, 0].set_title("Year of Manufacture by Seller Type")

# 4. Seats by Owner (Violin Plot)
sns.violinplot(x="owner", y="seats", data=car_details, palette="coolwarm", ax=axes[1, 1])
axes[1, 1].set_title("Seat Count by Ownership Type")
```

```
# Adjust layout
for ax in axes.flat:
    ax.tick_params(axis='x', rotation=45)

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

```
/tmp/ipython-input-29-1224858350.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.boxenplot(x="transmission", y="selling_price", data=car_details, palette="Set2", ax=axes[0, 0])
/tmp/ipython-input-29-1224858350.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.stripplot(x="fuel", y="km_driven", data=car_details, jitter=True, palette="pastel", ax=axes[0, 1])
/tmp/ipython-input-29-1224858350.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.boxplot(x="seller_type", y="year", data=car_details, palette="Set3", ax=axes[1, 0])
/tmp/ipython-input-29-1224858350.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.violinplot(x="owner", y="seats", data=car_details, palette="coolwarm", ax=axes[1, 1])
```




```
sns.pairplot(car_details[['year', 'selling_price', 'km_driven', 'seats']])
plt.suptitle("Pairwise Relationship Between Key Numeric Features", y=1.02)
plt.show()
```




```
plt.figure(figsize=(10, 6))
plt.hist(car_details['selling_price'], bins=30, color='blueviolet', edgecolor='black')
plt.title("Distribution of Selling Price")
plt.xlabel("Selling Price (₹)")
plt.ylabel("Number of Cars")
plt.grid(True)
plt.tight_layout()
plt.show()
```


```
plt.figure(figsize=(10, 6))
sns.histplot(data=car_details, x='selling_price', hue='fuel', bins=30, kde=True, multiple='stack')
plt.title("Selling Price Distribution by Fuel Type")
plt.xlabel("Selling Price (₹)")
plt.ylabel("Number of Cars")
plt.grid(True)
plt.tight_layout()
plt.show()
```