In [1]:
```
#Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")
```

Import Dataset

In [3]:
```
df = pd.read_csv(r"C:\Users\user\Downloads\car data.csv")
df
```

Out[3]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Tr
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	
...	...	...	...	...	...	...	...	
296	city	2016	9.50	11.60	33988	Diesel	Dealer	
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	
298	city	2009	3.35	11.00	87934	Petrol	Dealer	
299	city	2017	11.50	12.50	9000	Diesel	Dealer	
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	

301 rows × 9 columns

Exploratory Data Analysis

In [5]:
```
df.head()
```

Out[5]:

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

In [6]:
```
df.tail()
```

Out[6]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Tr
296	city	2016	9.50	11.6	33988	Diesel	Dealer	
297	brio	2015	4.00	5.9	60000	Petrol	Dealer	
298	city	2009	3.35	11.0	87934	Petrol	Dealer	
299	city	2017	11.50	12.5	9000	Diesel	Dealer	
300	brio	2016	5.30	5.9	5464	Petrol	Dealer	

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

Out[7]:
```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Driven_kms',
       'Fuel_Type', 'Selling_type', 'Transmission', 'Owner'],
      dtype='object')
```

In [8]:
```
df.shape
```

Out[8]:
```
(301, 9)
```

In [9]:
```
df.describe()
```

Out[9]:

	Year	Selling_Price	Present_Price	Driven_kms	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.642584	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

In [10]:
```
df.describe().style.format(precision=2).background_gradient(cmap='RdBu')
```

Out[10]:

	Year	Selling_Price	Present_Price	Driven_kms	Owner
count	301.00	301.00	301.00	301.00	301.00
mean	2013.63	4.66	7.63	36947.21	0.04
std	2.89	5.08	8.64	38886.88	0.25
min	2003.00	0.10	0.32	500.00	0.00
25%	2012.00	0.90	1.20	15000.00	0.00
50%	2014.00	3.60	6.40	32000.00	0.00
75%	2016.00	6.00	9.90	48767.00	0.00
max	2018.00	35.00	92.60	500000.00	3.00

In [11]:
```
df.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 [12]:
```
df.isnull().sum()
```

Out[12]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Driven_kms       0
Fuel_Type        0
Selling_type     0
Transmission     0
Owner            0
dtype: int64
```

In [13]:
```
df.dtypes
```

Out[13]:
```
Car_Name          object
Year               int64
Selling_Price    float64
Present_Price    float64
Driven_kms         int64
Fuel_Type         object
Selling_type      object
Transmission      object
Owner              int64
dtype: object
```

In [14]:
```
df.duplicated().sum()
```

Out[14]:
```
2
```

In [15]:
```
df = df.drop_duplicates()
df.duplicated().sum()
```

Out[15]:
```
0
```

Data visualization and EDA

In [17]:
```
df['Owner'].value_counts()
```

Out[17]:
```
Owner
0    288
1     10
3      1
Name: count, dtype: int64
```

From above we can observed that a single car owned by how many members and we found that there is 10 snigle owned cars,1 car is owned by 3 owners and 288 cars have no owner

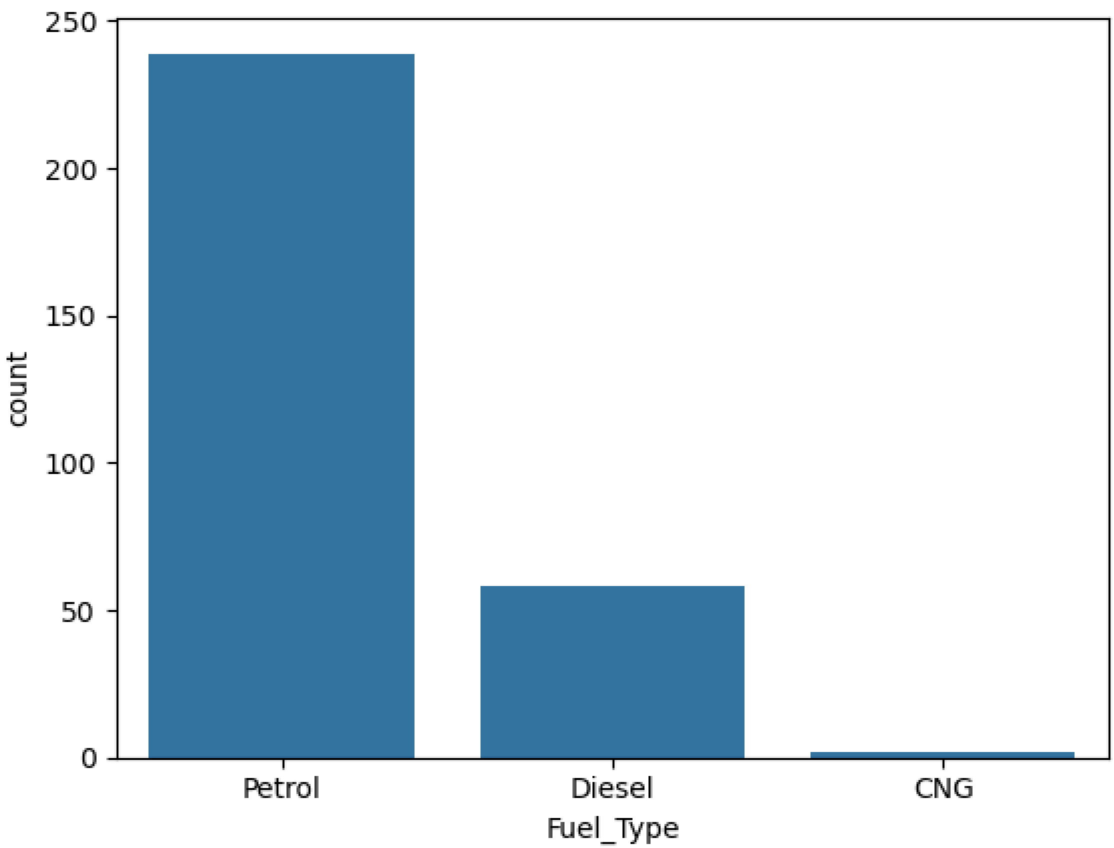
In [19]:
```
#Exploring Categorical Features
df['Car_Name'].value_counts()
```

Out[19]:
```
Car_Name
city                         26
corolla altis                16
verna                        14
brio                         10
fortuner                     10
                             ..
Honda CB Trigger              1
Yamaha FZ S                   1
Bajaj Pulsar 135 LS           1
Activa 4g                     1
Bajaj Avenger Street 220      1
Name: count, Length: 98, dtype: int64
```

In [20]:
```
df['Fuel_Type'].value_counts()
```

Out[20]:
```
Fuel_Type
Petrol    239
Diesel     58
CNG         2
Name: count, dtype: int64
```

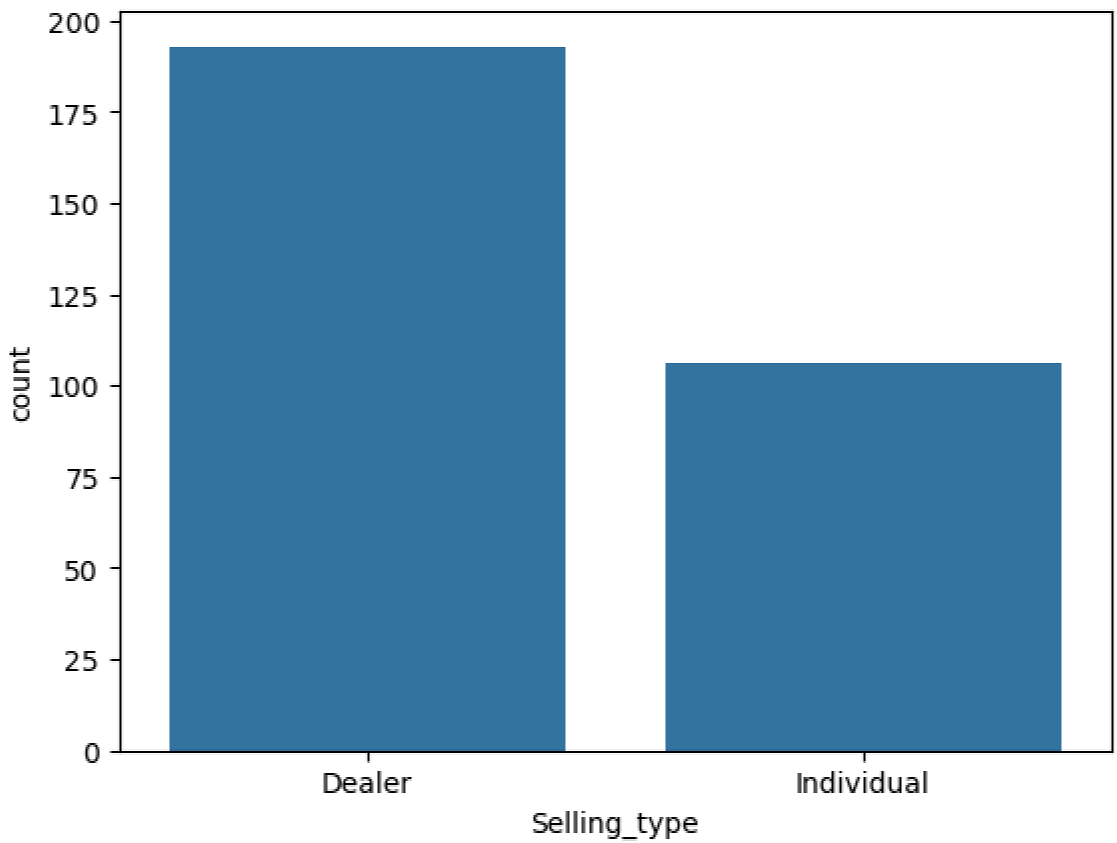
In [21]:
```
sns.countplot(x='Fuel_Type', data=df)
plt.show()
```



In [70]:
```
df['Selling_type'].value_counts()
```

Out[70]:
```
Selling_type
Dealer        193
Individual    106
Name: count, dtype: int64
```

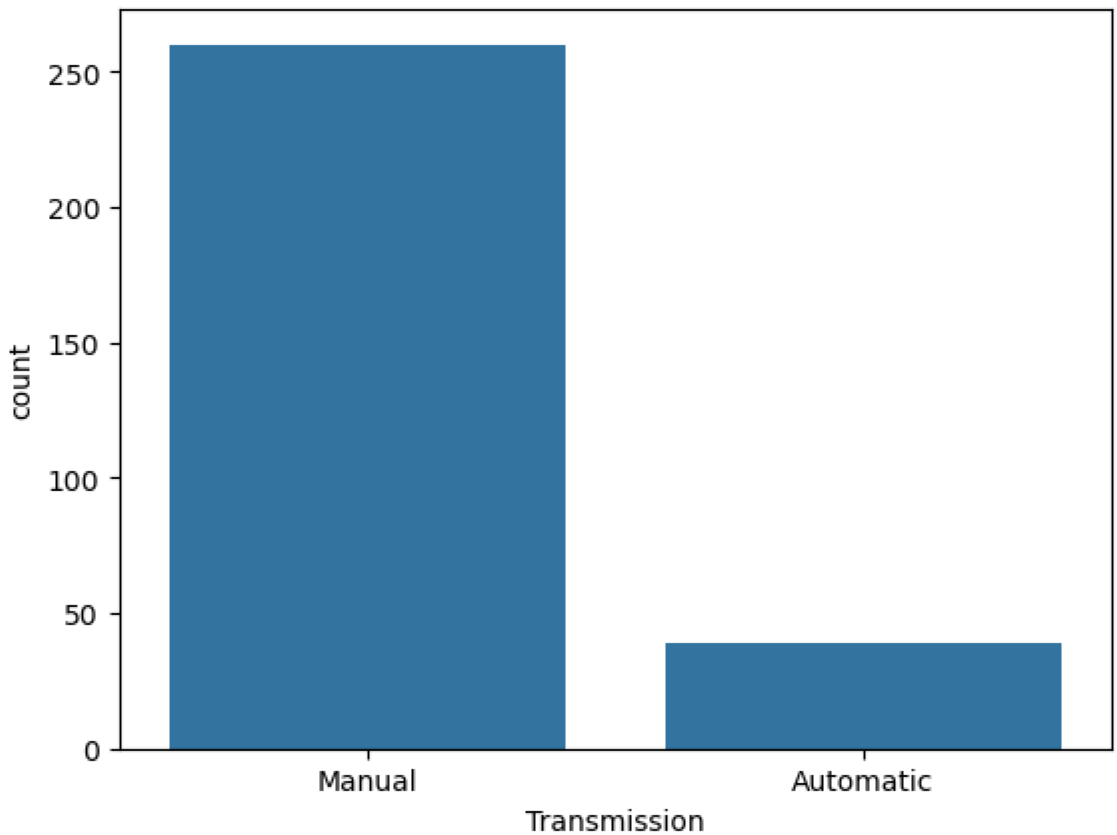
In [72]:
```
sns.countplot(x='Selling_type', data=df)
plt.show()
```



In [74]:
```
df['Transmission'].value_counts()
```

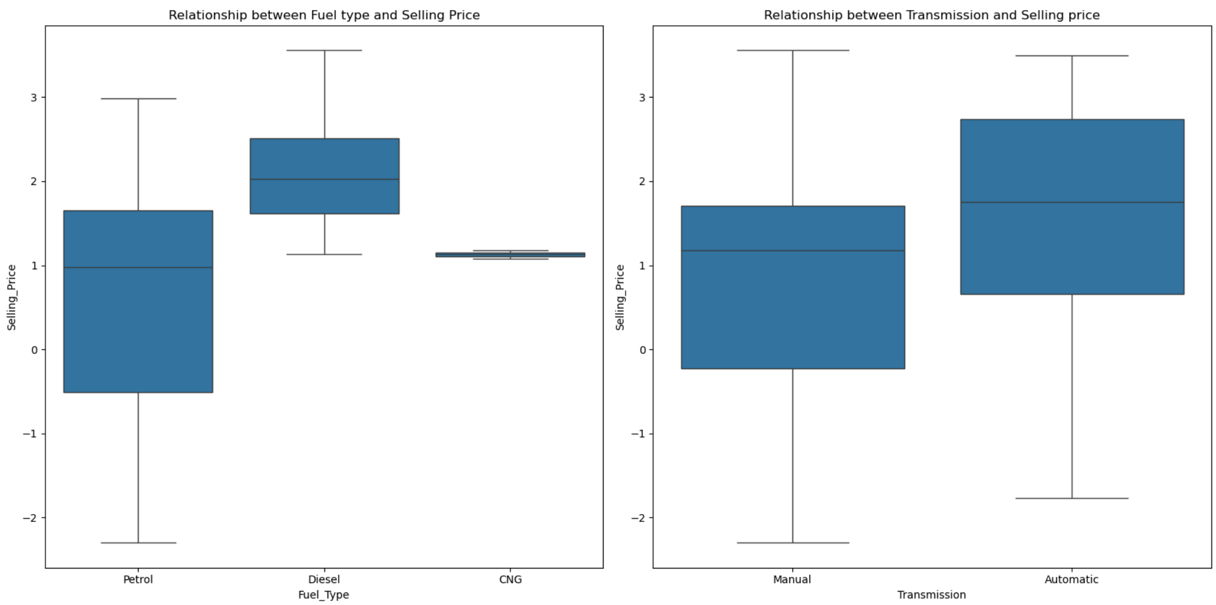
Out[74]:
```
Transmission
Manual       260
Automatic     39
Name: count, dtype: int64
```

In [76]:
```
sns.countplot(x='Transmission', data=df)
plt.show()
```

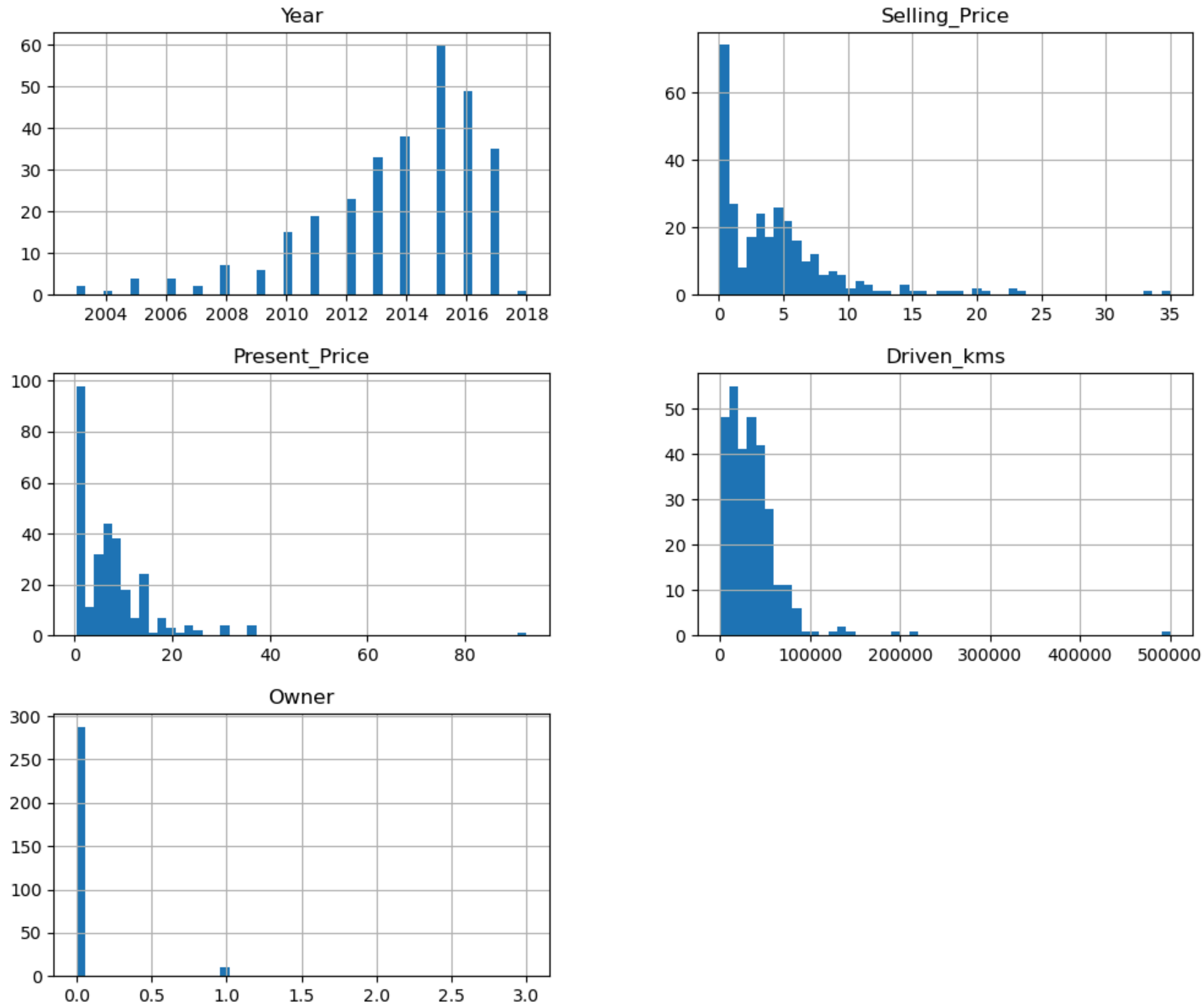


In [52]:
```
# Check for outliers using boxplots
plt.figure(figsize=(10, 6))
sns.boxplot(df['Selling_Price'],palette='pastel')
plt.title('Boxplot of Selling Price')
plt.show()
```

In [80]:
```
plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
sns.boxplot(x='Fuel_Type', y='Selling_Price', data=df)
plt.title('Relationship between Fuel type and Selling Price')
plt.subplot(1, 2, 2)
sns.boxplot(x='Transmission', y='Selling_Price', data=df)
plt.title('Relationship between Transmission and Selling price')
plt.tight_layout()
plt.show()
```



In [54]:
```
df.hist(figsize = (12,10), bins = 50)
plt.show()
```

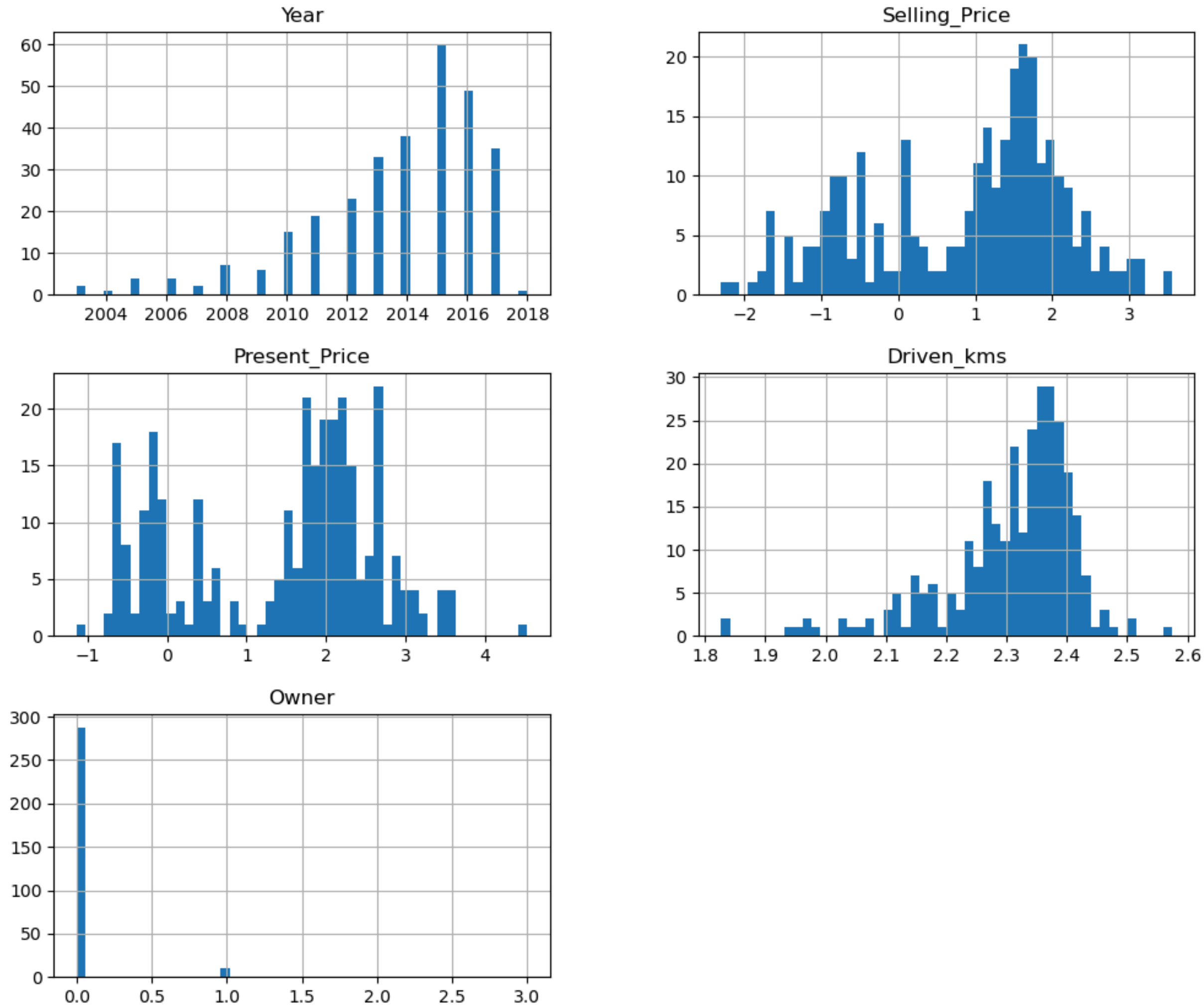


In [60]:

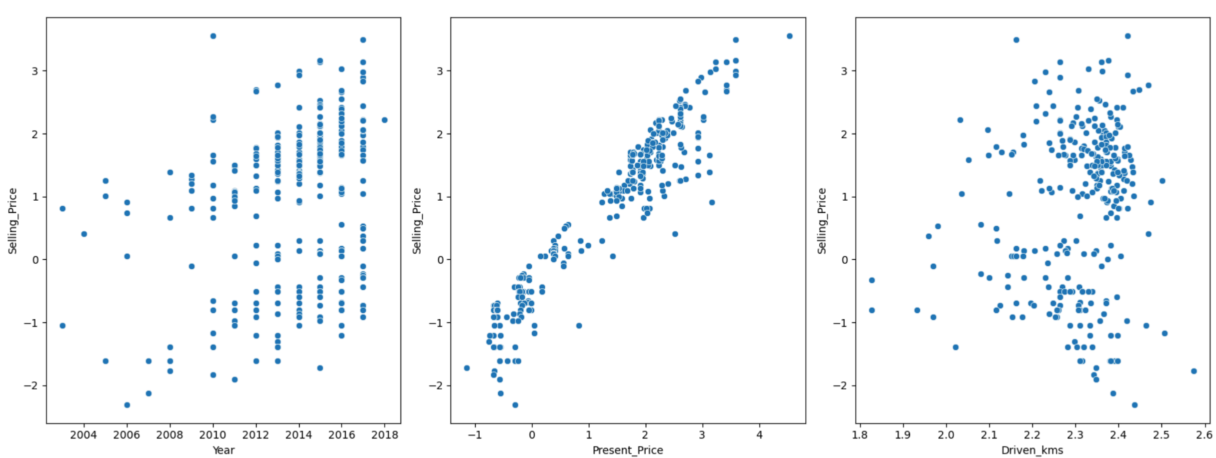
```
df['Driven_kms'] = np.log(df['Driven_kms'])
df['Selling_Price'] = np.log(df['Selling_Price'])
df['Present_Price'] = np.log(df['Present_Price'])
```

In [62]:

```
df.hist(figsize = (12,10), bins = 50)
plt.show()
```



In [78]:
```
#Finding relationships between different numerical features and our target features
plt.figure(figsize=(16, 6))
plt.subplot(1, 3, 1)
sns.scatterplot(x='Year', y='Selling_Price', data=df)
plt.subplot(1, 3, 2)
sns.scatterplot(x='Present_Price', y='Selling_Price', data=df)
plt.subplot(1, 3, 3)
sns.scatterplot(x='Driven_kms', y='Selling_Price', data=df)
plt.tight_layout()
plt.show()
```

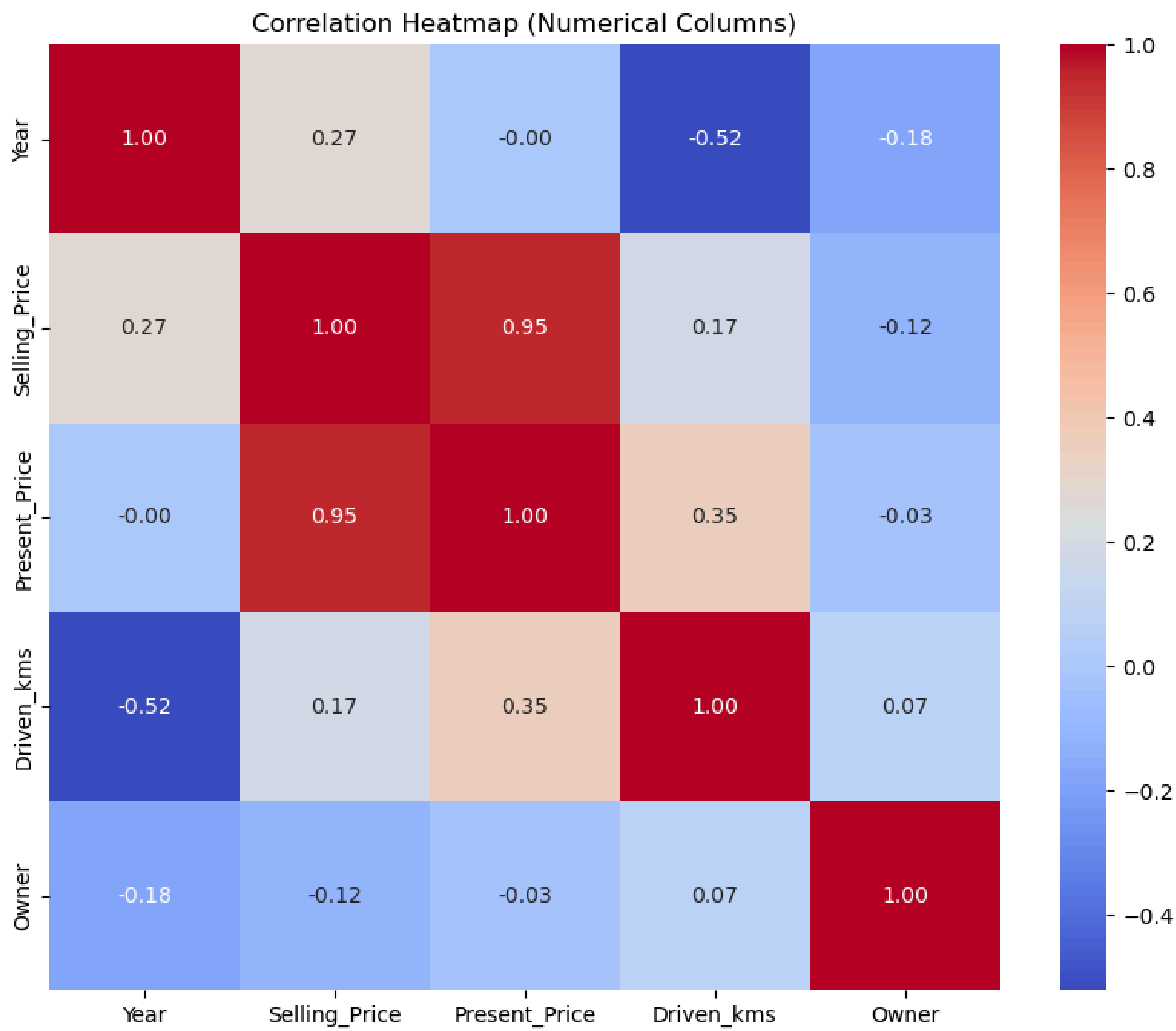


In [66]:
```
# Select numerical columns
numerical_columns = ['Year', 'Selling_Price', 'Present_Price', 'Driven_kms', 'Owner

# Create a DataFrame containing only the numerical columns
numerical_df = df[numerical_columns]

# Calculate the correlation matrix for numerical columns
correlation_matrix = numerical_df.corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap (Numerical Columns)')
plt.show()
```



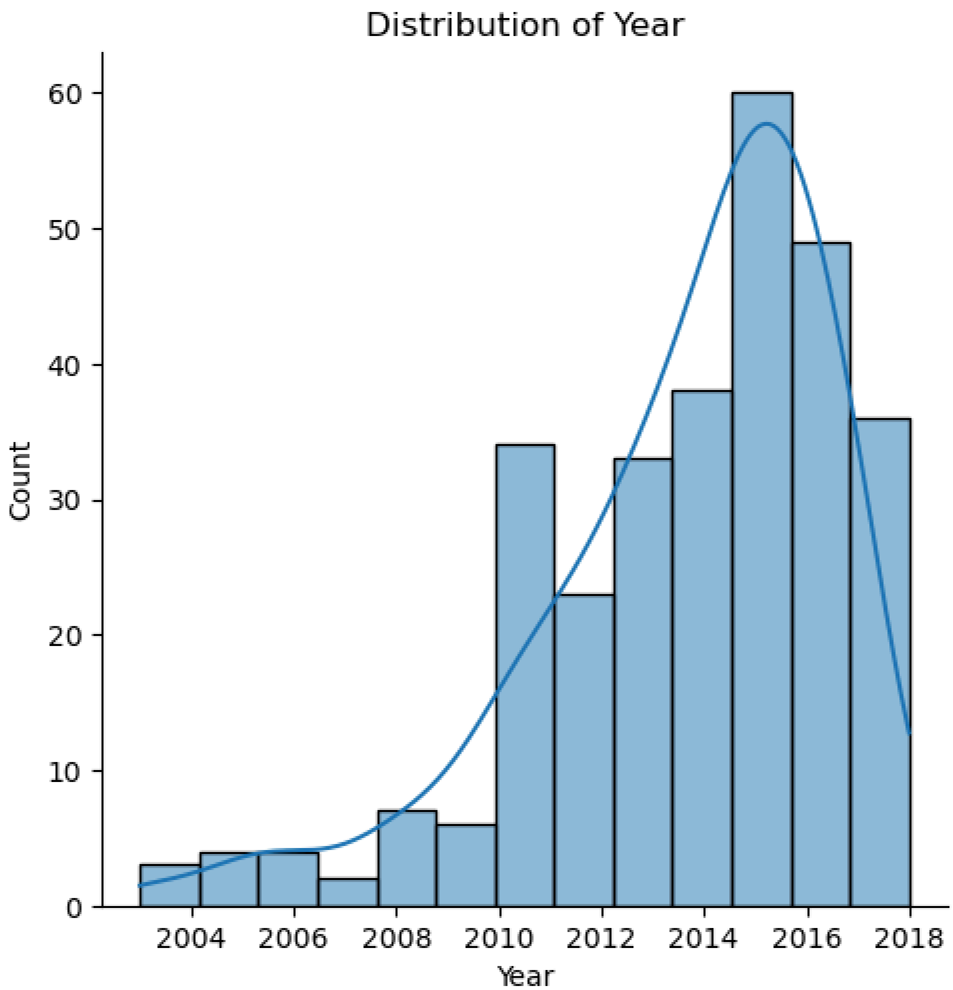
Strong Positive Correlation: Selling Price and Present Price have a very strong positive correlation indicating a nearly perfect linear relationship. This makes sense as present market value likely influences selling price.

Year's Influence: Year seems weakly negatively correlated with Selling Price, Present Price, and Driven_Kms. Newer cars (lower Year values) tend to have higher selling prices, lower present prices, and lower driven kilometers.

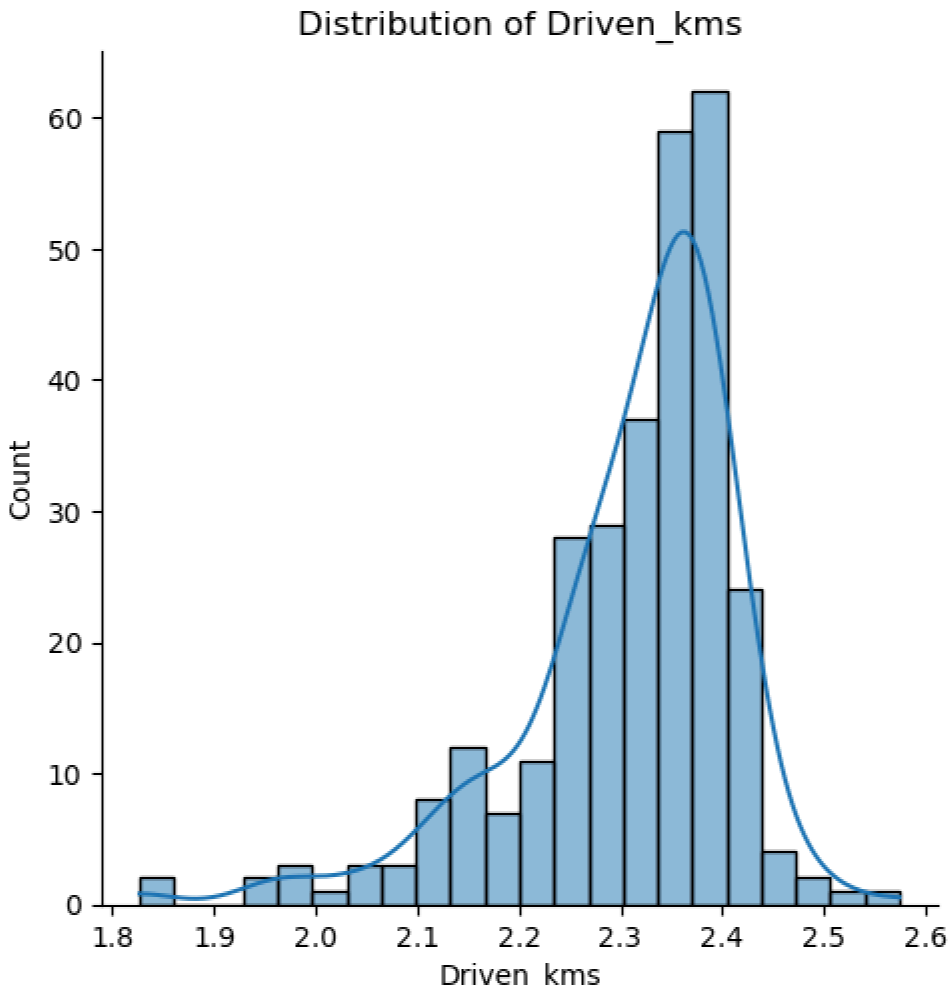
Mileage Matters: Driven_Kms shows a weak negative correlation with Selling Price and Present Price. Cars with lower mileage (lower Driven_Kms values) tend to sell for more.

In [82]:
```
numerical_features = ['Year', 'Driven_kms', 'Selling_Price', 'Present_Price']
for feature in numerical_features:
    plt.figure(figsize=(10, 6))
    sns.displot(data=df, x=feature, kde=True)
    plt.title(f'Distribution of {feature}')
    plt.show()
```

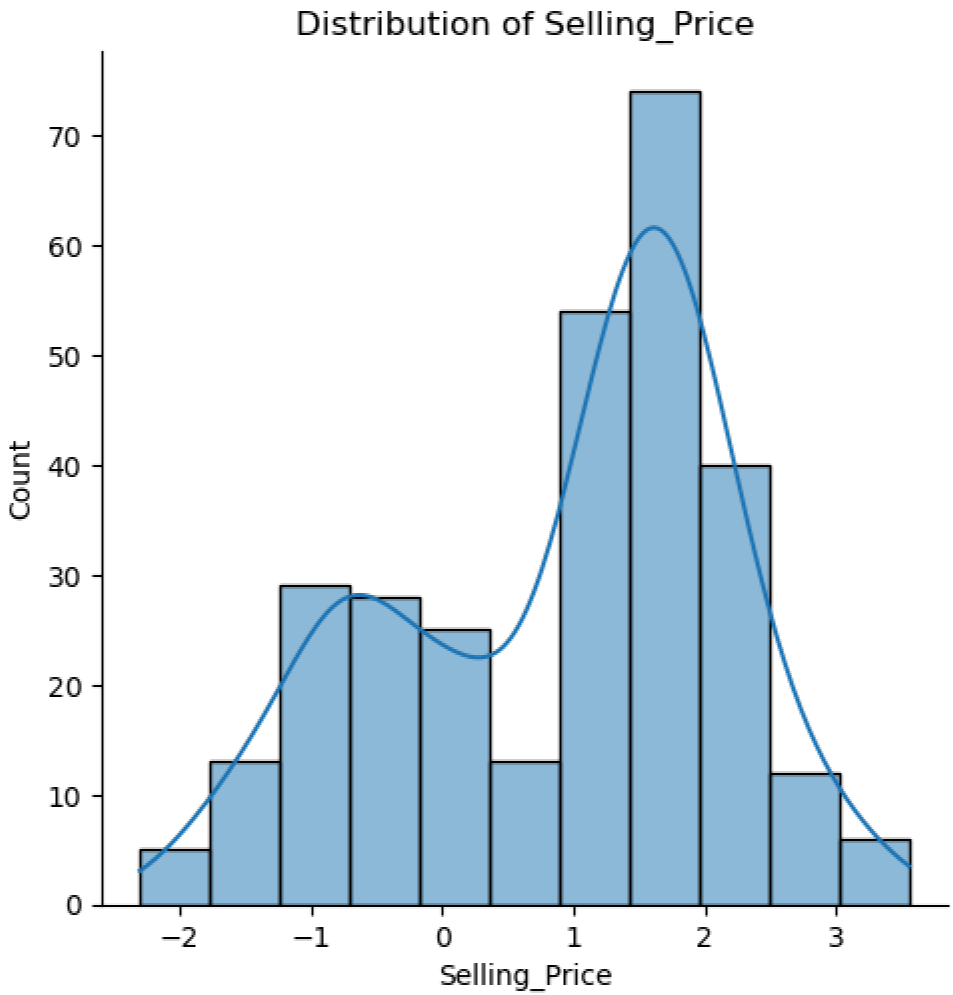
```
<Figure size 1000x600 with 0 Axes>
```



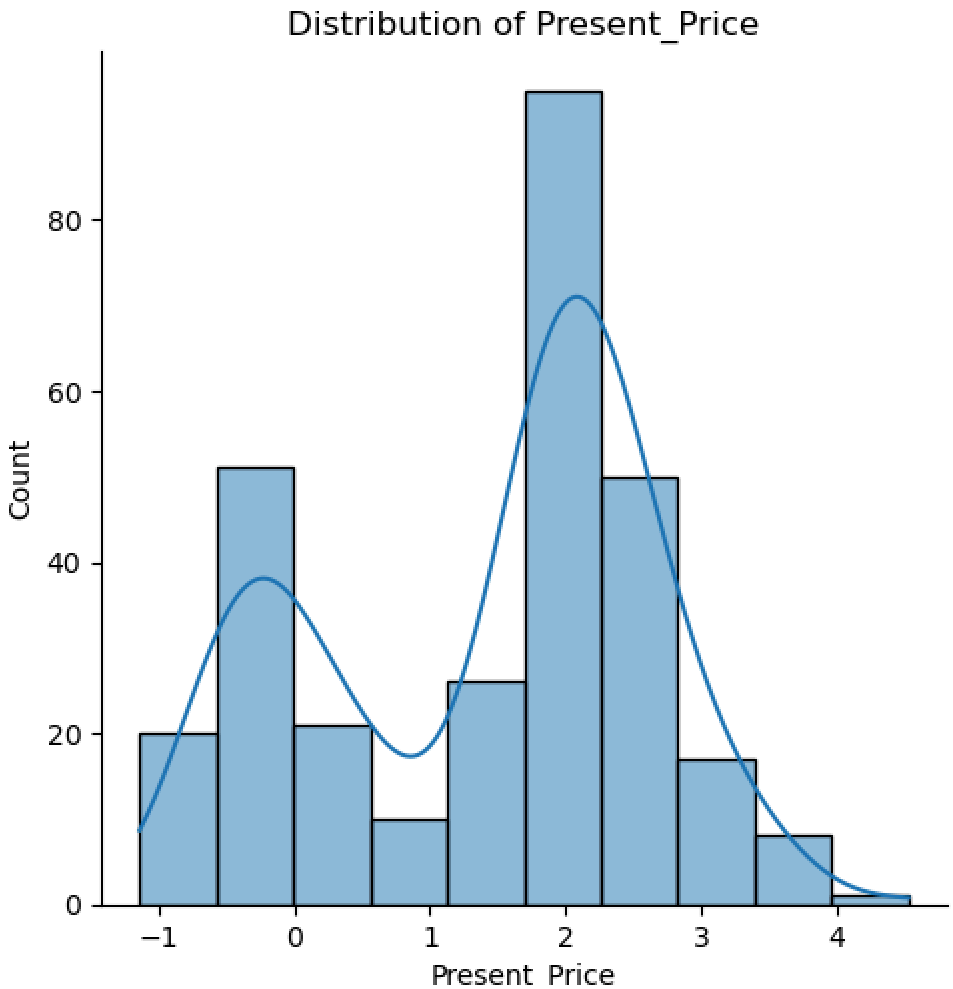

```
<Figure size 1000x600 with 0 Axes>
```



<Figure size 1000x600 with 0 Axes>



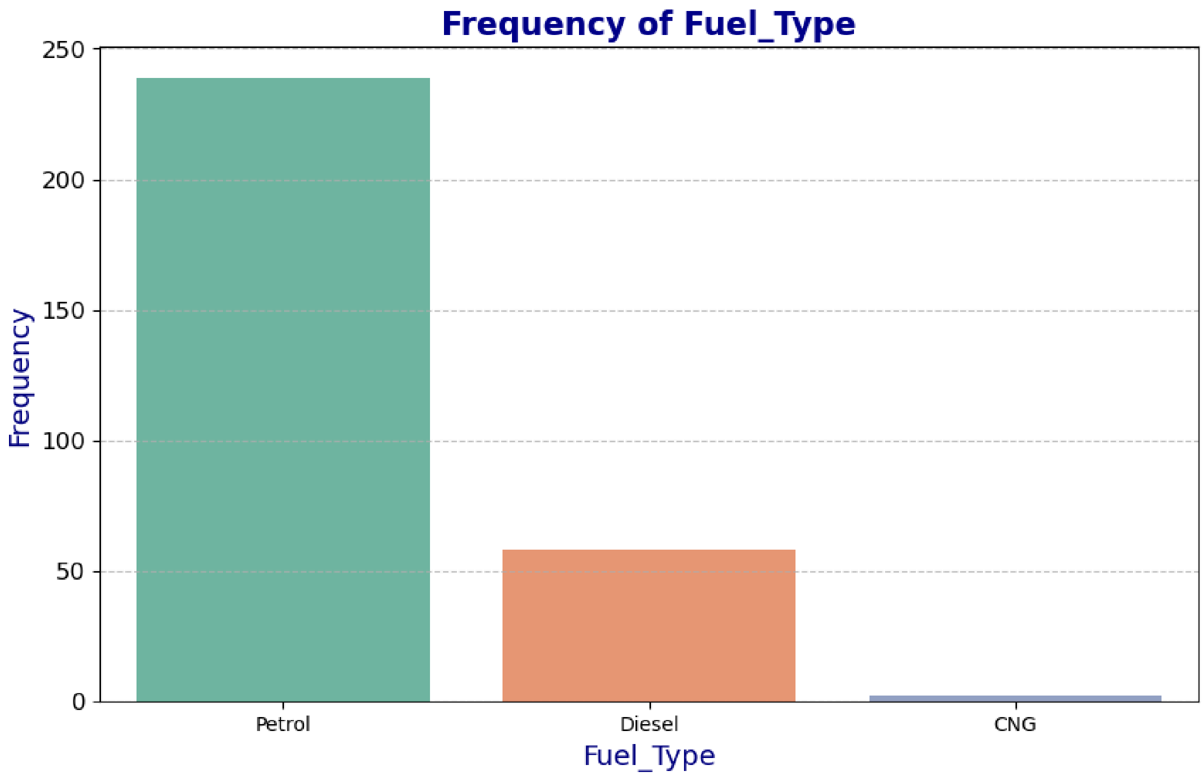
<Figure size 1000x600 with 0 Axes>

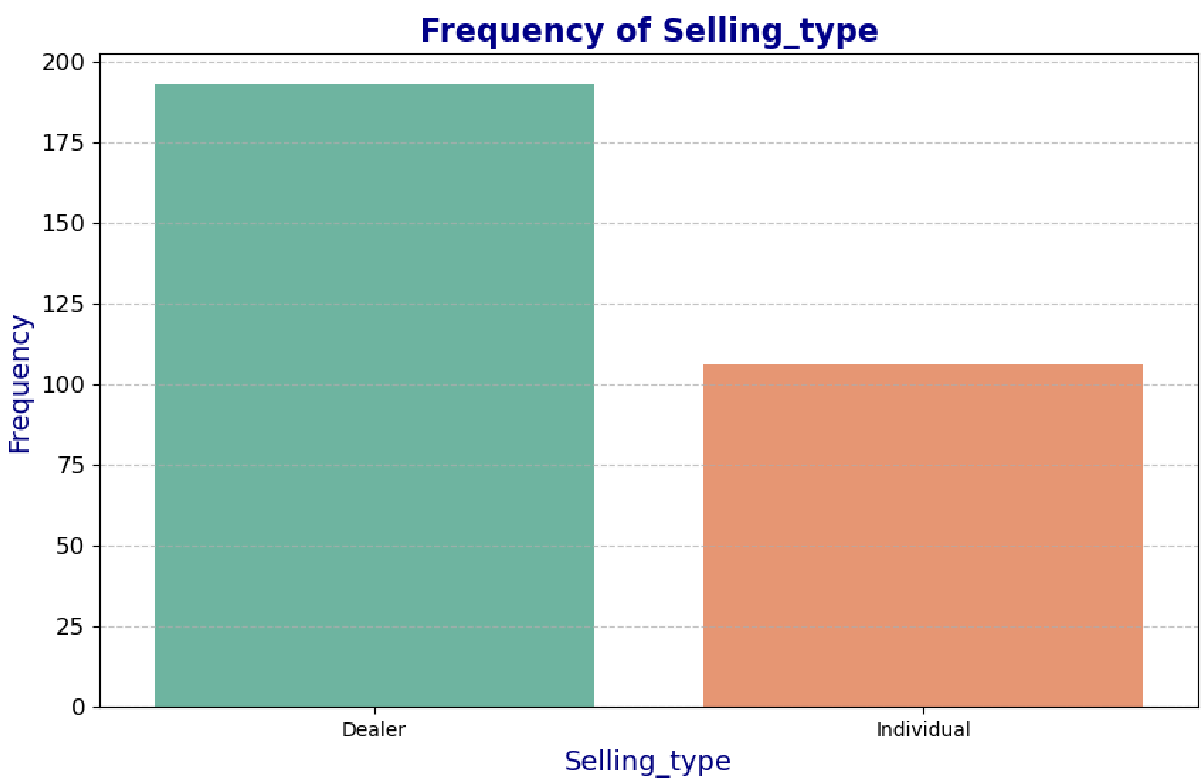


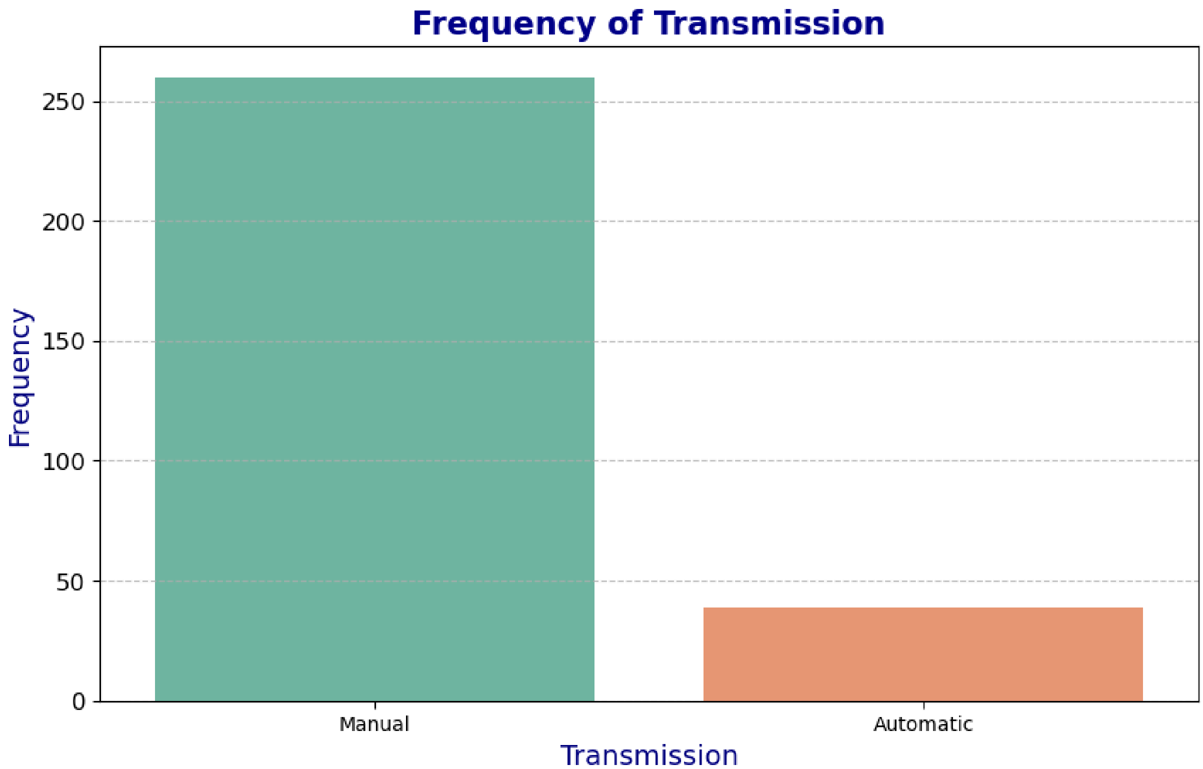
In [84]:
```
# Scatter plots
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Driven_kms', y='Selling_Price', data=df)
plt.title('Relationship between Driven_kms and Selling_Price')
plt.show()
```

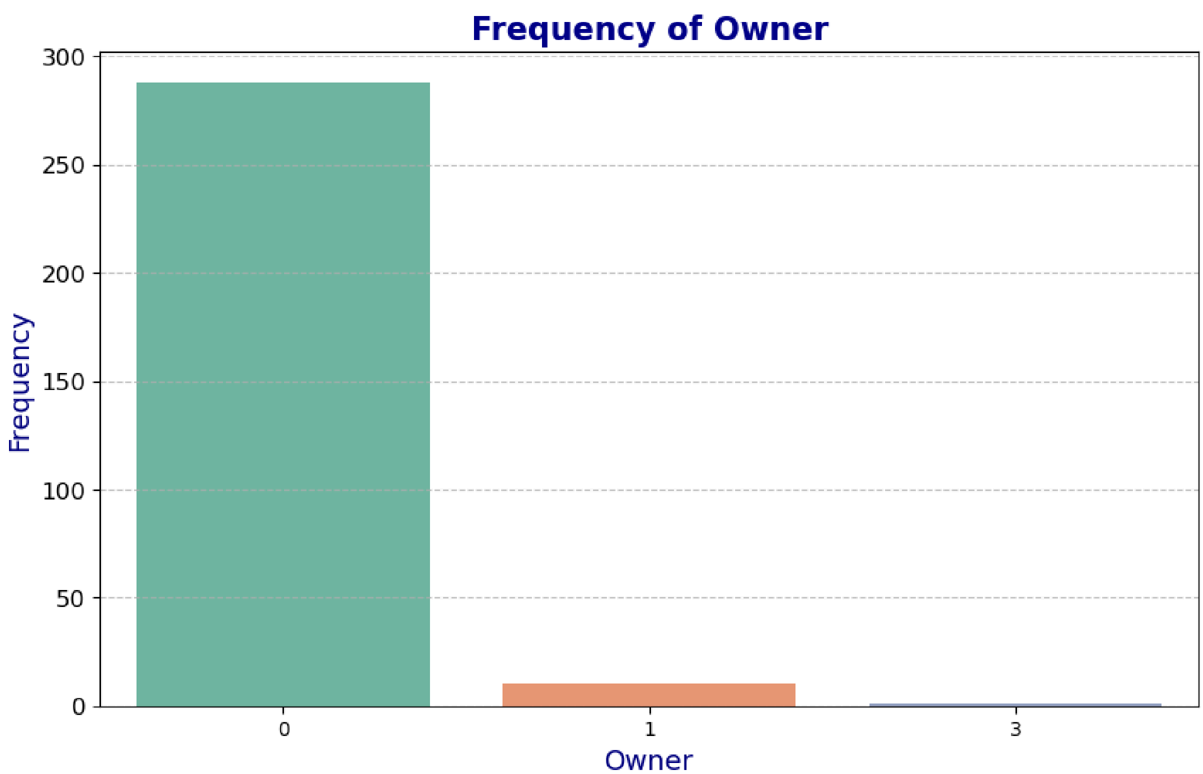
In [86]:
```
plt.figure(figsize=(12, 8))
sns.scatterplot(x='Present_Price', y='Selling_Price', data=df)
plt.title('Relationship between Present_Price and Selling_Price')
plt.xlabel('Present Price')
plt.ylabel('Selling Price')
plt.show()
```

In [88]:
```
# Definea color palette for the plots
colors=sns.color_palette("Set2")
categorical_features=['Fuel_Type', 'Selling_type', 'Transmission','Owner']
# Plot each caterical feature
for feature in categorical_features:
    plt.figure(figsize=(10,6))
    sns.countplot(x=feature,data=df,palette=colors)
    plt.title(f'Frequency of {feature}',fontsize=16,color='darkblue',fontweight='bo
    plt.xlabel(f'{feature}',fontsize=14,color='navy')
    plt.ylabel('Frequency',fontsize=14,color='navy')
    plt.yticks(fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
```


Frequency of Fuel_Type
250
200
150
100
50
0
Frequency
Petrol
Diesel
CNG
Fuel_Type


Frequency of Selling_type
200
175
150
125
100
75
50
25
0
Frequency
Dealer
Individual
Selling_type



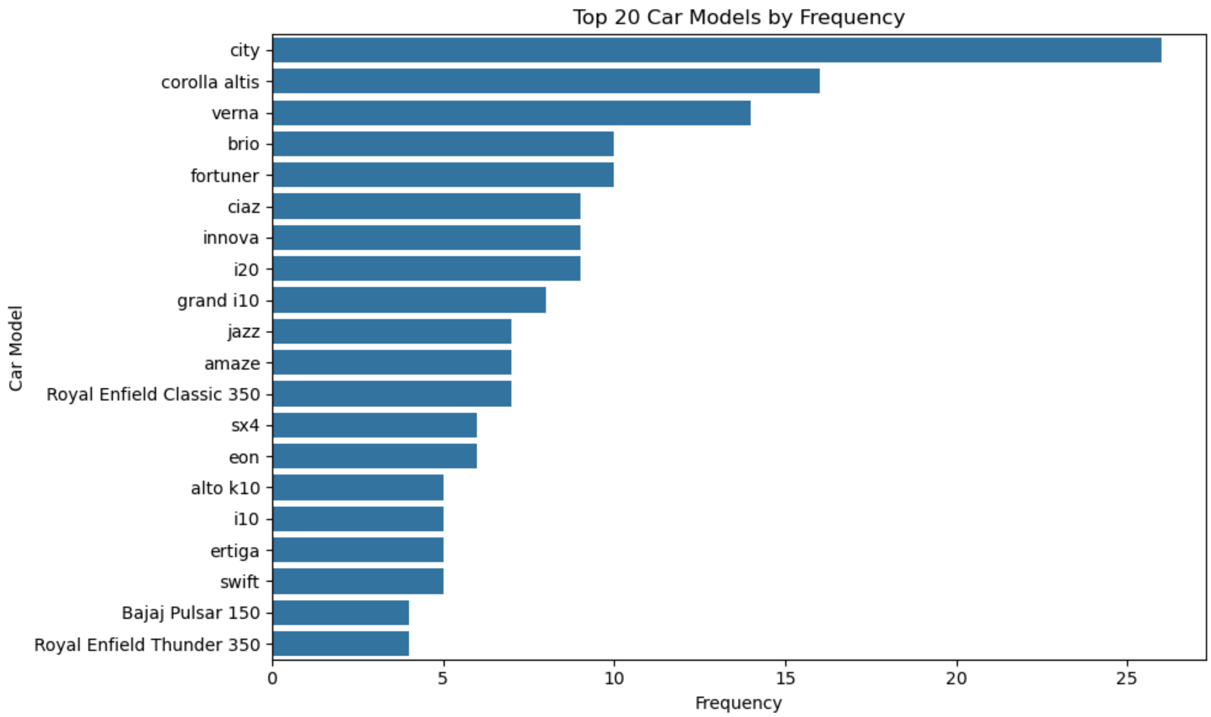


In [90]:
```
df.columns
```

Out[90]:
```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Driven_kms',
       'Fuel_Type', 'Selling_type', 'Transmission', 'Owner'],
      dtype='object')
```

In [92]:
```
n = 20  # Number of top car models to plot
top_car_models = df['Car_Name'].value_counts().head(n)
```

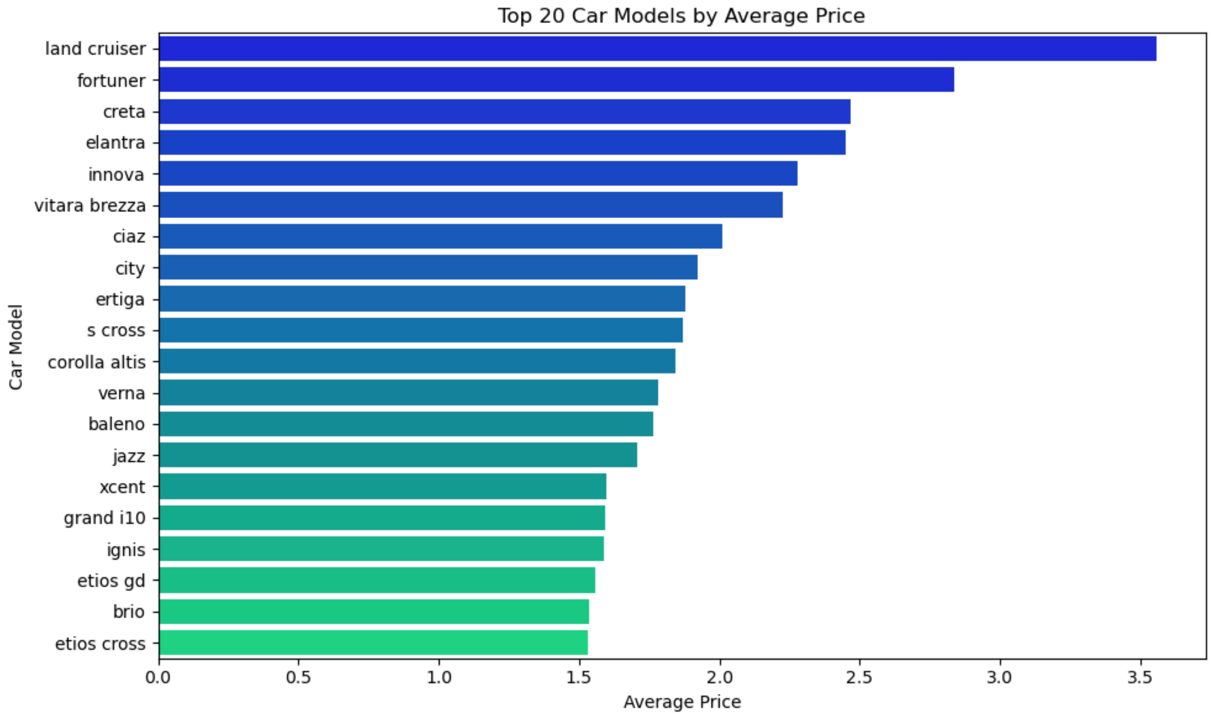
```
plt.figure(figsize=(10, 6))
sns.barplot(x=top_car_models.values, y=top_car_models.index)
plt.title(f'Top {n} Car Models by Frequency')
plt.xlabel('Frequency')
plt.ylabel('Car Model')
plt.tight_layout()
plt.show()
```



In [94]:
```
# Calculate average price for each car model
avg_prices_by_car = df.groupby('Car_Name')['Selling_Price'].mean().sort_values(asce

# Plot top N car models by average price
n = 20  # Number of top car models to plot
top_car_models = avg_prices_by_car.head(n)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_car_models.values, y=top_car_models.index,palette='winter')
plt.title(f'Top {n} Car Models by Average Price')
plt.xlabel('Average Price')
plt.ylabel('Car Model')
plt.tight_layout()
plt.show()
```



Data Cleaning and Transforming the Data

In [116…

```
#One-hot encoding "Fuel_Type" Column
df.replace({'Fuel_Type':{'Petrol':0,'Diesel':1,'CNG':2}},inplace=True)
#One-hot encoding "Seller_Type" Column
df.replace({'Selling_type':{'Dealer':0,'Individual':1}},inplace=True)
#One-hot encoding "Transmission" Column
df.replace({'Transmission':{'Manual':0,'Automatic':1}},inplace=True)
```

In [100…

```
df.head()
```

Out[100…

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Tran
0	ritz	2014	1.208960	1.720979	2.322740	0	0	
1	sx4	2013	1.558145	2.255493	2.367338	1	0	
2	ciaz	2017	1.981001	2.287471	2.179205	0	0	
3	wagon r	2011	1.047319	1.423108	2.146681	0	0	
4	swift	2014	1.526056	1.927164	2.366131	1	0	

Prediction

In [104…

```
X = df.drop(['Car_Name','Selling_Price'],axis=1)
Y = df['Selling_Price']
```

In [106…

```
X
```

Out[106…

	Year	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	2014	1.720979	2.322740	0	0	0	0
1	2013	2.255493	2.367338	1	0	0	0
2	2017	2.287471	2.179205	0	0	0	0
3	2011	1.423108	2.146681	0	0	0	0
4	2014	1.927164	2.366131	1	0	0	0
...	...	...	...	...	...	...	...
296	2016	2.451005	2.345047	1	0	0	0
297	2015	1.774952	2.398086	0	0	0	0
298	2009	2.397895	2.432239	0	0	0	0
299	2017	2.525729	2.208822	1	0	0	0
300	2016	1.774952	2.152452	0	0	0	0

299 rows × 7 columns

In [108…

```
Y
```

Out[108…

```
0      1.208960
1      1.558145
2      1.981001
3      1.047319
4      1.526056
         ...
296    2.251292
297    1.386294
298    1.208960
299    2.442347
300    1.667707
Name: Selling_Price, Length: 299, dtype: float64
```

In [110…

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_sta
```

In [112…

```
# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[112…

```
▾ LinearRegression
LinearRegression()
```

In [114…
```
# Evaluate the model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('Mean Squared Error:', mse)
print('R-squared:', r2)
```

```
Mean Squared Error: 0.042990833662269495
R-squared: 0.9728207242977547
```

In [154…
```
y_pred_linear = model.predict(X_test)
```

In [156…
```
#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: 0.20734231035239648
```

In [158…
```
# Make predictions
new_car = [[2024, 20000, 0, 1, 1, 0, 0]]  # Example new car data
predicted_price = model.predict(new_car)
print('Predicted Selling Price:', predicted_price[0])
```

```
Predicted Selling Price: 18547.17843658374
```

In [160…
```
# Make predictions
new_car = [[2024, 40000, 0, 1, 1, 0, 0]]  # Example new car data
predicted_price = model.predict(new_car)
print('Predicted Selling Price:', predicted_price[0])
```

```
Predicted Selling Price: 37092.70936776349
```

In [162…
```
# Make predictions
new_car = [[2024, 50000, 0, 1, 1, 0, 0]]  # Example new car data
predicted_price = model.predict(new_car)
print('Predicted Selling Price:', predicted_price[0])
```

```
Predicted Selling Price: 46365.474833353364
```

In [164…
```
# Make predictions
new_car = [[2024, 150000, 0, 1, 1, 0, 0]]  # Example new car data
predicted_price = model.predict(new_car)
print('Predicted Selling Price:', predicted_price[0])
```

```
Predicted Selling Price: 139093.1294892521
```

In [166…
```
#Train a Random Forest Model
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(random_state=40)
rf_model.fit(X_train, y_train)
```

Out[166…

```
▾ RandomForestRegressor
RandomForestRegressor(random_state=40)
```

In [168…

```
y_pred_rf = rf_model.predict(X_test)
```

In [170…

```
#Evaluating the Random Forest Model
from sklearn.metrics import mean_squared_error
from math import sqrt
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: 0.23307517098660957
```

In [150…

```
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()
```