

```
In [1]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import math

df = pd.read_csv('Car_sales.csv')
df.head()
```

```
Out[1]:
```

	Manufacturer	Model	Sales_in_thousands	_year_resale_value	Vehicle_type	Price_in_thousands	Engine_size	Horsepower	Wheelbase	Width	Length	Curb_weight	Fuel_capacity	Fuel_efficienc
0	Acura	Integra	16.919	16.360	Passenger	21.50	1.8	140.0	101.2	67.3	172.4	2.639	13.2	28.
1	Acura	TL	39.384	19.875	Passenger	28.40	3.2	225.0	108.1	70.3	192.9	3.517	17.2	25.
2	Acura	CL	14.114	18.225	Passenger	NaN	3.2	225.0	106.9	70.6	192.0	3.470	17.2	26.
3	Acura	RL	8.588	29.725	Passenger	42.00	3.5	210.0	114.6	71.4	196.6	3.850	18.0	22.
4	Audi	A4	20.397	22.255	Passenger	23.99	1.8	150.0	102.6	68.2	178.0	2.998	16.4	27.

```
In [2]: df.isnull().sum()
```

```
Out[2]: Manufacturer      0
Model                    0
Sales_in_thousands      0
_year_resale_value      36
Vehicle_type             0
Price_in_thousands      2
Engine_size              1
Horsepower               1
Wheelbase                1
Width                    1
Length                   1
Curb_weight              2
Fuel_capacity            1
Fuel_efficiency          3
Latest_Launch           0
Power_perf_factor        2
dtype: int64
```

```
In [3]: len(df)
```

```
Out[3]: 157
```

```
In [4]: missing_percentage = df.isnull().mean() * 100
columns_with_missing_30 = missing_percentage[missing_percentage > 30]
print("Columns with more than 30% missing values:")
print(columns_with_missing_30)
```

```
Columns with more than 30% missing values:
Series([], dtype: float64)
```

```
In [5]: print(df['_year_resale_value'].median())
print(df['_year_resale_value'].mean())
```

```
14.18
18.07297520661157
```

```
In [6]: df['_year_resale_value'].hist(bins=5)
plt.title('Missing Value Decision Histogram')
plt.xlabel('Resale')
plt.ylabel('Frequency')
plt.show()
```



```
In [90]: df.fillna({'_year_resale_value': df['_year_resale_value'].median(), inplace=True)
df.fillna({'Price_in_thousands': df['Price_in_thousands'].median(), inplace=True)
df.fillna({'Engine_size': df['Engine_size'].median(), inplace=True)
df.fillna({'Horsepower': df['Horsepower'].median(), inplace=True)
df.fillna({'Wheelbase': df['Wheelbase'].median(), inplace=True)
df.fillna({'Width': df['Width'].median(), inplace=True)
df.fillna({'Length': df['Length'].median(), inplace=True)
df.fillna({'Curb_weight': df['Curb_weight'].median(), inplace=True)
df.fillna({'Fuel_capacity': df['Fuel_capacity'].median(), inplace=True)
df.fillna({'Fuel_efficiency': df['Fuel_efficiency'].median(), inplace=True)
df.fillna({'Power_perf_factor': df['Power_perf_factor'].median(), inplace=True)
```

```
In [91]: df.isnull().sum()
```

```
Out[91]: Manufacturer      0
Model                    0
Sales_in_thousands       0
__year_resale_value       0
Vehicle_type             0
Price_in_thousands       0
Engine_size              0
Horsepower               0
Wheelbase                0
Width                    0
Length                   0
Curb_weight              0
Fuel_capacity             0
Fuel_efficiency           0
Latest_launch            0
Power_perf_factor         0
dtype: int64
```

```
In [92]: numerical_df = df.select_dtypes(include=['number'])

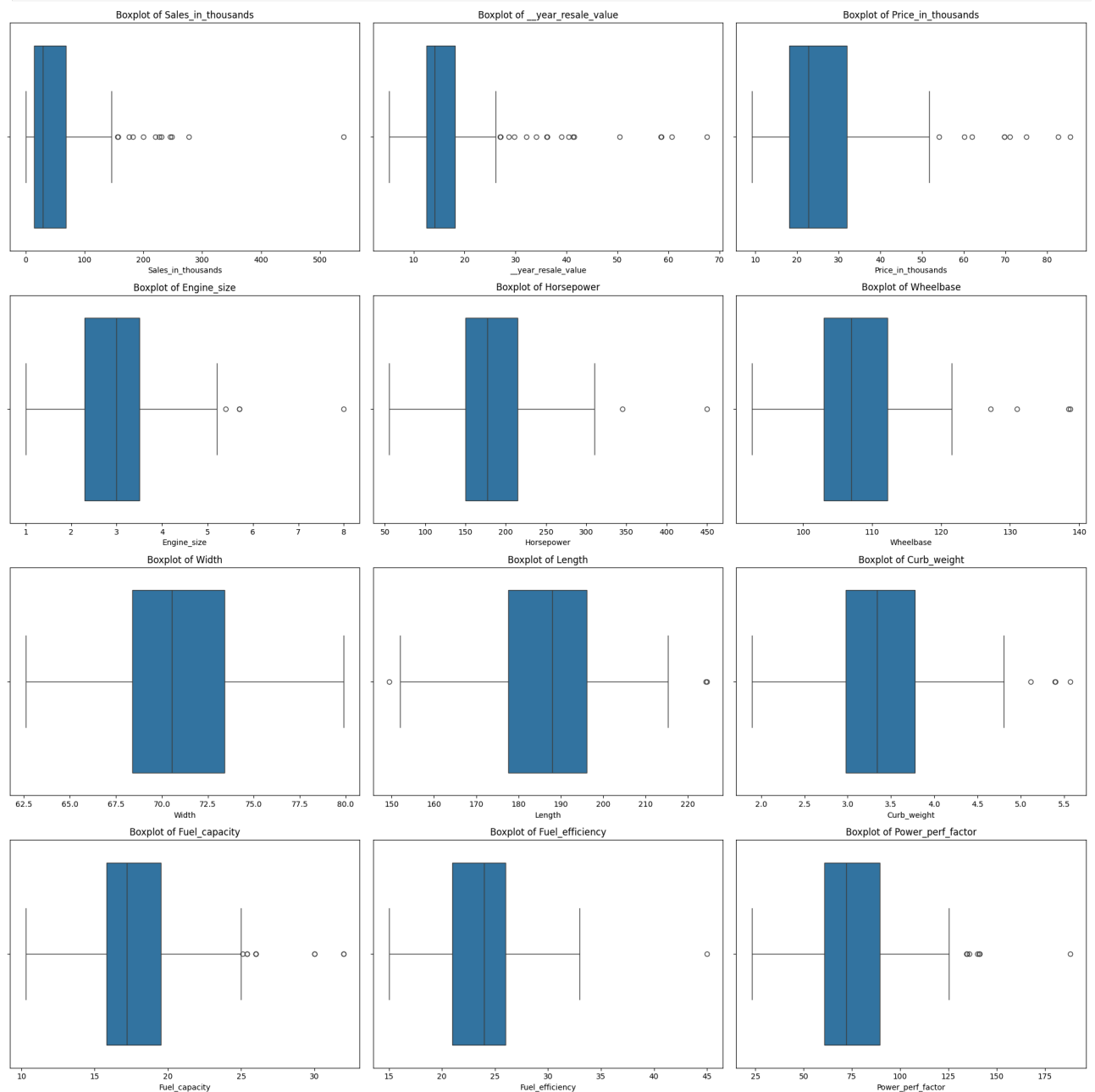
num_cols = numerical_df.columns
n = len(num_cols)

rows = math.ceil(n / 3) # Number of rows needed

plt.figure(figsize=(20, 5 * rows))

for i, col in enumerate(num_cols):
    plt.subplot(rows, 3, i + 1) # 3 plots per row
    sns.boxplot(x=numerical_df[col])
    plt.title(f"Boxplot of {col}")

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



```
In [93]: data = df.copy()

def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
```

```

        outliers = (column < lower_bound) | (column > upper_bound)
        return outliers

# Numeric columns only
numeric_cols = data.select_dtypes(include='number').columns

# Check each column for outliers
outlier_flag = {}
for col in numeric_cols:
    mask = detect_outliers_iqr(data[col])
    outlier_flag[col] = "Yes" if mask.any() else "No"

# Convert to DataFrame for nice display
outlier_df = pd.DataFrame(list(outlier_flag.items()), columns=['Column', 'Has Outliers'])
print(outlier_df)

```

```

      Column Has Outliers
0  Sales_in_thousands      Yes
1   _year_resale_value      Yes
2   Price_in_thousands      Yes
3      Engine_size         Yes
4      Horsepower         Yes
5      Wheelbase         Yes
6         Width          No
7         Length         Yes
8      Curb_weight         Yes
9      Fuel_capacity         Yes
10     Fuel_efficiency         Yes
11   Power_perf_factor         Yes

```

```

In [94]: data = df.copy()

def detect_outliers_zscore(column, threshold=3):
    mean = np.mean(column)
    std_dev = np.std(column)

    # Avoid division by zero
    if std_dev == 0:
        return np.zeros(len(column), dtype=bool)

    z_scores = (column - mean) / std_dev
    return np.abs(z_scores) > threshold

# Only evaluate numeric columns
numeric_cols = data.select_dtypes(include='number').columns

for col in numeric_cols:
    data[f'{col}_Outlier'] = detect_outliers_zscore(data[col])

# Filter rows where ANY outlier flag is True
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]

outliers_only

```

```

Out[94]:
   Manufacturer  Model  Sales_in_thousands  _year_resale_value  Vehicle_type  Price_in_thousands  Engine_size  Horsepower  Wheelbase  Width  ...  Price_in_thousands_Outlier  Engine_size_Ou
18    Cadillac  Escalade         14.785         14.180         Car         46.225         5.7         255.0         117.5         77.0  ...                False                I
26    Chevrolet  Metro          21.855          5.160    Passenger          9.235         1.0          55.0          93.1         62.6  ...                False                I
39     Dodge    Viper          0.916         58.470    Passenger         69.725         8.0         450.0          96.2         75.7  ...                False                I
40     Dodge    Ram Pickup        227.061         15.060         Car         19.460         5.2         230.0         138.7         79.3  ...                False                I
41     Dodge    Ram Wagon         16.767         15.510         Car         21.315         3.9         175.0         109.6         78.8  ...                False                I
42     Dodge    Ram Van          31.038         13.425         Car         18.575         3.9         175.0         127.2         78.8  ...                False                I
43     Dodge    Dakota          111.313         11.260         Car         16.980         2.5         120.0         131.0         71.5  ...                False                I
52     Ford    Explorer        276.747         16.640         Car         31.930         4.0         210.0         111.6         70.2  ...                False                I
56     Ford    F-Series        540.561         15.075         Car         26.935         4.6         220.0         138.5         79.1  ...                False                I
74     Lexus    LX470           9.126         14.180         Car         60.105         4.7         230.0         112.2         76.4  ...                False                I
78     Lincoln  Navigator        22.925         14.180         Car         42.660         5.4         300.0         119.0         79.9  ...                False                I
94    Mercedes-B  S-Class         16.774         50.375    Passenger         69.700         4.3         275.0         121.5         73.1  ...                False                I
95    Mercedes-B  SL-Class         3.311         58.600    Passenger         82.600         5.0         302.0          99.0         71.3  ...                 True                I
99    Mercedes-B  CL500          0.954         14.180    Passenger         85.500         5.0         302.0         113.6         73.1  ...                 True                I
125   Porsche    Carrera Coupe         1.280         60.625    Passenger         71.020         3.4         300.0          92.6         69.5  ...                 True                I
126   Porsche    Carrera Cabrio         1.866         67.550    Passenger         74.970         3.4         300.0          92.6         69.5  ...                 True                I

16 rows x 28 columns

```

```

In [95]: data = df.copy()

# Function to replace outliers with NaN using IQR
def replace_outliers_with_nan(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Replace outliers with NaN
    return column.where((column >= lower_bound) & (column <= upper_bound), np.nan)

# Select numeric columns
numeric_cols = data.select_dtypes(include='number').columns

# Apply to each numeric column
for col in numeric_cols:
    data[col] = replace_outliers_with_nan(data[col])

# Count null values after replacing outliers
null_counts = data[numeric_cols].isnull().sum()

print("Sum of null values in each numerical column:")
print(null_counts)

```

```

Sum of null values in each numerical column:
Sales_in_thousands    12
__year_resale_value    17
Price_in_thousands     9
Engine_size            4
Horsepower             2
Wheelbase              4
Width                  0
Length                 3
Curb_weight            4
Fuel_capacity          10
Fuel_efficiency         1
Power_perf_factor      7
dtype: int64

```

```

In [74]: from sklearn.impute import KNNImputer

# Separate numeric and non-numeric columns
numeric_cols = data.select_dtypes(include='number').columns
non_numeric_cols = data.select_dtypes(exclude='number').columns

# Apply KNN imputer only on numeric columns
imputer = KNNImputer(n_neighbors=2)
numeric_imputed = pd.DataFrame(
    imputer.fit_transform(data[numeric_cols]),
    columns=numeric_cols
)

# Recombine numeric + non-numeric columns
data_imputed_knn = pd.concat([numeric_imputed, data[non_numeric_cols]], axis=1)

# Check missing values
data_imputed_knn.isnull().sum()

```

```

Out[74]: Sales_in_thousands    0
__year_resale_value          0
Price_in_thousands          0
Engine_size                  0
Horsepower                   0
Wheelbase                    0
Width                        0
Length                       0
Curb_weight                  0
Fuel_capacity                 0
Fuel_efficiency               0
Power_perf_factor            0
Manufacturer                  0
Model                        0
Vehicle_type                  0
Latest_Launch                0
dtype: int64

```

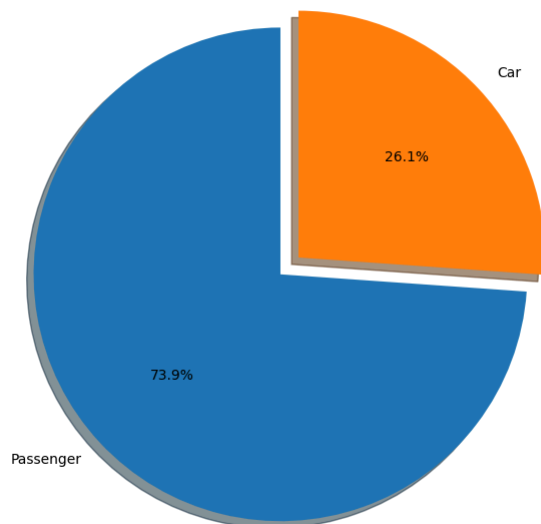
```

In [96]: category_counts = df['Vehicle_type'].value_counts()
explode = [0.1, 0]

# Create the pie chart
plt.figure(figsize=(8, 8)) # Optional: Adjust figure size
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', explode=explode, startangle=90, shadow=True)
plt.title('Distribution of Vehicle_type')
plt.show()

```

Distribution of Vehicle_type

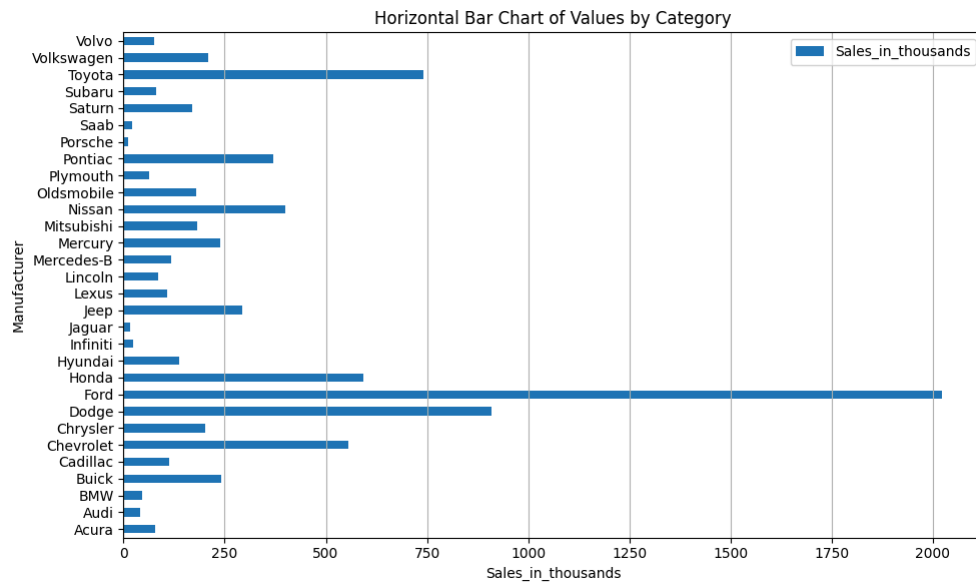


```

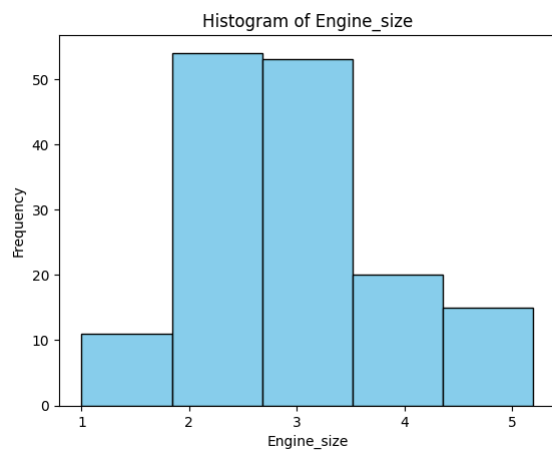
In [97]: grouped_df = df.groupby('Manufacturer')['Sales_in_thousands'].sum().reset_index()

grouped_df.plot.barh(y='Sales_in_thousands', x='Manufacturer', figsize=(10, 6), title='Horizontal Bar Chart of Values by Category')
plt.xlabel('Sales_in_thousands')
plt.ylabel('Manufacturer')
plt.tight_layout() # Adjust Layout to prevent Labels from overLapping
plt.grid(axis='x')
plt.show()

```



```
In [98]: plt.hist(data['Engine_size'], bins=5, color='skyblue', edgecolor='black')
plt.title('Histogram of Engine_size')
plt.xlabel('Engine_size')
plt.ylabel('Frequency')
plt.show()
```



```
In [99]: from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```
In [103]: data = df
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le
```

```
In [104]: X = data.drop(['Sales_in_thousands'], axis=1)
y = data['Sales_in_thousands']
X
```

```
Out[104]:
```

	Manufacturer	Model	_year	resale_value	Vehicle_type	Price_in_thousands	Engine_size	Horsepower	Wheelbase	Width	Length	Curb_weight	Fuel_capacity	Fuel_efficiency	Latest_Launch
0	0	79	16.360	1	21.500	1.8	140.0	101.2	67.3	172.4	2.639	13.2	28.0	47	
1	0	143	19.875	1	28.400	3.2	225.0	108.1	70.3	192.9	3.517	17.2	25.0	93	
2	0	25	18.225	1	22.799	3.2	225.0	106.9	70.6	192.0	3.470	17.2	26.0	9	
3	0	115	29.725	1	42.000	3.5	210.0	114.6	71.4	196.6	3.850	18.0	22.0	52	
4	1	8	22.255	1	23.990	1.8	150.0	102.6	68.2	178.0	2.998	16.4	27.0	20	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
152	29	148	14.180	1	24.400	1.9	160.0	100.5	67.6	176.6	3.042	15.8	25.0	124	
153	29	125	14.180	1	27.500	2.4	168.0	104.9	69.3	185.9	3.208	17.9	25.0	26	
154	29	149	14.180	1	28.800	2.4	168.0	104.9	69.3	186.2	3.259	17.9	25.0	89	
155	29	24	14.180	1	45.500	2.3	236.0	104.9	71.5	185.7	3.601	18.5	23.0	71	
156	29	126	14.180	1	36.000	2.9	201.0	109.9	72.1	189.8	3.600	21.1	24.0	22	

157 rows × 15 columns

```
In [106]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)
X
```

```
Out[106... array([[ -1.69422104,  0.02962311, -0.08085067, ...,  0.98226418,
        -0.4594708 , -0.75075723],
        [ -1.69422104,  1.45380028,  0.26558806, ...,  0.27268377,
         0.75557795,  0.57777638],
        [ -1.69422104, -1.17202637,  0.10296391, ...,  0.50921057,
        -1.46320673, -0.19868681],
        ...,
        [  1.69050402,  1.58731689, -0.29571167, ...,  0.27268377,
         0.64992154, -0.23381418],
        [  1.69050402, -1.19427914, -0.29571167, ..., -0.20036984,
         0.17446768,  0.98940037],
        [  1.69050402,  1.07550322, -0.29571167, ...,  0.03615696,
        -1.11982339,  0.35153557]], shape=(157, 15))
```

```
In [107... # Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split( X, y,
test_size=0.25, random_state=42)
```

```
In [108... import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def evaluate_model(model_name, y_true, y_pred):
    """
    Evaluate a regression model using MAE, MSE, RMSE, and R^2 score.
    Additionally, plot y_true vs. y_pred with a reference line and
    display model name and R^2 score on the plot.

    Parameters:
    model_name (str): Name of the regression model.
    y_true (array-like): True target values.
    y_pred (array-like): Predicted target values.

    Returns:
    dict: Dictionary containing MAE, MSE, RMSE, and R^2 score.
    """

    # Calculate metrics
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)

    # Store results
    results = {
        "Mean Absolute Error (MAE)": mae,
        "Mean Squared Error (MSE)": mse,
        "Root Mean Squared Error (RMSE)": rmse,
        "R^2 Score": r2
    }

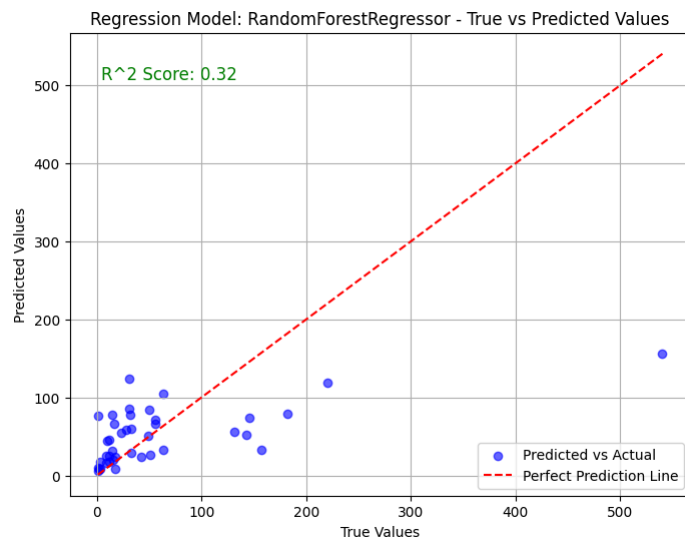
    # Plotting
    plt.figure(figsize=(8, 6))
    plt.scatter(y_true, y_pred, alpha=0.6, color='blue',
        label='Predicted vs Actual')
    plt.plot([min(y_true), max(y_true)],
        [min(y_true), max(y_true)],
        'r--', label='Perfect Prediction Line')
    plt.xlabel('True Values')
    plt.ylabel('Predicted Values')
    plt.title(f'Regression Model: {model_name} - True vs Predicted Values')
    plt.text(0.05, 0.9, f'R^2 Score: {r2:.2f}',
        transform=plt.gca().transAxes,
        fontsize=12, color='green')
    plt.legend()
    plt.grid(True)
    plt.show()

    return results
```

RandomForestRegressor

```
In [109... rf_regressor = RandomForestRegressor(n_estimators=100,
                                     random_state=42)

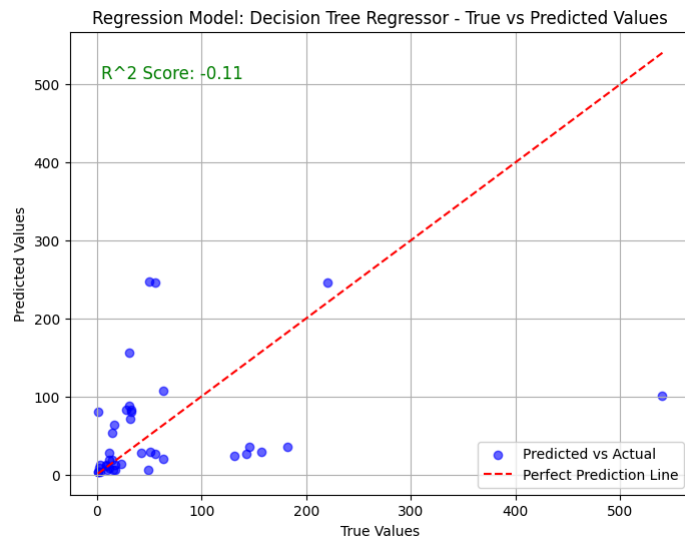
# Train the model
rf_regressor.fit(X_train, y_train)
# Make predictions
y_pred = rf_regressor.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)
# Print the evaluation results
evaluation_results
```



```
Out[109... {'Mean Absolute Error (MAE)': 44.744100749999994,
'Mean Squared Error (MSE)': 6010.729183884575,
'Root Mean Squared Error (RMSE)': np.float64(77.52889257486254),
'R^2 Score': 0.31943729710591673}
```

Decision Tree Regressor

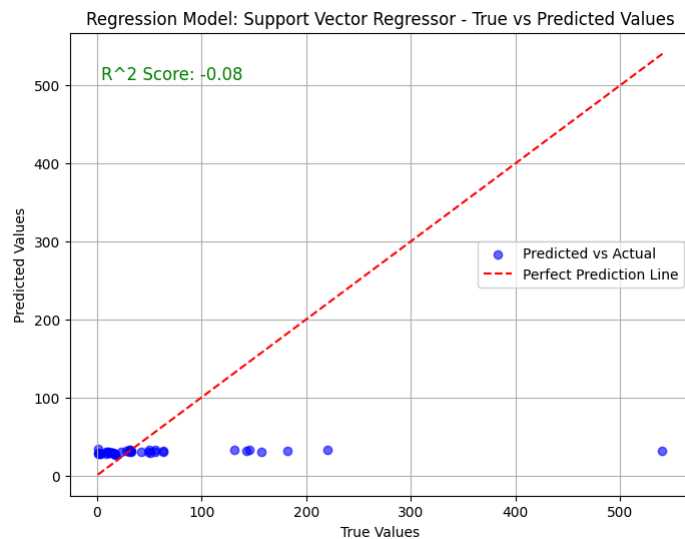
```
In [112... DT = DecisionTreeRegressor(random_state=42)
# Train the model
DT.fit(X_train, y_train)
# Make predictions
y_pred = DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Decision Tree Regressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f'{key}: {value:.4f}' if isinstance(value, float) else
f'{key}: \n{value}')
```



Mean Absolute Error (MAE): 57.3321
Mean Squared Error (MSE): 9759.9426
Root Mean Squared Error (RMSE): 98.7924
R² Score: -0.1051

Support Vector Regressor (SVR)

```
In [113... SVM = SVR()
# Train the model
SVM.fit(X_train, y_train)
# Make predictions
y_pred = SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Regressor',
y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f'{key}: {value:.4f}' if isinstance(value, float) else
f'{key}: \n{value}')
```

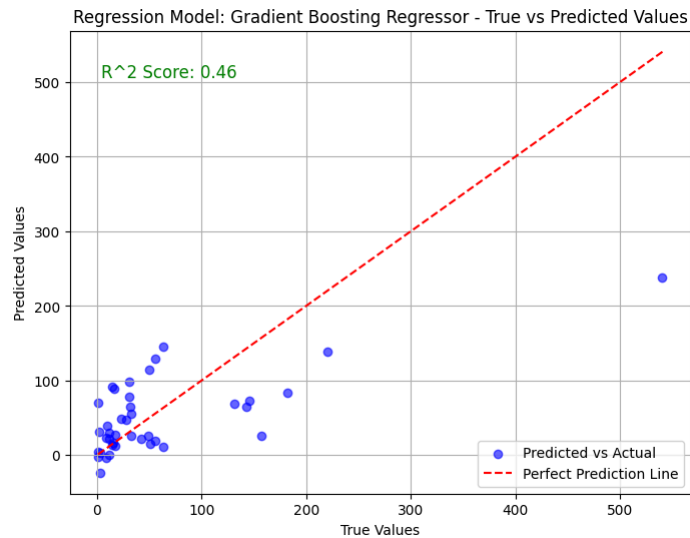


Mean Absolute Error (MAE): 46.0422
Mean Squared Error (MSE): 9499.4931
Root Mean Squared Error (RMSE): 97.4653
R² Score: -0.0756

Gradient Boosting Regressor

```
In [114... GBR = GradientBoostingRegressor(random_state=42)
# Train the model
GBR.fit(X_train, y_train)
# Make predictions
y_pred = GBR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Gradient Boosting Regressor',
y_test, y_pred)
```

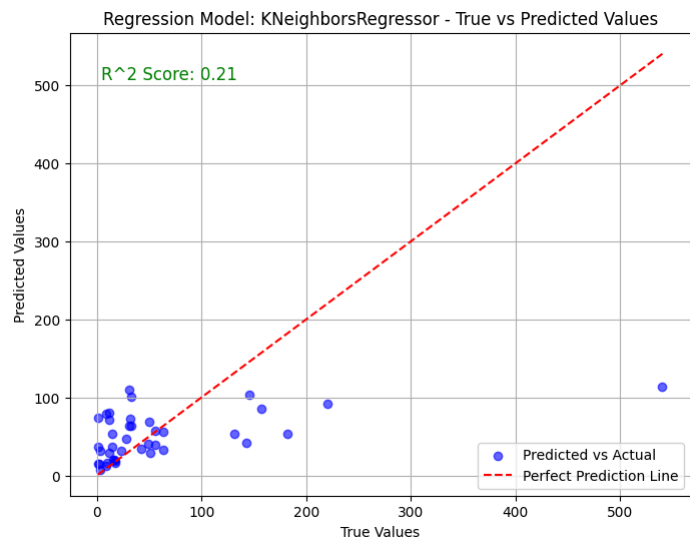
```
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: \n{value}")
```



Mean Absolute Error (MAE): 45.6848
 Mean Squared Error (MSE): 4778.4029
 Root Mean Squared Error (RMSE): 69.1260
 R^2 Score: 0.4590

KNeighborsRegressor

```
In [115]: knn = KNeighborsRegressor()
knn.fit(X_train, y_train)
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('KNeighborsRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: \n{value}")
```



Mean Absolute Error (MAE): 45.7872
 Mean Squared Error (MSE): 6968.9906
 Root Mean Squared Error (RMSE): 83.4885
 R^2 Score: 0.2109

Hybridparams optimization

RandomForestRegressor with grid search

```
In [116]: from sklearn.model_selection import train_test_split, GridSearchCV
rf = RandomForestRegressor(random_state=42)
# Define parameter grid (try all combinations)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
# Grid Search
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    verbose=2
)
grid_search.fit(X_train, y_train)
# Evaluate best model
best_rf = grid_search.best_estimator_
```

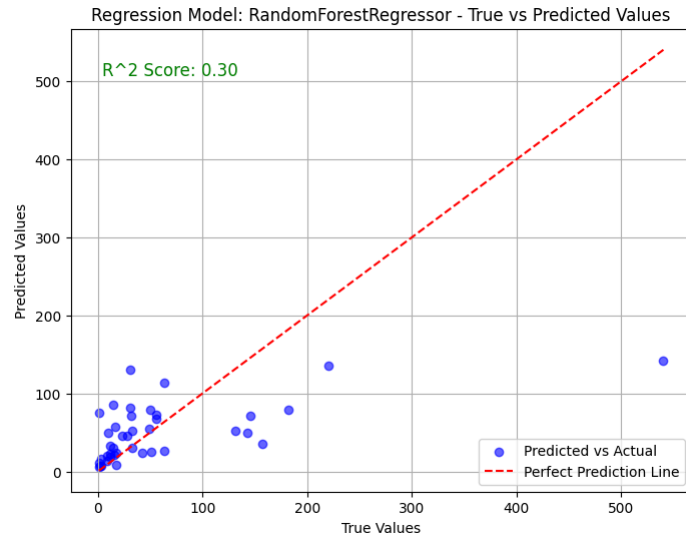
```

y_pred = best_rf.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

print("Best Parameters:", grid_search.best_params_)

```

Fitting 3 folds for each of 24 candidates, totalling 72 fits



Mean Absolute Error (MAE): 43.8476
 Mean Squared Error (MSE): 6221.4250
 Root Mean Squared Error (RMSE): 78.8760
 R^2 Score: 0.2956
 Best Parameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}

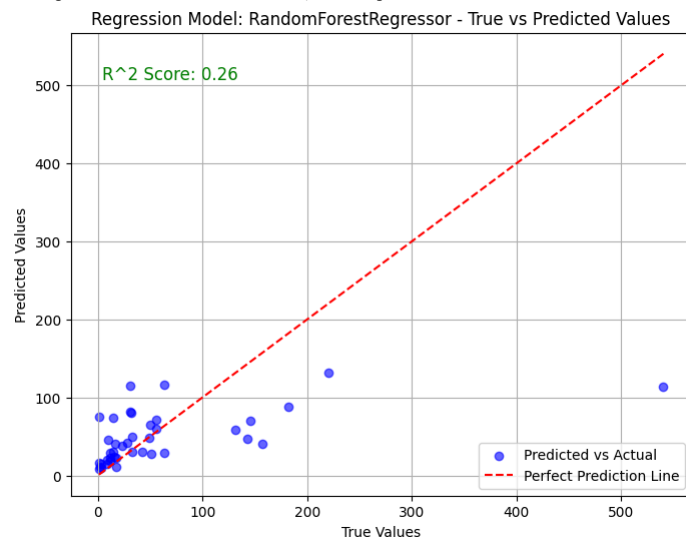
RandomForestRegressor with RandomizedSearchCV

```

In [119]: from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
# Define model
rf = RandomForestRegressor(random_state=42)
# Define parameter distributions (sample randomly)
param_distributions = {
    'n_estimators': randint(100, 500),
    'max_depth': [None, 5, 10, 16],
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 5)
}
# Random Search
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_distributions,
    n_iter=20, # number of random combinations to try
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    random_state=42,
    verbose=2
)
random_search.fit(X_train, y_train)
# Evaluate best model
best_rf_random = random_search.best_estimator_
y_pred = best_rf_random.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
print("Best Parameters:", grid_search.best_params_)

```

Fitting 3 folds for each of 20 candidates, totalling 60 fits

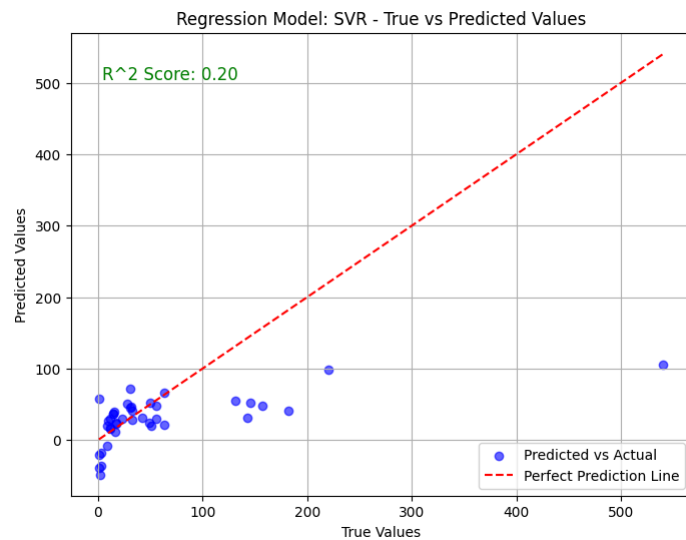


Mean Absolute Error (MAE): 42.2091
Mean Squared Error (MSE): 6565.3053
Root Mean Squared Error (RMSE): 81.0266
R² Score: 0.2566
Best Parameters: {'C': 10, 'kernel': 'linear'}

SVM grid search

```
In [118]: SVM = SVR()
param_grid = {
    'C': [1, 10],
    'kernel': ['linear', 'rbf']
}
grid_search = GridSearchCV(
    SVM,
    param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    verbose=2
)
grid_search.fit(X_train, y_train)
print("Best Parameters (Grid Search):", grid_search.best_params_)
y_pred = grid_search.best_estimator_.predict(X_test)
evaluation_results = evaluate_model("SVR", y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: \n{value}")
```

Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best Parameters (Grid Search): {'C': 10, 'kernel': 'linear'}



Mean Absolute Error (MAE): 43.3808
Mean Squared Error (MSE): 7066.3677
Root Mean Squared Error (RMSE): 84.0617
R² Score: 0.1999

Since the R² for the Gradient Boosting Regressor is the highest score among them all, We would say that Gradient Boosting Regressor is the best model for our dataset.

In []: