```
import pandas as pd

df = pd.read_csv("/content/heart_attack_prediction_dataset_edit.csv")
```

```
df.head()
```

	Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity	...	Sedentary Hours Per Day	Income	BMI	Tri
0	BMW7812	67.0	m	208	158/88	72	0.0	0.0	1	0	...	6.615001	261404.0	31.251233	
1	CZE1114	21.0	male	389	165/93	98	1.0	1.0	1	1	...	4.963459	285768.0	27.194973	
2	BNI9906	21.0	Female	324	174/99	72	1.0	0.0	0	0	...	9.463426	235282.0	28.176571	
3	JLN3497	84.0	NaN	383	163/100	73	1.0	1.0	1	0	...	7.648981	125640.0	36.464704	
4	GFO8847	66.0	Male	318	91/88	93	NaN	1.0	1	1	...	1.514821	160555.0	21.809144	

5 rows × 26 columns

```
df1 = df.dropna()
```

```
df = df.drop_duplicates(subset=['Patient ID'])
```

```
import numpy as np
# Assuming 'df' is your DataFrame and contains numeric columns you want to process.
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_filtered = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df_filtered

numeric_cols = df.select_dtypes(include=np.number).columns

for col in numeric_cols:
    df = remove_outliers_iqr(df, col)
```

```
df.isnull().sum()
```

```
                                0
              Patient ID        0
                     Age        2
                     Sex        1
             Cholesterol        0
          Blood Pressure        1
              Heart Rate        0
                Diabetes        1
          Family History        1
                 Smoking        0
                 Obesity        0
     Alcohol Consumption        0
 Exercise Hours Per Week        0
                    Diet        0
Previous Heart Problems         0
          Medication Use        0
            Stress Level        0
 Sedentary Hours Per Day        2
                  Income        1
                     BMI        0
           Triglycerides        0
Physical Activity Days Per Week 0
      Sleep Hours Per Day       1
                 Country        0
               Continent        0
              Hemisphere        0
       Heart Attack Risk        0

dtype: int64
```

```
# Create a list of columns to fill NA values
columns_to_fill = ['Age', 'Sedentary Hours Per Day','Income','Sleep Hours Per Day'] # Example columns, replace with your actual columns

# Fill NA values in specified columns with the mean of each column
for col in columns_to_fill:
    df[col] = df[col].fillna(df[col].mean())
```

Start coding or generate with AI.

```
# Create a list of columns to fill NA values
columns_to_fill = ['Diabetes','Family History','Sex'] # Example columns, replace with your actual columns

# Fill NA values in specified columns with the mean of each column
for col in columns_to_fill:
  # Calculate the mode of the column
  mode_value = df[col].mode()[0]

  # Replace NaN values with the mode
  df[col].fillna(mode_value, inplace=True)
```

```
<ipython-input-285-f5789353b791>:10: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df[col].fillna(mode_value, inplace=True)
```

```
df = df.dropna()
```

```
df.dtypes
```

	0
Patient ID	object
Age	float64
Sex	object
Cholesterol	int64
Blood Pressure	object
Heart Rate	int64
Diabetes	float64
Family History	float64
Smoking	int64
Obesity	int64
Alcohol Consumption	int64
Exercise Hours Per Week	float64
Diet	object
Previous Heart Problems	int64
Medication Use	int64
Stress Level	int64
Sedentary Hours Per Day	float64
Income	float64
BMI	float64
Triglycerides	int64
Physical Activity Days Per Week	int64
Sleep Hours Per Day	float64
Country	object
Continent	object
Hemisphere	object
Heart Attack Risk	int64

dtype: object

```
df.columns
```

```
Index(['Patient ID', 'Age', 'Sex', 'Cholesterol', 'Blood Pressure',
       'Heart Rate', 'Diabetes', 'Family History', 'Smoking', 'Obesity',
       'Alcohol Consumption', 'Exercise Hours Per Week', 'Diet',
       'Previous Heart Problems', 'Medication Use', 'Stress Level',
       'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides',
       'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
       'Continent', 'Hemisphere', 'Heart Attack Risk'],
      dtype='object')
```

```
category_cols = df.select_dtypes(include="object").columns

for col in category_cols:
    print(col)
    print(df[col].unique())
```

```
Patient ID
['BMW7812' 'CZE1114' 'JLN3497' ... 'MSV9918' 'XKA5925' 'EPE6801']
Sex
['m' 'male' 'Male' 'Female']
Blood Pressure
['158/88' '165/93' '163/100' ... '137/94' '94/76' '119/67']
Diet
['Average' 'Unhealthy' 'Healty' 'Healthy']
Country
['Argentina' 'Canada' 'Japan' 'Vietnam' 'China' 'Italy' 'Brazil'
 'Thailand' 'Spain' 'France' 'India' 'Nigeria' 'New Zealand'
 'United States' 'South Korea' 'Germany' 'Australia' 'South Africa'
 'Colombia' 'United Kingdom']
Continent
['South America' 'North America' 'Asia' 'Europe' 'Africa' 'Australia']
Hemisphere
['Southern Hemisphere' 'Northern Hemisphere']
```

```
# replace 'Female' with female in sex column

dict1 = {'Female':'female','f':'female','m':'male','Male':'male'}

df['Sex'] = df['Sex'].replace(dict1)

# Calculate the mode of the column
mode_value = df['Sex'].mode()[0]

# Replace NaN values with the mode
df['Sex'].fillna(mode_value, inplace=True)
```

```
<ipython-input-290-9f5343b09dab>:11: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['Sex'].fillna(mode_value, inplace=True)
```

```
dict2 = {'Healty':'Healthy'}

df['Diet'] = df['Diet'].replace(dict2)

# Calculate the mode of the column
mode_value = df['Diet'].mode()[0]

# Replace NaN values with the mode
df['Diet'].fillna(mode_value, inplace=True)
```

```
<ipython-input-291-fa7a37cc995d>:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['Diet'].fillna(mode_value, inplace=True)
```

```
# Splitting the column
split_cols = df['Blood Pressure'].str.split('/', expand=True)
split_cols.columns = ['Systolic', 'Diastolic']

# Convert to numeric (optional, for calculations)
df['Systolic'] = pd.to_numeric(split_cols['Systolic'])
df['Diastolic'] = pd.to_numeric(split_cols['Diastolic'])
```

```
df.head()
```

	Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity	...	BMI	Triglycerides	Physical Activity Days Per Week
0	BMW7812	67.0	male	208	158/88	72	0.0	0.0	1	0	...	31.251233	286	0
1	CZE1114	21.0	male	389	165/93	98	1.0	1.0	1	1	...	27.194973	235	1
3	JLN3497	84.0	male	383	163/100	73	1.0	1.0	1	0	...	36.464704	378	3
6	WYV0966	90.0	male	358	102/73	84	0.0	0.0	1	0	...	28.885811	284	4
7	XXM0972	84.0	male	220	131/68	107	0.0	0.0	1	1	...	22.221862	370	6

5 rows × 28 columns

```
# One-hot encode the gender column data
df = pd.get_dummies(df, columns=['Sex','Diet','Continent','Hemisphere'])
```

Start coding or generate with AI.

```
df.head()
```

Show hidden output

```
df.columns
```

```
Index(['Patient ID', 'Age', 'Cholesterol', 'Blood Pressure', 'Heart Rate',
       'Diabetes', 'Family History', 'Smoking', 'Obesity',
       'Alcohol Consumption', 'Exercise Hours Per Week',
       'Previous Heart Problems', 'Medication Use', 'Stress Level',
       'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides',
       'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
       'Heart Attack Risk', 'Systolic', 'Diastolic', 'Sex_female', 'Sex_male',
       'Diet_Average', 'Diet_Healthy', 'Diet_Unhealthy', 'Continent_Africa',
       'Continent_Asia', 'Continent_Australia', 'Continent_Europe',
       'Continent_North America', 'Continent_South America',
       'Hemisphere_Northern Hemisphere', 'Hemisphere_Southern Hemisphere'],
      dtype='object')
```

Start coding or generate with AI.

```
# Separate the features and target variable
X = df.drop(['Country','Patient ID','Blood Pressure','Heart Attack Risk'], axis=1)
cols = ["Diabetes", "Cholesterol","Exercise Hours Per Week"]
X = df[cols]
y = df['Heart Attack Risk']
```

```
df['Heart Attack Risk'].value_counts()
```

	count
Heart Attack Risk	
0	5044
1	2806

dtype: int64

```
# split the data into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from imblearn.over_sampling import SMOTE

# Convert y_train_top5 and y_train_rfe to pandas Series for easier manipulation
y_train = pd.Series(y_train)

# Display class distribution before oversampling
print('Before Oversampling for X_train:')
print(y_train.value_counts())

# Apply SMOTE for oversampling on the full training set
smote = SMOTE(random_state=42)

# Oversample X_train, y_train
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
# Display class distribution after oversampling
print('After Oversampling for X_train:')
print(y_train_smote.value_counts())
```

```
Before Oversampling for X_train:
Heart Attack Risk
0    4047
1    2233
Name: count, dtype: int64
After Oversampling for X_train:
Heart Attack Risk
1    4047
0    4047
Name: count, dtype: int64
```

```
# Logistic Regression model to predict the outcome
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train_smote, y_train_smote)
```

```
▾ LogisticRegression ⓘ ?
LogisticRegression()
```

```
# Predict the outcome using the trained model
import numpy as np
sample1 = np.array(X_test.iloc[0,:]).reshape(1, -1)
y_pred1 = logreg.predict(sample1)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Logi
  warnings.warn(
```

```
y_pred1
```

```
array([0])
```

```
y_pred = logreg.predict(X_test)
```

```
np.sum(y_pred)
```

```
790
```

```
pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

	Actual	Predicted
2313	0	0
6843	1	1
6404	1	1
1529	0	1
681	1	1
...	...	...
3367	0	1
5838	1	0
1827	1	1
1354	1	0
4020	1	1

1570 rows × 2 columns

```
# Evaluate the model
from sklearn.metrics import accuracy_score,precision_score, recall_score, f1_score
# print these scores
print('Accuracy: ', accuracy_score(y_test, y_pred))
print('Precision: ', precision_score(y_test, y_pred))
print('Recall: ', recall_score(y_test, y_pred))
print('F1: ', f1_score(y_test, y_pred))
```

```
Accuracy:  0.4961783439490446
Precision:  0.3620253164556962
Recall:  0.49912739965095987
F1:  0.41966250917094644
```

```
# Decision Tree model to predict the outcome
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train_smote, y_train_smote)
# Predict the outcome using the trained model
y_pred = dt.predict(X_test)
```

```
# Evaluate the model
# Get the precision score, recall score and f1 score

print('Accuracy: ', accuracy_score(y_test, y_pred))
print('Precision: ', precision_score(y_test, y_pred))
print('Recall: ', recall_score(y_test, y_pred))
print('F1: ', f1_score(y_test, y_pred))
```

```
Accuracy:  0.5484076433121019
Precision:  0.3924050632911392
Recall:  0.4328097731239092
F1:  0.41161825726141077
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

# Prepare models
models = [
    ('Logistic Regression', LogisticRegression()),
    ('Decision Tree', DecisionTreeClassifier()),
    ('Naive Bayes', GaussianNB()),
    ('Support Vector Machine', SVC()),
    ('Random Forest', RandomForestClassifier())
]

# Prepare lists to store results and names
results = []
names = []

# Evaluate each model in turn
for name, model in models:
    # Train the model using training data (SMOTE for handling imbalance)
    #model.fit(X_train, y_train)
    model.fit(X_train_smote, y_train_smote)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Accuracy score
    accuracy = model.score(X_test, y_test)
    results.append(accuracy)
    names.append(name)
    print(f"{name}: Accuracy: {accuracy:.3f}")

    # Classification report
    print(f"Classification Report for {name}:\n", classification_report(y_test, y_pred))
    print()

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt="d",
                xticklabels=['Healthy', 'Heart Attack'],
                yticklabels=['Healthy', 'Heart Attack'])
    plt.title(f'Confusion Matrix for {name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

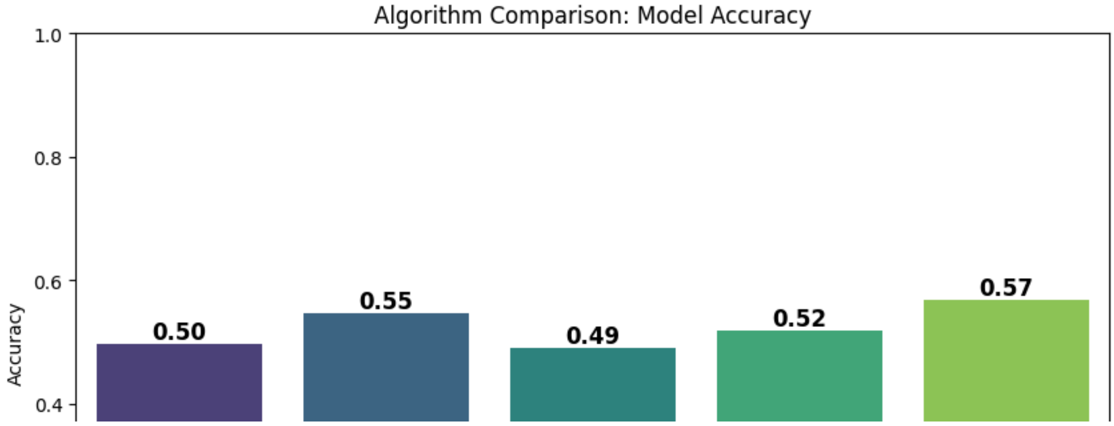
Show hidden output

```
# Bar plot for algorithm comparison (accuracies of models)
plt.figure(figsize=(10, 6))
ax = sns.barplot(x=names, y=results, palette='viridis')
plt.title('Algorithm Comparison: Model Accuracy')
for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=12, fontweight='bold', color='black')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.ylim(0, 1)  # Limiting y-axis from 0 to 1 for accuracy percentage
plt.xticks(rotation=45)  # Rotate model names for better readability
plt.show()
```

```
<ipython-input-325-37d461461e3c>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  ax = sns.barplot(x=names, y=results, palette='viridis')
```


Algorithm Comparison: Model Accuracy
Accuracy