

```
In [14]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

```
In [15]: df = pd.read_csv('Apartment_Market_Prices_missing.csv')
```

```
In [16]: df.head()
```

Out[16]:

	Year	OBJECTID	GEOID	Tract Label	Tract Name	Community Reporting Area Name	Community Reporting Area ID	Tract Median Apartment Contract Rent per Square Foot	Tract Median Apartment Contract Rent per Unit	Year over Year Change in Rent per Square Foot	Year over Year Change in Rent per Unit
0	2001	1	53033001201	12.01	Census Tract 12.01	Northgate/Maple Leaf	8.1	1.52	1085.0	482.758.620.689.656	513.565
1	2001	2	53033000404	4.04	Census Tract 4.04	Broadview/Bitter Lake	9.1	1.29	1042.0	403.225.806.451.613	
2	2001	3	53033004901	49.01	Census Tract 49.01	Fremont	11.1	1.79	1042.0	529.411.764.705.883	525.252
3	2001	4	53033003601	36.01	Census Tract 36.01	Green Lake	9.4	2.10	1197.0	552.763.819.095.478	555.555
4	2001	5	53033002800	28.00	Census Tract 28	Greenwood/Phinney Ridge	9.3	1.60	NaN	526.315.789.473.685	521.276

```
In [17]: len(df)
```

```
Out[17]: 4320
```

```
In [18]: df.isnull().sum()
```

Out[18]:

Year	0
OBJECTID	0
GEOID	0
Tract Label	0
Tract Name	0
Community Reporting Area Name	0
Community Reporting Area ID	0
Tract Median Apartment Contract Rent per Square Foot	0
Tract Median Apartment Contract Rent per Unit	418
Year over Year Change in Rent per Square Foot	0
Year over Year Change in Rent per Unit	0
Cost Category	0
Year over Year Change in Rent Category	0
Mixed-Rate or Mixed-Income Apartments in Tract	0
PROPERTIES	0
Shape__Area	0
Shape__Length	0
dtype:	int64

```
In [19]: # Calculate the percentage of missing values in each column
missing_percentage = df.isnull().mean() * 100

# Select columns with more than 30% missing values
columns_with_missing_30 = missing_percentage[missing_percentage > 30]

# Print the result
print("Columns with more than 30% missing values:")
print(columns_with_missing_30)
```

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

```
In [20]: df.describe()
```

Out[20]:

	Year	OBJECTID	GEOID	Tract Label	Community Reporting Area ID	Tract Median Apartment Contract Rent per Square Foot	Tract Median Apartment Contract Rent per Unit	Mixed-Rate or Mixed-Income Apartments in Tract	PROPERTIES
count	4320.000000	4320.000000	4.320000e+03	4320.000000	4320.000000	4320.000000	3902.000000	4320.000000	4320.000000
mean	2012.500000	2160.500000	5.303301e+10	66.461500	7.476111	1.956067	1291.778831	547.108565	22.421296
std	6.922988	1247.220911	4.255099e+03	42.550986	3.471062	0.932163	622.979289	528.865123	21.058124
min	2001.000000	1.000000	5.303300e+10	1.010000	1.100000	0.000000	0.000000	0.000000	0.000000
25%	2006.750000	1080.750000	5.303300e+10	37.505000	4.300000	1.510000	1050.000000	122.000000	6.000000
50%	2012.500000	2160.500000	5.303301e+10	67.015000	7.400000	2.100000	1355.000000	396.000000	16.000000
75%	2018.250000	3240.250000	5.303301e+10	94.250000	10.300000	2.600000	1652.750000	852.000000	33.000000
max	2024.000000	4320.000000	5.303303e+10	265.000000	13.300000	4.220000	3683.000000	3708.000000	109.000000

```
In [21]: print(df['Tract Median Apartment Contract Rent per Unit'].mean())
```

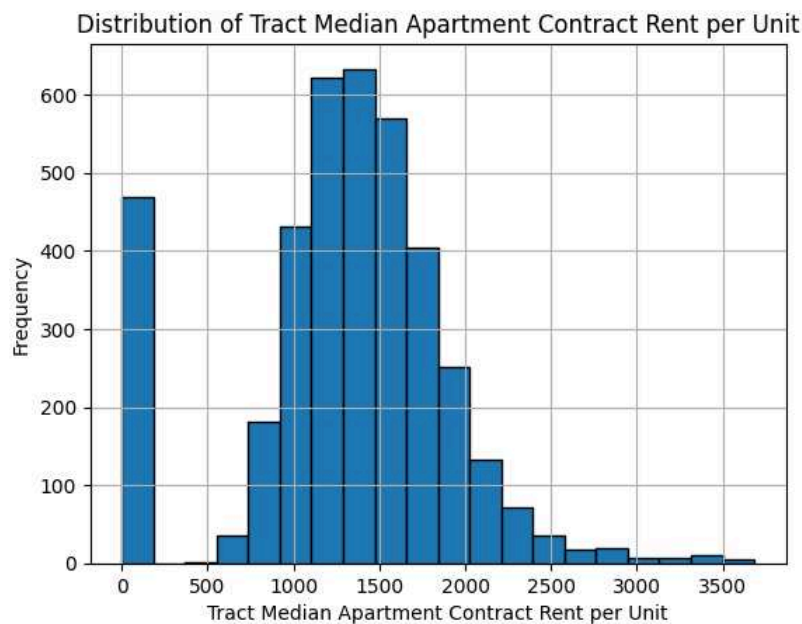
1291.778831368529

```
In [22]: print(df['Tract Median Apartment Contract Rent per Unit'].median())
```

1355.0

```
In [23]: # Histogram for the column
column_name = 'Tract Median Apartment Contract Rent per Unit'

# Plot histogram
df[column_name].hist(bins=20, edgecolor='black') # More bins give better detail
plt.title('Distribution of Tract Median Apartment Contract Rent per Unit')
plt.xlabel(column_name)
plt.ylabel('Frequency')
plt.show()
```



```
In [25]: import numpy as np
numeric_cols = df.select_dtypes(include=np.number).columns
```

```
for col in numeric_cols:
    df[col] = df[col].fillna(df[col].median())
```

```
In [26]: print(df['Tract Median Apartment Contract Rent per Unit'].isna().sum())
```

0

```
In [27]: # Function to detect outliers using IQR
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
```

```
# Apply the function to all numeric columns
numeric_cols = df.select_dtypes(include='number').columns

outlier_df = pd.DataFrame({
    'Column': numeric_cols,
    'Has Outliers': ["Yes" if detect_outliers_iqr(df[col]).any() else "No" for col in numeric_cols]
})

print(outlier_df)
```

		Column Has Outliers
0	Year	No
1	OBJECTID	No
2	GEOID	Yes
3	Tract Label	Yes
4	Community Reporting Area ID	No
5	Tract Median Apartment Contract Rent per Squar...	No
6	Tract Median Apartment Contract Rent per Unit	Yes
7	Mixed-Rate or Mixed-Income Apartments in Tract	Yes
8	PROPERTIES	Yes

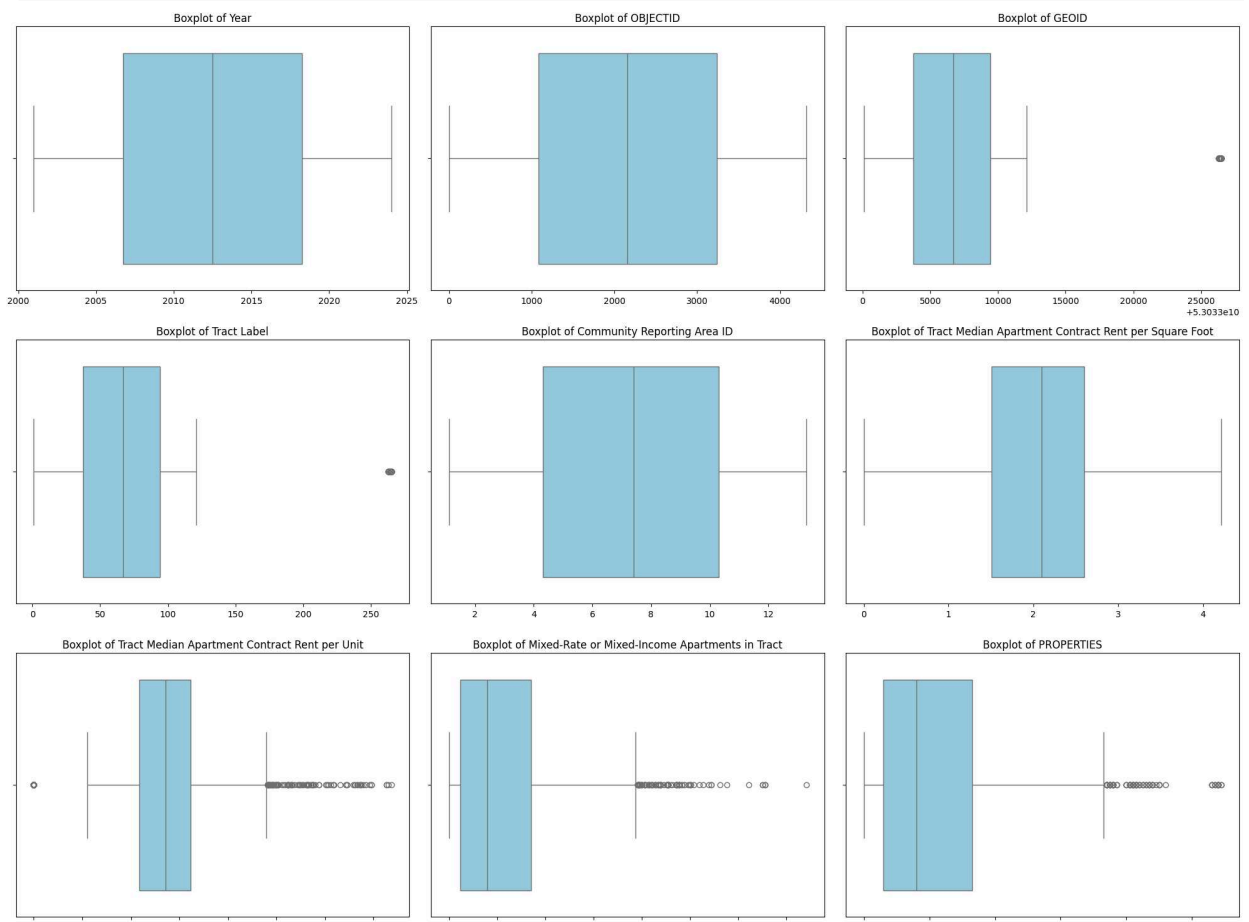
```
In [28]: # Select numeric columns
numerical_df = df.select_dtypes(include=['number'])
num_cols = numerical_df.columns

# Calculate number of rows (3 plots per row)
rows = math.ceil(len(num_cols) / 3)

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

for i, col in enumerate(num_cols):
    plt.subplot(rows, 3, i + 1)
    sns.boxplot(x=numerical_df[col].dropna(), color='skyblue')
    plt.title(f"Boxplot of {col}")
    plt.xlabel('') # optional

plt.tight_layout()
plt.show() # no extra spaces or parentheses
```



```
In [29]: def remove_outliers_iqr(data):
# Select numeric columns
numeric_data = data.select_dtypes(include='number')

# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = numeric_data.quantile(0.25)
```

```

Q3 = numeric_data.quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter data: keep only rows where all numeric columns are within bounds
filtered_data = data[~((numeric_data < lower_bound) | (numeric_data > upper_bound)).any(axis=1)]

return filtered_data

# Apply the function
cleaned_data = remove_outliers_iqr(df)
print(f"Original rows: {len(df)}, After removing outliers: {len(cleaned_data)}")

```

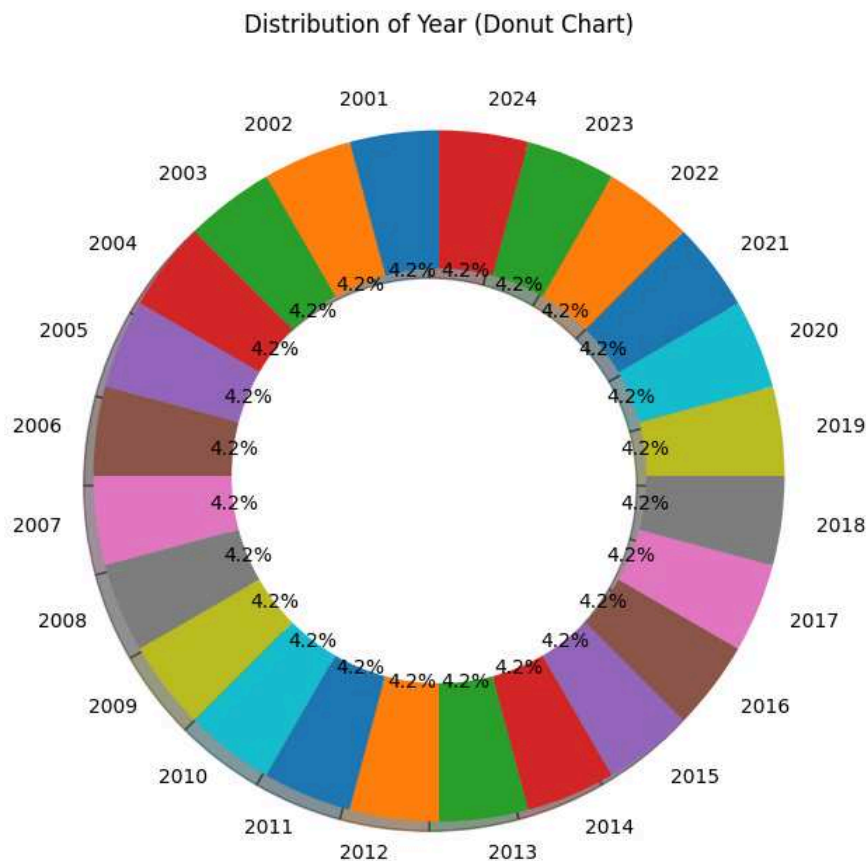
Original rows: 4320, After removing outliers: 3536

```

In [30]: # Count values in PROPERTIES
category_counts = df['Year'].value_counts()

plt.figure(figsize=(8,8))
plt.pie(
    category_counts,
    labels=category_counts.index,
    autopct='%1.1f%%',
    startangle=90,
    shadow=True,
    wedgeprops={'width': 0.4} # Donut hole
)
plt.title('Distribution of Year (Donut Chart)')
plt.show()

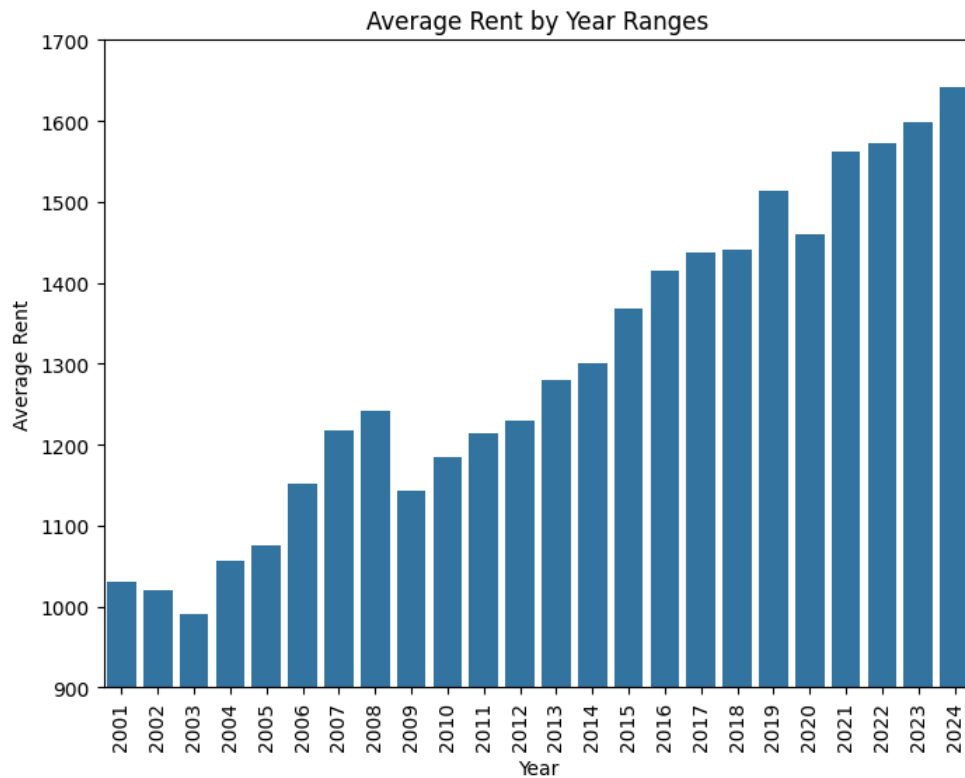
```



```

In [31]: plt.figure(figsize=(8,6))
sns.barplot(
    x=df['Year'],
    y=df['Tract Median Apartment Contract Rent per Unit'],
    estimator='mean',
    errorbar=None
)
plt.xticks(rotation=90)
plt.title('Average Rent by Year Ranges')
plt.ylabel('Average Rent')
plt.xlabel('Year')
plt.ylim(900, 1700)
plt.show()

```



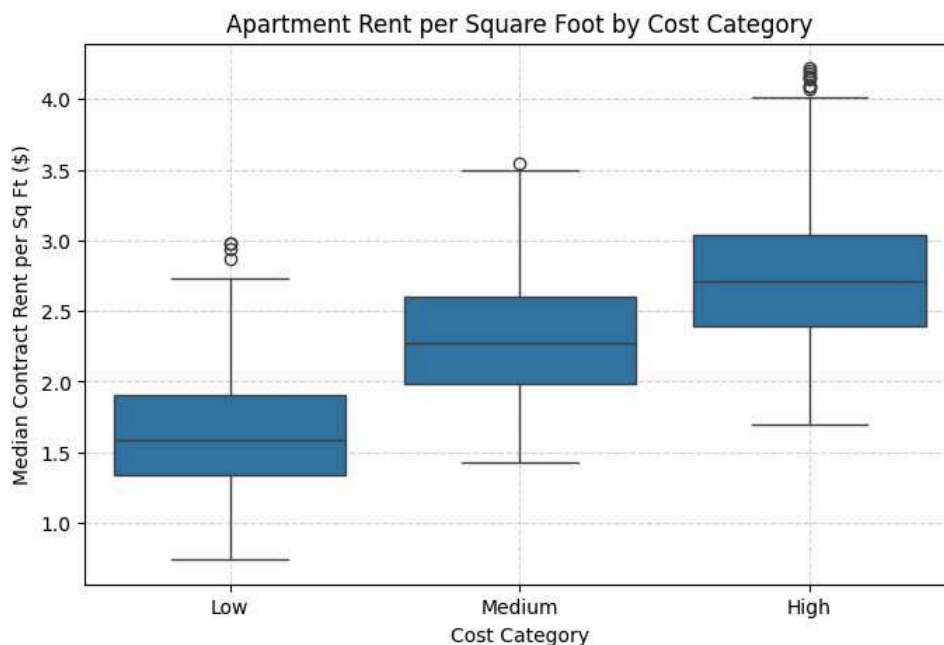
```
In [33]: def clean_dotted_number(s):
    try:
        if isinstance(s, float) or s is None:
            return s
        s_clean = s.replace('.', '')
        return float(s_clean) / 1e9
    except Exception:
        return np.nan

df['Tract Median Apartment Contract Rent per Unit'] = df['Tract Median Apartment Contract Rent per Unit'].apply(clean_dotted)

df['Tract Median Apartment Contract Rent per Square Foot'] = pd.to_numeric(df['Tract Median Apartment Contract Rent per Square Foot'], errors='coerce')

df = df[df['Cost Category'] != '0'] # remove invalid rows if any

plt.figure(figsize=(8,5))
sns.boxplot(data=df, x='Cost Category', y='Tract Median Apartment Contract Rent per Square Foot')
plt.title('Apartment Rent per Square Foot by Cost Category')
plt.ylabel('Median Contract Rent per Sq Ft ($)')
plt.xlabel('Cost Category')
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```



```
In [34]: 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 [35]: 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 [38]: X = data.drop(['Year over Year Change in Rent per Unit'], axis=1)
y = data['Year over Year Change in Rent per Unit']
```

```
In [40]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
In [41]: X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
```

```
In [42]: 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
    }

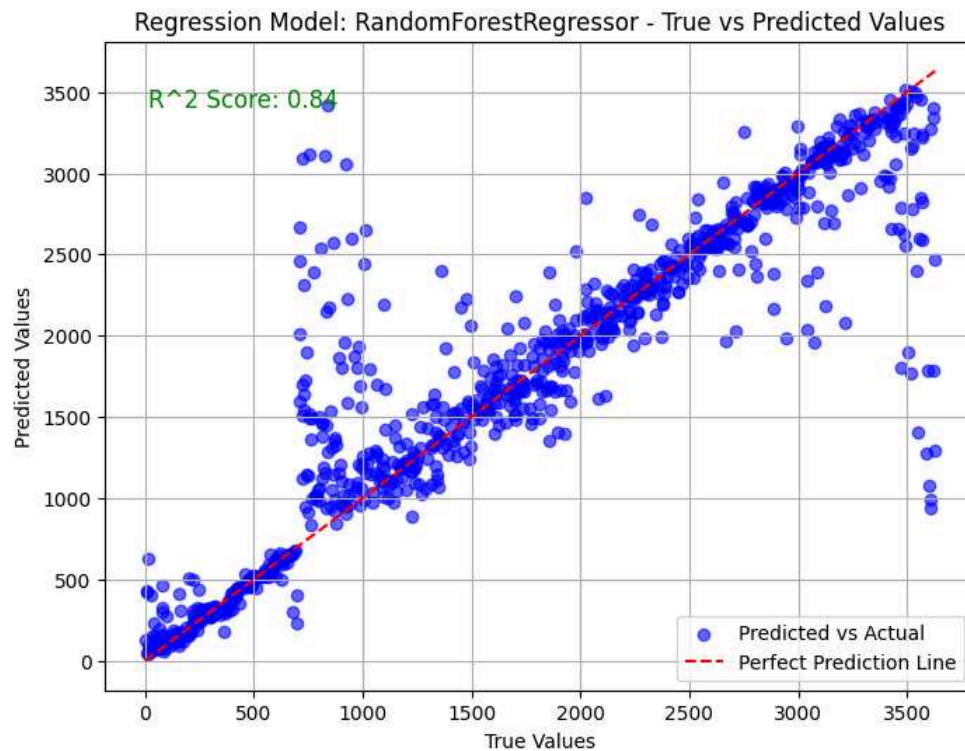
    # Plotting
    plt.figure(figsize=(8, 6))
    plt.scatter(y_true, y_pred, alpha=0.6, color='blue',
        label='Predicted vs Actual')
    plt.plot([min(y_true), max(y_true)],
        [min(y_true), max(y_true)],
        'r--', label='Perfect Prediction Line')
    plt.xlabel('True Values')
    plt.ylabel('Predicted 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 [43]: rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)
```

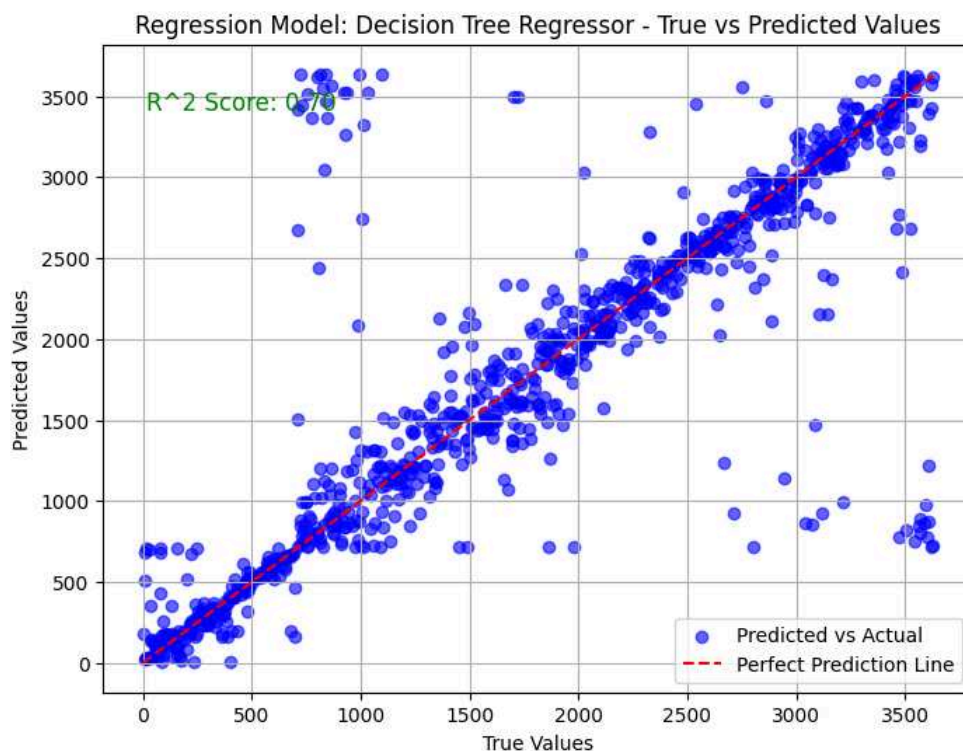
```
# Train the model
rf_regressor.fit(X_train, y_train)
# Make predictions
y_pred = rf_regressor.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)
# Print the evaluation results
evaluation_results
```



```
Out[43]: {'Mean Absolute Error (MAE)': 198.67751317175976,
'Mean Squared Error (MSE)': 179595.04474099053,
'Root Mean Squared Error (RMSE)': np.float64(423.7865563973535),
'R^2 Score': 0.8415520170760479}
```

Decision Tree Regressor

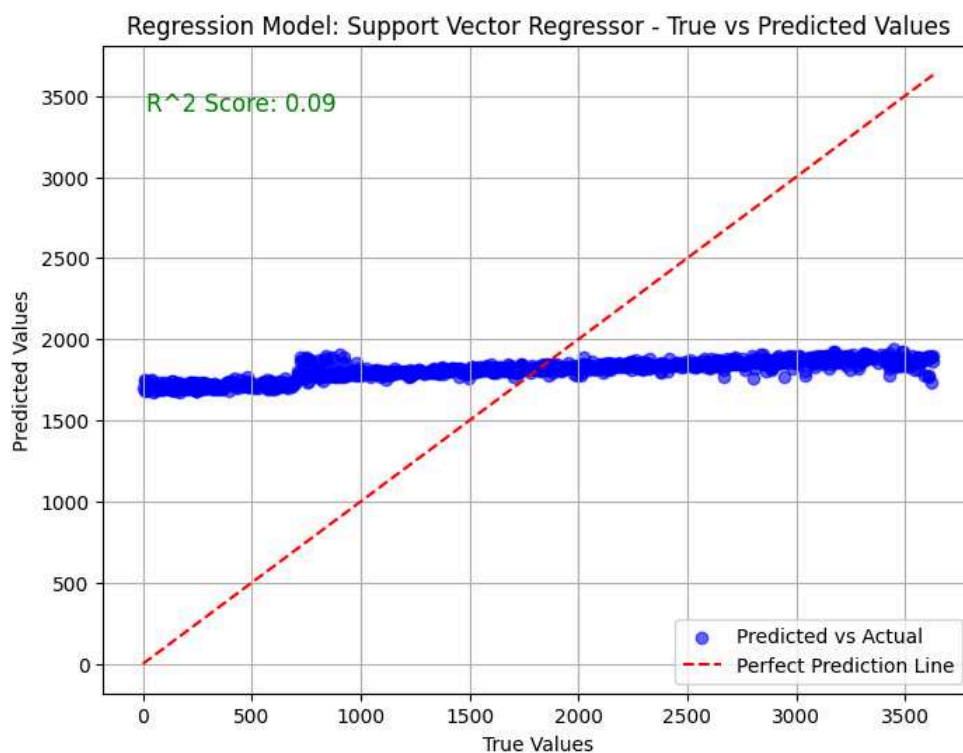
```
In [44]: DT = DecisionTreeRegressor(random_state=42)
# Train the model
DT.fit(X_train, y_train)
# Make predictions
y_pred = DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Decision Tree Regressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```



Mean Absolute Error (MAE): 244.8894
Mean Squared Error (MSE): 344811.4247
Root Mean Squared Error (RMSE): 587.2065
R² Score: 0.6958

Support Vector Regressor

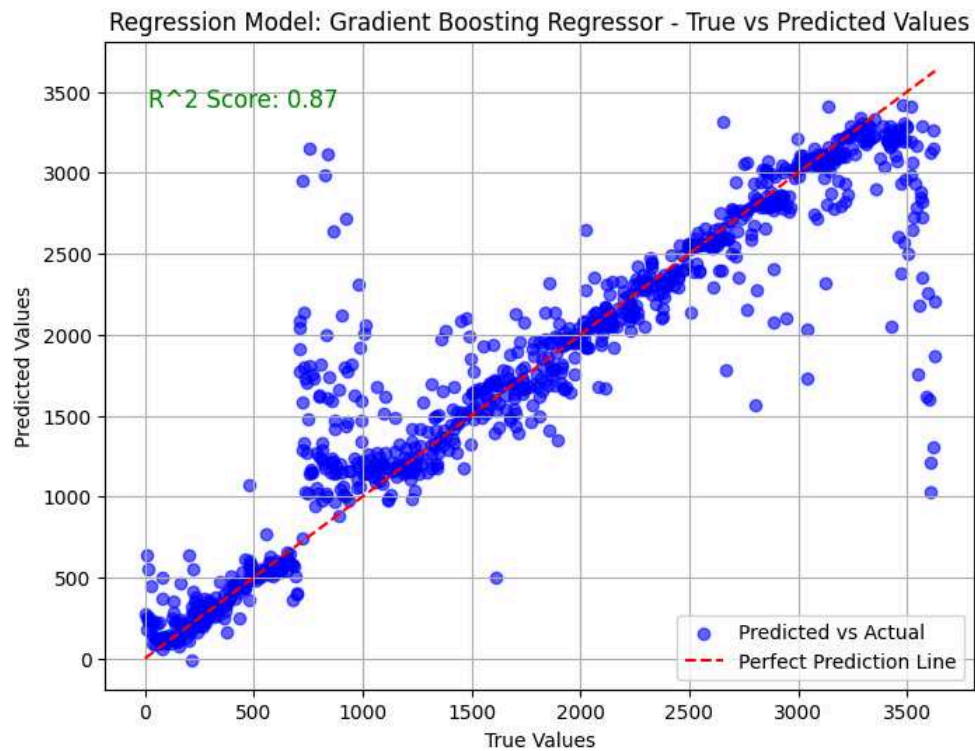
```
In [45]: SVM = SVR()
# Train the model
SVM.fit(X_train, y_train)
# Make predictions
y_pred = SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Regressor',
y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```



Mean Absolute Error (MAE): 883.3731
Mean Squared Error (MSE): 1030804.3989
Root Mean Squared Error (RMSE): 1015.2854
R² Score: 0.0906

Gradient Boosting Regressor

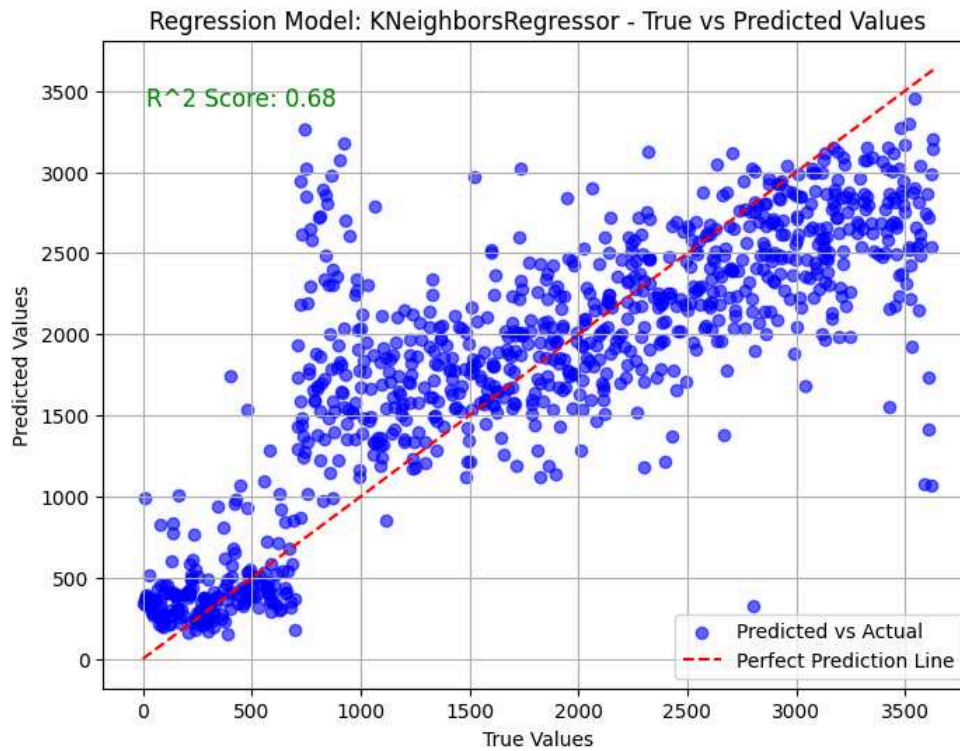
```
In [46]: GBR = GradientBoostingRegressor(random_state=42)
# Train the model
GBR.fit(X_train, y_train)
# Make predictions
y_pred = GBR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Gradient Boosting Regressor',
y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```



Mean Absolute Error (MAE): 199.2288
Mean Squared Error (MSE): 151240.6999
Root Mean Squared Error (RMSE): 388.8968
R² Score: 0.8666

KNeighborsRegressor

```
In [47]: 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): 439.4190
Mean Squared Error (MSE): 368305.8096
Root Mean Squared Error (RMSE): 606.8820
R² Score: 0.6751

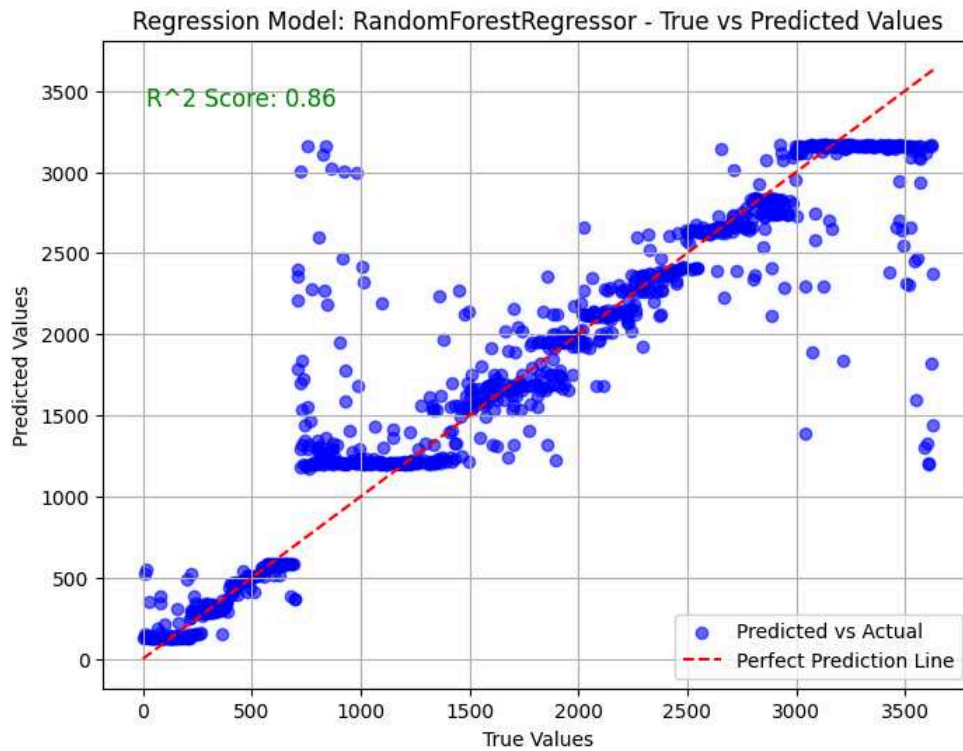
Hybridparamets optimization

RandomForestRegressor with grid search

```
In [49]: 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): 203.4034

Mean Squared Error (MSE): 162752.8553

Root Mean Squared Error (RMSE): 403.4264

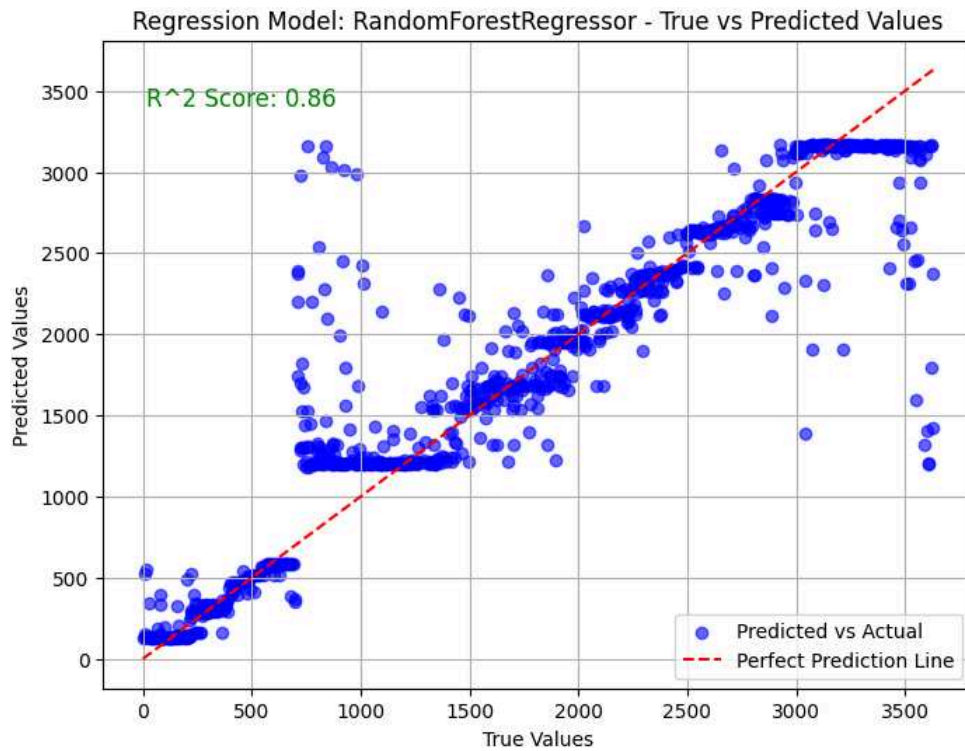
R^2 Score: 0.8564

Best Parameters: {'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}

RandomForestRegressor with RandomizedSearchCV

```
In [50]: 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): 201.8879

Mean Squared Error (MSE): 160704.9384

Root Mean Squared Error (RMSE): 400.8802

R^2 Score: 0.8582

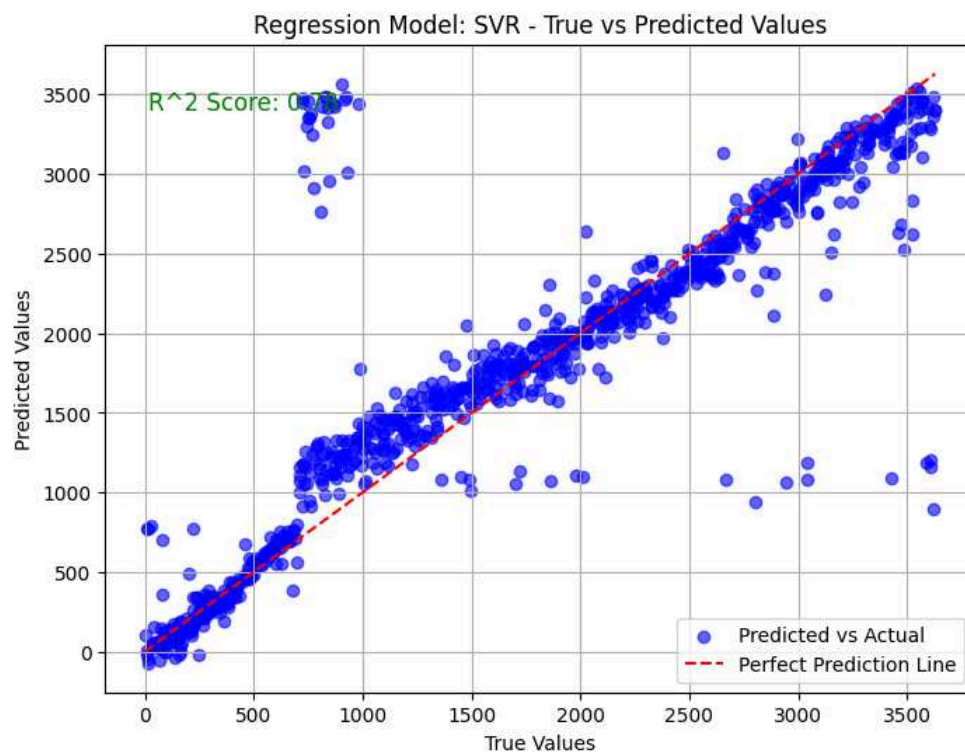
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}

SVM grid search

```
In [51]: 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': 1, 'kernel': 'linear'}



Mean Absolute Error (MAE): 227.9995
Mean Squared Error (MSE): 251269.5676
Root Mean Squared Error (RMSE): 501.2680
R^2 Score: 0.7783

The best model for prediction 'Tract Median Apartment Contract Rent per Unit' is Gradient Boosting Regressor highest R^2 is 0.87