Load Libraries

In [58]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import Preprocessing
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
%matplotlib inline
sns.set()
```

In [79]:
```
df = pd.read_csv( 'car_price_prediction.csv')
df
```

Out[79]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fu typ
0	45654403	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybr
1	44731507	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petr
2	45774419	8467	-	HONDA	FIT	2006	Hatchback	No	Petr
3	45769185	3607	862	FORD	Escape	2011	Jeep	Yes	Hybr
4	45809263	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petr
...	...	...	...	...	...	...	...	...	
19232	45798355	8467	-	MERCEDES-BENZ	CLK 200	1999	Coupe	Yes	CN
19233	45778856	15681	831	HYUNDAI	Sonata	2011	Sedan	Yes	Petr
19234	45804997	26108	836	HYUNDAI	Tucson	2010	Jeep	Yes	Dies
19235	45793526	5331	1288	CHEVROLET	Captiva	2007	Jeep	Yes	Dies
19236	45813273	470	753	HYUNDAI	Sonata	2012	Sedan	Yes	Hybr

19237 rows × 18 columns

In [80]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                19237 non-null  int64  
 1   Price             19237 non-null  int64  
 2   Levy              19237 non-null  object 
 3   Manufacturer      19237 non-null  object 
 4   Model             19237 non-null  object 
 5   Prod. year        19237 non-null  int64  
 6   Category          19237 non-null  object 
 7   Leather interior  19237 non-null  object 
 8   Fuel type         19237 non-null  object 
 9   Engine volume     19237 non-null  object 
 10  Mileage           19237 non-null  object 
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  object 
 13  Drive wheels      19237 non-null  object 
 14  Doors             19237 non-null  object 
 15  Wheel             19237 non-null  object 
 16  Color             19237 non-null  object 
 17  Airbags           19237 non-null  int64  
dtypes: float64(1), int64(4), object(13)
memory usage: 2.6+ MB
```

Clean Dataset

In [81]:
```
'''
Preprocessing:
Drop Duplicates - Replacing categorical values - Clean Outliers - Add a new Feat
'''
df = Preprocessing.preprocessing_pipeline(df)
```

```
Preprocessing started...
Initial shape: (19237, 18)
After dropping duplicates: (18924, 18)
Replacing categorical values...
After cleaning outliers: (16035, 18)
Feature engineering...
Dropping columns...
Final shape: (16035, 16)
```

In [82]:
```
# Handling the Object columns and Transform to numerical value
df['Levy']= df['Levy'].replace('-','0')
df['Levy']= pd.to_numeric(df['Levy'])

df['Engine volume']= df['Engine volume'].replace('Turbo','')
df['Engine volume']= pd.to_numeric(df['Engine volume'])

df['Mileage km']= df['Mileage'].replace('km','',)
df['Mileage km']= pd.to_numeric(df['Mileage km'])

df.drop(columns='Mileage',inplace=True)
```

In [ ]:
```
```

EDA:

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 16035 entries, 0 to 18921
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Price             16035 non-null  int64  
 1   Levy              16035 non-null  int64  
 2   Manufacturer      16035 non-null  object 
 3   Model             16035 non-null  object 
 4   Category          16035 non-null  object 
 5   Leather interior  16035 non-null  object 
 6   Fuel type         16035 non-null  object 
 7   Engine volume     16035 non-null  float64
 8   Cylinders         16035 non-null  float64
 9   Gear box type     16035 non-null  object 
 10  Drive wheels      16035 non-null  object 
 11  Wheel             16035 non-null  object 
 12  Color             16035 non-null  object 
 13  Airbags           16035 non-null  int64  
 14  Age               16035 non-null  int64  
 15  Mileage km        16035 non-null  int64  
dtypes: float64(2), int64(5), object(9)
memory usage: 2.1+ MB
```

In [7]:
```
df.isna().sum()
```

Out[7]:
```
Price               0
Levy                0
Manufacturer        0
Model               0
Category            0
Leather interior    0
Fuel type           0
Engine volume       0
Cylinders           0
Gear box type       0
Drive wheels        0
Wheel               0
Color               0
Airbags             0
Age                 0
Mileage km          0
dtype: int64
```

In [8]:
```
df.describe().round(2).T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
Price	16035.0	14294.82	11287.26	1.0	5217.0	12544.0	20385.0	47120.0
Levy	16035.0	575.38	457.74	0.0	0.0	640.0	862.0	2209.0
Engine volume	16035.0	2.13	0.60	0.8	1.6	2.0	2.5	3.5
Cylinders	16035.0	4.37	0.88	1.0	4.0	4.0	4.0	16.0
Airbags	16035.0	6.57	4.25	0.0	4.0	6.0	12.0	16.0
Age	16035.0	14.22	5.52	5.0	11.0	13.0	16.0	86.0
Mileage km	16035.0	130490.58	80334.73	0.0	70994.0	124892.0	180000.0	363661.0

In [9]:
```
df.select_dtypes(include='object').describe()
```

Out[9]:

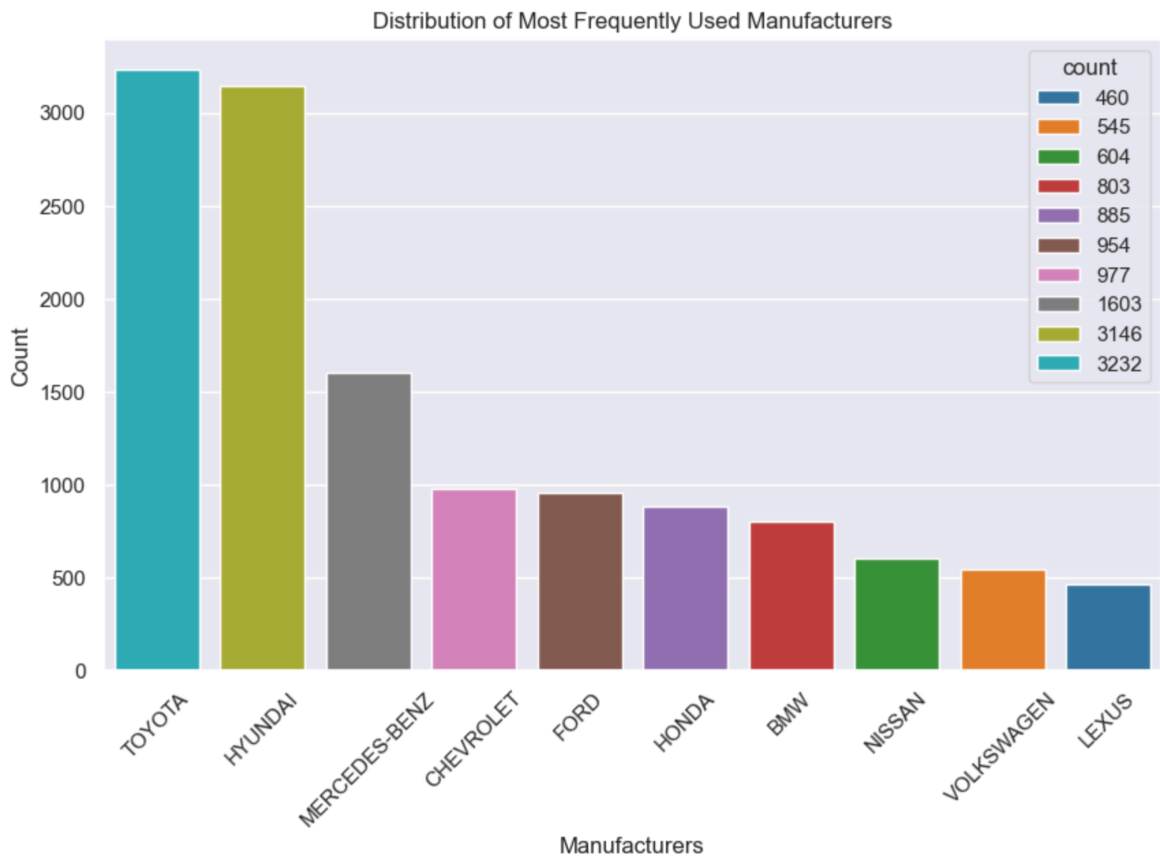
	Manufacturer	Model	Category	Leather interior	Fuel type	Gear box type	Drive wheels	Wheel	C
count	16035	16035	16035	16035	16035	16035	16035	16035	16
unique	59	1318	11	2	7	4	3	2	
top	TOYOTA	Prius	Sedan	Yes	Petrol	Automatic	Front	Left wheel	B
freq	3232	989	7514	11174	8162	11208	11525	14668	3

In [10]:
```
# Most frequently used manufacturer
df['Manufacturer'].value_counts().head(10)
```

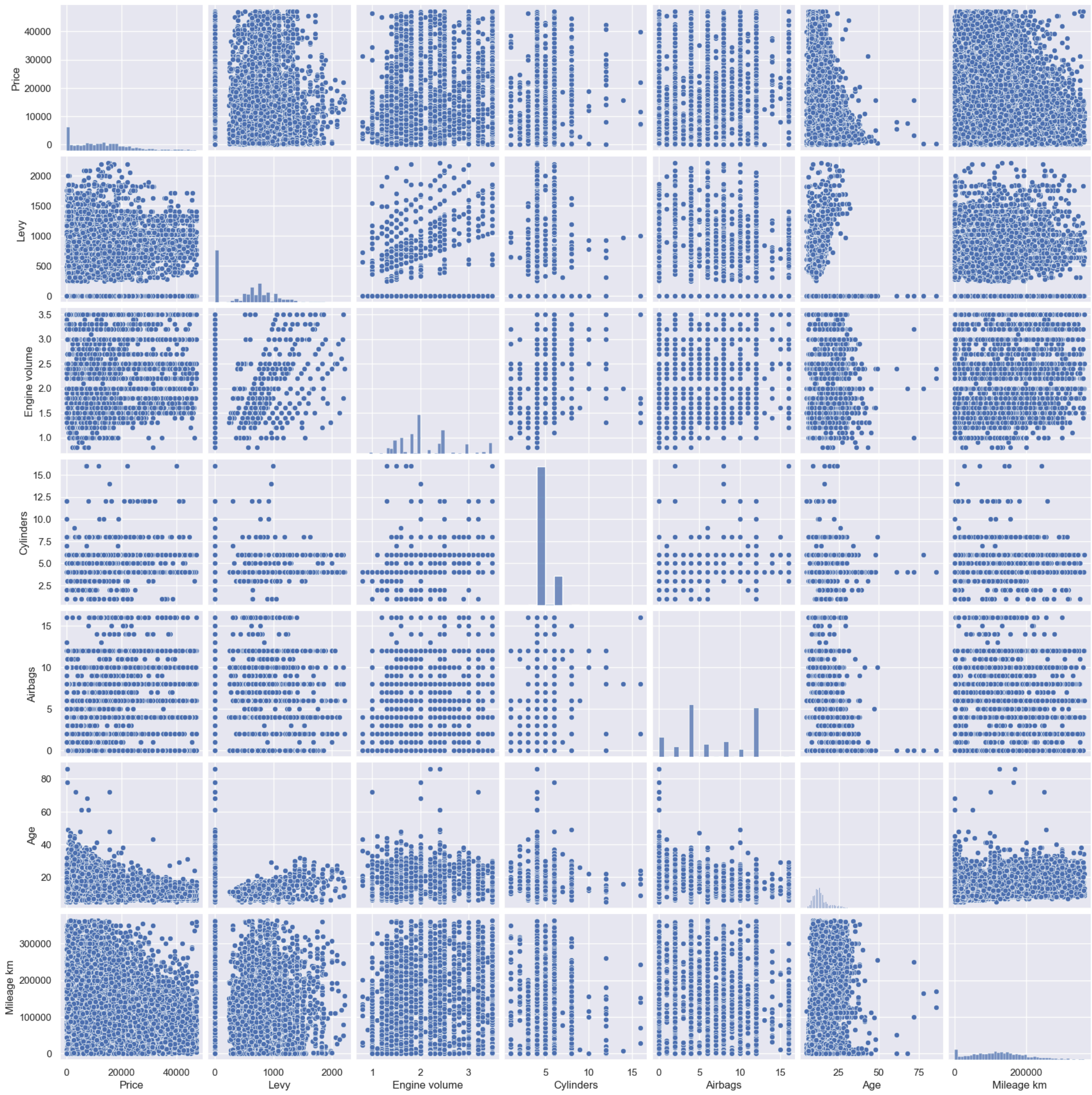
Out[10]:
```
Manufacturer
TOYOTA           3232
HYUNDAI          3146
MERCEDES-BENZ    1603
CHEVROLET         977
FORD              954
HONDA             885
BMW               803
NISSAN            604
VOLKSWAGEN        545
LEXUS             460
Name: count, dtype: int64
```

In [11]:
```
# Top 10 most frequently used manufacturers
top_manufacturers = df['Manufacturer'].value_counts().head(10)

plt.figure(figsize=(10,6))
sns.barplot(x= top_manufacturers.index, y=top_manufacturers.values,hue=top_manuf
plt.title('Distribution of Most Frequently Used Manufacturers')
plt.xticks(rotation=45)
plt.xlabel('Manufacturers')
plt.ylabel('Count')
plt.show()
```



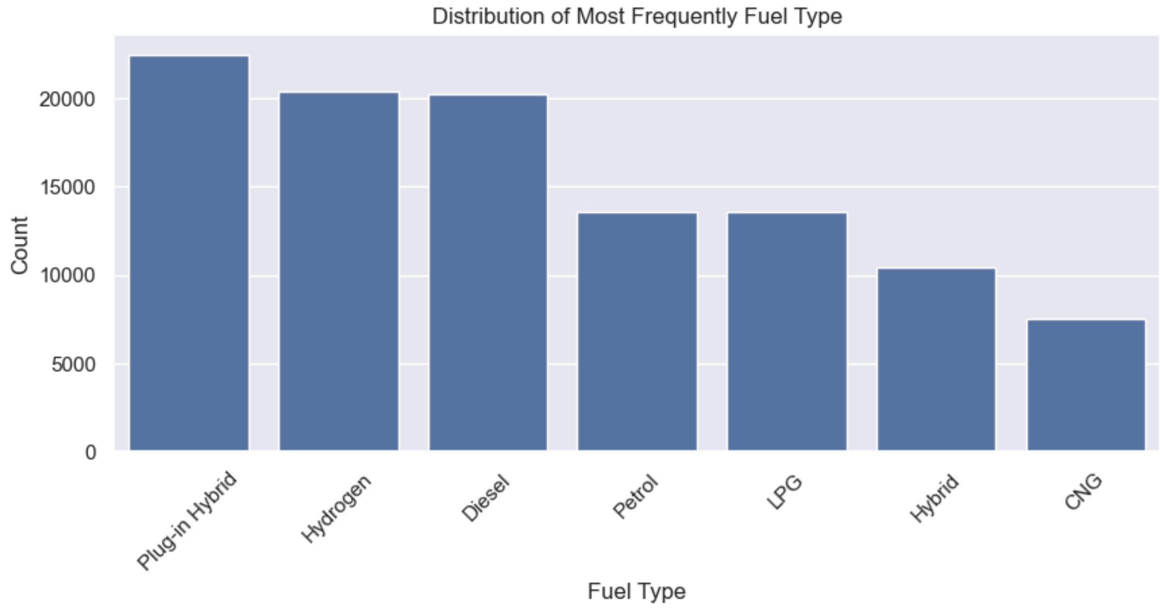
In [12]:
```
#Relationship between the columns
sns.pairplot(df)
plt.show()
```



In [13]:
```
# most frequently fuel type
fuel_type = df.groupby('Fuel type')['Price'].mean().round(1).sort_values(ascendi
fuel_type
```

Out[13]:
```
Fuel type
Plug-in Hybrid    22445.0
Hydrogen          20385.0
Diesel            20245.0
Petrol            13580.9
LPG               13547.9
Hybrid            10463.2
CNG                7539.1
Name: Price, dtype: float64
```

In [14]:
```
plt.figure(figsize=(10,4))
sns.barplot(x= fuel_type.index, y=fuel_type.values)
plt.title('Distribution of Most Frequently Fuel Type')
plt.xticks(rotation=45)
plt.xlabel('Fuel Type')
plt.ylabel('Count')
plt.show()
```



In [15]:
```
#Relationship between numerical columns
numeric_column = df.select_dtypes('number')
numeric_column.corr().round(2)
```

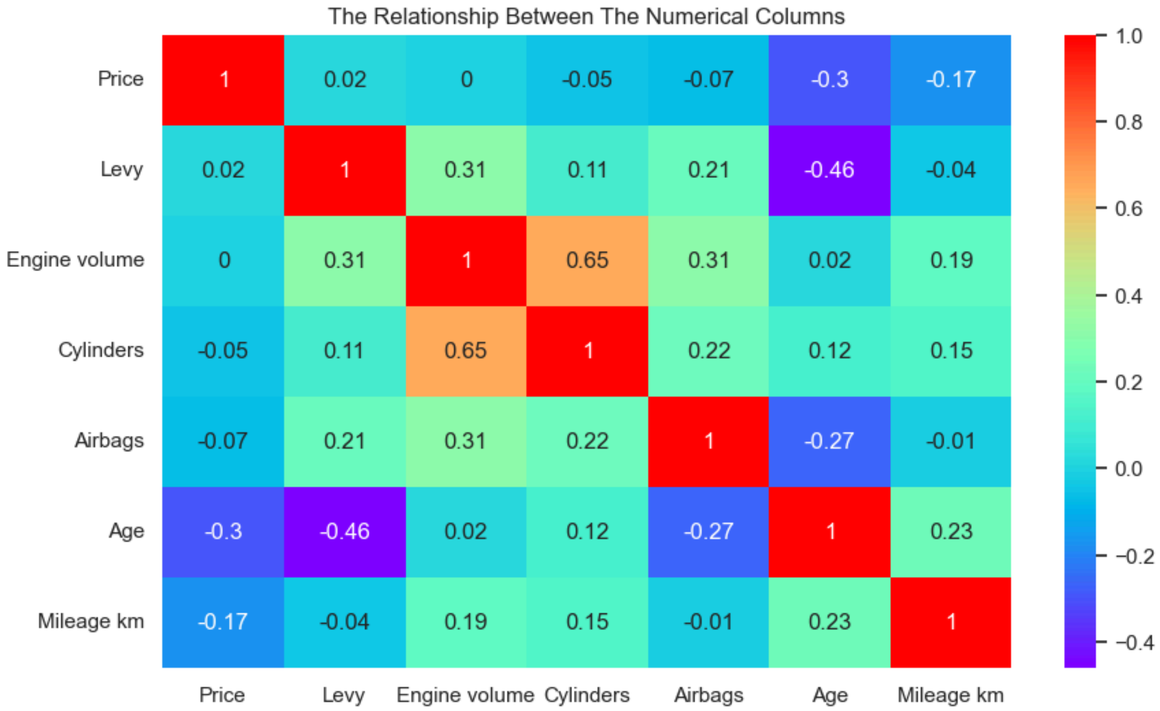
Out[15]:

	Price	Levy	Engine volume	Cylinders	Airbags	Age	Mileage km
Price	1.00	0.02	0.00	-0.05	-0.07	-0.30	-0.17
Levy	0.02	1.00	0.31	0.11	0.21	-0.46	-0.04
Engine volume	0.00	0.31	1.00	0.65	0.31	0.02	0.19
Cylinders	-0.05	0.11	0.65	1.00	0.22	0.12	0.15
Airbags	-0.07	0.21	0.31	0.22	1.00	-0.27	-0.01
Age	-0.30	-0.46	0.02	0.12	-0.27	1.00	0.23
Mileage km	-0.17	-0.04	0.19	0.15	-0.01	0.23	1.00

In [16]:
```
#Columns that affect Price
price_corr = numeric_column.corr()['Price'].sort_values(ascending=False)
price_corr
```

Out[16]:
```
Price            1.000000
Levy             0.019057
Engine volume    0.003895
Cylinders       -0.052854
Airbags         -0.068340
Mileage km      -0.171228
Age             -0.296334
Name: Price, dtype: float64
```

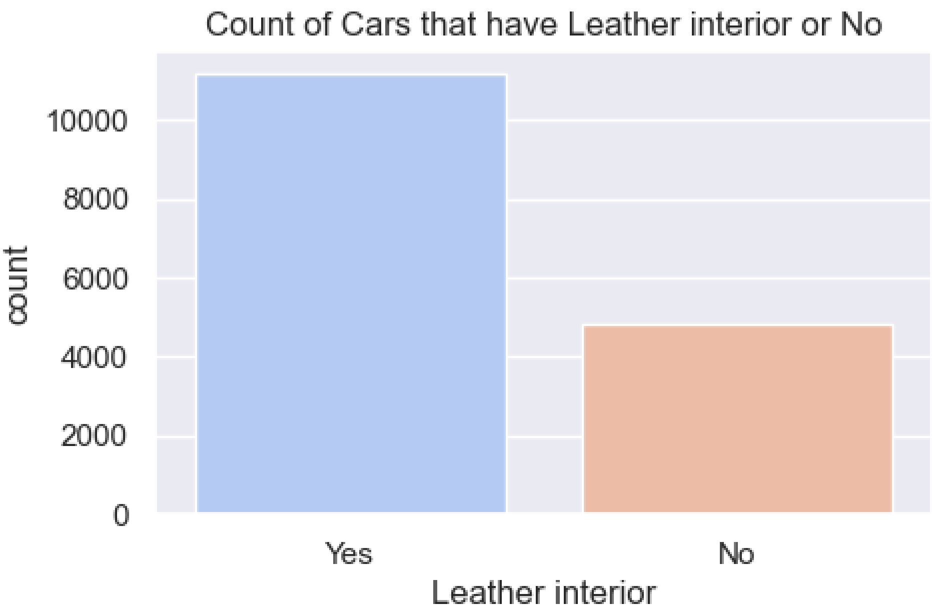
In [17]:
```
plt.figure(figsize=(10,6))
sns.heatmap(numeric_column.corr().round(2), annot=True, cmap= 'rainbow')
plt.title('The Relationship Between The Numerical Columns')
plt.show()
```



In [18]:
```
#Count of cars that have genunie Leather interior and thet have artificial Leath
leather_interior = df.groupby('Leather interior')['Price'].count()
leather_interior
```

Out[18]:
```
Leather interior
No      4861
Yes    11174
Name: Price, dtype: int64
```

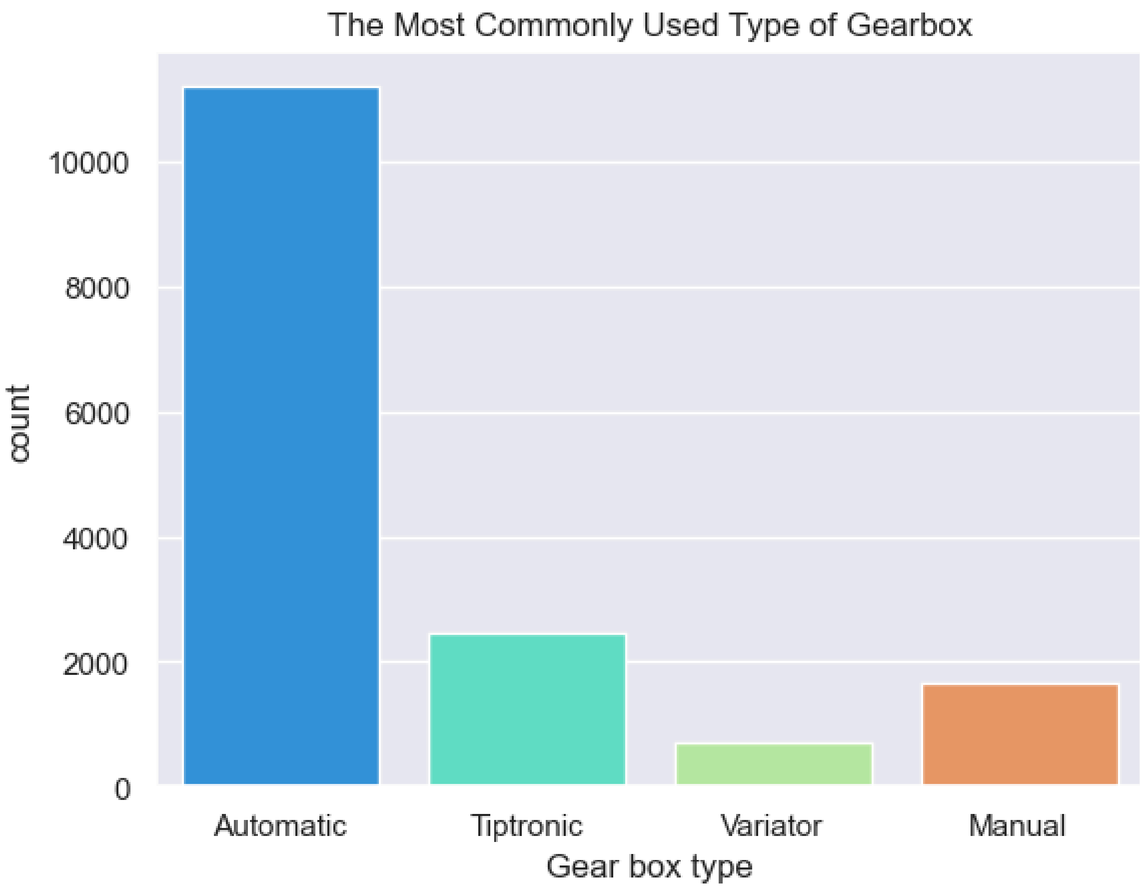
In [19]:
```
plt.figure(figsize=(5,3))
sns.countplot(df, x='Leather interior', hue='Leather interior', palette= 'coolwa
plt.title('Count of Cars that have Leather interior or No')
plt.show()
```



In [20]:
```
#Most used gearbox
gear_box= df.groupby('Gear box type')['Price'].count()
gear_box
```

Out[20]:
```
Gear box type
Automatic    11208
Manual        1668
Tiptronic     2451
Variator       708
Name: Price, dtype: int64
```

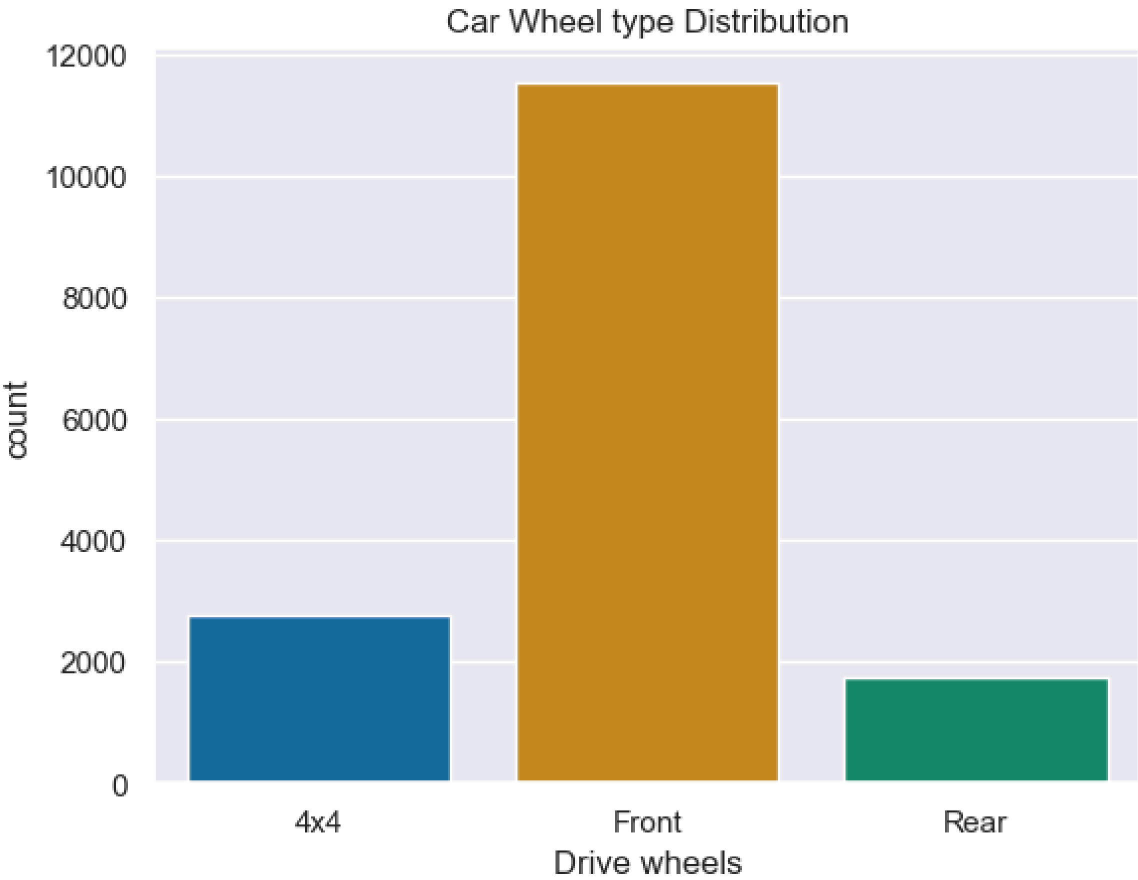
In [21]:
```
sns.countplot(df,x='Gear box type',hue='Gear box type', palette= 'rainbow')
plt.title('The Most Commonly Used Type of Gearbox')
plt.show()
```



In [22]:
```
#Most used Drive Wheels type
drive_wheels= df.groupby('Drive wheels')['Price'].count()
drive_wheels
```

Out[22]:
```
Drive wheels
4x4       2753
Front    11525
Rear      1757
Name: Price, dtype: int64
```

In [23]:
```
sns.countplot(df,x= 'Drive wheels',hue='Drive wheels', palette= 'colorblind')
plt.title('Car Wheel type Distribution')
plt.show()
```



Machine Learning:

In [30]:
```
df.head()
```

Out[30]:

	Price	Levy	Manufacturer	Model	Category	Fuel type	Engine volume	Cylinders	Color	Airba
0	13328	1399	28	1035	4	2	3.5	6.0	12	
1	16621	1018	6	563	4	5	3.0	6.0	1	
2	8467	0	18	585	3	5	1.3	4.0	1	
3	3607	862	13	565	4	2	2.5	4.0	14	
4	11726	446	18	585	3	5	1.3	4.0	12	

5 rows × 23 columns

In [83]:
```
# Define the columns to be one-hot encoded
one_hot_columns = ['Leather interior', 'Gear box type', 'Drive wheels', 'Wheel']

# Create a OneHotEncoder object with the desired settings
oh_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Fit the encoder and transform the training data
oh_encoder_train = oh_encoder.fit_transform(df[one_hot_columns])

# Get the new column names after one-hot encoding
```

```
oh_encoded_columns = oh_encoder.get_feature_names_out(one_hot_columns)

# Create a new DataFrame for the one-hot encoded columns
oh_encoded_train_df = pd.DataFrame(oh_encoder_train, columns=oh_encoded_columns,

# Concatenate the original DataFrame with the one-hot encoded DataFrame
df = pd.concat([df, oh_encoded_train_df], axis=1)

# Drop the original columns that were one-hot encoded
df.drop(columns=one_hot_columns, inplace=True)
```

In [84]:
```
# Define the columns to be label encoded
label_encode_columns = ['Manufacturer', 'Model', 'Category', 'Fuel type', 'Color

# Create a dictionary to store the label encoders for each column
label_encoders = {}

# Iterate over each column to apply label encoding
for column in label_encode_columns:
    # Create a LabelEncoder object
    label_encoder = LabelEncoder()

    # Fit the encoder and transform the column in the DataFrame
    df[column] = label_encoder.fit_transform(df[column])

    # Store the fitted label encoder in the dictionary
    label_encoders[column] = label_encoder
```

In [ ]:

1- Linear Regression

In [85]:
```
lr = LinearRegression()
scores_linear_regression = cross_val_score(lr, X, y, cv=5, scoring='r2')
print(f'R2 Score with Linear Regression = {scores_linear_regression.mean().round
```

```
R2 Score with Linear Regression = 0.24
```

2- Random Forest Regressor

In [86]:
```
rf = RandomForestRegressor()
scores_random_forest = cross_val_score(rf, X, y, cv=5, scoring='r2')
print(f'R2 Score with Random Forest Regressor = {scores_random_forest.mean().rou
```

```
C:\Users\student7\anaconda3\Lib\site-packages\sklearn\base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
C:\Users\student7\anaconda3\Lib\site-packages\sklearn\base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
C:\Users\student7\anaconda3\Lib\site-packages\sklearn\base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
C:\Users\student7\anaconda3\Lib\site-packages\sklearn\base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
C:\Users\student7\anaconda3\Lib\site-packages\sklearn\base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
R2 Score with Random Forest Regressor = 0.77
```

In [ ]:

Deep Learning:

In [87]:
```
X = StandardScaler().fit_transform(X)
```

In [88]:
```
ann = Sequential()
ann.add(Dense(25, input_dim= 22, activation='relu')) #input

ann.add(Dense(625, activation='relu')) #hidden

ann.add(Dense(1,activation='linear')) #output
```

```
C:\Users\student7\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [89]:
```
ann.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape
dense_9 (Dense)	(None, 25)
dense_10 (Dense)	(None, 625)
dense_11 (Dense)	(None, 1)

Total params: 17,451 (68.17 KB)

Trainable params: 17,451 (68.17 KB)

Non-trainable params: 0 (0.00 B)

In [90]:
```
ann.compile(optimizer='adam', loss='mean_squared_error')
```

In [91]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [92]:
```
hist= ann.fit(X_train,y_train, epochs=100, batch_size=25, verbose=1)
```

```
Epoch 1/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 304765504.0000
Epoch 2/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 99093088.0000
Epoch 3/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 86885568.0000
Epoch 4/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 84314024.0000
Epoch 5/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 84023512.0000
Epoch 6/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 80020856.0000
Epoch 7/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 78018256.0000
Epoch 8/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 79333424.0000
Epoch 9/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 78017736.0000
Epoch 10/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 75039536.0000
Epoch 11/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 75010448.0000
Epoch 12/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 75903280.0000
Epoch 13/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 72387528.0000
Epoch 14/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 72255848.0000
Epoch 15/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 73274344.0000
Epoch 16/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 70961760.0000
Epoch 17/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 69953408.0000
Epoch 18/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 67956784.0000
Epoch 19/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 68356096.0000
Epoch 20/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 66821184.0000
Epoch 21/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 65494604.0000
Epoch 22/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 63735976.0000
Epoch 23/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 62734204.0000
Epoch 24/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 61502452.0000
Epoch 25/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 61591064.0000
Epoch 26/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 59354372.0000
Epoch 27/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 57166568.0000
Epoch 28/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 54785160.0000
Epoch 29/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 54924224.0000
Epoch 30/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 54371556.0000
```

```
Epoch 31/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 50525308.0000
Epoch 32/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 50854356.0000
Epoch 33/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 50095600.0000
Epoch 34/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 49564228.0000
Epoch 35/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 47634500.0000
Epoch 36/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 48587860.0000
Epoch 37/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 46469396.0000
Epoch 38/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 48332208.0000
Epoch 39/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 48324068.0000
Epoch 40/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 47665772.0000
Epoch 41/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 46887376.0000
Epoch 42/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 46719536.0000
Epoch 43/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 46522112.0000
Epoch 44/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44215292.0000
Epoch 45/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45187036.0000
Epoch 46/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45744004.0000
Epoch 47/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43647804.0000
Epoch 48/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45254292.0000
Epoch 49/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 46476420.0000
Epoch 50/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45242312.0000
Epoch 51/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45000052.0000
Epoch 52/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43651512.0000
Epoch 53/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45041604.0000
Epoch 54/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45271284.0000
Epoch 55/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44024032.0000
Epoch 56/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45567520.0000
Epoch 57/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45127076.0000
Epoch 58/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44853144.0000
Epoch 59/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44972316.0000
Epoch 60/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44729316.0000
```

```
Epoch 61/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43066952.0000
Epoch 62/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43586984.0000
Epoch 63/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45392864.0000
Epoch 64/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45365012.0000
Epoch 65/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43374176.0000
Epoch 66/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43206288.0000
Epoch 67/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45235908.0000
Epoch 68/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43240308.0000
Epoch 69/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43073660.0000
Epoch 70/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43547672.0000
Epoch 71/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43944820.0000
Epoch 72/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42863504.0000
Epoch 73/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43050120.0000
Epoch 74/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44079348.0000
Epoch 75/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 41865744.0000
Epoch 76/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44269896.0000
Epoch 77/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42601840.0000
Epoch 78/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42896972.0000
Epoch 79/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42076788.0000
Epoch 80/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42522552.0000
Epoch 81/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43827352.0000
Epoch 82/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42613732.0000
Epoch 83/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44798056.0000
Epoch 84/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42482340.0000
Epoch 85/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44200848.0000
Epoch 86/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42278164.0000
Epoch 87/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42647160.0000
Epoch 88/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 41852220.0000
Epoch 89/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43746044.0000
Epoch 90/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42720916.0000
```

```
Epoch 91/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44095084.0000
Epoch 92/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42318800.0000
Epoch 93/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 41620384.0000
Epoch 94/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 44715224.0000
Epoch 95/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42595912.0000
Epoch 96/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43198044.0000
Epoch 97/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 41303592.0000
Epoch 98/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43372816.0000
Epoch 99/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43033144.0000
Epoch 100/100
514/514 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43761436.0000
```

In [93]:

```
hist.history.keys()
```

Out[93]:

```
dict_keys(['loss'])
```

In [94]:

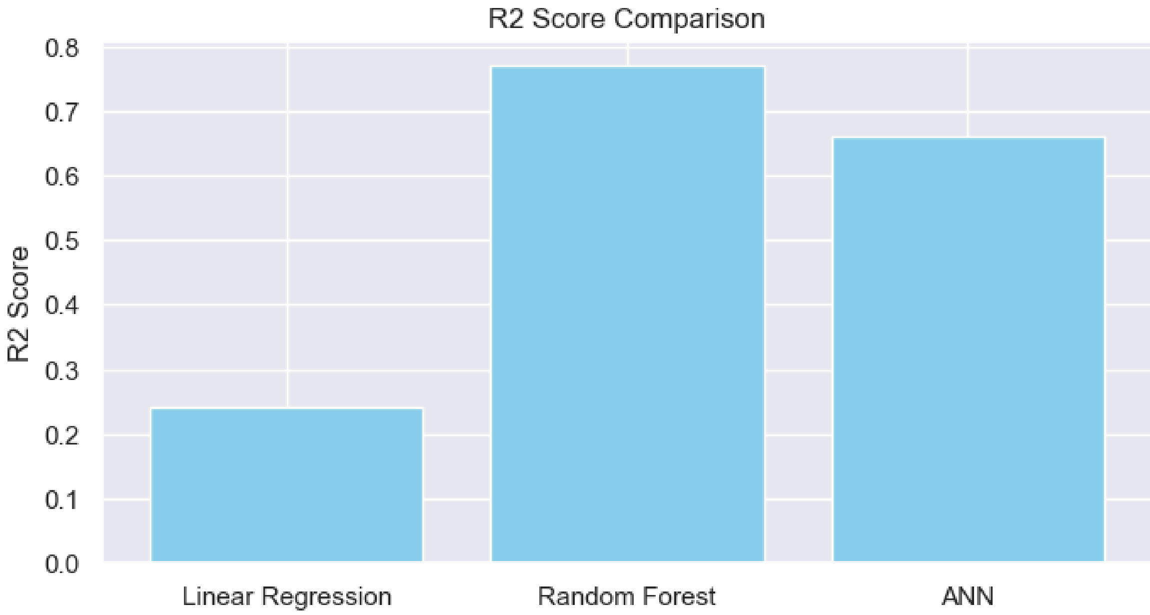
```
y_pred= ann.predict(X_test)
score_ann = r2_score(y_test,y_pred)
print(f'R2 Score with ANN = {score_ann:.2f}')
```

```
101/101 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
R2 Score with ANN = 0.64
```

R2 score Comparison:

In [101…

```
plt.figure(figsize=(8,4))
models = ['Linear Regression', 'Random Forest', 'ANN']
scores = [0.24, 0.77, 0.66]
plt.bar(models,scores, color='skyblue')
plt.title('R2 Score Comparison')
plt.ylabel('R2 Score')
plt.show()
```



Summary

- Random Forest performed best with R2 = 0.77
- The performance of the ANN was okay, and could be improved
- Linear Regression underperformed, likely due to data complexity
- Futuer imporovement: hyperparameter tuning, deep model enhancements

Thank You