In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style()
```

In [2]:
```
# importing libraries for machine learning model evaluation
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

import warnings
warnings.filterwarnings('ignore')
sns.set_theme(style='darkgrid', palette='colorblind')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [3]:
```
df=pd.read_csv('FastFoodNutritionMenu.csv') #importing dataset
```

In [4]:
```
#data claening and preparation
df.head()
```

Out[4]:

	Company	Item	Calories	Calories from\nFat	Total Fat\n(g)	Saturated Fat\n(g)	Trans Fat\n(g)	Cholesterol\n(mg)
0	McDonald's	Hamburger	250	80	9	3.5	0.5	25
1	McDonald's	Cheeseburger	300	110	12	6	0.5	40
2	McDonald's	Double Cheeseburger	440	210	23	11	1.5	80
3	McDonald's	McDouble	390	170	19	8	1	65
4	McDonald's	Quarter Pounder® with Cheese	510	230	26	12	1.5	90

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1159 entries, 0 to 1158
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Company               1159 non-null   object
 1   Item                  1159 non-null   object
 2   Calories              1157 non-null   object
 3   Calories from
Fat     1098 non-null   object
 4   Total Fat
(g)         1101 non-null   object
 5   Saturated Fat
(g)     1101 non-null   object
 6   Trans Fat
(g)         1101 non-null   object
 7   Cholesterol
(mg)      1157 non-null   object
 8   Sodium
(mg)          1157 non-null   object
 9   Carbs
(g)             1101 non-null   object
 10  Fiber
(g)             1101 non-null   object
 11  Sugars
(g)            1157 non-null   object
 12  Protein
(g)           1027 non-null   object
 13  Weight Watchers
Pnts  524 non-null    object
dtypes: object(14)
memory usage: 126.9+ KB
```

In [6]:
```
df.dtypes
```

Out[6]:
```
Company                  object
Item                     object
Calories                 object
Calories from\nFat       object
Total Fat\n(g)           object
Saturated Fat\n(g)       object
Trans Fat\n(g)           object
Cholesterol\n(mg)        object
Sodium \n(mg)            object
Carbs\n(g)               object
Fiber\n(g)               object
Sugars\n(g)              object
Protein\n(g)             object
Weight Watchers\nPnts    object
dtype: object
```

In [7]:
```
df.describe().T
```

Out[7]:

	count	unique	top	freq
Company	1159	6	McDonald’s	330
Item	1159	1083	29 fl oz	11
Calories	1157	106	0	83
Calories from\nFat	1098	105	0	357
Total Fat\n(g)	1101	70	0	364
Saturated Fat\n(g)	1101	35	0	616
Trans Fat\n(g)	1101	52	0	739
Cholesterol\n(mg)	1157	153	0	182
Sodium \n(mg)	1157	243	0	85
Carbs\n(g)	1101	119	0	264
Fiber\n(g)	1101	95	0	426
Sugars\n(g)	1157	111	0	267
Protein\n(g)	1027	293	0	169
Weight Watchers\nPnts	524	356	0	29

In [8]:
```
df.shape
```

Out[8]:
```
(1159, 14)
```

In [9]:
```
df['Company'].value_counts()
```

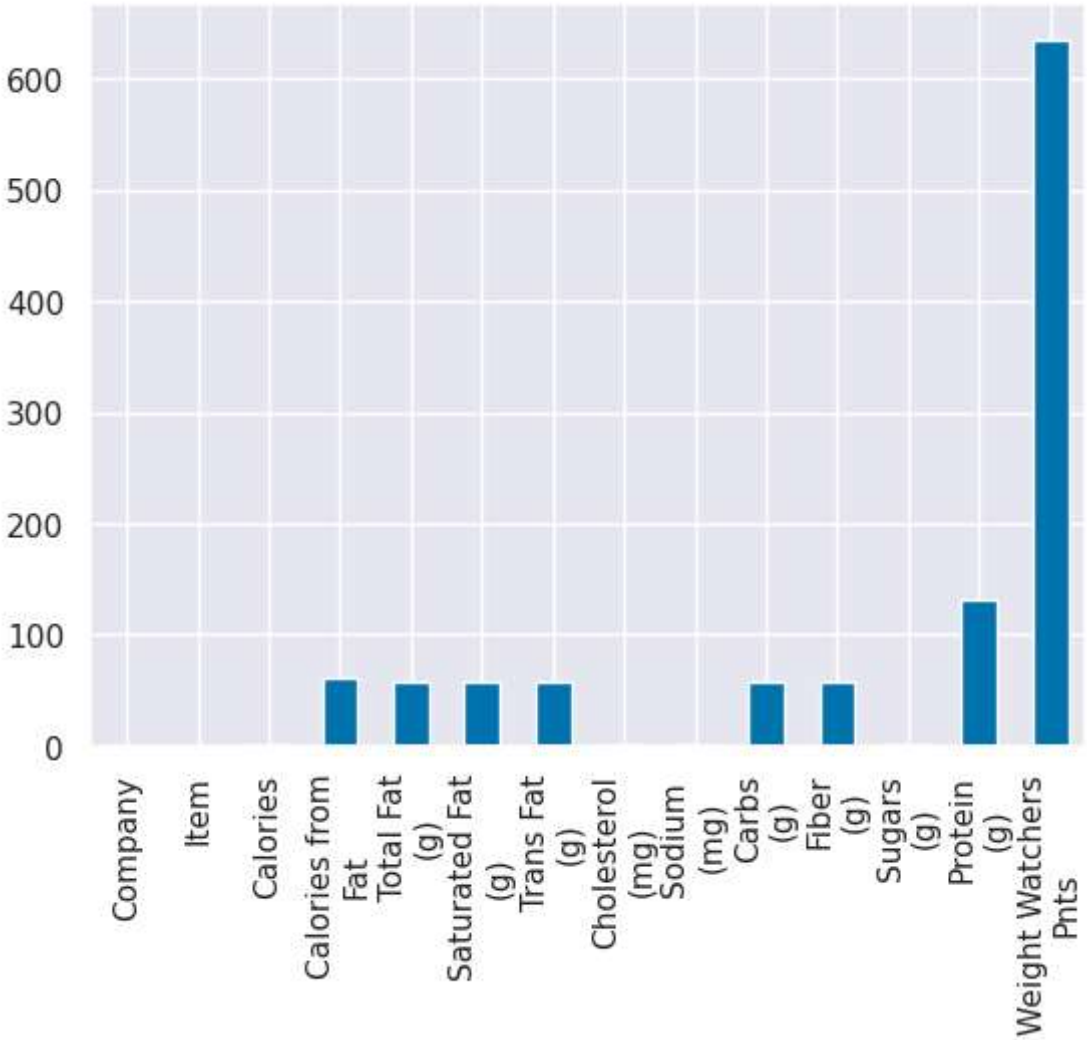
Out[9]:
```
McDonald’s     330
KFC            218
Burger King    199
Taco Bell      183
Wendy’s        155
Pizza Hut       74
Name: Company, dtype: int64
```

In [10]:
```
df.isna().sum()
```

Out[10]:
```
Company                    0
Item                       0
Calories                   2
Calories from\nFat         61
Total Fat\n(g)             58
Saturated Fat\n(g)         58
Trans Fat\n(g)             58
Cholesterol\n(mg)          2
Sodium \n(mg)              2
Carbs\n(g)                 58
Fiber\n(g)                 58
Sugars\n(g)                2
Protein\n(g)               132
Weight Watchers\nPnts      635
dtype: int64
```

In [11]:
```
df.isna().sum().plot(kind = 'bar')
```

Out[11]:
```
<Axes: >
```



REMOVE NULL

In [12]:
```
df = df.drop('Weight Watchers\nPnts', axis=1)
```

In [13]:
```
df=df.dropna()
```

REMOVE DUPLICATE

In [14]:
```
# Finding duplicate rows
duplicate_rows = df[df.duplicated(keep='first')]
# Number of duplicate rows
num_duplicates = duplicate_rows.shape[0]
# Displaying the duplicate rows
print(f"Number of duplicate rows: {num_duplicates}")
duplicate_rows
```

```
Number of duplicate rows: 7
```

Out[14]:

	Company	Item	Calories	Calories from\nFat	Total Fat\n(g)	Saturated Fat\n(g)	Trans Fat\n(g)	Cholesterol\n(mg
387	Burger King	Chicken Nuggets- 4pc	170	100	11	1.5	0	2
388	Burger King	Chicken Nuggets- 6pc	260	150	16	2.5	0	3
389	Burger King	Hamburger	240	90	10	3.5	0.5	3
390	Burger King	Cheeseburger	280	120	13	6	0.5	4
403	Burger King	Soft Serve Cone	190	40	4.5	3	0	2
442	Burger King	Fat FREE Milk (8 fl oz)	90	0	0	0	0	
443	Burger King	1% Low Fat Chocolate Milk (8 fl oz)	160	25	2.5	1.5	0	1

In [15]:
```
df = df.drop_duplicates()
```

In [16]:
```
df.isna().sum()
```

Out[16]:
```
Company                0
Item                   0
Calories               0
Calories from\nFat     0
Total Fat\n(g)         0
Saturated Fat\n(g)     0
Trans Fat\n(g)         0
Cholesterol\n(mg)      0
Sodium \n(mg)          0
Carbs\n(g)             0
Fiber\n(g)             0
Sugars\n(g)            0
Protein\n(g)           0
dtype: int64
```

In [ ]:
```

```

categorical

In [26]:
```
df['Company'] = le.fit_transform(df['Company'])
df['Item'] = le.fit_transform(df['Item'])
```

In [27]:
```
df.head()
```

Out[27]:

	Company	Item	Calories	Calories from\nFat	Total Fat\n(g)	Saturated Fat\n(g)	Trans Fat\n(g)	Cholesterol\n(mg)	Sodium \n(mg)
0	2	370	250.0	80.0	9.0	3.5	0.5	25.0	520.0
1	2	172	300.0	110.0	12.0	6.0	0.5	40.0	750.0
2	2	297	440.0	210.0	23.0	11.0	1.5	80.0	1150.0
3	2	566	390.0	170.0	19.0	8.0	1.0	65.0	920.0
4	2	0	510.0	230.0	26.0	12.0	1.5	90.0	1190.0

Top 5 Most Positively Correlated

In [28]:
```
print('Top 5 Most Positively Correlated to the Target Variable')
Corr_Matrix['Total Fat\n(g)'].sort_values(ascending=False).head(5)
```

```
Top 5 Most Positively Correlated to the Target Variable

Total Fat\n(g)        1.000000
Calories from\nFat    0.986373
Saturated Fat\n(g)    0.893753
Sodium \n(mg)         0.856355
Calories              0.773533
Name: Total Fat\n(g), dtype: float64
```

Top 5 Most Negatively Correlated

In [29]:
```
print('Top 5 Most Negatively Correlated to the Target Variable')
Corr_Matrix['Total Fat\n(g)'].sort_values(ascending=True).head(5)
```

```
Top 5 Most Negatively Correlated to the Target Variable

Fiber\n(g)          -0.192907
Protein\n(g)        -0.143426
Trans Fat\n(g)      -0.010390
Sugars\n(g)          0.003783
Cholesterol\n(mg)    0.031759
Name: Total Fat\n(g), dtype: float64
```

DROP LOW Correlated

In [30]:
```
columns_to_drop = [col for col in Corr_Matrix.columns if abs(Corr_Matrix.loc[
columns_to_drop
```

```
['Trans Fat\n(g)',
 'Cholesterol\n(mg)',
 'Carbs\n(g)',
 'Fiber\n(g)',
 'Sugars\n(g)',
 'Protein\n(g)']
```

In [31]:
```
df = df.drop(columns_to_drop, axis=1)
df.shape
```

```
(958, 7)
```

In [32]:
```
df=df.dropna()
```

split

In [33]:
```
X = df.drop(columns=['Total Fat\n(g)'])
y = df['Total Fat\n(g)']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
# Display the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (766, 6)
X_test shape: (192, 6)
y_train shape: (766,)
y_test shape: (192,)
```

MODEL

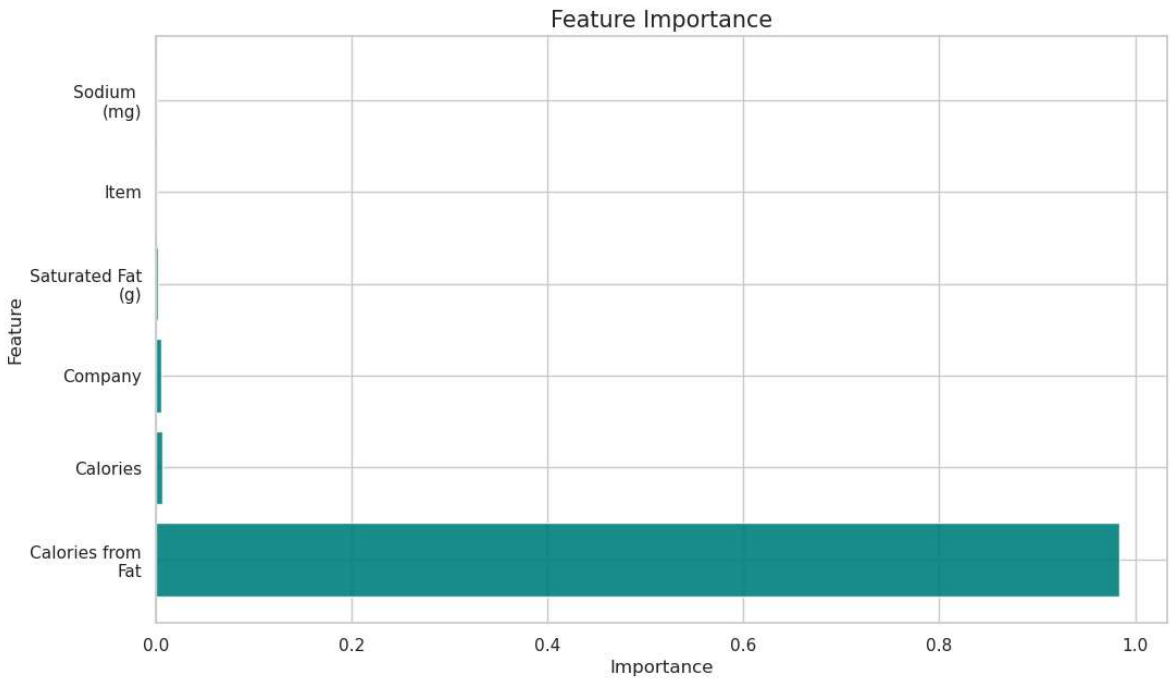
In [34]:
```
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42),
}
best_model = None
best_r2 = 0

for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred= model.predict(X_test)

    # Evaluate the model
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    submit = pd.DataFrame()
    submit['Actual Price'] = y_test
    submit['Predict_price'] = y_pred
    submit = submit.reset_index()
    print(submit.head(8))
    r2 = r2_score(y_test, y_pred)

    if r2 > best_r2:
        best_r2 = r2
        best_model = model.__class__.__name__

    print(f'{model_name}:')
    print(f'R2 Score: {r2:.2f}')
    print(f'Mean Absolute Error (MAE): {mae:.2f}')
    print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
    print('----------------------------------------')
print(f"The best performing model is: {best_model} with accuracy: {best_r2:.2
```

```
   index  Actual Price  Predict_price
0    885           0.0      -0.667609
1    505           0.0      -0.265401
2    356          52.0      50.771840
3    947          27.0      27.575316
4    905          12.0      12.392847
5    266           5.0       5.468171
6    219           0.0       0.449149
7    340          84.0      82.080301
Linear Regression:
R2 Score: 0.97
Mean Absolute Error (MAE): 1.18
Root Mean Squared Error (RMSE): 2.29
---------------------------------------
   index  Actual Price  Predict_price
0    885           0.0          0.000
1    505           0.0          0.000
2    356          52.0         51.340
3    947          27.0         26.930
4    905          12.0         12.010
5    266           5.0          5.215
6    219           0.0          0.000
7    340          84.0         77.470
Random Forest:
R2 Score: 0.98
Mean Absolute Error (MAE): 0.63
Root Mean Squared Error (RMSE): 1.81
---------------------------------------
   index  Actual Price  Predict_price
0    885           0.0      -0.002772
1    505           0.0      -0.079240
2    356          52.0      50.738376
3    947          27.0      26.716950
4    905          12.0      12.088734
5    266           5.0       5.186943
6    219           0.0      -0.087667
7    340          84.0      81.082459
Gradient Boosting:
R2 Score: 0.99
Mean Absolute Error (MAE): 0.68
Root Mean Squared Error (RMSE): 1.66
---------------------------------------
The best performing model is: GradientBoostingRegressor with accuracy: 0.99
```

feature_importances

In [35]:
```
importances = model.feature_importances_
feature_names = X.columns
feature_importance_dict = dict(zip(feature_names, importances))
sorted_feature_importance = sorted(feature_importance_dict.items(), key=lambda
for feature, importance in sorted_feature_importance:
    print(f"{feature}: {importance:.2f}")
plt.figure(figsize=(12, 7))
plt.barh(*zip(*sorted_feature_importance), alpha=0.9, color='teal')
plt.title('Feature Importance', fontsize=15)
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

```
Calories from
Fat: 0.98
Calories: 0.01
Company: 0.01
Saturated Fat
(g): 0.00
Item: 0.00
Sodium
(mg): 0.00
```



forward_selection with column (Total Fat\n(g))

In [36]:
```
import pandas as pd
import statsmodels.api as sm

# Your DataFrame
# df = ...

X = df.drop(columns=['Total Fat\n(g)'])
y = df['Total Fat\n(g)']

def forward_selection(df, target, significance_level=0.05):
    initial_features = df.columns.tolist()
    best_features = []
    while len(initial_features) > 0:
        remaining_features = list(set(initial_features) - set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.add_constant(df[best_features + [new_co
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if min_p_value < significance_level:
            best_features.append(new_pval.idxmin())
        else:
            break
    return best_features

# Assuming you have already defined X and y as the features and target variabl
selected_features = forward_selection(X, y)
print("Selected features:", selected_features)
```

```
Selected features: ['Calories from\nFat', 'Calories', 'Company', 'Saturated
Fat\n(g)', 'Sodium \n(mg)']
```

In [37]:
```
Selected_features = [ 'Saturated Fat\n(g)', 'Calories', 'Company', 'Sodium \n(

X = df[selected_features]
y = df['Total Fat\n(g)']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
# Display the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (766, 5)
X_test shape: (192, 5)
y_train shape: (766,)
y_test shape: (192,)
```

In [38]:
```
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42),
}
best_model = None
best_r2 = 0

for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred= model.predict(X_test)

    # Evaluate the model
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    submit = pd.DataFrame()
    submit['Actual Price'] = y_test
    submit['Predict_price'] = y_pred
    submit = submit.reset_index()
    print(submit.head(8))
    r2 = r2_score(y_test, y_pred)

    if r2 > best_r2:
        best_r2 = r2
        best_model = model.__class__.__name__

    print(f'{model_name}:')
    print(f'R2 Score: {r2:.2f}')
    print(f'Mean Absolute Error (MAE): {mae:.2f}')
    print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
    print('----------------------------------------')
print(f"The best performing model is: {best_model} with accuracy: {best_r2:.2
```

```
   index  Actual Price  Predict_price
0    885           0.0      -0.492373
1    505           0.0      -0.192654
2    356          52.0      50.650508
3    947          27.0      27.413670
4    905          12.0      12.428511
5    266           5.0       5.343815
6    219           0.0       0.535975
7    340          84.0      82.006517
Linear Regression:
R2 Score: 0.97
Mean Absolute Error (MAE): 1.17
Root Mean Squared Error (RMSE): 2.29
---------------------------------------
   index  Actual Price  Predict_price
0    885           0.0           0.00
1    505           0.0           0.00
2    356          52.0          51.45
3    947          27.0          26.90
4    905          12.0          12.15
5    266           5.0           5.01
6    219           0.0           0.00
7    340          84.0          78.56
Random Forest:
R2 Score: 0.98
Mean Absolute Error (MAE): 0.57
Root Mean Squared Error (RMSE): 1.76
---------------------------------------
   index  Actual Price  Predict_price
0    885           0.0      -0.057465
1    505           0.0      -0.036629
2    356          52.0      50.755482
3    947          27.0      26.674474
4    905          12.0      12.087299
5    266           5.0       5.049242
6    219           0.0      -0.067705
7    340          84.0      81.715817
Gradient Boosting:
R2 Score: 0.98
Mean Absolute Error (MAE): 0.69
Root Mean Squared Error (RMSE): 1.77
---------------------------------------
The best performing model is: RandomForestRegressor with accuracy: 0.98
```

forward_selection with colums(Calories from\nFat)

In [39]:
```
import pandas as pd
import statsmodels.api as sm

# Your DataFrame
# df = ...

X = df.drop(columns=['Calories from\nFat'])
y = df['Calories from\nFat']

def forward_selection(df, target, significance_level=0.05):
    initial_features = df.columns.tolist()
    best_features = []
    while len(initial_features) > 0:
        remaining_features = list(set(initial_features) - set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.add_constant(df[best_features + [new_co
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if min_p_value < significance_level:
            best_features.append(new_pval.idxmin())
        else:
            break
    return best_features

# Assuming you have already defined X and y as the features and target variabl
selected_features = forward_selection(X, y)
print("Selected features:", selected_features)
```

```
Selected features: ['Total Fat\n(g)', 'Saturated Fat\n(g)', 'Sodium \n(mg)',
'Calories', 'Company']
```

In [40]:
```
Selected_features = [ 'Total Fat\n(g)', 'Saturated Fat\n(g)', 'Sodium \n(mg)'

X = df[selected_features]
y = df['Calories from\nFat']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
# Display the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (766, 5)
X_test shape: (192, 5)
y_train shape: (766,)
y_test shape: (192,)
```

In [41]:
```
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42),
}
best_model = None
best_r2 = 0

for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred= model.predict(X_test)

    # Evaluate the model
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    submit = pd.DataFrame()
    submit['Actual Price'] = y_test
    submit['Predict_price'] = y_pred
    submit = submit.reset_index()
    print(submit.head(8))
    r2 = r2_score(y_test, y_pred)

    if r2 > best_r2:
        best_r2 = r2
        best_model = model.__class__.__name__

    print(f'{model_name}:')
    print(f'R2 Score: {r2:.2f}')
    print(f'Mean Absolute Error (MAE): {mae:.2f}')
    print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
    print('----------------------------------------')
print(f"The best performing model is: {best_model} with accuracy: {best_r2:.2
```

```
   index  Actual Price  Predict_price
0    885           0.0       2.648273
1    505           0.0       2.465423
2    356         460.0     457.797053
3    947         240.0     224.900147
4    905         110.0     107.722988
5    266          45.0      43.227027
6    219           0.0      -3.405984
7    340         750.0     754.568768
Linear Regression:
R2 Score: 0.98
Mean Absolute Error (MAE): 9.20
Root Mean Squared Error (RMSE): 17.84
----------------------------------------
   index  Actual Price  Predict_price
0    885           0.0           0.03
1    505           0.0           0.00
2    356         460.0         466.70
3    947         240.0         240.70
4    905         110.0         108.00
5    266          45.0          45.60
6    219           0.0           0.00
7    340         750.0         712.40
Random Forest:
R2 Score: 0.99
Mean Absolute Error (MAE): 4.18
Root Mean Squared Error (RMSE): 13.99
----------------------------------------
   index  Actual Price  Predict_price
0    885           0.0       0.024667
1    505           0.0       0.245821
2    356         460.0     470.367984
3    947         240.0     240.613812
4    905         110.0     110.845796
5    266          45.0      46.331870
6    219           0.0       0.123393
7    340         750.0     737.482903
Gradient Boosting:
R2 Score: 0.99
Mean Absolute Error (MAE): 4.19
Root Mean Squared Error (RMSE): 11.34
----------------------------------------
The best performing model is: GradientBoostingRegressor with accuracy: 0.99
```