Name- Kore Pratiksha Jayant

Oasis Infobyte (Data Science)

Task-5 CAR PRICE PREDICTION WITH MACHINE LEARNING

INTRODUCTION- This project focuses on creating a machine learning model for predicting car prices. We'll analyze factors like brand reputation, car features, horsepower, and mileage to develop an effective prediction system.

Importing necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt
```

Loding Dataset

In [3]:
```
data = pd.read_csv('C:\\Users\\stati\\OneDrive\\Desktop\\Car Dataset.csv')
```

EDA (Exploratory Data Analysis)

In [4]:
```
# To check first few rows of the dataframe
(data.head())
```

Out[4]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmis
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Ma
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Ma
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Ma
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Ma
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Ma

In [5]:
```
# To get information about the dataframe
(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Name       301 non-null    object 
 1   Year           301 non-null    int64  
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Driven_kms     301 non-null    int64  
 5   Fuel_Type      301 non-null    object 
 6   Selling_type   301 non-null    object 
 7   Transmission   301 non-null    object 
 8   Owner          301 non-null    int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

In [6]:
```
# Displaying a random sample of 5 rows
data.sample(5)
```

Out[6]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transm
66	innova	2017	19.75	23.15	11000	Petrol	Dealer	Aut
6	ciaz	2015	6.75	8.12	18796	Petrol	Dealer	I
239	eon	2012	2.00	4.43	23709	Petrol	Dealer	I
141	Bajaj Avenger 150 street	2016	0.60	0.80	20000	Petrol	Individual	I
295	city	2015	8.55	13.09	60076	Diesel	Dealer	I

In [7]:
```
colunas = data['Car_Name'].str.split(' ', n=1, expand=True)
data['Names'] = colunas[0]
```

In [8]:
```
car_df = data.drop('Car_Name',axis=1)
car_df = data[data['Owner'] !=3]
```

In [9]:
```
data.duplicated().sum()
```

Out[9]:
```
2
```

In [10]:
```
# Remove duplicat
new_df = data.drop_duplicates()
```

In [11]:
```
new_df.shape
```

Out[11]:
```
(299, 10)
```

In [12]:
```
new_df.columns
```

Out[12]:
```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Driven_kms',
       'Fuel_Type', 'Selling_type', 'Transmission', 'Owner', 'Names'],
      dtype='object')
```

In [13]:
```
# Check the number of unique values of each column
new_df.nunique()
```

Out[13]:
```
Car_Name          98
Year              16
Selling_Price    156
Present_Price    148
Driven_kms       206
Fuel_Type          3
Selling_type       2
Transmission       2
Owner              3
Names             44
dtype: int64
```

In [14]:
```
# Checking the distribution of categorical data
categorical_columns = ['Fuel_Type', 'Selling_type', 'Transmission', 'Year', 'Pre

for column in categorical_columns:
    print(new_df[column].value_counts())
```

```
Fuel_Type
Petrol    239
Diesel     58
CNG         2
Name: count, dtype: int64
Selling_type
Dealer        193
Individual    106
Name: count, dtype: int64
Transmission
Manual       260
Automatic     39
Name: count, dtype: int64
Year
2015    60
2016    49
2014    38
2017    35
2013    33
2012    23
2011    19
2010    15
2008     7
2009     6
2006     4
2005     4
2003     2
2007     2
2018     1
2004     1
Name: count, dtype: int64
Present_Price
9.40     14
13.60    13
5.70      8
1.47      7
0.51      6
         ..
36.23     1
18.54     1
7.27      1
15.04     1
12.50     1
Name: count, Length: 148, dtype: int64
Owner
0    288
1     10
3      1
Name: count, dtype: int64
```

In [15]:
```
# Descriptive statistics
new_df.describe()
```

Out[15]:

	Year	Selling_Price	Present_Price	Driven_kms	Owner
count	299.000000	299.000000	299.000000	299.000000	299.000000
mean	2013.615385	4.589632	7.541037	36916.752508	0.043478
std	2.896868	4.984240	8.566332	39015.170352	0.248720
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.850000	1.200000	15000.000000	0.000000
50%	2014.000000	3.510000	6.100000	32000.000000	0.000000
75%	2016.000000	6.000000	9.840000	48883.500000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

In [16]:
```
# List of categorical columns
categorical_features = ['Fuel_Type', 'Selling_type', 'Transmission']
```

In [17]:
```
numerical_features = ['Year', 'Selling_Price', 'Present_Price', 'Driven_kms', 'C
```

In [18]:
```
# Display unique categories for each categorical column
for column in categorical_features:
    unique_categories = new_df[column].unique()
    print(f"{column} categories: {unique_categories}")
```

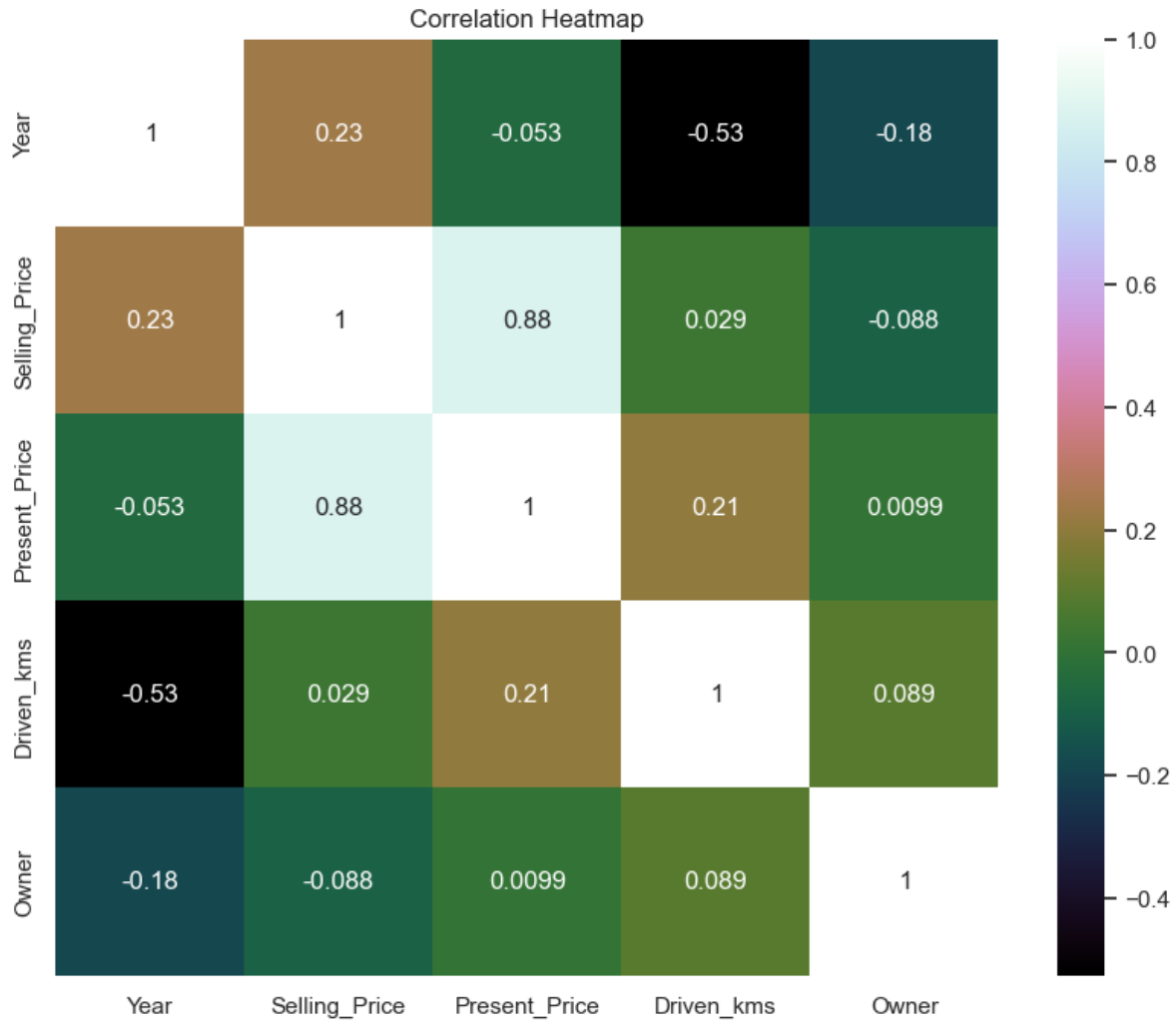
```
Fuel_Type categories: ['Petrol' 'Diesel' 'CNG']
Selling_type categories: ['Dealer' 'Individual']
Transmission categories: ['Manual' 'Automatic']
```

Visualization

In [19]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Create a heatmap of the correlation matrix

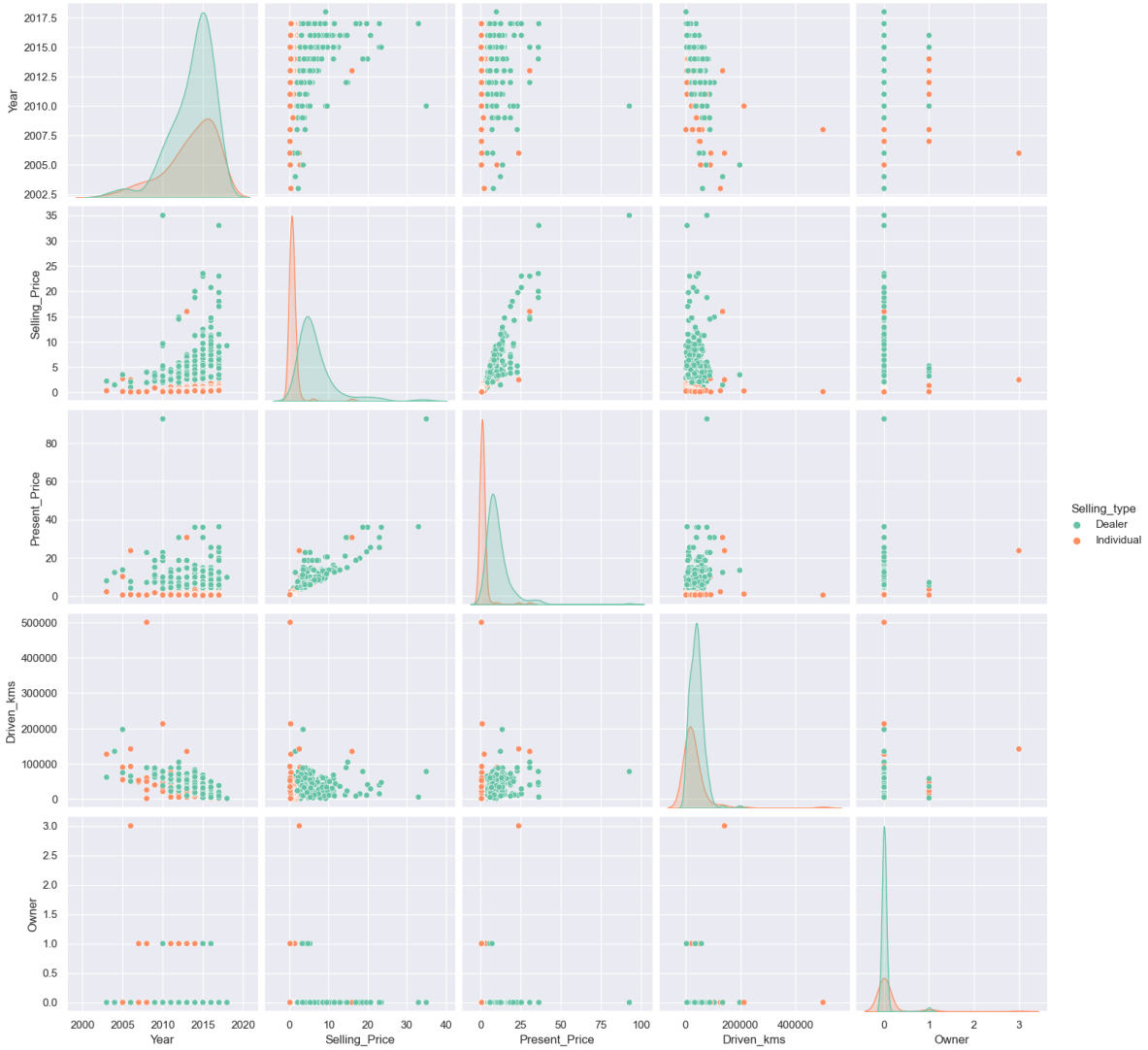
In [20]:
```
correlation_matrix = new_df[numerical_features].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='cubehelix')
plt.title('Correlation Heatmap')
plt.show()
```



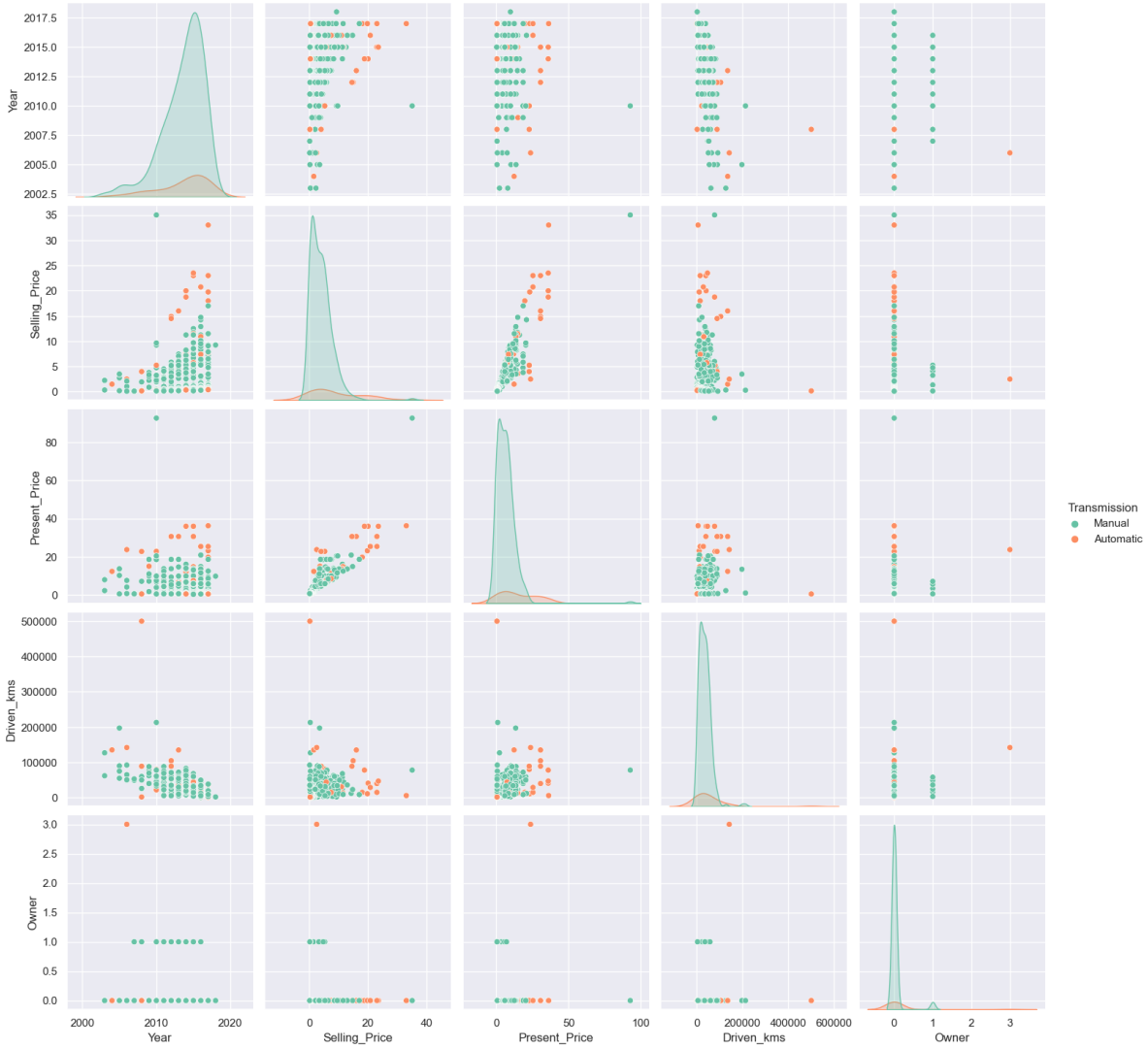
Pair Plot

In [21]:
```
sns.pairplot(new_df, hue='Fuel_Type', palette="Set2", height=3)
plt.show()
```

In [22]:
```
sns.pairplot(new_df, hue='Selling_type', palette="Set2", height=3)
plt.show()
```



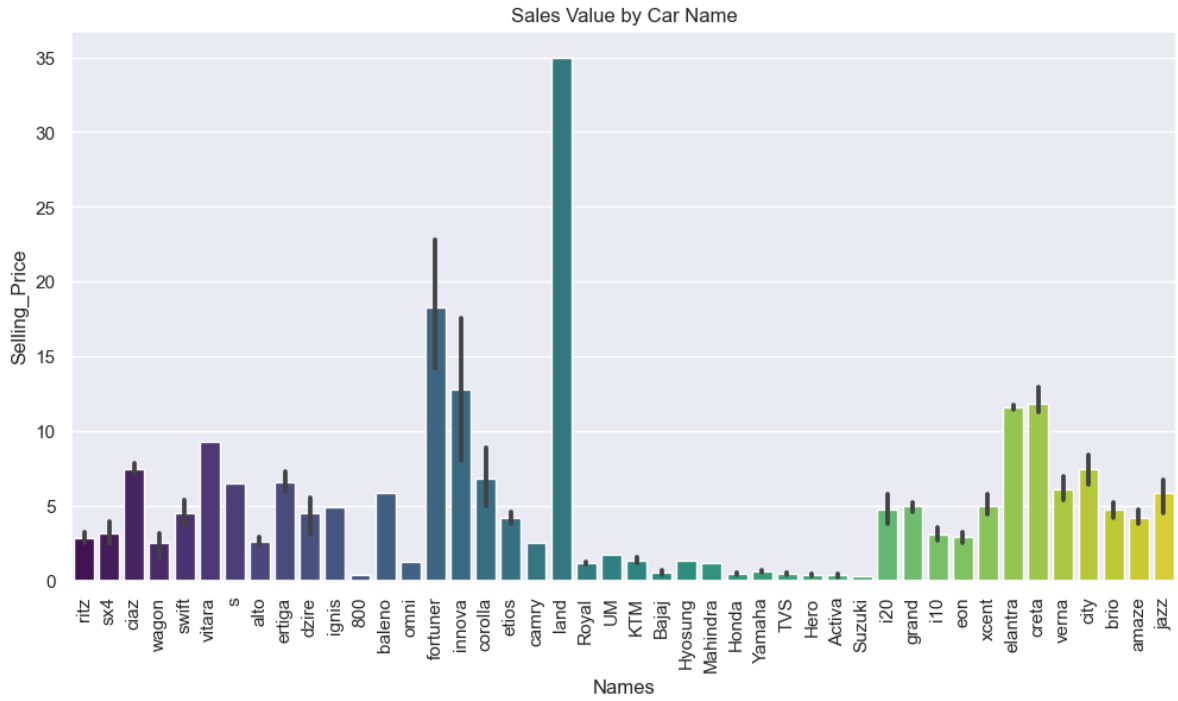
In [23]:
```
sns.pairplot(new_df, hue='Transmission', palette="Set2", height=3)
plt.show()
```



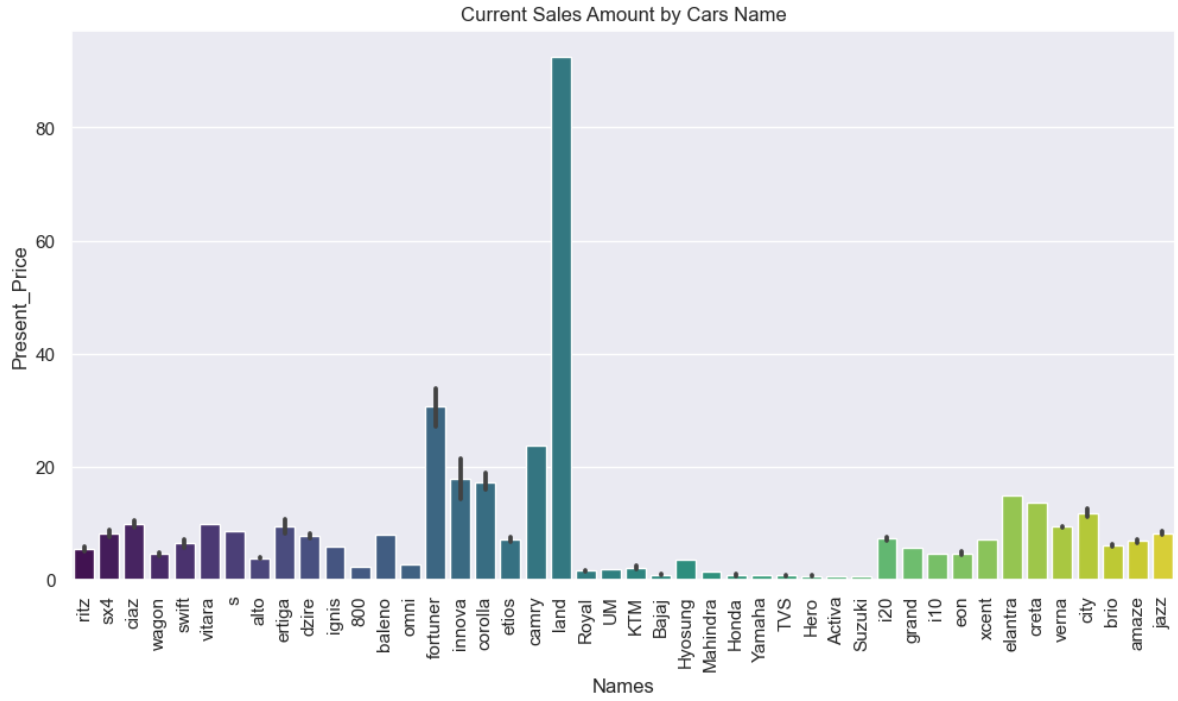
Bar Plot

In [24]:
```
def plot_bar(x, y, title, size=(12, 6)):
    sns.barplot(x=x, y=y, data=new_df, palette="viridis").set(title=title, xlabe
    plt.xticks(rotation=90)
    plt.gcf().set_size_inches(size)
    plt.show()
```

In [25]:
```
plot_bar('Names', 'Selling_Price', 'Sales Value by Car Name')
```

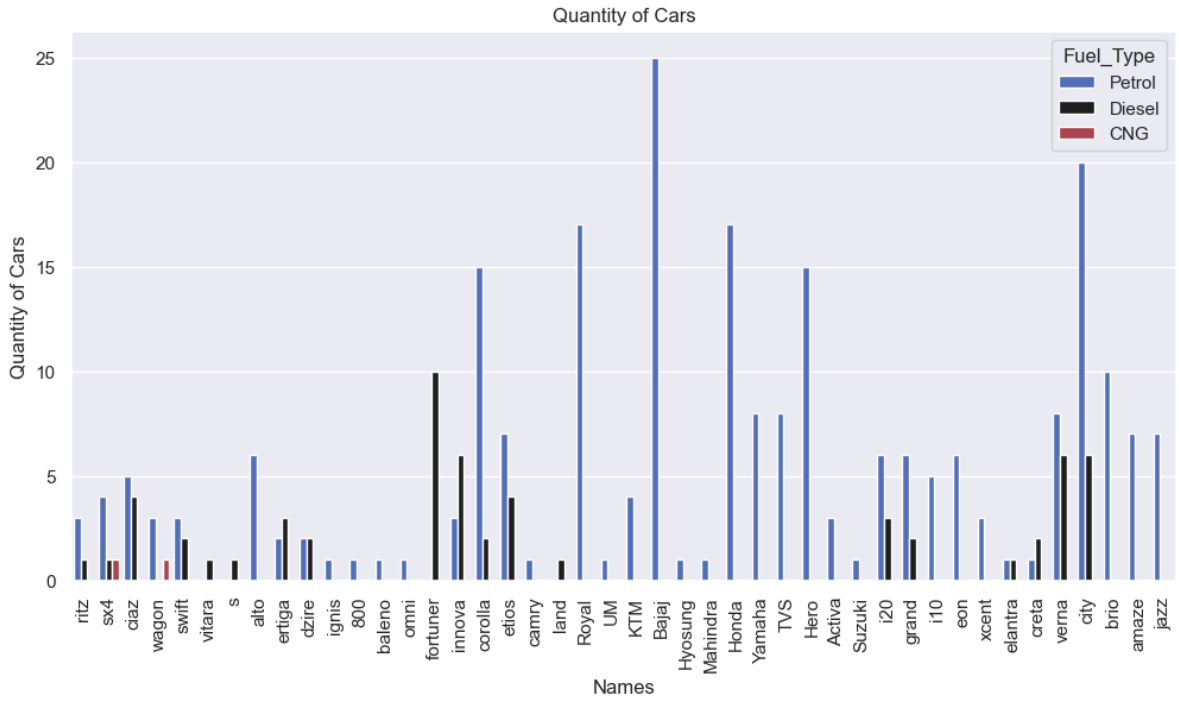


In [26]:
```
plot_bar('Names', 'Present_Price', 'Current Sales Amount by Cars Name')
```

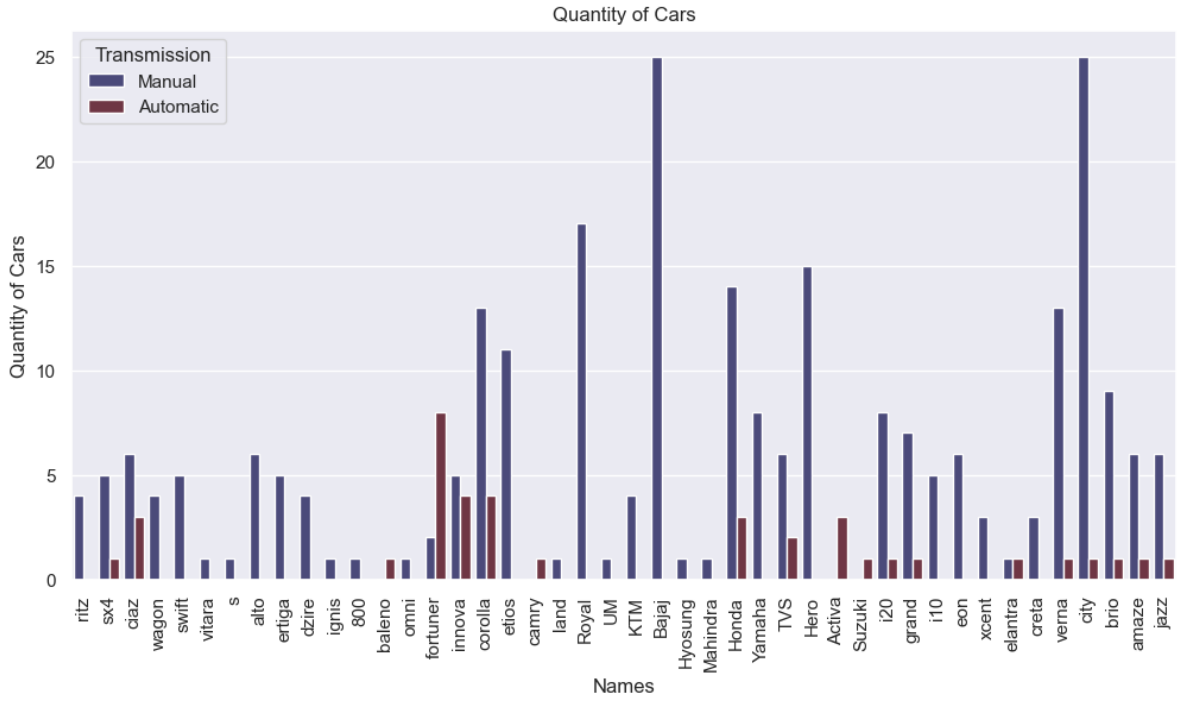


In [27]:
```
def plot_count(x, hue, title):
    plt.figure(figsize=(12, 6))
    sns.countplot(x=x, hue=hue, data=new_df, palette='icefire')
    plt.title(title)
    plt.xticks(rotation=90)
    plt.xlabel(x)
    plt.ylabel('Quantity of Cars')
    plt.legend(title=hue)
    plt.show()
```

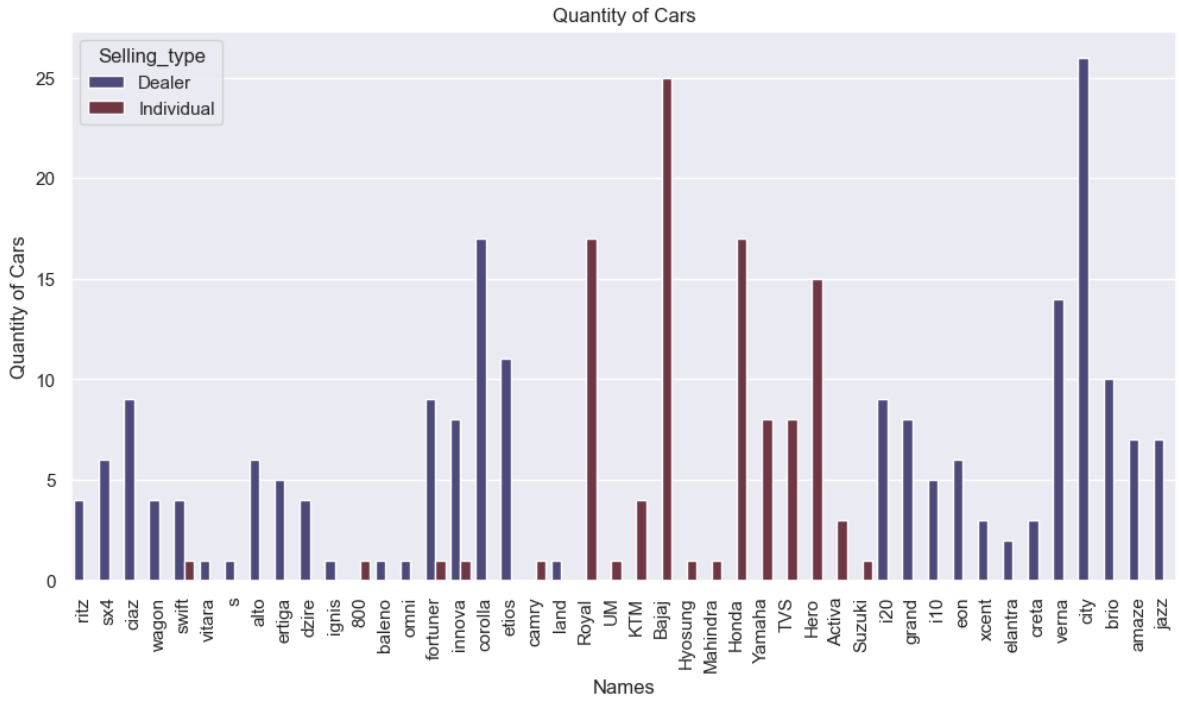
In [28]:
```
plot_count('Names','Fuel_Type','Quantity of Cars')
```



In [29]:
```
plot_count('Names','Transmission','Quantity of Cars')
```

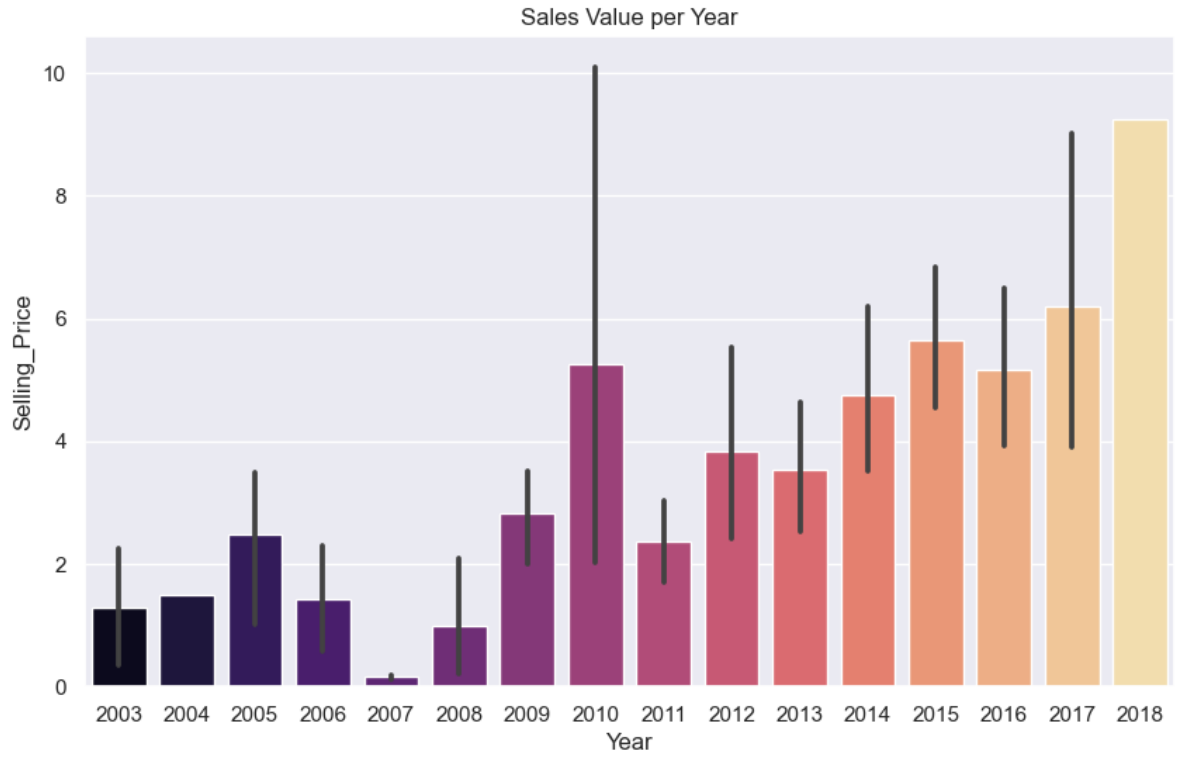


In [30]:
```
plot_count('Names','Selling_type','Quantity of Cars')
```

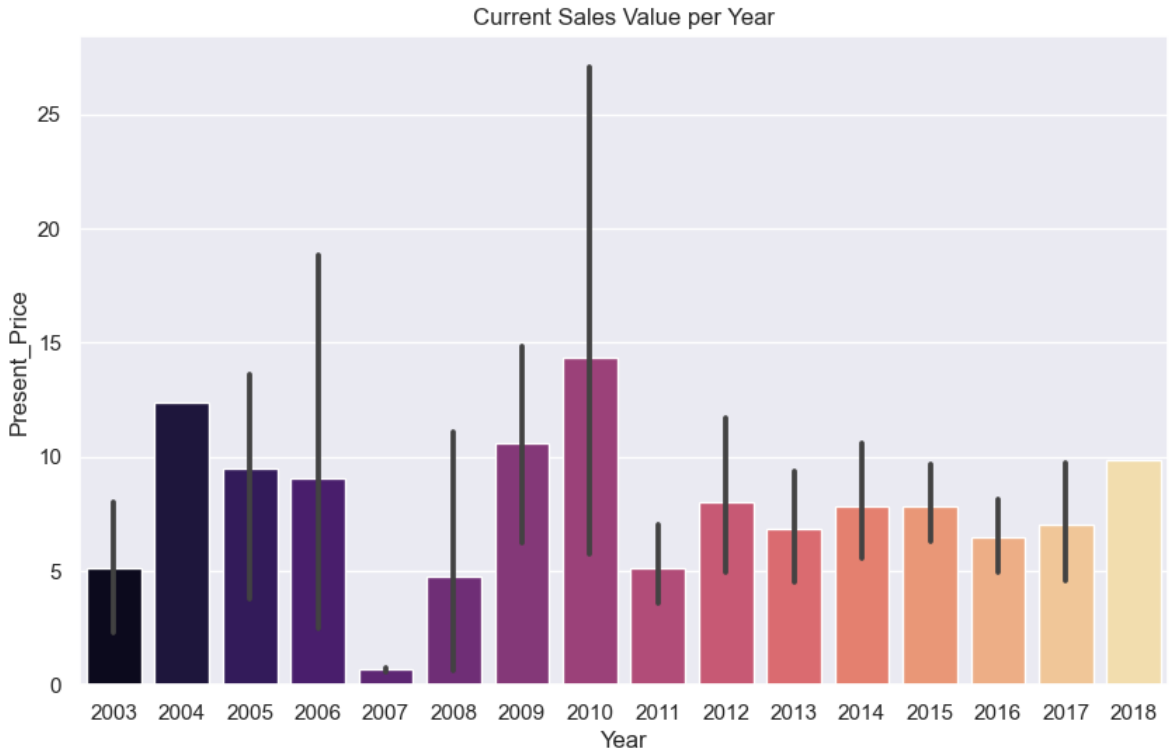


In [31]:
```
def plot_bar(x, y, title, size=(10, 6), palette='magma'):
    sns.barplot(x=x, y=y, data=new_df, palette=palette).set(title=title, xlabel=
    plt.gcf().set_size_inches(size)
    plt.show()
```

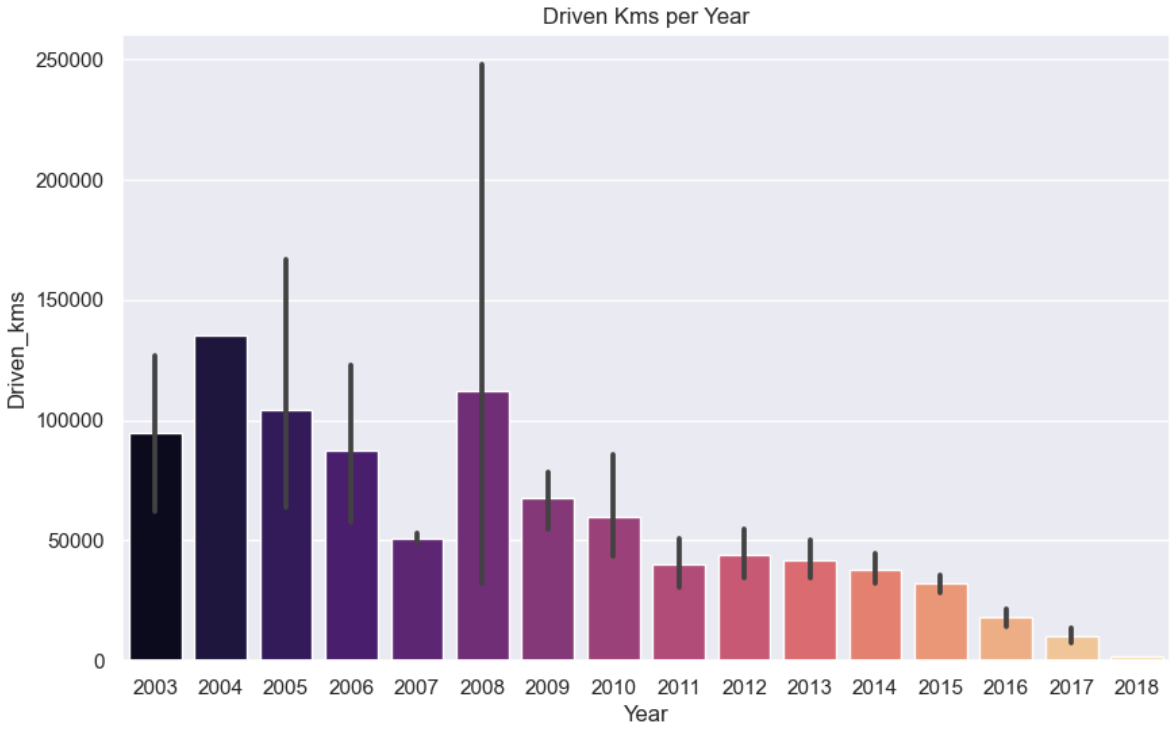
In [32]:
```
plot_bar('Year', 'Selling_Price', 'Sales Value per Year')
```



In [33]:
```
plot_bar('Year', 'Present_Price', 'Current Sales Value per Year')
```



In [34]:
```
plot_bar('Year', 'Driven_kms', 'Driven Kms per Year')
```



In [56]:
```
**Scatter Plot**
```

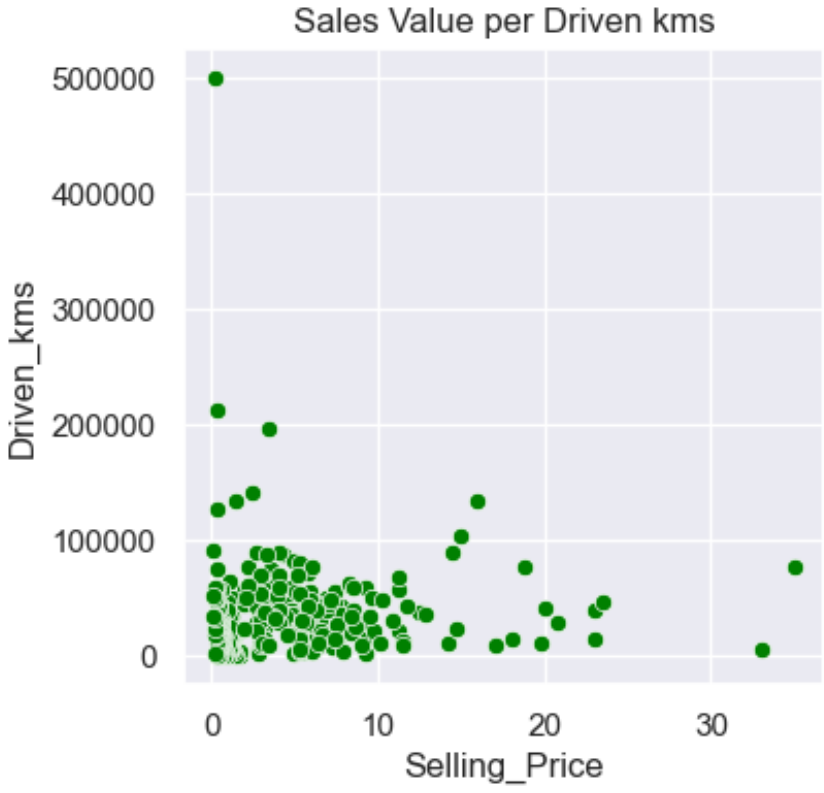
```
  Cell In[56], line 1
    **Scatter Plot**
    ^
SyntaxError: invalid syntax
```

In [35]:
```
def plot_scatter(x, y, title, size=4
                 , color='green'):
    plt.figure(figsize=(size, size))
    sns.scatterplot(x=x, y=y,
                    data=new_df, color=color).set(
```

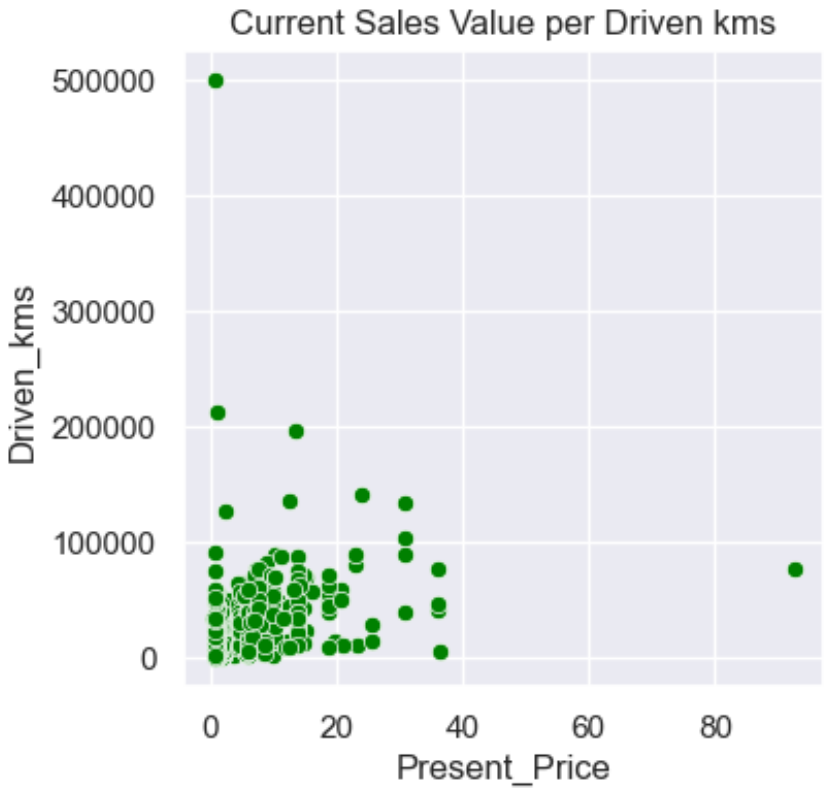
```
        title=title, xlabel=x, ylabel=y)
    plt.show()
```

In [36]:
```
# Rewritten scatter plots with specified size and palette
plot_scatter('Present_Price', 'Selling_Price', 'Sales Value vs Current Value')
```

In [37]:
```
plot_scatter('Selling_Price', 'Driven_kms', 'Sales Value per Driven kms')
```

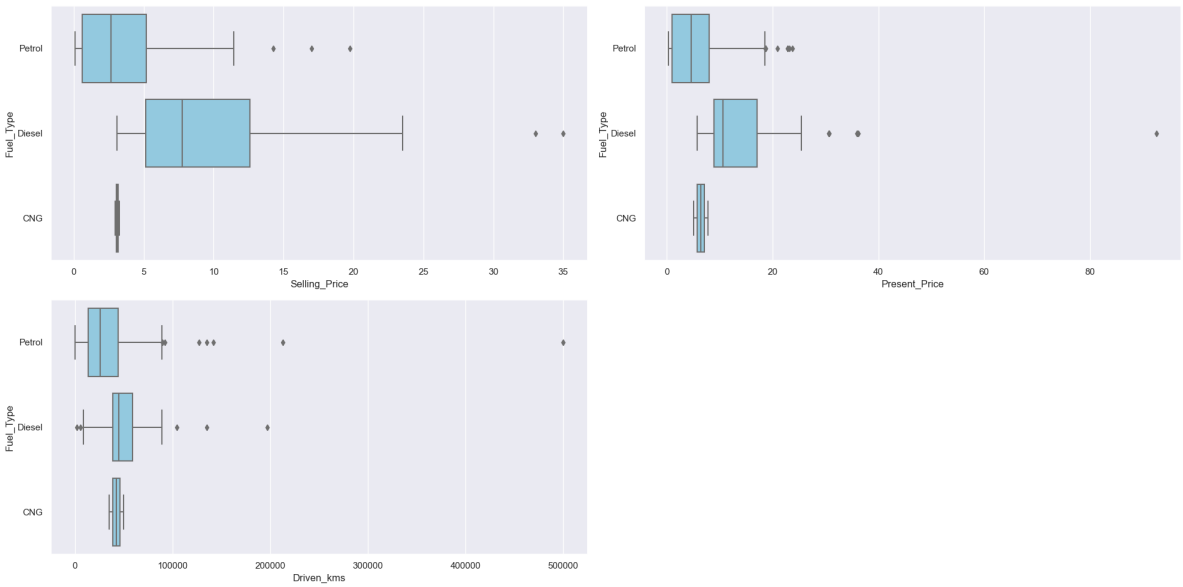


In [38]:
```
plot_scatter('Present_Price', 'Driven_kms', 'Current Sales Value per Driven kms'
```



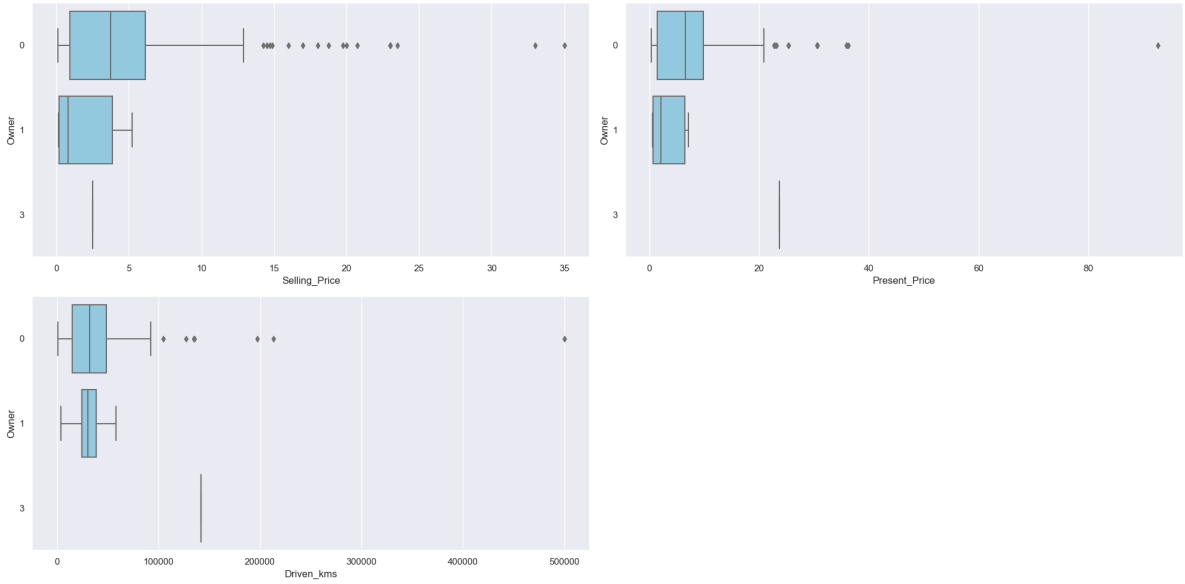
Box Plot

In [39]:
```
plt.figure(figsize=(20, 10))

for i, col in enumerate(['Selling_Price', 'Present_Price', 'Driven_kms']):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(data=data, y='Fuel_Type', x=col, orient='h', color='skyblue')

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

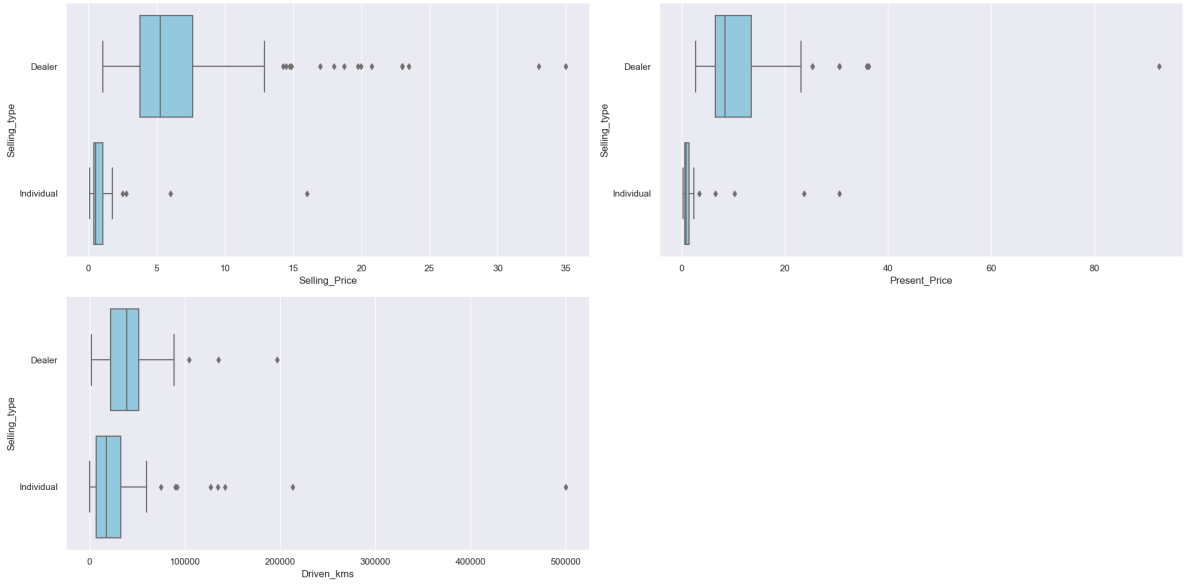


In [40]:
```
plt.figure(figsize=(20, 10))

for i, col in enumerate(['Selling_Price', 'Present_Price', 'Driven_kms']):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(data=data, y='Owner', x=col, orient='h', color='skyblue')
```

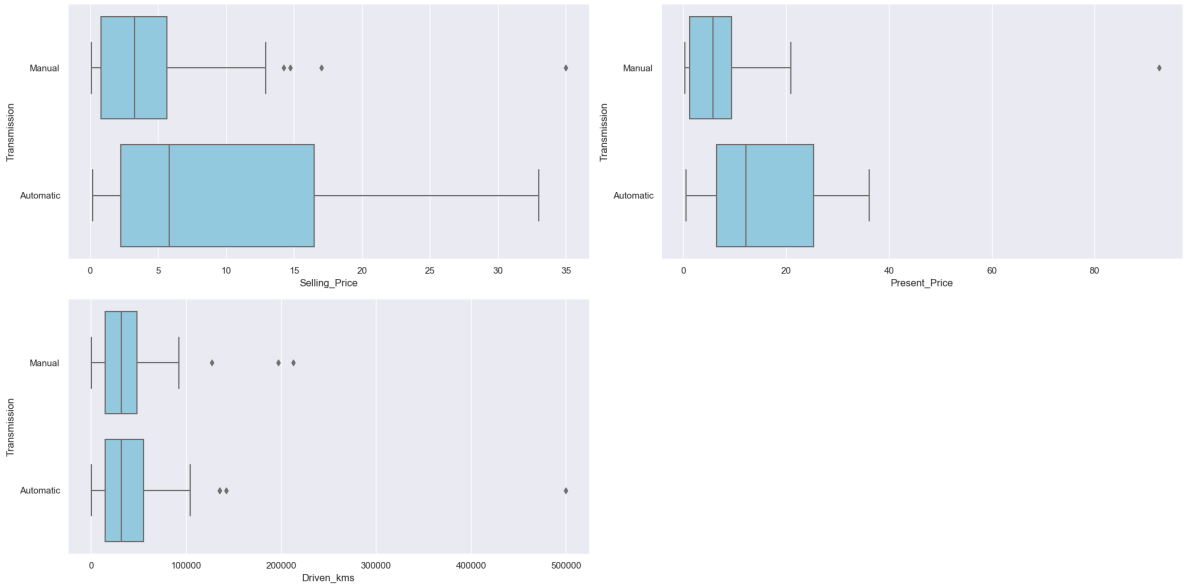
```
plt.tight_layout()
plt.show()
```



In [41]:
```
plt.figure(figsize=(20, 10))

for i, col in enumerate(['Selling_Price', 'Present_Price', 'Driven_kms']):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(data=data, y='Selling_type', x=col, orient='h', color='skyblue')

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



In [42]:
```
plt.figure(figsize=(20, 10))

for i, col in enumerate(['Selling_Price', 'Present_Price', 'Driven_kms']):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(data=data, y='Transmission', x=col, orient='h', color='skyblue')

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



In [43]:
```
def remove_outliers(col):
    q25, q75 = new_df[col].quantile([0.25, 0.75])
    iqr = q75 - q25
    upper_limit, lower_limit = q75 + 1.5 * iqr, q25 - 1.5 * iqr
    return new_df[(new_df[col] >= lower_limit) & (new_df[col] <= upper_limit)]

# Remove outliers for each specified column
data = remove_outliers('Selling_Price')
data = remove_outliers('Present_Price')
data = remove_outliers('Driven_kms')
```

In [44]:
```
from sklearn.preprocessing import LabelEncoder

# Define categorical columns for encoding
categorical_columns = ['Fuel_Type', 'Selling_type', 'Transmission', 'Car_Name']

# Create dictionaries to store label mappings
label_mapping = {}

# Apply label encoding and store mappings
for col in categorical_columns:
    encoder = LabelEncoder()
    data[col] = encoder.fit_transform(data[col])
    label_mapping[col] = dict(enumerate(encoder.classes_))

data.head()
```

Out[44]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmis
0	87	2014	3.35	5.59	27000	2	0	
1	90	2013	4.75	9.54	43000	1	0	
2	66	2017	7.25	9.85	6900	2	0	
3	93	2011	2.85	4.15	5200	2	0	
4	89	2014	4.60	6.87	42450	1	0	

In [45]:
```
X = data.drop('Selling_Price', axis=1).values
Y = data['Selling_Price'].values
```

In [46]:
```
print(X.shape)
print(type(X))
```

```
(291, 9)
<class 'numpy.ndarray'>
```

In [47]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Categorical column indices
categorical_cols = [5, 6, 7, 8]  # Adjust these indices based on your data

# Create a column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(categories='auto', sparse=False, drop='first'), ca
    ],
    remainder='passthrough'
)

# Fit and transform the data
X_encoded = preprocessor.fit_transform(X)
```

In [48]:
```
#Splitting the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_encoded, Y, test_size=0.2,
```

In [49]:
```
#Train a Regression Model
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
```

Out[49]:
```
▾ LinearRegression
LinearRegression()
```

In [50]:
```
y_pred_linear = linear_model.predict(X_test)
```

In [51]:
```
#Evaluating the Regression Model
from sklearn.metrics import mean_squared_error
from math import sqrt
mse_linear = mean_squared_error(y_test, y_pred_linear)
rmse_linear = sqrt(mse_linear)
print(f'Linear Regression RMSE: {rmse_linear}')
```

```
Linear Regression RMSE: 3.1461375715530244
```

In [52]:
```
#Train a Random Forest Model
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
```

Out[52]:
```
▾ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In [53]:
```
y_pred_rf = rf_model.predict(X_test)
```

In [54]:
```
#Evaluating the Random Forest Model
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = sqrt(mse_rf)
print(f'Random Forest RMSE: {rmse_rf}')
```

```
Random Forest RMSE: 3.3337385297763307
```

In [55]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred_rf)
plt.xlabel('Actual Selling Price')
plt.ylabel('Predicted Selling Price (Random Forest)')
plt.title('Actual vs. Predicted Selling Price (Random Forest)')
plt.show()
```

As wev can see from above scatter plot that datapoints are close to eachother we can say that our model works well.