```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
df = pd.read_csv("train_dataset.csv")
df.head()
```

	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	hearing(right)	systolic	relaxation	...	HDL	LDL	hemoglobin	Urine protein	crea
0	35	170	85	97.0	0.9	0.9	1	1	118	78	...	70	142	19.8	1	
1	20	175	110	110.0	0.7	0.9	1	1	119	79	...	71	114	15.9	1	
2	45	155	65	86.0	0.9	0.9	1	1	110	80	...	57	112	13.7	3	
3	45	165	80	94.0	0.8	0.7	1	1	158	88	...	46	91	16.9	1	
4	20	165	60	81.0	1.5	0.1	1	1	109	64	...	47	92	14.9	1	

5 rows × 23 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38984 entries, 0 to 38983
Data columns (total 23 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  38984 non-null  int64  
 1   height(cm)           38984 non-null  int64  
 2   weight(kg)           38984 non-null  int64  
 3   waist(cm)            38984 non-null  float64
 4   eyesight(left)       38984 non-null  float64
 5   eyesight(right)      38984 non-null  float64
 6   hearing(left)        38984 non-null  int64  
 7   hearing(right)       38984 non-null  int64  
 8   systolic             38984 non-null  int64  
 9   relaxation           38984 non-null  int64  
 10  fasting blood sugar  38984 non-null  int64  
 11  Cholesterol          38984 non-null  int64  
 12  triglyceride         38984 non-null  int64  
 13  HDL                  38984 non-null  int64  
 14  LDL                  38984 non-null  int64  
 15  hemoglobin           38984 non-null  float64
 16  Urine protein        38984 non-null  int64  
 17  serum creatinine     38984 non-null  float64
 18  AST                  38984 non-null  int64  
 19  ALT                  38984 non-null  int64  
 20  Gtp                  38984 non-null  int64  
 21  dental caries        38984 non-null  int64  
 22  smoking              38984 non-null  int64  
dtypes: float64(5), int64(18)
memory usage: 6.8 MB
```

```
for i in df.columns:
    print(i + " : ",format(len(df[i].value_counts())))
```

```
age :  14
height(cm) :  13
weight(kg) :  22
waist(cm) :  545
eyesight(left) :  19
eyesight(right) :  17
hearing(left) :  2
hearing(right) :  2
systolic :  125
relaxation :  94
fasting blood sugar :  258
Cholesterol :  279
triglyceride :  389
HDL :  122
LDL :  286
hemoglobin :  143
Urine protein :  6
serum creatinine :  34
AST :  195
ALT :  230
Gtp :  439
dental caries :  2
smoking :  2
```

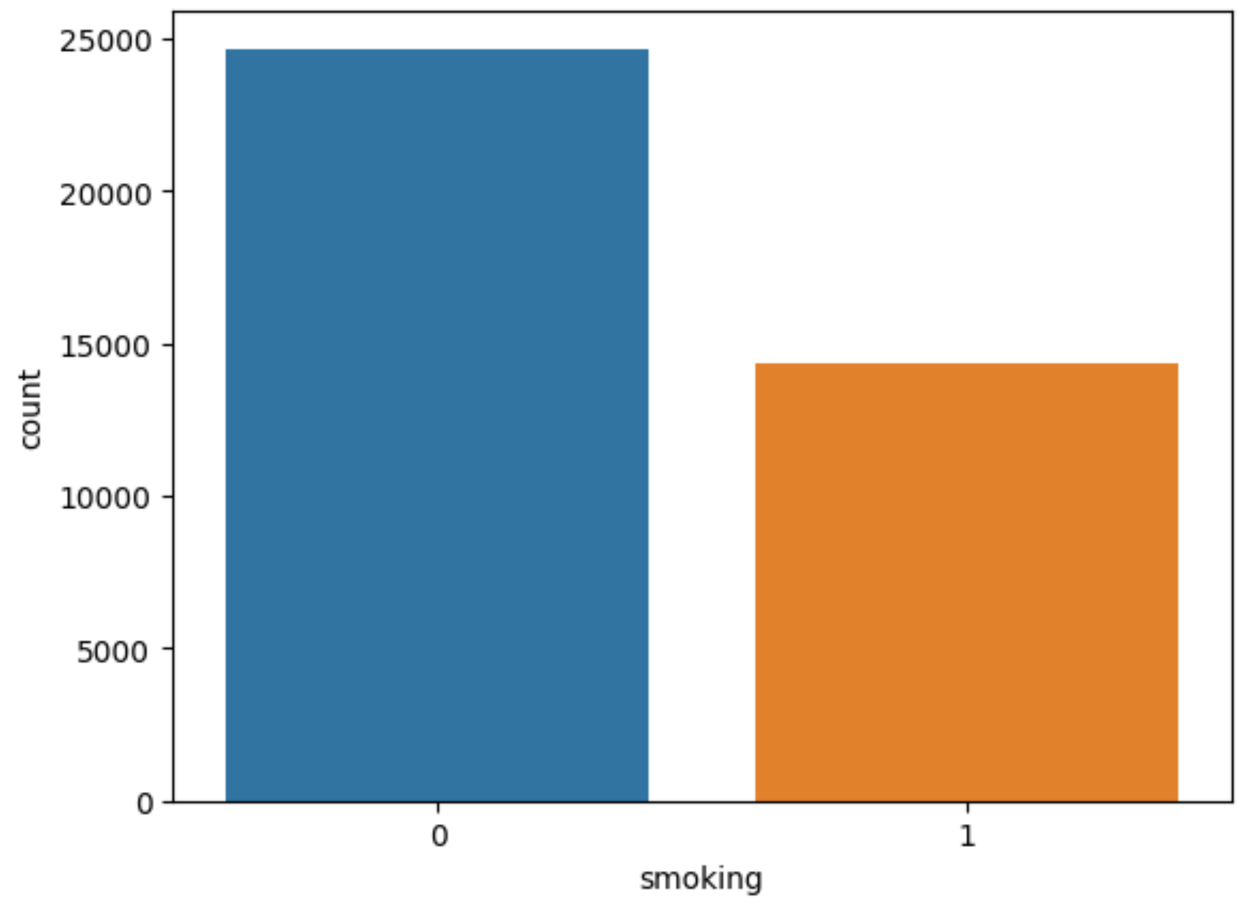
```
# Check for null values in each column

null_columns = df.columns[df.isnull().any()]

# Print the name of columns with null values

if len(null_columns) > 0:
    print("The following columns contain null values:")
    for col in null_columns:
        print(col)
else:
    print("No null values found in any column.")
```

```
No null values found in any column.
```

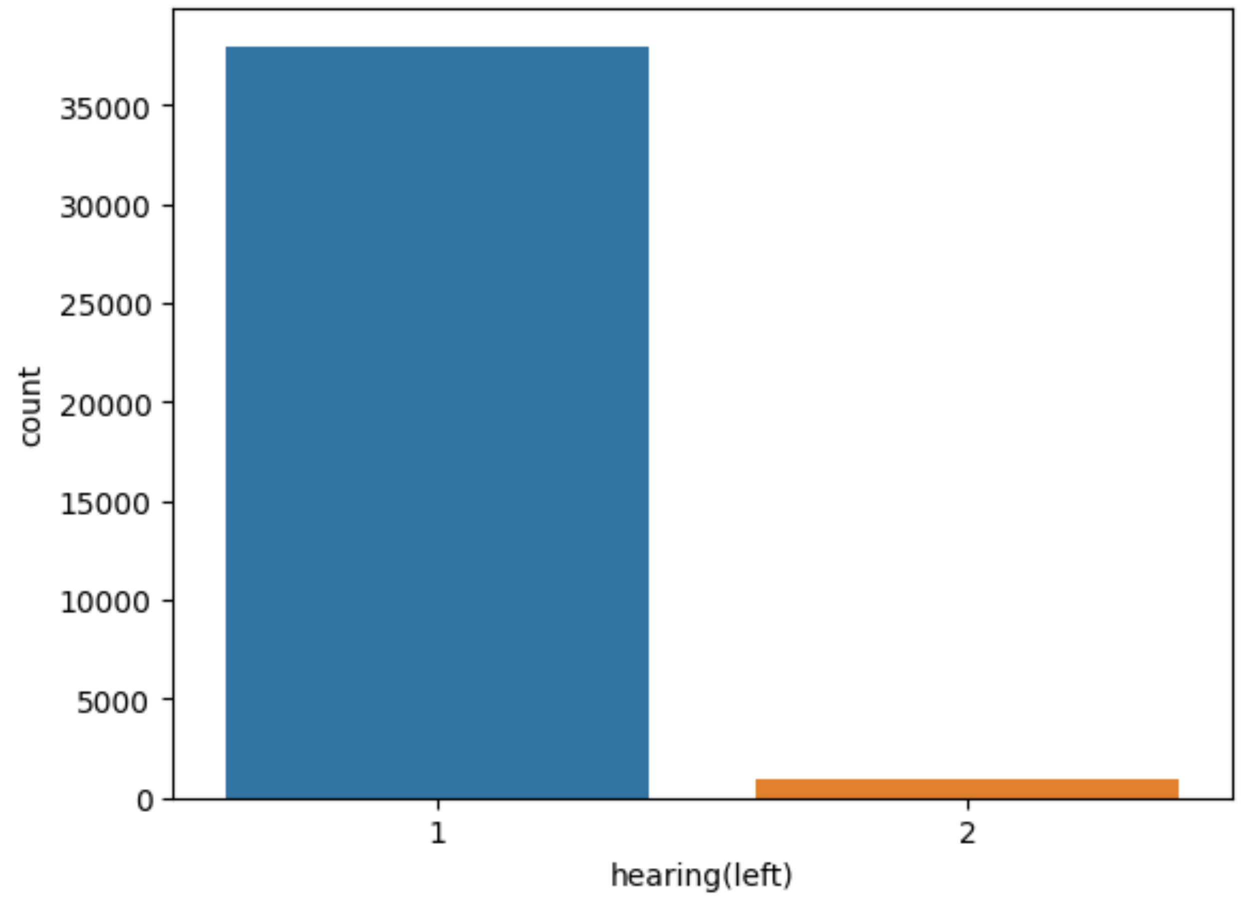
```
# plot the graphs of Categorical Variables

l1 = ['hearing(left)','hearing(right)','Urine protein','dental caries','smoking']
for i in l1:
  print(i+" : \n",format(df[i].value_counts()))
  print('\n')
```

```
hearing(left) :
 1    37995
2      989
Name: hearing(left), dtype: int64


hearing(right) :
 1    37963
2     1021
Name: hearing(right), dtype: int64


Urine protein :
 1    36836
2     1236
3      667
4      182
5       58
6        5
Name: Urine protein, dtype: int64


dental caries :
 0    30625
1     8359
Name: dental caries, dtype: int64


smoking :
 0    24666
1    14318
Name: smoking, dtype: int64
```

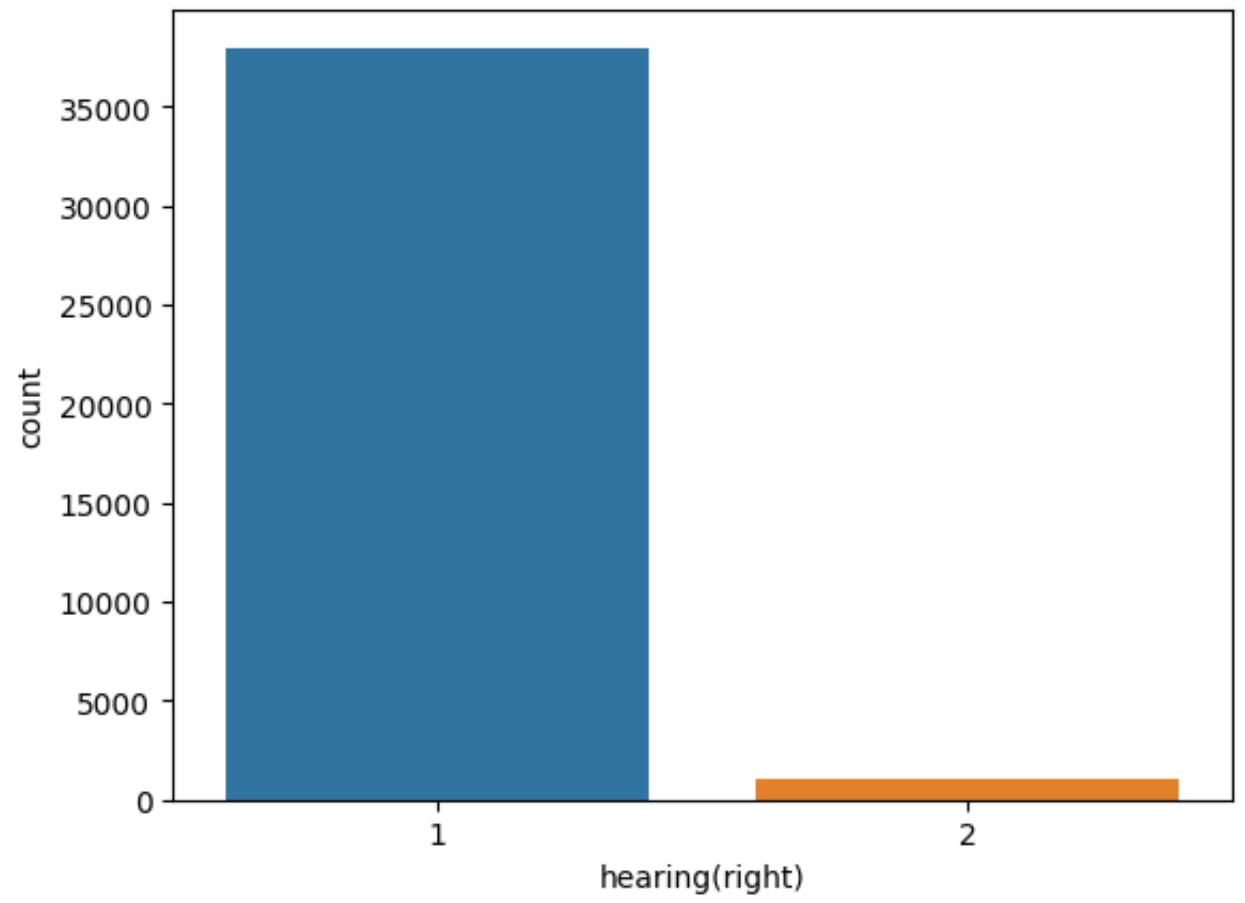
```
sns.countplot(x="smoking", data=df)

plt.show()
```



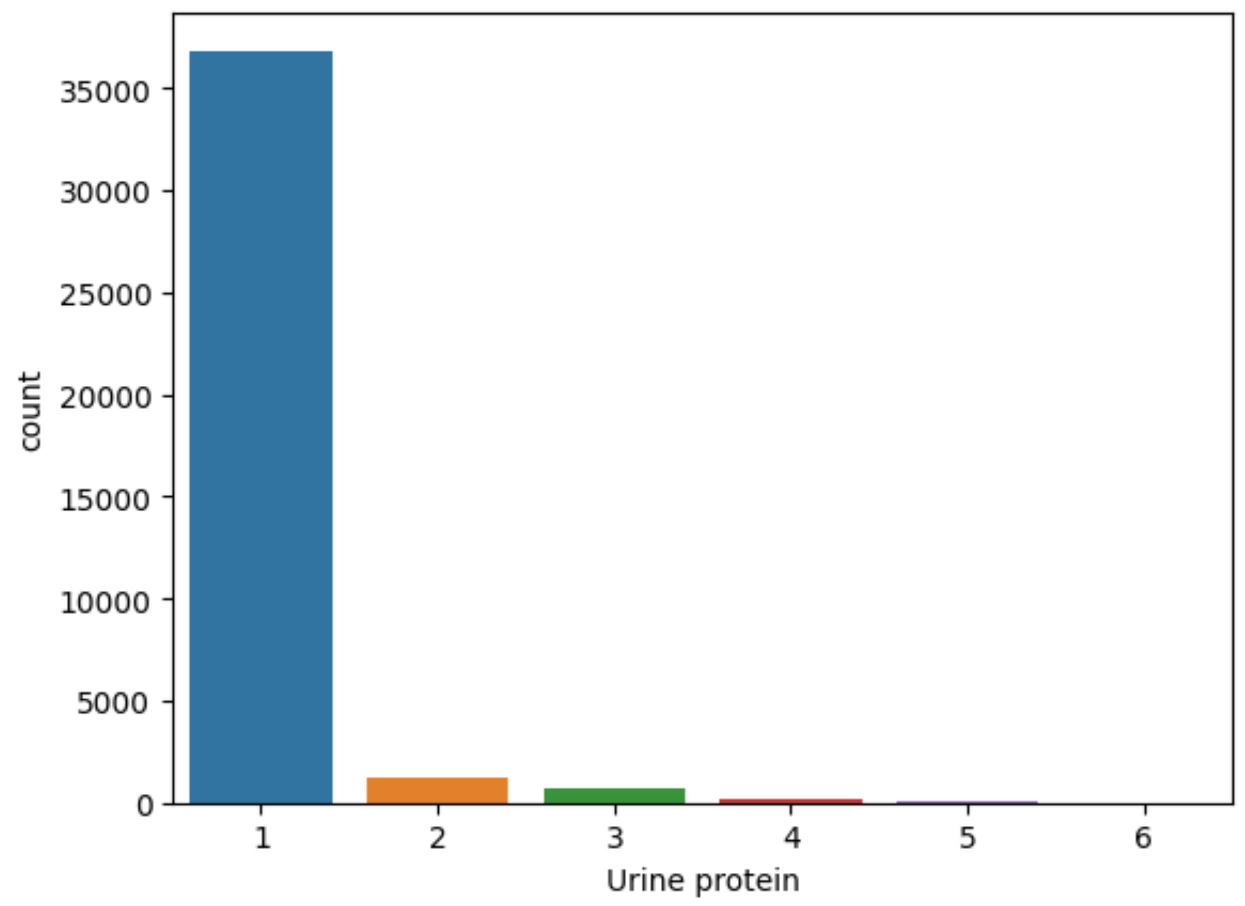
```
sns.countplot(x="hearing(left)", data=df)

plt.show()
```



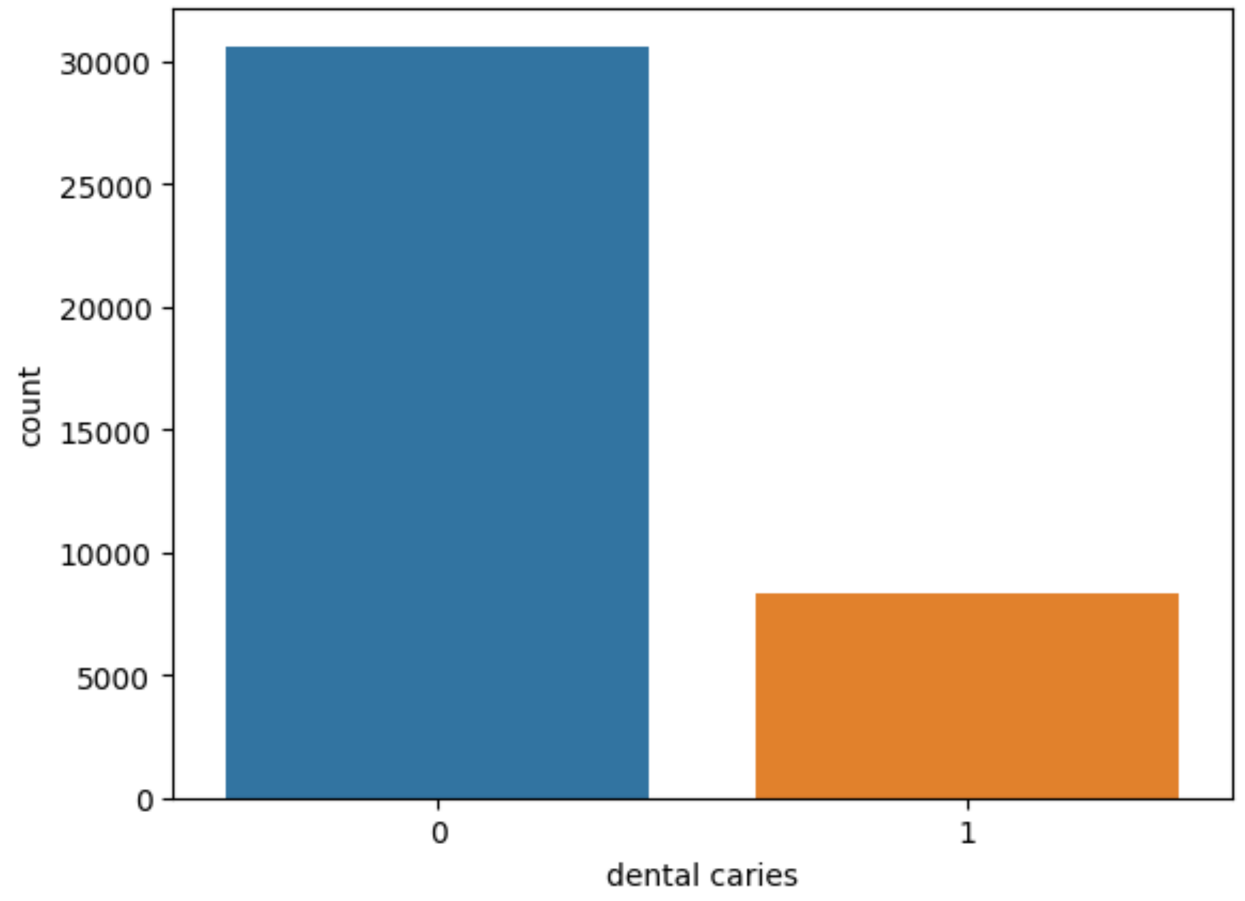
```
sns.countplot(x="hearing(right)", data=df)

plt.show()
```



```
sns.countplot(x="Urine protein", data=df)

plt.show()
```



```
sns.countplot(x="dental caries", data=df)

plt.show()
```



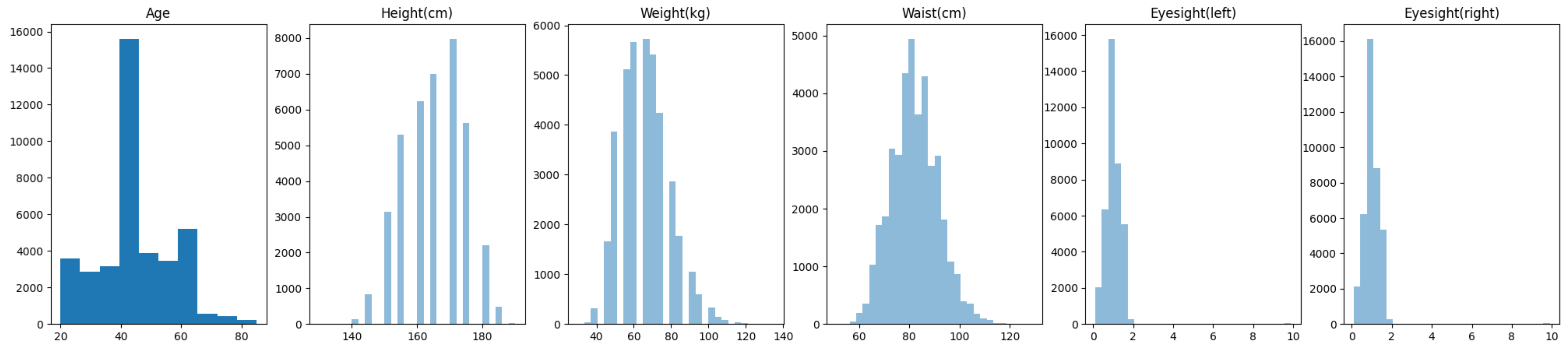
```
from scipy.stats import chi2_contingency

# Create a contingency table
contingency_table = pd.crosstab(df['smoking'], df['dental caries'])
contingency_table
# Perform chi-square test
stat, p, dof, expected = chi2_contingency(contingency_table)

# interpret p-value
alpha = 0.05
print("Chi-Square test result for Smoking and Dental caries")
print("p value is " + str(p))
if p <= alpha:
    print('Corelated (reject H0)')
else:
    print('Independent (H0 holds true)')

# Create a contingency table
contingency_table = pd.crosstab(df['smoking'], df['Urine protein'])
```

```
contingency_table
# Perform chi-square test
stat, p, dof, expected = chi2_contingency(contingency_table)

# interpret p-value
alpha = 0.05
print("Chi-Square test result for Smoking and Urine protein")
print("p value is " + str(p))
if p <= alpha:
    print('Corelated (reject H0)')
else:
    print('Independent (H0 holds true)')

# Create a contingency table
contingency_table = pd.crosstab(df['smoking'], df['hearing(right)'])
contingency_table
# Perform chi-square test
stat, p, dof, expected = chi2_contingency(contingency_table)

# interpret p-value
alpha = 0.05
print("Chi-Square test result for Smoking and hearing(right)")
print("p value is " + str(p))
if p <= alpha:
    print('Corelated (reject H0)')
else:
    print('Independent (H0 holds true)')

# Create a contingency table
contingency_table = pd.crosstab(df['smoking'], df['hearing(left)'])
contingency_table
# Perform chi-square test
stat, p, dof, expected = chi2_contingency(contingency_table)

# interpret p-value
alpha = 0.05
print("Chi-Square test result for Smoking and hearing(left)")
print("p value is " + str(p))
if p <= alpha:
    print('Corelated (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
Chi-Square test result for Smoking and Dental caries
p value is 4.809679216057804e-100
Corelated (reject H0)
Chi-Square test result for Smoking and Urine protein
p value is 0.049511278397280714
Corelated (reject H0)
Chi-Square test result for Smoking and hearing(right)
p value is 0.0002020239535809226
Corelated (reject H0)
Chi-Square test result for Smoking and hearing(left)
p value is 1.5214418715754427e-05
Corelated (reject H0)
```

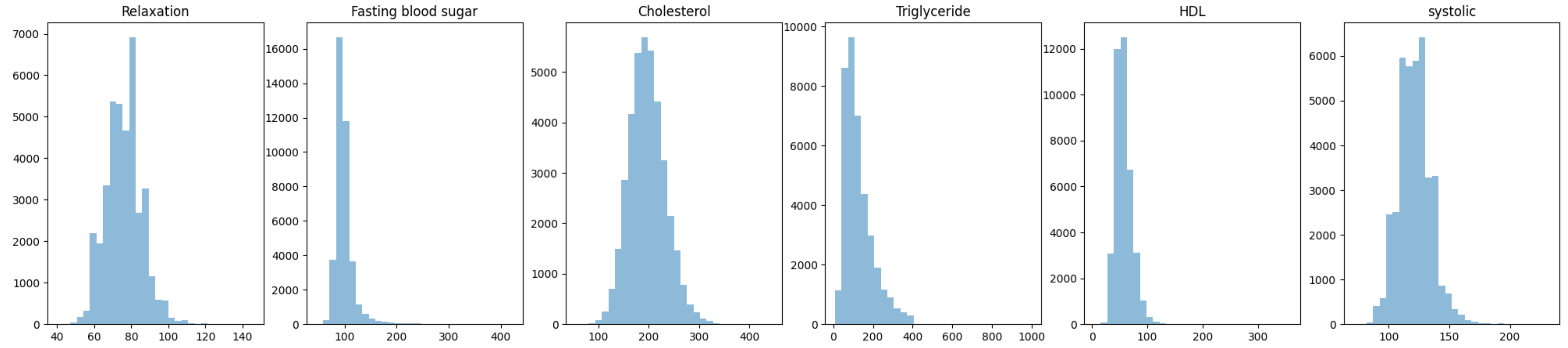
```
# Plot the graphs for Numerical Variables

fig,axs = plt.subplots(1,6,figsize=(25,5))
axs[0].hist(df['age'])
axs[0].set_title('Age')
axs[1].hist(df['height(cm)'], bins=30, alpha=0.5)
axs[1].set_title('Height(cm)')
axs[2].hist(df['weight(kg)'], bins=30, alpha=0.5)
axs[2].set_title('Weight(kg)')
axs[3].hist(df['waist(cm)'], bins=30, alpha=0.5)
axs[3].set_title('Waist(cm)')
axs[4].hist(df['eyesight(left)'], bins=30, alpha=0.5)
axs[4].set_title('Eyesight(left)')
axs[5].hist(df['eyesight(right)'], bins=30, alpha=0.5)
axs[5].set_title('Eyesight(right)')
```

```
Text(0.5, 1.0, 'Eyesight(right)')
```



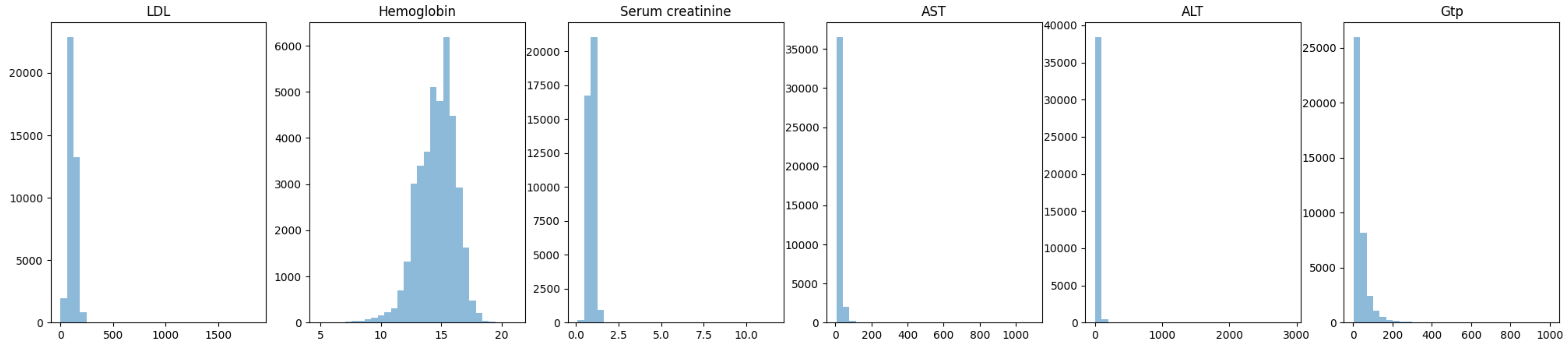
```
fig,axs = plt.subplots(1,6,figsize=(25,5))
axs[0].hist(df['relaxation'], bins=30, alpha=0.5)
axs[0].set_title('Relaxation')
axs[1].hist(df['fasting blood sugar'], bins=30, alpha=0.5)
axs[1].set_title('Fasting blood sugar')
axs[2].hist(df['Cholesterol'], bins=30, alpha=0.5)
axs[2].set_title('Cholesterol')
axs[3].hist(df['triglyceride'], bins=30, alpha=0.5)
axs[3].set_title('Triglyceride')
axs[4].hist(df['HDL'], bins=30, alpha=0.5)
axs[4].set_title('HDL')
axs[5].hist(df['systolic'], bins=30, alpha=0.5)
axs[5].set_title('systolic')
```

```
Text(0.5, 1.0, 'systolic')
```



```
fig,axs = plt.subplots(1,6,figsize=(25,5))
axs[0].hist(df['LDL'], bins=30, alpha=0.5)
axs[0].set_title('LDL')
axs[1].hist(df['hemoglobin'], bins=30, alpha=0.5)
axs[1].set_title('Hemoglobin')
axs[2].hist(df['serum creatinine'], bins=30, alpha=0.5)
axs[2].set_title('Serum creatinine')
axs[3].hist(df['AST'], bins=30, alpha=0.5)
axs[3].set_title('AST')
axs[4].hist(df['ALT'], bins=30, alpha=0.5)
axs[4].set_title('ALT')
axs[5].hist(df['Gtp'], bins=30, alpha=0.5)
axs[5].set_title('Gtp')
```

```
Text(0.5, 1.0, 'Gtp')
```



```
# Remove the outliers

df_removed_outlier = df[
(df['eyesight(left)'] < 2) &
(df['eyesight(right)'] < 2) &
(df['fasting blood sugar'] < 250) &
(df['triglyceride'] < 400) &
(df['HDL'] < 125) &
(df['LDL'] < 250) &
(df['serum creatinine'] < 2) &
(df['AST'] < 100) &
(df['ALT'] < 200) &
(df['Gtp'] < 300)]

print("Original dataset length: ", len(df))
print("New dataset: ", len(df_removed_outlier))
print("Difference: ", len(df) - len(df_removed_outlier))
```

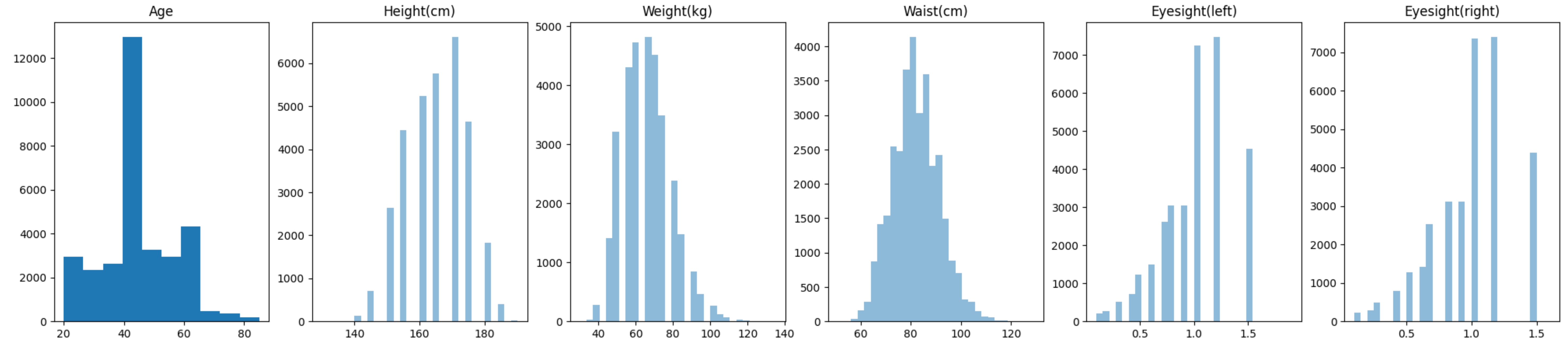
```
    Original dataset length:  38984
    New dataset:  37813
    Difference:  1171
```

```
# Remove the duplicate rows

print("Number of duplicate rows: ", df_removed_outlier[df_removed_outlier.duplicated() == True].shape[0])
print(f"\nRows in original Dataframe: {df_removed_outlier.shape[0]}")

new_df = df_removed_outlier.drop_duplicates()
print(f"Dataframe rows after removing duplicates: {new_df.shape[0]}")
```

```
    Number of duplicate rows:  5362

    Rows in original Dataframe: 37813
    Dataframe rows after removing duplicates: 32451
```

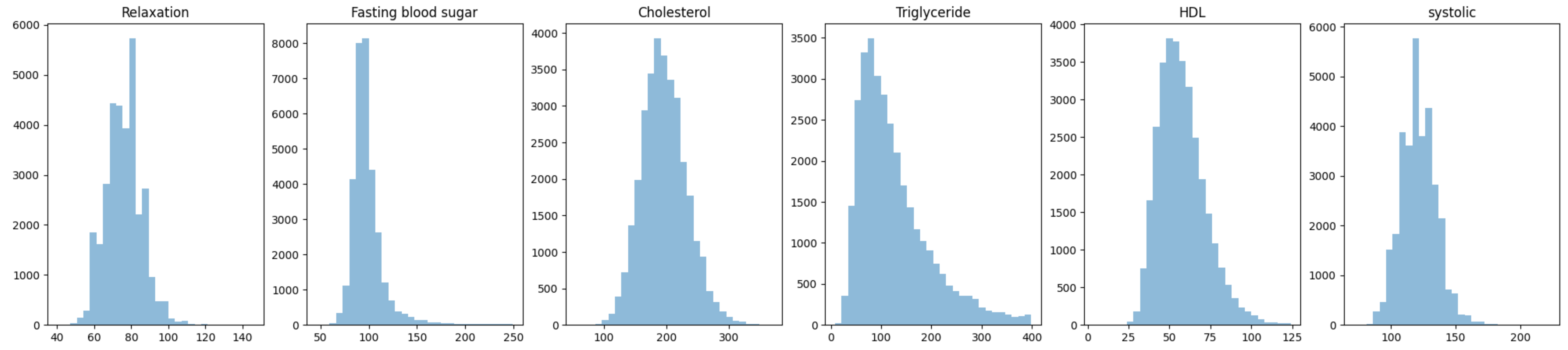
```
# plot the graph after removing outliers and duplicate rows

fig,axs = plt.subplots(1,6,figsize=(25,5))
axs[0].hist(new_df['age'])
axs[0].set_title('Age')
axs[1].hist(new_df['height(cm)'], bins=30, alpha=0.5)
axs[1].set_title('Height(cm)')
axs[2].hist(new_df['weight(kg)'], bins=30, alpha=0.5)
axs[2].set_title('Weight(kg)')
axs[3].hist(new_df['waist(cm)'], bins=30, alpha=0.5)
axs[3].set_title('Waist(cm)')
axs[4].hist(new_df['eyesight(left)'], bins=30, alpha=0.5)
axs[4].set_title('Eyesight(left)')
axs[5].hist(new_df['eyesight(right)'], bins=30, alpha=0.5)
axs[5].set_title('Eyesight(right)')
```

```
Text(0.5, 1.0, 'Eyesight(right)')
```



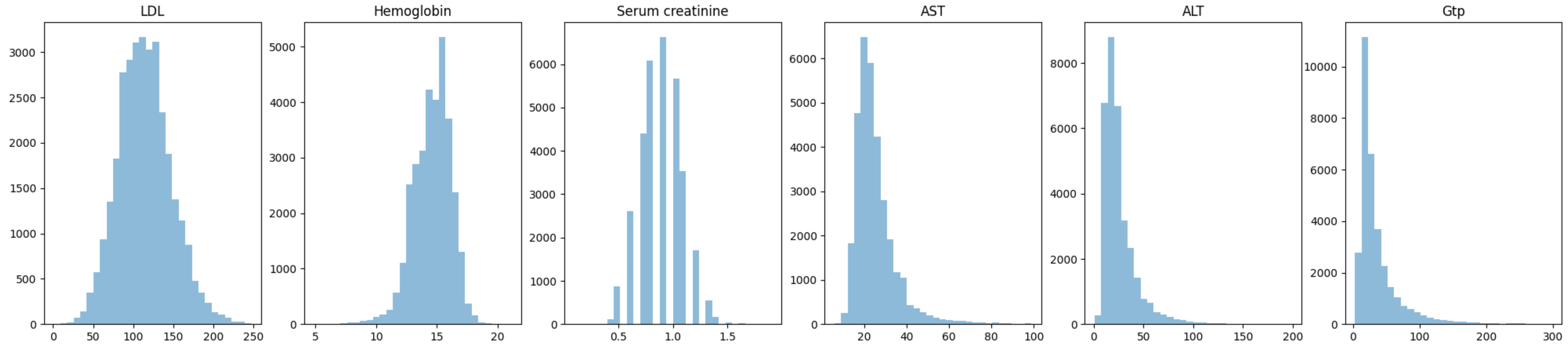
```
fig,axs = plt.subplots(1,6,figsize=(25,5))
axs[0].hist(new_df['relaxation'], bins=30, alpha=0.5)
axs[0].set_title('Relaxation')
axs[1].hist(new_df['fasting blood sugar'], bins=30, alpha=0.5)
axs[1].set_title('Fasting blood sugar')
axs[2].hist(new_df['Cholesterol'], bins=30, alpha=0.5)
axs[2].set_title('Cholesterol')
axs[3].hist(new_df['triglyceride'], bins=30, alpha=0.5)
axs[3].set_title('Triglyceride')
axs[4].hist(new_df['HDL'], bins=30, alpha=0.5)
axs[4].set_title('HDL')
axs[5].hist(new_df['systolic'], bins=30, alpha=0.5)
axs[5].set_title('systolic')
```

```
Text(0.5, 1.0, 'systolic')
```

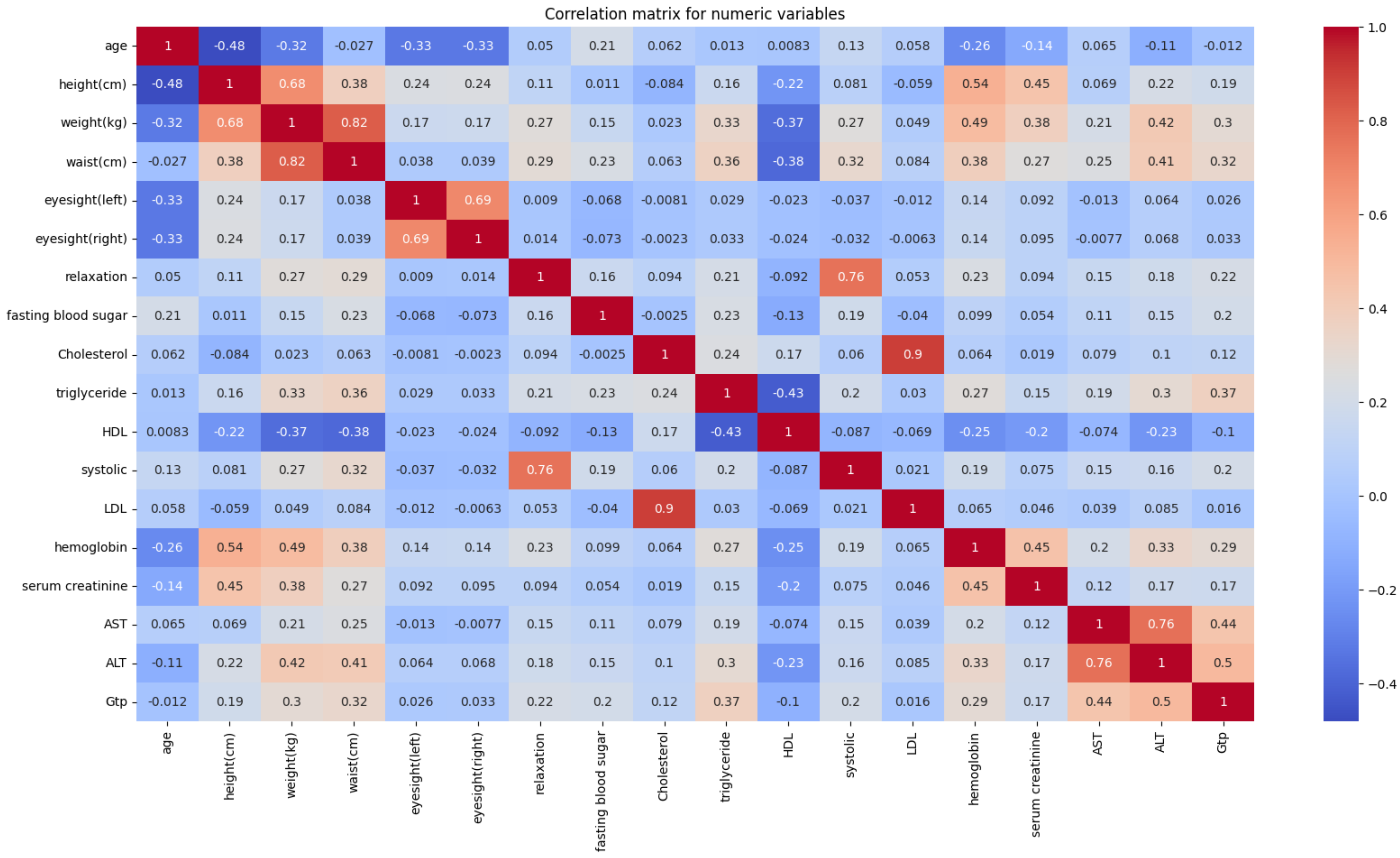


```
fig,axs = plt.subplots(1,6,figsize=(25,5))
axs[0].hist(new_df['LDL'], bins=30, alpha=0.5)
axs[0].set_title('LDL')
axs[1].hist(new_df['hemoglobin'], bins=30, alpha=0.5)
axs[1].set_title('Hemoglobin')
axs[2].hist(new_df['serum creatinine'], bins=30, alpha=0.5)
axs[2].set_title('Serum creatinine')
axs[3].hist(new_df['AST'], bins=30, alpha=0.5)
axs[3].set_title('AST')
axs[4].hist(new_df['ALT'], bins=30, alpha=0.5)
axs[4].set_title('ALT')
axs[5].hist(new_df['Gtp'], bins=30, alpha=0.5)
axs[5].set_title('Gtp')
```

```
Text(0.5, 1.0, 'Gtp')
```



```
# Correlation between Numerical variables

cols = ["age", "height(cm)", "weight(kg)","waist(cm)", "eyesight(left)", "eyesight(right)", "relaxation",
        "fasting blood sugar", "Cholesterol", "triglyceride", "HDL", "systolic", "LDL", "hemoglobin", "serum creatinine", "AST", "ALT", "Gtp"]
subset = new_df[cols]

plt.figure(figsize=(20, 10))
plt.title("Correlation matrix for numeric variables")
sns.heatmap(subset.corr(), annot=True, cmap="coolwarm")
plt.show()
```

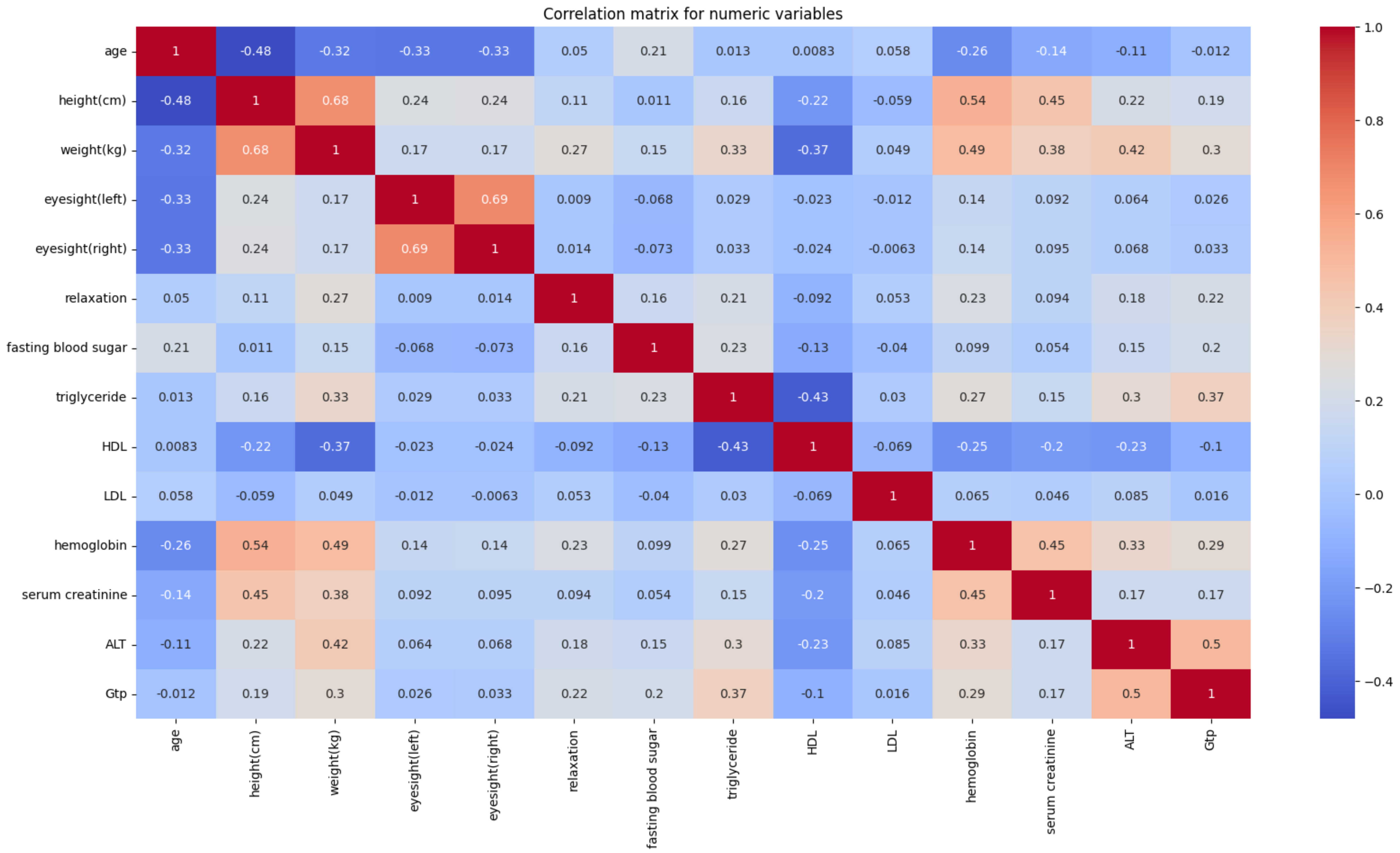


	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	relaxation	fasting blood sugar	Cholesterol	triglyceride	HDL	systolic	LDL	hemoglobin	serum creatinine	AST	ALT	Gtp
age	1	-0.48	-0.32	-0.027	-0.33	-0.33	0.05	0.21	0.062	0.013	0.0083	0.13	0.058	-0.26	-0.14	0.065	-0.11	-0.012
height(cm)	-0.48	1	0.68	0.38	0.24	0.24	0.11	0.011	-0.084	0.16	-0.22	0.081	-0.059	0.54	0.45	0.069	0.22	0.19
weight(kg)	-0.32	0.68	1	0.82	0.17	0.17	0.27	0.15	0.023	0.33	-0.37	0.27	0.049	0.49	0.38	0.21	0.42	0.3
waist(cm)	-0.027	0.38	0.82	1	0.038	0.039	0.29	0.23	0.063	0.36	-0.38	0.32	0.084	0.38	0.27	0.25	0.41	0.32
eyesight(left)	-0.33	0.24	0.17	0.038	1	0.69	0.009	-0.068	-0.0081	0.029	-0.023	-0.037	-0.012	0.14	0.092	-0.013	0.064	0.026
eyesight(right)	-0.33	0.24	0.17	0.039	0.69	1	0.014	-0.073	-0.0023	0.033	-0.024	-0.032	-0.0063	0.14	0.095	-0.0077	0.068	0.033
relaxation	0.05	0.11	0.27	0.29	0.009	0.014	1	0.16	0.094	0.21	-0.092	0.76	0.053	0.23	0.094	0.15	0.18	0.22
fasting blood sugar	0.21	0.011	0.15	0.23	-0.068	-0.073	0.16	1	-0.0025	0.23	-0.13	0.19	-0.04	0.099	0.054	0.11	0.15	0.2
Cholesterol	0.062	-0.084	0.023	0.063	-0.0081	-0.0023	0.094	-0.0025	1	0.24	0.17	0.06	0.9	0.064	0.019	0.079	0.1	0.12
triglyceride	0.013	0.16	0.33	0.36	0.029	0.033	0.21	0.23	0.24	1	-0.43	0.2	0.03	0.27	0.15	0.19	0.3	0.37
HDL	0.0083	-0.22	-0.37	-0.38	-0.023	-0.024	-0.092	-0.13	0.17	-0.43	1	-0.087	-0.069	-0.25	-0.2	-0.074	-0.23	-0.1
systolic	0.13	0.081	0.27	0.32	-0.037	-0.032	0.76	0.19	0.06	0.2	-0.087	1	0.021	0.19	0.075	0.15	0.16	0.2
LDL	0.058	-0.059	0.049	0.084	-0.012	-0.0063	0.053	-0.04	0.9	0.03	-0.069	0.021	1	0.065	0.046	0.039	0.085	0.016
hemoglobin	-0.26	0.54	0.49	0.38	0.14	0.14	0.23	0.099	0.064	0.27	-0.25	0.19	0.065	1	0.45	0.2	0.33	0.29
serum creatinine	-0.14	0.45	0.38	0.27	0.092	0.095	0.094	0.054	0.019	0.15	-0.2	0.075	0.046	0.45	1	0.12	0.17	0.17
AST	0.065	0.069	0.21	0.25	-0.013	-0.0077	0.15	0.11	0.079	0.19	-0.074	0.15	0.039	0.2	0.12	1	0.76	0.44
ALT	-0.11	0.22	0.42	0.41	0.064	0.068	0.18	0.15	0.1	0.3	-0.23	0.16	0.085	0.33	0.17	0.76	1	0.5
Gtp	-0.012	0.19	0.3	0.32	0.026	0.033	0.22	0.2	0.12	0.37	-0.1	0.2	0.016	0.29	0.17	0.44	0.5	1

```
# Dimension Reduction
cols_to_drop = ['waist(cm)','systolic','Cholesterol','AST']

new_df = new_df.drop(columns=cols_to_drop)

new_df.head()
```

	age	height(cm)	weight(kg)	eyesight(left)	eyesight(right)	hearing(left)	hearing(right)	relaxation	fasting blood sugar	triglyceride	HDL	LDL	hemoglobin	Urine protein	s creati
0	35	170	85	0.9	0.9	1	1	78	97	153	70	142	19.8	1	
1	20	175	110	0.7	0.9	1	1	79	88	128	71	114	15.9	1	
3	45	165	80	0.8	0.7	1	1	88	249	366	46	91	16.9	1	
4	20	165	60	1.5	0.1	1	1	64	100	200	47	92	14.9	1	
5	60	160	50	1.0	0.9	2	2	75	114	74	98	64	13.9	1	

```
# Heatmap after dimension reduction

cols = ["age", "height(cm)", "weight(kg)", "eyesight(left)", "eyesight(right)", "relaxation",
        "fasting blood sugar", "triglyceride", "HDL", "LDL", "hemoglobin", "serum creatinine", "ALT", "Gtp"]
subset = new_df[cols]

plt.figure(figsize=(20, 10))
plt.title("Correlation matrix for numeric variables")
sns.heatmap(subset.corr(), annot=True, cmap="coolwarm")
plt.show()
```



	age	height(cm)	weight(kg)	eyesight(left)	eyesight(right)	relaxation	fasting blood sugar	triglyceride	HDL	LDL	hemoglobin	serum creatinine	ALT	Gtp
age	1	-0.48	-0.32	-0.33	-0.33	0.05	0.21	0.013	0.0083	0.058	-0.26	-0.14	-0.11	-0.012
height(cm)	-0.48	1	0.68	0.24	0.24	0.11	0.011	0.16	-0.22	-0.059	0.54	0.45	0.22	0.19
weight(kg)	-0.32	0.68	1	0.17	0.17	0.27	0.15	0.33	-0.37	0.049	0.49	0.38	0.42	0.3
eyesight(left)	-0.33	0.24	0.17	1	0.69	0.009	-0.068	0.029	-0.023	-0.012	0.14	0.092	0.064	0.026
eyesight(right)	-0.33	0.24	0.17	0.69	1	0.014	-0.073	0.033	-0.024	-0.0063	0.14	0.095	0.068	0.033
relaxation	0.05	0.11	0.27	0.009	0.014	1	0.16	0.21	-0.092	0.053	0.23	0.094	0.18	0.22
fasting blood sugar	0.21	0.011	0.15	-0.068	-0.073	0.16	1	0.23	-0.13	-0.04	0.099	0.054	0.15	0.2
triglyceride	0.013	0.16	0.33	0.029	0.033	0.21	0.23	1	-0.43	0.03	0.27	0.15	0.3	0.37
HDL	0.0083	-0.22	-0.37	-0.023	-0.024	-0.092	-0.13	-0.43	1	-0.069	-0.25	-0.2	-0.23	-0.1
LDL	0.058	-0.059	0.049	-0.012	-0.0063	0.053	-0.04	0.03	-0.069	1	0.065	0.046	0.085	0.016
hemoglobin	-0.26	0.54	0.49	0.14	0.14	0.23	0.099	0.27	-0.25	0.065	1	0.45	0.33	0.29
serum creatinine	-0.14	0.45	0.38	0.092	0.095	0.094	0.054	0.15	-0.2	0.046	0.45	1	0.17	0.17
ALT	-0.11	0.22	0.42	0.064	0.068	0.18	0.15	0.3	-0.23	0.085	0.33	0.17	1	0.5
Gtp	-0.012	0.19	0.3	0.026	0.033	0.22	0.2	0.37	-0.1	0.016	0.29	0.17	0.5	1

```
# Point Biserial test between numerical variables and target categorial variable
```

```
from scipy.stats import pointbiserialr


# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["age"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and age")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')



# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["height(cm)"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and height(cm)")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')




# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["weight(kg)"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and weight(kg)")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')



# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["eyesight(left)"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and eyesight(left)")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')




# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["eyesight(right)"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and eyesight(right)")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')





# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["relaxation"]
```

```
# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and relaxation")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["fasting blood sugar"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and fasting blood sugar")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["triglyceride"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and triglyceride")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["HDL"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and HDL")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["LDL"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and LDL")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["hemoglobin"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
```

```
print("Point-biserial correlation result for Smoking and hemoglobin")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')



# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["serum creatinine"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and serum creatinine")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')



# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["ALT"]

# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and ALT")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')



# Select the binary and continuous variables
binary_var = new_df ["smoking"]
continuous_var = new_df ["Gtp"]
```

```
# Calculate the point-biserial correlation coefficient
r_pb, p_value = pointbiserialr(binary_var, continuous_var)
alpha=0.05
print("Point-biserial correlation result for Smoking and Gtp")
print("Point-biserial correlation coefficient:", r_pb)
print("P-value:", p_value)
if p <= alpha:
    print('Correlated (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
Point-biserial correlation result for Smoking and age
Point-biserial correlation coefficient: -0.16932003540717697
P-value: 2.851830580380366e-207
Correlated (reject H0)
Point-biserial correlation result for Smoking and height(cm)
Point-biserial correlation coefficient: 0.3957231677997295
P-value: 0.0
Correlated (reject H0)
Point-biserial correlation result for Smoking and weight(kg)
Point-biserial correlation coefficient: 0.3030128002908918
P-value: 0.0
Correlated (reject H0)
Point-biserial correlation result for Smoking and eyesight(left)
Point-biserial correlation coefficient: 0.09535593715018403
P-value: 2.009153736703375e-66
Correlated (reject H0)
Point-biserial correlation result for Smoking and eyesight(right)
Point-biserial correlation coefficient: 0.10408779508383485
P-value: 7.423698239432036e-79
Correlated (reject H0)
Point-biserial correlation result for Smoking and relaxation
Point-biserial correlation coefficient: 0.10271846702778656
P-value: 7.821518695686961e-77
Correlated (reject H0)
Point-biserial correlation result for Smoking and fasting blood sugar
Point-biserial correlation coefficient: 0.09499050920611098
P-value: 6.298966635265718e-66
Correlated (reject H0)
Point-biserial correlation result for Smoking and triglyceride
Point-biserial correlation coefficient: 0.2489267522197282
P-value: 0.0
Correlated (reject H0)
Point-biserial correlation result for Smoking and HDL
Point-biserial correlation coefficient: -0.18321969058712761
P-value: 6.083515653313439e-243
```

```
Correlated (reject H0)
Point-biserial correlation result for Smoking and LDL
Point-biserial correlation coefficient: -0.05684539705552884
P-value: 1.2067548862549454e-24
Correlated (reject H0)
Point-biserial correlation result for Smoking and hemoglobin
Point-biserial correlation coefficient: 0.4008163424929112
P-value: 0.0
Correlated (reject H0)
Point-biserial correlation result for Smoking and serum creatinine
Point-biserial correlation coefficient: 0.25058092348469024
P-value: 0.0
Correlated (reject H0)
Point-biserial correlation result for Smoking and ALT
Point-biserial correlation coefficient: 0.15861909007699465
P-value: 7.840452173195513e-182
Correlated (reject H0)
Point-biserial correlation result for Smoking and Gtp
Point-biserial correlation coefficient: 0.2867833165025298
P-value: 0.0
Correlated (reject H0)
```

```
# Data Partitioning

new_df['smoking'].value_counts()
```

```
0    20701
1    11750
Name: smoking, dtype: int64
```

```
X = new_df.drop('smoking',axis=1)
y=new_df['smoking']

from imblearn.over_sampling import SMOTE

smote=SMOTE()

X_new,y_new = smote.fit_resample(X,y)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_new, y_new, test_size=0.2, random_state=25)
```

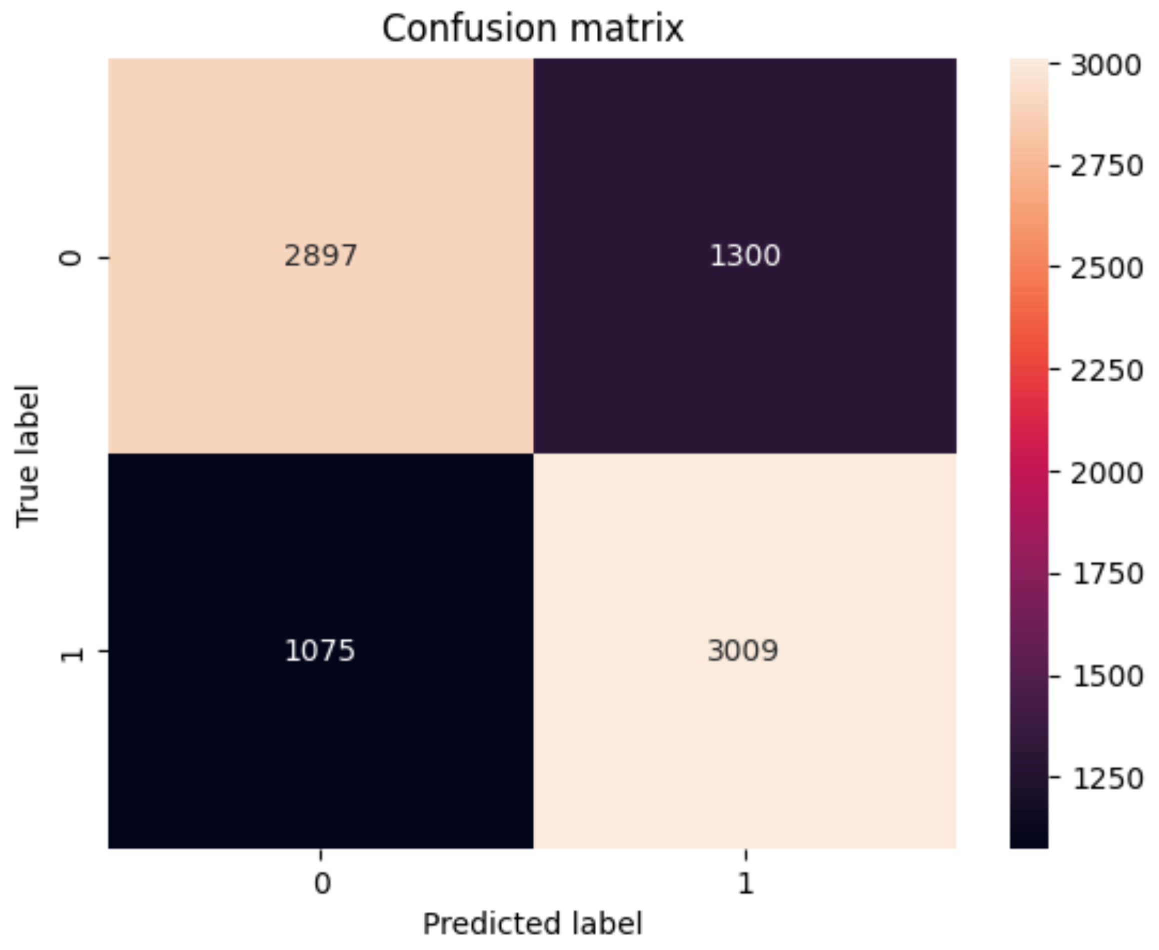
```
#Logistic Regression

from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state=250)
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾        LogisticRegression
LogisticRegression(random_state=250)
```

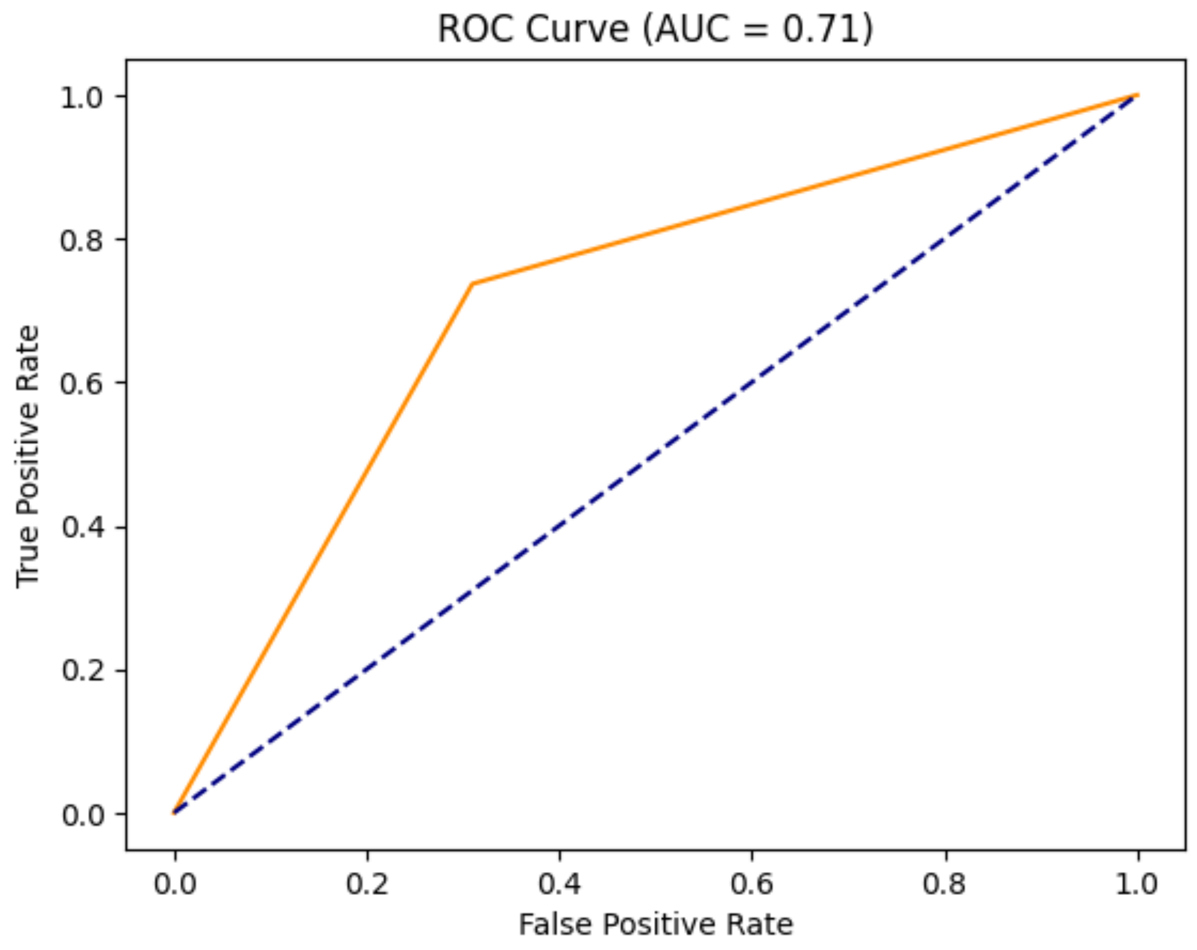
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import f1_score

# Accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# calculate the F1 score
f1 = f1_score(y_test, y_pred)

print('F1 score:', f1)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Get TP, FP, TN, FN
tn, fp, fn, tp = cm.ravel()

# Calculate sensitivity and specificity
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

# Print the results
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
```

```
# Visualize confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion matrix')
plt.show()

# Compute ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
print('AUC score:', auc)

# Plot ROC curve
plt.plot(fpr, tpr, color='darkorange', label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.2f})'.format(auc))
plt.show()
```

```
Accuracy: 0.7131988890230648
F1 score: 0.717026093172882
Sensitivity: 0.7367776689520078
Specificity: 0.6902549440076245
```



```
AUC score: 0.7135163064798161
```



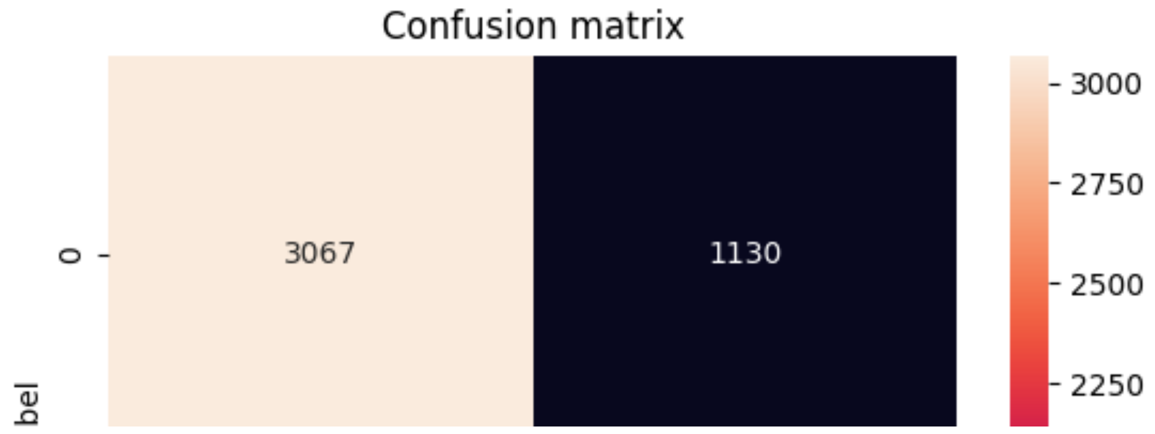
```
#Decision tree classifier

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(min_samples_split=5, random_state=42)
model.fit(X_train, y_train)
```

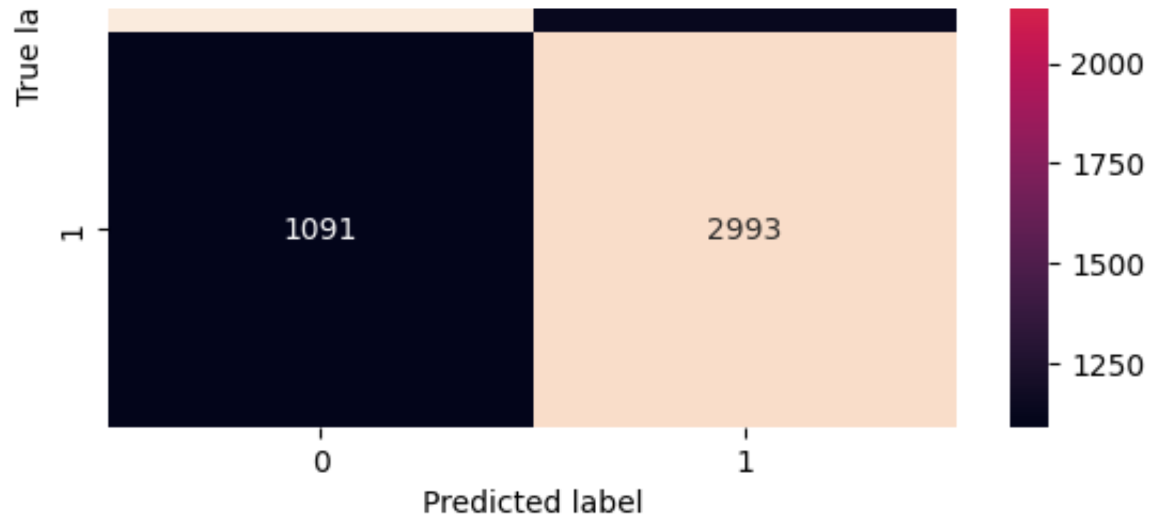
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(min_samples_split=5, random_state=42)
```

```
# Accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# calculate the F1 score
f1 = f1_score(y_test, y_pred)

print('F1 score:', f1)
```

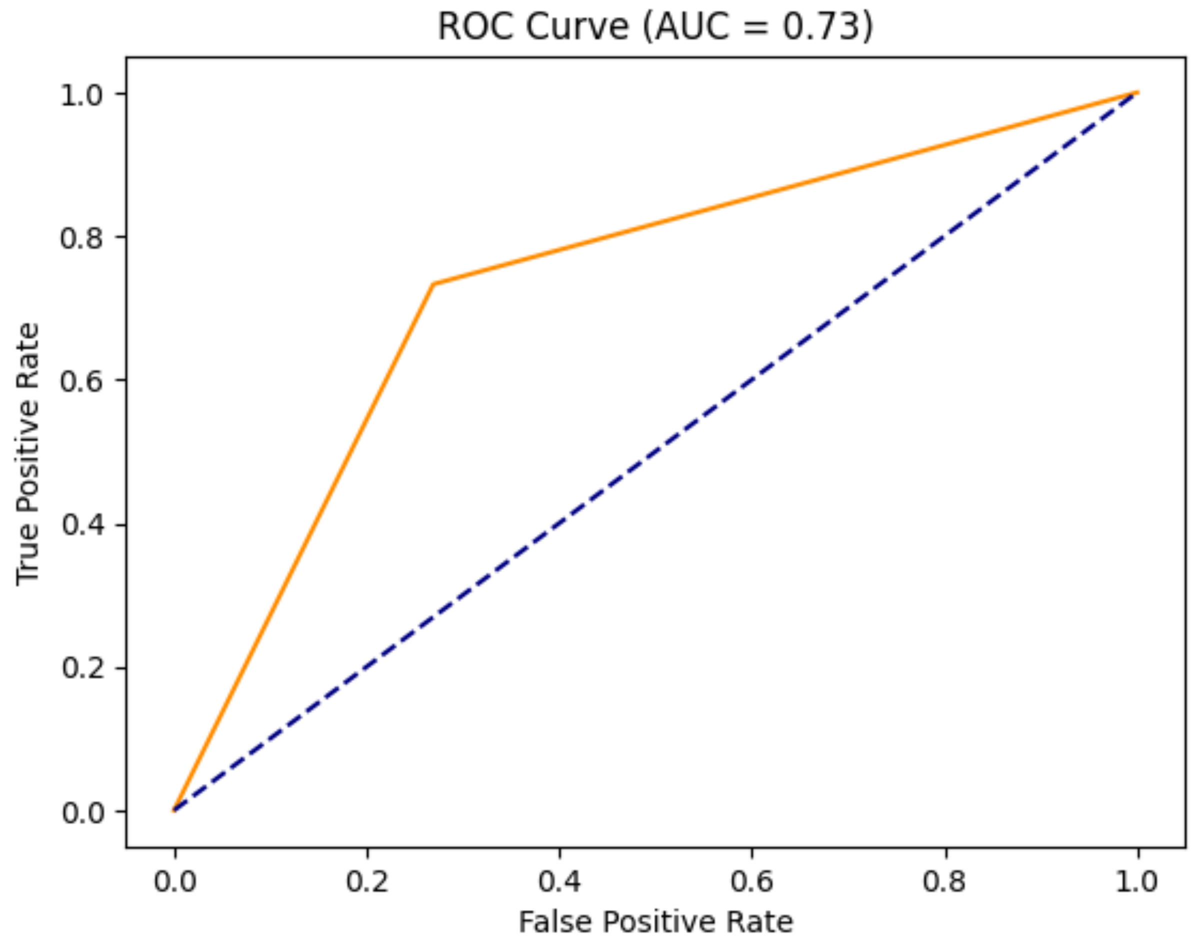
```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Get TP, FP, TN, FN
tn, fp, fn, tp = cm.ravel()

# Calculate sensitivity and specificity
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

# Print the results
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)

# Visualize confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion matrix')
plt.show()

# Compute ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
print('AUC score:', auc)

# Plot ROC curve
plt.plot(fpr, tpr, color='darkorange', label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.2f})'.format(auc))
plt.show()
```

```
Accuracy: 0.7317956768506219
F1 score: 0.7293773607895698
Sensitivity: 0.7328599412340843
Specificity: 0.7307600667143197
```





```
AUC score: 0.731810003974202
```



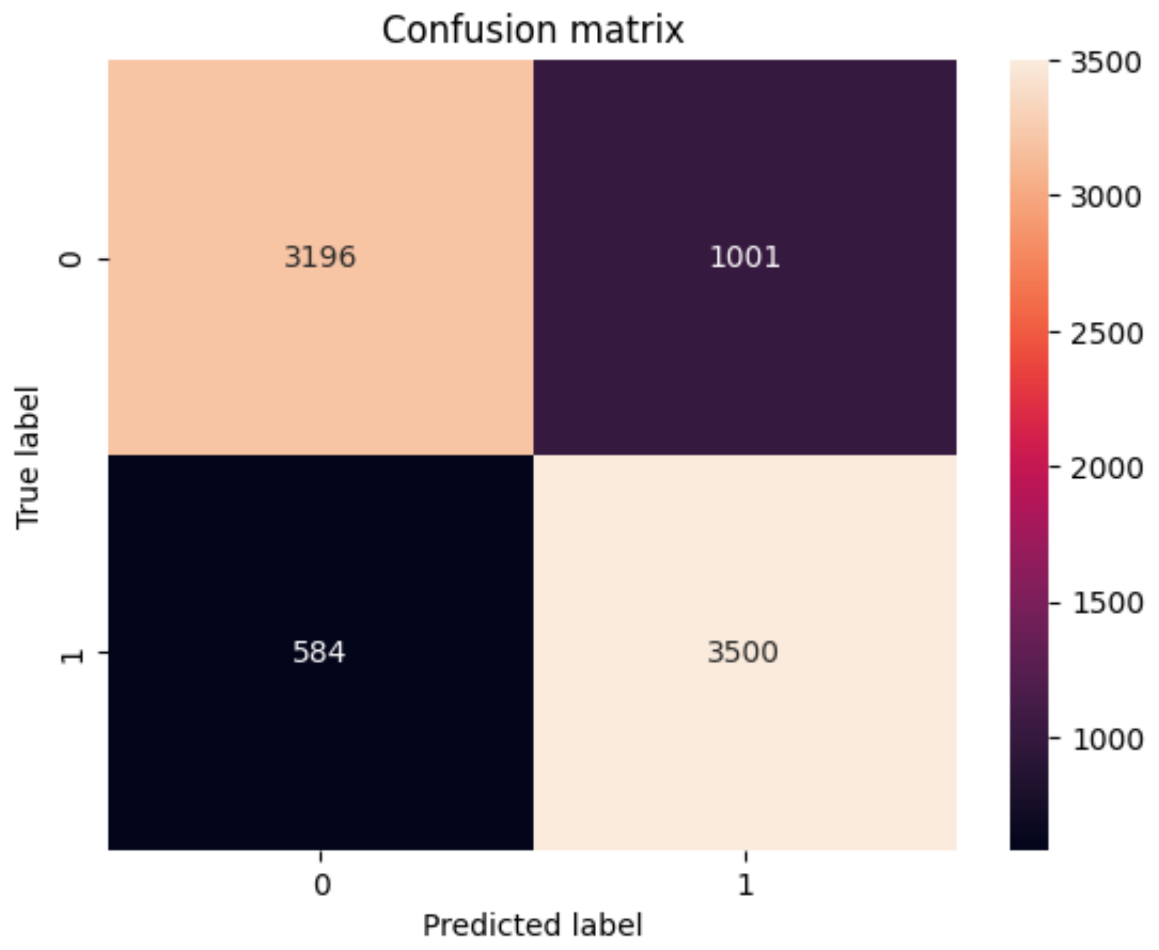
```
# Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

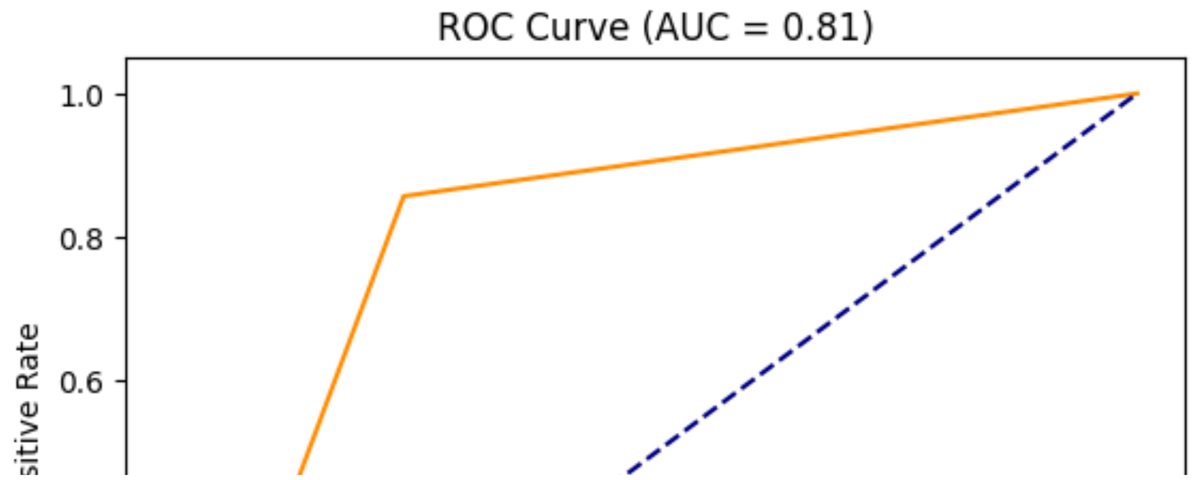
```
# Accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# calculate the F1 score
f1 = f1_score(y_test, y_pred)

print('F1 score:', f1)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Get TP, FP, TN, FN
tn, fp, fn, tp = cm.ravel()

# Calculate sensitivity and specificity
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

# Print the results
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)

# Visualize confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion matrix')
plt.show()

# Compute ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
print('AUC score:', auc)
```

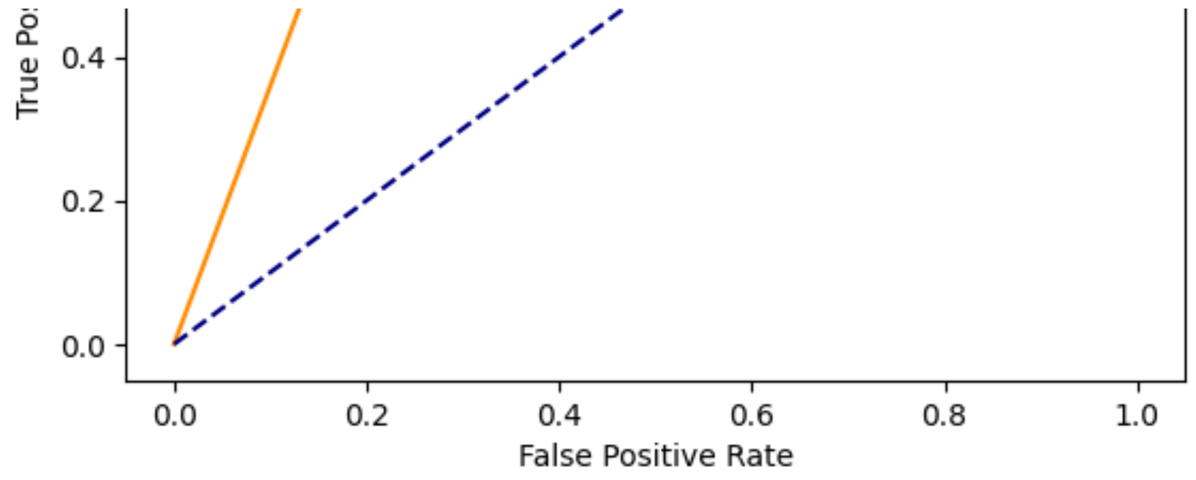
```
# Plot ROC curve
plt.plot(fpr, tpr, color='darkorange', label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.2f})'.format(auc))
plt.show()
```

```
Accuracy: 0.8085979954111823
F1 score: 0.81537565521258
Sensitivity: 0.8570029382957884
Specificity: 0.7614963068858709
```



```
AUC score: 0.8092496225908297
```





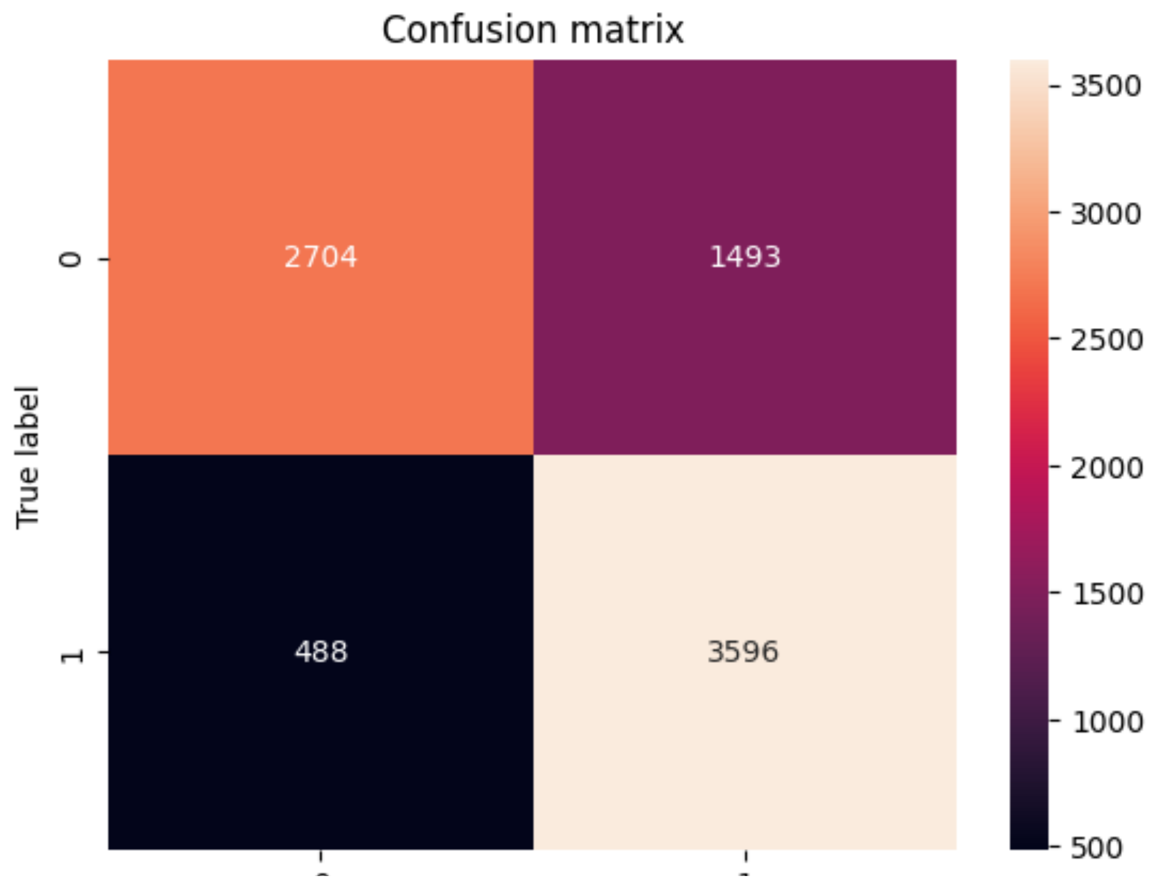
```
# KNN classifier

from sklearn.neighbors import KNeighborsClassifier
# Create a KNN classifier with k=3
model = KNeighborsClassifier(n_neighbors=3, weights='distance', metric='euclidean')

# Fit the classifier to the training data
model.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(metric='euclidean', n_neighbors=3, weights='distance')
```

```
# Accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# calculate the F1 score
f1 = f1_score(y_test, y_pred)

print('F1 score:', f1)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Get TP, FP, TN, FN
tn, fp, fn, tp = cm.ravel()

# Calculate sensitivity and specificity
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

# Print the results
```

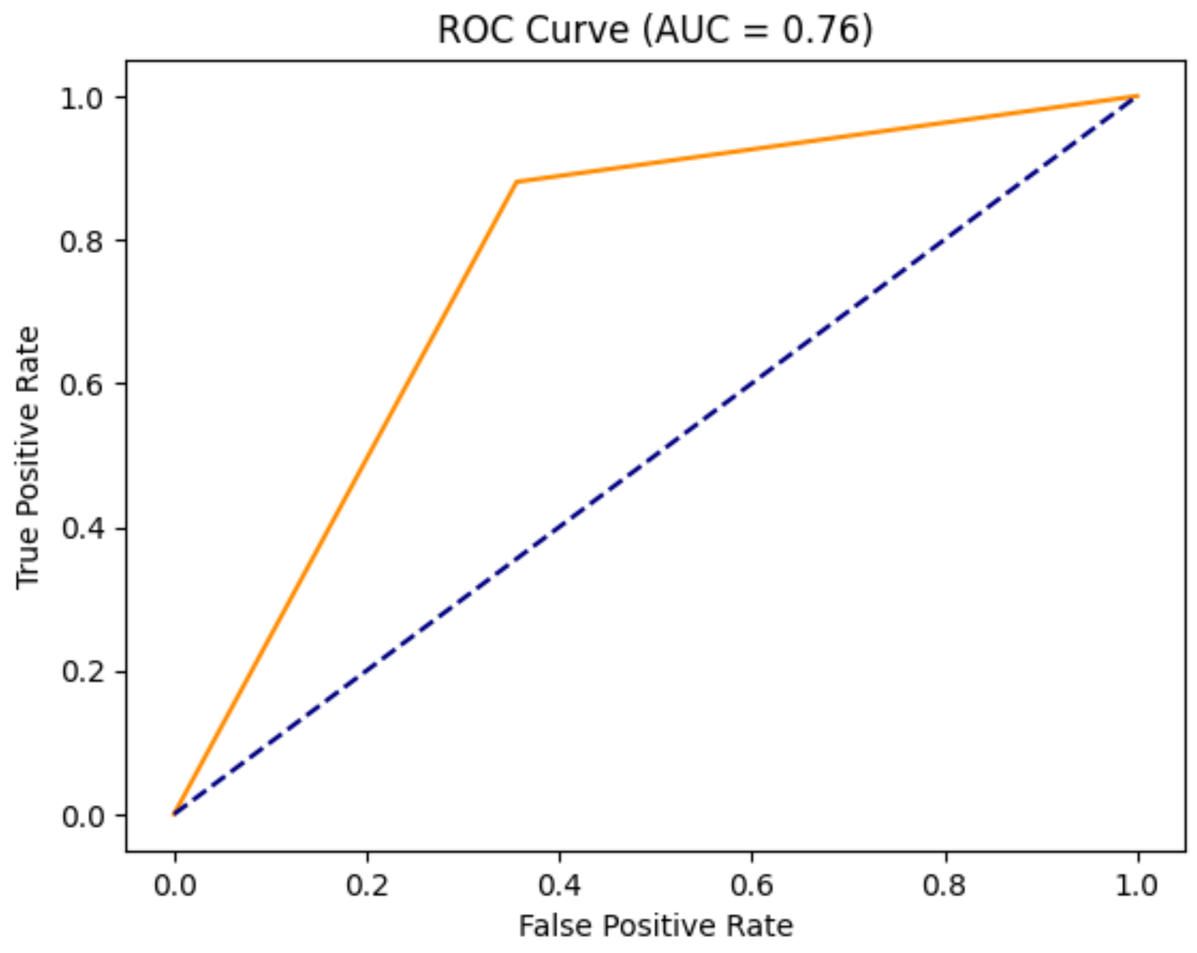
```
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)

# Visualize confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion matrix')
plt.show()

# Compute ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
print('AUC score:', auc)

# Plot ROC curve
plt.plot(fpr, tpr, color='darkorange', label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.2f})'.format(auc))
plt.show()
```

```
Accuracy: 0.760777683854607
F1 score: 0.7840401177368364
Sensitivity: 0.8805093046033301
Specificity: 0.6442697164641411
```

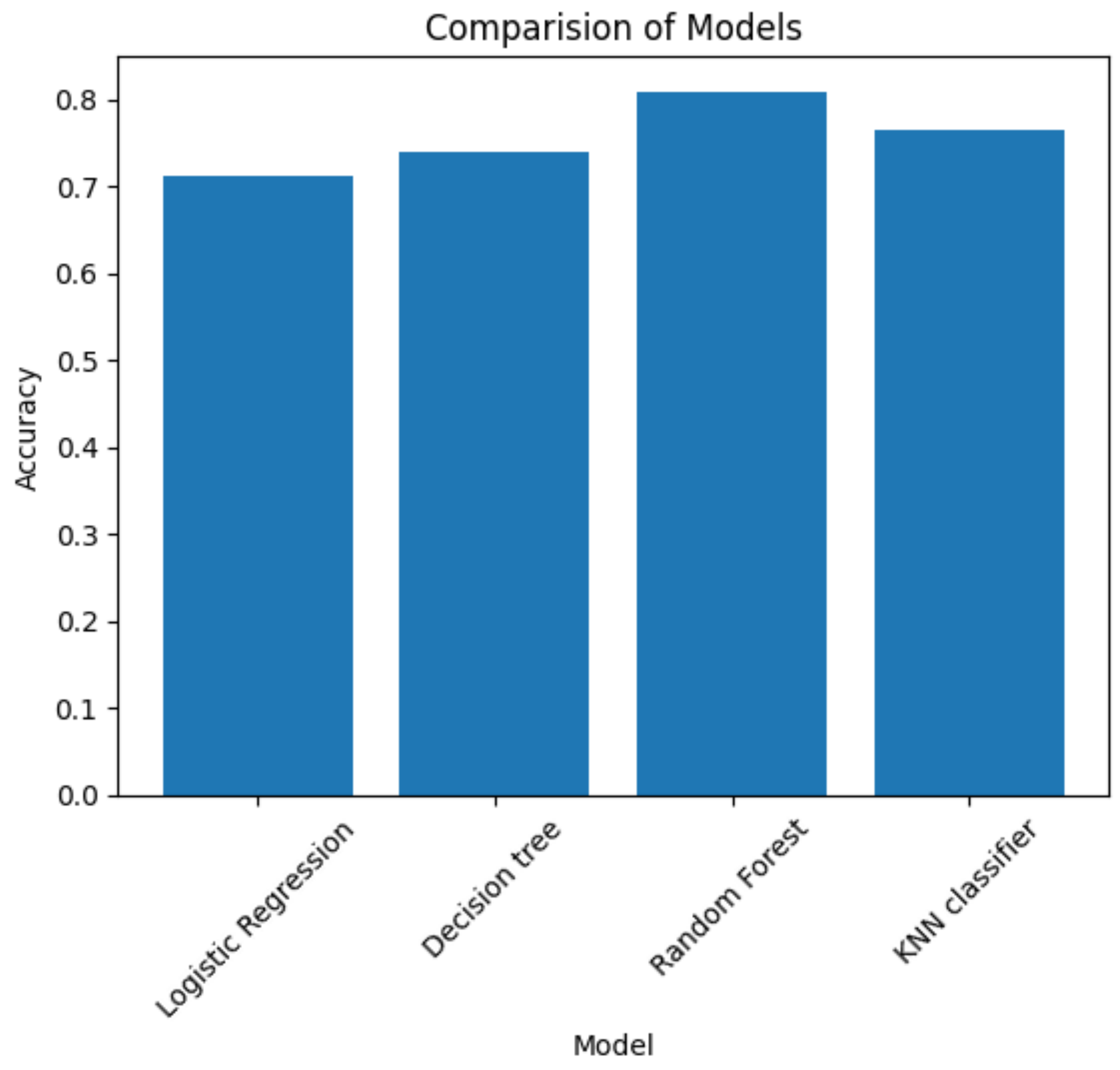


0 1
Predicted label

```
AUC score: 0.7623895105337356
```



```
# data
x = ['Logistic Regression', 'Decision tree', 'Random Forest', 'KNN classifier']
y = [0.712, 0.740, 0.809, 0.764]

# create a bar chart
plt.bar(x, y)

# add labels and title
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Comparision of Models')
plt.xticks(rotation=45)

# show the plot
plt.show()
```


Comparision of Models
Accuracy
Logistic Regression
Decision tree
Random Forest
KNN classifier
Model