DOWNLOAD LIBRARIES

In [1]:
```
pip install pandoc
```

```
Requirement already satisfied: pandoc in c:\users\nancy\anaconda3\lib\site-packag
es (2.4)
Requirement already satisfied: plumbum in c:\users\nancy\anaconda3\lib\site-packa
ges (from pandoc) (1.9.0)
Requirement already satisfied: ply in c:\users\nancy\anaconda3\lib\site-packages
(from pandoc) (3.11)
Requirement already satisfied: pywin32 in c:\users\nancy\anaconda3\lib\site-packa
ges (from plumbum->pandoc) (305.1)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
from IPython import get_ipython
from IPython.display import display
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pickle
import joblib
from datetime import datetime,timedelta
```

LOAD DATA

In [3]:
```
df= pd.read_excel("Data_CW2.xlsx")
```

In [4]:
```
df.head()
```

Out[4]:

	Sale ID	sale date	Model age	proximity to urban centres	number of dealerships nearby	vechicle sale price
0	1	2013.650	38.6	265.347718	6	41.98014
1	2	2012.350	19.5	4077.055125	1	29.02716
2	3	2012.918	20.9	937.831933	5	58.83462
3	4	2013.000	16.9	179.732757	3	51.65586
4	5	2013.416	32.5	190.054496	7	66.32550

DATA CLEANING

In [5]:
```
def decimal_year_date(decimal_year):
    year = int(decimal_year)
    start = datetime(year, 1, 1)
    end= datetime(year+1, 1, 1)
    days_in_year = (end - start).days
    frac = decimal_year - year
    return start +timedelta(days=int(frac*days_in_year))

df["sale date"] = df["sale date"].apply(decimal_year_date)
```

In [6]:
```
df
```

Out[6]:

	Sale ID	sale date	Model age	proximity to urban centres	number of dealerships nearby	vechicle sale price
0	1	2013-08-26	38.6	265.347718	6	41.98014
1	2	2012-05-08	19.5	4077.055125	1	29.02716
2	3	2012-12-01	20.9	937.831933	5	58.83462
3	4	2013-01-01	16.9	179.732757	3	51.65586
4	5	2013-06-01	32.5	190.054496	7	66.32550
...	...	...	...	...	...	...
434	435	2013-01-01	17.7	3964.985322	1	24.03324
435	436	2012-09-01	9.6	86.395962	10	78.03000
436	437	2013-04-02	22.8	378.404331	8	63.36036
437	438	2013-01-01	12.1	100.343752	6	81.93150
438	439	2013-07-02	10.5	86.395962	10	99.72234

439 rows × 6 columns

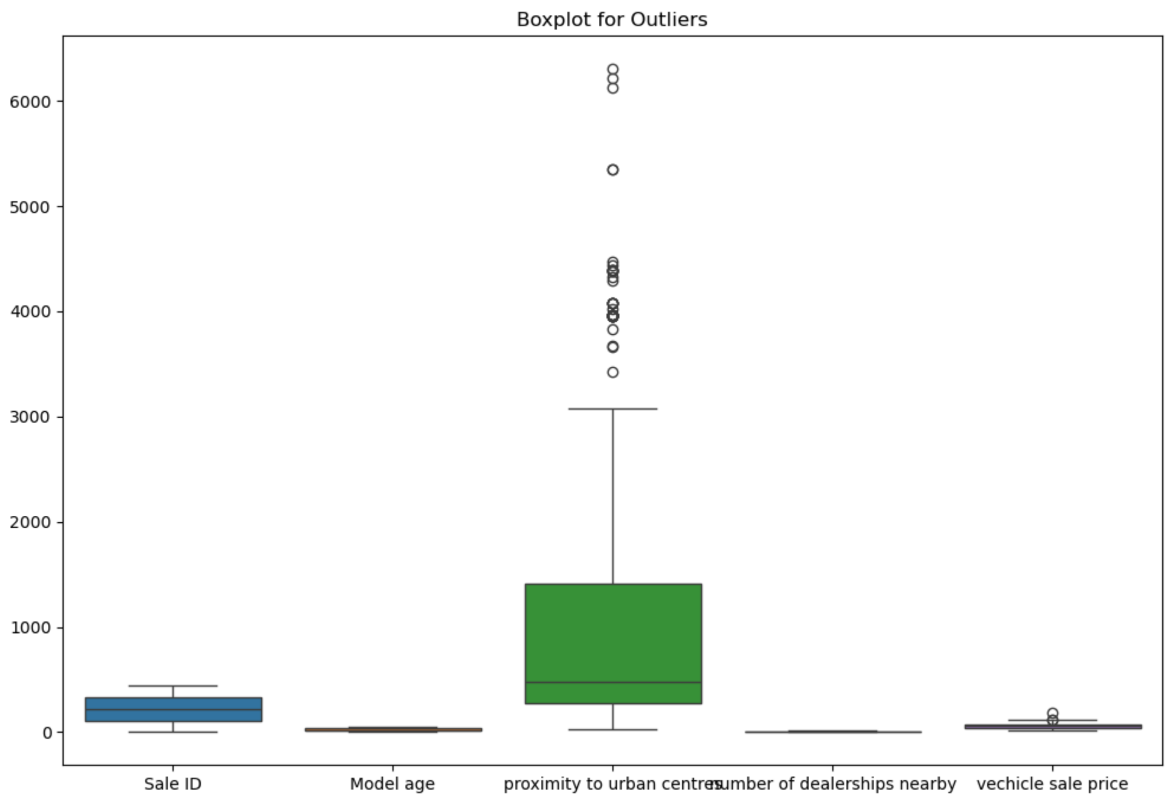
In [7]:
```
# Descriptive statistics for numerical features
df.describe()
```

Out[7]:

	Sale ID	sale date	Model age	proximity to urban centres	number of dealerships nearby	vechicle sale price
count	439.00000	439	439.000000	439.000000	439.000000	439.000000
mean	220.00000	2013-02-22 03:49:36.765375744	21.870843	1051.552291	5.088838	59.002077
min	1.00000	2012-05-08 00:00:00	4.000000	21.221056	1.000000	11.860560
25%	110.50000	2012-12-01 00:00:00	13.050000	279.636295	2.000000	42.916500
50%	220.00000	2013-03-02 00:00:00	20.200000	476.800111	5.000000	59.458860
75%	329.50000	2013-06-02 00:00:00	32.850000	1410.143812	7.000000	72.255780
max	439.00000	2013-08-26 00:00:00	47.800000	6302.896251	11.000000	183.370500
std	126.87264	NaN	11.443429	1227.734307	2.937529	21.113341

In [8]:

```
plt.figure(figsize=(12,8))
sns.boxplot(df)
plt.title('Boxplot for Outliers')

plt.show()
```



In [9]:

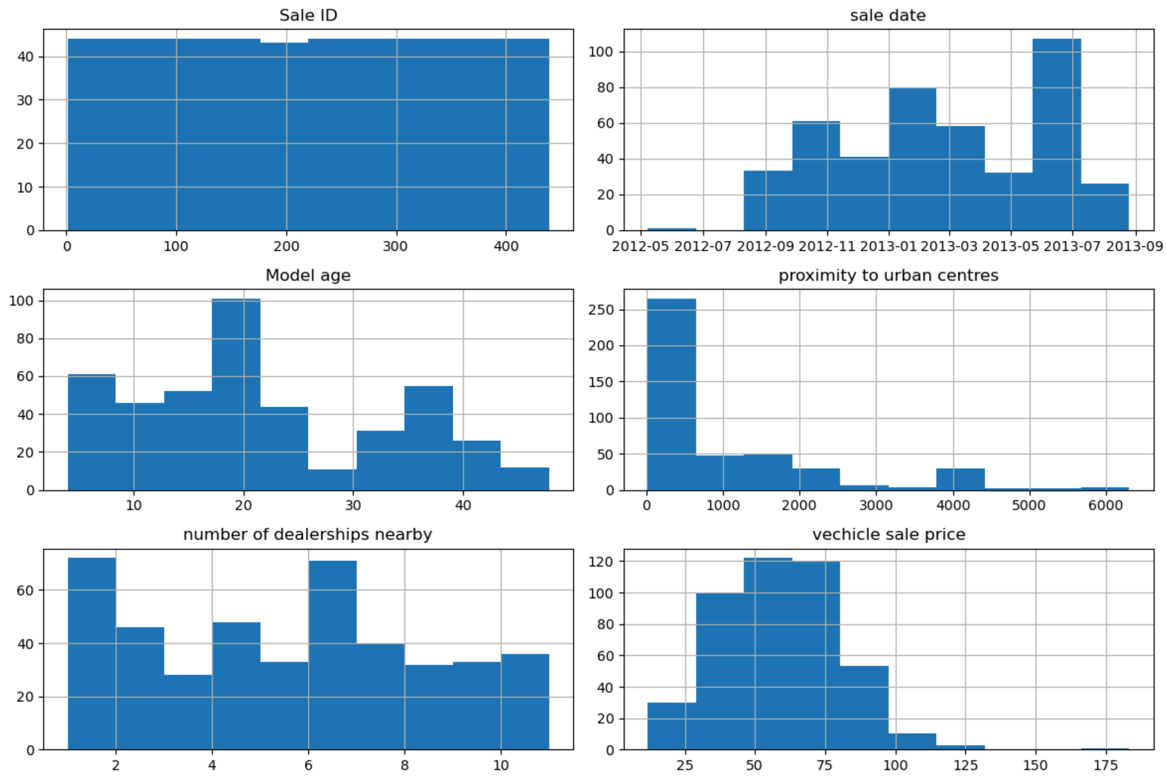
```
# Examine data types
print(df.dtypes)
```

```
Sale ID                                int64
sale date                     datetime64[ns]
Model age                            float64
proximity to urban centres           float64
number of dealerships nearby           int64
vechicle sale price                  float64
dtype: object
```

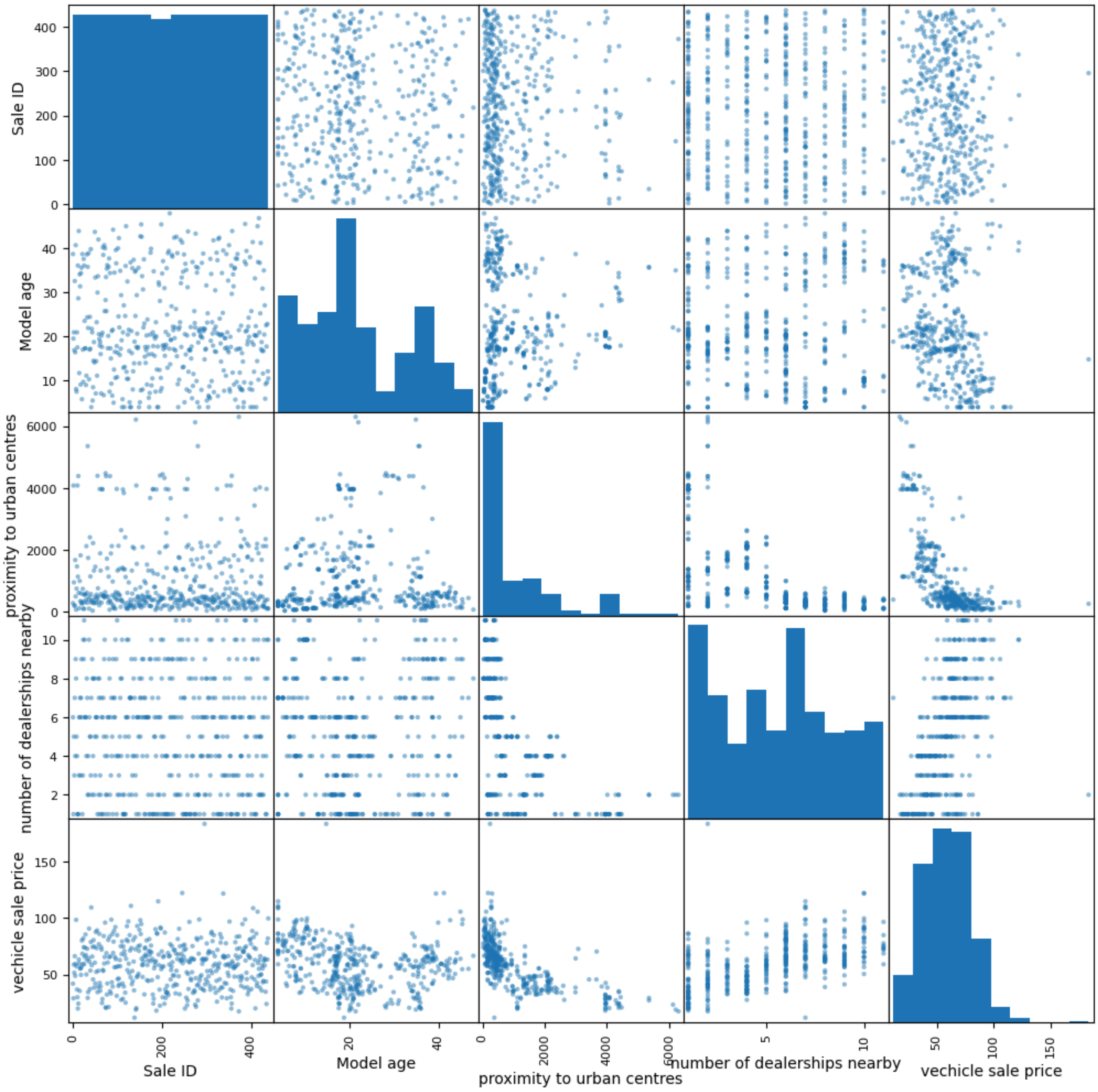
In [10]:
```
# Check for missing values
print(df.isnull().sum())
```

```
Sale ID                         0
sale date                       0
Model age                       0
proximity to urban centres      0
number of dealerships nearby    0
vechicle sale price             0
dtype: int64
```

In [11]:
```
# Histograms for numerical features
df.hist(figsize=(12, 8))
plt.tight_layout()
plt.show()
```



In [12]:
```
# Identify potential relationships (e.g., using scatter plots)
pd.plotting.scatter_matrix(df, figsize=(12, 12))
plt.show()
```



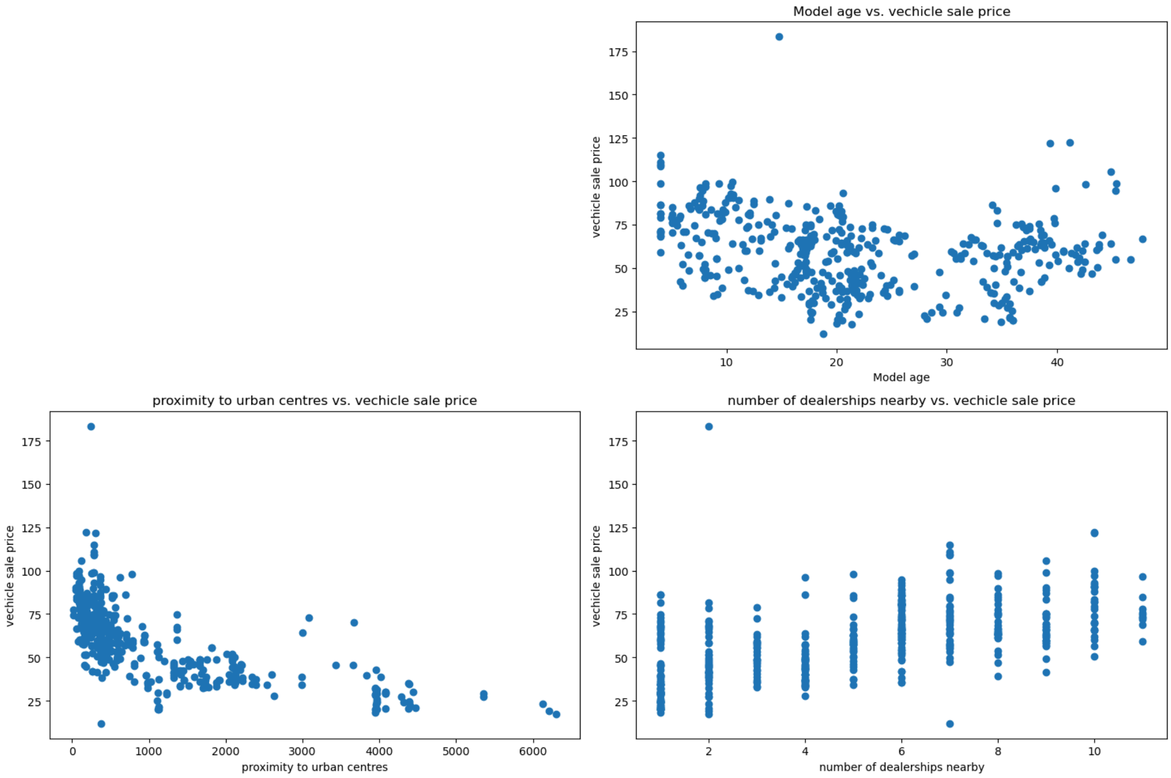
In [13]:
```
# Correlation matrix
print(df.corr())
```

```
                              Sale ID  sale date  Model age  \
Sale ID                      1.000000  -0.016074  -0.051581
sale date                   -0.016074   1.000000   0.026230
Model age                   -0.051581   0.026230   1.000000
proximity to urban centres  -0.012623   0.036083   0.017310
number of dealerships nearby -0.007645   0.016447   0.049101
vechicle sale price         -0.003723   0.100505  -0.215453

                              proximity to urban centres  \
Sale ID                                        -0.012623
sale date                                       0.036083
Model age                                       0.017310
proximity to urban centres                      1.000000
number of dealerships nearby                   -0.604435
vechicle sale price                            -0.672083

                              number of dealerships nearby  \
Sale ID                                          -0.007645
sale date                                         0.016447
Model age                                         0.049101
proximity to urban centres                       -0.604435
number of dealerships nearby                      1.000000
vechicle sale price                               0.572657

                              vechicle sale price
Sale ID                                -0.003723
sale date                               0.100505
Model age                              -0.215453
proximity to urban centres             -0.672083
number of dealerships nearby            0.572657
vechicle sale price                     1.000000
```

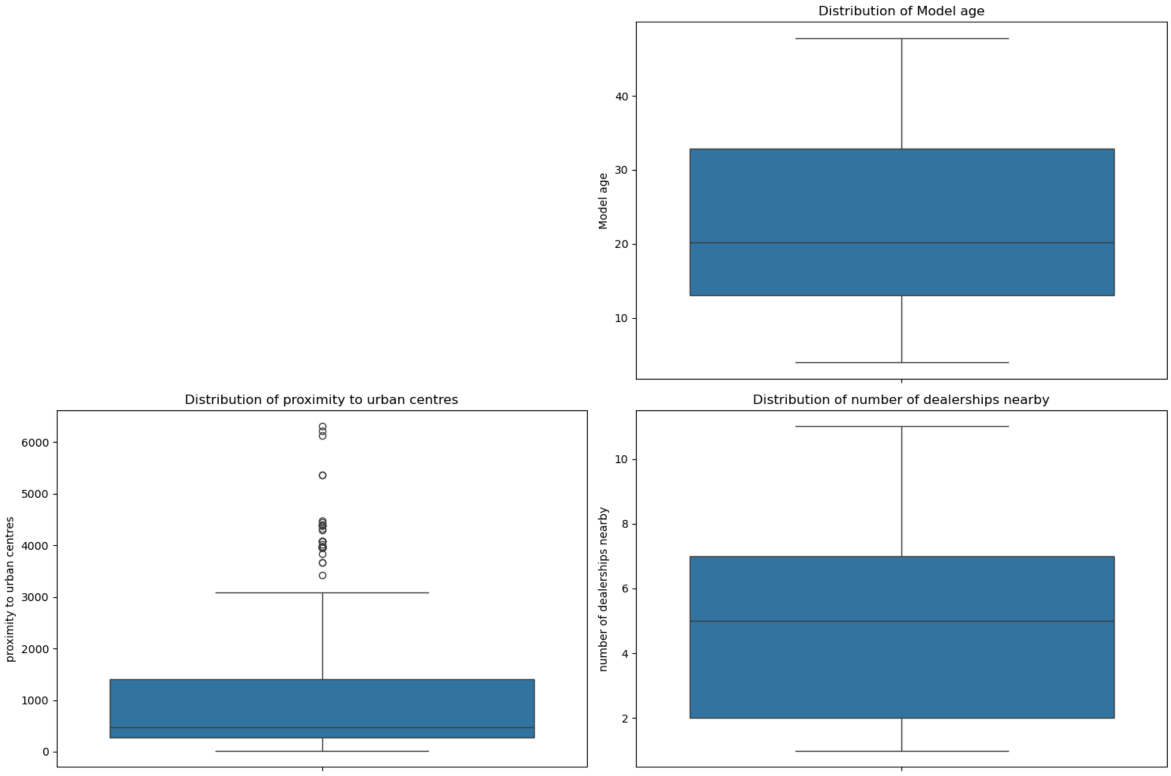
DATA VISUALISATION

In [14]:

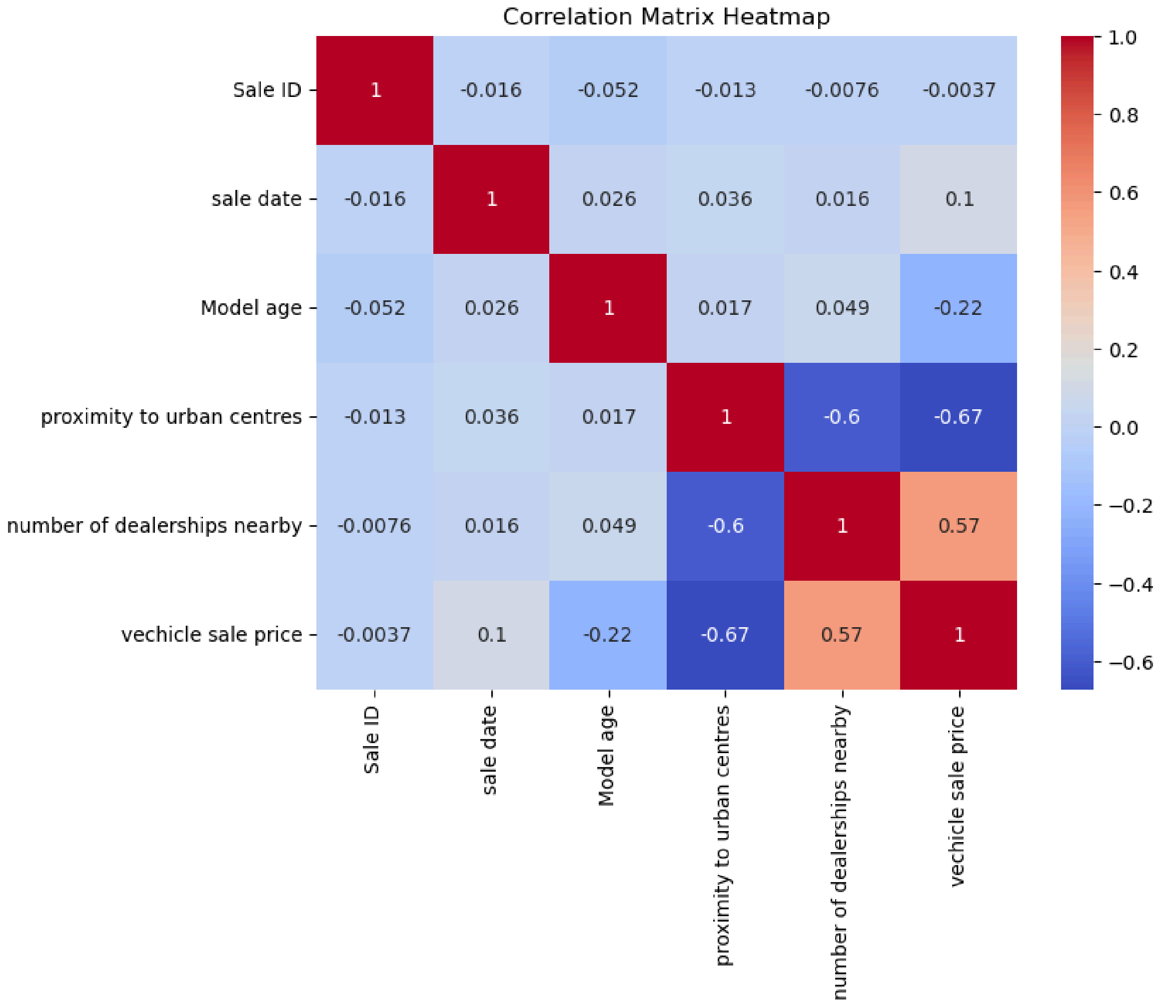
```
# Scatter plots
plt.figure(figsize=(15, 10))
for i, col in enumerate(df.columns):
    if col != 'vechicle sale price' and col != 'Sale ID' and col != 'sale date':
        plt.subplot(2, 2, i)
        plt.scatter(df[col], df['vechicle sale price'])
        plt.xlabel(col)
        plt.ylabel('vechicle sale price')
        plt.title(f'{col} vs. vechicle sale price')
plt.tight_layout()
plt.show()
```



In [15]:
```
# Box plots
plt.figure(figsize=(15, 10))
for i, col in enumerate(df.columns):
    if col != 'vechicle sale price' and col != 'Sale ID' and col != 'sale date':
        plt.subplot(2, 2, i)
        sns.boxplot(y=df[col])
        plt.ylabel(col)
        plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



In [16]:
```python
# Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



QUESTIONS

In [17]:
```python
# Question 1.1 (a) Build a linear regression model

import statsmodels.api as sm

# Define your independent variables (features) and dependent variable (target)
X = df[['proximity to urban centres', 'number of dealerships nearby', 'Model age
y = df['vechicle sale price']
```

In [18]:
```python
# Add a constant term for the intercept
X = sm.add_constant(X)
```

In [19]:
```python
# Build and fit the linear regression model
model = sm.OLS(y, X).fit()
```

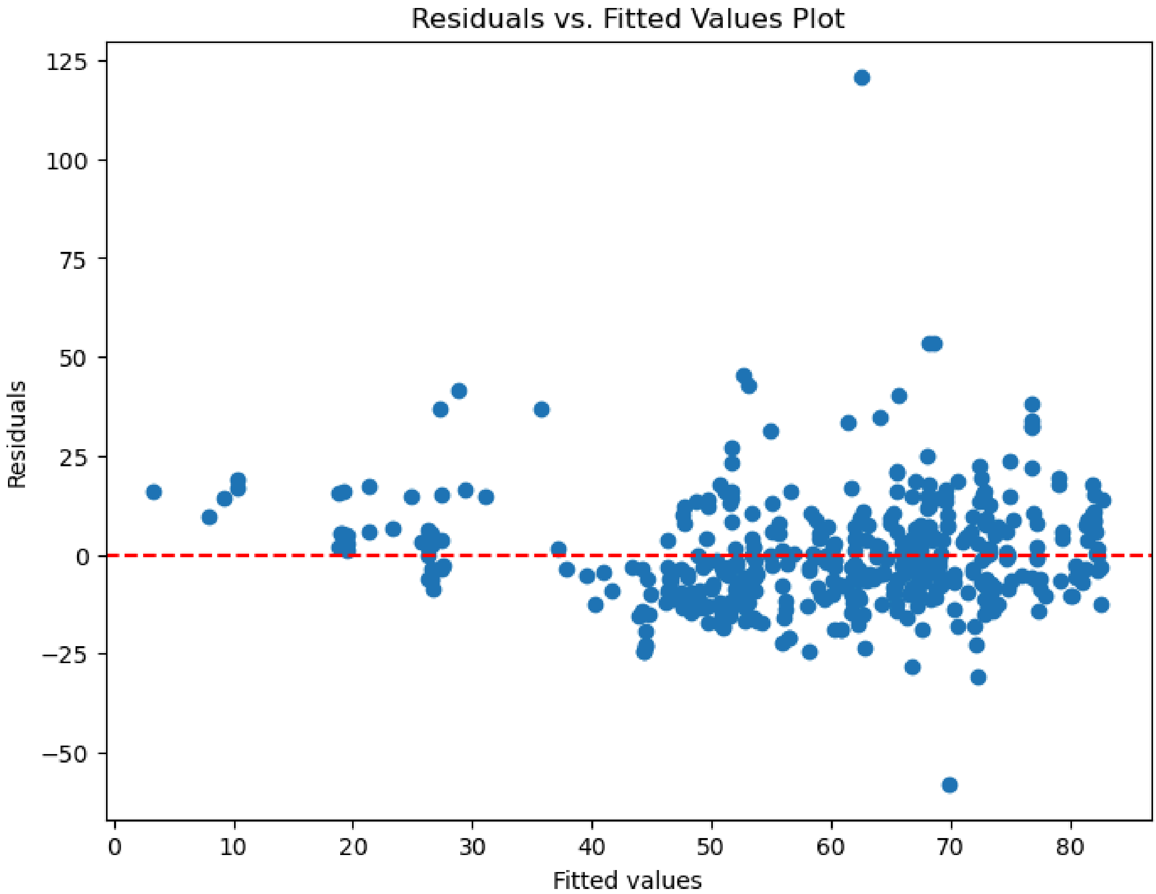
In [20]:
```python
# Print the regression summary
print(model.summary())
```

```
                               OLS Regression Results
================================================================================
Dep. Variable:     vechicle sale price   R-squared:                       0.544
Model:                             OLS   Adj. R-squared:                  0.541
Method:                  Least Squares   F-statistic:                     172.8
Date:                 Fri, 16 May 2025   Prob (F-statistic):           9.15e-74
Time:                         21:07:00   Log-Likelihood:                -1789.1
No. Observations:                  439   AIC:                             3586.
Df Residuals:                      435   BIC:                             3602.
Df Model:                            3
Covariance Type:             nonrobust
==================================================================================
===============
                                coef    std err          t      P>|t|      [0.
025      0.975]
----------------------------------------------------------------------------------
---------------
const                        66.5522      2.424     27.458      0.000      61.
789      71.316
proximity to urban centres   -0.0086      0.001    -12.210      0.000      -0.
010      -0.007
number of dealerships nearby  2.0337      0.293      6.941      0.000       1.
458       2.610
Model age                    -0.4073      0.060     -6.796      0.000      -0.
525      -0.289
==============================================================================
Omnibus:                      230.725   Durbin-Watson:                   2.108
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3156.853
Skew:                           1.919   Prob(JB):                         0.00
Kurtosis:                      15.564   Cond. No.                     5.76e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.76e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [21]:
```
# Question 1.2 (b) Evidence against heteroskedasticity

# Get the residuals
residuals = model.resid
```

In [22]:
```
# Get the fitted values
fitted_values = model.fittedvalues
```

In [23]:
```
# Plot residuals vs. fitted values
plt.figure(figsize=(8, 6))
plt.scatter(fitted_values, residuals)
plt.xlabel("Fitted values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Fitted Values Plot")
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [24]:
```
# Question 1.2 (b) Evidence against multicollinearity using VIF

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [25]:
```
# Calculate VIF for each predictor
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                   for i in range(len(X.columns))]
print("\nVIF Data:")
print(vif_data)
```

```
VIF Data:
                       feature        VIF
0                        const  12.593882
1    proximity to urban centres   1.581166
2  number of dealerships nearby   1.584512
3                    Model age   1.005925
```

In [ ]: