```
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(r"C:\Users\Lenovo\Desktop\jkm\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')
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()
import pickle
with open('model1.pkl', 'wb') as file:
    pickle.dump(linear_model, file)

with open('model2.pkl', 'wb') as file:
    pickle.dump(ridge_model, file)
with open('model3.pkl', 'wb') as file:
    pickle.dump(lasso_model, file)
with open('model4.pkl', 'wb') as file:
    pickle.dump(decision_tree, file)
with open('model5.pkl', 'wb') as file:
    pickle.dump(random_forest , file)
with open('model6.pkl', 'wb') as file:
    pickle.dump(gradient_boosting, file)
```

```
import streamlit as st
import pickle
import pandas as pd
modell = pickle.load(open(r'C:\Users\Lenovo\Desktop\jkm\model1.pkl','rb'))
model2 = pickle.load(open(r'C:\Users\Lenovo\Desktop\jkm\model2.pkl','rb'))
model3 = pickle.load(open(r'C:\Users\Lenovo\Desktop\jkm\model3.pkl','rb'))
model4 = pickle.load(open(r'C:\Users\Lenovo\Desktop\jkm\model4.pkl','rb'))
model5 = pickle.load(open(r'C:\Users\Lenovo\Desktop\jkm\model5.pkl','rb'))
model6 = pickle.load(open(r'C:\Users\Lenovo\Desktop\jkm\model6.pkl','rb'))
def main():
    st.title('Health Insurance Prediction')

    # Sidebar inputs
    st.sidebar.header('User Inputs')

    age = st.sidebar.number_input('Age', min_value=0, max_value=120, value=30)

    sex=st.sidebar.selectbox('sex',['MALE','FEMALE'])

    bmi = st.sidebar.number_input('BMI', min_value=10, max_value=50, value=25)
    children = st.sidebar.number_input('Number of Children', min_value=0, max_value=10,
value=0)
    smoker = st.sidebar.selectbox('Smoker', ['No', 'Yes'])
    region = st.sidebar.selectbox('Region', ['Northeast', 'Northwest', 'Southeast', 'Southwest'])
    charges=st.sidebar.number_input('CHARGES', min_value=1000, max_value=70000,
value=1000)
    # Convert smoker to binary (0 or 1)
    smoker = 1 if smoker == 'Yes' else 0

    # Create a DataFrame for prediction
    input_data = pd.DataFrame({
        'age': [age],
        'bmi': [bmi],
        'children': [children],
        'smoker': [smoker],
        'region': [region],
        'charges': [charges],
    })
    # Display user inputs
    st.write('User Inputs:')
    st.write(input_data)

    # Make prediction
    prediction = modell.predict(input_data)
    prediction_probability = modell.predict_proba(input_data)

    # Display prediction
    st.subheader('Prediction')
    if prediction[0] == 1:
        st.write('The user is likely to have health insurance.')
    else:
        st.write('The user is unlikely to have health insurance.')

    # Display prediction probability
    st.subheader('Prediction Probability')
    st.write(f'Probability of having health insurance: {prediction_probability[0][1]:.2f}')

if __name__ == '__main__':
    main()
```