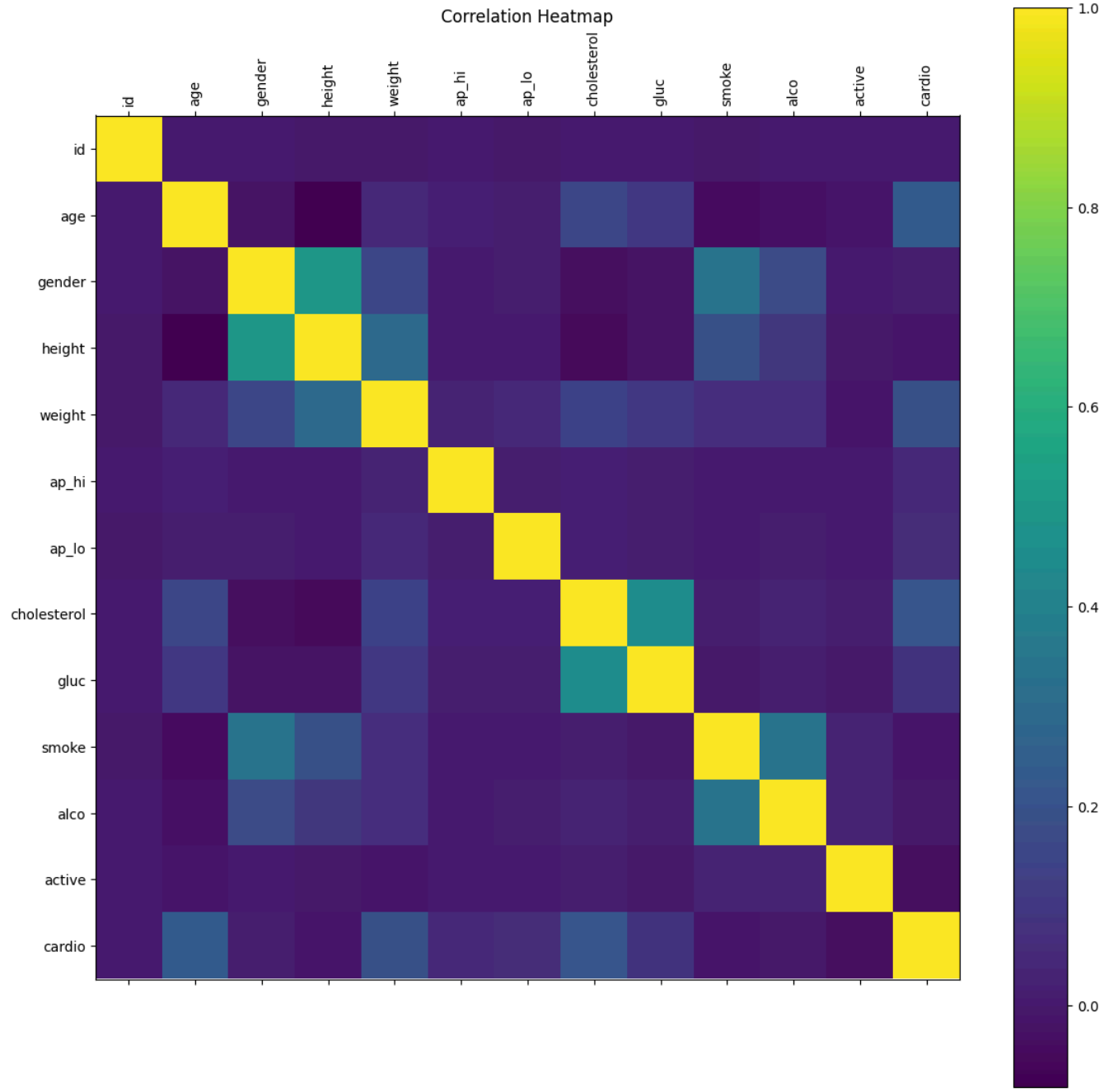
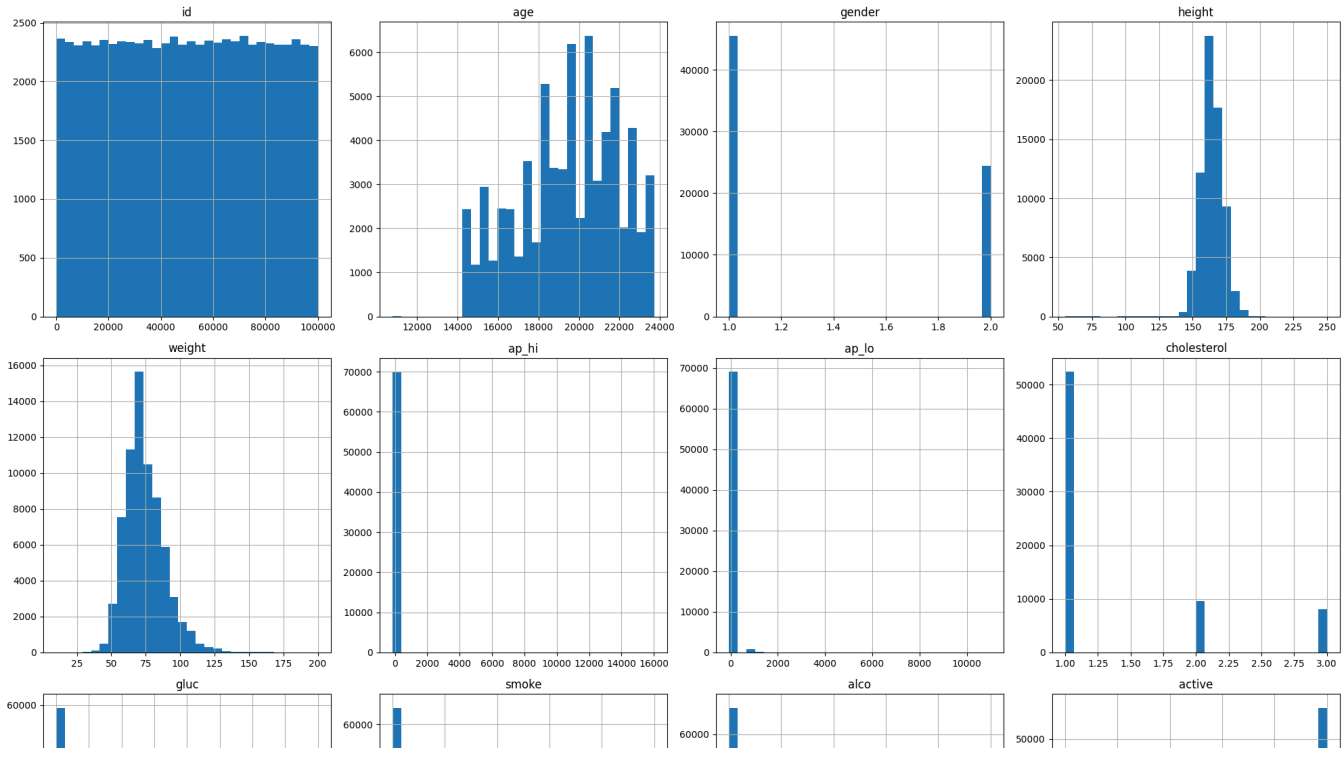
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')

# Load dataset
dataset = pd.read_csv('/content/cardio_train.csv', sep=';')  # semicolon separator is common in cardio dataset

# Basic info
print(dataset.info())
print(dataset.describe())
```

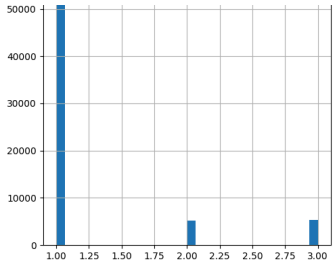
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           70000 non-null  int64
 1   age          70000 non-null  int64
 2   gender       70000 non-null  int64
 3   height       70000 non-null  int64
 4   weight       70000 non-null  float64
 5   ap_hi        70000 non-null  int64
 6   ap_lo        70000 non-null  int64
 7   cholesterol  70000 non-null  int64
 8   gluc         70000 non-null  int64
 9   smoke        70000 non-null  int64
 10  alco         70000 non-null  int64
 11  active       70000 non-null  int64
 12  cardio       70000 non-null  int64
dtypes: float64(1), int64(12)
memory usage: 6.9 MB
None
                 id           age        gender        height        weight  \
count  70000.000000  70000.000000  70000.000000  70000.000000  70000.000000
mean   49972.419900  19468.865814      1.349571    164.359229     74.205690
std    28851.302323   2467.251667      0.476838      8.210126     14.395757
min        0.000000  10798.000000      1.000000     55.000000     10.000000
25%    25006.750000  17664.000000      1.000000    159.000000     65.000000
50%    50001.500000  19703.000000      1.000000    165.000000     72.000000
75%    74889.250000  21327.000000      2.000000    170.000000     82.000000
max    99999.000000  23713.000000      2.000000    250.000000    200.000000

               ap_hi         ap_lo   cholesterol          gluc         smoke  \
count  70000.000000  70000.000000  70000.000000  70000.000000  70000.000000
mean     128.817286     96.630414      1.366871      1.226457      0.088129
std      154.011419    188.472530      0.680250      0.572270      0.283484
min     -150.000000    -70.000000      1.000000      1.000000      0.000000
25%      120.000000     80.000000      1.000000      1.000000      0.000000
50%      120.000000     80.000000      1.000000      1.000000      0.000000
75%      140.000000     90.000000      2.000000      1.000000      0.000000
max    16020.000000  11000.000000      3.000000      3.000000      1.000000

                alco        active        cardio
count  70000.000000  70000.000000  70000.000000
mean       0.053771      0.803729      0.499700
std        0.225568      0.397179      0.500003
min        0.000000      0.000000      0.000000
25%        0.000000      1.000000      0.000000
50%        0.000000      1.000000      0.000000
75%        0.000000      1.000000      1.000000
max        1.000000      1.000000      1.000000
```

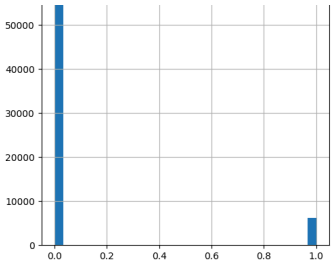
```
# Correlation heatmap
rcParams['figure.figsize'] = 20, 14
plt.matshow(dataset.corr())
plt.xticks(np.arange(dataset.shape[1]), dataset.columns, rotation=90)
plt.yticks(np.arange(dataset.shape[1]), dataset.columns)
plt.colorbar()
plt.title('Correlation Heatmap')
plt.show()
```

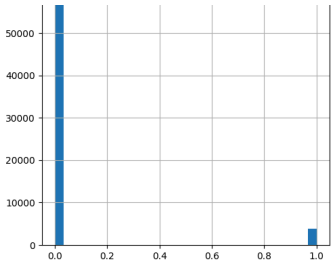
```
# Histogram of all features
dataset.hist(figsize=(20, 20), bins=30)
plt.tight_layout()
plt.show()

# Target distribution
rcParams['figure.figsize'] = 8, 6
plt.bar(dataset['cardio'].unique(), dataset['cardio'].value_counts(), color=['red', 'green'])
plt.xticks([0, 1])
plt.xlabel('Target Classes')
plt.ylabel('Count')
plt.title('Count of each Target Class')
plt.show()
```

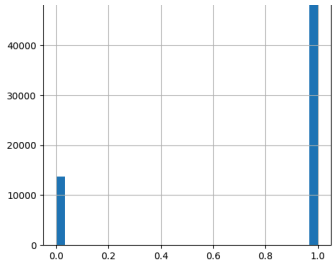

Correlation Heatmap
id
age
gender
height
weight
ap_hi
ap_lo
cholesterol
gluc
smoke
alco
active
cardio
1.0
0.8
0.6
0.4
0.2
0.0

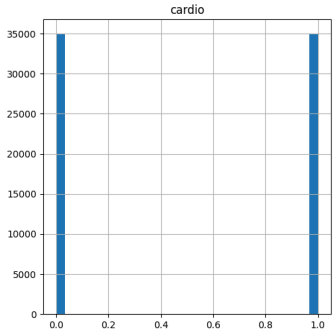

id
age
gender
height
weight
ap_hi
ap_lo
cholesterol
gluc
smoke
alco
active

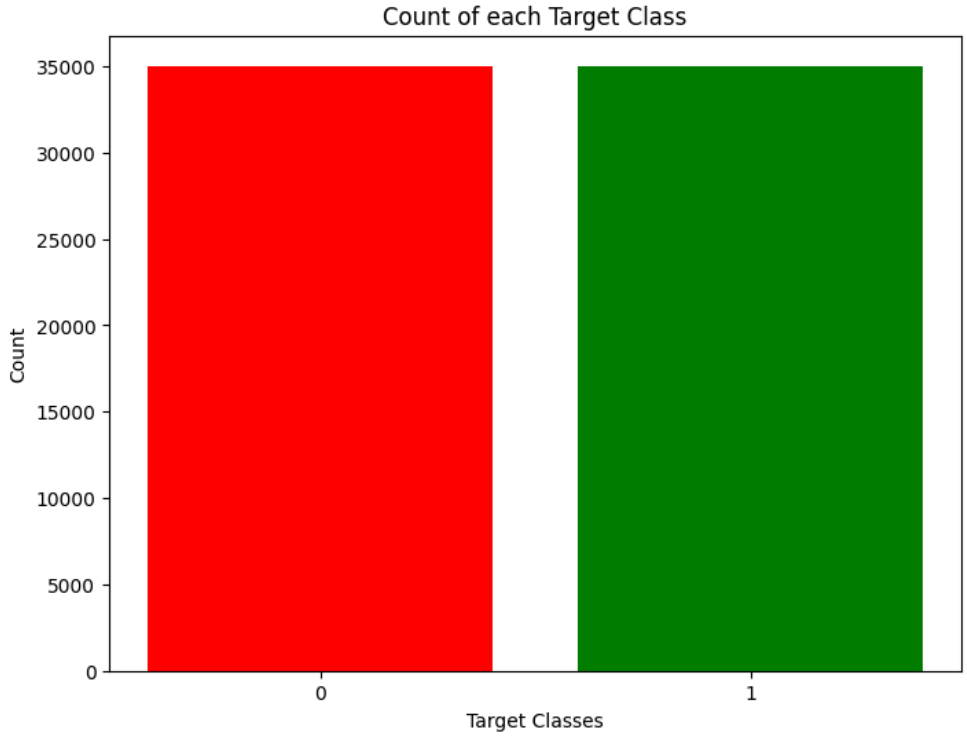




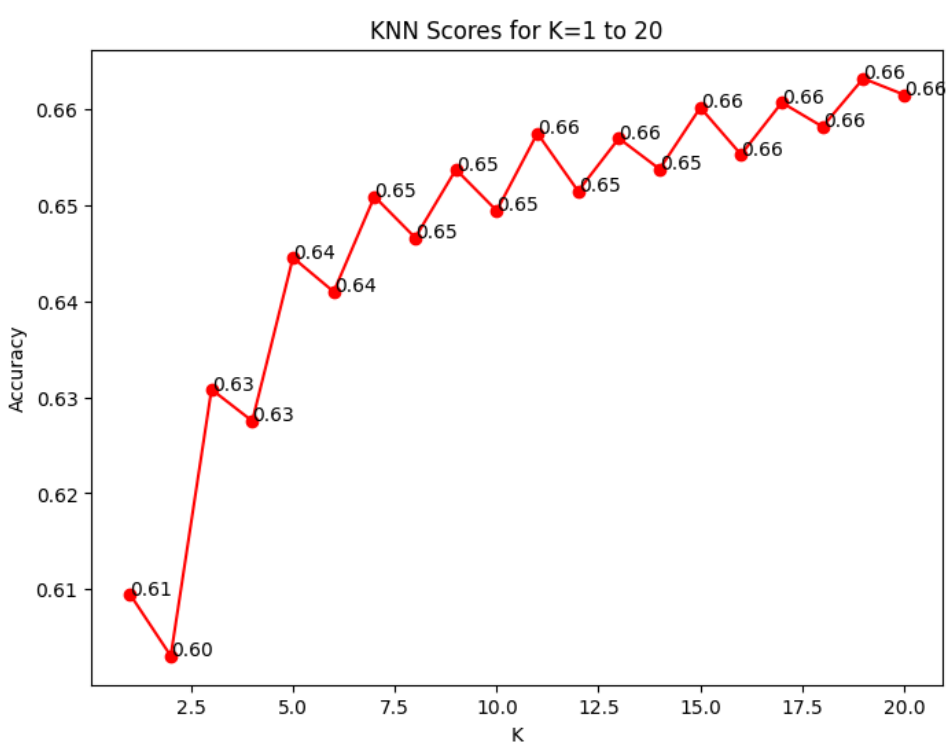





cardio


Count of each Target Class
Count
Target Classes
0
1

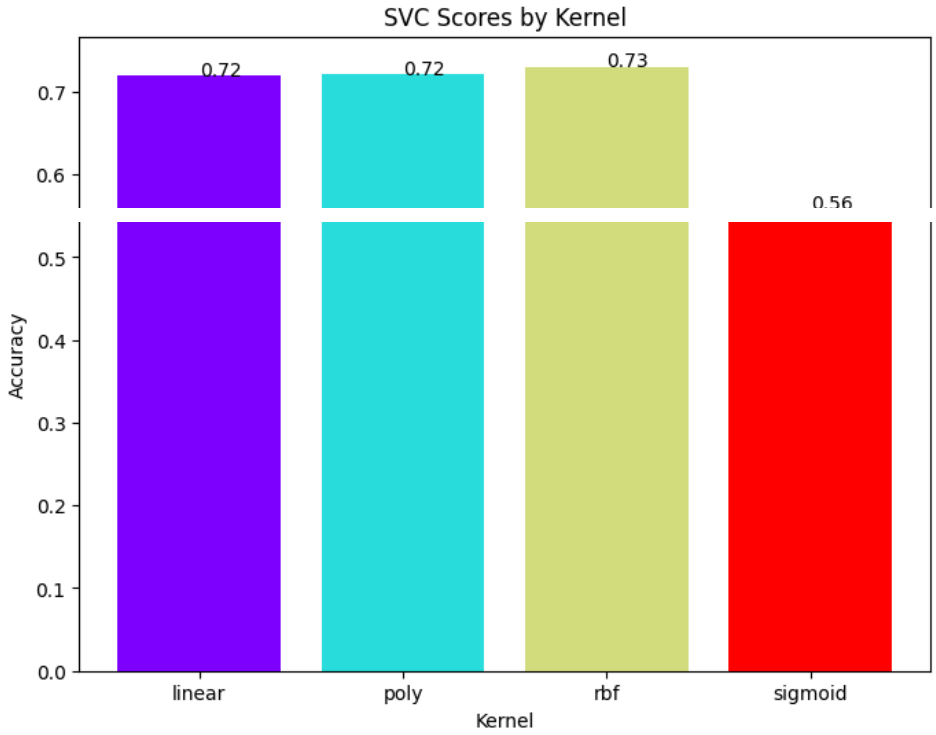
```
# Create a copy for processing
df = dataset.copy()

# Drop irrelevant columns (e.g., ID if present)
if 'id' in df.columns:
    df.drop('id', axis=1, inplace=True)

# Define target and features
y = df['cardio']
X = df.drop('cardio', axis=1)

# Standardize numeric features
columns_to_scale = ['age', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc']
scaler = StandardScaler()
X[columns_to_scale] = scaler.fit_transform(X[columns_to_scale])

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)



knn_scores = []
for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    knn_scores.append(knn.score(X_test, y_test))

plt.figure()
plt.plot(range(1, 21), knn_scores, color='red', marker='o')
for i in range(1, 21):
    plt.text(i, knn_scores[i-1], f"{knn_scores[i-1]:.2f}")
plt.title("KNN Scores for K=1 to 20")
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.show()

print("Best KNN score: {:.2f}% at K={}".format(max(knn_scores)*100, knn_scores.index(max(knn_scores))+1))
```




```
Best KNN score: 66.32% at K=19
```

```
svc_scores = []
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for kernel in kernels:
    svc = SVC(kernel=kernel)
    svc.fit(X_train, y_train)
    svc_scores.append(svc.score(X_test, y_test))

plt.figure()
colors = rainbow(np.linspace(0, 1, len(kernels)))
```

```
plt.bar(kernels, svc_scores, color=colors)
for i, score in enumerate(svc_scores):
    plt.text(i, score, f"{score:.2f}")
plt.title("SVC Scores by Kernel")
plt.xlabel("Kernel")
plt.ylabel("Accuracy")
plt.show()

print("Best SVC score: {:.2f}% using '{}' kernel".format(max(svc_scores)*100, kernels[svc_scores.index(max(svc_scores))])
```




```
Best SVC score: 73.03% using 'rbf' kernel
```

```
dt_scores = []
max_features_range = range(1, X.shape[1] + 1)
for i in max_features_range:
    dt = DecisionTreeClassifier(max_features=i, random_state=0)
    dt.fit(X_train, y_train)
    dt_scores.append(dt.score(X_test, y_test))

plt.figure()
plt.plot(max_features_range, dt_scores, color='green', marker='x')
for i in max_features_range:
    plt.text(i, dt_scores[i-1], f"{dt_scores[i-1]:.2f}")
plt.title("Decision Tree Scores for Different max_features")
plt.xlabel("max_features")
plt.ylabel("Accuracy")
plt.show()

best_index = dt_scores.index(max(dt_scores)) + 1
print("Best Decision Tree score: {:.2f}% with max_features={}".format(max(dt_scores)*100, best_index))
```