Projecte AA1

April 22, 2024

1 Introducció

En aquest projecte treballarem amb les dades trobades a https://archive.ics.uci.edu/dataset/45/heart+disease, amb l'objectiu de determinar si una persona té una malaltia cardiovascular. La taula que fem servir té 76 variables, barreja de numèriques i categòriques, i és producte d'haver combinat tres taules diferents: Una de dades de suïssa, una de dades d'hongria, i una de long-beach. Es pot trobar el significat de cada variable a l'Annex

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[19]:
```python
df = pd.read_csv("merge_data.data")

df.head()
```

[19]:
```
   0  1   2  3  4  5  6  7  8    9  …  66  67  68  69  70  71  72   73   74  \
0  0  0  40  1  1  0  0 -9  2  140  …  -9  -9   1   1   1   1   1 -9.0 -9.0
1  1  0  49  0  1  0  0 -9  3  160  …  -9  -9   1   1   1   1   1 -9.0 -9.0
2  2  0  37  1  1  0  0 -9  2  130  …  -9  -9   1   1   1   1   1 -9.0 -9.0
3  3  0  48  0  1  1  1 -9  4  138  …   2  -9   1   1   1   1   1 -9.0 -9.0
4  4  0  54  1  1  0  1 -9  3  150  …   1  -9   1   1   1   1   1 -9.0 -9.0

     75
0  name
1  name
2  name
3  name
4  name

[5 rows x 76 columns]
```

2 Data preprocessing

Tots els -9 són missing values, per tant els substituim per NaN

```
[20]: df.replace(-9, np.nan, inplace=True)
      df.head()
```

```
[20]:    0  1   2  3  4  5    6   7  8      9  …   66  67   68   69   70   71  \
0  0  0  40  1  1  0  0.0 NaN  2  140.0  …  NaN NaN  1.0  1.0  1.0  1.0
1  1  0  49  0  1  0  0.0 NaN  3  160.0  …  NaN NaN  1.0  1.0  1.0  1.0
2  2  0  37  1  1  0  0.0 NaN  2  130.0  …  NaN NaN  1.0  1.0  1.0  1.0
3  3  0  48  0  1  1  1.0 NaN  4  138.0  …  2.0 NaN  1.0  1.0  1.0  1.0
4  4  0  54  1  1  0  1.0 NaN  3  150.0  …  1.0 NaN  1.0  1.0  1.0  1.0

    72  73  74   75
0  1.0 NaN NaN  name
1  1.0 NaN NaN  name
2  1.0 NaN NaN  name
3  1.0 NaN NaN  name
4  1.0 NaN NaN  name

[5 rows x 76 columns]
```

Ara transformem les variables categòriques en el tipus categòric, doncs per defecte no ho són.

```
[21]: categoriques = ['3','4','5','6','8','10','12','15','16','17','18']+[str(i) for␣
       ↪i in range(22,27)] + ['37','38','40','50'] + [str(i) for i in range(57,68)]
      for c in categoriques:
          df[c]=df[c].astype('category')
```

```
[22]: feat_names = ["id",
       "ccf",
       "age",
       "sex",
       "painloc",
       "painexer",
       "relrest",
       "pncaden",
       "cp",
       "trestbps",
       "htn",
       "chol",
       "smoke",
       "cigs",
       "years",
       "fbs",
       "dm",
       "famhist",
       "restecg",
       "ekgmo",
       "ekgday",
```

```
"ekgyr",
"dig",
"prop",
"nitr",
"pro",
"diuretic",
"proto",
"thaldur",
"thaltime",
"met",
"thalach",
"thalrest",
"tpeakbps",
"tpeakbpd",
"dummy",
"trestbpd",
"exang",
"xhypo",
"oldpeak",
"slope",
"rldv5",
"rldv5e",
"ca",
"restckm",
"exerckm",
"restef",
"restwm",
"exeref",
"exerwm",
"thal",
"thalsev",
"thalpul",
"earlobe",
"cmo",
"cday",
"cyr",
"num",
"lmt",
"ladprox",
"laddist",
"diag",
"cxmain",
"ramus",
"om1",
"om2",
"rcaprox",
"rcadist",
```

```
 "lvx1",
 "lvx2",
 "lvx3",
 "lvx4",
 "lvf",
 "cathef",
 "junk",
 "name"
]

dict_names = {str(i): feat_names[i] for i in range(len(feat_names))}
```

Canviem el header numèric pel nom de la variable que representa cada columna.

[23]:
```
df.rename(columns=dict_names, inplace=True)
df.columns
```

[23]:
```
Index(['id', 'ccf', 'age', 'sex', 'painloc', 'painexer', 'relrest', 'pncaden',
       'cp', 'trestbps', 'htn', 'chol', 'smoke', 'cigs', 'years', 'fbs', 'dm',
       'famhist', 'restecg', 'ekgmo', 'ekgday', 'ekgyr', 'dig', 'prop', 'nitr',
       'pro', 'diuretic', 'proto', 'thaldur', 'thaltime', 'met', 'thalach',
       'thalrest', 'tpeakbps', 'tpeakbpd', 'dummy', 'trestbpd', 'exang',
       'xhypo', 'oldpeak', 'slope', 'rldv5', 'rldv5e', 'ca', 'restckm',
       'exerckm', 'restef', 'restwm', 'exeref', 'exerwm', 'thal', 'thalsev',
       'thalpul', 'earlobe', 'cmo', 'cday', 'cyr', 'num', 'lmt', 'ladprox',
       'laddist', 'diag', 'cxmain', 'ramus', 'om1', 'om2', 'rcaprox',
       'rcadist', 'lvx1', 'lvx2', 'lvx3', 'lvx4', 'lvf', 'cathef', 'junk',
       'name'],
      dtype='object')
```

Eliminem les columnes indicades com a “not used”, “irrelevant”, dates de proves o que són l’agregació d’altres variables. Les columnes “not used” o “irrelevant” les identifiquem perquè així estàn indicades al lloc d’on hem extret les dades.

[24]:
```
print("Nombre de features abans:", len(df.columns))
useless_columns = ["id", "ccf", "dummy", "thalsev","thalpul", "earlobe",␣
 ↪"lvx1", "lvx2", "lvx3", "lvx4", "lvf", "cathef", "junk", "name", "restckm",␣
 ↪"exerckm", "pncaden", "cmo", "cday", "cyr", "ekgmo", "ekgday", "ekgyr"]
df.drop(useless_columns, axis = 1, inplace = True)

print("Nombre de features després:", len(df.columns))
```

```
Nombre de features abans: 76
Nombre de features després: 53
```

Eliminem les columnes categòriques amb més de 10% de missing values o les numèriques amb més de 80% de missing values

[25]:
```
print("Nombre de features abans:", len(df.columns))
too_nan = [c for c in df.columns if df[c].isna().sum()/len(df) > 0.3 and df[c].
  ↪dtype == "category" or df[c].isna().sum()/len(df) > 0.8]
df.drop(too_nan, axis = 1, inplace = True)
print("Nombre de features després:", len(df.columns))
```

```
Nombre de features abans: 53
Nombre de features després: 33
```

[26]:
```
len(df)
```

[26]:
```
617
```

[27]:
```
df.dtypes
```

[27]:
```
age            int64
sex         category
painloc     category
painexer    category
relrest     category
cp          category
trestbps     float64
htn         category
chol         float64
cigs         float64
years        float64
fbs         category
restecg     category
dig         category
prop        category
nitr        category
pro         category
diuretic    category
proto        float64
thaldur      float64
thaltime     float64
met          float64
thalach      float64
thalrest     float64
tpeakbps     float64
tpeakbpd     float64
trestbpd     float64
exang       category
xhypo       category
oldpeak      float64
rldv5        float64
rldv5e       float64
```

```
num         category
dtype: object
```

[28]:
```
for c in df.columns:
    count = df[c].isna().sum()
    if count > 0:
        print(c, df[c].dtype, count, str(count/len(df)*100) + "%")
```

```
relrest category 4 0.6482982171799028%
trestbps float64 59 9.562398703403566%
htn category 34 5.510534846029174%
chol float64 30 4.862236628849271%
cigs float64 415 67.26094003241491%
years float64 427 69.20583468395462%
fbs category 90 14.58670988654781%
restecg category 2 0.3241491085899514%
dig category 66 10.696920583468396%
prop category 64 10.372771474878444%
nitr category 63 10.21069692058347%
pro category 61 9.886547811993516%
diuretic category 80 12.965964343598054%
proto float64 112 18.152350081037277%
thaldur float64 56 9.076175040518638%
thaltime float64 384 62.23662884927067%
met float64 105 17.01782820097245%
thalach float64 55 8.914100486223662%
thalrest float64 56 9.076175040518638%
tpeakbps float64 63 10.21069692058347%
tpeakbpd float64 63 10.21069692058347%
trestbpd float64 59 9.562398703403566%
exang category 55 8.914100486223662%
xhypo category 58 9.40032414910859%
oldpeak float64 62 10.048622366288493%
rldv5 float64 143 23.176661264181522%
rldv5e float64 142 23.014586709886547%
```

Observem les columnes restants amb molts NaN per determinar la seva importància:

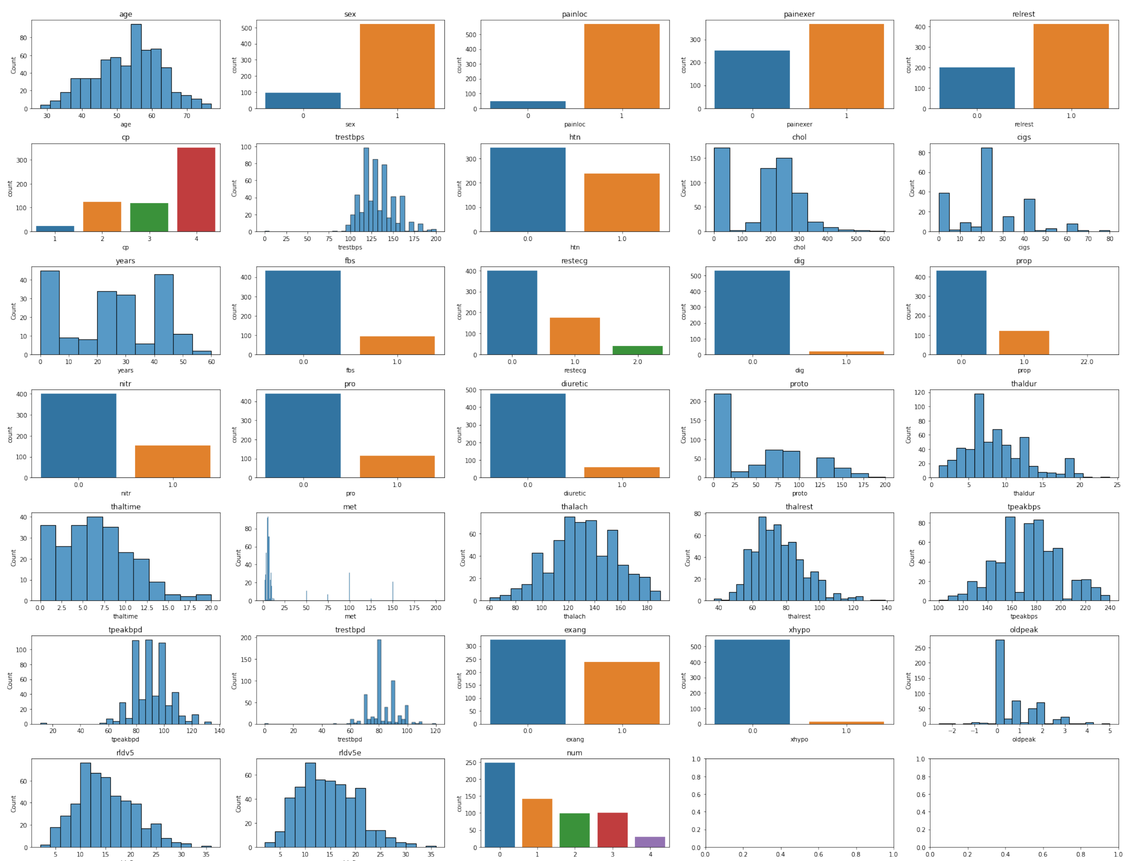
```
13 cigs (cigarettes per day)
14 years (number of years as a smoker)
17 famhist: family history of coronary artery disease (1 = yes; 0 = no)
29 thaltime (time when ST measure depression was noted)
```

A priori no les eliminarem perquè tindria sentit que estiguessin relacionades amb patir una malatia cardiovascular.

2.1 Valors estranys

Anem a veure si alguna columna té valors fora del comú, per exemple una categòrica que té valors diferents als de les categories que representa o alguna numèrica que té algun valor aillat estrany.

[29]:
```
fig, axes = plt.subplots(7,5,figsize=(26,20))

for i, c in enumerate(df.columns):
    ax = axes.reshape(-1)[i]
    if df[c].dtype.kind == 'O':
        a = sns.countplot(x=c,data=df,ax=ax)
    else:
        b = sns.histplot(x=c,data=df,ax=ax)
    t = ax.set_title(c)
plt.tight_layout()
```



- trestbps té un valor igual a 0, molt lluny de la resta de valors. Sospitem que deu ser un NaN.
- chol té 172 rows iguals a 0. Són NaN.
- tpeakbddp deu tenir un error
- trestbpd té errors
- prop té un valor 22 únic, seria un NaN

[30]:
```python
print((df["trestbps"] == 0).sum())
print((df["chol"] == 0).sum())
print((df["tpeakbpd"] < 20).sum())
print((df["trestbpd"] < 30).sum())

df["tpeakbpd"].value_counts()
df["trestbpd"].value_counts()
```

```
1
172
1
1
```

[30]:
```
80.0     194
90.0     100
70.0      67
100.0     42
85.0      25
75.0      16
95.0      13
78.0      12
60.0      11
84.0       8
74.0       7
82.0       7
65.0       6
86.0       6
94.0       6
96.0       5
88.0       5
98.0       5
110.0      5
72.0       4
92.0       3
105.0      3
64.0       2
0.0        1
50.0       1
120.0      1
104.0      1
106.0      1
58.0       1
Name: trestbpd, dtype: int64
```

Canviem els valors estranys per NaN

[31]:
```
df["chol"].replace(0, np.nan, inplace=True)
df["trestbps"].replace(0, np.nan, inplace=True)
df["tpeakbpd"].replace(11, np.nan, inplace=True)
df["trestbpd"].replace(0, np.nan, inplace=True)
df["prop"].replace(22, np.nan, inplace = True)
```

[32]:
```
for c in df.columns:
    count = df[c].isna().sum()
    if count > 0:
        print(c, df[c].dtype, count, str(count/len(df)*100) + "%")
```

```
relrest category 4 0.6482982171799028%
trestbps float64 60 9.724473257698541%
htn category 34 5.510534846029174%
chol float64 202 32.739059967585085%
cigs float64 415 67.26094003241491%
years float64 427 69.20583468395462%
fbs category 90 14.58670988654781%
restecg category 2 0.3241491085899514%
dig category 66 10.696920583468396%
prop category 65 10.53484602917342%
nitr category 63 10.21069692058347%
pro category 61 9.886547811993516%
diuretic category 80 12.965964343598054%
proto float64 112 18.152350081037277%
thaldur float64 56 9.076175040518638%
thaltime float64 384 62.23662884927067%
met float64 105 17.01782820097245%
thalach float64 55 8.914100486223662%
thalrest float64 56 9.076175040518638%
tpeakbps float64 63 10.21069692058347%
tpeakbpd float64 64 10.372771474878444%
trestbpd float64 60 9.724473257698541%
exang category 55 8.914100486223662%
xhypo category 58 9.40032414910859%
oldpeak float64 62 10.048622366288493%
rldv5 float64 143 23.176661264181522%
rldv5e float64 142 23.014586709886547%
```

Proto és una categoria però s'utilitza com a float, a banda els valors no són consistents doncs trobem valors que pertanyen a la categoria i molts que no. Convindria eliminar.

[33]:
```
df["proto"].value_counts()

df.drop("proto", axis = 1, inplace = True)
```

Cal veure si thalrest, thalach i rldv5 i rldv5e estan correlades, doncs les descripcions de les variables s'assimilen molt. No és el cas, per tant les conservem.

[34]:
```
print(df['thalrest'].corr(df['rldv5']))
print(df['thalach'].corr(df['rldv5e']))
```

```
-0.14038113014142173
-0.12134305546423077
```

[35]:
```
fig, axes = plt.subplots(7,5,figsize=(26,20))

for i, c in enumerate(df.columns):
    ax = axes.reshape(-1)[i]
    if df[c].dtype.kind == 'O':
        a = sns.countplot(x=c,data=df,ax=ax)
    else:
        b = sns.histplot(x=c,data=df,ax=ax)
    t = ax.set_title(c)
plt.tight_layout()
```

[36]:
```
len(df.columns)
```

[36]: 32

Ara veurem quins valors pren la variable que volem predir

[37]:
```
df['num'].value_counts()
```

[37]:
```
0    247
1    141
3    100
2     99
4     30
Name: num, dtype: int64
```

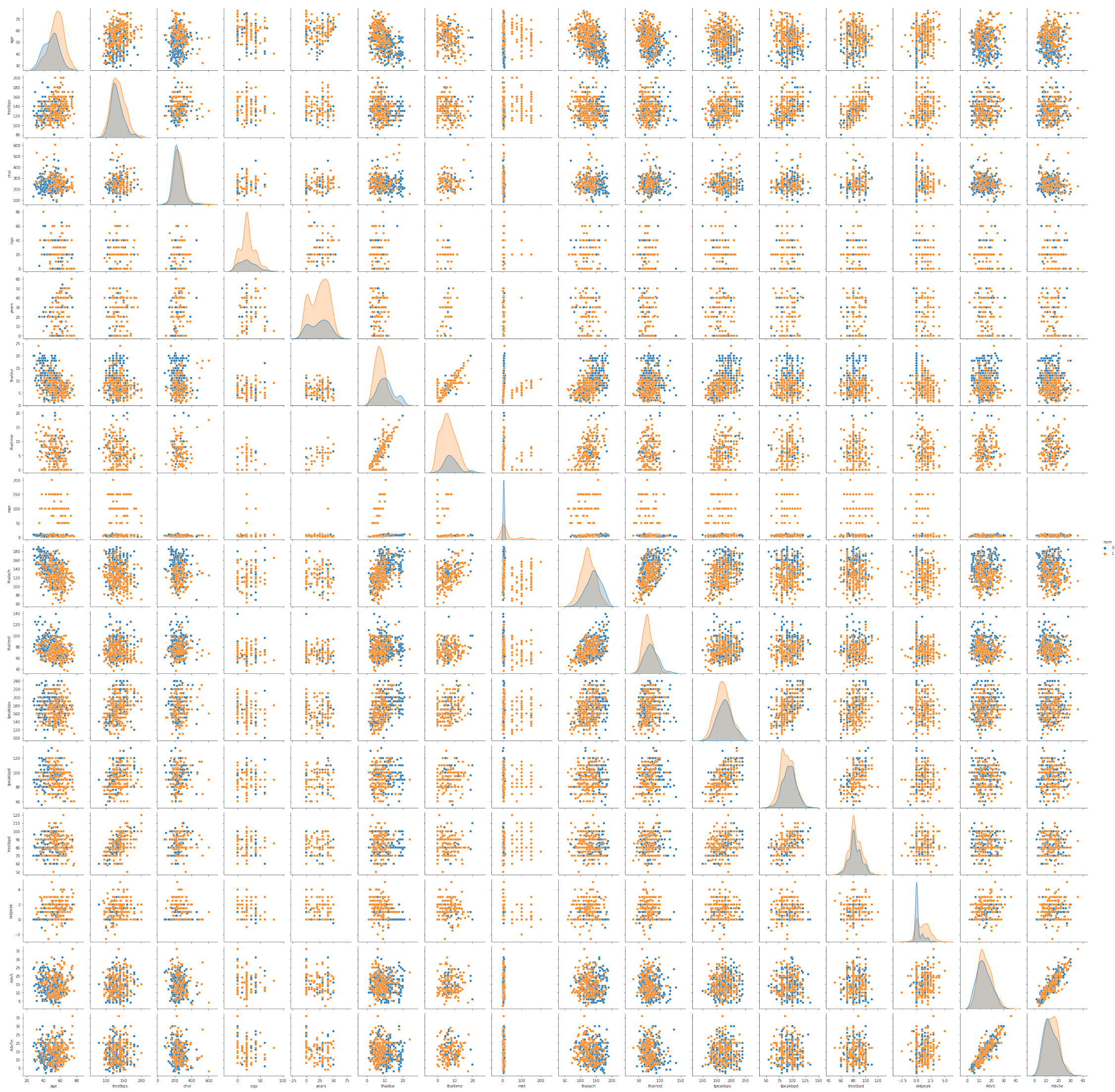
Com que tenim poques dades en general i encara menys per les categories 4, i 3, i tenint en compte que 0 vol dir no tenir malaltia cardiovascular i la resta son graus de malaltia cardiovadscular nosaltres considerarem únicament 0 o 1 (tenir o no tenir)

[38]:
```
df['num'].replace([3,2,4], 1, inplace = True)
df['num'].value_counts()
```

[38]:
```
1    370
0    247
Name: num, dtype: int64
```

De moment no toquem cap columna més, anem a veure si a priori es podran separar fàcilment les categories.

[39]:
```
sns.pairplot(data=df, hue='num');
```



Sembla que no, per tant procedirem a intentar algún algoritme d'aprenentatge supervisat per poder classificar començant amb totes les variables que tenim.

3 Modelatge

Anem a plantejar models tenint en compte que volem un classificador que utilitzi variables numèriques i categòriques.

Possibles models:

1. Regressió logística
2. LDA
3. QDA
4. Naive Bayes classfier
5. Random forest

```
[40]: from sklearn.model_selection import train_test_split,  KFold, cross_validate,␣
      ↪GridSearchCV
      from sklearn.preprocessing import MinMaxScaler
      from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,␣
      ↪QuadraticDiscriminantAnalysis
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.naive_bayes import BernoulliNB, GaussianNB, CategoricalNB
      from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
      from sklearn.ensemble import RandomForestClassifier


      from sklearn.metrics import confusion_matrix, classification_report,␣
      ↪accuracy_score,  precision_score, recall_score, f1_score
```

```
[41]: cat = df.select_dtypes(include=['category']).columns.tolist()
      cat.remove('num')
      num = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
```

Com que els algoritmes que volem fer servir no accepten NaN cal “afegir” valors. En el cas de les numèriques posarem la mitjana, i en les categòriques la moda.

```
[42]: from sklearn.impute import SimpleImputer
      imputer_mean = SimpleImputer(strategy='mean')
      imputer_moda = SimpleImputer(strategy='most_frequent')
      for c in num:
        df[c] = imputer_mean.fit_transform(df[[c]])
      for c in cat:
        df[c] = imputer_moda.fit_transform(df[[c]])
```

Definim conjunts de test i entrenament

```
[51]: X = df.loc[:, df.columns != 'num']
      y = df['num']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
      ↪random_state=42)
      results_df = pd.DataFrame(index=[], columns= ['Accuracy', 'F1 Macro',␣
      ↪'Precision Macro', 'Recall Macro'])
```

3.1 Regressió logística:

Cal preprocessar les dades per evitar biaixos a causa dels diferents ordres de magnitud, així com també cal aplicar one-hot encoding a les variables categòriques.

```
[52]: def preprocessing(X, y, scaler=None):
          if scaler is None:
              scaler = MinMaxScaler()
              X.loc[:,num] = scaler.fit_transform(X[num])
          else:
```

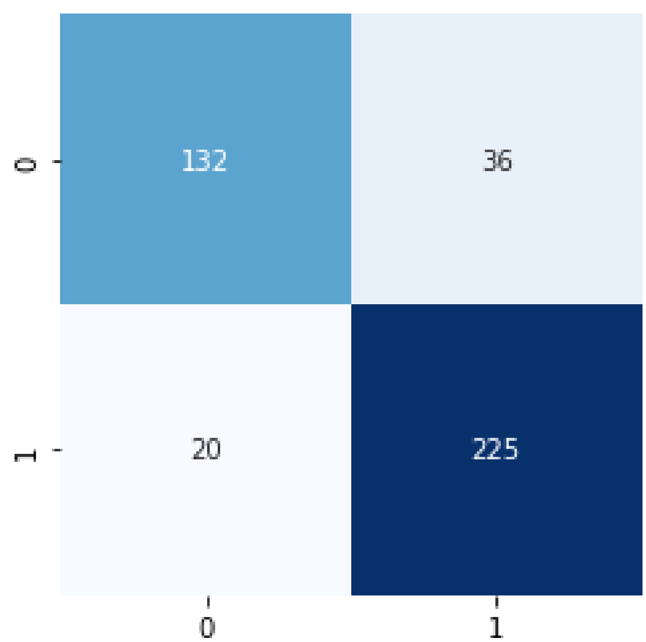
```
        X.loc[:,num] = scaler.transform(X[num])
    X = pd.get_dummies(X, columns = cat)
    return X, y, scaler


X_train, y_train, scaler = preprocessing(X_train,y_train)
X_test, y_test, _ = preprocessing(X_test,y_test,scaler)
```

[53]:

```
logModel=LogisticRegression(solver='lbfgs', max_iter=10000)
logModel.fit(X_train, y_train)
predictions = logModel.predict(X_test)

cmatrix = confusion_matrix(y_train, pd.Series(logModel.predict(X_train)))
sns.heatmap(cmatrix, square=True, annot=True, cmap='Blues', fmt='d', cbar=False)

cross_val_results = pd.DataFrame(cross_validate(logModel , X_train, y_train, cv␣
 ↪= 5, scoring = ['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'] ))
results_df.loc['RegLog',:] = cross_val_results[['test_accuracy',␣
 ↪'test_f1_macro','test_precision_macro', 'test_recall_macro']].mean().values
results_df
```

[53]:

	Accuracy	F1 Macro	Precision Macro	Recall Macro
RegLog	0.825654	0.816386	0.822761	0.813909

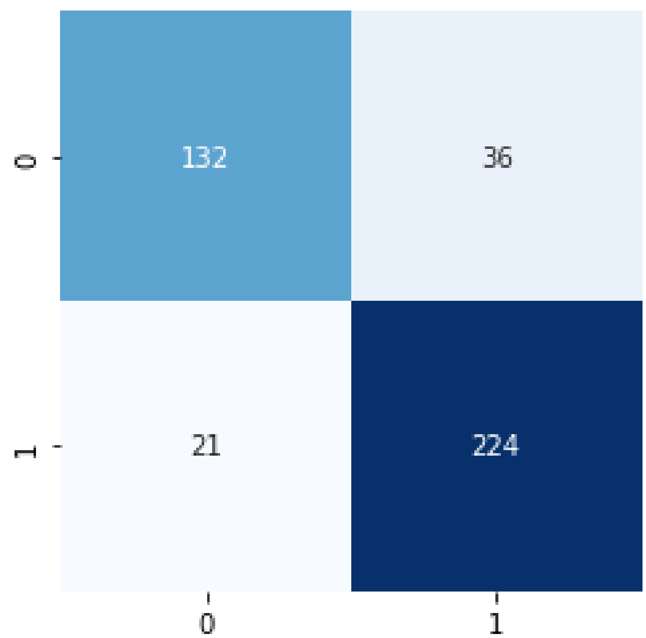


3.2 LDA

[54]:
```
lda_model = LinearDiscriminantAnalysis()
lda_model.fit(X_train, y_train)
predictions = lda_model.predict(X_test)

cmatrix = confusion_matrix(y_train, pd.Series(lda_model.predict(X_train)))
sns.heatmap(cmatrix, square=True, annot=True, cmap='Blues', fmt='d', cbar=False)

cross_val_results = pd.DataFrame(cross_validate(lda_model , X_train, y_train,␣
↪cv = 5, scoring = ['accuracy', 'f1_macro', 'precision_macro',␣
↪'recall_macro'] ))
results_df.loc['LDA',:] = cross_val_results[['test_accuracy',␣
↪'test_f1_macro','test_precision_macro', 'test_recall_macro']].mean().values
results_df
```

[54]:

	Accuracy	F1 Macro	Precision Macro	Recall Macro
RegLog	0.825654	0.816386	0.822761	0.813909
LDA	0.825742	0.816043	0.823103	0.813909



3.3 QDA

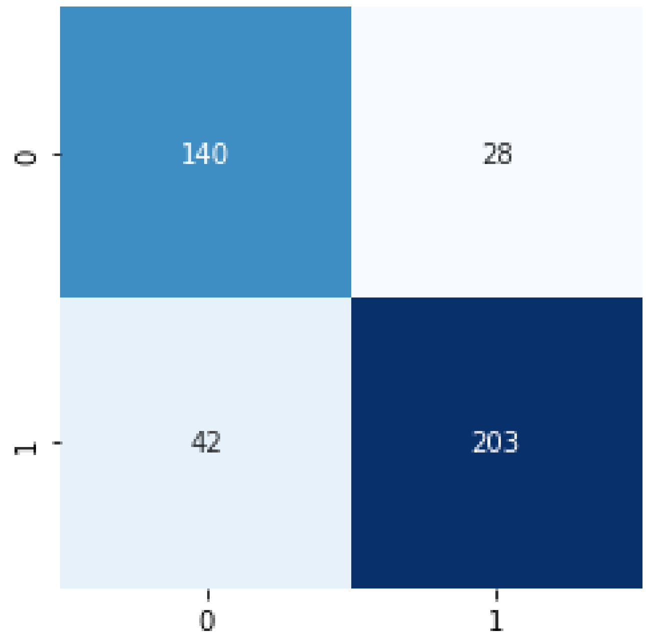
[55]:
```
qda_model = QuadraticDiscriminantAnalysis(reg_param=0.1).fit(X_train, y_train)
predictions = qda_model.predict(X_test)

cmatrix = confusion_matrix(y_train, pd.Series(qda_model.predict(X_train)))
```

```
sns.heatmap(cmatrix, square=True, annot=True, cmap='Blues', fmt='d', cbar=False)

cross_val_results = pd.DataFrame(cross_validate(qda_model , X_train, y_train,␣
 ↪cv = 5, scoring = ['accuracy', 'f1_macro', 'precision_macro',␣
 ↪'recall_macro'] ))
results_df.loc['QDA',:] = cross_val_results[['test_accuracy',␣
 ↪'test_f1_macro','test_precision_macro', 'test_recall_macro']].mean().values
results_df
```

```
C:\Users\rbarg\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:878:
UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\rbarg\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:878:
UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\rbarg\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:878:
UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\rbarg\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:878:
UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\rbarg\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:878:
UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\rbarg\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:878:
UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
```

[55]:

	Accuracy	F1 Macro	Precision Macro	Recall Macro
RegLog	0.825654	0.816386	0.822761	0.813909
LDA	0.825742	0.816043	0.823103	0.813909
QDA	0.791713	0.787245	0.78691	0.79263

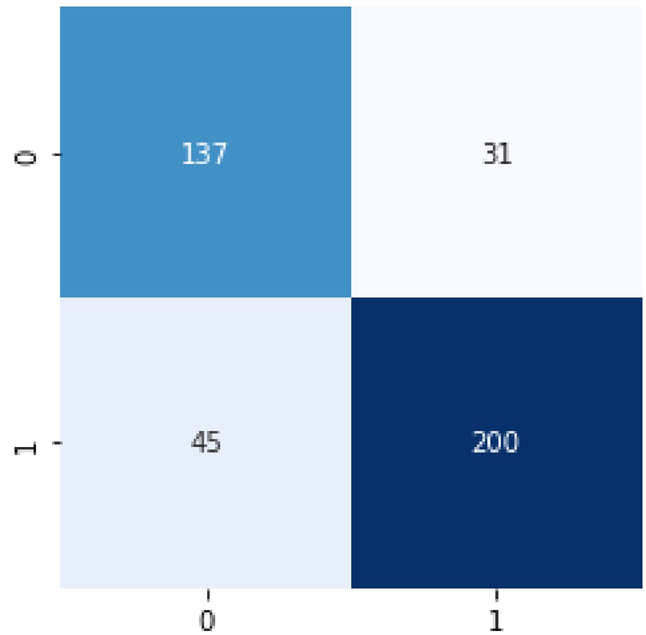


3.4 Gaussian naive bayes

[56]:
```
gaussian_nb = GaussianNB()

gaussian_nb.fit(X_train,y_train)

cmatrix = confusion_matrix(y_train, pd.Series(gaussian_nb.predict(X_train)))
sns.heatmap(cmatrix, square=True, annot=True, cmap='Blues', fmt='d', cbar=False)

cross_val_results = pd.DataFrame(cross_validate(gaussian_nb , X_train, y_train,␣
↪cv = 5, scoring = ['accuracy', 'f1_macro', 'precision_macro',␣
↪'recall_macro'] ))
results_df.loc['Naive Bayes',:] = cross_val_results[['test_accuracy',␣
↪'test_f1_macro','test_precision_macro', 'test_recall_macro']].mean().values
results_df
```

[56]:

	Accuracy	F1 Macro	Precision Macro	Recall Macro
RegLog	0.825654	0.816386	0.822761	0.813909
LDA	0.825742	0.816043	0.823103	0.813909
QDA	0.791713	0.787245	0.78691	0.79263
Naive Bayes	0.786659	0.78206	0.796938	0.794396

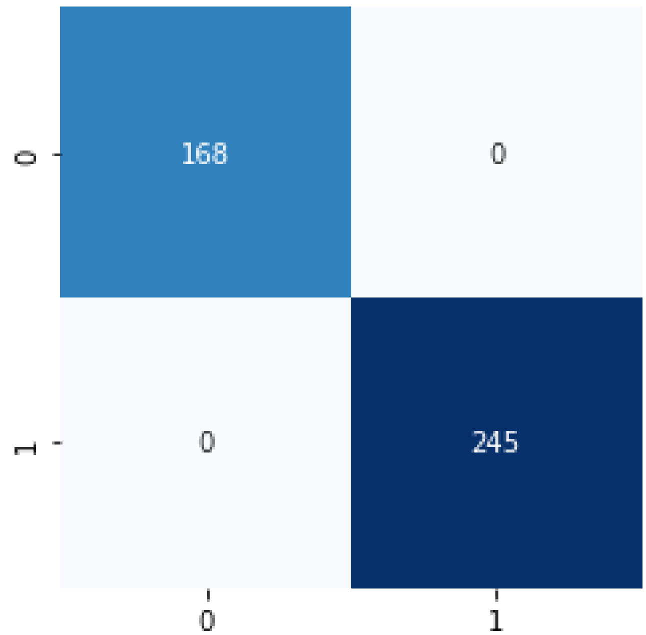


3.5 Random forest

[57]:
```
clf = RandomForestClassifier(n_estimators = 100)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

cmatrix = confusion_matrix(y_train, pd.Series(clf.predict(X_train)))
sns.heatmap(cmatrix, square=True, annot=True, cmap='Blues', fmt='d', cbar=False)

cross_val_results = pd.DataFrame(cross_validate(clf , X_train, y_train, cv = 5,␣
 ↪scoring = ['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'] ))
results_df.loc['Random forest',:] = cross_val_results[['test_accuracy',␣
 ↪'test_f1_macro','test_precision_macro', 'test_recall_macro']].mean().values
results_df
```

[57]:

	Accuracy	F1 Macro	Precision Macro	Recall Macro
RegLog	0.825654	0.816386	0.822761	0.813909
LDA	0.825742	0.816043	0.823103	0.813909
QDA	0.791713	0.787245	0.78691	0.79263
Naive Bayes	0.786659	0.78206	0.796938	0.794396
Random forest	0.832883	0.8251	0.829886	0.822554



4 Annex:

Complete attribute documentation:

```
  0 id: patient identification number
  1 ccf: social security number (I replaced this with a dummy value of 0)
  2 age: age in years
  3 sex: sex (1 = male; 0 = female)
  4 painloc: chest pain location (1 = substernal; 0 = otherwise)
  5 painexer (1 = provoked by exertion; 0 = otherwise)
  6 relrest (1 = relieved after rest; 0 = otherwise)
  7 pncaden (sum of 5, 6, and 7)
  8 cp: chest pain type
    -- Value 1: typical angina
    -- Value 2: atypical angina
    -- Value 3: non-anginal pain
    -- Value 4: asymptomatic
 9 trestbps: resting blood pressure (in mm Hg on admission to the hospital)
 10 htn
 11 chol: serum cholestoral in mg/dl
 12 smoke: I believe this is 1 = yes; 0 = no (is or is not a smoker)
 13 cigs (cigarettes per day)
 14 years (number of years as a smoker)
 15 fbs: (fasting blood sugar > 120 mg/dl)  (1 = true; 0 = false)
 16 dm (1 = history of diabetes; 0 = no such history)
 17 famhist: family history of coronary artery disease (1 = yes; 0 = no)
```

```
18 restecg: resting electrocardiographic results
   -- Value 0: normal
   -- Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depress
   -- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
19 ekgmo (month of exercise ECG reading)
20 ekgday(day of exercise ECG reading)
21 ekgyr (year of exercise ECG reading)
22 dig (digitalis used furing exercise ECG: 1 = yes; 0 = no)
23 prop (Beta blocker used during exercise ECG: 1 = yes; 0 = no)
24 nitr (nitrates used during exercise ECG: 1 = yes; 0 = no)
25 pro (calcium channel blocker used during exercise ECG: 1 = yes; 0 = no)
26 diuretic (diuretic used used during exercise ECG: 1 = yes; 0 = no)
27 proto: exercise protocol
     1 = Bruce
     2 = Kottus
     3 = McHenry
     4 = fast Balke
     5 = Balke
     6 = Noughton
     7 = bike 150 kpa min/min  (Not sure if "kpa min/min" is what was written!)
     8 = bike 125 kpa min/min
     9 = bike 100 kpa min/min
    10 = bike 75 kpa min/min
    11 = bike 50 kpa min/min
    12 = arm ergometer
28 thaldur: duration of exercise test in minutes
29 thaltime: time when ST measure depression was noted
30 met: mets achieved
31 thalach: maximum heart rate achieved
32 thalrest: resting heart rate
33 tpeakbps: peak exercise blood pressure (first of 2 parts)
34 tpeakbpd: peak exercise blood pressure (second of 2 parts)
35 dummy
36 trestbpd: resting blood pressure
37 exang: exercise induced angina (1 = yes; 0 = no)
38 xhypo: (1 = yes; 0 = no)
39 oldpeak = ST depression induced by exercise relative to rest
40 slope: the slope of the peak exercise ST segment
   -- Value 1: upsloping
   -- Value 2: flat
   -- Value 3: downsloping
41 rldv5: height at rest
42 rldv5e: height at peak exercise
43 ca: number of major vessels (0-3) colored by flourosopy
44 restckm: irrelevant
45 exerckm: irrelevant
46 restef: rest raidonuclid (sp?) ejection fraction
47 restwm: rest wall (sp?) motion abnormality
```

```
   0 = none
   1 = mild or moderate
   2 = moderate or severe
   3 = akinesis or dyskmem (sp?)
48 exeref: exercise radinalid (sp?) ejection fraction
49 exerwm: exercise wall (sp?) motion
50 thal: A blood disorder called thalassemia 3 = normal; 6 = fixed defect; 7 = reversable def
51 thalsev: not used
52 thalpul: not used
53 earlobe: not used
54 cmo: month of cardiac cath (sp?)  (perhaps "call")
55 cday: day of cardiac cath (sp?)
56 cyr: year of cardiac cath (sp?)
57 num: diagnosis of heart disease (angiographic disease status)
   -- Value 0: < 50% diameter narrowing
   -- Value 1: > 50% diameter narrowing
   (in any major vessel: attributes 59 through 68 are vessels)
58 lmt
59 ladprox
60 laddist
61 diag
62 cxmain
63 ramus
64 om1
65 om2
66 rcaprox
67 rcadist
68 lvx1: not used
69 lvx2: not used
70 lvx3: not used
71 lvx4: not used
72 lvf: not used
73 cathef: not used
74 junk: not used
75 name: last name of patient
```