jupyter Machine Learning Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[377]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.impute import SimpleImputer
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import Lasso
from sklearn.feature_selection import RFE
import statsmodels.api as sm
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

[379]:
```
import warnings
warnings.filterwarnings("ignore")
```

[17]:
```
print(df.describe())
```

```
            car_ID   symboling   wheelbase   carlength    carwidth   carheight  \
count   205.000000  205.000000  205.000000  205.000000  205.000000  205.000000
mean    103.000000    0.834146   98.756585  174.049268   65.907805   53.724878
std      59.322565    1.245307    6.021776   12.337289    2.145204    2.443522
min       1.000000   -2.000000   86.600000  141.100000   60.300000   47.800000
25%      52.000000    0.000000   94.500000  166.300000   64.100000   52.000000
50%     103.000000    1.000000   97.000000  173.200000   65.500000   54.100000
75%     154.000000    2.000000  102.400000  183.100000   66.900000   55.500000
max     205.000000    3.000000  120.900000  208.100000   72.300000   59.800000

          curbweight  enginesize   boreratio      stroke  compressionratio  \
count     205.000000  205.000000  205.000000  205.000000        205.000000
mean     2555.565854  126.907317    3.329756    3.255415         10.142537
std       520.680204   41.642693    0.270844    0.313597          3.972040
min      1488.000000   61.000000    2.540000    2.070000          7.000000
25%      2145.000000   97.000000    3.150000    3.110000          8.600000
50%      2414.000000  120.000000    3.310000    3.290000          9.000000
75%      2935.000000  141.000000    3.580000    3.410000          9.400000
max      4066.000000  326.000000    3.940000    4.170000         23.000000

        horsepower      peakrpm     citympg  highwaympg         price
count   205.000000   205.000000  205.000000  205.000000    205.000000
mean    104.117073  5125.121951   25.219512   30.751220  13276.710571
std      39.544167   476.985643    6.542142    6.886443   7988.852332
min      48.000000  4150.000000   13.000000   16.000000   5118.000000
25%      70.000000  4800.000000   19.000000   25.000000   7788.000000
50%      95.000000  5200.000000   24.000000   30.000000  10295.000000
75%     116.000000  5500.000000   30.000000   34.000000  16503.000000
max     288.000000  6600.000000   49.000000   54.000000  45400.000000
```

[19]:
```
df.isnull().sum()
```

[19]:
```
car_ID            0
symboling         0
CarName           0
fueltype          0
aspiration        0
doornumber        0
carbody           0
drivewheel        0
```

[21]:
```
df.duplicated().sum()
```

[21]:
```
0
```

[23]:
```
null_counts = df.isnull().sum()
print(null_counts)
```

```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

[9]:
```
df = df.drop("car_ID", axis=1)
df.describe()
```

[9]:

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	city
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.00

[9]:
```
df = df.drop("car_ID", axis=1)
df.describe()
```

[9]:

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	city
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.00
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.142537	104.117073	5125.121951	25.21
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.972040	39.544167	476.985643	6.54
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000	4150.000000	13.00
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.000000	4800.000000	19.00
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000	5200.000000	24.00
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.400000	116.000000	5500.000000	30.00
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.000000	6600.000000	49.00

[11]:
```
df.tail()
```

[11]:

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	enginesize	fuelsystem	boreratio	stroke
200	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	188.8	...	141	mpfi	3.78	3.15
201	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	188.8	...	141	mpfi	3.78	3.15
202	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	188.8	...	173	mpfi	3.58	2.87
203	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	188.8	...	145	idi	3.01	3.40
204	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	188.8	...	141	mpfi	3.78	3.15

5 rows × 25 columns

[170]:
```
label_encoder = LabelEncoder()
df["CarName"] = label_encoder.fit_transform(df["CarName"])
df["fueltype"] = label_encoder.fit_transform(df["fueltype"])
df["aspiration"] = label_encoder.fit_transform(df["aspiration"])
df["doornumber"] = label_encoder.fit_transform(df["doornumber"])
df["carbody"] = label_encoder.fit_transform(df["carbody"])
df["drivewheel"] = label_encoder.fit_transform(df["drivewheel"])
df["enginelocation"] = label_encoder.fit_transform(df["enginelocation"])
df["enginetype"] = label_encoder.fit_transform(df["enginetype"])
df["cylindernumber"] = label_encoder.fit_transform(df["cylindernumber"])
df["fuelsystem"] = label_encoder.fit_transform(df["fuelsystem"])
df.head()
```

[170]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	com
0	1	3	2	1	0	1	0	2	0	88.6	...	130	5	3.47	2.68	
1	2	3	3	1	0	1	0	2	0	88.6	...	130	5	3.47	2.68	
2	3	1	1	1	0	1	2	2	0	94.5	...	152	5	2.68	3.47	
3	4	2	4	1	0	0	3	1	0	99.8	...	109	5	3.19	3.40	
4	5	2	5	1	0	0	3	0	0	99.4	...	136	5	3.19	3.40	

5 rows × 26 columns

[172]:
```
df.describe().round(2)
```

[172]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke
count	205.00	205.00	205.00	205.0	205.00	205.00	205.00	205.00	205.00	205.00	...	205.00	205.00	205.00	205.00
mean	103.00	0.83	77.21	0.9	0.18	0.44	2.61	1.33	0.01	98.76	...	126.91	3.25	3.33	3.26
std	59.32	1.25	41.01	0.3	0.39	0.50	0.86	0.56	0.12	6.02	...	41.64	2.01	0.27	0.31
min	1.00	-2.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	86.60	...	61.00	0.00	2.54	2.07
25%	52.00	0.00	44.00	1.0	0.00	0.00	2.00	1.00	0.00	94.50	...	97.00	1.00	3.15	3.11

[331]:
```
label_encoder = LabelEncoder()
df["CarName"] = label_encoder.fit_transform(df["CarName"])
df["fueltype"] = label_encoder.fit_transform(df["fueltype"])
df["aspiration"] = label_encoder.fit_transform(df["aspiration"])
df["doornumber"] = label_encoder.fit_transform(df["doornumber"])
df["carbody"] = label_encoder.fit_transform(df["carbody"])
df["drivewheel"] = label_encoder.fit_transform(df["drivewheel"])
df["enginelocation"] = label_encoder.fit_transform(df["enginelocation"])
df["enginetype"] = label_encoder.fit_transform(df["enginetype"])
df["cylindernumber"] = label_encoder.fit_transform(df["cylindernumber"])
df["fuelsystem"] = label_encoder.fit_transform(df["fuelsystem"])
df.head()
```

[331]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	com
0	1	3	2	1	0	1	0	2	0	88.6	...	130	5	3.47	2.68	
1	2	3	3	1	0	1	0	2	0	88.6	...	130	5	3.47	2.68	
2	3	1	1	1	0	1	2	2	0	94.5	...	152	5	2.68	3.47	
3	4	2	4	1	0	0	3	1	0	99.8	...	109	5	3.19	3.40	
4	5	2	5	1	0	0	3	0	0	99.4	...	136	5	3.19	3.40	

5 rows × 26 columns

[ ]:

[329]:
```
strong_correlation_cols = [col for col in corr_matrix.columns if abs(corr_matrix.loc["price", col]) >= 0.09 and col != "pric"]

# Select features (X) and target (y)
X = df[strong_correlation_cols]
y = df["price"]

strong_correlation_cols
```

[329]:
```
['car_ID',
 'CarName',
```

[229]:
```
x = df.drop(columns = "price")
y = df["price"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(164, 25) (164,) (41, 25) (41,)
```

[245]:
```
imputer = SimpleImputer(strategy = "mean")
x_train_imputed = imputer.fit_transform(x_train)
x_test_imputed = imputer.transform(x_test)

lin_reg =  LinearRegression()
lin_reg.fit(x_train_imputed, y_train)
```

[245]:
```
LinearRegression()
```

[249]:
```
dec_tree_reg =  DecisionTreeRegressor()
dec_tree_reg.fit(x_train_imputed, y_train)
```

[249]:
```
DecisionTreeRegressor()
```

[261]:
```
ran_for_reg = RandomForestRegressor()
ran_for_reg.fit(x_train_imputed, y_train)
```

[261]:
```
RandomForestRegressor()
```

[263]:
```
gra_bost_reg = GradientBoostingRegressor()
gra_bost_reg.fit(x_train_imputed, y_train)
```

[263]:
```
GradientBoostingRegressor
```

```
svr.fit(x_train_imputed, y_train)
```

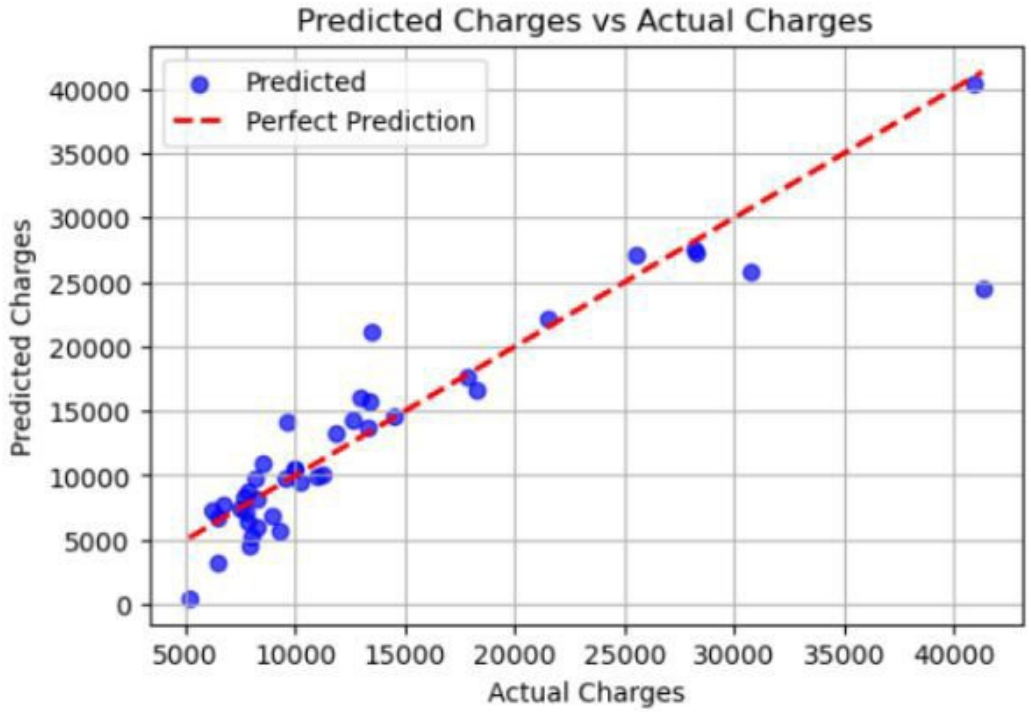
[265]:

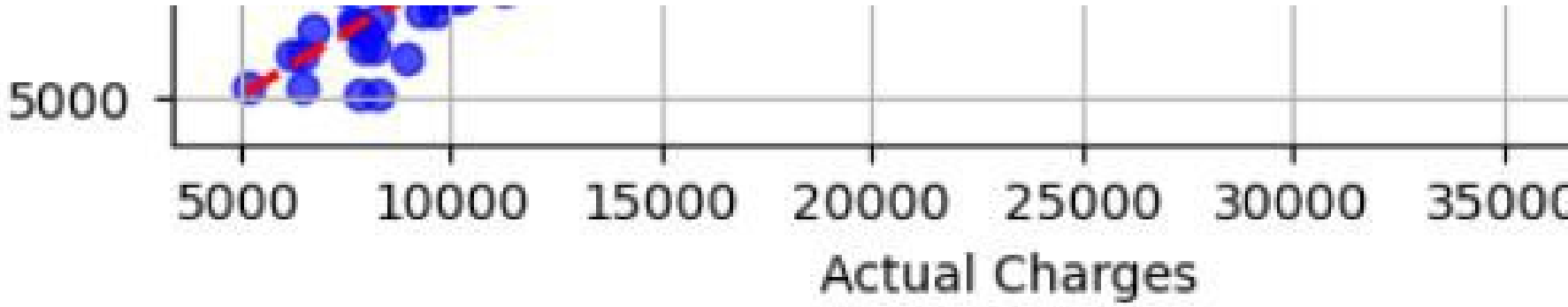
SVR

SVR()

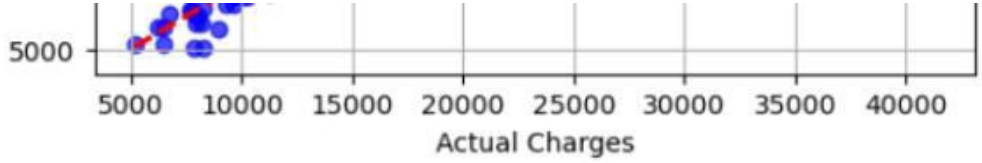
[297]:

```
lin_reg_y_pred = lin_reg.predict(x_test_imputed)

plt.figure(figsize=(6, 4))
plt.scatter(y_test,lin_reg_y_pred, color='blue', alpha=0.7, label='Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', linewidth=2, label='Perfect Prediction')
plt.title('Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.legend()
plt.grid(True)
plt.show()
```

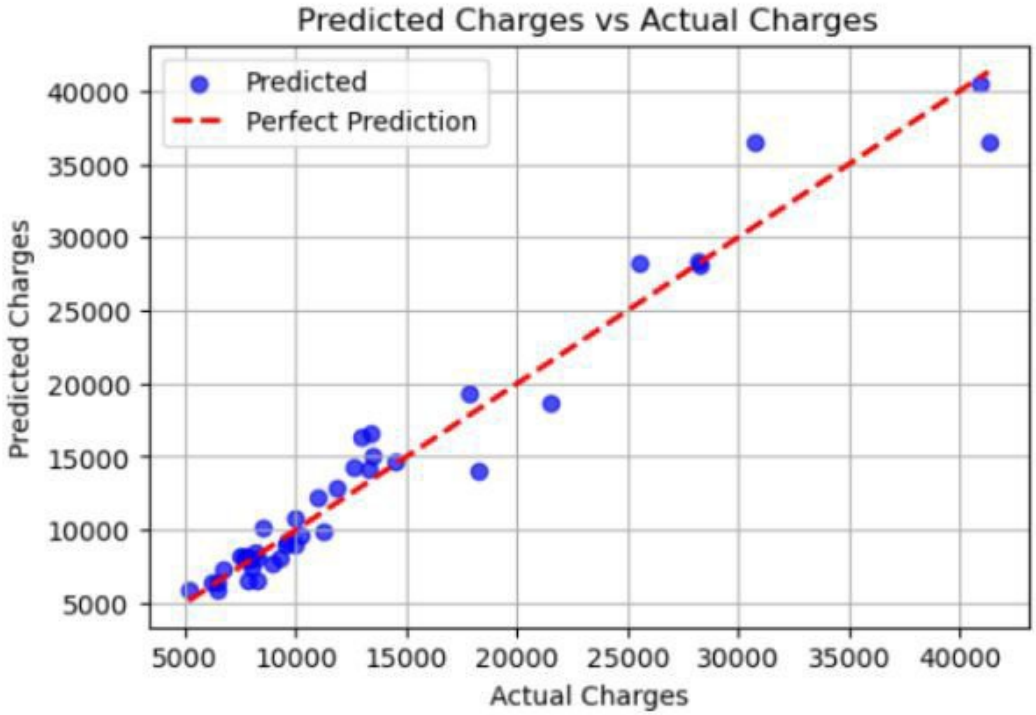


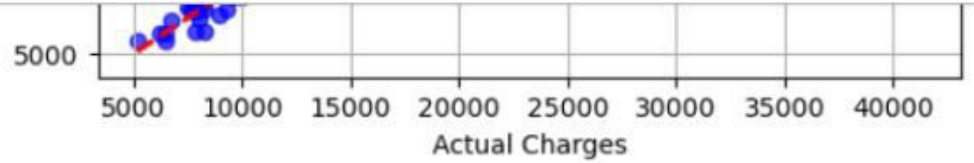


[301]:
```
ran_for_reg_y_pred = ran_for_reg.predict(x_test_imputed)

plt.figure(figsize=(6, 4))
plt.scatter(y_test, ran_for_reg_y_pred, color='blue', alpha=0.7,
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max(
plt.title('Predicted Charges vs Actual Charges')
plt.ylabel('Actual Charges')
```

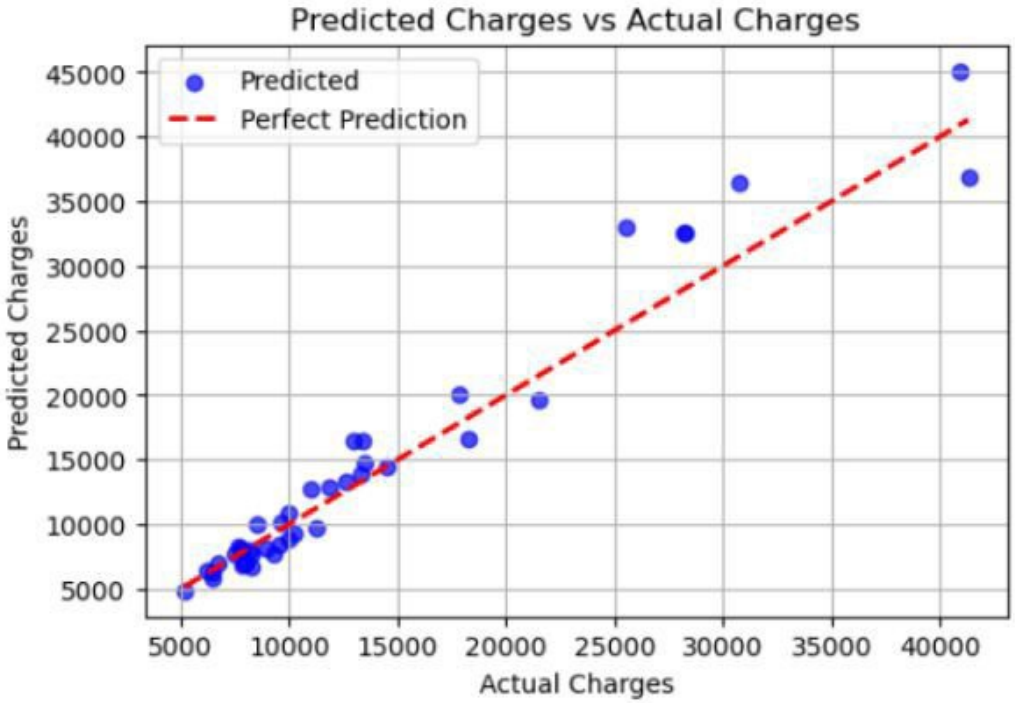


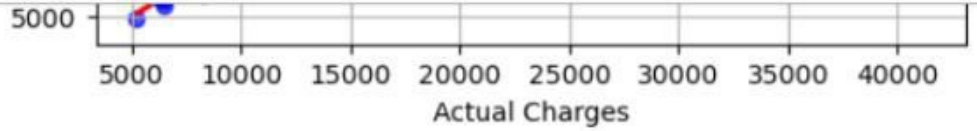
[301]:
```
ran_for_reg_y_pred = ran_for_reg.predict(x_test_imputed)

plt.figure(figsize=(6, 4))
plt.scatter(y_test, ran_for_reg_y_pred, color='blue', alpha=0.7, label='Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', linewidth=2, label='Perfect Prediction')
plt.title('Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.legend()
plt.grid(True)
plt.show()
```



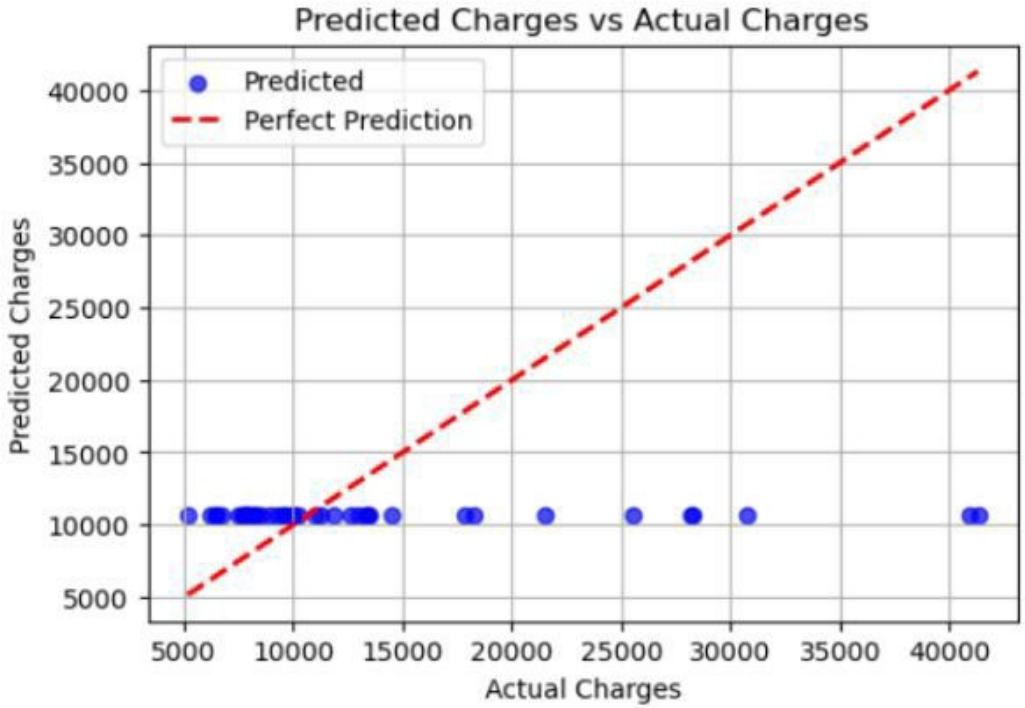


[303]:
```
gra_bost_reg_y_pred = gra_bost_reg.predict(x_test_imputed)

plt.figure(figsize=(6, 4))
plt.scatter(y_test, gra_bost_reg_y_pred, color='blue', alpha=0.7, label='Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', linewidth=2, label='Perfect Prediction')
plt.title('Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.legend()
plt.grid(True)
plt.show()
```



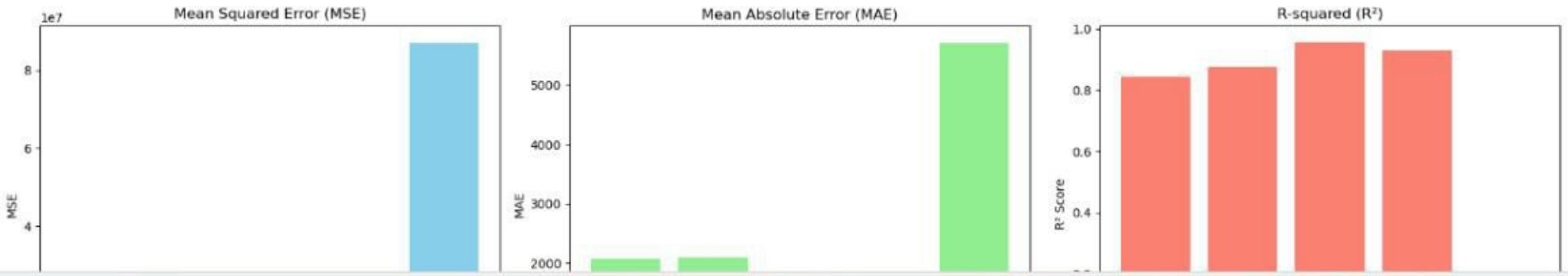


[307]:
```
svr_y_pred = svr.predict(x_test_imputed)


plt.figure(figsize=(6, 4))
plt.scatter(y_test, svr_y_pred, color='blue', alpha=0.7, label='Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', linewidth=2, label='Perfect Prediction')
plt.title('Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.legend()
plt.grid(True)
plt.show()
```



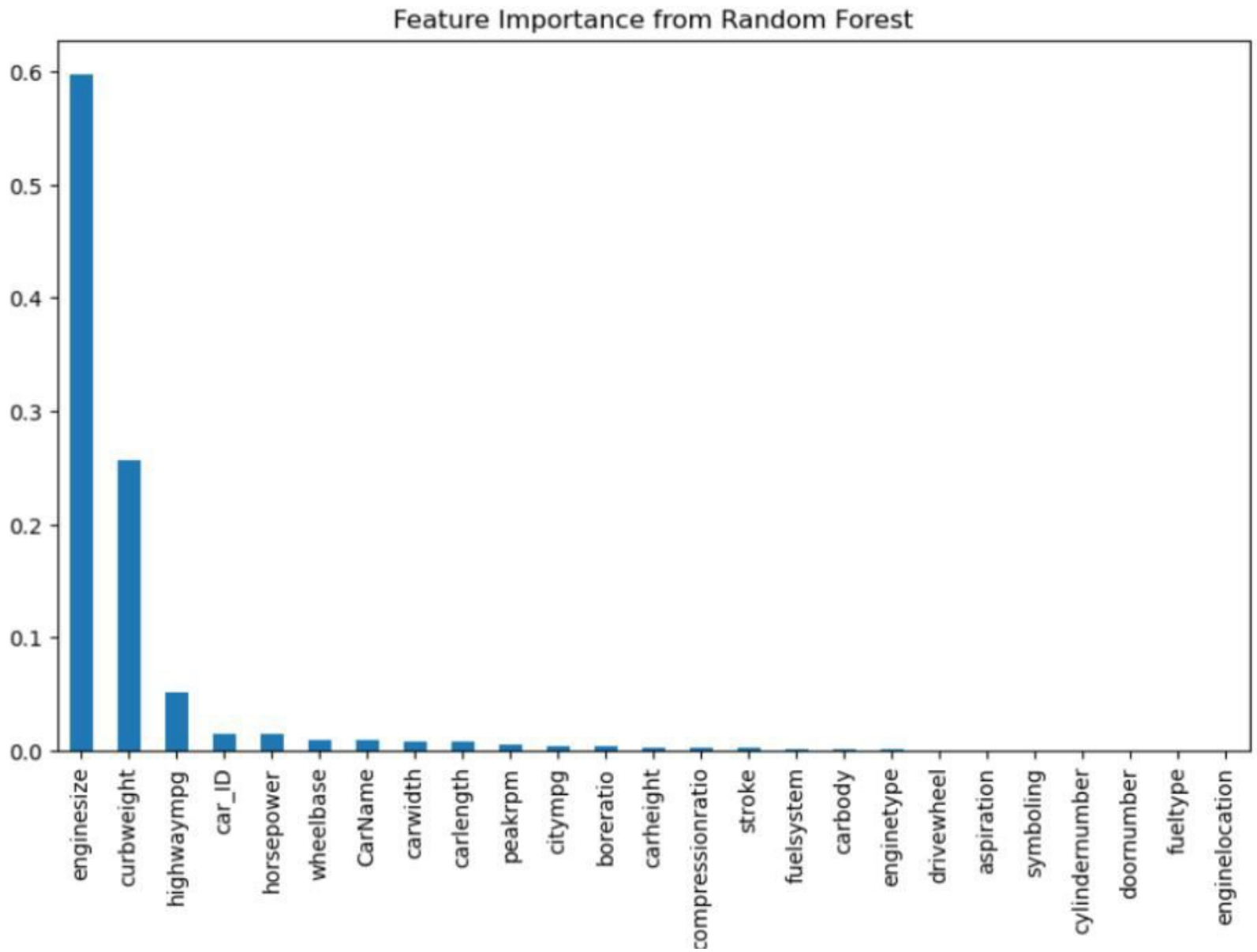
[311]:
```
models = ["Linear Regression", "Decision Tree", "Random Forest","(

mse_values = [lin_reg_mse, dec_tree_reg_mse, ran_for_reg_mse, gra_
mae_values = [lin_reg_mae, dec_tree_reg_mae, ran_for_reg_mae, gra_
r2_values = [lin_reg_r2, dec_tree_reg_r2, ran_for_reg_r2, gra_bost

fig, ax = plt.subplots(1, 3, figsize=(18, 5))

ax[0].bar(models, mse_values, color='skyblue')
ax[0].set_title('Mean Squared Error (MSE)')
ax[0].set_ylabel('MSE')

ax[1].bar(models, mae_values, color='lightgreen')
ax[1].set_title('Mean Absolute Error (MAE)')
```

[311]:
```python
models = ["Linear Regression", "Decision Tree", "Random Forest","Gradient Boosting", "SVR"]

mse_values = [lin_reg_mse, dec_tree_reg_mse, ran_for_reg_mse, gra_bost_reg_mse, svr_mse]
mae_values = [lin_reg_mae, dec_tree_reg_mae, ran_for_reg_mae, gra_bost_reg_mae, svr_mae]
r2_values = [lin_reg_r2, dec_tree_reg_r2, ran_for_reg_r2, gra_bost_reg_r2, svr_r2]

fig, ax = plt.subplots(1, 3, figsize=(18, 5))

ax[0].bar(models, mse_values, color='skyblue')
ax[0].set_title('Mean Squared Error (MSE)')
ax[0].set_ylabel('MSE')

ax[1].bar(models, mae_values, color='lightgreen')
ax[1].set_title('Mean Absolute Error (MAE)')
ax[1].set_ylabel('MAE')

ax[2].bar(models, r2_values, color='salmon')
ax[2].set_title('R-squared (R²)')
ax[2].set_ylabel('R² Score')

for axis in ax:
    axis.set_xlabel('Models')
    axis.set_xticks(np.arange(len(models)))
    axis.set_xticklabels(models)

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

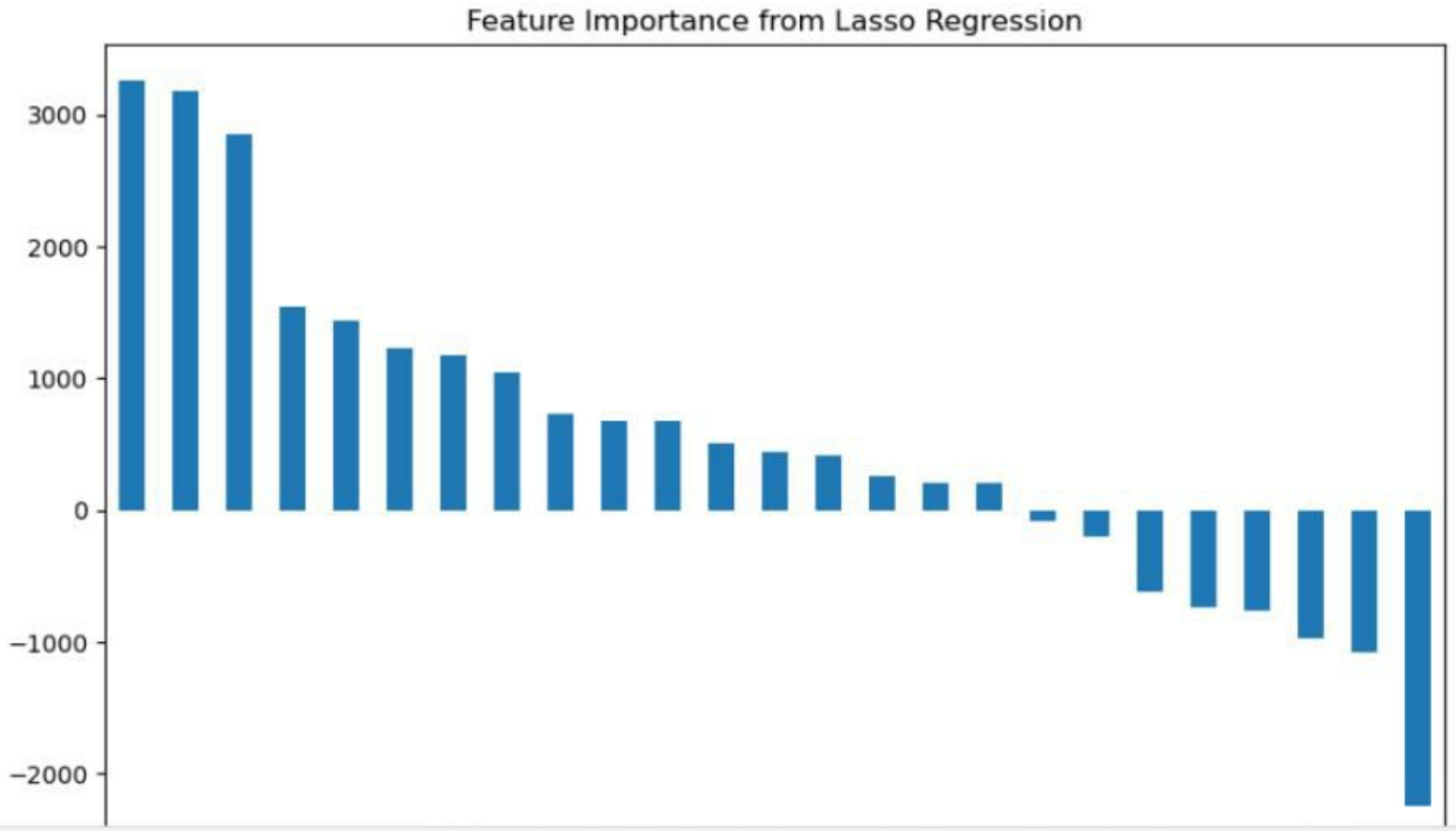


[375]:
```
corr_matrix = df.corr()
plt.figure(figsize=(25, 15))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", vmin=-1, vmax=1)
plt.title("Correlation Matrix Heatmap")
plt.show()
```

[339]:
```
feature_importances = pd.Series(ran_for_reg.feature_importances_, index=x_train.columns)
feature_importances.sort_values(ascending=False).plot(kind='bar', figsize=(10,6))
plt.title('Feature Importance from Random Forest')
plt.show()
```

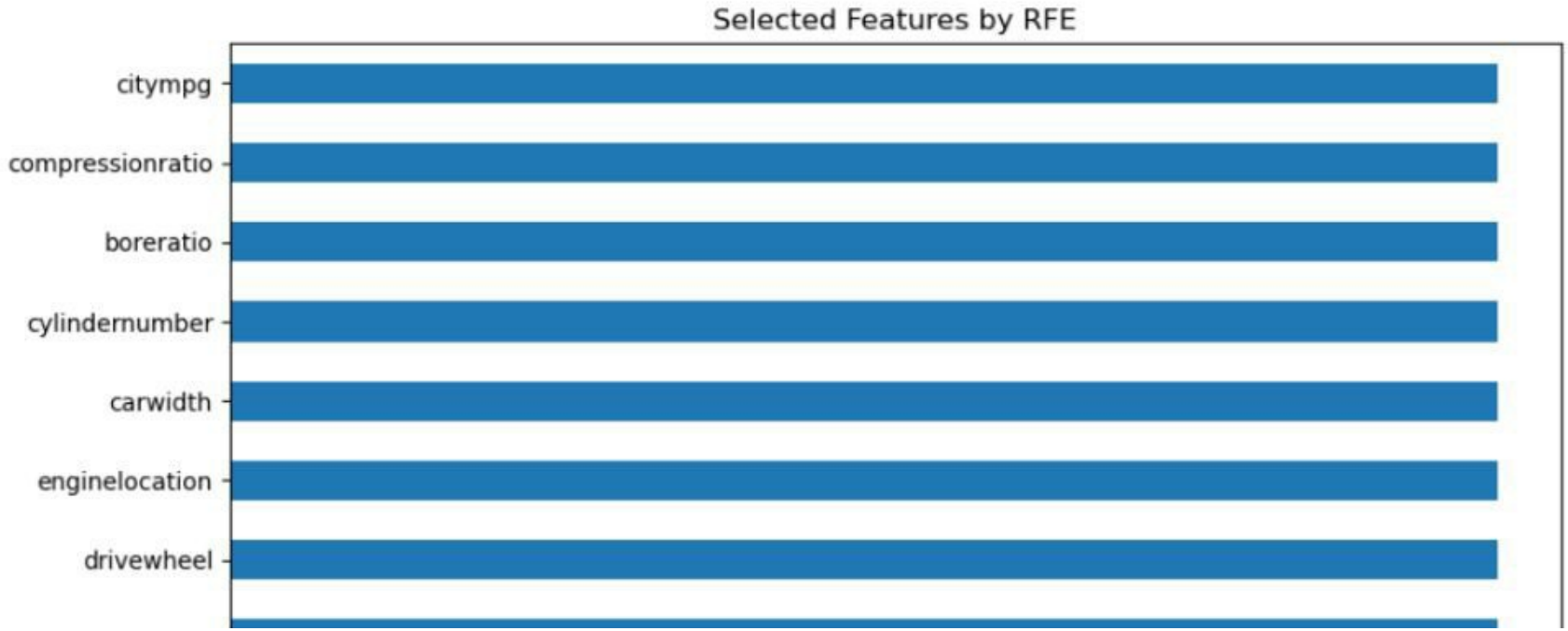


[349]:
```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
lasso = Lasso(alpha=0.01)

lasso.fit(x_train_scaled, y_train)
lasso_coef = pd.Series(lasso.coef_, index=x_train.columns)

lasso_coef.sort_values(ascending=False).plot(kind='bar', figsize=(10,6))
plt.title('Feature Importance from Lasso Regression')
plt.show()
```



```
dtype: float64
```

[361]:
```
rfe = RFE(lin_reg, n_features_to_select=10)
rfe.fit(x_train, y_train)

ranking = pd.Series(rfe.ranking_, index=x_train.columns)

ranking[ranking == 1].plot(kind='barh', figsize=(10,6))
plt.title('Selected Features by RFE')
plt.show()
```



[369]:
```
x_train_const = sm.add_constant(x_train)

ols_model = sm.OLS(y_train, x_train_const).fit()
```

[367]:
```
print(ols_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.910
Model:                            OLS   Adj. R-squared:                  0.894
Method:                 Least Squares   F-statistic:                     56.06
Date:                Wed, 23 Oct 2024   Prob (F-statistic):           5.03e-60
Time:                        02:17:47   Log-Likelihood:                -1503.0
No. Observations:                 164   AIC:                             3058.
Df Residuals:                     138   BIC:                             3139.
Df Model:                          25
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const        -7.296e+04   1.83e+04     -3.989      0.000   -1.09e+05   -3.68e+04
car_ID          19.6695     17.206      1.143      0.255     -14.353      53.692
symboling      166.6995    278.895      0.598      0.551    -384.761     718.160
CarName         -55.1231     25.203     -2.187      0.030    -104.958      -5.289
fueltype       9878.1847   6746.472      1.464      0.145   -3461.638    2.32e+04
aspiration     1081.8476    890.156      1.215      0.226    -678.261    2841.956
doornumber    -1539.7780    697.099     -2.209      0.029   -2918.155    -161.401
carbody       -1127.6753    394.135     -2.861      0.005   -1907.001    -348.350
drivewheel      802.6692    596.989      1.345      0.181    -377.760    1983.098
enginelocation 1.151e+04   2050.118      5.616      0.000    7459.233    1.56e+04
wheelbase       175.9950    110.709      1.590      0.114     -42.911     394.901
carlength       -61.5747     55.680     -1.106      0.271    -171.672      48.523
carwidth        580.7626    268.100      2.166      0.032      50.647    1110.879
carheight       205.2522    144.021      1.425      0.156     -79.521     490.025
curbweight        2.8120      1.638      1.717      0.088      -0.427       6.051
enginetype      235.6935    229.500      1.027      0.306    -218.099     689.486
cylindernumber -105.0100    414.633     -0.253      0.800    -924.866     714.846
```

[394]:
```
print(f"R-squared: {r2}")
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
```

```
R-squared: 0.9523037215575819
Mean Squared Error: 3765336.00124022
Mean Absolute Error: 1329.724707317073
```

[398]:
```
random_search = RandomizedSearchCV(estimator=ran_for_reg, param_distributions=param_grid, n_iter=10, cv=5, n_jobs=-1, verbose=2, scoring='r2')
random_search.fit(x_train_imputed, y_train)
print("Best Hyperparameters: ", random_search.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best Hyperparameters:  {'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': 10, 'bootstrap': True}
```