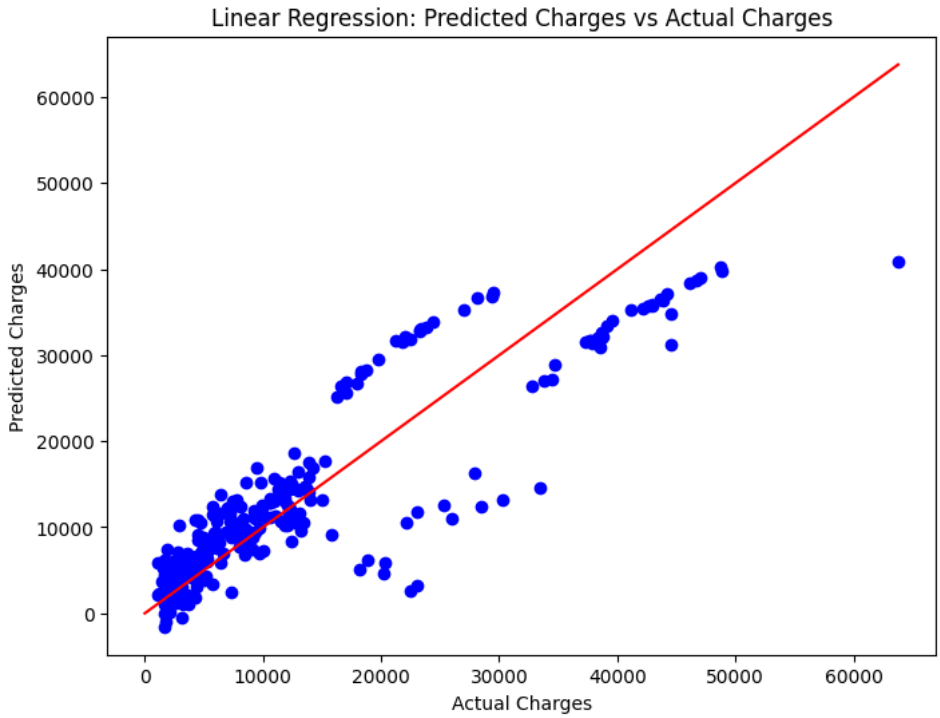
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, accuracy_score, precision_score, recall_score, f1_score


# Load the dataset
df = pd.read_csv('/content/health insurance.csv')

# Convert categorical variables to numerical using one-hot encoding
df = pd.get_dummies(df, columns=['sex', 'smoker', 'region'])


# Split the dataset into features (X) and target variable (y)
X = df.drop('charges', axis=1)
y = df['charges']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initialize the regression models
linear_model = LinearRegression()
ridge_model = Ridge()
lasso_model = Lasso()
decision_tree = DecisionTreeRegressor()
random_forest = RandomForestRegressor()
gradient_boosting = GradientBoostingRegressor()


# Fit the models
linear_model.fit(X_train, y_train)
ridge_model.fit(X_train, y_train)
lasso_model.fit(X_train, y_train)
decision_tree.fit(X_train, y_train)
random_forest.fit(X_train, y_train)
gradient_boosting.fit(X_train, y_train)
# Predict on the test set
linear_preds = linear_model.predict(X_test)
ridge_preds = ridge_model.predict(X_test)
lasso_preds = lasso_model.predict(X_test)
dt_preds = decision_tree.predict(X_test)
rf_preds = random_forest.predict(X_test)
gb_preds = gradient_boosting.predict(X_test)


# Compute performance metrics for each model
models = {
    'Linear Regression': linear_preds,
    'Ridge Regression': ridge_preds,
    'Lasso Regression': lasso_preds,
    'Decision Tree Regression': dt_preds,
    'Random Forest Regression': rf_preds,
    'Gradient Boosting Regression': gb_preds
}
metrics = {}

for model_name, y_pred in models.items():
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    metrics[model_name] = {
        'RMSE': rmse,
        'MAE': mae,
        'R2 Score': r2,
        'Accuracy': None,
        'Precision': None,
        'Recall': None,
        'F1-score': None
    }


# Calculate accuracy, precision, recall, and F1-score for each model
threshold = 20000  # Define a threshold to classify charges as high or low

for model_name, y_pred in models.items():
    # Convert the continuous predicted charges to binary (high/low) based on the threshold
    y_pred_binary = np.where(y_pred > threshold, 1, 0)
    y_test_binary = np.where(y_test > threshold, 1, 0)
```

```
    accuracy = accuracy_score(y_test_binary, y_pred_binary)
    precision = precision_score(y_test_binary, y_pred_binary)
    recall = recall_score(y_test_binary, y_pred_binary)
    f1 = f1_score(y_test_binary, y_pred_binary)

    metrics[model_name]['Accuracy'] = accuracy
    metrics[model_name]['Precision'] = precision
    metrics[model_name]['Recall'] = recall
    metrics[model_name]['F1-score'] = f1


# Print the metrics for each model
for model_name, metric_dict in metrics.items():
    print(model_name + ':')
    print('RMSE:', metric_dict['RMSE'])
    print('MAE:', metric_dict['MAE'])
    print('R2 Score:', metric_dict['R2 Score'])
    print('Accuracy:', metric_dict['Accuracy'])
    print('Precision:', metric_dict['Precision'])
    print('Recall:', metric_dict['Recall'])
    print('F1-score:', metric_dict['F1-score'])
    print('\n')
```

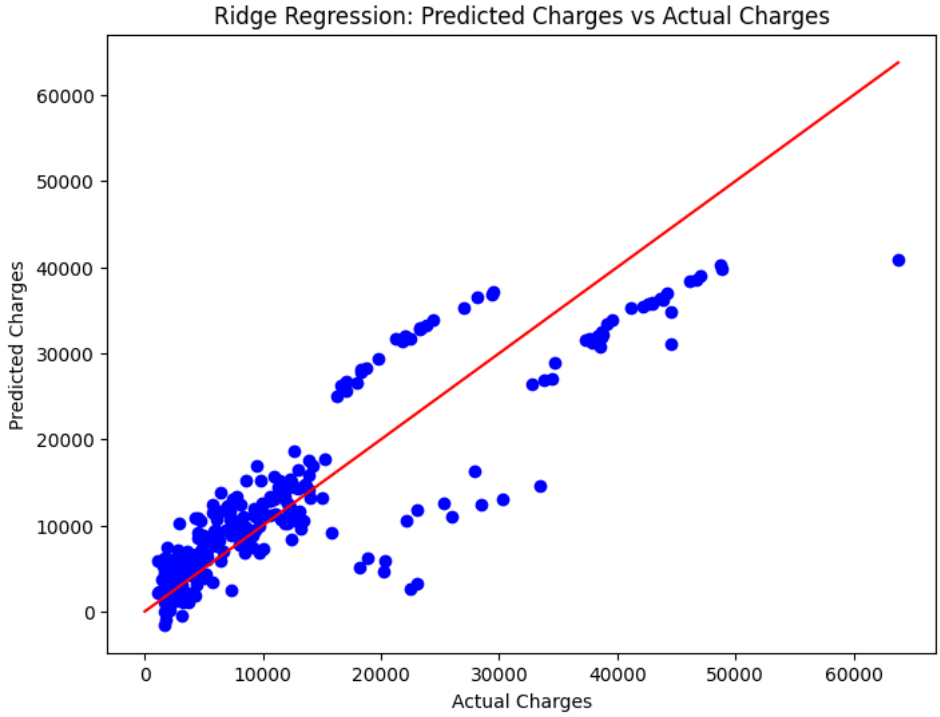
```
MAE: 4181.194473753643
R2 Score: 0.7835929767120723
Accuracy: 0.9216417910447762
Precision: 0.8333333333333334
Recall: 0.7894736842105263
F1-score: 0.8108108108108109


Ridge Regression:
RMSE: 5798.271036060347
MAE: 4187.302782980897
R2 Score: 0.7834446266673823
Accuracy: 0.9216417910447762
Precision: 0.8333333333333334
Recall: 0.7894736842105263
F1-score: 0.8108108108108109


Lasso Regression:
RMSE: 5797.062392287168
MAE: 4182.297003584162
R2 Score: 0.7835348987579266
Accuracy: 0.9216417910447762
Precision: 0.8333333333333334
Recall: 0.7894736842105263
F1-score: 0.8108108108108109


Decision Tree Regression:
RMSE: 6789.665578779052
MAE: 3341.4699602574624
R2 Score: 0.703059997793928
Accuracy: 0.8955223880597015
Precision: 0.7301587301587301
Recall: 0.8070175438596491
F1-score: 0.7666666666666667


Random Forest Regression:
RMSE: 4639.894885165047
MAE: 2545.0182216660764
R2 Score: 0.8613281774445832
Accuracy: 0.9440298507462687
Precision: 0.9565217391304348
Recall: 0.7719298245614035
F1-score: 0.8543689320388349


Gradient Boosting Regression:
RMSE: 4335.806918835916
MAE: 2411.4175931545183
R2 Score: 0.8789090161524343
Accuracy: 0.9402985074626866
Precision: 0.9361702127659575
Recall: 0.7719298245614035
F1-score: 0.8461538461538461
```

```
# Plot the predicted charges vs actual charges for linear regression
plt.figure(figsize=(8, 6))
plt.scatter(y_test, linear_preds, color='blue')
plt.plot([0, np.max(y)], [0, np.max(y)], color='red')
```

```
plt.title('Linear Regression: Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.show()
```

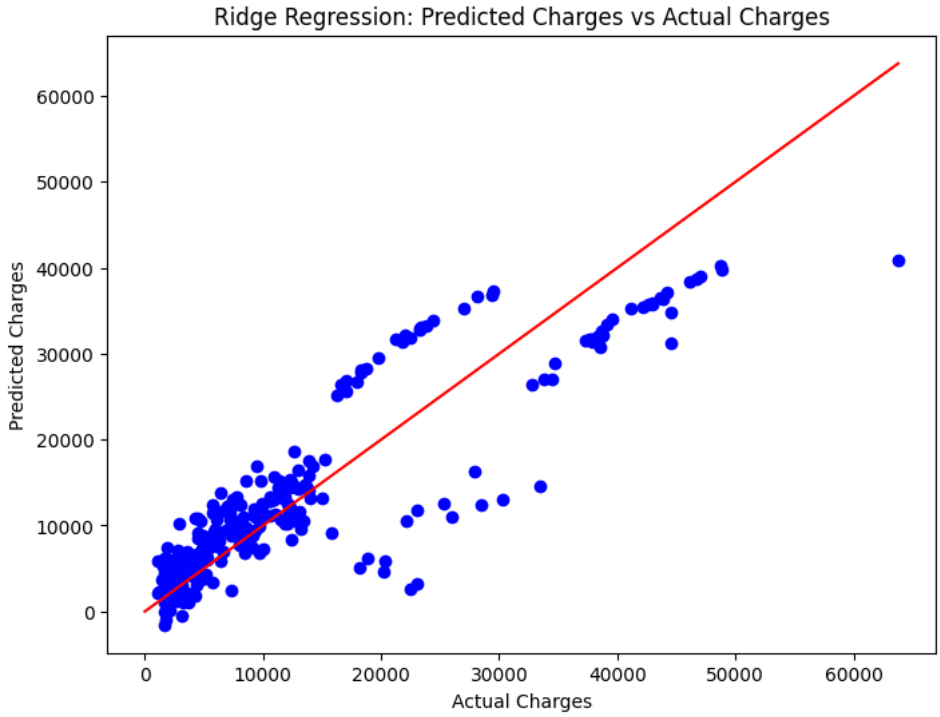



```
# Plot the predicted charges vs actual charges for Ridge Regression
plt.figure(figsize=(8, 6))
plt.scatter(y_test, ridge_preds, color='blue')
plt.plot([0, np.max(y)], [0, np.max(y)], color='red')
plt.title('Ridge Regression: Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.show()
```

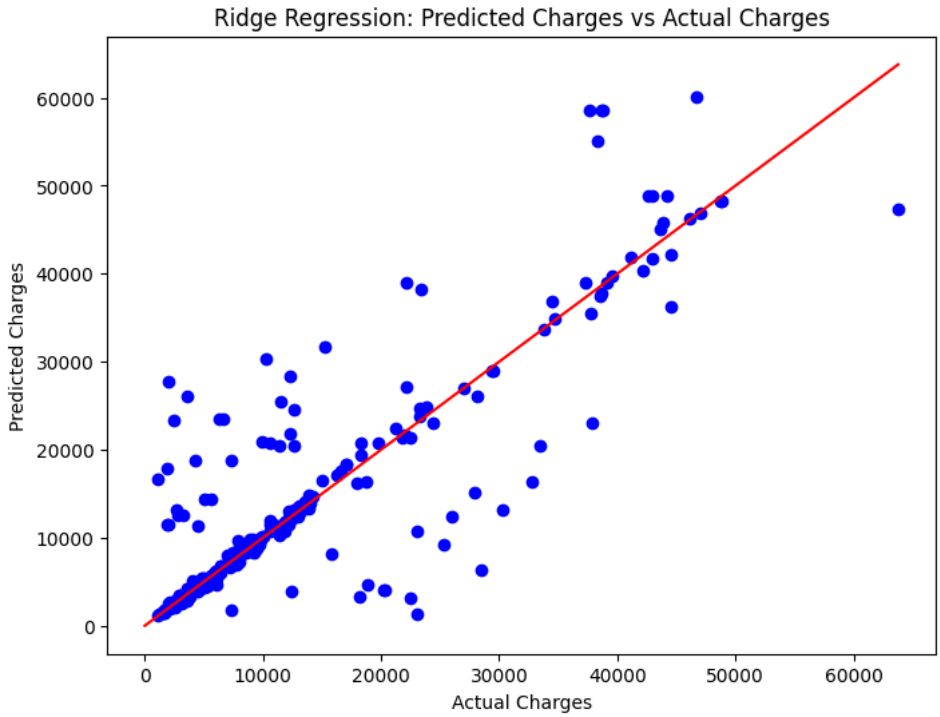



```
# Plot the predicted charges vs actual charges for Lasso Regression
plt.figure(figsize=(8, 6))
plt.scatter(y_test, lasso_preds, color='blue')
plt.plot([0, np.max(y)], [0, np.max(y)], color='red')
plt.title('Ridge Regression: Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
```

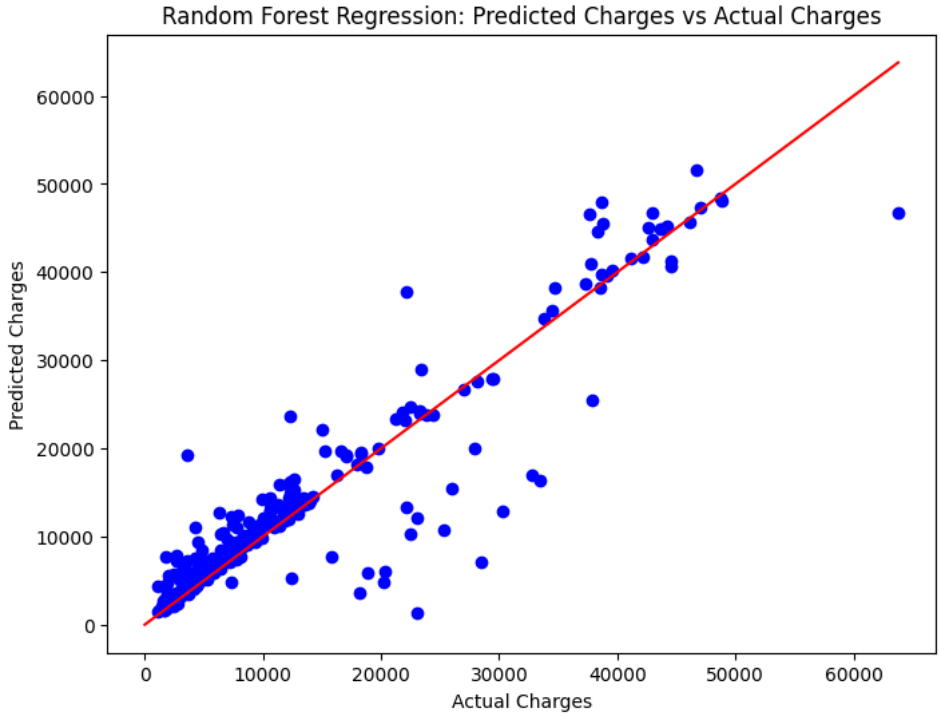
```
plt.ylabel('Predicted Charges')
plt.show()
```



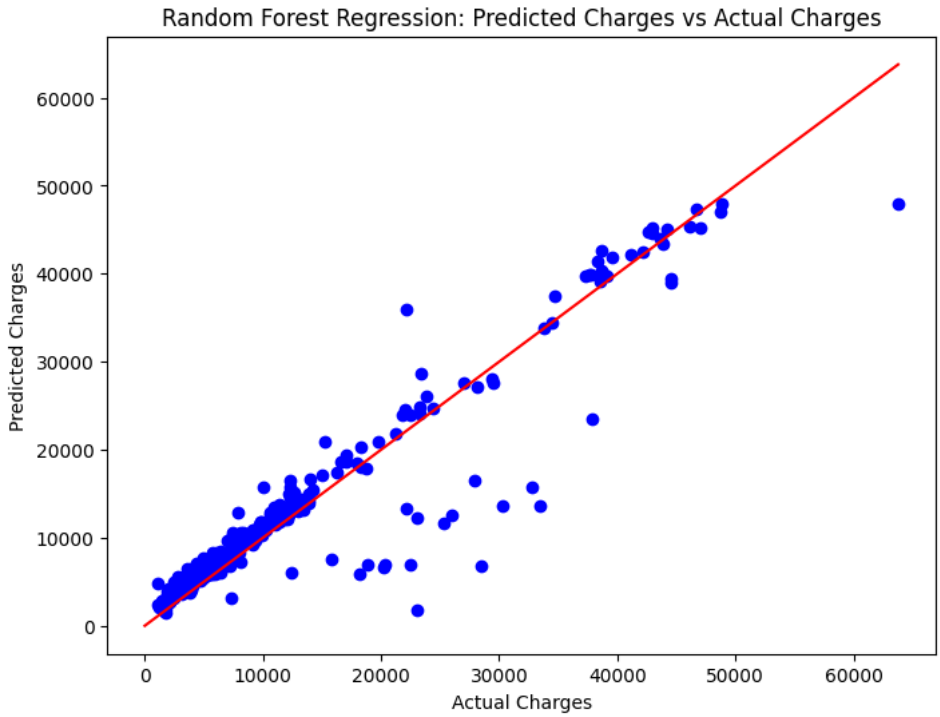

```
# Plot the predicted charges vs actual charges for Decision Tree Regression:
plt.figure(figsize=(8, 6))
plt.scatter(y_test, dt_preds, color='blue')
plt.plot([0, np.max(y)], [0, np.max(y)], color='red')
plt.title('Ridge Regression: Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.show()
```




```
# Plot the predicted charges vs actual charges for random forest regression
plt.figure(figsize=(8, 6))
plt.scatter(y_test, rf_preds, color='blue')
plt.plot([0, np.max(y)], [0, np.max(y)], color='red')
plt.title('Random Forest Regression: Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.show()
```

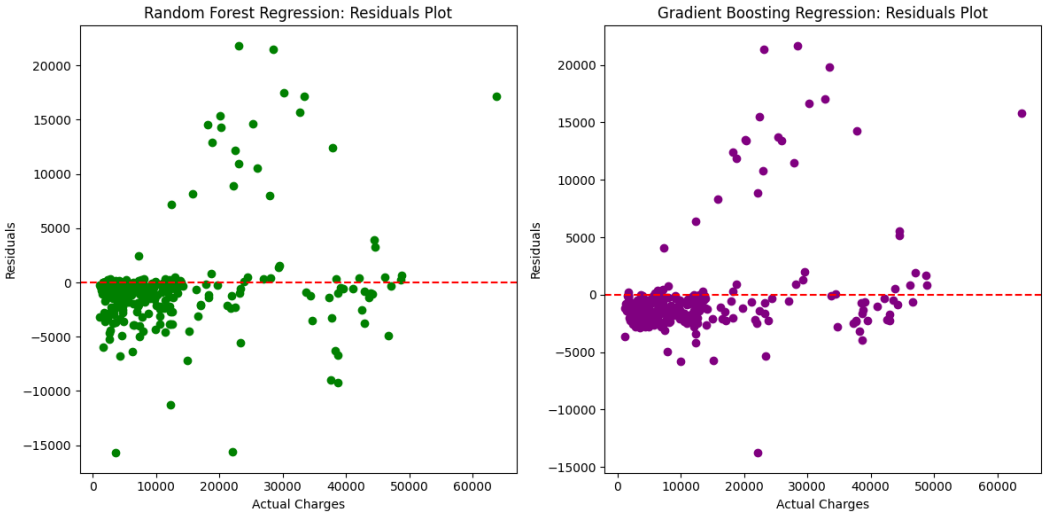



```
# Plot the predicted charges vs actual charges for Gradient Boosting regression
plt.figure(figsize=(8, 6))
plt.scatter(y_test, gb_preds, color='blue')
plt.plot([0, np.max(y)], [0, np.max(y)], color='red')
plt.title('Random Forest Regression: Predicted Charges vs Actual Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.show()
```

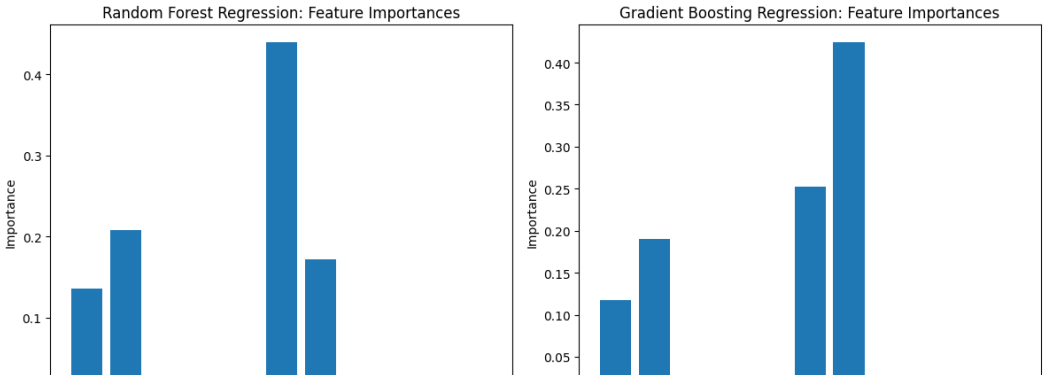



```
# Scatter plots of residuals for Random Forest and Gradient Boosting models
rf_residuals = y_test - rf_preds
gb_residuals = y_test - gb_preds

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_test, rf_residuals, color='green')
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Random Forest Regression: Residuals Plot')
plt.xlabel('Actual Charges')
plt.ylabel('Residuals')

plt.subplot(1, 2, 2)
```

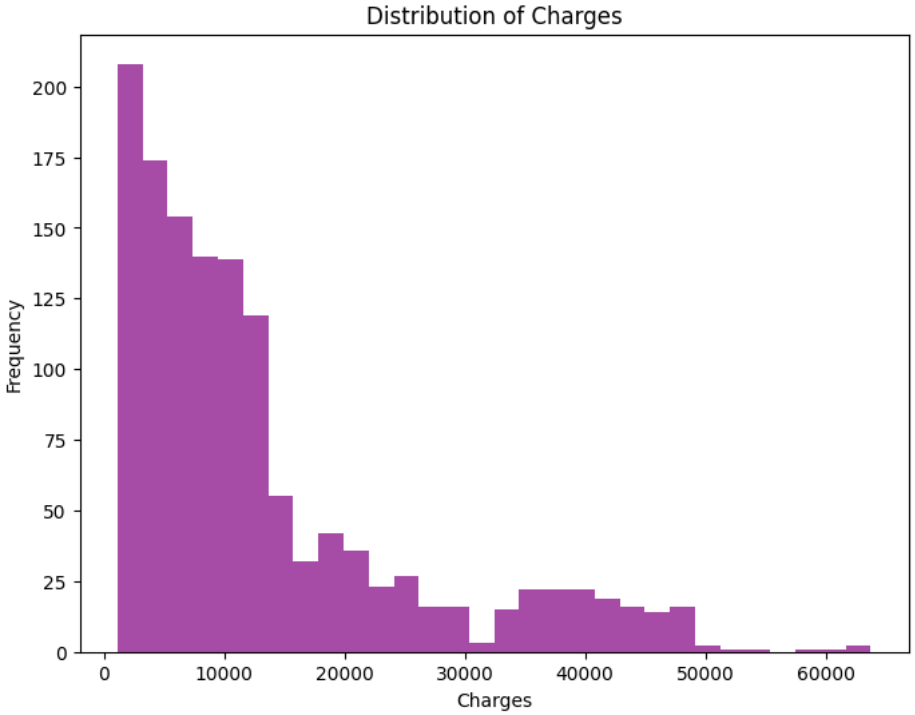
```
plt.scatter(y_test, gb_residuals, color='purple')
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Gradient Boosting Regression: Residuals Plot')
plt.xlabel('Actual Charges')
plt.ylabel('Residuals')

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



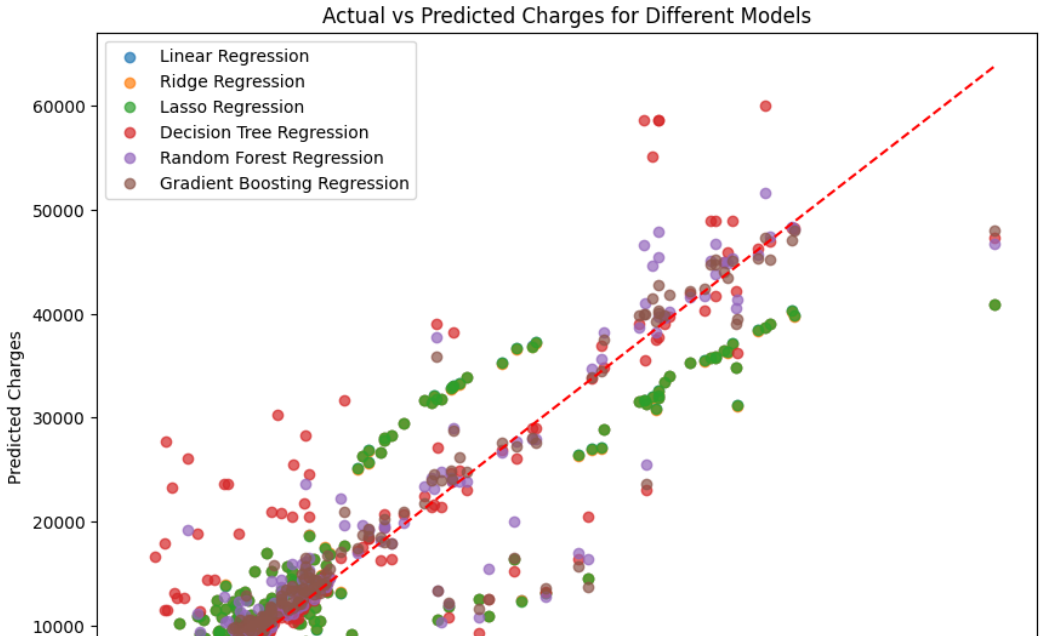

```
# Bar plot of feature importances for Random Forest and Gradient Boosting models
rf_feature_importances = random_forest.feature_importances_
gb_feature_importances = gradient_boosting.feature_importances_

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.bar(X.columns, rf_feature_importances)
plt.title('Random Forest Regression: Feature Importances')
plt.xlabel('Features')
plt.ylabel('Importance')

plt.xticks(rotation=45, ha='right')
plt.tight_layout()

plt.subplot(1, 2, 2)
plt.bar(X.columns, gb_feature_importances)
plt.title('Gradient Boosting Regression: Feature Importances')
plt.xlabel('Features')
plt.ylabel('Importance')

plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```




```
# Distribution of the target variable 'charges'
plt.figure(figsize=(8, 6))
plt.hist(y, bins=30, color='purple', alpha=0.7)
plt.title('Distribution of Charges')
plt.xlabel('Charges')
plt.ylabel('Frequency')
plt.show()
```




```
# Scatter plot of actual vs predicted charges for all models
plt.figure(figsize=(10, 8))
plt.scatter(y_test, linear_preds, label='Linear Regression', alpha=0.7)
plt.scatter(y_test, ridge_preds, label='Ridge Regression', alpha=0.7)
plt.scatter(y_test, lasso_preds, label='Lasso Regression', alpha=0.7)
plt.scatter(y_test, dt_preds, label='Decision Tree Regression', alpha=0.7)
plt.scatter(y_test, rf_preds, label='Random Forest Regression', alpha=0.7)
plt.scatter(y_test, gb_preds, label='Gradient Boosting Regression', alpha=0.7)

plt.plot([0, np.max(y)], [0, np.max(y)], color='red', linestyle='--')
plt.title('Actual vs Predicted Charges for Different Models')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.legend()
plt.show()
```




```
# Bar plot for model comparison of different metrics
metrics_df = pd.DataFrame.from_dict(metrics, orient='index')
metrics_df.plot(kind='bar', figsize=(10, 6))
plt.title('Model Comparison for Different Metrics')
plt.xlabel('Models')
plt.ylabel('Scores')
plt.xticks(rotation=45, ha='right')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```