

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

```
In [1]: import pandas as pd
```

```
In [2]: df = pd.read_csv('breast-cancer.csv')
```

```
In [3]: df.head()
```

```
Out[3]:
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
--	----	-----------	-------------	--------------	----------------	-----------	---------

0	842302	M	17.99	10.38	122.80	1001.0
1	842517	M	20.57	17.77	132.90	1326.0
2	84300903	M	19.69	21.25	130.00	1203.0
3	84348301	M	11.42	20.38	77.58	386.1
4	84358402	M	20.29	14.34	135.10	1297.0

5 rows × 7 columns



```
In [4]: df.isnull().sum()
```

```
Out[4]: id          0
        diagnosis   0
        radius_mean 0
        texture_mean 0
        perimeter_mean 0
        area_mean    0
        smoothness_mean 0
        compactness_mean 0
        concavity_mean 0
        concave points_mean 0
        symmetry_mean 0
        fractal_dimension_mean 0
        radius_se     0
        texture_se     0
        perimeter_se   0
        area_se        0
        smoothness_se  0
        compactness_se 0
        concavity_se   0
        concave points_se 0
        symmetry_se     0
        fractal_dimension_se 0
        radius_worst   0
        texture_worst  0
        perimeter_worst 0
        area_worst     0
        smoothness_worst 0
        compactness_worst 0
        concavity_worst 0
        concave points_worst 0
        symmetry_worst  0
        fractal_dimension_worst 0
        dtype: int64
```

```
In [5]: df.dtypes
```

```
Out[5]: id                int64
        diagnosis         str
        radius_mean       float64
        texture_mean      float64
        perimeter_mean     float64
        area_mean          float64
        smoothness_mean    float64
        compactness_mean   float64
        concavity_mean     float64
        concave points_mean float64
        symmetry_mean      float64
        fractal_dimension_mean float64
        radius_se          float64
        texture_se         float64
        perimeter_se       float64
        area_se            float64
        smoothness_se      float64
        compactness_se     float64
        concavity_se       float64
        concave points_se  float64
        symmetry_se        float64
        fractal_dimension_se float64
        radius_worst       float64
        texture_worst      float64
        perimeter_worst    float64
        area_worst         float64
        smoothness_worst   float64
        compactness_worst  float64
        concavity_worst    float64
        concave points_worst float64
        symmetry_worst     float64
        fractal_dimension_worst float64
        dtype: object
```

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

Out[6]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.054687
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.018101
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.037461
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.047445
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.051219
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.057118
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.173403

8 rows × 31 columns



```
In [15]: from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis'])
```

```
In [14]: pip install scikit-learn
```

Requirement already satisfied: scikit-learn in C:\Users\M\anaconda3\envs\Code1\Lib\site-packages (1.8.0)
Requirement already satisfied: numpy>=1.24.1 in C:\Users\M\anaconda3\envs\Code1\Lib\site-packages (from scikit-learn) (2.4.2)
Requirement already satisfied: scipy>=1.10.0 in C:\Users\M\anaconda3\envs\Code1\Lib\site-packages (from scikit-learn) (1.17.1)
Requirement already satisfied: joblib>=1.3.0 in C:\Users\M\anaconda3\envs\Code1\Lib\site-packages (from scikit-learn) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.2.0 in C:\Users\M\anaconda3\envs\Code1\Lib\site-packages (from scikit-learn) (3.6.0)
Note: you may need to restart the kernel to use updated packages.

```
In [16]: X = df.drop(['diagnosis', 'id'], axis=1)
y = df['diagnosis']
```

```
In [18]: from sklearn.feature_selection import SelectKBest, chi2
selector = SelectKBest(score_func=chi2, k=20)
selector.fit(X, y)
print("Selected Features:", selector.get_support())
print("Scores:", selector.scores_)
```

```

Selected Features: [ True  True  True  True False  True  True  True False False  True
 True  True False  True  True False False False  True  True  True  True
 False  True  True  True  True False]
Scores: [2.66104917e+02 9.38975081e+01 2.01110286e+03 5.39916559e+04
 1.49899264e-01 5.40307549e+00 1.97123536e+01 1.05440354e+01
 2.57379775e-01 7.43065536e-05 3.46752472e+01 9.79353970e-03
 2.50571896e+02 8.75850471e+03 3.26620664e-03 6.13785332e-01
 1.04471761e+00 3.05231563e-01 8.03633831e-05 6.37136566e-03
 4.91689157e+02 1.74449400e+02 3.66503542e+03 1.12598432e+05
 3.97365694e-01 1.93149220e+01 3.95169151e+01 1.34854195e+01
 1.29886140e+00 2.31522407e-01]

```

```

In [19]: import pandas as pd
import matplotlib.pyplot as plt

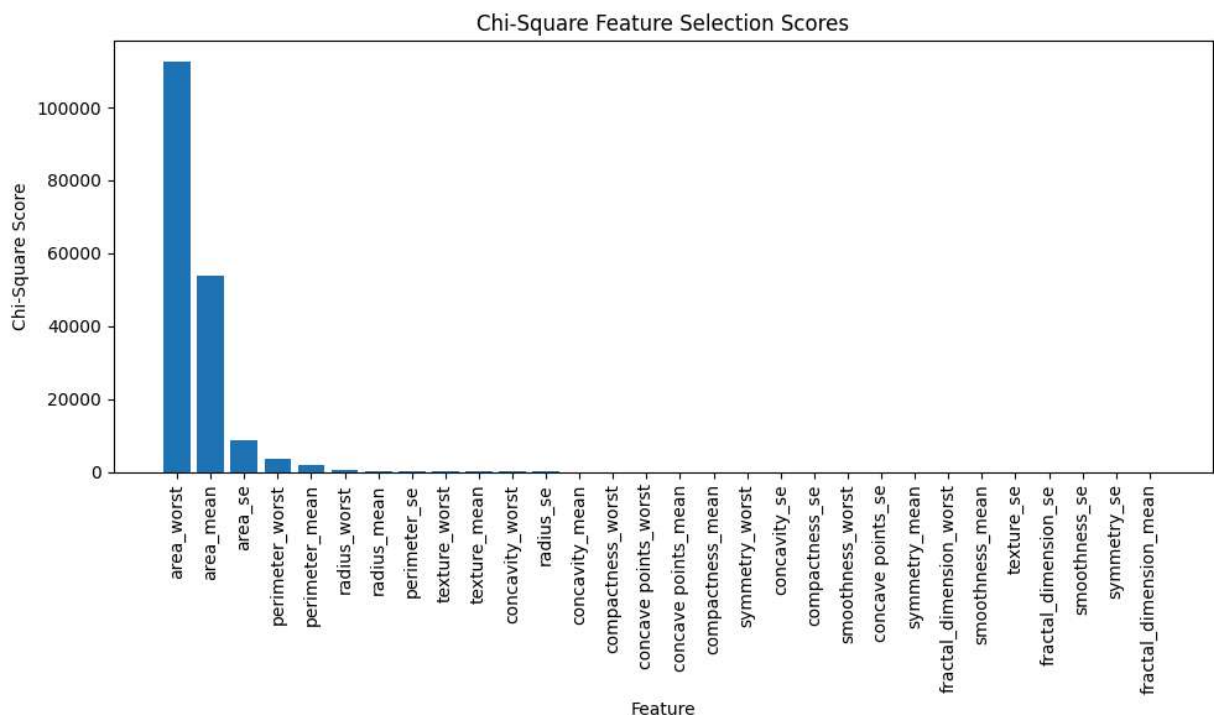
```

```

In [21]: data = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
data = data.sort_values(by='Score', ascending=False)

plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("Chi-Square Score")
plt.title("Chi-Square Feature Selection Scores")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```



```

In [24]: from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import warnings

```

```

In [25]: model = LogisticRegression(max_iter=500)

```

```
In [29]: rfe = RFE(model, n_features_to_select=15)
rfe.fit(X, y)
print("Selected Features:", rfe.get_support())
print("Ranking:", rfe.ranking_)
```

```
C:\Users\M\anaconda3\envs\Code1\Lib\site-packages\sklearn\linear_model\_logistic.py:
406: ConvergenceWarning: lbfgs failed to converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence (max_iter=500).
You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

```
C:\Users\M\anaconda3\envs\Code1\Lib\site-packages\sklearn\linear_model\_logistic.py:
406: ConvergenceWarning: lbfgs failed to converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence (max_iter=500).
You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

```
C:\Users\M\anaconda3\envs\Code1\Lib\site-packages\sklearn\linear_model\_logistic.py:
406: ConvergenceWarning: lbfgs failed to converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence (max_iter=500).
You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

```
C:\Users\M\anaconda3\envs\Code1\Lib\site-packages\sklearn\linear_model\_logistic.py:
406: ConvergenceWarning: lbfgs failed to converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence (max_iter=500).
You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

```
C:\Users\M\anaconda3\envs\Code1\Lib\site-packages\sklearn\linear_model\_logistic.py:
406: ConvergenceWarning: lbfgs failed to converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence (max_iter=500).
You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

```
C:\Users\M\anaconda3\envs\Code1\Lib\site-packages\sklearn\linear_model\_logistic.py:
406: ConvergenceWarning: lbfgs failed to converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence (max_iter=500).
You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
C:\Users\M\anaconda3\envs\Code1\Lib\site-packages\sklearn\linear_model_logistic.py:
406: ConvergenceWarning: lbfgs failed to converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

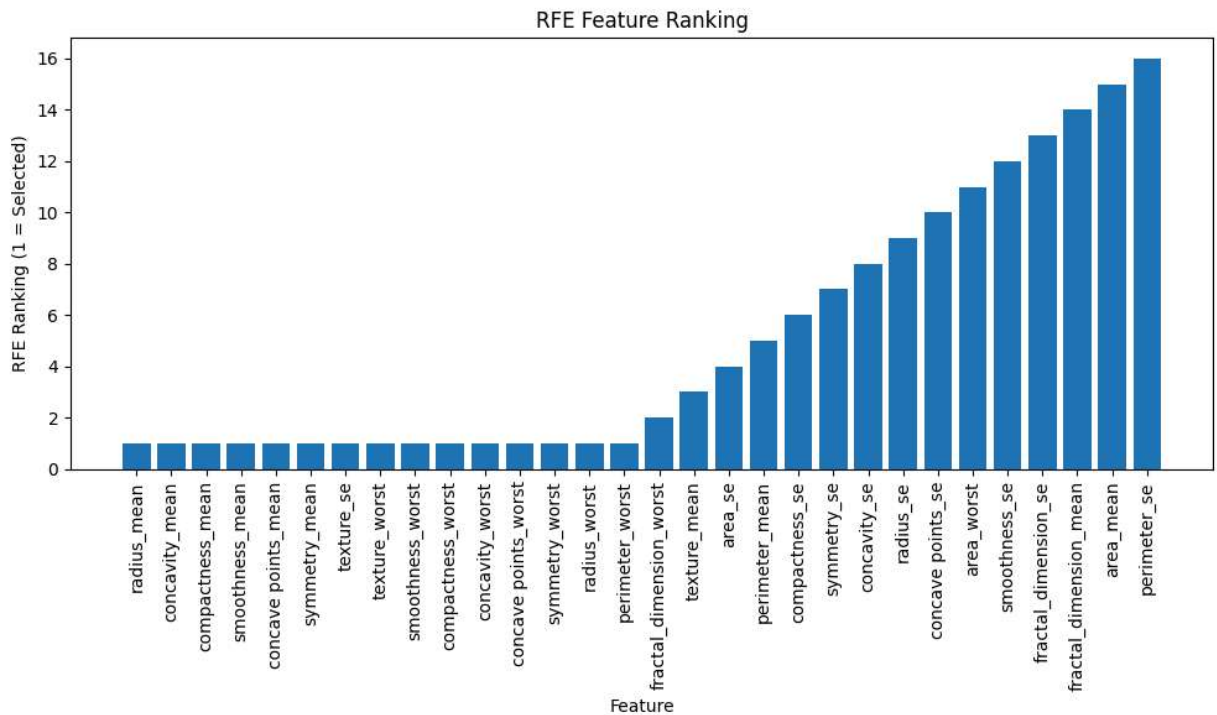
Increase the number of iterations to improve the convergence (max_iter=500).
You might also want to scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
Selected Features: [True False False False True True True True True False False
True
False False False False False False False False True True True False
True True True True True False]
Ranking: [1 3 5 15 1 1 1 1 1 14 9 1 16 4 12 6 8 10 7 13 1 1 1 11
1 1 1 1 1 2]

```
In [28]: import pandas as pd
```

```
In [34]: feature_ranking = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_
})
```

```
In [35]: feature_ranking = feature_ranking.sort_values(by='Ranking', ascending=True)
```

```
In [36]: plt.figure(figsize=(10, 6))
plt.bar(feature_ranking['Feature'], feature_ranking['Ranking'])
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("RFE Feature Ranking")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



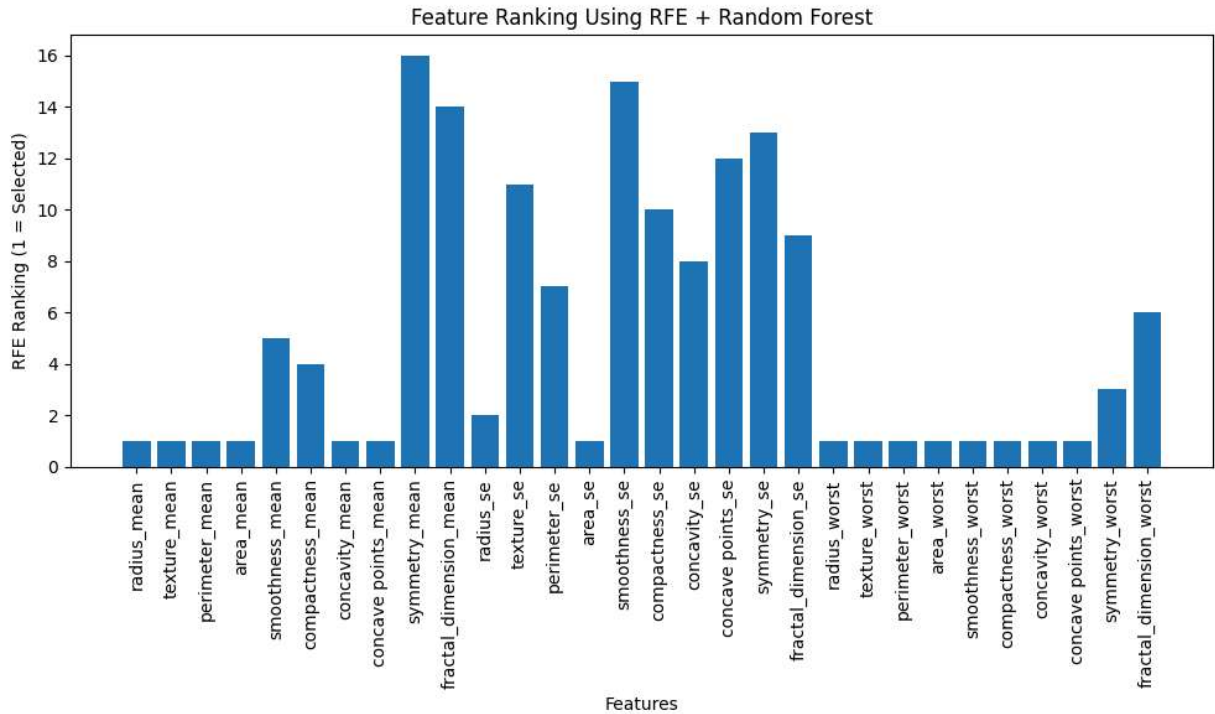
```
In [38]: from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rfe = RFE(estimator=rf, n_features_to_select=15)
rfe.fit(X, y)
print("Selected Features Mask:", rfe.get_support())
print("Ranking of Features:", rfe.ranking_)
```

```
Selected Features Mask: [ True  True  True  True False False  True  True False False
 False False
  False True False False False False False True  True  True  True
  True True True True False False]
Ranking of Features: [ 1  1  1  1  5  4  1  1 16 14  2 11  7  1 15 10  8 12 13  9  1
 1  1  1
 1  1  1  1  3  6]
```

```
In [39]: import pandas as pd
pdranking_df = pd.DataFrame({
    'Feature': X.columns,
    'RFE_Ranking': rfe.ranking_
}).sort_values('RFE_Ranking')
```

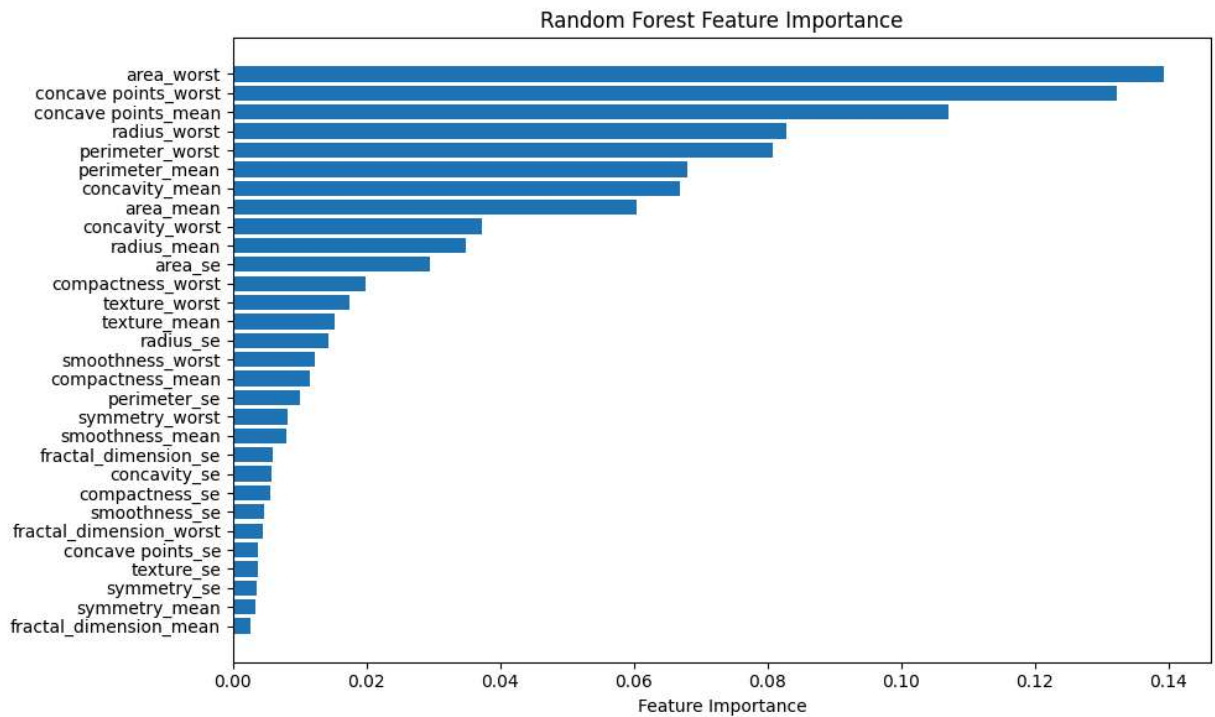
```
In [44]: ranking_df = pd.DataFrame({
    'Feature': X.columns,
    'RFE_Ranking': rfe.ranking_
})
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.bar(ranking_df['Feature'], ranking_df['RFE_Ranking'])
plt.xlabel("Features")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("Feature Ranking Using RFE + Random Forest")
plt.xticks(rotation=90)
```

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



```
In [46]: df = pd.read_csv('breast-cancer.csv')
```

```
In [52]: from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis'])
X = df.drop(['diagnosis', 'id'], axis=1)
y = df['diagnosis']
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)
importances = rf.feature_importances_
feature_names = X.columns if hasattr(X, "columns") else [f"Feature{i}" for i in range(X.shape[1])]
df_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})
df_importance = df_importance.sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10, 6))
plt.barh(df_importance['Feature'][::-1], df_importance['Importance'][::-1])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()
```



```
In [62]: # Comparing models using features selected by the chi-square
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

```
In [66]: from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
# Make sure X and y exist first
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
k = 15
selector = SelectKBest(chi2, k=k)
X_train_k = selector.fit_transform(X_train, y_train)
X_test_k = selector.transform(X_test)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", list(selected_features))

models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
```

```
Selected Features: ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'concavity_mean', 'radius_se', 'perimeter_se', 'area_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'compactness_worst', 'concavity_worst', 'concave points_worst']
```

```

-----
NameError                                Traceback (most recent call last)
Cell In[66], line 16
      9 selected_features = X.columns[selector.get_support()]
     10 print("Selected Features:", list(selected_features))
     13 models = {
     14     "Random Forest": RandomForestClassifier(n_estimators=200,
     15 random_state=42),
--> 16     "Decision Tree": DecisionTreeClassifier(random_state=42),
     17     "SVM": SVC(kernel='rbf')
     18 }

NameError: name 'DecisionTreeClassifier' is not defined

```

```

In [73]: from sklearn.ensemble import RandomForestClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.svm import SVC
        models = {
            "Random Forest": RandomForestClassifier(n_estimators=200,
            random_state=42),
            "Decision Tree": DecisionTreeClassifier(random_state=42),
            "SVM": SVC(kernel='rbf')
        }
        metrics = []
        for name, model in models.items():
            model.fit(X_train_k, y_train)
            y_pred = model.predict(X_test_k)

            metrics.append({
                "Model": name,
                "Accuracy": accuracy_score(y_test, y_pred),
                "Precision": precision_score(y_test, y_pred, average='weighted'),
                "Recall": recall_score(y_test, y_pred, average='weighted')
            })

        results_df = pd.DataFrame(metrics)
        print("\nModel Performance:")
        print(results_df)

```

Model Performance:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.964912	0.965205	0.964912
1	Decision Tree	0.921053	0.921213	0.921053
2	SVM	0.947368	0.951470	0.947368

```

In [75]: # Comparing models using features selected by the Random Forest Feature Importance
        import numpy as np
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        k = 18

        keeprf = RandomForestClassifier(n_estimators=200, random_state=42)
        rf.fit(X_train, y_train)

```

```

importances = rf.feature_importances_
sorted_idx = np.argsort(importances)[::-1]
top_k_idx = sorted_idx[:k]

selected_features = X.columns[top_k_idx]
print("Selected Features by RF Importance:", list(selected_features))

```

Selected Features by RF Importance: ['area_worst', 'concave points_worst', 'concave points_mean', 'radius_worst', 'concavity_mean', 'perimeter_worst', 'perimeter_mean', 'radius_mean', 'area_mean', 'concavity_worst', 'area_se', 'texture_worst', 'compactness_worst', 'radius_se', 'compactness_mean', 'texture_mean', 'perimeter_se', 'smoothness_worst']

```

In [77]: models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = []
for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)
    metrics.append(
        ({ "Model": name,
          "Accuracy": accuracy_score(y_test, y_pred),
          "Precision": precision_score(y_test, y_pred,
          average='weighted'), "Recall": recall_score(y_test, y_pred, average='weighted')
        })
    )
results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RF-Selected Features:")
results_df

```

Model Performance Using RF-Selected Features:

Out[77]: —

```

In [3]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

```

```

In [4]: data = pd.read_csv("WineQT.csv")

```

```

In [5]: data.head()

```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	a
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	



In [6]: data.describe()

Out[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289



In [8]: data.isnull().sum()

Out[8]: fixed acidity 0
volatile acidity 0
citric acid 0
residual sugar 0
chlorides 0
free sulfur dioxide 0
total sulfur dioxide 0
density 0
pH 0
sulphates 0
alcohol 0
quality 0
Id 0
dtype: int64

In [15]: X = data.drop(['quality', 'Id'], axis=1)
y = data['quality']

X

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.50
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.60
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.60
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.50
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.50
...	...	...	...	...	...	...	...	...	...	...
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.70
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.80
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.50
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.70
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.70

1143 rows × 11 columns



In [14]: `from sklearn.preprocessing import StandardScaler`

`scaler = StandardScaler()`

`X = scaler.fit_transform(X)`

X

Out[14]: `array([[-0.52157961, 0.93933222, -1.36502663, ..., 1.27069495,
 -0.57365783, -0.96338181],
 [-0.29259344, 1.94181282, -1.36502663, ..., -0.70892755,
 0.1308811 , -0.59360107],
 [-0.29259344, 1.27349242, -1.16156762, ..., -0.32577481,
 -0.04525363, -0.59360107],
 ...,
 [-1.20853813, 0.38239855, -0.9581086 , ..., 0.88754221,
 -0.45623467, 0.05351522],
 [-1.38027776, 0.10393172, -0.8563791 , ..., 1.33455374,
 0.60057372, 0.70063152],
 [-1.38027776, 0.6330187 , -0.75464959, ..., 1.65384769,
 0.30701583, -0.22382033]], shape=(1143, 11))`

In [16]: `setsX_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.25, rand`

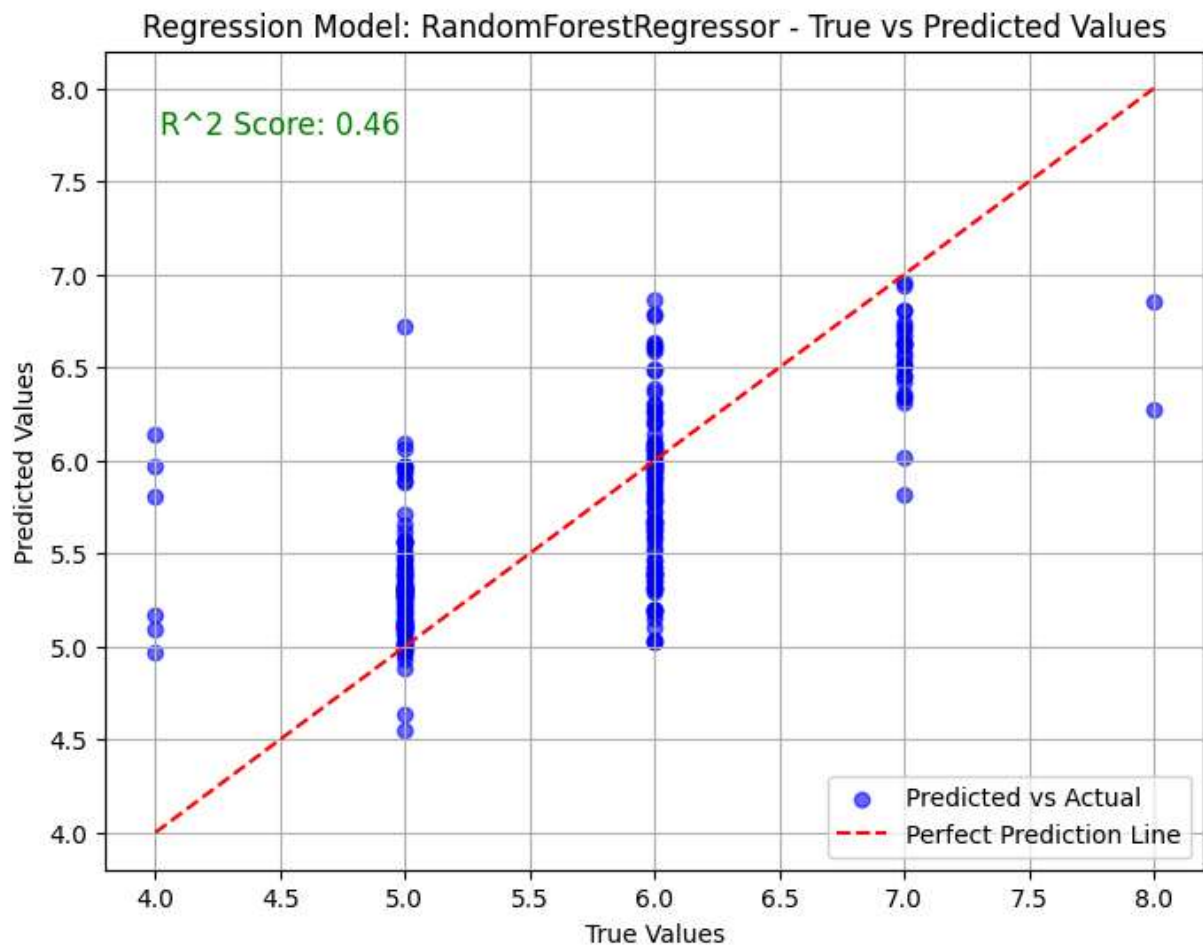
In [18]: `import numpy as np`

`import matplotlib.pyplot as plt`

```
In [19]: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

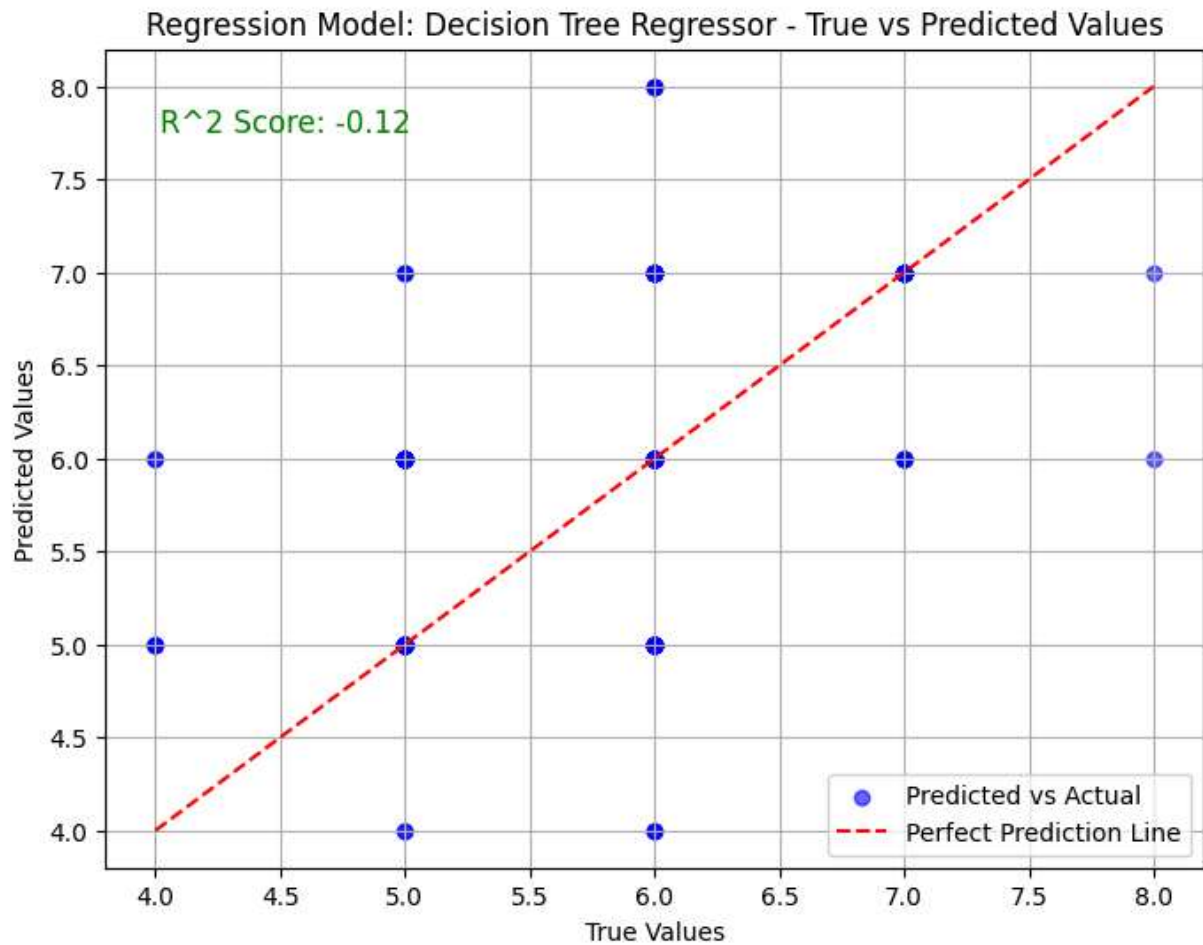
```
In [27]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
def evaluate_model(model_name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    results = {
        "Mean Absolute Error (MAE)": mae,
        "Mean Squared Error (MSE)": mse,
        "Root Mean Squared Error (RMSE)": rmse,
        "R^2 Score": r2
    }
    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
```

```
In [30]: from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
# Example: X = features, y = target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)
y_pred = rf_regressor.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)
print(evaluation_results)
```



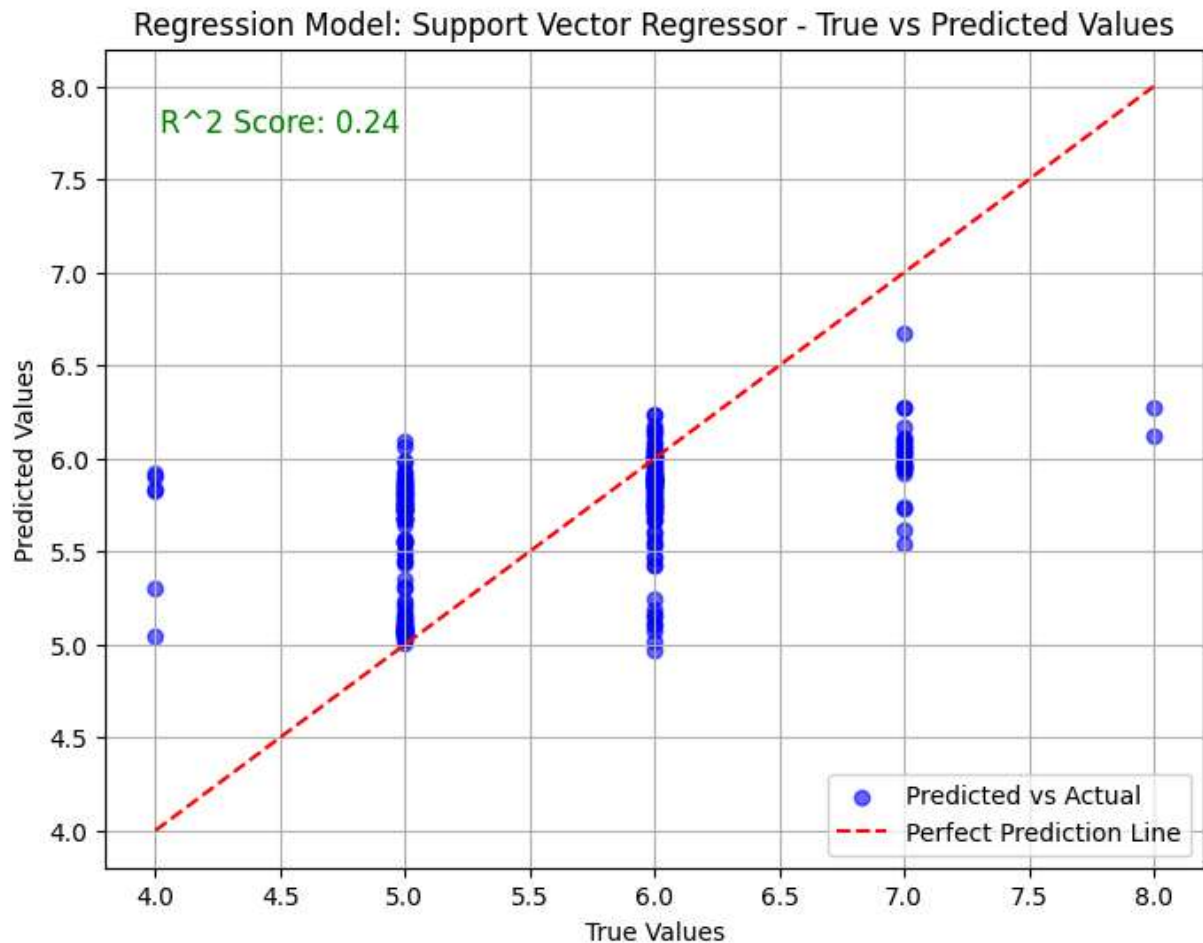
```
{'Mean Absolute Error (MAE)': 0.41082969432314403, 'Mean Squared Error (MSE)': 0.298
9554585152839, 'Root Mean Squared Error (RMSE)': np.float64(0.5467681944986229), 'R^
2 Score': 0.462767349736139}
```

```
In [32]: from sklearn.tree import DecisionTreeRegressor
DT = DecisionTreeRegressor(random_state=42)
DT.fit(X_train, y_train)
y_pred = DT.predict(X_test)
evaluation_results = evaluate_model('Decision Tree Regressor', y_test, y_pred)
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): 0.5109
Mean Squared Error (MSE): 0.6245
Root Mean Squared Error (RMSE): 0.7902
R² Score: -0.1222

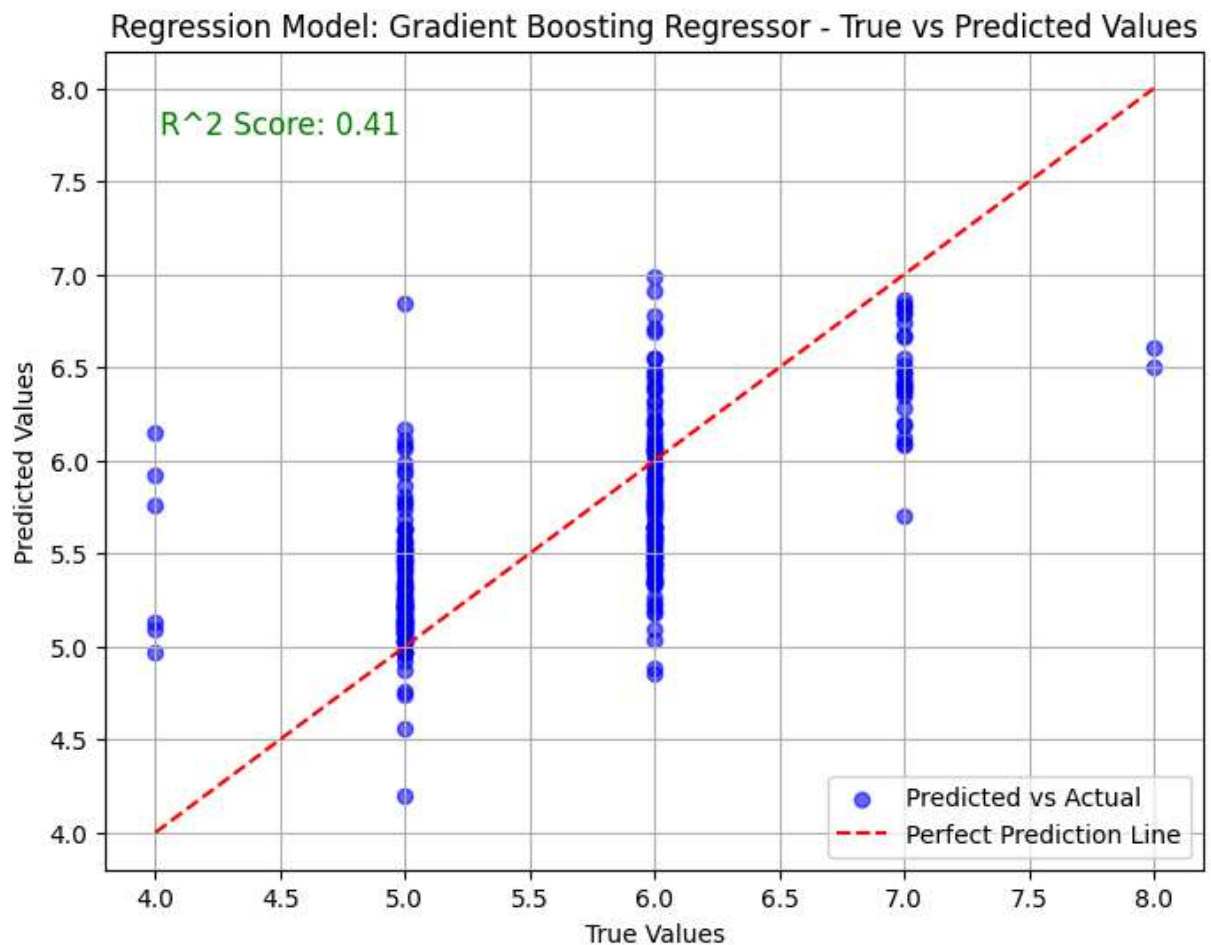
```
In [35]: from sklearn.svm import SVR
SVM = SVR()
SVM.fit(X_train, y_train)
y_pred = SVM.predict(X_test)
evaluation_results = evaluate_model('Support Vector Regressor',
y_test, y_pred)
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: \n{value}")
```



Mean Absolute Error (MAE): 0.4859
Mean Squared Error (MSE): 0.4233
Root Mean Squared Error (RMSE): 0.6506
R^2 Score: 0.2394

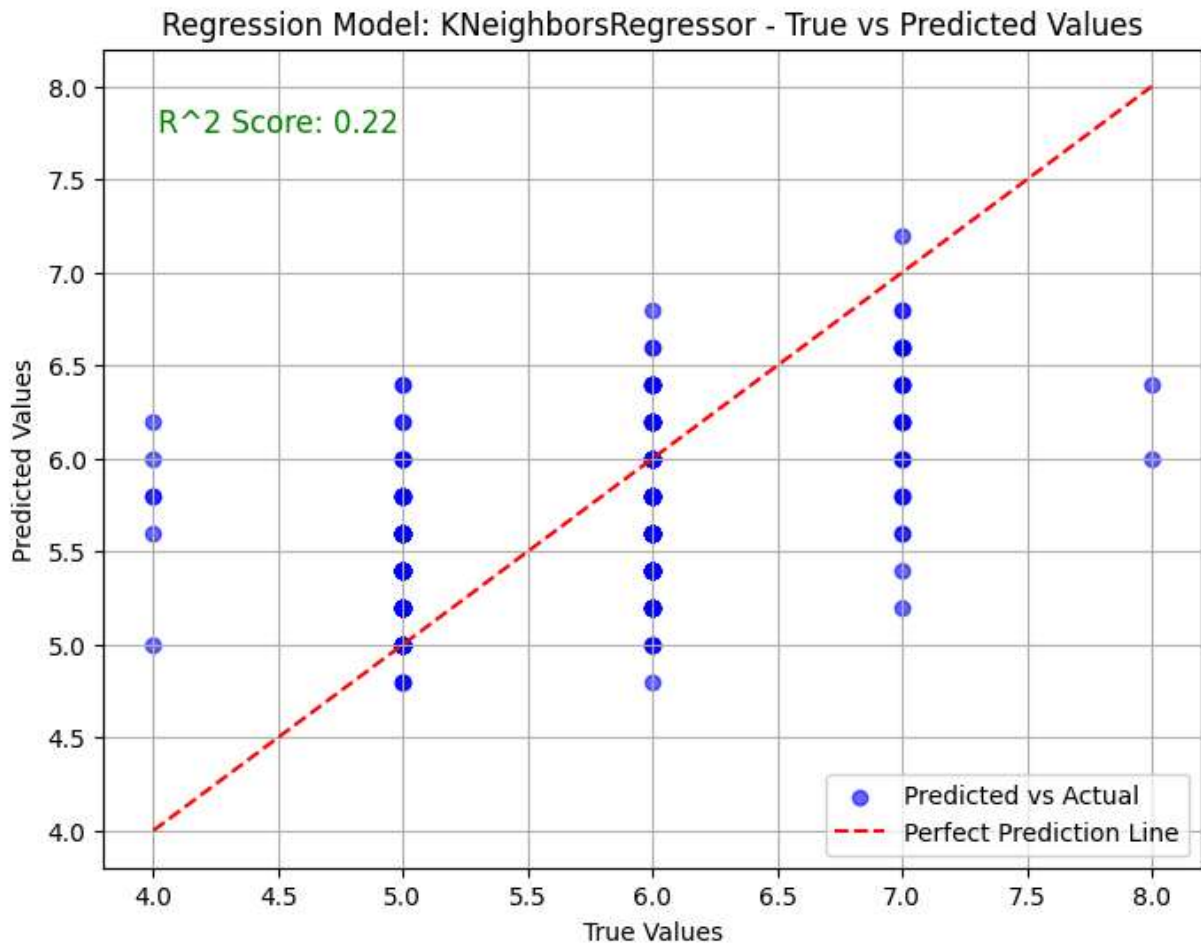
```
In [37]: from sklearn.ensemble import GradientBoostingRegressor
GBR = GradientBoostingRegressor(random_state=42)
GBR.fit(X_train, y_train)
y_pred = GBR.predict(X_test)
evaluation_results = evaluate_model('Gradient Boosting Regressor', y_test, y_pred)

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): 0.4458
Mean Squared Error (MSE): 0.3297
Root Mean Squared Error (RMSE): 0.5742
R^2 Score: 0.4075

```
In [40]: from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
evaluation_results = evaluate_model('KNeighborsRegressor', y_test, y_pred)
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: \n{value}")
```

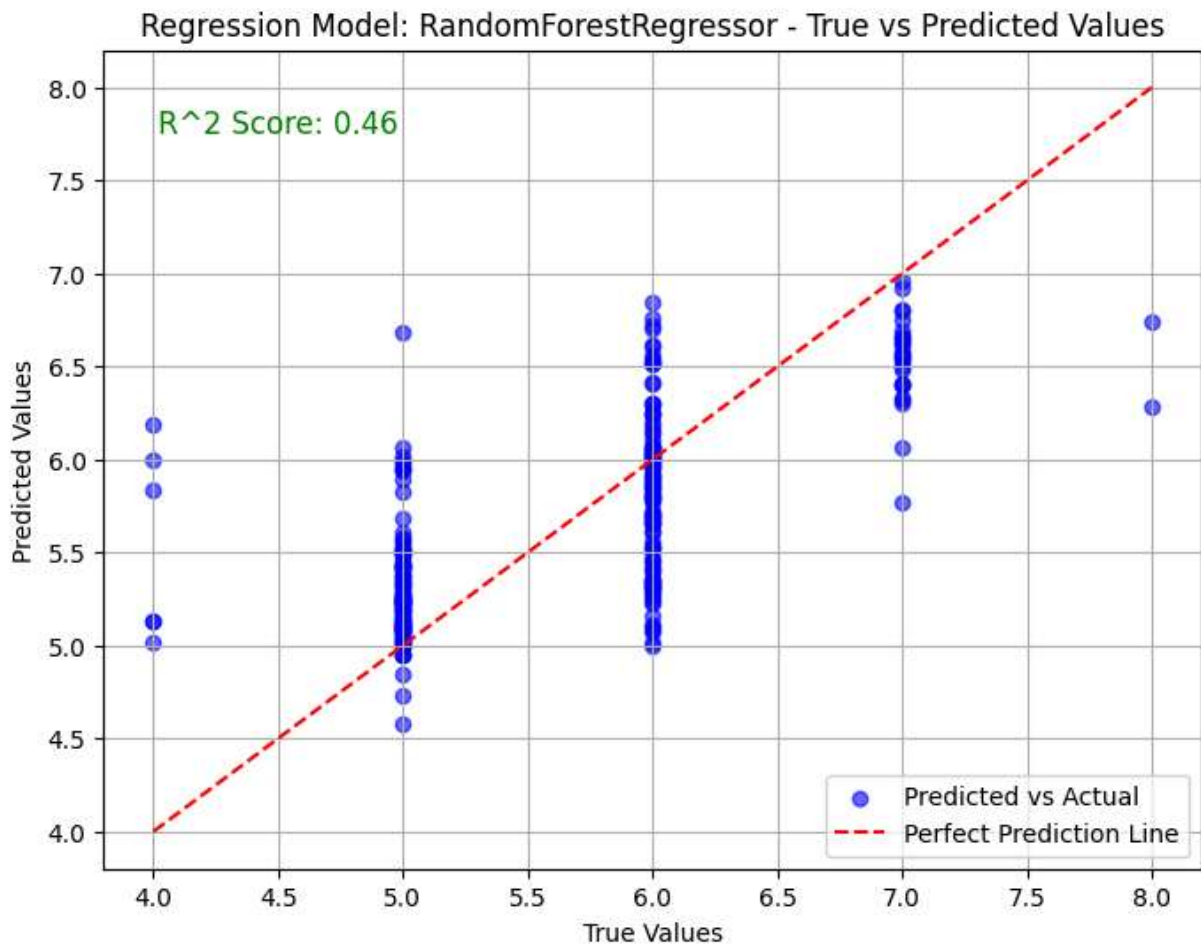


Mean Absolute Error (MAE): 0.5039
Mean Squared Error (MSE): 0.4330
Root Mean Squared Error (RMSE): 0.6580
R² Score: 0.2219

```
In [43]: from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=42)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
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)
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)
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): 0.4124

Mean Squared Error (MSE): 0.3014

Root Mean Squared Error (RMSE): 0.5490

R² Score: 0.4584

Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

```
In [ ]: from sklearn.model_selection import RandomizedSearchCV
        from scipy.stats import randint
        rf = RandomForestRegressor(random_state=42)
        param_distributions = {
            'n_estimators': randint(100, 500),
            'max_depth': [None, 10, 20, 30],
            'min_samples_split': randint(2, 10),
            'min_samples_leaf': randint(1, 5)
        }
```

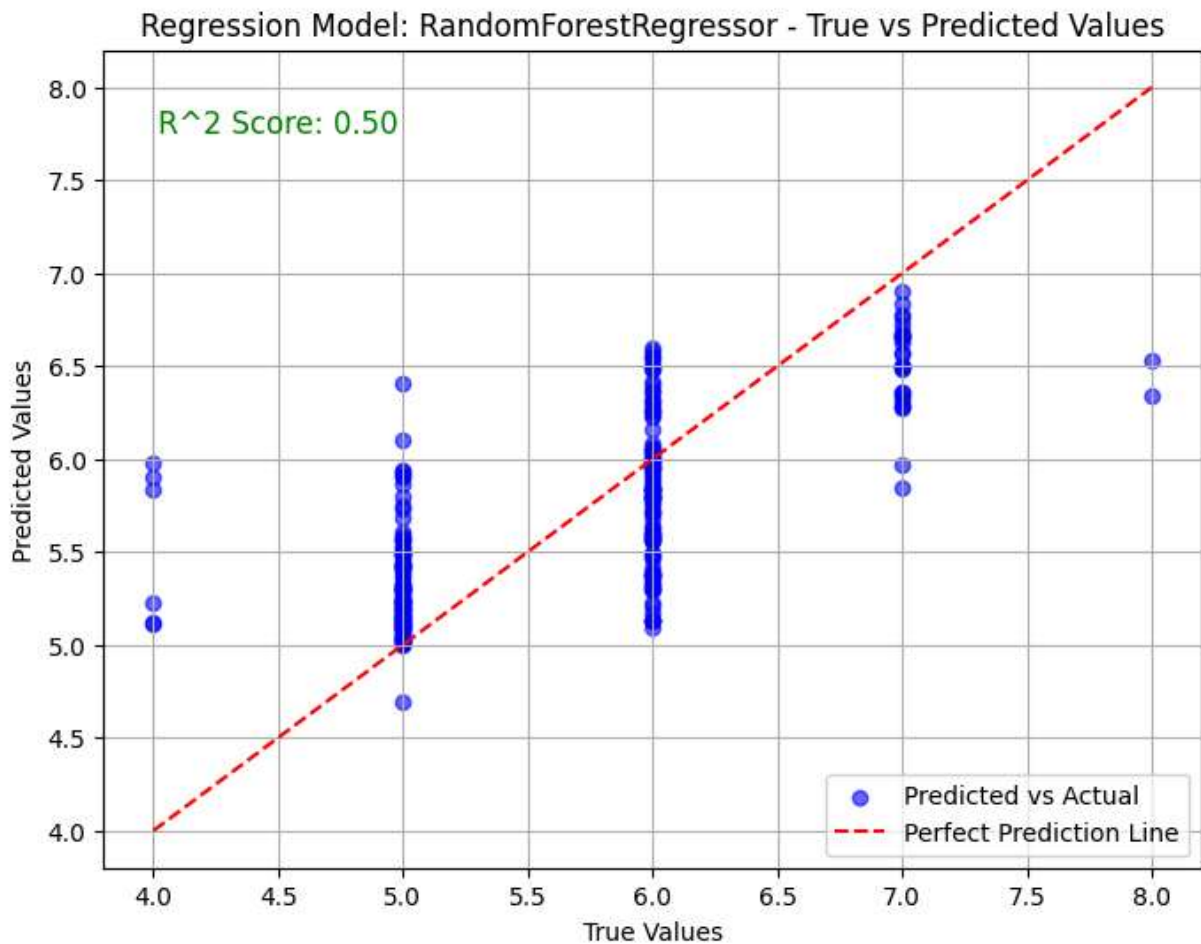
```
In [48]: from sklearn.model_selection import RandomizedSearchCV
         from sklearn.ensemble import RandomForestRegressor
         rf = RandomForestRegressor(random_state=42)
         param_distributions = {
             'n_estimators': [100, 200, 300, 500],
             'max_depth': [None, 5, 10, 20, 30],
```

```

    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None]
}
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_distributions,
    n_iter=20,
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    random_state=42,
    verbose=2
)
random_search.fit(X_train, y_train)
best_rf_random = random_search.best_estimator_
y_pred = best_rf_random.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}")
print("Best Parameters:", random_search.best_params_)

```

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

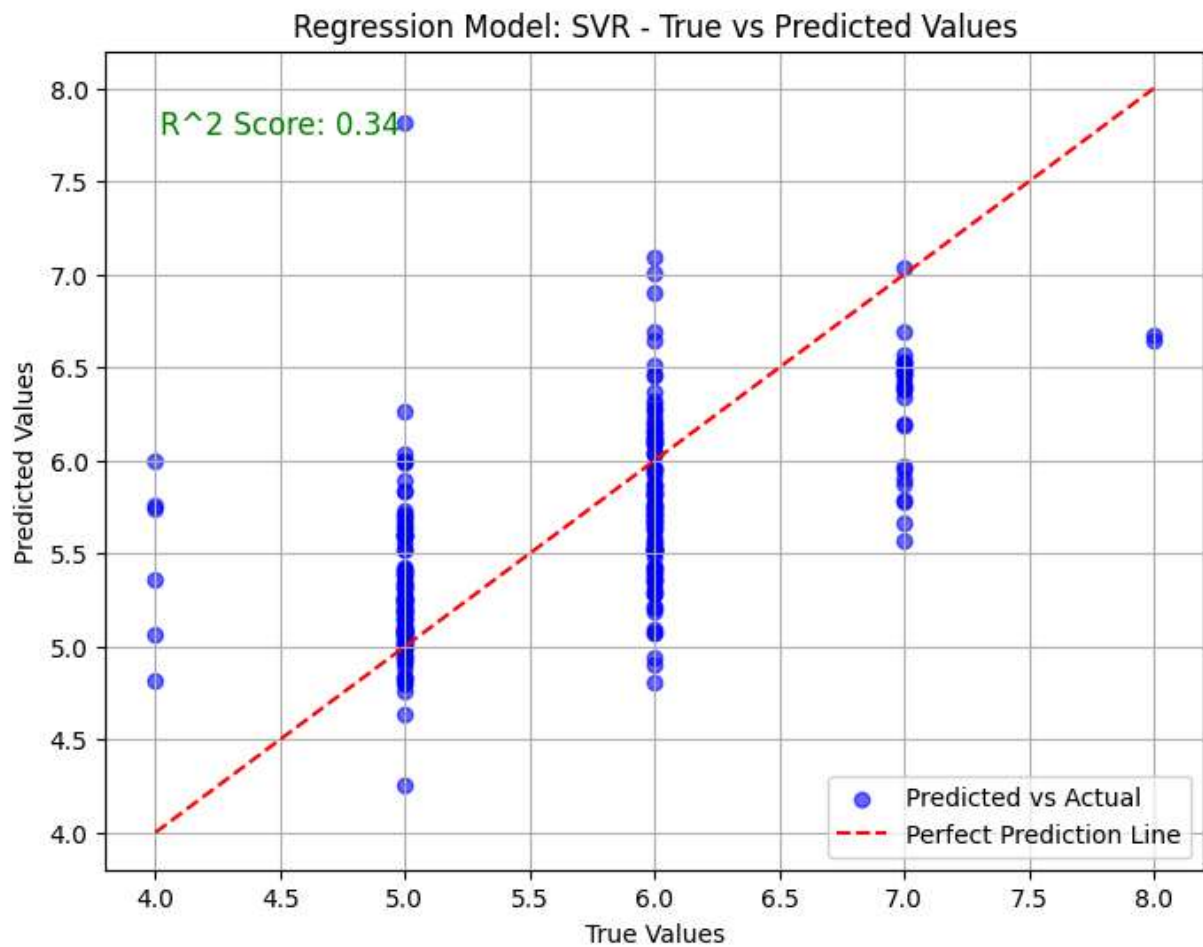


Mean Absolute Error (MAE): 0.4041
Mean Squared Error (MSE): 0.2804
Root Mean Squared Error (RMSE): 0.5295
R^2 Score: 0.4961
Best Parameters: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': 30}

```
In [49]: 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)
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): 0.4585
Mean Squared Error (MSE): 0.3694
Root Mean Squared Error (RMSE): 0.6078
R^2 Score: 0.3362

In []:

In []: