Heart Disease Classification

December 2, 2019

1 Cardiovascular Heart Disease Risk Prediction

2 Machine Learning Final Project

Name: Troy Zhongyi Zhang
Netid: zhongyiz@uchicago.edu

2.0.1 Importing packages and the dataset

[1]:
```
%matplotlib inline

import numpy as np
import pandas as pd
import datetime as dt
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
from sklearn import preprocessing

import seaborn as sns
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import metrics
from sklearn.ensemble import VotingClassifier
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

[2]:
```
df0=pd.read_csv('Heart_disease.csv')
df0.shape
```

[2]:
```
(4238, 16)
```

[3]:
```
df0.head()
```

[3]:
```
   male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0     1   39        4.0              0         0.0     0.0                0
1     0   46        2.0              0         0.0     0.0                0
2     1   48        1.0              1        20.0     0.0                0
3     0   61        3.0              1        30.0     0.0                0
4     0   46        3.0              1        23.0     0.0                0

   prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  \
0             0         0    195.0  106.0   70.0  26.97       80.0     77.0
1             0         0    250.0  121.0   81.0  28.73       95.0     76.0
2             0         0    245.0  127.5   80.0  25.34       75.0     70.0
3             1         0    225.0  150.0   95.0  28.58       65.0    103.0
4             0         0    285.0  130.0   84.0  23.10       85.0     85.0

   TenYearCHD
0           0
1           0
2           0
3           1
4           0
```

[4]:
```
df0.shape
```

[4]:
```
(4238, 16)
```

[5]:
```
df0.isnull().sum()
```

[5]:
```
male                 0
age                  0
education          105
currentSmoker        0
cigsPerDay          29
BPMeds              53
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol             50
sysBP                0
diaBP                0
BMI                 19
```

```
heartRate            1
glucose            388
TenYearCHD           0
dtype: int64
```

3 Data Cleaning

[6]:
```
df0['cigsPerDay']=df0['cigsPerDay'].fillna(df0['cigsPerDay'].mean())
df0['BPMeds']=df0['BPMeds'].fillna(0)
df0['totChol']=df0['totChol'].fillna(df0['totChol'].mean())
df0['BMI']=df0['BMI'].fillna(df0['BMI'].mean())
df0['heartRate']=df0['heartRate'].fillna(df0['heartRate'].mean())
```

[7]:
```
df0.isnull().sum()
```

[7]:
```
male                 0
age                  0
education          105
currentSmoker        0
cigsPerDay           0
BPMeds               0
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol              0
sysBP                0
diaBP                0
BMI                  0
heartRate            0
glucose            388
TenYearCHD           0
dtype: int64
```

[8]:
```
df0.dtypes
```

[8]:
```
male                 int64
age                  int64
education          float64
currentSmoker        int64
cigsPerDay         float64
BPMeds             float64
prevalentStroke      int64
prevalentHyp         int64
diabetes             int64
totChol            float64
```

```
sysBP               float64
diaBP               float64
BMI                 float64
heartRate           float64
glucose             float64
TenYearCHD            int64
dtype: object
```

[9]:
```
df0['male'] = df0['male'].astype('category').cat.codes
df0['currentSmoker'] = df0['currentSmoker'].astype('category').cat.codes
df0['BPMeds'] = df0['BPMeds'].astype('int').astype('category').cat.codes
df0['prevalentStroke'] = df0['prevalentStroke'].astype('category').cat.codes
df0['prevalentHyp'] = df0['prevalentHyp'].astype('category').cat.codes
df0['diabetes'] = df0['diabetes'].astype('category').cat.codes
```

3.1 Using KNN to impute the missing "education" column of data

[10]:
```
clu = df0.drop(['education','glucose','TenYearCHD'],axis=1)
```

[11]:
```
clu.dtypes
```

[11]:
```
male                   int8
age                   int64
currentSmoker          int8
cigsPerDay          float64
BPMeds                 int8
prevalentStroke        int8
prevalentHyp           int8
diabetes               int8
totChol             float64
sysBP               float64
diaBP               float64
BMI                 float64
heartRate           float64
dtype: object
```

[12]:
```
clu.isnull().sum()
```

[12]:
```
male                0
age                 0
currentSmoker       0
cigsPerDay          0
BPMeds              0
prevalentStroke     0
prevalentHyp        0
diabetes            0
```

```
totChol          0
sysBP            0
diaBP            0
BMI              0
heartRate        0
dtype: int64
```

[13]:
```python
import numpy as np
import pandas as pd
from collections import defaultdict
from scipy.stats import hmean
from scipy.spatial.distance import cdist
from scipy import stats
import numbers


def weighted_hamming(data):
    """ Compute weighted hamming distance on categorical variables. For one␣
↪variable, it is equal to 1 if
        the values between point A and point B are different, else it is equal␣
↪the relative frequency of the
        distribution of the value across the variable. For multiple variables,␣
↪the harmonic mean is computed
        up to a constant factor.
        @params:
            - data = a pandas data frame of categorical variables
        @returns:
            - distance_matrix = a distance matrix with pairwise distance for␣
↪all attributes
    """
    categories_dist = []

    for category in data:
        X = pd.get_dummies(data[category])
        X_mean = X * X.mean()
        X_dot = X_mean.dot(X.transpose())
        X_np = np.asarray(X_dot.replace(0,1,inplace=False))
        categories_dist.append(X_np)
    categories_dist = np.array(categories_dist)
    distances = hmean(categories_dist, axis=0)
    return distances


def distance_matrix(data, numeric_distance = "euclidean", categorical_distance␣
↪= "jaccard"):
    """ Compute the pairwise distance attribute by attribute in order to␣
↪account for different variables type:
```

```
        - Continuous
        - Categorical
        For ordinal values, provide a numerical representation taking the order␣
↪into account.
        Categorical variables are transformed into a set of binary ones.
        If both continuous and categorical distance are provided, a Gower-like␣
↪distance is computed and the numeric
        variables are all normalized in the process.
        If there are missing values, the mean is computed for numerical␣
↪attributes and the mode for categorical ones.

        Note: If weighted-hamming distance is chosen, the computation time␣
↪increases a lot since it is not coded in C
        like other distance metrics provided by scipy.
        @params:
            - data                  = pandas dataframe to compute distances on.
            - numeric_distances     = the metric to apply to continuous␣
↪attributes.
                                      "euclidean" and "cityblock" available.
                                      Default = "euclidean"
            - categorical_distances = the metric to apply to binary attributes.
                                      "jaccard", "hamming", "weighted-hamming"␣
↪and "euclidean"
                                      available. Default = "jaccard"
        @returns:
            - the distance matrix
    """
    possible_continuous_distances = ["euclidean", "cityblock"]
    possible_binary_distances = ["euclidean", "jaccard", "hamming",␣
↪"weighted-hamming"]
    number_of_variables = data.shape[1]
    number_of_observations = data.shape[0]

    # Get the type of each attribute (Numeric or categorical)
    is_numeric = [all(isinstance(n, numbers.Number) for n in data.iloc[:, i])␣
↪for i, x in enumerate(data)]
    is_all_numeric = sum(is_numeric) == len(is_numeric)
    is_all_categorical = sum(is_numeric) == 0
    is_mixed_type = not is_all_categorical and not is_all_numeric

    # Check the content of the distances parameter
    if numeric_distance not in possible_continuous_distances:
        print("The continuous distance " + numeric_distance + " is not␣
↪supported.")
        return None
    elif categorical_distance not in possible_binary_distances:
```

```
        print("The binary distance " + categorical_distance + " is not␣
↪supported.")
        return None

    # Separate the data frame into categorical and numeric attributes and␣
↪normalize numeric data
    if is_mixed_type:
        number_of_numeric_var = sum(is_numeric)
        number_of_categorical_var = number_of_variables - number_of_numeric_var
        data_numeric = data.iloc[:, is_numeric]
        data_numeric = (data_numeric - data_numeric.mean()) / (data_numeric.
↪max() - data_numeric.min())
        data_categorical = data.iloc[:, [not x for x in is_numeric]]

    # Replace missing values with column mean for numeric values and mode for␣
↪categorical ones. With the mode, it
    # triggers a warning: "SettingWithCopyWarning: A value is trying to be set␣
↪on a copy of a slice from a DataFrame"
    # but the value are properly replaced
    if is_mixed_type:
        data_numeric.fillna(data_numeric.mean(), inplace=True)
        for x in data_categorical:
            data_categorical[x].fillna(data_categorical[x].mode()[0],␣
↪inplace=True)
    elif is_all_numeric:
        data.fillna(data.mean(), inplace=True)
    else:
        for x in data:
            data[x].fillna(data[x].mode()[0], inplace=True)

    # "Dummifies" categorical variables in place
    if not is_all_numeric and not (categorical_distance == 'hamming' or␣
↪categorical_distance == 'weighted-hamming'):
        if is_mixed_type:
            data_categorical = pd.get_dummies(data_categorical)
        else:
            data = pd.get_dummies(data)
    elif not is_all_numeric and categorical_distance == 'hamming':
        if is_mixed_type:
            data_categorical = pd.DataFrame([pd.
↪factorize(data_categorical[x])[0] for x in data_categorical]).transpose()
        else:
            data = pd.DataFrame([pd.factorize(data[x])[0] for x in data]).
↪transpose()

    if is_all_numeric:
```

```
        result_matrix = cdist(data, data, metric=numeric_distance)
    elif is_all_categorical:
        if categorical_distance == "weighted-hamming":
            result_matrix = weighted_hamming(data)
        else:
            result_matrix = cdist(data, data, metric=categorical_distance)
    else:
        result_numeric = cdist(data_numeric, data_numeric,␣
↪metric=numeric_distance)
        if categorical_distance == "weighted-hamming":
            result_categorical = weighted_hamming(data_categorical)
        else:
            result_categorical = cdist(data_categorical, data_categorical,␣
↪metric=categorical_distance)
        result_matrix = np.array([[1.0*(result_numeric[i, j] *␣
↪number_of_numeric_var + result_categorical[i, j] *
                               number_of_categorical_var) / number_of_variables␣
↪for j in range(number_of_observations)] for i in␣
↪range(number_of_observations)])

    # Fill the diagonal with NaN values
    np.fill_diagonal(result_matrix, np.nan)

    return pd.DataFrame(result_matrix)


def knn_impute(target, attributes, k_neighbors, aggregation_method="mean",␣
↪numeric_distance="euclidean",
               categorical_distance="jaccard", missing_neighbors_threshold = 0.
↪5):
    """ Replace the missing values within the target variable based on its k␣
↪nearest neighbors identified with the
        attributes variables. If more than 50% of its neighbors are also␣
↪missing values, the value is not modified and
        remains missing. If there is a problem in the parameters provided,␣
↪returns None.
        If to many neighbors also have missing values, leave the missing value␣
↪of interest unchanged.
        @params:
            - target                        = a vector of n values with missing␣
↪values that you want to impute. The length has
                                              to be at least n = 3.
            - attributes                    = a data frame of attributes with n␣
↪rows to match the target variable
            - k_neighbors                   = the number of neighbors to look␣
↪at to impute the missing values. It has to be a
```

```python
                                                    value between 1 and n.
            - aggregation_method            = how to aggregate the values from␣
↪the nearest neighbors (mean, median, mode)
                                                    Default = "mean"
            - numeric_distances             = the metric to apply to continuous␣
↪attributes.
                                                    "euclidean" and "cityblock"␣
↪available.
                                                    Default = "euclidean"
            - categorical_distances         = the metric to apply to binary␣
↪attributes.
                                                    "jaccard", "hamming",␣
↪"weighted-hamming" and "euclidean"
                                                    available. Default = "jaccard"
            - missing_neighbors_threshold   = minimum of neighbors among the k␣
↪ones that are not also missing to infer
                                                    the correct value. Default = 0.5
        @returns:
            target_completed        = the vector of target values with missing␣
↪value replaced. If there is a problem
                                            in the parameters, return None
    """

    # Get useful variables
    possible_aggregation_method = ["mean", "median", "mode"]
    number_observations = len(target)
    is_target_numeric = all(isinstance(n, numbers.Number) for n in target)

    # Check for possible errors
    if number_observations < 3:
        print("Not enough observations.")
        return None
    if attributes.shape[0] != number_observations:
        print("The number of observations in the attributes variable is not␣
↪matching the target variable length.")
        return None
    if k_neighbors > number_observations or k_neighbors < 1:
        print("The range of the number of neighbors is incorrect.")
        return None
    if aggregation_method not in possible_aggregation_method:
        print("The aggregation method is incorrect.")
        return None
    if not is_target_numeric and aggregation_method != "mode":
        print("The only method allowed for categorical target variable is the␣
↪mode.")
        return None
```

```
    # Make sure the data are in the right format
    target = pd.DataFrame(target)
    attributes = pd.DataFrame(attributes)

    # Get the distance matrix and check whether no error was triggered when␣
↪computing it
    distances = distance_matrix(attributes, numeric_distance,␣
↪categorical_distance)
    if distances is None:
        return None

    # Get the closest points and compute the correct aggregation method
    for i, value in enumerate(target.iloc[:, 0]):
        if pd.isnull(value):
            order = distances.iloc[i,:].values.argsort()[:k_neighbors]
            closest_to_target = target.iloc[order, :]
            missing_neighbors = [x for x  in closest_to_target.isnull().iloc[:,␣
↪0]]
            # Compute the right aggregation method if at least more than 50% of␣
↪the closest neighbors are not missing
            if sum(missing_neighbors) >= missing_neighbors_threshold *␣
↪k_neighbors:
                continue
            elif aggregation_method == "mean":
                target.iloc[i] = np.ma.mean(np.ma.
↪masked_array(closest_to_target,np.isnan(closest_to_target)))
            elif aggregation_method == "median":
                target.iloc[i] = np.ma.median(np.ma.
↪masked_array(closest_to_target,np.isnan(closest_to_target)))
            else:
                target.iloc[i] = stats.mode(closest_to_target,␣
↪nan_policy='omit')[0][0]

    return target
```

[14]:
```
clus=clu.copy()
```

[15]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler = scaler.fit(clus)
clus = scaler.transform(clus)
```

[16]:
```
type(clus)
```

[16]: numpy.ndarray

[17]:
```
clus=pd.DataFrame(clus)
```

[18]:
```
clus.shape
```

[18]: (4238, 13)

[19]:
```
clus.columns =␣
 ↪['male','age','currentSmoker','cigsPerDay','BPMeds','prevalentStroke','prevalentHyp','diabe
```

[20]:
```
df0['education']=knn_impute(target=df0['education'], attributes=clus,
                            aggregation_method="median", k_neighbors=9,␣
 ↪numeric_distance='euclidean',
                            categorical_distance='hamming',␣
 ↪missing_neighbors_threshold=0.8)
```

[21]:
```
a = df0['education'].unique()
print(sorted(a))
```

```
[1.0, 2.0, 3.0, 4.0]
```

[22]:
```
# There is no more 1.5 and 2.5 in the 'education' column. I comment the␣
 ↪following codes out. Please ignore this part.
# df0[df0['education']==1.5]
```

[23]:
```
# df0['education'].loc[[943]]
```

[24]:
```
# df0[df0['education']==2.5]
```

[25]:
```
# print(df0['education'].loc[[306]], '\n', df0['education'].loc[[1604]], '\n',
#       df0['education'].loc[[2885]], '\n', df0['education'].loc[[4012]])
```

[26]:
```
# # df11 = df0.copy()
```

[27]:
```
# def education_adjust(df):
#     if df['education']==1.5 and (df['cigsPerDay']>=9.0 and␣
 ↪df['currentSmoker'] == 1):
#         return 1.0
#     elif df['education']==2.5 and (df['cigsPerDay']>=9.0 and␣
 ↪df['currentSmoker'] == 1):
#         return 2.0
#     elif df['education']==2.5 and (df['cigsPerDay']<9.0 or␣
 ↪df['currentSmoker'] == 0):
#         return 3.0
#     else:
#         return df['education']
```

```python
# df0['education'] = df0.apply (lambda df: education_adjust(df), axis=1)
```

[28]:
```python
# df0.head()
```

[29]:
```python
# b = df0['education'].unique()
# print(sorted(b))
```

[30]:
```python
# df0['education'].loc[[943]]
```

[31]:
```python
# print(df0['education'].loc[[306]], '\n', df0['education'].loc[[1604]], '\n',
#       df0['education'].loc[[2885]], '\n', df0['education'].loc[[4012]])
```

[32]:
```python
# df0
```

[33]:
```python
# df0['education'].loc[[36]]
df0.isnull().sum()
```

[33]:
```
male                 0
age                  0
education            0
currentSmoker        0
cigsPerDay           0
BPMeds               0
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol              0
sysBP                0
diaBP                0
BMI                  0
heartRate            0
glucose            388
TenYearCHD           0
dtype: int64
```

[34]:
```python
df0['education'] = df0['education'].astype(int)
df0.dtypes
```

[34]:
```
male                  int8
age                  int64
education            int64
currentSmoker         int8
cigsPerDay         float64
BPMeds                int8
prevalentStroke       int8
prevalentHyp          int8
diabetes              int8
```

```
totChol             float64
sysBP               float64
diaBP               float64
BMI                 float64
heartRate           float64
glucose             float64
TenYearCHD            int64
dtype: object
```

[35]:
```
df0.head()
```

[35]:
```
   male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0     1   39          4              0         0.0       0                0
1     0   46          2              0         0.0       0                0
2     1   48          1              1        20.0       0                0
3     0   61          3              1        30.0       0                0
4     0   46          3              1        23.0       0                0

   prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  \
0             0         0    195.0  106.0   70.0  26.97       80.0     77.0
1             0         0    250.0  121.0   81.0  28.73       95.0     76.0
2             0         0    245.0  127.5   80.0  25.34       75.0     70.0
3             1         0    225.0  150.0   95.0  28.58       65.0    103.0
4             0         0    285.0  130.0   84.0  23.10       85.0     85.0

   TenYearCHD
0           0
1           0
2           0
3           1
4           0
```

3.2 Using regression to impute the missing "glucose" column of data

[36]:
```
glu_test = df0[df0['glucose'].isnull()]
glu_test.shape
```

[36]: (388, 16)

[37]:
```
glu_test_y = glu_test[['TenYearCHD']]
glu_test_z = glu_test[['education']]
```

[38]:
```
glu = df0.dropna()
glu.shape
```

[38]: (3850, 16)

```
[39]: glu_x = glu.drop(['glucose','TenYearCHD','education'],axis=1)
      glu_y = glu[['glucose']]
```

3.2.1 Test the Multiple Regression to impute the missing "glucose" data

```
[40]: from sklearn.model_selection import train_test_split
      from sklearn.model_selection import cross_val_score
      # Set SEED for reproducibility
      SEED = 1

      # Split the data into 70% train and 30% test
      glu_xtrain, glu_xtest, glu_ytrain, glu_ytest = train_test_split(glu_x, glu_y,␣
       ↪test_size=0.075, random_state=SEED)

      from sklearn.linear_model import LinearRegression
      reg = LinearRegression().fit(glu_xtrain, glu_ytrain)
      reg.score(glu_xtrain, glu_ytrain)
```

```
[40]: 0.37686114497944034
```

```
[41]: glu_pred0 = reg.predict(glu_xtest)
      # glu_pred_proba0 = reg.predict_proba(glu_xtest)[:,1]
      # from sklearn.metrics import roc_auc_score
      # reg_roc_auc = roc_auc_score(glu_ytest, glu_pred_proba0)
```

```
[42]: glu_ytest['glucose'] = glu_ytest['glucose'].astype('float')
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

```
[43]: from sklearn.metrics import accuracy_score
      from sklearn.metrics import mean_squared_error as MSE
      rmse_glu0 = MSE(glu_ytest, glu_pred0)**(1/2)
      print("Linear Regression RMSE:", rmse_glu0)
```

```
Linear Regression RMSE: 15.107395590918358
```

```
[44]: # print(reg.summary())
```

3.2.2 Test XGBoost to impute the missing "glucose" data

[45]:
```
# import library
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
# direct xgboost library and possibly use "cv" from library
import xgboost as xgb
# sklearn wrapper for XGBoost. Allows Grid Search parellel processing like GBM
from xgboost.sklearn import XGBRegressor

# create a dictionary of parameters using range(start, stop but not including,␣
↪step)
param_grid = {'n_estimators': list(range(300, 500, 100)),
              'learning_rate':[i/10.0 for i in range(1,3)],
              'max_depth':[2,3],
              'gamma': [0.1,0.5,1,5]
              }
xgb0 = XGBRegressor(random_state = 1)

# create randomizedsearchCV object with various combinations of parameters
xgb0 = GridSearchCV(xgb0, param_grid, cv = 5,
                    refit = True,
                    n_jobs=-1, verbose = 5)

xgb0.fit(glu_xtrain, glu_ytrain)
xgb0.best_estimator_
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    6.7s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed:   16.8s finished
```

```
[16:30:46] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

[45]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.1,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=2, min_child_weight=1, missing=None, n_estimators=300,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=1,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

[46]:
```
glu_pred1 = xgb0.best_estimator_.predict(glu_xtest)
rmse_glu1 = MSE(glu_ytest, glu_pred1)**(1/2)
print("XGBoostRegressor RMSE:", rmse_glu1)
```

```
XGBoostRegressor RMSE: 17.77712432420023
```

3.2.3 Test GradientBoosting to impute the missing "glucose" data

[47]:
```
from sklearn.ensemble import GradientBoostingRegressor

# create a dictionary of parameters
param_grid = {'n_estimators':list(range(400, 700, 100)),
              'learning_rate':[i/10.0 for i in range(1,5)],
              'max_depth':[2,3]
              }
# create AdaBoostClassifier model
gbr0 = GradientBoostingRegressor(random_state=1)

# create gridsearch object with various combinations of parameters
gbr0 = GridSearchCV(gbr0, param_grid, cv = 5,
                    refit = True,
                    n_jobs=-1, verbose = 5)

gbr0.fit(glu_xtrain, glu_ytrain)
gbr0.best_estimator_
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:   12.3s finished
/usr/local/lib/python3.7/site-
packages/sklearn/ensemble/gradient_boosting.py:1450: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[47]:
```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.1, loss='ls', max_depth=2,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=400,
                          n_iter_no_change=None, presort='auto', random_state=1,
                          subsample=1.0, tol=0.0001, validation_fraction=0.1,
                          verbose=0, warm_start=False)
```

[48]:
```
glu_pred2 = gbr0.best_estimator_.predict(glu_xtest)
rmse_glu2 = MSE(glu_ytest, glu_pred2)**(1/2)
print("GradientBoostingRegressor RMSE:", rmse_glu2)
```

```
GradientBoostingRegressor RMSE: 18.360976197400383
```

3.2.4 Using multiple regression to predict the missing "glucose" data

```
[49]: glu_test = glu_test.drop(['education','glucose','TenYearCHD'],axis=1)
```

```
[50]: missing = reg.predict(glu_test)
```

```
[51]: pd.DataFrame(missing).head(10)
```

```
[51]:            0
      0  80.309112
      1  78.824030
      2  78.118658
      3  78.758119
      4  78.485850
      5  86.308363
      6  79.310843
      7  79.518931
      8  79.249279
      9  79.962604
```

```
[52]: glu_test['glucose']=missing
```

```
[53]: glu_test['education'] = glu_test_z
      glu_test['TenYearCHD'] = glu_test_y
      glu_test
```

```
[53]:       male  age  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
      14       0   39              1         9.0       0                0
      21       0   43              0         0.0       0                0
      26       0   60              0         0.0       0                0
      42       0   52              0         0.0       1                0
      54       0   39              1        20.0       0                0
      …     …   …              …          …       …                …
      4170     0   41              1         5.0       0                0
      4208     0   51              1         9.0       0                0
      4229     0   51              1        20.0       0                0
      4230     0   56              1         3.0       0                0
      4236     0   44              1        15.0       0                0

            prevalentHyp  diabetes     totChol  sysBP  diaBP    BMI  heartRate  \
      14               0         0  226.000000  114.0   64.0  22.35       85.0
      21               0         0  185.000000  123.5   77.5  29.89       70.0
      26               0         0  260.000000  110.0   72.5  26.59       65.0
      42               1         0  236.721585  148.0   92.0  25.09       70.0
```

```
54                0         0  209.000000  115.0   75.0  22.54       90.0
…              …       …          …      …     …     …           …
4170              0         0  205.000000  105.0   74.0  20.85       87.0
4208              0         0  340.000000  152.0   76.0  25.74       70.0
4229              1         0  251.000000  140.0   80.0  25.60       75.0
4230              1         0  268.000000  170.0  102.0  22.89       57.0
4236              0         0  210.000000  126.5   87.0  19.16       86.0

        glucose  education  TenYearCHD
14    80.309112          2           0
21    78.824030          1           0
26    78.118658          1           0
42    78.758119          1           1
54    78.485850          2           0
…           …         …          …
4170  78.076656          2           0
4208  83.152034          1           0
4229  79.142985          3           0
4230  79.148916          1           0
4236  78.185804          1           0

[388 rows x 16 columns]
```

[54]:
```
glu_test=glu_test.iloc[:, [*list(range(0,2)),-2,*list(range(2,14)),15]]
```

[55]:
```
glu_test.head()
```

[55]:
```
    male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
14     0   39          2              1         9.0       0                0
21     0   43          1              0         0.0       0                0
26     0   60          1              0         0.0       0                0
42     0   52          1              0         0.0       1                0
54     0   39          2              1        20.0       0                0

    prevalentHyp  diabetes     totChol  sysBP  diaBP    BMI  heartRate  \
14             0         0  226.000000  114.0   64.0  22.35       85.0
21             0         0  185.000000  123.5   77.5  29.89       70.0
26             0         0  260.000000  110.0   72.5  26.59       65.0
42             1         0  236.721585  148.0   92.0  25.09       70.0
54             0         0  209.000000  115.0   75.0  22.54       90.0

      glucose  TenYearCHD
14  80.309112           0
21  78.824030           0
26  78.118658           0
42  78.758119           1
54  78.485850           0
```

[56]:
```
glu.shape
```

[56]:
```
(3850, 16)
```

[57]:
```
glu.head()
```

[57]:
```
   male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0     1   39          4              0         0.0       0                0
1     0   46          2              0         0.0       0                0
2     1   48          1              1        20.0       0                0
3     0   61          3              1        30.0       0                0
4     0   46          3              1        23.0       0                0

   prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  \
0             0         0    195.0  106.0   70.0  26.97       80.0     77.0
1             0         0    250.0  121.0   81.0  28.73       95.0     76.0
2             0         0    245.0  127.5   80.0  25.34       75.0     70.0
3             1         0    225.0  150.0   95.0  28.58       65.0    103.0
4             0         0    285.0  130.0   84.0  23.10       85.0     85.0

   TenYearCHD
0           0
1           0
2           0
3           1
4           0
```

[313]:
```
df = pd.concat([glu, glu_test], ignore_index=False)
```

[314]:
```
df.iloc[3845:3855,:]
```

[314]:
```
      male  age  education  currentSmoker  cigsPerDay  BPMeds  \
4232     1   68          1              0         0.0       0
4233     1   50          1              1         1.0       0
4234     1   51          3              1        43.0       0
4235     0   48          2              1        20.0       0
4237     0   52          2              0         0.0       0
14       0   39          2              1         9.0       0
21       0   43          1              0         0.0       0
26       0   60          1              0         0.0       0
42       0   52          1              0         0.0       1
54       0   39          2              1        20.0       0

      prevalentStroke  prevalentHyp  diabetes     totChol  sysBP  diaBP  \
4232                0             1         0  176.000000  168.0   97.0
4233                0             1         0  313.000000  179.0   92.0
4234                0             0         0  207.000000  126.5   80.0
```

```
4235                 0               0           0  248.000000  131.0   72.0
4237                 0               0           0  269.000000  133.5   83.0
14                   0               0           0  226.000000  114.0   64.0
21                   0               0           0  185.000000  123.5   77.5
26                   0               0           0  260.000000  110.0   72.5
42                   0               1           0  236.721585  148.0   92.0
54                   0               0           0  209.000000  115.0   75.0

        BMI  heartRate     glucose  TenYearCHD
4232  23.14       60.0   79.000000           1
4233  25.97       66.0   86.000000           1
4234  19.71       65.0   68.000000           0
4235  22.00       84.0   86.000000           0
4237  21.47       80.0  107.000000           0
14    22.35       85.0   80.309112           0
21    29.89       70.0   78.824030           0
26    26.59       65.0   78.118658           0
42    25.09       70.0   78.758119           1
54    22.54       90.0   78.485850           0
```

[315]:

```python
df = df.rename_axis('MyIdx').sort_values(by = ['MyIdx'], ascending = [True])
df.iloc[12:28,:]
```

[315]:

```
       male  age  education  currentSmoker  cigsPerDay  BPMeds  \
MyIdx
12        1   46          1              1        15.0       0
13        0   41          3              0         0.0       1
14        0   39          2              1         9.0       0
15        0   38          2              1        20.0       0
16        1   48          3              1        10.0       0
17        0   46          2              1        20.0       0
18        0   38          2              1         5.0       0
19        1   41          2              0         0.0       0
20        0   42          2              1        30.0       0
21        0   43          1              0         0.0       0
22        0   52          1              0         0.0       0
23        0   52          3              1        20.0       0
24        1   44          2              1        30.0       0
25        1   47          4              1        20.0       0
26        0   60          1              0         0.0       0
27        1   35          2              1        20.0       0

       prevalentStroke  prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  \
MyIdx
12                   0             1         0    294.0  142.0   94.0  26.31
13                   0             1         0    332.0  124.0   88.0  31.31
14                   0             0         0    226.0  114.0   64.0  22.35
```

```
15                    0            1         0     221.0  140.0    90.0  21.35
16                    0            1         0     232.0  138.0    90.0  22.37
17                    0            0         0     291.0  112.0    78.0  23.38
18                    0            0         0     195.0  122.0    84.5  23.24
19                    0            0         0     195.0  139.0    88.0  26.88
20                    0            0         0     190.0  108.0    70.5  21.59
21                    0            0         0     185.0  123.5    77.5  29.89
22                    0            0         0     234.0  148.0    78.0  34.17
23                    0            0         0     215.0  132.0    82.0  25.11
24                    0            1         0     270.0  137.5    90.0  21.96
25                    0            0         0     294.0  102.0    68.0  24.18
26                    0            0         0     260.0  110.0    72.5  26.59
27                    0            1         0     225.0  132.0    91.0  26.09

       heartRate     glucose  TenYearCHD
MyIdx
12          98.0   64.000000           0
13          65.0   84.000000           0
14          85.0   80.309112           0
15          95.0   70.000000           1
16          64.0   72.000000           0
17          80.0   89.000000           1
18          75.0   78.000000           0
19          85.0   65.000000           0
20          72.0   85.000000           0
21          70.0   78.824030           0
22          70.0  113.000000           0
23          71.0   75.000000           0
24          75.0   83.000000           0
25          62.0   66.000000           1
26          65.0   78.118658           0
27          73.0   83.000000           0
```

[316]:
```
# df.iloc[3845:3855,:]
```

[317]:
```
df.isnull().sum()
```

[317]:
```
male               0
age                0
education          0
currentSmoker      0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol            0
```

```
sysBP               0
diaBP               0
BMI                 0
heartRate           0
glucose             0
TenYearCHD          0
dtype: int64
```

[318]:
```
df.shape
```

[318]:
```
(4238, 16)
```

[319]:
```
df.dtypes
```

[319]:
```
male                   int8
age                   int64
education             int64
currentSmoker          int8
cigsPerDay          float64
BPMeds                 int8
prevalentStroke        int8
prevalentHyp           int8
diabetes               int8
totChol             float64
sysBP               float64
diaBP               float64
BMI                 float64
heartRate           float64
glucose             float64
TenYearCHD            int64
dtype: object
```

[320]:
```
df = df.rename_axis('')
df
```

[320]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds \
0	1	39	4	0	0.0	0
1	0	46	2	0	0.0	0
2	1	48	1	1	20.0	0
3	0	61	3	1	30.0	0
4	0	46	3	1	23.0	0
…	…	…	…	…	…	…
4233	1	50	1	1	1.0	0
4234	1	51	3	1	43.0	0
4235	0	48	2	1	20.0	0
4236	0	44	1	1	15.0	0

```
4237     0   52          2              0         0.0       0

      prevalentStroke  prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  \
0                   0             0         0    195.0  106.0   70.0  26.97
1                   0             0         0    250.0  121.0   81.0  28.73
2                   0             0         0    245.0  127.5   80.0  25.34
3                   0             1         0    225.0  150.0   95.0  28.58
4                   0             0         0    285.0  130.0   84.0  23.10
…                 …             …         …        …      …      …      …
4233                0             1         0    313.0  179.0   92.0  25.97
4234                0             0         0    207.0  126.5   80.0  19.71
4235                0             0         0    248.0  131.0   72.0  22.00
4236                0             0         0    210.0  126.5   87.0  19.16
4237                0             0         0    269.0  133.5   83.0  21.47

      heartRate     glucose  TenYearCHD
0          80.0   77.000000           0
1          95.0   76.000000           0
2          75.0   70.000000           0
3          65.0  103.000000           1
4          85.0   85.000000           0
…           …          …           …
4233       66.0   86.000000           1
4234       65.0   68.000000           0
4235       84.0   86.000000           0
4236       86.0   78.185804           0
4237       80.0  107.000000           0

[4238 rows x 16 columns]
```

[321]:
```
dff = df.copy()
```

[322]:
```
dff.head()
```

[322]:
```
   male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0     1   39          4              0         0.0       0                0
1     0   46          2              0         0.0       0                0
2     1   48          1              1        20.0       0                0
3     0   61          3              1        30.0       0                0
4     0   46          3              1        23.0       0                0

   prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  \
0             0         0    195.0  106.0   70.0  26.97       80.0     77.0
```

```
1               0         0    250.0   121.0    81.0  28.73         95.0      76.0
2               0         0    245.0   127.5    80.0  25.34         75.0      70.0
3               1         0    225.0   150.0    95.0  28.58         65.0     103.0
4               0         0    285.0   130.0    84.0  23.10         85.0      85.0

   TenYearCHD
0           0
1           0
2           0
3           1
4           0
```

[ ]:

```

```

↑↑↑↑↑ **Data Cleaning Ends Here** ↑↑↑↑↑

---

4 Exploratory Data Analysis

[325]:

```python
df.head(2)
```

[325]:

```
   male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0     1   39          4              0         0.0       0                0
1     0   46          2              0         0.0       0                0

   prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  \
0             0         0    195.0  106.0   70.0  26.97       80.0     77.0
1             0         0    250.0  121.0   81.0  28.73       95.0     76.0

   TenYearCHD
0           0
1           0
```

[326]:

```python
from tableone import TableOne
columns = ['age','education','male','BPMeds','diabetes']
categorical = ['male','education','BPMeds','diabetes']
groupby = 'TenYearCHD'
mytable = TableOne(df, columns, categorical, groupby, pval=True)
print(mytable)
```

```
                Grouped by TenYearCHD
                               isnull          0           1    pval
ptest
variable  level
n                                           3594        644
age                              0   48.8 (8.4)  54.1 (8.0)  <0.001  Two
Sample T-test
education 1                      0  1437 (40.0)  330 (51.2)  <0.001
Chi-squared
          2                         1147 (31.9)  155 (24.1)
          3                          607 (16.9)   88 (13.7)
          4                          403 (11.2)   71 (11.0)
male      0                      0  2118 (58.9)  301 (46.7)  <0.001
Chi-squared
          1                         1476 (41.1)  343 (53.3)
BPMeds    0                      0  3511 (97.7)  603 (93.6)  <0.001
Chi-squared
          1                            83 (2.3)    41 (6.4)
diabetes  0                      0  3525 (98.1)  604 (93.8)  <0.001
Chi-squared
          1                            69 (1.9)    40 (6.2)
[1] Warning, Hartigan's Dip Test reports possible multimodal distributions for:
age.
[2] Warning, test for normality reports non-normal distributions for: age.
```

[327]:

```python
#Imports
%matplotlib inline
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import pylab as pl
import scipy as sc
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import linear_model, metrics
# from sklearn import cross_validation
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
```

[328]:

```python
pd.DataFrame(dff['glucose'].describe())
```

[328]:

```
           glucose
count  4238.000000
mean     81.829728
```

```
std       23.008855
min       40.000000
25%       72.000000
50%       78.000000
75%       85.987577
max      394.000000
```

[329]:
```
pd.DataFrame(dff['heartRate'].describe())
```

[329]:

	heartRate
count	4238.000000
mean	75.878924
std	12.025177
min	44.000000
25%	68.000000
50%	75.000000
75%	83.000000
max	143.000000

[330]:
```
pd.DataFrame(dff['BMI'].describe())
```

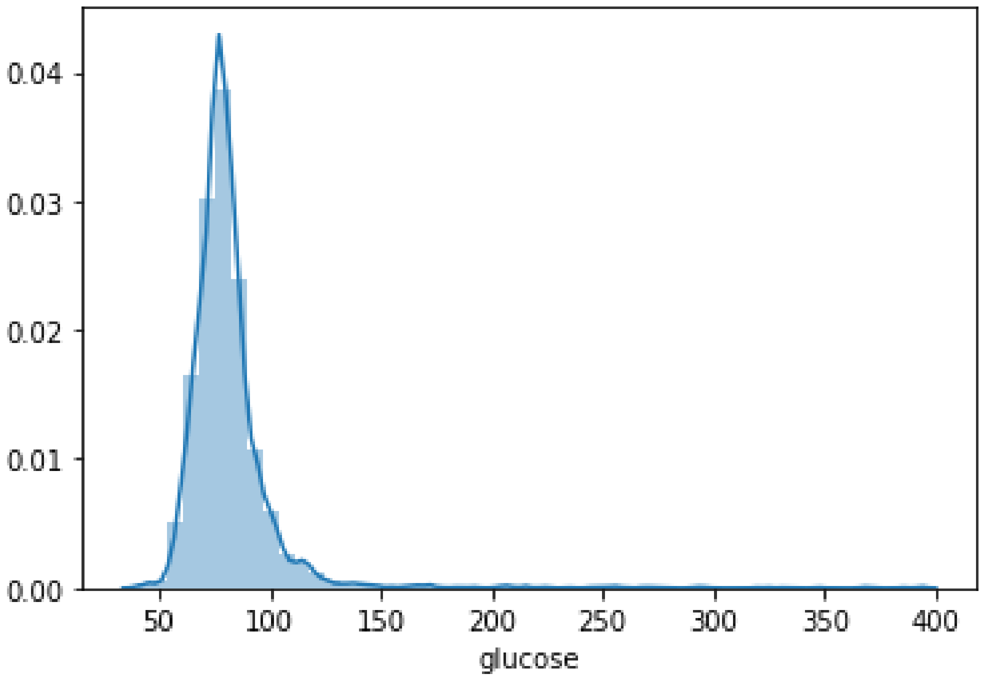
[330]:

	BMI
count	4238.000000
mean	25.802008
std	4.070953
min	15.540000
25%	23.080000
50%	25.410000
75%	28.037500
max	56.800000

[331]:
```
print(dff['glucose'].skew())
print(dff['glucose'].kurt())
sns.distplot(dff['glucose'])
```

```
6.437460632034153
63.19691484140628
```

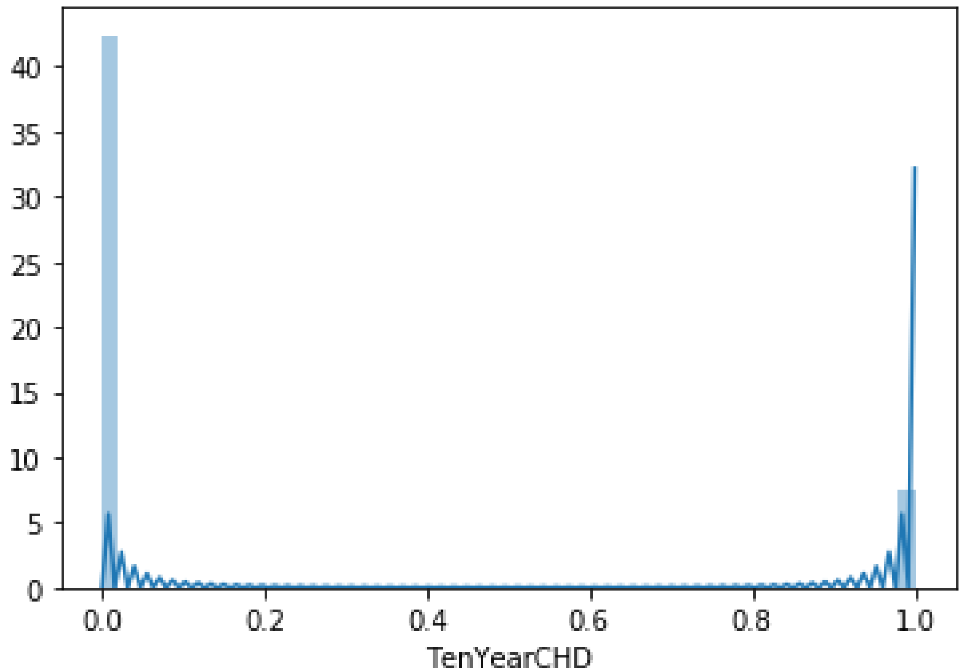
[331]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14d74a490>
```



[332]:
```
print(dff['TenYearCHD'].skew())
print(dff['TenYearCHD'].kurt())
sns.distplot(dff['TenYearCHD'])
```

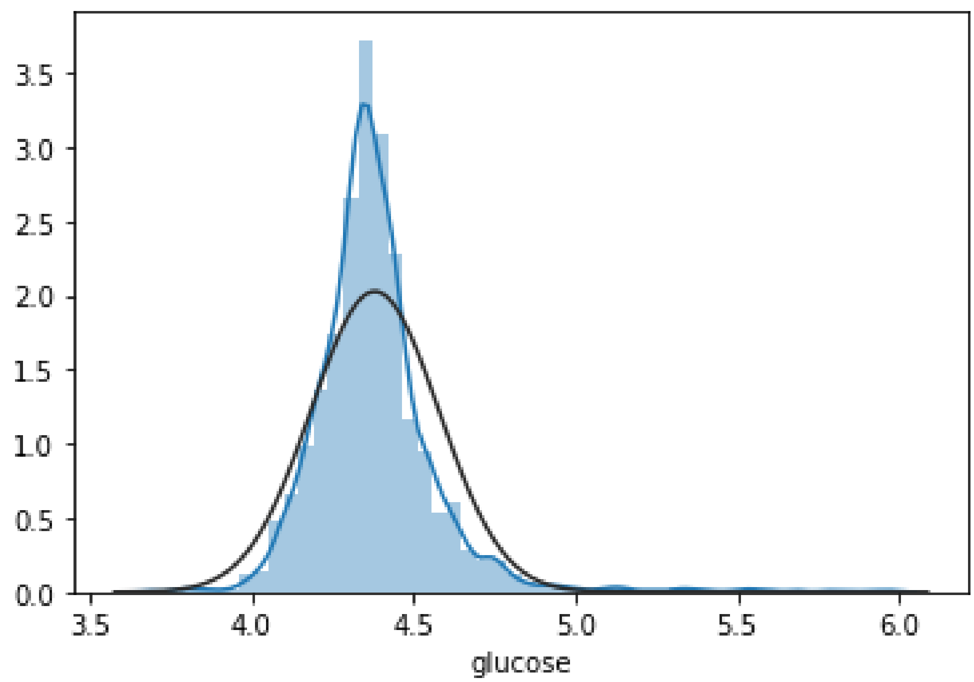
```
1.9397412552855473
1.763428113047742
```

[332]: <matplotlib.axes._subplots.AxesSubplot at 0x14d6d6c10>



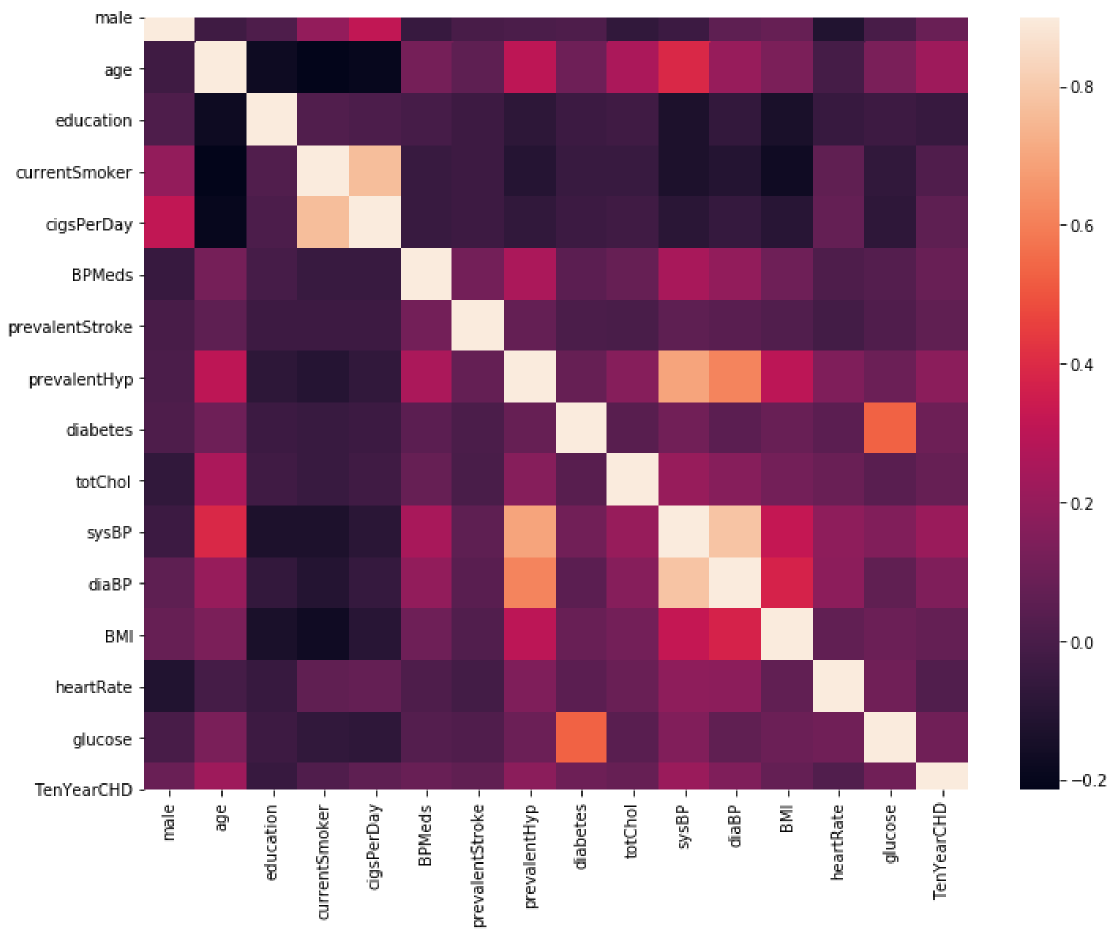
[333]:
```
from scipy.stats import norm
#setting transformed dependent variable with a new name
dff['glucose'] = np.log(dff['glucose'])
sns.distplot(dff['glucose'], fit=norm)
```

[333]: <matplotlib.axes._subplots.AxesSubplot at 0x14df988d0>



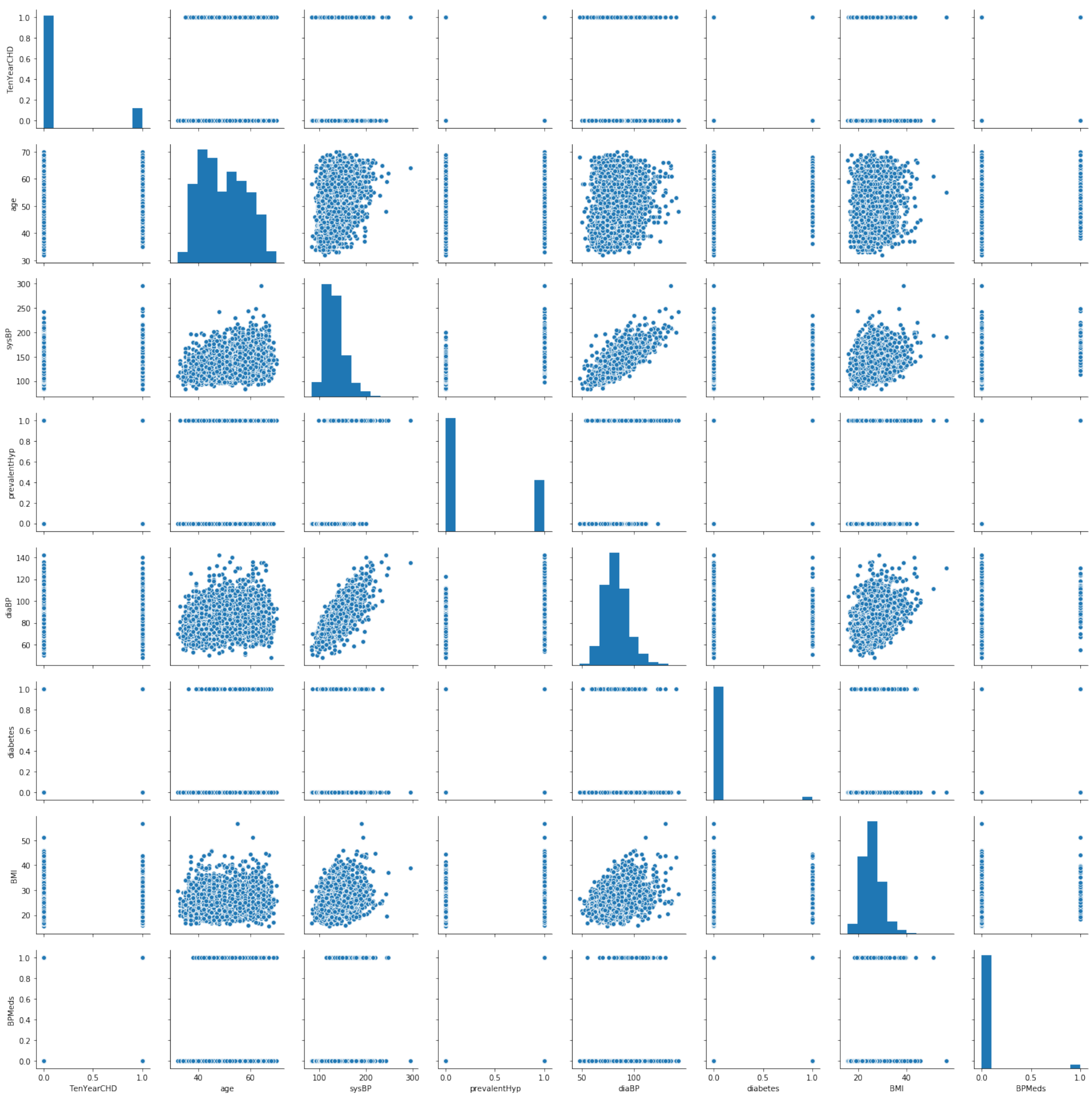
[348]:
```
corrmat = dff.corr()
plt.subplots(figsize=(12,9))
sns.heatmap(corrmat, vmax=0.9, square=True)
```

[348]: <matplotlib.axes._subplots.AxesSubplot at 0x14db0a890>

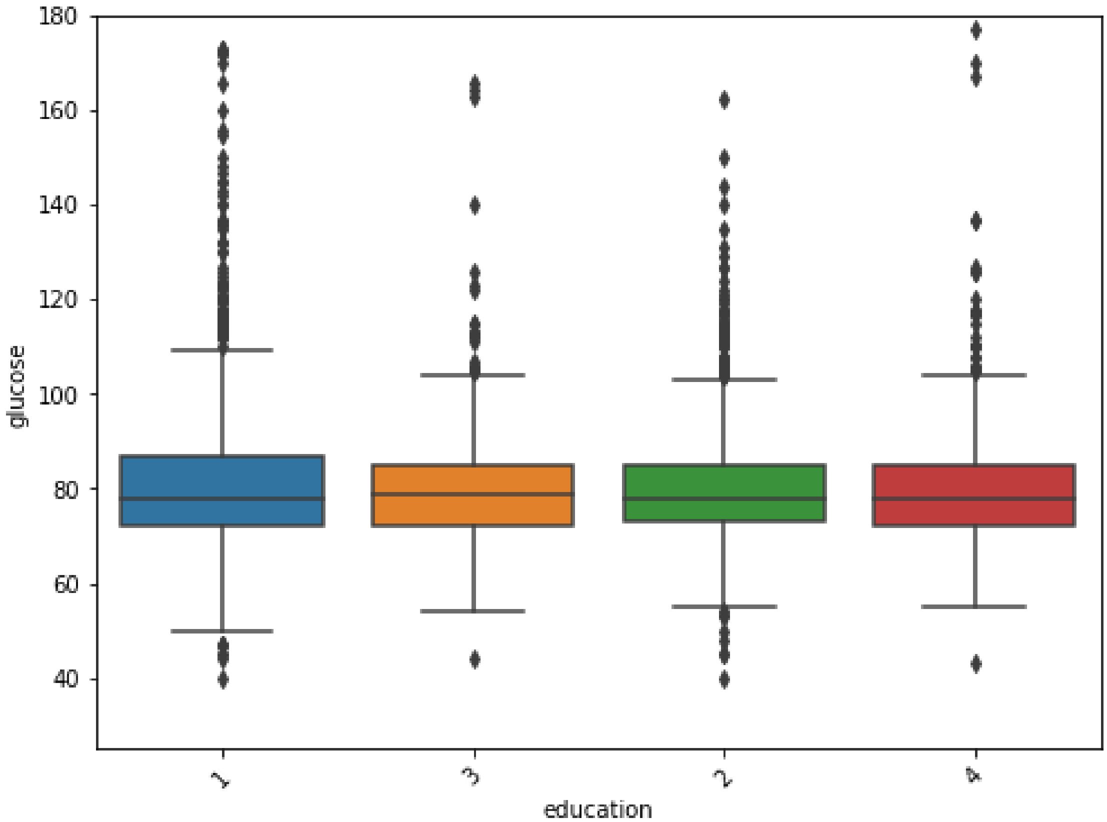


[335]:
```
#Plots scatterplots and histograms for the most highly correlated variables to␣
↪TenYearCHD
cols = ['TenYearCHD', 'age', 'sysBP', 'prevalentHyp', 'diaBP', 'diabetes',␣
↪'BMI', 'BPMeds']
sns.pairplot(dff[cols], size = 2.5)
plt.show()
```

```
/usr/local/lib/python3.7/site-packages/seaborn/axisgrid.py:2065: UserWarning:
The `size` parameter has been renamed to `height`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```



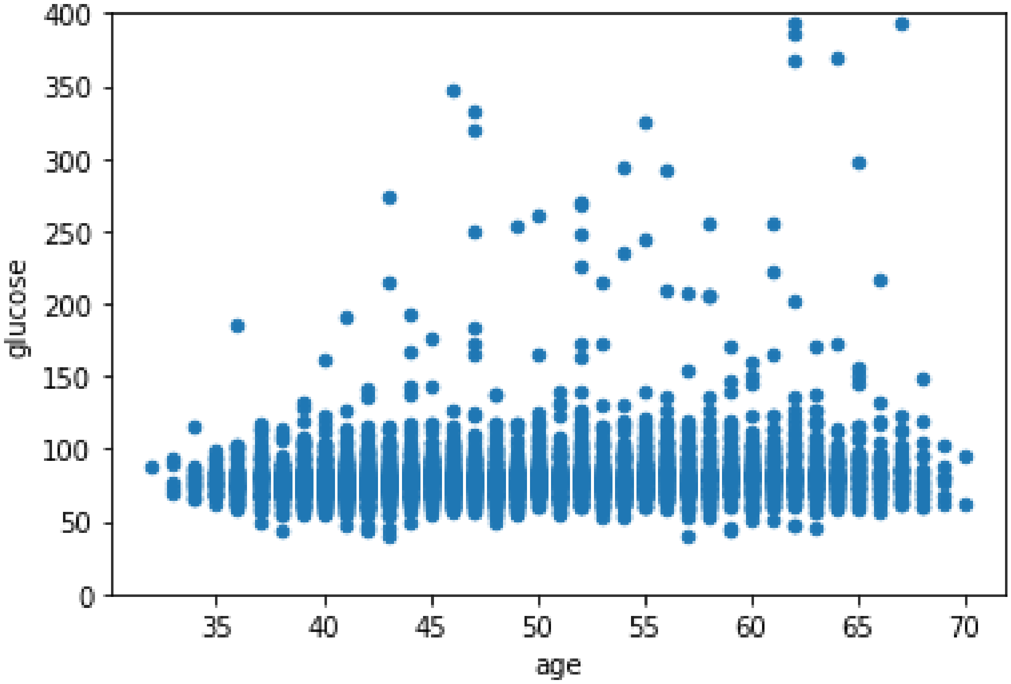
[336]:
```
#box plot of Neighborhood, sorted by glucose
var = 'education'
data = pd.concat([np.exp(dff['glucose']), dff[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
ranks = dff.groupby(var)['glucose'].mean().fillna(0).sort_values()[::-1].index
fig = sns.boxplot(x=var, y="glucose", data=data, order=ranks)
plt.xticks(rotation=45)
fig.axis(ymin=25, ymax=180);
```



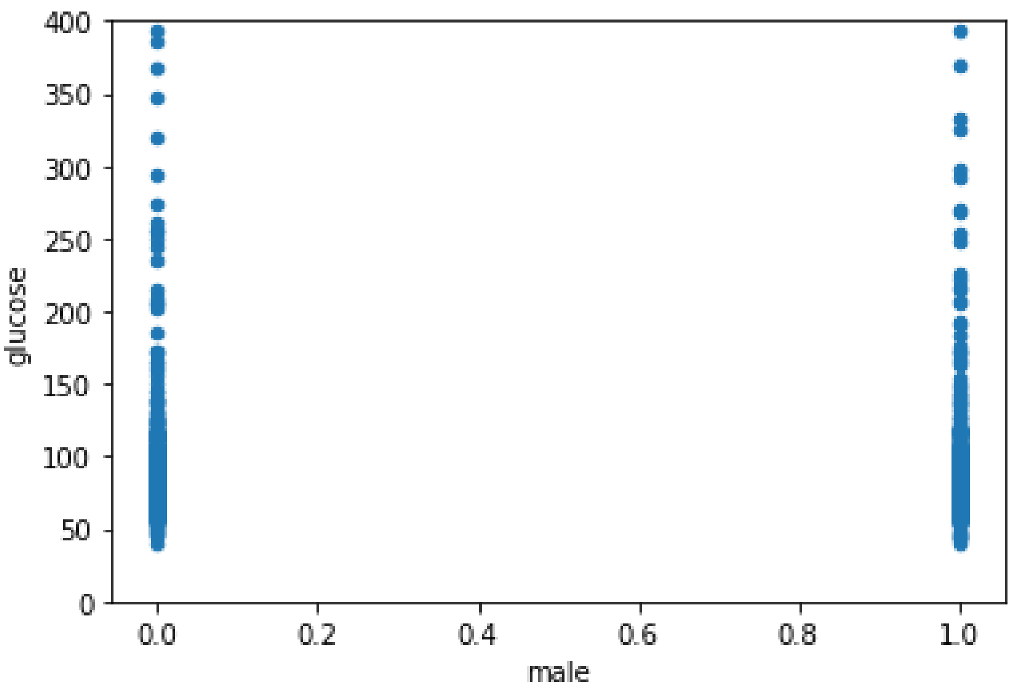
[78]:
```
a = dff['education'].unique()
print(sorted(a))
```

```
[1, 2, 3, 4]
```

[79]:
```
var = 'age'
data = pd.concat([np.exp(dff['glucose']), dff[var]], axis=1)
data.plot.scatter(x=var, y='glucose', ylim=(0,400));
```



[349]:
```
var = 'male'
data = pd.concat([np.exp(dff['glucose']), dff[var]], axis=1)
data.plot.scatter(x=var, y='glucose', ylim=(0,400));
```



[80]:
```
dummy_education  = pd.get_dummies(dff['education'])
dummy_education.head()
```

[80]:

	1	2	3	4
0	0	0	0	1
1	0	1	0	0
2	1	0	0	0
3	0	0	1	0
4	0	0	1	0

[81]:
```
#Join dummify variables with self selected features to run in linear regression
trainGL = dff.join(dummy_education.loc[:,:])
trainGL['intercept'] = 1.0

#matrix of self selected covariates to use for linear regression
trainGL_dff = trainGL[trainGL.columns[1:]]
trainGL_dff = trainGL_dff.drop(['education','TenYearCHD'],axis=1)
trainGL_dff.head()
```

[81]:

	age	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	\
0	39	0	0.0	0	0	0	
1	46	0	0.0	0	0	0	
2	48	1	20.0	0	0	0	
3	61	1	30.0	0	0	1	
4	46	1	23.0	0	0	0	

	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	1	2	3	4	\
0	0	195.0	106.0	70.0	26.97	80.0	4.343805	0	0	0	1	
1	0	250.0	121.0	81.0	28.73	95.0	4.330733	0	1	0	0	
2	0	245.0	127.5	80.0	25.34	75.0	4.248495	1	0	0	0	
3	0	225.0	150.0	95.0	28.58	65.0	4.634729	0	0	1	0	
4	0	285.0	130.0	84.0	23.10	85.0	4.442651	0	0	1	0	

	intercept
0	1.0
1	1.0
2	1.0
3	1.0
4	1.0

[82]:
```
glu_xtrain, glu_xtest, glu_ytrain, glu_ytest = train_test_split(glu_x, glu_y,␣
↪test_size=0.075, random_state=SEED)
```

[83]:
```
#Changing into a categorical variable
trainGL_dff['currentSmoker'] = trainGL_dff['currentSmoker'].apply(str)
trainGL_dff['BPMeds'] = trainGL_dff['BPMeds'].astype(str)
trainGL_dff['prevalentStroke'] = trainGL_dff['prevalentStroke'].astype(str)
trainGL_dff['prevalentHyp'] = trainGL_dff['prevalentHyp'].astype(str)
trainGL_dff['diabetes'] = trainGL_dff['diabetes'].astype(str)
```

[84]:
```
y = trainGL_dff['glucose']
X = trainGL_dff.drop(['glucose'],axis=1)
```

[85]:
```
print(sm.OLS(y, X.astype(float)).fit().summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                glucose   R-squared:                       0.303
Model:                            OLS   Adj. R-squared:                  0.301
Method:                 Least Squares   F-statistic:                     122.5
Date:                Sun, 01 Dec 2019   Prob (F-statistic):          5.15e-317
Time:                        16:31:14   Log-Likelihood:                 1640.0
No. Observations:                4238   AIC:                            -3248.
Df Residuals:                    4222   BIC:                            -3146.
Df Model:                          15
Covariance Type:            nonrobust
================================================================================
===
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---
age                 0.0010      0.000      2.906      0.004       0.000
0.002
currentSmoker       0.0057      0.008      0.706      0.480      -0.010
0.021
cigsPerDay         -0.0009      0.000     -2.792      0.005      -0.002
-0.000
BPMeds             -0.0201      0.016     -1.278      0.201      -0.051
0.011
prevalentStroke     0.0271      0.033      0.813      0.416      -0.038
0.092
prevalentHyp       -0.0075      0.008     -0.952      0.341      -0.023
0.008
diabetes            0.6351      0.016     39.247      0.000       0.603
0.667
totChol          -2.95e-05   5.99e-05     -0.492      0.622      -0.000
```

```
8.8e-05
sysBP               0.0012      0.000      5.526      0.000       0.001
0.002
diaBP              -0.0015      0.000     -4.097      0.000      -0.002
-0.001
BMI                 0.0016      0.001      2.381      0.017       0.000
0.003
heartRate           0.0013      0.000      5.921      0.000       0.001
0.002
1                   0.8255      0.008     99.150      0.000       0.809
0.842
2                   0.8309      0.008    106.462      0.000       0.816
0.846
3                   0.8353      0.008     99.047      0.000       0.819
0.852
4                   0.8307      0.009     93.170      0.000       0.813
0.848
intercept           3.3224      0.027    124.117      0.000       3.270
3.375
==============================================================================
Omnibus:                      434.303   Durbin-Watson:                   2.034
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3247.658
Skew:                           0.160   Prob(JB):                         0.00
Kurtosis:                       7.277   Cond. No.                     2.58e+18
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 5.78e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

[ ]:

```

```

5 PCA and t-SNE

[86]:

```
x = df.values
class0=df[['TenYearCHD']]#.values[:,0]
```

[87]:

```
class0=class0['TenYearCHD'].astype('category').cat.codes
```

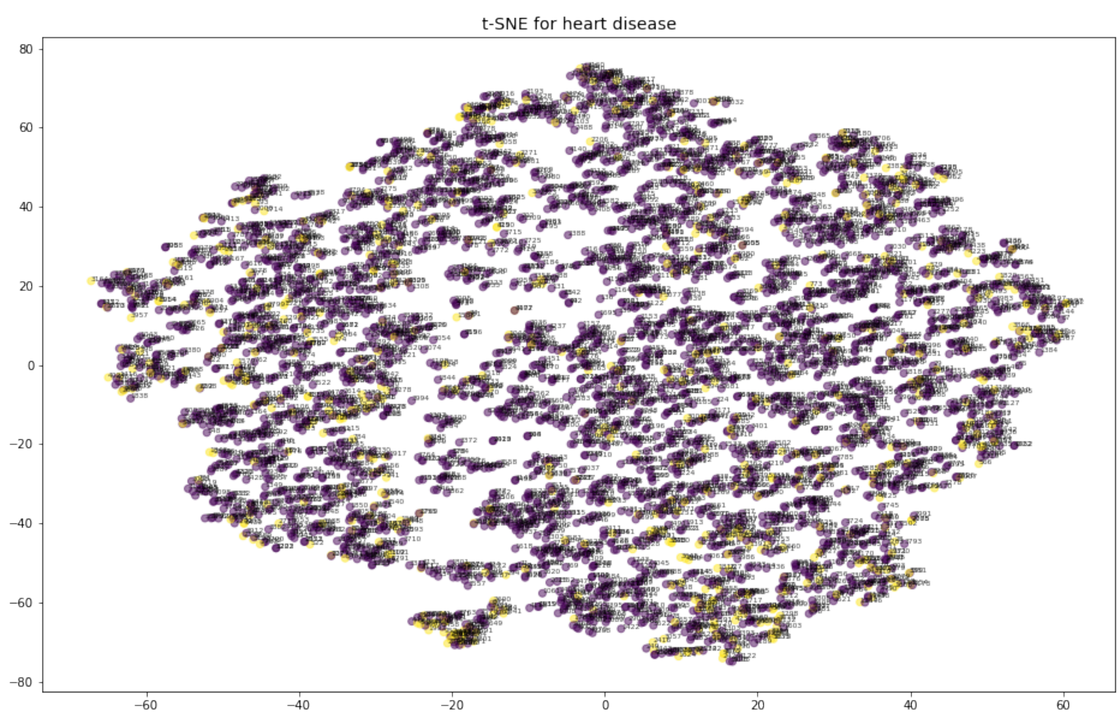
[88]:

```
index0=np.arange(0, 4238).tolist()
```

[89]:

```
from sklearn.preprocessing import normalize
```

```python
normalized_df = normalize(x)

from sklearn.manifold import TSNE

plt.figure(figsize=(16,10))


# Create a TSNE instance: model
tsne = TSNE(learning_rate=150)

# Apply fit_transform to normalized_movements: tsne_features
tsne_features = tsne.fit_transform(normalized_df)

# Select the 0th feature: xs
xs = tsne_features[:,0]

# Select the 1th feature: ys
ys = tsne_features[:,1]

# Scatter plot
plt.scatter(xs, ys, alpha=0.5,c=class0)
# Annotate the points
for x0, y0, patient in zip(xs, ys, index0):
    plt.annotate(patient, (x0, y0), fontsize=6, alpha=0.75)
#plt.figure(figsize=(18,20))
plt.title('t-SNE for heart disease',fontsize=14)
plt.show()
```



[90]:
```
df.head()
```

[90]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	\
0	1	39	4	0	0.0	0	0	
1	0	46	2	0	0.0	0	0	
2	1	48	1	1	20.0	0	0	
3	0	61	3	1	30.0	0	0	
4	0	46	3	1	23.0	0	0	

	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	\
0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	
1	0	0	250.0	121.0	81.0	28.73	95.0	76.0	
2	0	0	245.0	127.5	80.0	25.34	75.0	70.0	
3	1	0	225.0	150.0	95.0	28.58	65.0	103.0	
4	0	0	285.0	130.0	84.0	23.10	85.0	85.0	

	TenYearCHD
0	0
1	0
2	0
3	1
4	0

[91]:
```
df_X=df.iloc[:,0:15]
df_X.head()
```

[91]:

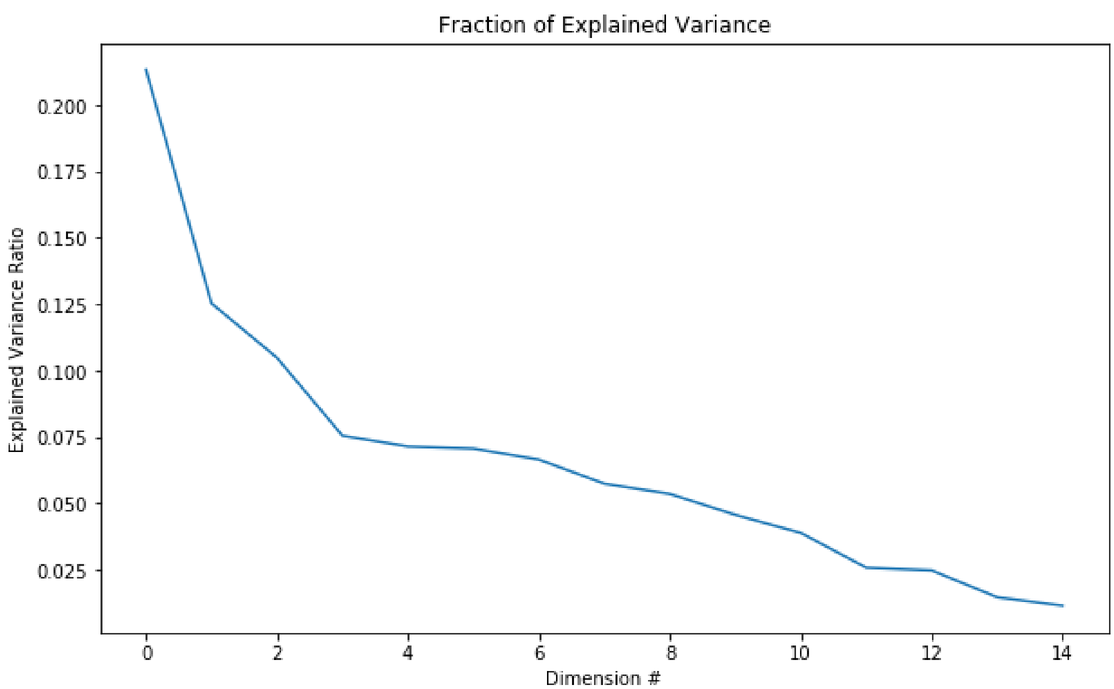
	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	\
0	1	39	4	0	0.0	0	0	
1	0	46	2	0	0.0	0	0	
2	1	48	1	1	20.0	0	0	
3	0	61	3	1	30.0	0	0	
4	0	46	3	1	23.0	0	0	

	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose
0	0	0	195.0	106.0	70.0	26.97	80.0	77.0
1	0	0	250.0	121.0	81.0	28.73	95.0	76.0
2	0	0	245.0	127.5	80.0	25.34	75.0	70.0
3	1	0	225.0	150.0	95.0	28.58	65.0	103.0
4	0	0	285.0	130.0	84.0	23.10	85.0	85.0

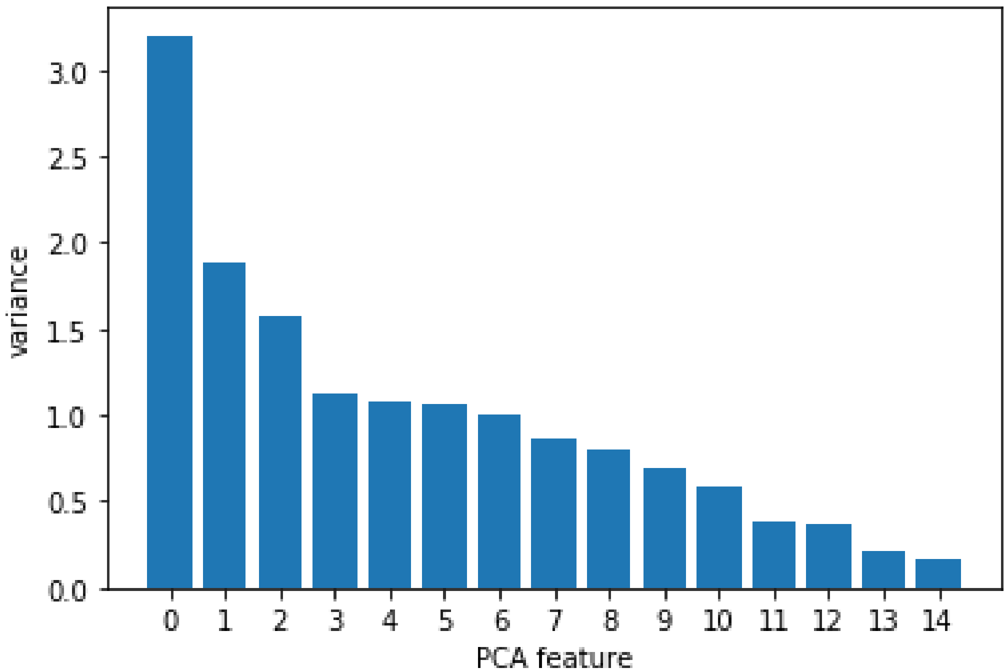
```
[92]: def standardize(df):
          stscaler = StandardScaler().fit(df)
          scaled = stscaler.transform(df)
          return scaled
      df_Xs = standardize(df_X)
```

```
[93]: def fit_pca(df, n_components):
          pca = PCA(n_components)
          pca.fit(df)
          return pca
      pca = fit_pca(df_Xs, n_components=15)
```

```
[94]: def plot_scaled_variance(pca):
          fig, ax = plt.subplots(figsize=(10,6))

          ax.set_xlabel('Dimension #')
          ax.set_ylabel('Explained Variance Ratio')
          ax.set_title('Fraction of Explained Variance')
          ax.plot(pca.explained_variance_ratio_)

          return ax

      ax = plot_scaled_variance(pca)
```



[95]:
```
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.xticks(features)
plt.show()
```



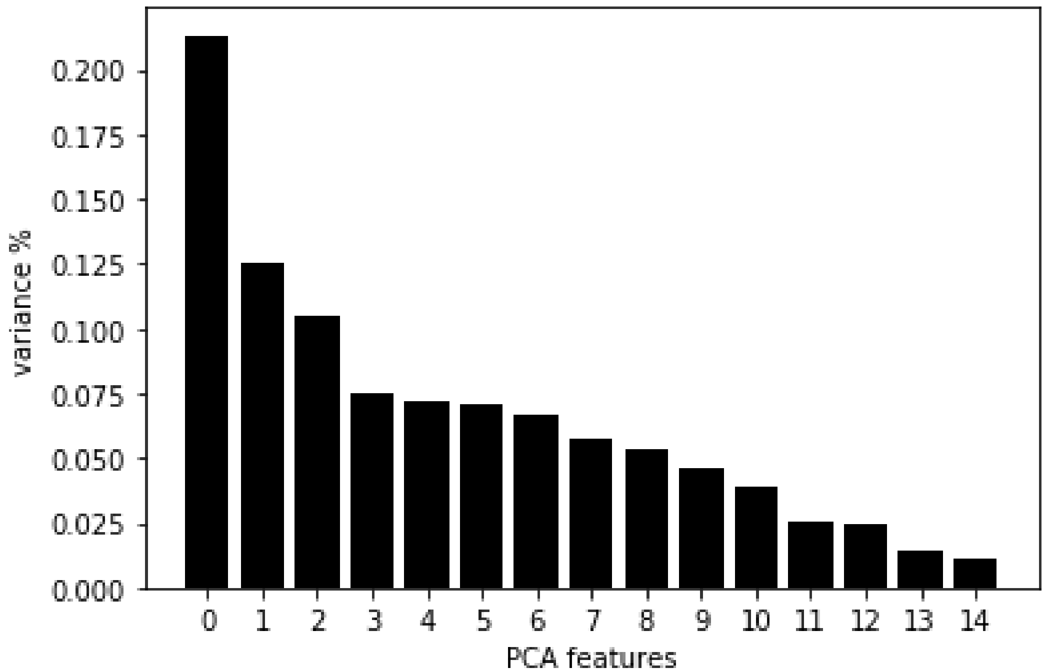
[96]:
```
vars = pca.explained_variance_ratio_
c_names = ['f1','f2','f3','f4','f5','f6','f7','f8','f9','f10']

print('Variance:  Projected dimension')
print('------------------------------')
for idx, row in enumerate(pca.components_):
    output = '{0:4.1f}%:    '.format(100.0 * vars[idx])
    output += " + ".join("{0:5.2f} * {1:s}".format(val, name) for val, name in␣
 ↪zip(row, c_names))
    print(output)
```

```
Variance:  Projected dimension
------------------------------
21.3%:    -0.05 * f1 +  0.30 * f2 + -0.11 * f3 + -0.20 * f4 + -0.17 * f5 +  0.20
* f6 +  0.07 * f7 +  0.43 * f8 +  0.14 * f9 +  0.19 * f10
12.5%:     0.35 * f1 + -0.11 * f2 + -0.02 * f3 +  0.59 * f4 +  0.63 * f5 +  0.04
* f6 + -0.02 * f7 +  0.16 * f8 + -0.02 * f9 +  0.02 * f10
10.5%:     0.06 * f1 +  0.02 * f2 + -0.03 * f3 +  0.06 * f4 +  0.05 * f5 + -0.05
```

```
* f6 + -0.02 * f7 + -0.12 * f8 +  0.69 * f9 + -0.02 * f10
 7.5%:     0.56 * f1 +  0.11 * f2 + -0.08 * f3 + -0.11 * f4 + -0.03 * f5 +  0.08
* f6 +  0.28 * f7 +  0.01 * f8 +  0.03 * f9 + -0.27 * f10
 7.1%:    -0.20 * f1 + -0.21 * f2 +  0.48 * f3 +  0.06 * f4 +  0.00 * f5 +  0.55
* f6 +  0.53 * f7 +  0.09 * f8 +  0.05 * f9 + -0.16 * f10
 7.1%:    -0.13 * f1 +  0.46 * f2 + -0.44 * f3 +  0.17 * f4 +  0.13 * f5 +  0.17
* f6 +  0.39 * f7 + -0.10 * f8 + -0.04 * f9 +  0.46 * f10
 6.7%:    -0.14 * f1 + -0.24 * f2 + -0.60 * f3 + -0.01 * f4 + -0.03 * f5 + -0.05
* f6 +  0.38 * f7 + -0.02 * f8 + -0.02 * f9 + -0.52 * f10
 5.7%:     0.18 * f1 + -0.07 * f2 +  0.30 * f3 + -0.10 * f4 + -0.03 * f5 + -0.54
* f6 +  0.57 * f7 + -0.08 * f8 + -0.03 * f9 +  0.34 * f10
 5.4%:     0.03 * f1 + -0.32 * f2 + -0.06 * f3 + -0.04 * f4 +  0.01 * f5 +  0.49
* f6 + -0.10 * f7 + -0.21 * f8 +  0.01 * f9 +  0.36 * f10
 4.6%:     0.60 * f1 +  0.28 * f2 +  0.03 * f3 + -0.23 * f4 + -0.12 * f5 +  0.27
* f6 + -0.07 * f7 + -0.08 * f8 + -0.10 * f9 + -0.10 * f10
 3.9%:     0.26 * f1 + -0.59 * f2 + -0.31 * f3 + -0.17 * f4 + -0.14 * f5 +  0.01
* f6 +  0.01 * f7 +  0.06 * f8 +  0.05 * f9 +  0.36 * f10
 2.6%:    -0.03 * f1 +  0.08 * f2 +  0.01 * f3 +  0.03 * f4 +  0.01 * f5 +  0.07
* f6 +  0.01 * f7 + -0.75 * f8 + -0.29 * f9 + -0.03 * f10
 2.5%:     0.01 * f1 + -0.08 * f2 + -0.02 * f3 +  0.02 * f4 + -0.02 * f5 + -0.02
* f6 + -0.02 * f7 +  0.33 * f8 + -0.64 * f9 +  0.03 * f10
 1.5%:    -0.11 * f1 + -0.02 * f2 +  0.00 * f3 + -0.68 * f4 +  0.71 * f5 + -0.00
* f6 +  0.00 * f7 + -0.01 * f8 + -0.02 * f9 + -0.02 * f10
 1.1%:     0.08 * f1 + -0.16 * f2 +  0.03 * f3 +  0.03 * f4 + -0.07 * f5 + -0.03
* f6 +  0.01 * f7 + -0.15 * f8 +  0.01 * f9 +  0.01 * f10
```

[97]:

```python
principalComponents = pca.fit_transform(df_Xs)
# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)
# Save components to a DataFrame
PCA_components = pd.DataFrame(principalComponents)
```



[ ]:

6 Clustering for Principal Components

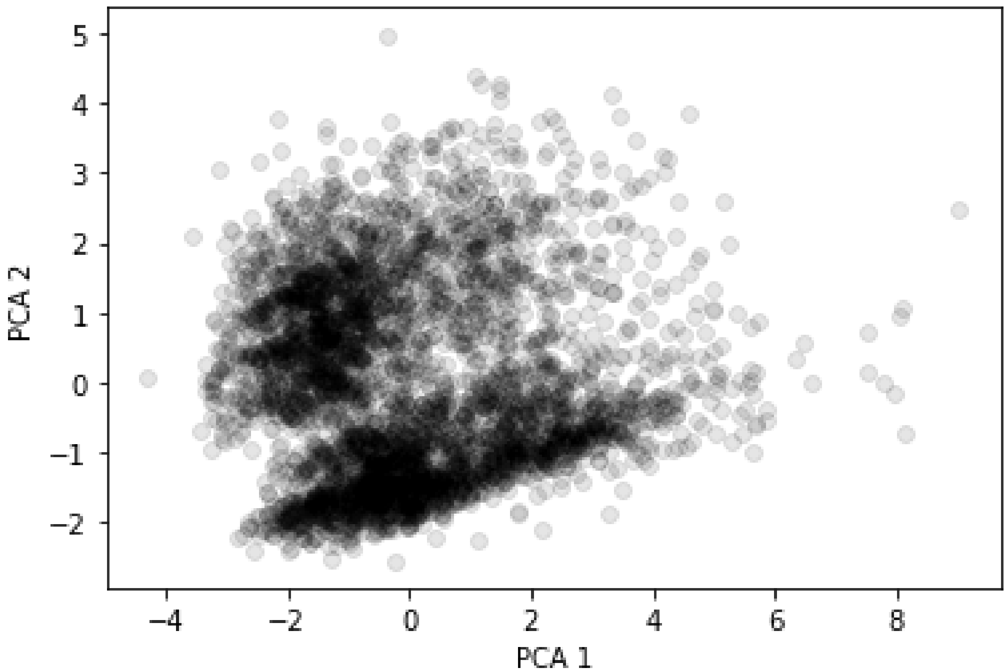
[350]:
```
PCA_components.head()
```

[350]:
```
          0         1         2         3         4         5         6  \
0 -1.813039 -1.009011  0.040934  0.572890  0.729685 -1.998924 -0.385466
1 -0.016741 -1.249846 -0.212414 -1.418829 -0.247657 -0.442280  0.744243
2 -1.005064  1.425905 -0.124089  0.492265 -0.829970  0.708200  0.286842
3  1.414644  1.680388  0.143867  0.183797  0.390372 -0.060862 -0.979922
4 -0.828819  1.056407  0.146481 -1.751294  0.606916  0.375413 -0.858495

          7         8         9        10        11        12        13  \
0  0.995189  0.691701  1.251872 -0.215770 -0.439701  0.128156  0.021566
1  0.783781  1.016209  0.505236 -0.232795  0.117530 -0.216148  0.146467
2 -0.067226  0.263629  0.312235  0.393185  0.209927 -0.320133 -0.157888
3 -0.308663 -1.013803 -1.564527 -1.698945  0.049059  0.803893  0.649640
4  0.560571  0.112961 -0.493387 -0.046228  0.585178 -0.066861  0.246891

         14
0  0.233463
```

```
1 -0.113956
2  0.169666
3 -0.583588
4 -0.082717
```

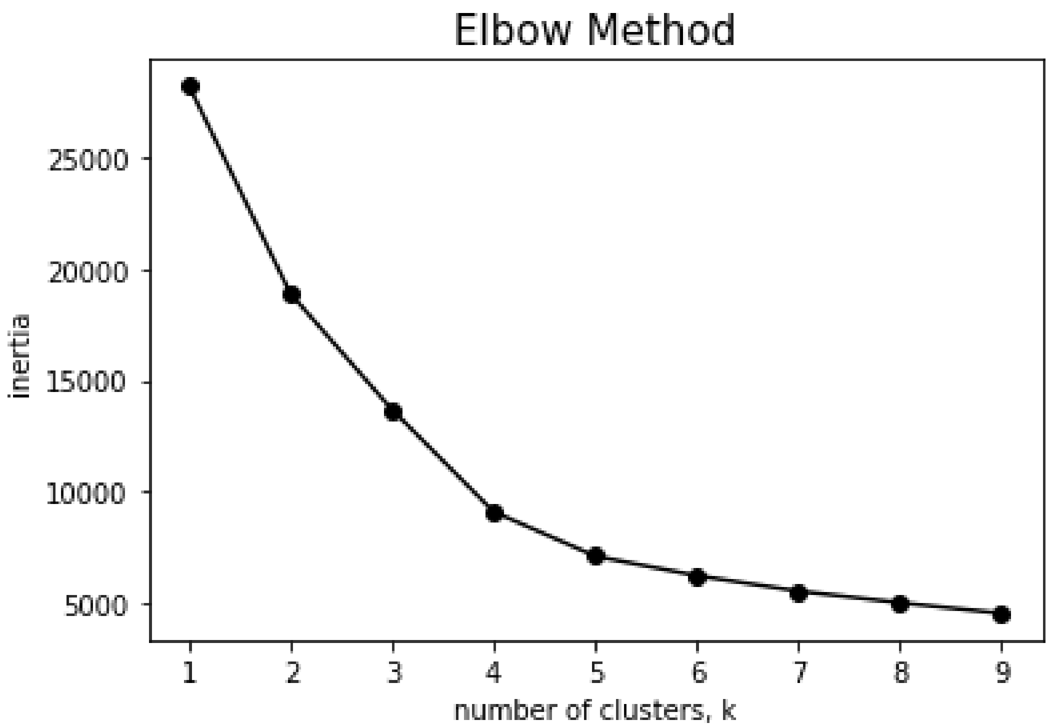
[351]:
```
plt.scatter(PCA_components[0], PCA_components[1], alpha=.1, color='black')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
```

[351]:
```
Text(0, 0.5, 'PCA 2')
```



[355]:
```
from sklearn.cluster import KMeans
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
```

```
plt.title('Elbow Method',fontsize=15)
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



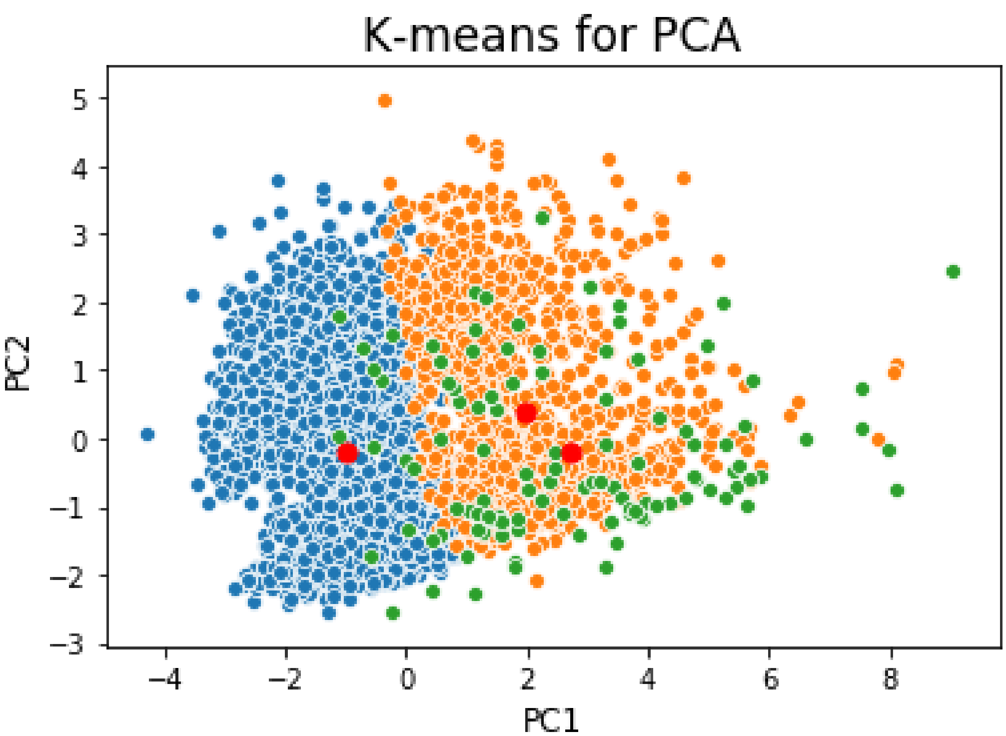
6.0.1 2D K-Means

[101]:
```
PCA_comp_v = PCA_components.values
```

[102]:
```
# 3 clusters
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 500,
                n_init = 10, random_state = 0)
pca_kmeans = kmeans.fit_predict(PCA_comp_v)

for i in range(0,3):
    sns.scatterplot(PCA_comp_v[pca_kmeans == i, 0], PCA_comp_v[pca_kmeans == i,␣
 ↪1])
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:,1], s = 50,
            c = 'red')
```

```
# xk=df_clu2[clu2_kmeans == 0, 0]
# yk=df_clu2[clu2_kmeans == 0, 1]
# mydict = {i:np.where(kmeans.labels_ == i)[0] for i in range(kmeans.
↪n_clusters)}
# print(mydict10)
# l10 = [pcaname2016[i] for i in ( 0,  5, 32, 34, 55)]
# for x, y, company in zip(xk, yk, l10):
#     plt.annotate(company, (x, y), fontsize=12, alpha=0.75)

# xk=df_clu2[df_clu2 == 2, 0]
# yk=df_clu2[df_clu2 == 2, 1]

# plt.annotate(pcaname2016[37], (xk101, yk101), fontsize=15, alpha=0.75)
plt.xlabel('PC1', fontsize=12)
plt.ylabel('PC2', fontsize=12)
plt.title('K-means for PCA',fontsize=17)
```

[102]: Text(0.5, 1.0, 'K-means for PCA')



[103]:
```
PCA_comp_v.shape
```

[103]: (4238, 15)

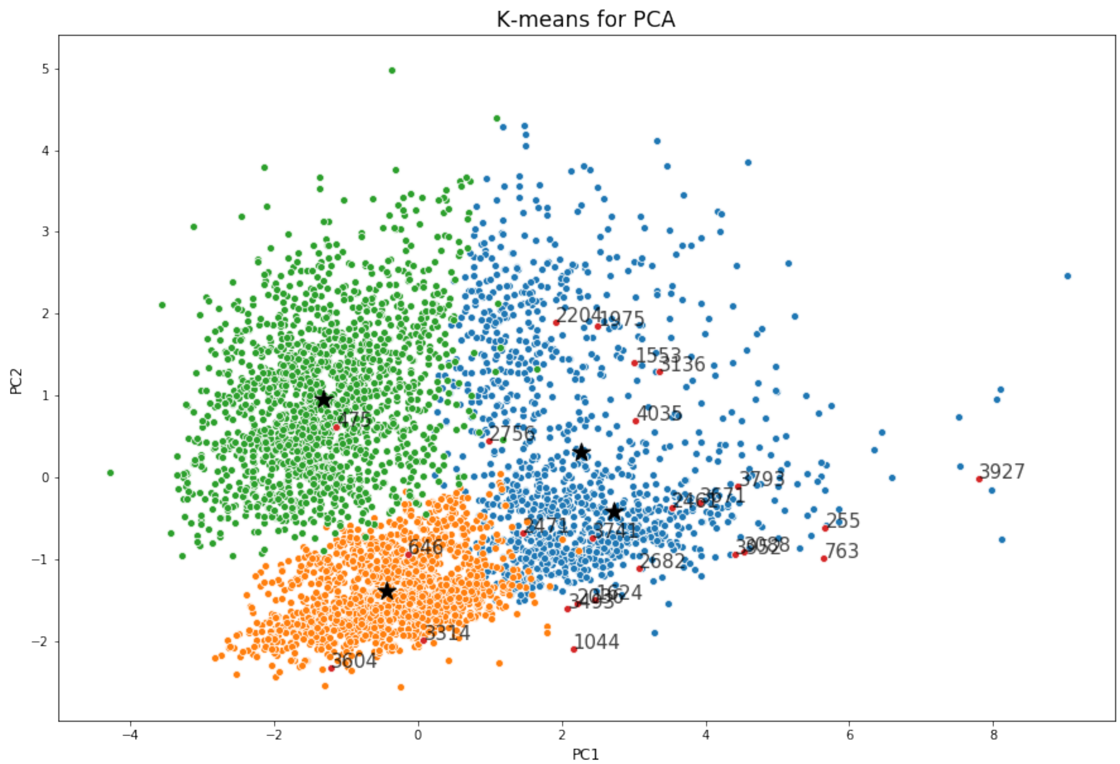
[104]:
```
l0=list(range(0,4238))
len(l0)
```

[104]: 4238

[105]:
```
# 4 clusters
kmeans = KMeans(n_clusters = 4, init = 'k-means++', max_iter = 500,
                n_init = 10, random_state = 0)
pca_kmeans = kmeans.fit_predict(PCA_comp_v)
plt.figure(figsize=(15,10))
for i in range(0,4):
    sns.scatterplot(PCA_comp_v[pca_kmeans == i, 0], PCA_comp_v[pca_kmeans == i,␣
 ↪1])
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:,1], s = 225, marker='*',
            c = 'black')

xk=PCA_comp_v[pca_kmeans == 3, 0]
yk=PCA_comp_v[pca_kmeans == 3, 1]
mydict1 = {i:np.where(kmeans.labels_ == i)[0] for i in range(kmeans.n_clusters)}
print(mydict1)
l1 = [l0[i] for i in (255,  475,  646,  763, 1044, 1553, 1624, 1975, 2036,
                      2204, 2461,2471, 2682, 2756, 3088, 3136, 3314, 3493,
                      3604, 3671, 3741, 3793,3927, 3952, 4035)]
for x, y, patient in zip(xk, yk, l1):
    plt.annotate(patient, (x, y), fontsize=15, color='black',alpha=0.75)

plt.xlabel('PC1', fontsize=12)
plt.ylabel('PC2', fontsize=12)
plt.title('K-means for PCA',fontsize=17)
```

```
{0: array([   3,    5,    8, …, 4231, 4232, 4233]), 1: array([   0,    1,
6, …, 4221, 4226, 4237]), 2: array([   2,    4,    7, …, 4234, 4235, 4236]),
3: array([ 255,  475,  646,  763, 1044, 1553, 1624, 1975, 2036, 2204, 2461,
       2471, 2682, 2756, 3088, 3136, 3314, 3493, 3604, 3671, 3741, 3793,
       3927, 3952, 4035])}
```

[105]: Text(0.5, 1.0, 'K-means for PCA')



6.0.2 3D K-Means

[106]:
```
print(PCA_components.iloc[:,0].min(),PCA_components.iloc[:,0].max())
print(PCA_components.iloc[:,1].min(),PCA_components.iloc[:,1].max())
print(PCA_components.iloc[:,2].min(),PCA_components.iloc[:,2].max())
```

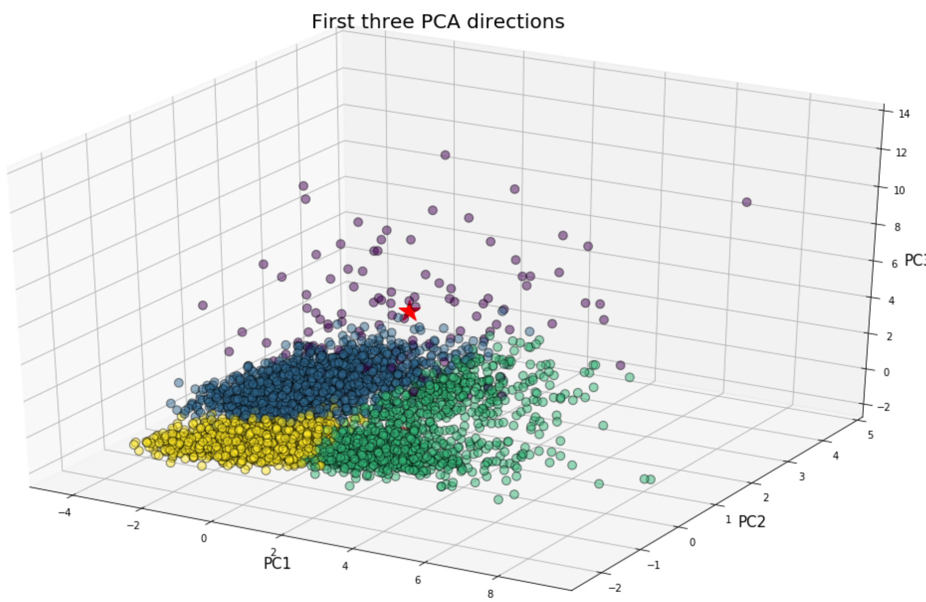
```
-4.289239269379817 9.027447803923971
-2.5453631218887507 4.983884101327462
-2.3800216139011945 13.898184226024776
```

[107]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

kmeans_PCA = KMeans(n_clusters=4, init='k-means++', max_iter= 300, n_init= 10,␣
 ↪random_state= 3)

kmeans_PCA2 = kmeans_PCA.fit_predict(PCA_components)
```

```
kmeans_PCA2

fig = plt.figure(figsize=(20,12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(PCA_components.iloc[:,0], PCA_components.iloc[:,1], PCA_components.
 ↪iloc[:,2],
            c=kmeans_PCA2, cmap='viridis',
            edgecolor='k', s=75, alpha = 0.5)

# xk2=PCA_comp_v[kmeans_PCA2 == 0]
# yk2=PCA_components[kmeans_PCA2 == 0]
# zk2=PCA_components[kmeans_PCA2 == 0]
mydict2 = {i:np.where(kmeans_PCA.labels_ == i)[0] for i in range(kmeans_PCA.
 ↪n_clusters)}
print(mydict2)
# l2 = [l0[i] for i in (37,   44,   56,   66,   96,  247,  249,  260,  284, ␣
 ↪294,  311,
#                        357,  421,  443,  451,  471,  585,  763,  833,  903, ␣
 ↪952, 1030,
#                        1068, 1111, 1123, 1165, 1238, 1268, 1303, 1333, 1340,␣
 ↪1363, 1485,
#                        1649, 1674, 1854, 1895, 1907, 1931, 1997, 2024, 2041,␣
 ↪2091, 2098,
#                        2180, 2217, 2234, 2378, 2393, 2406, 2498, 2503, 2528,␣
 ↪2570, 2600,
#                        2645, 2649, 2668, 2784, 2801, 2802, 2849, 2855, 2891,␣
 ↪2893, 2909,
#                        2926, 2961, 3002, 3051, 3112, 3140, 3203, 3242, 3256,␣
 ↪3300, 3321,
#                        3327, 3449, 3458, 3552, 3606, 3620, 3680, 3682, 3721,␣
 ↪3739, 3749,
#                        3763, 3778, 3797, 3809, 3817, 3839, 3844, 3849, 3868,␣
 ↪3895, 3971,
#                        3974, 4042, 4064, 4076, 4084, 4096, 4154, 4203, 4215,␣
 ↪4228)]
# for x, patient in zip(xk2, l2):
#     plt.annotate(patient, x, fontsize=15, color='black',alpha=0.75)


ax.set_title("First three PCA directions").set_size(20)
ax.set_xlabel("PC1").set_size(15)
ax.set_xlim([-4.3,9.2])
ax.set_ylabel("PC2").set_size(15)
ax.set_ylim([-2.6,5])
ax.set_zlabel("PC3").set_size(15)
ax.set_zlim([-2.4,14])
```

```
ax.dist = 10

ax.scatter(kmeans_PCA.cluster_centers_[:,0], kmeans_PCA.cluster_centers_[:,1],
           kmeans_PCA.cluster_centers_[:,2],
           s = 500, c = 'r', marker='*', label = 'Centroid')

plt.autoscale(enable=True, axis='x', tight=True)

plt.show()
```

```
{0: array([  37,   44,   56,   66,   96,  247,  249,  260,  284,  294,  311,
        357,  421,  443,  451,  471,  585,  763,  833,  903,  952, 1030,
       1068, 1111, 1123, 1165, 1238, 1268, 1303, 1333, 1340, 1363, 1485,
       1649, 1674, 1854, 1895, 1907, 1931, 1997, 2024, 2041, 2091, 2098,
       2180, 2217, 2234, 2378, 2393, 2406, 2498, 2503, 2528, 2570, 2600,
       2645, 2649, 2668, 2784, 2801, 2802, 2849, 2855, 2891, 2893, 2909,
       2926, 2961, 3002, 3051, 3112, 3140, 3203, 3242, 3256, 3300, 3321,
       3327, 3449, 3458, 3552, 3606, 3620, 3680, 3682, 3721, 3739, 3749,
       3763, 3778, 3797, 3809, 3817, 3839, 3844, 3849, 3868, 3895, 3971,
       3974, 4042, 4064, 4076, 4084, 4096, 4154, 4203, 4215, 4228]), 1: array([
2,    4,    7, …, 4234, 4235, 4236]), 2: array([   3,    5,    8, …, 4231,
4232, 4233]), 3: array([   0,    1,    6, …, 4221, 4226, 4237])}
```



[108]:
```
sus_l1 = [255,  475,  646,  763, 1044, 1553, 1624, 1975, 2036, 2204, 2461,
          2471, 2682, 2756, 3088, 3136, 3314, 3493, 3604, 3671, 3741, 3793,
          3927, 3952, 4035]
```

```
sus_l2 = [37,   44,   56,   66,   96,  247,  249,  260,  284,  294,  311,
          357,  421,  443,  451,  471,  585,  763,  833,  903,  952, 1030,
          1068, 1111, 1123, 1165, 1238, 1268, 1303, 1333, 1340, 1363, 1485,
          1649, 1674, 1854, 1895, 1907, 1931, 1997, 2024, 2041, 2091, 2098,
          2180, 2217, 2234, 2378, 2393, 2406, 2498, 2503, 2528, 2570, 2600,
          2645, 2649, 2668, 2784, 2801, 2802, 2849, 2855, 2891, 2893, 2909,
          2926, 2961, 3002, 3051, 3112, 3140, 3203, 3242, 3256, 3300, 3321,
          3327, 3449, 3458, 3552, 3606, 3620, 3680, 3682, 3721, 3739, 3749,
          3763, 3778, 3797, 3809, 3817, 3839, 3844, 3849, 3868, 3895, 3971,
          3974, 4042, 4064, 4076, 4084, 4096, 4154, 4203, 4215, 4228]
c = set(sus_l1) & set(sus_l2)  # & calculates the intersection.
print(c)
```

{763}

[109]:
```
df_X[df_X.index==763]
```

[109]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	\
763	0	58	1	0	0.0	0	1	

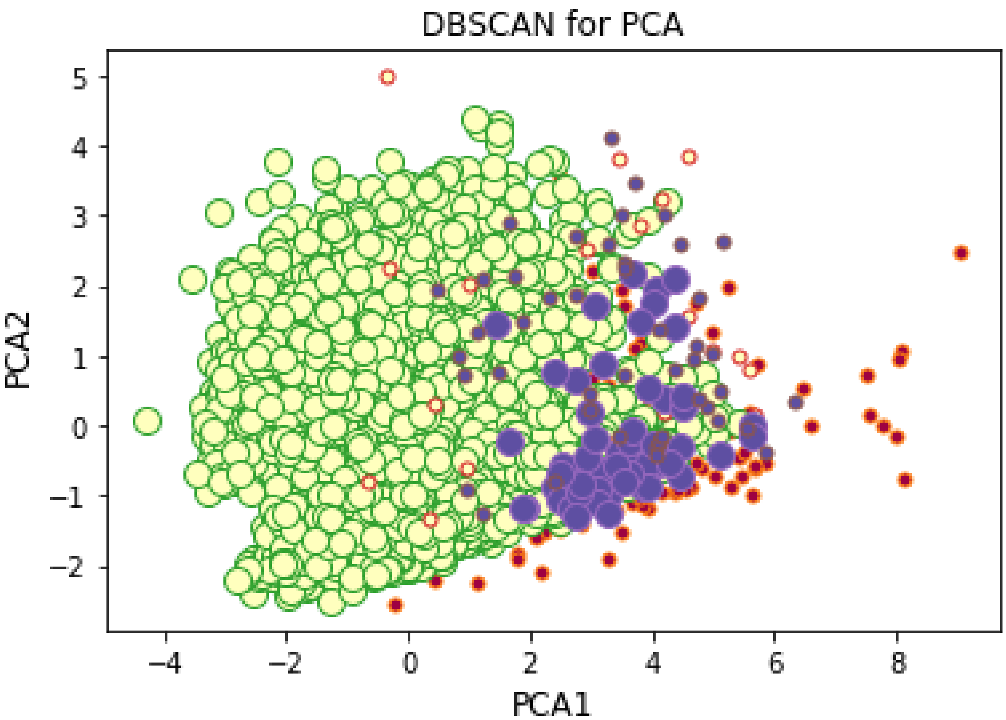
	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose
763	1	1	267.0	157.0	94.0	33.32	92.0	205.0

6.0.3 2D DBSCAN

[110]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN

dbsc = DBSCAN(eps = 3.9, min_samples = 30).fit(PCA_comp_v)
labels = dbsc.labels_
core_samples = np.zeros_like(labels, dtype = bool)
core_samples[dbsc.core_sample_indices_] = True

unique_labels = np.unique(labels)
colors = plt.cm.Spectral(np.linspace(0,1, len(unique_labels)))
for (label, color) in zip(unique_labels, colors):
    class_member_mask = (labels == label)
    xy = PCA_comp_v[class_member_mask & core_samples]
    plt.plot(xy[:,0],xy[:,1], 'o', markerfacecolor = color, markersize = 10)

    xy2 = PCA_comp_v[class_member_mask & ~core_samples]
    plt.plot(xy2[:,0],xy2[:,1], 'o', markerfacecolor = color, markersize = 5)

plt.xlabel('PCA1', fontsize=12)
```

```
plt.ylabel('PCA2', fontsize=12)
plt.title("DBSCAN for PCA")
```

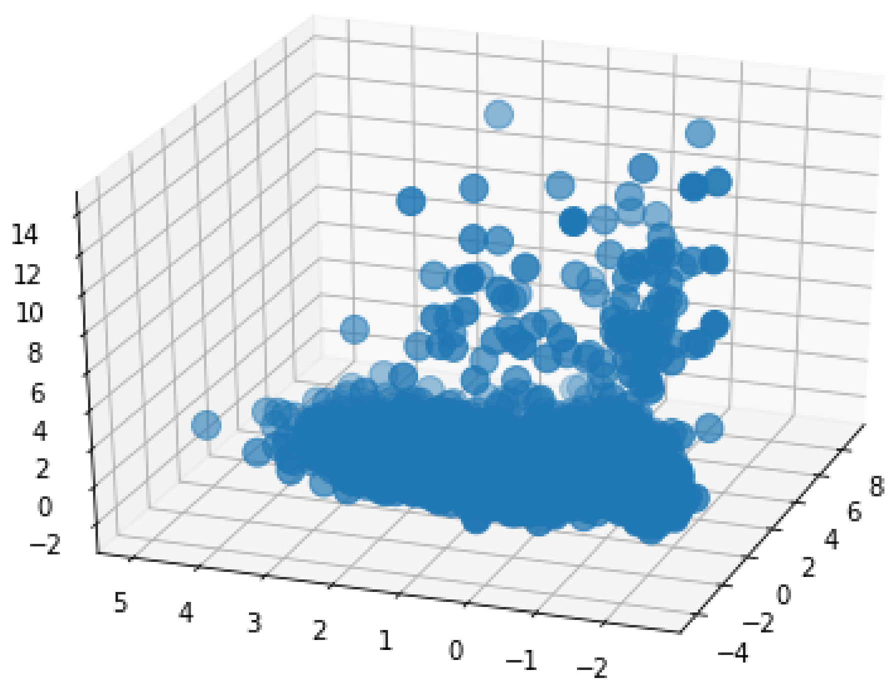
[110]: Text(0.5, 1.0, 'DBSCAN for PCA')

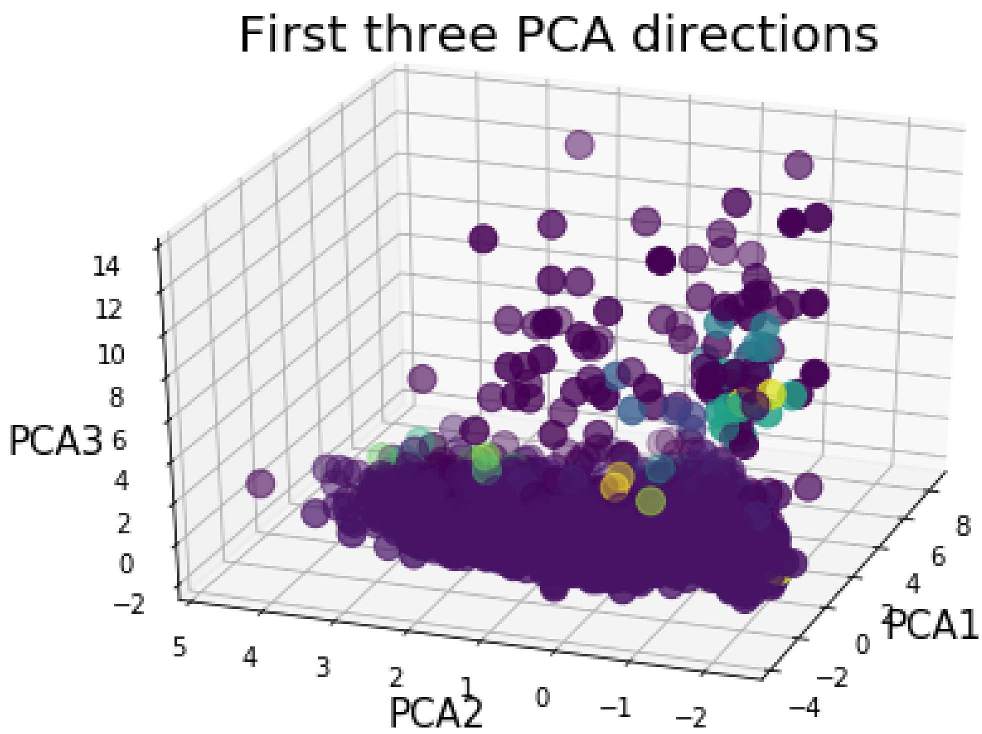


6.0.4 3D DBSCAN

[111]:
```
from sklearn.cluster import DBSCAN

# -4.289239269379817 9.027447803923971
# -2.5453631218887507 4.983884101327462
# -2.3800216139011945 13.898184226024776

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(PCA_components.iloc[:,0], PCA_components.iloc[:,1], PCA_components.
↪iloc[:,2], s=120)
ax.view_init(azim=200)
plt.show()

dbscan = DBSCAN(eps=2.5, min_samples=2)
dbscan.fit_predict(PCA_components)
```

```
pred_dbscan = dbscan.fit_predict(PCA_components)

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(PCA_components.iloc[:,0], PCA_components.iloc[:,1], PCA_components.
 ↪iloc[:,2], c=dbscan.labels_, s=120)
ax.set_title("First three PCA directions").set_size(20)
ax.set_xlabel("PCA1").set_size(15)
ax.set_xlim([-4.3,9.1])
ax.set_ylabel("PCA2").set_size(15)
ax.set_ylim([-2.6,5])
ax.set_zlabel("PCA3").set_size(15)
ax.set_zlim([-2.4,14])
ax.view_init(azim=200)
plt.show()

print("number of cluster found: {}".format(len(set(dbscan.labels_))))
print('cluster for each point: ', dbscan.labels_)
```






```
number of cluster found: 22
cluster for each point:  [0 0 0 … 0 0 0]
```

[112]:
```
# Hierarchical Clustering is not appropriate for this dataset due to too many␣
↪data points;
# the dendrogram won't be able to show all the labes for every patient
# df_clu2

# from scipy.cluster.hierarchy import linkage, dendrogram
# from scipy.cluster.hierarchy import fcluster
# plt.figure(figsize=(15, 10))
# plt.title("Dendograms for Glucose vs. sysBP vs. totChol")
# mergings = linkage(df_clu2, method='complete')#ward)
# labels=df_clu1.index
# #labels = fcluster(mergings, 70, criterion='distance')
# dendrogram(mergings,
#             labels=df_clu1.index,
#             leaf_rotation=90,
#             leaf_font_size=6)

# plt.show()

# print(labels)
```

```
[ ]:
```

7 Supervised Learning Prediction starts here ↓↓↓

```
[113]: # df.iloc[13:25,:]
```

```
[114]: dummy_education  = pd.get_dummies(df['education'])
       # dummy_education.head()
       df = df.join(dummy_education.loc[:,:]).drop(['education'],axis=1)
       df['currentSmoker'] = df['currentSmoker'].apply(str)
       df['BPMeds'] = df['BPMeds'].astype(str)
       df['prevalentStroke'] = df['prevalentStroke'].astype(str)
       df['prevalentHyp'] = df['prevalentHyp'].astype(str)
       df['diabetes'] = df['diabetes'].astype(str)
```

```
[115]: # I can simply drop the "TenYearCHD" column for dataset X and recombine this␣
        ↪column,
       # but I just want to play with the new function in Python 3.5 of *list␣
        ↪concatenation to change the column orders.
       df=df.iloc[:, [*list(range(0,14)),*list(range(15,19)),14]]
```

```
[116]: df.head(3)
```

```
[116]:    male  age currentSmoker  cigsPerDay BPMeds prevalentStroke prevalentHyp  \

       0     1   39             0         0.0      0               0            0
       1     0   46             0         0.0      0               0            0
       2     1   48             1        20.0      0               0            0

         diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  1  2  3  4  \

       0        0    195.0  106.0   70.0  26.97       80.0     77.0  0  0  0  1
       1        0    250.0  121.0   81.0  28.73       95.0     76.0  0  1  0  0
       2        0    245.0  127.5   80.0  25.34       75.0     70.0  1  0  0  0

          TenYearCHD

       0           0
       1           0
       2           0
```

```
[117]: df.shape
```

```
[117]: (4238, 19)
```

[118]:
```
df_X = df.iloc[:,0:18]
df_X.head(1)
```

[118]:

	male	age	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	\
0	1	39	0	0.0	0	0	0	

	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	1	2	3	4
0	0	195.0	106.0	70.0	26.97	80.0	77.0	0	0	0	1

[119]:
```
df_y = df[['TenYearCHD']]
df_y.head(2)
```

[119]:

	TenYearCHD
0	0
1	0

[120]:
```
X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, test_size=0.2,␣
↪random_state=1)
```

[121]:
```
X_train.head(4)
```

[121]:

	male	age	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	\
3873	1	46	0	0.0	0	0	0	
781	1	35	1	40.0	0	0	0	
703	0	47	0	0.0	0	0	0	
2675	1	37	1	30.0	0	0	0	

	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	1	2	3	4
3873	0	188.0	135.0	95.0	26.84	60.0	78.0	0	1	0	0
781	0	175.0	112.0	62.5	21.03	73.0	69.0	0	0	1	0
703	0	294.0	109.0	72.5	28.59	82.0	77.0	0	0	1	0
2675	0	179.0	131.5	81.0	24.99	64.0	68.0	0	1	0	0

[122]:
```
y_train.head(4)
```

[122]:

	TenYearCHD
3873	0
781	1
703	0
2675	0

```
[123]: X_test.head(3)
```

```
[123]:       male  age  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  prevalentHyp  \

906      1   53              0         0.0       0                0             0
857      0   44              0         0.0       0                0             0
2761     0   44              0         0.0       0                0             0

     diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  1  2  3  4

906         0    220.0  127.0   76.0  24.27       75.0     74.0  1  0  0  0
857         0    195.0  118.0   86.0  23.09       70.0     75.0  1  0  0  0
2761        0    205.0  109.0   73.0  17.48       75.0     57.0  0  0  0  1
```

```
[124]: y_test.head(3)
```

```
[124]:       TenYearCHD

906            0
857            0
2761           0
```

```
[ ]:
```

8 Logistic Regression

```
[125]: from sklearn.linear_model import LogisticRegression
from random import seed
seed(0)
# skf = StratifiedKFold(n_splits=5)
params0 = {'tol' : [1e-6,1e-5,1e-4,1e-3,1e-2],
           'C': [0.5,1.0,1.5,2.0,2.5]}
lg = LogisticRegression(random_state=0,␣
 ↪solver='lbfgs',multi_class='multinomial')
lg = GridSearchCV(lg, cv=5, param_grid=params0, scoring = 'roc_auc',refit =␣
 ↪True,
                  n_jobs=-1, verbose = 5, return_train_score=True)

lg.fit(X_train, y_train)
lg.cv_results_
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done 125 out of 125 | elapsed:    3.9s finished
```

```
/usr/local/lib/python3.7/site-packages/sklearn/utils/validation.py:724:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

[125]:
```
{'mean_fit_time': array([0.13494382, 0.15549984, 0.1663054 , 0.16501551,
0.12920084,
        0.13545723, 0.11134796, 0.12574835, 0.12917285, 0.11852283,
        0.13262506, 0.13438334, 0.15777016, 0.15159383, 0.14510322,
        0.12966862, 0.14908481, 0.11831198, 0.13588028, 0.13074684,
        0.14269876, 0.13632259, 0.13733511, 0.13395033, 0.12280803]),
 'std_fit_time': array([0.01281036, 0.00659525, 0.01652828, 0.01965519,
0.01130766,
        0.01334009, 0.01367391, 0.01897641, 0.00743502, 0.00912159,
        0.01000369, 0.01106381, 0.03444004, 0.00291463, 0.01739689,
        0.00855178, 0.01061831, 0.01884763, 0.00902231, 0.01674911,
        0.03728541, 0.00787666, 0.02326014, 0.01398581, 0.0072793 ]),
 'mean_score_time': array([0.00597911, 0.00655198, 0.0062551 , 0.00560064,
0.00539465,
        0.00525498, 0.00517278, 0.00522857, 0.00479655, 0.00497141,
        0.005233  , 0.00482764, 0.00749407, 0.0062799 , 0.00567918,
        0.00526562, 0.00503817, 0.00531459, 0.00674019, 0.00601277,
        0.00668201, 0.00510526, 0.00527782, 0.00446091, 0.00397067]),
 'std_score_time': array([2.62285119e-04, 4.91027078e-04, 2.27473136e-04,
3.79400674e-04,
        1.59232014e-04, 2.18491188e-04, 9.34141396e-05, 7.50495941e-05,
        6.64019814e-04, 4.20124944e-04, 2.74456732e-04, 5.98254740e-04,
        2.63132434e-03, 1.79737410e-03, 4.64220514e-04, 9.57797911e-05,
        1.43202908e-04, 1.90126821e-04, 2.18247856e-03, 1.58411531e-03,
        2.49426530e-03, 8.86371250e-04, 1.62947920e-04, 2.30555160e-04,
        7.86989364e-04]),
 'param_C': masked_array(data=[0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0,
1.5,
                    1.5, 1.5, 1.5, 1.5, 2.0, 2.0, 2.0, 2.0, 2.0, 2.5, 2.5,
                    2.5, 2.5, 2.5],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False],
        fill_value='?',
             dtype=object),
 'param_tol': masked_array(data=[1e-06, 1e-05, 0.0001, 0.001, 0.01, 1e-06,
1e-05,
```

```
                    0.0001, 0.001, 0.01, 1e-06, 1e-05, 0.0001, 0.001, 0.01,
                    1e-06, 1e-05, 0.0001, 0.001, 0.01, 1e-06, 1e-05,
                    0.0001, 0.001, 0.01],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.5, 'tol': 1e-06},
  {'C': 0.5, 'tol': 1e-05},
  {'C': 0.5, 'tol': 0.0001},
  {'C': 0.5, 'tol': 0.001},
  {'C': 0.5, 'tol': 0.01},
  {'C': 1.0, 'tol': 1e-06},
  {'C': 1.0, 'tol': 1e-05},
  {'C': 1.0, 'tol': 0.0001},
  {'C': 1.0, 'tol': 0.001},
  {'C': 1.0, 'tol': 0.01},
  {'C': 1.5, 'tol': 1e-06},
  {'C': 1.5, 'tol': 1e-05},
  {'C': 1.5, 'tol': 0.0001},
  {'C': 1.5, 'tol': 0.001},
  {'C': 1.5, 'tol': 0.01},
  {'C': 2.0, 'tol': 1e-06},
  {'C': 2.0, 'tol': 1e-05},
  {'C': 2.0, 'tol': 0.0001},
  {'C': 2.0, 'tol': 0.001},
  {'C': 2.0, 'tol': 0.01},
  {'C': 2.5, 'tol': 1e-06},
  {'C': 2.5, 'tol': 1e-05},
  {'C': 2.5, 'tol': 0.0001},
  {'C': 2.5, 'tol': 0.001},
  {'C': 2.5, 'tol': 0.01}],
 'split0_test_score': array([0.6793276 , 0.6793276 , 0.6793276 , 0.6793276 ,
0.6793276 ,
        0.67985446, 0.67985446, 0.67985446, 0.67985446, 0.67985446,
        0.67988739, 0.67988739, 0.67988739, 0.67988739, 0.67988739,
        0.67992031, 0.67992031, 0.67992031, 0.67992031, 0.67992031,
        0.67960749, 0.67960749, 0.67960749, 0.67960749, 0.67960749]),
 'split1_test_score': array([0.68752474, 0.68752474, 0.68752474, 0.68752474,
0.68752474,
        0.68455601, 0.68455601, 0.68455601, 0.68455601, 0.68455601,
        0.69547434, 0.69547434, 0.69547434, 0.69547434, 0.69547434,
        0.68600739, 0.68600739, 0.68600739, 0.68600739, 0.68600739,
        0.68363241, 0.68363241, 0.68363241, 0.68363241, 0.68363241]),
 'split2_test_score': array([0.65694683, 0.65694683, 0.65694683, 0.65694683,
```

```
0.65694683,
       0.64822206, 0.64822206, 0.64822206, 0.64822206, 0.64822206,
       0.6451049 , 0.6451049 , 0.6451049 , 0.6451049 , 0.6451049 ,
       0.6446266 , 0.6446266 , 0.6446266 , 0.6446266 , 0.6446266 ,
       0.64787571, 0.64787571, 0.64787571, 0.64787571, 0.64787571]),
 'split3_test_score': array([0.67179047, 0.67179047, 0.67179047, 0.67179047,
0.67179047,
       0.67177398, 0.67177398, 0.67177398, 0.67177398, 0.67177398,
       0.67004222, 0.67004222, 0.67004222, 0.67004222, 0.67004222,
       0.67170801, 0.67170801, 0.67170801, 0.67170801, 0.67170801,
       0.6719719 , 0.6719719 , 0.6719719 , 0.6719719 , 0.6719719 ]),
 'split4_test_score': array([0.63051948, 0.63051948, 0.63051948, 0.63051948,
0.63051948,
       0.6495671 , 0.6495671 , 0.6495671 , 0.6495671 , 0.6495671 ,
       0.63994339, 0.63994339, 0.63994339, 0.63994339, 0.63994339,
       0.62873793, 0.62873793, 0.62873793, 0.62873793, 0.62873793,
       0.63106893, 0.63106893, 0.63106893, 0.63106893, 0.63106893]),
 'mean_test_score': array([0.66523622, 0.66523622, 0.66523622, 0.66523622,
0.66523622,
       0.66680366, 0.66680366, 0.66680366, 0.66680366, 0.66680366,
       0.66610223, 0.66610223, 0.66610223, 0.66610223, 0.66610223,
       0.66221515, 0.66221515, 0.66221515, 0.66221515, 0.66221515,
       0.66284561, 0.66284561, 0.66284561, 0.66284561, 0.66284561]),
 'std_test_score': array([0.02005385, 0.02005385, 0.02005385, 0.02005385,
0.02005385,
       0.01518131, 0.01518131, 0.01518131, 0.01518131, 0.01518131,
       0.02094144, 0.02094144, 0.02094144, 0.02094144, 0.02094144,
       0.02190228, 0.02190228, 0.02190228, 0.02190228, 0.02190228,
       0.02014508, 0.02014508, 0.02014508, 0.02014508, 0.02014508]),
 'rank_test_score': array([11, 11, 11, 11, 11,  1,  1,  1,  1,  1,  6,  6,  6,
6,  6, 21, 21,
       21, 21, 21, 16, 16, 16, 16, 16], dtype=int32),
 'split0_train_score': array([0.66607462, 0.66607462, 0.66607462, 0.66607462,
0.66607462,
       0.66632466, 0.66632466, 0.66632466, 0.66632466, 0.66632466,
       0.66653441, 0.66653441, 0.66653441, 0.66653441, 0.66653441,
       0.66686815, 0.66686815, 0.66686815, 0.66686815, 0.66686815,
       0.66578324, 0.66578324, 0.66578324, 0.66578324, 0.66578324]),
 'split1_train_score': array([0.67339119, 0.67339119, 0.67339119, 0.67339119,
0.67339119,
       0.67133593, 0.67133593, 0.67133593, 0.67133593, 0.67133593,
       0.68031711, 0.68031711, 0.68031711, 0.68031711, 0.68031711,
       0.67202687, 0.67202687, 0.67202687, 0.67202687, 0.67202687,
       0.67129772, 0.67129772, 0.67129772, 0.67129772, 0.67129772]),
 'split2_train_score': array([0.68689085, 0.68689085, 0.68689085, 0.68689085,
0.68689085,
       0.69079584, 0.69079584, 0.69079584, 0.69079584, 0.69079584,
```

```
        0.68091716, 0.68091716, 0.68091716, 0.68091716, 0.68091716,
        0.68078083, 0.68078083, 0.68078083, 0.68078083, 0.68078083,
        0.68074365, 0.68074365, 0.68074365, 0.68074365, 0.68074365]),
 'split3_train_score': array([0.67030417, 0.67030417, 0.67030417, 0.67030417,
0.67030417,
        0.67085775, 0.67085775, 0.67085775, 0.67085775, 0.67085775,
        0.67084432, 0.67084432, 0.67084432, 0.67084432, 0.67084432,
        0.67100544, 0.67100544, 0.67100544, 0.67100544, 0.67100544,
        0.67080714, 0.67080714, 0.67080714, 0.67080714, 0.67080714]),
 'split4_train_score': array([0.68101647, 0.68101647, 0.68101647, 0.68101647,
0.68101647,
        0.68735008, 0.68735008, 0.68735008, 0.68735008, 0.68735008,
        0.6888204 , 0.6888204 , 0.6888204 , 0.6888204 , 0.6888204 ,
        0.68117308, 0.68117308, 0.68117308, 0.68117308, 0.68117308,
        0.68170887, 0.68170887, 0.68170887, 0.68170887, 0.68170887]),
 'mean_train_score': array([0.67553546, 0.67553546, 0.67553546, 0.67553546,
0.67553546,
        0.67733285, 0.67733285, 0.67733285, 0.67733285, 0.67733285,
        0.67748668, 0.67748668, 0.67748668, 0.67748668, 0.67748668,
        0.67437087, 0.67437087, 0.67437087, 0.67437087, 0.67437087,
        0.67406812, 0.67406812, 0.67406812, 0.67406812, 0.67406812]),
 'std_train_score': array([0.00748943, 0.00748943, 0.00748943, 0.00748943,
0.00748943,
        0.00980477, 0.00980477, 0.00980477, 0.00980477, 0.00980477,
        0.00790324, 0.00790324, 0.00790324, 0.00790324, 0.00790324,
        0.00566514, 0.00566514, 0.00566514, 0.00566514, 0.00566514,
        0.00616266, 0.00616266, 0.00616266, 0.00616266, 0.00616266])}
```

[126]:

```
lg.best_estimator_
```

[126]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='multinomial', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=1e-06, verbose=0,
                   warm_start=False)
```

[127]:

```
lg_pred = lg.best_estimator_.predict(X_test)
lg_prob = lg.best_estimator_.predict_proba(X_test)
lg_prob
```

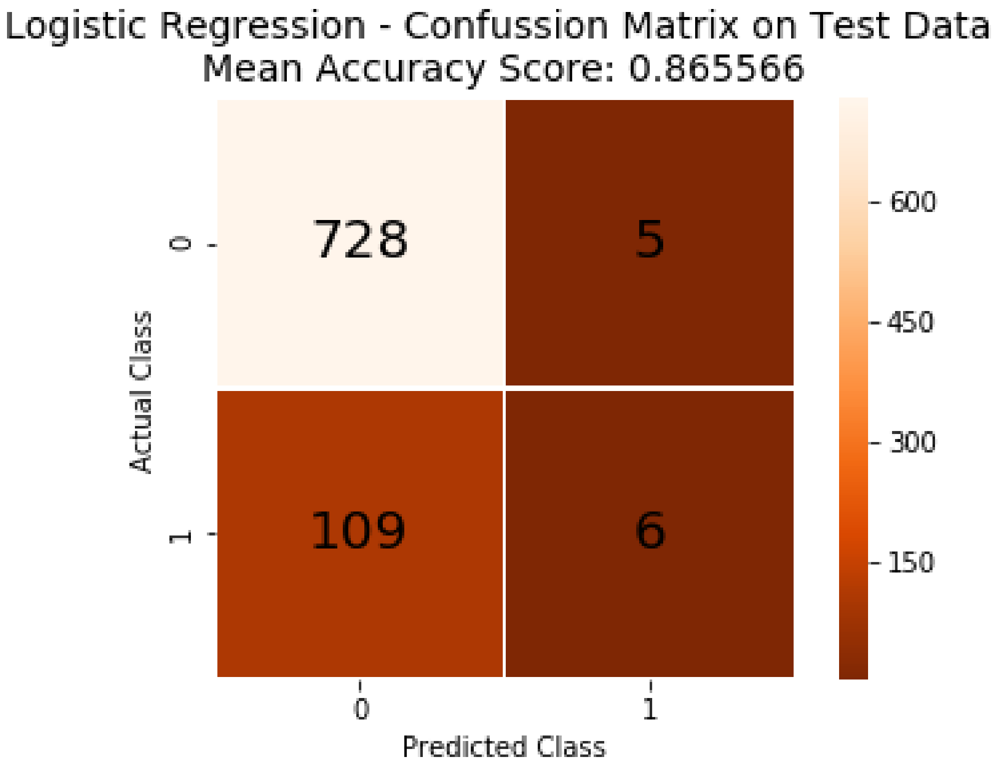
[127]:

```
array([[0.78618475, 0.21381525],
       [0.89521862, 0.10478138],
       [0.90267542, 0.09732458],
       …,
       [0.90201788, 0.09798212],
       [0.86992896, 0.13007104],
       [0.92622619, 0.07377381]])
```

[128]:
```python
from sklearn import metrics
lg_matrix = metrics.confusion_matrix(y_test, lg_pred)
lg_matrix
```

[128]:
```
array([[728,   5],
       [109,   6]])
```

[129]:
```python
# from matplotlib.pyplot import figure

lg_test = lg.best_estimator_.score(X_test, y_test)

lg_matrix = metrics.confusion_matrix(y_test, lg_pred)
# lg_matrix = metrics.confusion_matrix(y_test, lg_pred)
lg_cm = pd.DataFrame(lg_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (lg_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

lg_title = 'Logistic Regression - Confussion Matrix on Test Data \nMean␣
 ↪Accuracy Score: {0:2f}'.format(lg_test)
plt.title(lg_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



[130]:
```
from sklearn.metrics import classification_report
target_names = ['No risk in 10 years', 'Risky in 10 years']
print("", classification_report(y_test, lg_pred, target_names=target_names))
```

```
                      precision    recall  f1-score   support

No risk in 10 years       0.87      0.99      0.93       733
  Risky in 10 years       0.55      0.05      0.10       115

           accuracy                           0.87       848
          macro avg       0.71      0.52      0.51       848
       weighted avg       0.83      0.87      0.81       848
```

[131]:
```
from sklearn.metrics import accuracy_score
acc_lg = accuracy_score(y_test, lg_pred)
print("Logistic Regression accuracy:", acc_lg)
```

```
Logistic Regression accuracy: 0.8655660377358491
```

[132]:
```
from sklearn.metrics import roc_auc_score
lg_probs = lg.best_estimator_.predict_proba(X_test)[:,1]
```

```
print(roc_auc_score(y_test, lg_probs))
```

```
0.6629100183878047
```

[133]:
```
error_lg = 1-acc_lg
error_lg
```

[133]:
```
0.13443396226415094
```

8.0.1 For training set:

[134]:
```
lg_pred_tr = lg.best_estimator_.predict(X_train)
lg_prob_tr = lg.best_estimator_.predict_proba(X_train)
lg_prob_tr
```

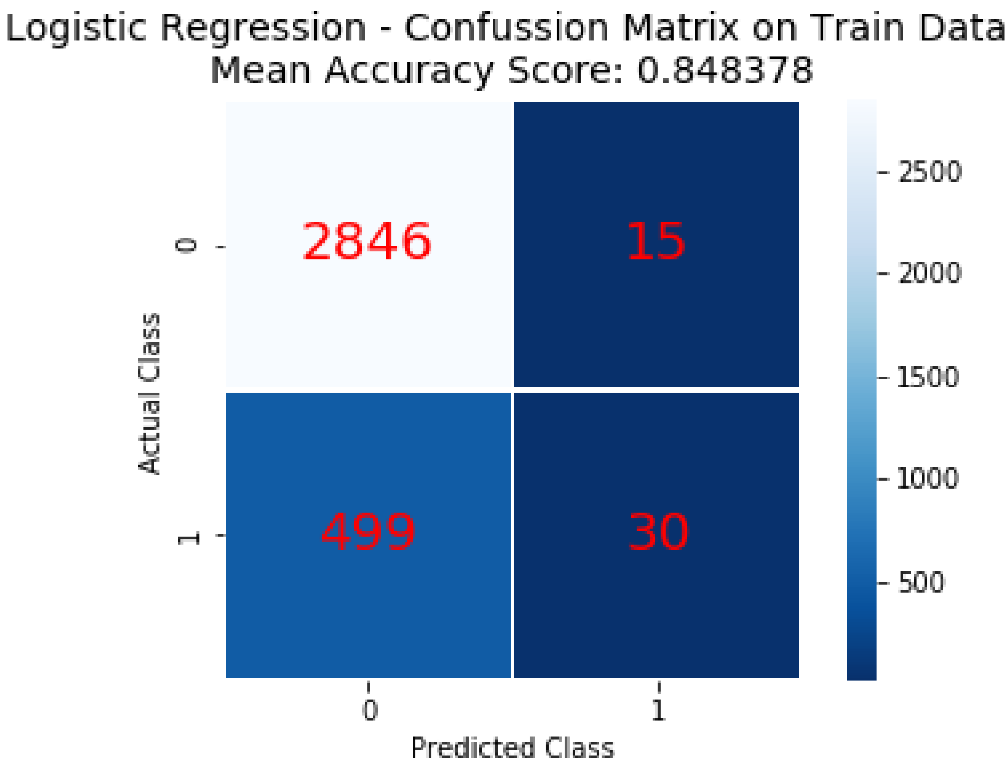
[134]:
```
array([[0.90857767, 0.09142233],
       [0.79616068, 0.20383932],
       [0.95181101, 0.04818899],
       …,
       [0.93938182, 0.06061818],
       [0.94126663, 0.05873337],
       [0.90208632, 0.09791368]])
```

[135]:
```
lg_matrix_tr = metrics.confusion_matrix(y_train, lg_pred_tr)
lg_matrix_tr
```

[135]:
```
array([[2846,   15],
       [ 499,   30]])
```

[136]:
```
lg_train = lg.best_estimator_.score(X_train, y_train)

lg_matrix = metrics.confusion_matrix(y_train, lg_pred_tr)

lg_cm_tr = pd.DataFrame(lg_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (lg_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
 ↪5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')
```

```
lg_title = 'Logistic Regression - Confussion Matrix on Train Data \nMean␣
↪Accuracy Score: {0:2f}'.format(lg_train)
plt.title(lg_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



[137]:
```
print("", classification_report(y_train, lg_pred_tr, target_names=target_names))
```

```
                   precision    recall  f1-score   support

 No risk in 10 years       0.85      0.99      0.92      2861
   Risky in 10 years       0.67      0.06      0.10       529

            accuracy                           0.85      3390
           macro avg       0.76      0.53      0.51      3390
        weighted avg       0.82      0.85      0.79      3390
```

8.0.2 Confidence Interval

[138]:
```
import math
n_lg = len(lg_pred)
error_lg + 1.96 * math.sqrt((error_lg * (1 - error_lg)) / n_lg)
```

[138]: 0.1573934793191731

[139]:
```
error_lg - 1.96 * math.sqrt((error_lg * (1 - error_lg)) / n_lg)
```

[139]: 0.11147444520912878

[140]:
```
import numpy as np, scipy.stats as st
st.t.interval(0.95, len(y_test)-1, loc=np.mean(y_test), scale=st.sem(y_test))
```

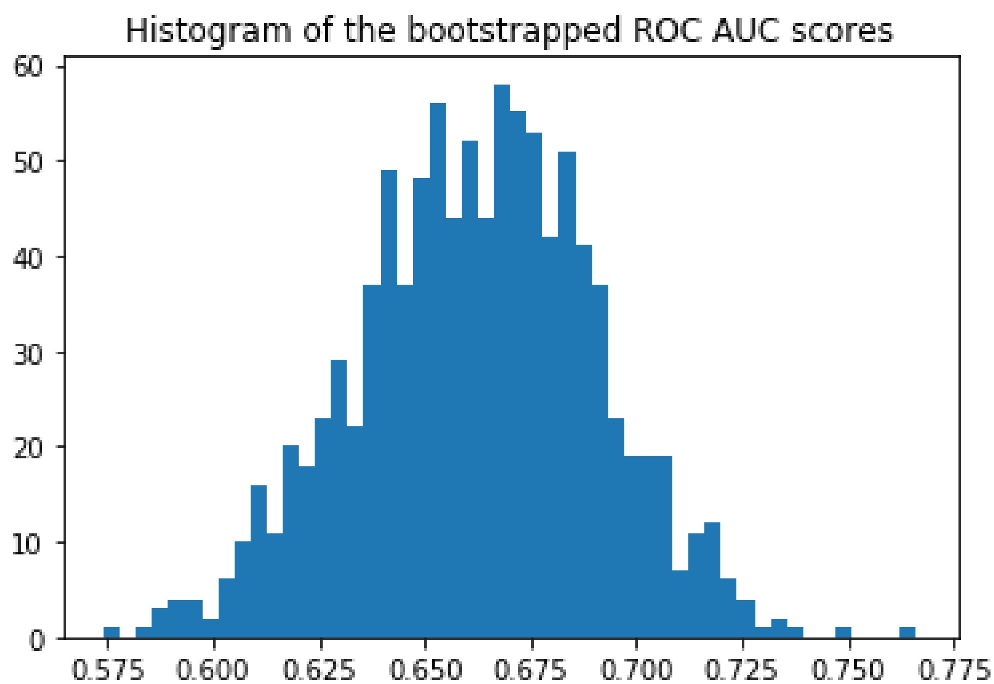
[140]: (array([0.11252275]), array([0.15870366]))

[141]:
```
y_test_v = y_test.values
```

[142]:
```
#Calculated the Confidence Interval by bootstrapping
import numpy as np
from scipy.stats import sem
from sklearn.metrics import roc_auc_score

y_pred = lg_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
```

```python
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.6629
```




```
Confidence interval for the score: [0.6148 - 0.7075]
```

[143]:

```python
#Transplanted the pROC package from R into Python for CI computation
import numpy as np
import scipy.stats
from scipy import stats

y_test_v2 = y_test_v.reshape((848,))
```

```python
# AUC comparison adapted from
# https://github.com/Netflix/vmaf/
def compute_midrank(x):
    """Computes midranks.
    Args:
       x - a 1D numpy array
    Returns:
       array of midranks
    """
    J = np.argsort(x)
    Z = x[J]
    N = len(x)
    T = np.zeros(N, dtype=np.float)
    i = 0
    while i < N:
        j = i
        while j < N and Z[j] == Z[i]:
            j += 1
        T[i:j] = 0.5*(i + j - 1)
        i = j
    T2 = np.empty(N, dtype=np.float)
    # Note(kazeevn) +1 is due to Python using 0-based indexing
    # instead of 1-based in the AUC formula in the paper
    T2[J] = T + 1
    return T2


def compute_midrank_weight(x, sample_weight):
    """Computes midranks.
    Args:
       x - a 1D numpy array
    Returns:
       array of midranks
    """
    J = np.argsort(x)
    Z = x[J]
    cumulative_weight = np.cumsum(sample_weight[J])
    N = len(x)
    T = np.zeros(N, dtype=np.float)
    i = 0
    while i < N:
        j = i
        while j < N and Z[j] == Z[i]:
            j += 1
        T[i:j] = cumulative_weight[i:j].mean()
        i = j
    T2 = np.empty(N, dtype=np.float)
```

```python
    T2[J] = T
    return T2


def fastDeLong(predictions_sorted_transposed, label_1_count, sample_weight):
    if sample_weight is None:
        return fastDeLong_no_weights(predictions_sorted_transposed,␣
↪label_1_count)
    else:
        return fastDeLong_weights(predictions_sorted_transposed, label_1_count,␣
↪sample_weight)


def fastDeLong_weights(predictions_sorted_transposed, label_1_count,␣
↪sample_weight):
    """
    The fast version of DeLong's method for computing the covariance of
    unadjusted AUC.
    Args:
       predictions_sorted_transposed: a 2D numpy.array[n_classifiers,␣
↪n_examples]
          sorted such as the examples with label "1" are first
    Returns:
       (AUC value, DeLong covariance)
    Reference:
     @article{sun2014fast,
       title={Fast Implementation of DeLong's Algorithm for
              Comparing the Areas Under Correlated Receiver Oerating␣
↪Characteristic Curves},
       author={Xu Sun and Weichao Xu},
       journal={IEEE Signal Processing Letters},
       volume={21},
       number={11},
       pages={1389--1393},
       year={2014},
       publisher={IEEE}
     }
    """
    # Short variables are named as they are in the paper
    m = label_1_count
    n = predictions_sorted_transposed.shape[1] - m
    positive_examples = predictions_sorted_transposed[:, :m]
    negative_examples = predictions_sorted_transposed[:, m:]
    k = predictions_sorted_transposed.shape[0]

    tx = np.empty([k, m], dtype=np.float)
    ty = np.empty([k, n], dtype=np.float)
```

```
    tz = np.empty([k, m + n], dtype=np.float)
    for r in range(k):
        tx[r, :] = compute_midrank_weight(positive_examples[r, :],␣
↪sample_weight[:m])
        ty[r, :] = compute_midrank_weight(negative_examples[r, :],␣
↪sample_weight[m:])
        tz[r, :] = compute_midrank_weight(predictions_sorted_transposed[r, :],␣
↪sample_weight)
    total_positive_weights = sample_weight[:m].sum()
    total_negative_weights = sample_weight[m:].sum()
    pair_weights = np.dot(sample_weight[:m, np.newaxis], sample_weight[np.
↪newaxis, m:])
    total_pair_weights = pair_weights.sum()
    aucs = (sample_weight[:m]*(tz[:, :m] - tx)).sum(axis=1) / total_pair_weights
    v01 = (tz[:, :m] - tx[:, :]) / total_negative_weights
    v10 = 1. - (tz[:, m:] - ty[:, :]) / total_positive_weights
    sx = np.cov(v01)
    sy = np.cov(v10)
    delongcov = sx / m + sy / n
    return aucs, delongcov


def fastDeLong_no_weights(predictions_sorted_transposed, label_1_count):
    """
    The fast version of DeLong's method for computing the covariance of
    unadjusted AUC.
    Args:
       predictions_sorted_transposed: a 2D numpy.array[n_classifiers,␣
↪n_examples]
          sorted such as the examples with label "1" are first
    Returns:
       (AUC value, DeLong covariance)
    Reference:
     @article{sun2014fast,
       title={Fast Implementation of DeLong's Algorithm for
              Comparing the Areas Under Correlated Receiver Oerating
              Characteristic Curves},
       author={Xu Sun and Weichao Xu},
       journal={IEEE Signal Processing Letters},
       volume={21},
       number={11},
       pages={1389--1393},
       year={2014},
       publisher={IEEE}
     }
    """
    # Short variables are named as they are in the paper
```

```python
    m = label_1_count
    n = predictions_sorted_transposed.shape[1] - m
    positive_examples = predictions_sorted_transposed[:, :m]
    negative_examples = predictions_sorted_transposed[:, m:]
    k = predictions_sorted_transposed.shape[0]

    tx = np.empty([k, m], dtype=np.float)
    ty = np.empty([k, n], dtype=np.float)
    tz = np.empty([k, m + n], dtype=np.float)
    for r in range(k):
        tx[r, :] = compute_midrank(positive_examples[r, :])
        ty[r, :] = compute_midrank(negative_examples[r, :])
        tz[r, :] = compute_midrank(predictions_sorted_transposed[r, :])
    aucs = tz[:, :m].sum(axis=1) / m / n - float(m + 1.0) / 2.0 / n
    v01 = (tz[:, :m] - tx[:, :]) / n
    v10 = 1.0 - (tz[:, m:] - ty[:, :]) / m
    sx = np.cov(v01)
    sy = np.cov(v10)
    delongcov = sx / m + sy / n
    return aucs, delongcov


def calc_pvalue(aucs, sigma):
    """Computes log(10) of p-values.
    Args:
       aucs: 1D array of AUCs
       sigma: AUC DeLong covariances
    Returns:
       log10(pvalue)
    """
    l = np.array([[1, -1]])
    z = np.abs(np.diff(aucs)) / np.sqrt(np.dot(np.dot(l, sigma), l.T))
    return np.log10(2) + scipy.stats.norm.logsf(z, loc=0, scale=1) / np.log(10)


def compute_ground_truth_statistics(ground_truth, sample_weight):
    assert np.array_equal(np.unique(ground_truth), [0, 1])
    order = (-ground_truth).argsort()
    label_1_count = int(ground_truth.sum())
    if sample_weight is None:
        ordered_sample_weight = None
    else:
        ordered_sample_weight = sample_weight[order]

    return order, label_1_count, ordered_sample_weight
```

```
def delong_roc_variance(ground_truth, predictions, sample_weight=None):
    """
    Computes ROC AUC variance for a single set of predictions
    Args:
       ground_truth: np.array of 0 and 1
       predictions: np.array of floats of the probability of being class 1
    """
    order, label_1_count, ordered_sample_weight =␣
 ↪compute_ground_truth_statistics(
        ground_truth, sample_weight)
    predictions_sorted_transposed = predictions[np.newaxis, order]
    aucs, delongcov = fastDeLong(predictions_sorted_transposed, label_1_count,␣
 ↪ordered_sample_weight)
    assert len(aucs) == 1, "There is a bug in the code, please forward this to␣
 ↪the developers"
    return aucs[0], delongcov


alpha = .95
y_pred = lg_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.6629100183878047
AUC COV: 0.0008011363518342227
95% AUC CI: [0.60743451 0.71838553]
```

[ ]:

9 Decision Tree Classification

[144]:
```
from sklearn.tree import DecisionTreeClassifier
seed(0)
# skf = StratifiedKFold(n_splits=5)
params1 = {'min_samples_split' : [2,3,4],
           'min_impurity_split': [1e-9, 1e-7, 1e-5],
           'min_samples_leaf': [1,2]}
dt = DecisionTreeClassifier(random_state=0)
dt = GridSearchCV(dt, cv=5, param_grid=params1, scoring = 'roc_auc',refit =␣
 ↪True,
                  n_jobs=-1, verbose = 5, return_train_score=True)

dt.fit(X_train, y_train)
# dt.cv_results_
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    2.4s
[Parallel(n_jobs=-1)]: Done  86 out of  90 | elapsed:    2.6s remaining:    0.1s
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:    2.7s finished
/usr/local/lib/python3.7/site-packages/sklearn/tree/tree.py:297:
DeprecationWarning: The min_impurity_split parameter is deprecated. Its default
value will change from 1e-7 to 0 in version 0.23, and it will be removed in
0.25. Use the min_impurity_decrease parameter instead.
  DeprecationWarning)
```

[144]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=0,
                                              splitter='best'),
             iid='warn', n_jobs=-1,
             param_grid={'min_impurity_split': [1e-09, 1e-07, 1e-05],
                         'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 3, 4]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

[145]:
```
dt.best_estimator_
```

[145]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=1e-09,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

[146]:
```
dt_pred = dt.best_estimator_.predict(X_test)
dt_prob = dt.best_estimator_.predict_proba(X_test)
dt_prob
```

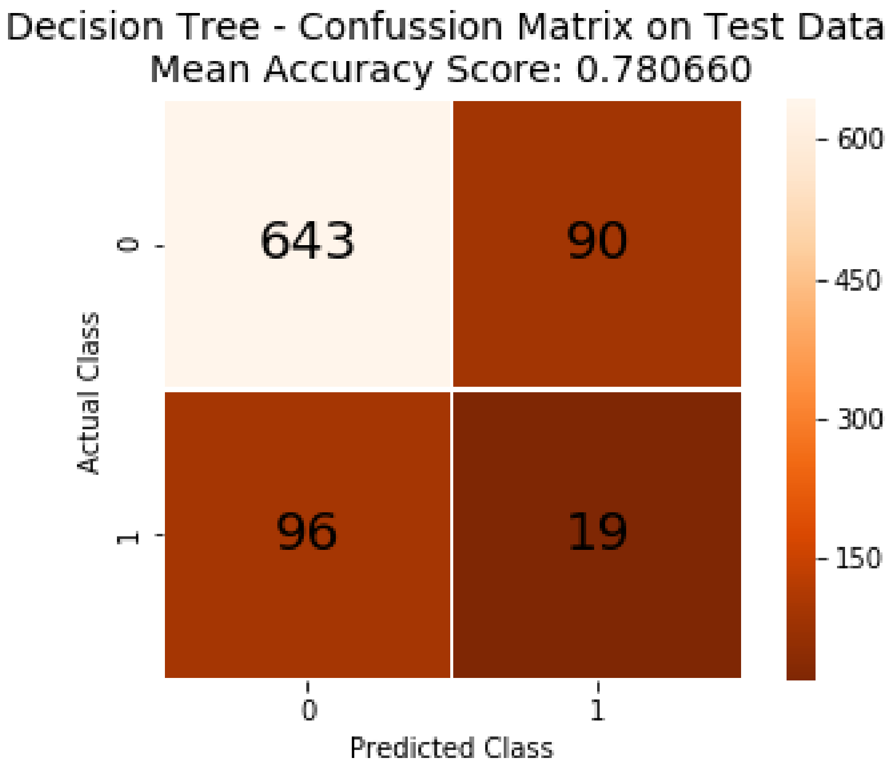
[146]:
```
array([[1. , 0. ],
       [1. , 0. ],
       [0.5, 0.5],
       …,
       [1. , 0. ],
       [1. , 0. ],
       [1. , 0. ]])
```

[147]:
```
dt_matrix = metrics.confusion_matrix(y_test, dt_pred)
dt_matrix
```

[147]:
```
array([[643,  90],
       [ 96,  19]])
```

[148]:
```
dt_test = dt.best_estimator_.score(X_test, y_test)

dt_matrix = metrics.confusion_matrix(y_test, dt_pred)

dt_cm = pd.DataFrame(dt_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (dt_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

dt_title = 'Decision Tree - Confussion Matrix on Test Data \nMean Accuracy␣
 ↪Score: {0:2f}'.format(dt_test)
plt.title(dt_title, size = 14)
```

```python
# plt.figure(figsize=(16, 26))
plt.show;
```



[149]:
```python
print("", classification_report(y_test, dt_pred, target_names=target_names))
```

```
                   precision    recall  f1-score   support

No risk in 10 years       0.87      0.88      0.87       733
  Risky in 10 years       0.17      0.17      0.17       115

           accuracy                           0.78       848
          macro avg       0.52      0.52      0.52       848
       weighted avg       0.78      0.78      0.78       848
```

[150]:
```python
acc_dt = accuracy_score(y_test, dt_pred)
print("Decision Tree accuracy:", acc_dt)
```

```
Decision Tree accuracy: 0.7806603773584906
```

[151]:
```python
dt_probs = dt.best_estimator_.predict_proba(X_test)[:,1]
print(roc_auc_score(y_test, dt_probs))
```

0.540453170413429

[152]:
```
error_dt = 1-acc_dt
error_dt
```

[152]: 0.2193396226415094

9.0.1 For training set:

[153]:
```
dt_pred_tr = dt.best_estimator_.predict(X_train)
dt_prob_tr = dt.best_estimator_.predict_proba(X_train)
dt_prob_tr
```

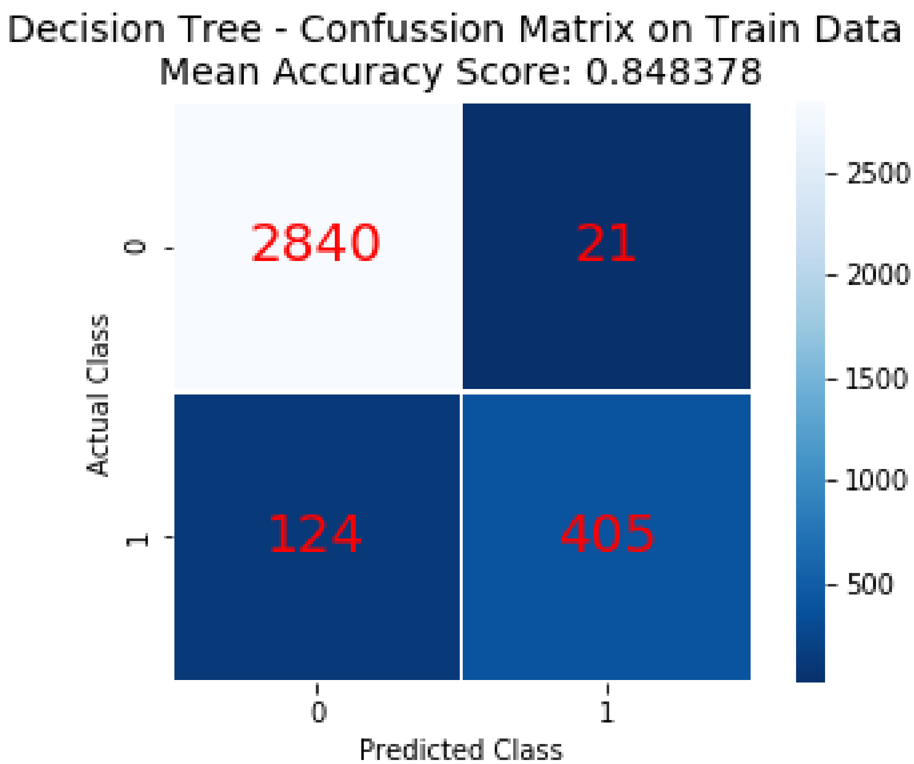
[153]:
```
array([[1. , 0. ],
       [0.5, 0.5],
       [1. , 0. ],
       …,
       [1. , 0. ],
       [1. , 0. ],
       [1. , 0. ]])
```

[154]:
```
dt_matrix_tr = metrics.confusion_matrix(y_train, dt_pred_tr)
dt_matrix_tr
```

[154]:
```
array([[2840,   21],
       [ 124,  405]])
```

[155]:
```
dt_train = dt.best_estimator_.score(X_train, y_train)

dt_matrix = metrics.confusion_matrix(y_train, dt_pred_tr)

dt_cm_tr = pd.DataFrame(dt_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (dt_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
↪5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

dt_title = 'Decision Tree - Confussion Matrix on Train Data \nMean Accuracy␣
↪Score: {0:2f}'.format(lg_train)
```

```
plt.title(dt_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



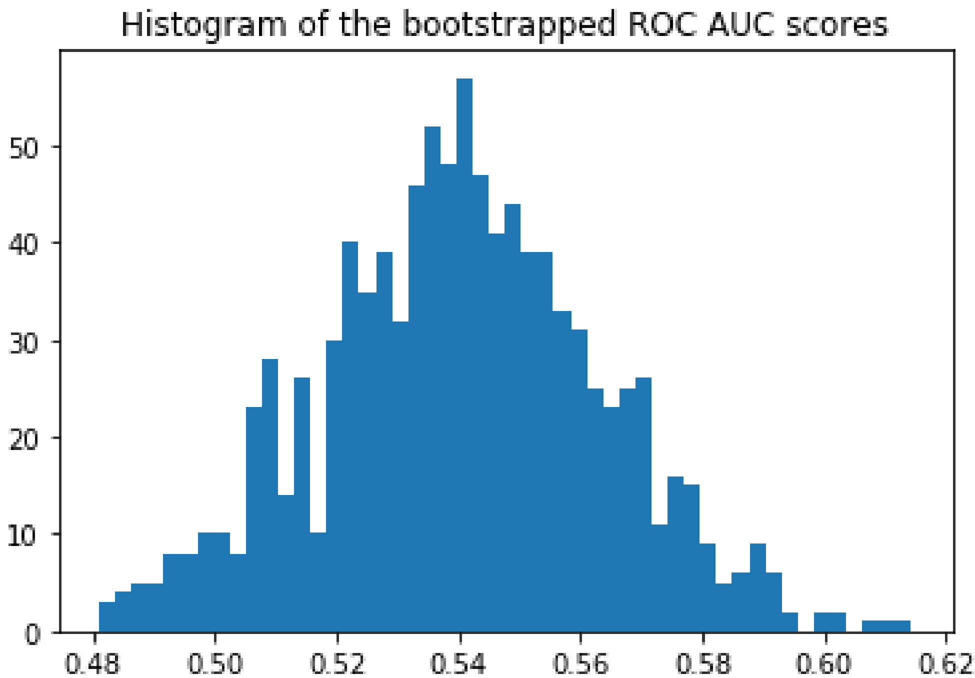
[156]:
```
print("", classification_report(y_train, dt_pred_tr, target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years       0.96      0.99      0.98      2861
  Risky in 10 years       0.95      0.77      0.85       529

           accuracy                           0.96      3390
          macro avg       0.95      0.88      0.91      3390
       weighted avg       0.96      0.96      0.96      3390
```

[157]:
```
#Calculated the Confidence Interval by bootstrapping
import numpy as np
from scipy.stats import sem
from sklearn.metrics import roc_auc_score

y_pred = dt_probs
```

```python
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
confidence_lower, confidence_upper))
```

```
Original ROC area: 0.5405
```




```
Confidence interval for the score: [0.5016 - 0.579]
```

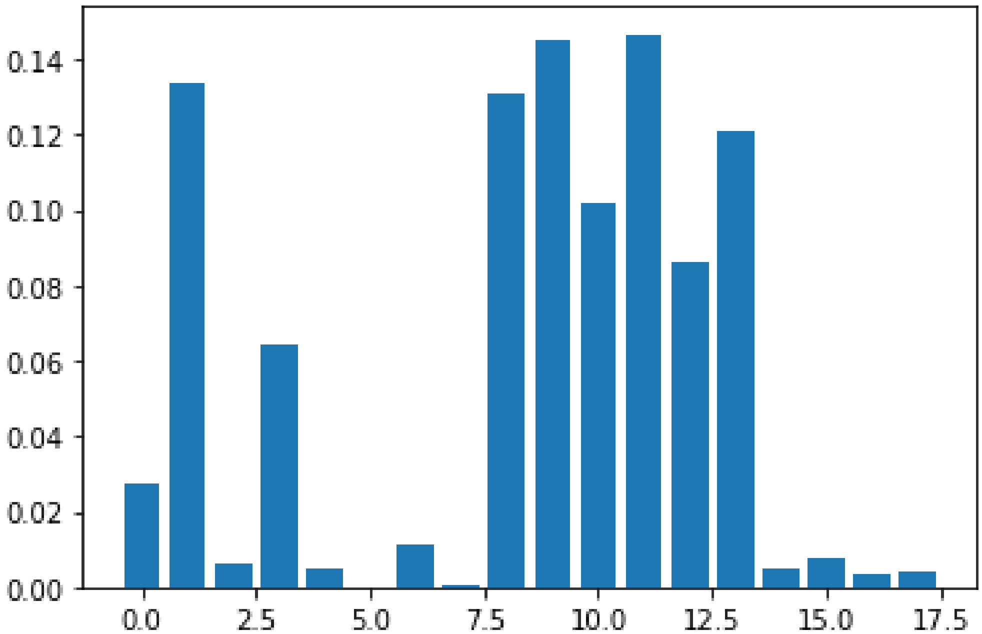
[158]:
```
alpha = .95
y_pred = dt_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.540453170413429
```

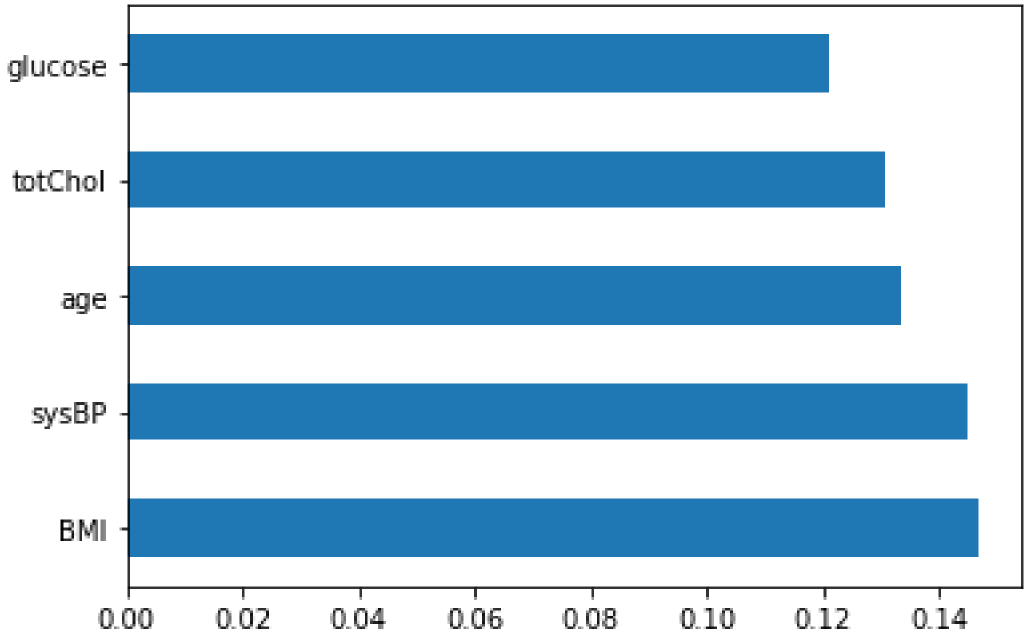
```
AUC COV: 0.000522200198428501
95% AUC CI: [0.49566466 0.58524168]
```

[159]:
```
from matplotlib import pyplot
print(dt.best_estimator_.feature_importances_)
pyplot.bar(range(len(dt.best_estimator_.feature_importances_)), dt.
 ↪best_estimator_.feature_importances_)
pyplot.show()
```

```
[0.02723776 0.13369716 0.00650036 0.06403178 0.00449774 0.
 0.01142852 0.00076323 0.13076926 0.14491392 0.10188886 0.1467003
 0.08625045 0.12118579 0.00509344 0.007737   0.00347245 0.00383197]
```



[160]:
```
feat_importances_dt = pd.Series(dt.best_estimator_.feature_importances_,␣
 ↪index=X_train.columns)
feat_importances_dt.nlargest(5).plot(kind='barh')
pyplot.show()
```



[ ]:

10 Random Forest

[161]:
```
from sklearn.ensemble import RandomForestClassifier
seed(42)
params2 = {'n_estimators' : [300,500,700],
           'min_samples_split': [2,3,4],
           'min_samples_leaf':[1,2,3]}

rf = RandomForestClassifier(random_state=42)
rf = GridSearchCV(rf, cv=5, param_grid=params2, scoring = 'roc_auc',refit =␣
 ↪True,
                  n_jobs=-1, verbose = 5, return_train_score=True)

rf.fit(X_train, y_train)
rf.cv_results_
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:   17.4s
[Parallel(n_jobs=-1)]: Done 135 out of 135 | elapsed:   44.0s finished
/usr/local/lib/python3.7/site-packages/sklearn/model_selection/_search.py:715:
DataConversionWarning: A column-vector y was passed when a 1d array was
```

```
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  self.best_estimator_.fit(X, y, **fit_params)
```

[161]:
```
{'mean_fit_time': array([1.56902456, 2.63530879, 3.86090255, 1.62363539,
3.07899728,
        4.5594264 , 1.88598852, 3.36880651, 4.92601862, 2.00065007,
        3.35344806, 4.78613105, 2.0652626 , 3.37529016, 4.67701683,
        1.9279788 , 3.5603951 , 4.86630421, 1.94122424, 3.08242664,
        4.27251024, 1.82467918, 2.94786892, 4.10665116, 1.76197138,
        2.91285195, 3.27225852]),
 'std_fit_time': array([0.00993963, 0.01503649, 0.12125068, 0.02248349,
0.1013447 ,
        0.07931839, 0.01757026, 0.07952316, 0.08178043, 0.06754291,
        0.08187914, 0.09213244, 0.04338397, 0.04452007, 0.07111751,
        0.03650741, 0.04385904, 0.07062032, 0.03592434, 0.03941081,
        0.02335248, 0.02491455, 0.02385428, 0.03452894, 0.02326861,
        0.04580176, 0.18416429]),
 'mean_score_time': array([0.07234316, 0.12080102, 0.21098709, 0.08009195,
0.16944971,
        0.19889932, 0.08427806, 0.16187148, 0.21823301, 0.10175943,
        0.15683727, 0.21625853, 0.10006905, 0.15936036, 0.24642115,
        0.09600816, 0.15206442, 0.197365  , 0.09309855, 0.15123014,
        0.19277196, 0.08896551, 0.14010129, 0.19629011, 0.08588076,
        0.12486124, 0.11437931]),
 'std_score_time': array([0.0017688 , 0.00238966, 0.02759725, 0.00499177,
0.01165562,
        0.01420825, 0.00156668, 0.01637713, 0.02232754, 0.01808884,
        0.00980395, 0.01702912, 0.00609053, 0.01224111, 0.03013758,
        0.00883677, 0.00931859, 0.00400189, 0.00265305, 0.01021875,
        0.00285883, 0.00431484, 0.00254806, 0.00687279, 0.00204028,
        0.00960641, 0.00269504]),
 'param_min_samples_leaf': masked_array(data=[1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
2, 2, 2, 2, 2, 2, 2,
                    3, 3, 3, 3, 3, 3, 3, 3, 3],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[2, 2, 2, 3, 3, 3, 4, 4, 4, 2, 2,
2, 3, 3, 3, 4, 4, 4,
                    2, 2, 2, 3, 3, 3, 4, 4, 4],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
```

```
                    False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[300, 500, 700, 300, 500, 700, 300,
500, 700, 300, 500,
                    700, 300, 500, 700, 300, 500, 700, 300, 500, 700, 300,
                    500, 700, 300, 500, 700],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'min_samples_leaf': 1,
   'min_samples_split': 2,
   'n_estimators': 300},
  {'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500},
  {'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 700},
  {'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 300},
  {'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 500},
  {'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 700},
  {'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 300},
  {'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 500},
  {'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 700},
  {'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 300},
  {'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 500},
  {'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 700},
  {'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 300},
  {'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 500},
  {'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 700},
  {'min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 300},
  {'min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 500},
  {'min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 700},
  {'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 300},
  {'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 500},
  {'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 700},
  {'min_samples_leaf': 3, 'min_samples_split': 3, 'n_estimators': 300},
  {'min_samples_leaf': 3, 'min_samples_split': 3, 'n_estimators': 500},
  {'min_samples_leaf': 3, 'min_samples_split': 3, 'n_estimators': 700},
  {'min_samples_leaf': 3, 'min_samples_split': 4, 'n_estimators': 300},
  {'min_samples_leaf': 3, 'min_samples_split': 4, 'n_estimators': 500},
  {'min_samples_leaf': 3, 'min_samples_split': 4, 'n_estimators': 700}],
 'split0_test_score': array([0.71028845, 0.708428  , 0.70856795, 0.70609009,
0.70941585,
        0.71085647, 0.7130462 , 0.71232178, 0.71452797, 0.71233824,
        0.71136685, 0.71691528, 0.71233824, 0.71136685, 0.71691528,
        0.71233824, 0.71136685, 0.71691528, 0.72763344, 0.72662913,
```

```
       0.72475221, 0.72763344, 0.72662913, 0.72475221, 0.72763344,
       0.72662913, 0.72475221]),
 'split1_test_score': array([0.72377622, 0.72509566, 0.72393291, 0.72784998,
0.72930961,
       0.7270913 , 0.72271243, 0.72511215, 0.7294333 , 0.72440296,
       0.72493073, 0.72712429, 0.72440296, 0.72493073, 0.72712429,
       0.72440296, 0.72493073, 0.72712429, 0.73944452, 0.73919712,
       0.7369046 , 0.73944452, 0.73919712, 0.7369046 , 0.73944452,
       0.73919712, 0.7369046 ]),
 'split2_test_score': array([0.68053998, 0.68255212, 0.68275003, 0.68236245,
0.68206558,
       0.68006168, 0.68947915, 0.68712891, 0.68775564, 0.68572701,
       0.68735981, 0.68613933, 0.68572701, 0.68735981, 0.68613933,
       0.68572701, 0.68735981, 0.68613933, 0.68460549, 0.68516625,
       0.6843416 , 0.68460549, 0.68516625, 0.6843416 , 0.68460549,
       0.68516625, 0.6843416 ]),
 'split3_test_score': array([0.67866803, 0.67991325, 0.68188415, 0.6848199 ,
0.68344274,
       0.68228823, 0.67810727, 0.68064718, 0.68300567, 0.68782161,
       0.683418  , 0.68262634, 0.68782161, 0.683418  , 0.68262634,
       0.68782161, 0.683418  , 0.68262634, 0.68650218, 0.68813498,
       0.68945441, 0.68650218, 0.68813498, 0.68945441, 0.68650218,
       0.68813498, 0.68945441]),
 'split4_test_score': array([0.65117383, 0.65018315, 0.65233933, 0.65799201,
0.65693473,
       0.65762571, 0.63616384, 0.64408924, 0.64535465, 0.65827506,
       0.65804196, 0.65965701, 0.65827506, 0.65804196, 0.65965701,
       0.65827506, 0.65804196, 0.65965701, 0.65461205, 0.65659341,
       0.65909091, 0.65461205, 0.65659341, 0.65909091, 0.65461205,
       0.65659341, 0.65909091]),
 'mean_test_score': array([0.68890674, 0.68925162, 0.68991146, 0.69183707,
0.69224918,
       0.69160038, 0.68792446, 0.68987998, 0.69203585, 0.69372892,
       0.6930392 , 0.69450934, 0.69372892, 0.6930392 , 0.69450934,
       0.69372892, 0.6930392 , 0.69450934, 0.69858108, 0.69916484,
       0.69892812, 0.69858108, 0.69916484, 0.69892812, 0.69858108,
       0.69916484, 0.69892812]),
 'std_test_score': array([0.02557436, 0.02572646, 0.02462252, 0.02359316,
0.02488025,
       0.0245087 , 0.030385  , 0.02804352, 0.02891711, 0.02298592,
       0.02324346, 0.02445673, 0.02298592, 0.02324346, 0.02445673,
       0.02298592, 0.02324346, 0.02445673, 0.03094106, 0.02995081,
       0.02827459, 0.03094106, 0.02995081, 0.02827459, 0.03094106,
       0.02995081, 0.02827459]),
 'rank_test_score': array([26, 25, 23, 21, 19, 22, 27, 24, 20, 13, 16, 10, 13,
16, 10, 13, 16,
       10,  7,  1,  4,  7,  1,  4,  7,  1,  4], dtype=int32),
```

```
 'split0_train_score': array([1.        , 1.        , 1.        , 1.        , 1.
,
       1.        , 1.        , 1.        , 1.        , 0.99999173,
       0.9999938 , 0.99999587, 0.99999173, 0.9999938 , 0.99999587,
       0.99999173, 0.9999938 , 0.99999587, 0.99789734, 0.9980885 ,
       0.99808953, 0.99789734, 0.9980885 , 0.99808953, 0.99789734,
       0.9980885 , 0.99808953]),
 'split1_train_score': array([1.        , 1.        , 1.        , 1.        , 1.
,
       1.        , 1.        , 1.        , 1.        , 0.99997418,
       0.99998038, 0.99998967, 0.99997418, 0.99998038, 0.99998967,
       0.99997418, 0.99998038, 0.99998967, 0.99786728, 0.998016  ,
       0.99811308, 0.99786728, 0.998016  , 0.99811308, 0.99786728,
       0.998016  , 0.99811308]),
 'split2_train_score': array([1.        , 1.        , 1.        , 1.        , 1.
,
       1.        , 1.        , 1.        , 1.        , 0.99999484,
       0.9999969 , 0.99999897, 0.99999484, 0.9999969 , 0.99999897,
       0.99999484, 0.9999969 , 0.99999897, 0.99855512, 0.99876168,
       0.99878337, 0.99855512, 0.99876168, 0.99878337, 0.99855512,
       0.99876168, 0.99878337]),
 'split3_train_score': array([1.        , 1.        , 1.        , 1.        , 1.
,
       1.        , 1.        , 1.        , 1.        , 0.99995972,
       0.99996798, 0.99996695, 0.99995972, 0.99996798, 0.99996695,
       0.99995972, 0.99996798, 0.99996695, 0.99727342, 0.9974521 ,
       0.9974459 , 0.99727342, 0.9974521 , 0.9974459 , 0.99727342,
       0.9974521 , 0.9974459 ]),
 'split4_train_score': array([1.        , 1.        , 1.        , 1.        , 1.
,
       1.        , 1.        , 1.        , 1.        , 0.99995363,
       0.99996188, 0.99997321, 0.99995363, 0.99996188, 0.99997321,
       0.99995363, 0.99996188, 0.99997321, 0.9976446 , 0.99784964,
       0.99776824, 0.9976446 , 0.99784964, 0.99776824, 0.9976446 ,
       0.99784964, 0.99776824]),
 'mean_train_score': array([1.        , 1.        , 1.        , 1.        , 1.
,
       1.        , 1.        , 1.        , 1.        , 0.99997482,
       0.99998019, 0.99998493, 0.99997482, 0.99998019, 0.99998493,
       0.99997482, 0.99998019, 0.99998493, 0.99784755, 0.99803358,
       0.99804002, 0.99784755, 0.99803358, 0.99804002, 0.99784755,
       0.99803358, 0.99804002]),
 'std_train_score': array([4.96506831e-17, 4.96506831e-17, 0.00000000e+00,
0.00000000e+00,
       0.00000000e+00, 4.96506831e-17, 0.00000000e+00, 7.02166694e-17,
       4.96506831e-17, 1.65164566e-05, 1.37763331e-05, 1.26471379e-05,
       1.65164566e-05, 1.37763331e-05, 1.26471379e-05, 1.65164566e-05,
```

```
        1.37763331e-05, 1.26471379e-05, 4.18171131e-04, 4.25537339e-04,
        4.44341692e-04, 4.18171131e-04, 4.25537339e-04, 4.44341692e-04,
        4.18171131e-04, 4.25537339e-04, 4.44341692e-04])}
```

[162]:
```
rf.best_estimator_
```

[162]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=3, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

[163]:
```
rf_pred = rf.best_estimator_.predict(X_test)
rf_prob = rf.best_estimator_.predict_proba(X_test)
rf_prob
```

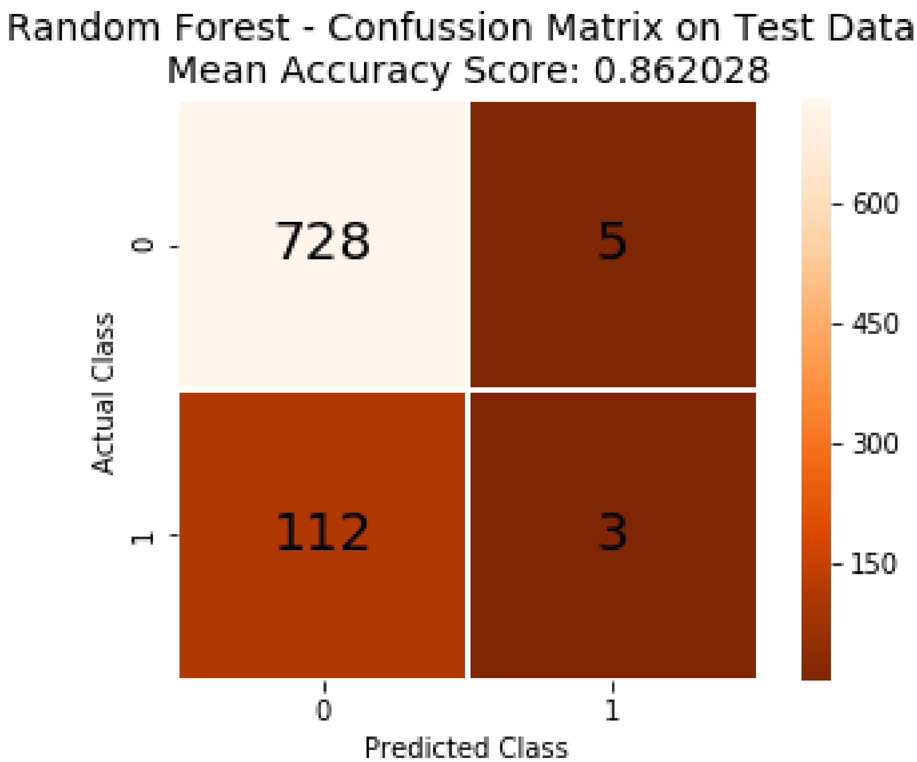
[163]:
```
array([[0.84294986, 0.15705014],
       [0.95708803, 0.04291197],
       [0.91832239, 0.08167761],
       …,
       [0.82137516, 0.17862484],
       [0.63761168, 0.36238832],
       [0.93518498, 0.06481502]])
```

[164]:
```
rf_matrix = metrics.confusion_matrix(y_test, rf_pred)
rf_matrix
```

[164]:
```
array([[728,   5],
       [112,   3]])
```

[165]:
```
rf_test = rf.best_estimator_.score(X_test, y_test)

rf_matrix = metrics.confusion_matrix(y_test, rf_pred)

rf_cm = pd.DataFrame(rf_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (rf_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
```

```
plt.ylabel('Actual Class')

rf_title = 'Random Forest - Confussion Matrix on Test Data \nMean Accuracy␣
 ↪Score: {0:2f}'.format(rf_test)
plt.title(rf_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



[166]:
```
print("", classification_report(y_test, rf_pred, target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years       0.87      0.99      0.93       733
  Risky in 10 years       0.38      0.03      0.05       115

           accuracy                           0.86       848
          macro avg       0.62      0.51      0.49       848
       weighted avg       0.80      0.86      0.81       848
```

[167]:
```
acc_rf = accuracy_score(y_test, rf_pred)
print("Random Forest accuracy:", acc_rf)
```

```
Random Forest accuracy: 0.8620283018867925
```

[168]:
```python
rf_probs = rf.best_estimator_.predict_proba(X_test)[:,1]
print(roc_auc_score(y_test, rf_probs))
```

```
0.7125333649682662
```

[169]:
```python
error_rf = 1-acc_rf
error_rf
```

[169]:
```
0.13797169811320753
```

10.0.1 For training set:

[170]:
```python
rf_pred_tr = rf.best_estimator_.predict(X_train)
rf_prob_tr = rf.best_estimator_.predict_proba(X_train)
rf_prob_tr
```

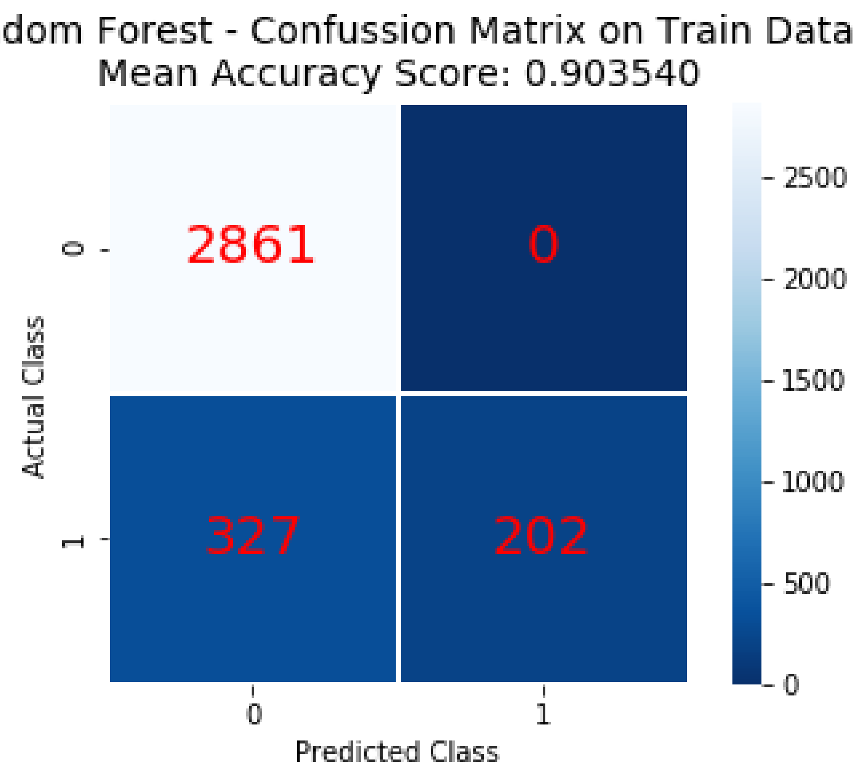
[170]:
```
array([[0.86035111, 0.13964889],
       [0.67175127, 0.32824873],
       [0.94870577, 0.05129423],
       …,
       [0.7441357 , 0.2558643 ],
       [0.97400231, 0.02599769],
       [0.94729549, 0.05270451]])
```

[171]:
```python
rf_matrix_tr = metrics.confusion_matrix(y_train, rf_pred_tr)
rf_matrix_tr
```

[171]:
```
array([[2861,    0],
       [ 327,  202]])
```

[172]:
```python
rf_train = rf.best_estimator_.score(X_train, y_train)

rf_matrix = metrics.confusion_matrix(y_train, rf_pred_tr)

rf_cm_tr = pd.DataFrame(rf_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (rf_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
 ↪5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

```
plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

rf_title = 'Random Forest - Confussion Matrix on Train Data \nMean Accuracy␣
↪Score: {0:2f}'.format(rf_train)
plt.title(rf_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



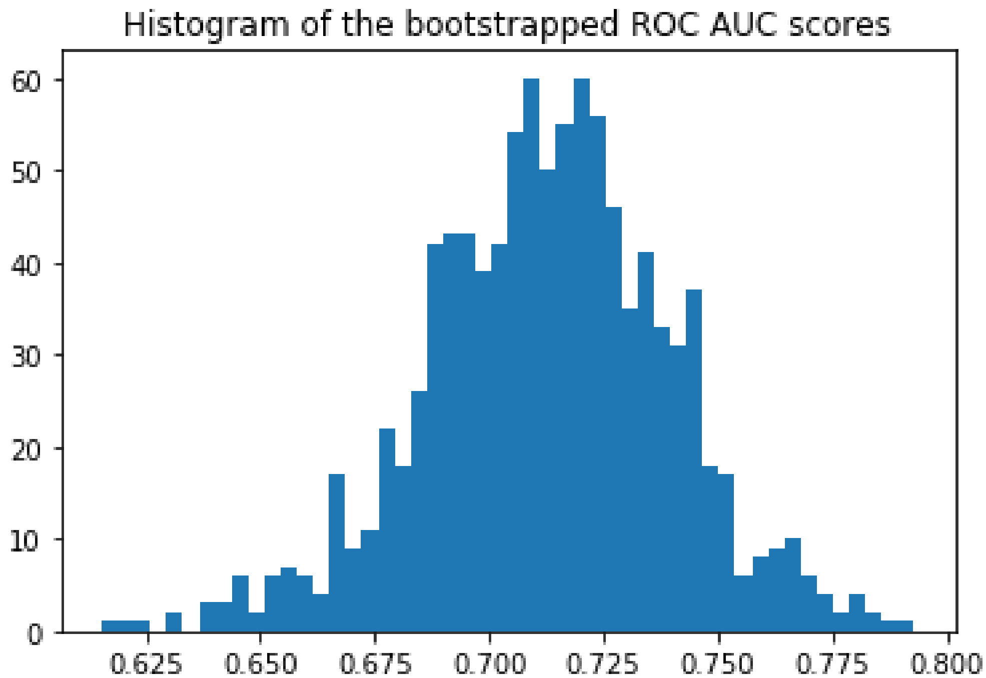
[173]:
```
print("", classification_report(y_train, rf_pred_tr, target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years       0.90      1.00      0.95      2861
  Risky in 10 years       1.00      0.38      0.55       529

           accuracy                           0.90      3390
          macro avg       0.95      0.69      0.75      3390
       weighted avg       0.91      0.90      0.88      3390
```

[174]:

```python
y_pred = rf_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.7125
```




```
Confidence interval for the score: [0.6667 - 0.7566]
```

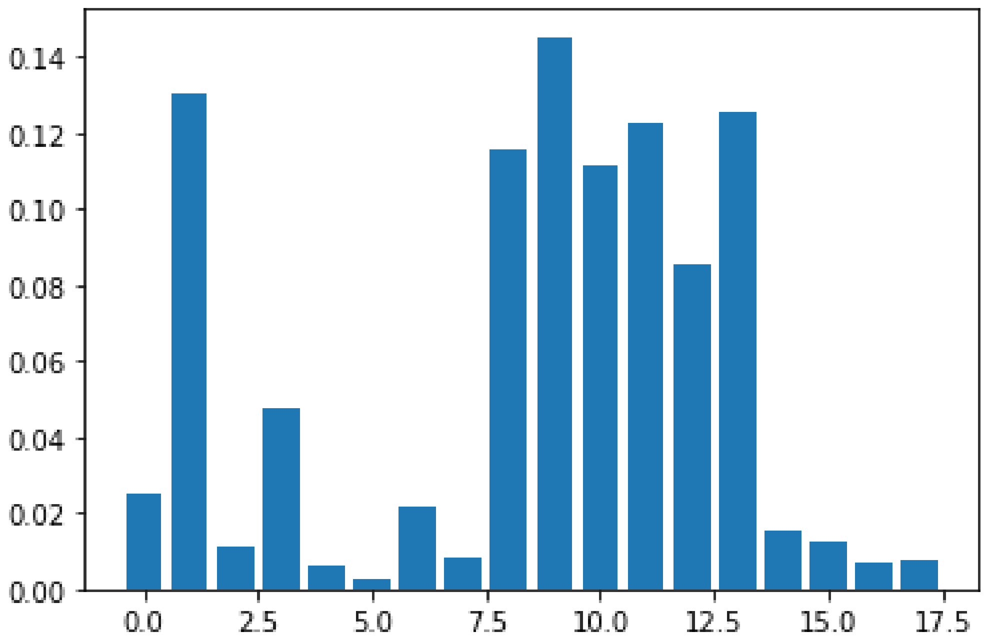
[175]:
```python
alpha = .95
y_pred = rf_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.7125333649682662
```

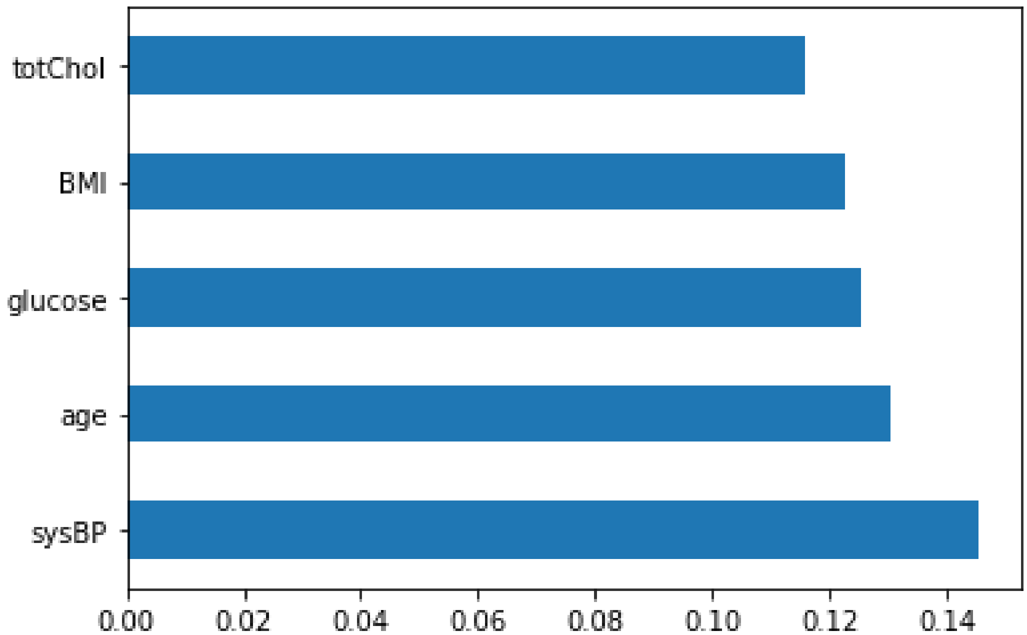
```
AUC COV: 0.0007198901751595977
95% AUC CI: [0.65994602 0.76512071]
```

[176]:
```
print(rf.best_estimator_.feature_importances_)
pyplot.bar(range(len(rf.best_estimator_.feature_importances_)), rf.
 ↪best_estimator_.feature_importances_)
pyplot.show()
```

```
[0.02496806 0.13021948 0.01110899 0.04771876 0.00593456 0.00256256
 0.0214176  0.0080941  0.11575566 0.14527444 0.11145112 0.12273689
 0.08531736 0.12526185 0.01562058 0.0123477  0.00693725 0.00727305]
```



[177]:
```
feat_importances_rf = pd.Series(rf.best_estimator_.feature_importances_,␣
 ↪index=X_train.columns)
feat_importances_rf.nlargest(5).plot(kind='barh')
pyplot.show()
```



[ ]:

```

```

11 KNN Classifier

[178]:

```
# df.iloc[13:22,:]
```

[179]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler = scaler.fit(X_train)
X_trains = scaler.transform(X_train)
X_tests = scaler.transform(X_test)
```

[180]:

```
# from random import seed
seed(1)
params3 = {'n_neighbors' : [3,5,7],
           'leaf_size': [20,30,40]}
knn = KNN()
knn = GridSearchCV(knn, cv=5, param_grid=params3, scoring = 'roc_auc',refit =␣
 ↪True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

knn.fit(X_trains, y_train)
knn.cv_results_
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  32 out of  45 | elapsed:    3.3s remaining:    1.3s
[Parallel(n_jobs=-1)]: Done  42 out of  45 | elapsed:    3.6s remaining:    0.3s
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:    3.6s finished
/usr/local/lib/python3.7/site-packages/sklearn/model_selection/_search.py:715:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  self.best_estimator_.fit(X, y, **fit_params)
```

[180]:
```
{'mean_fit_time': array([0.00796089, 0.00896688, 0.00782228, 0.0078167 ,
0.00740767,
        0.0070888 , 0.00710282, 0.0069149 , 0.00767436]),
 'std_fit_time': array([0.00018556, 0.00170471, 0.00058437, 0.00074476,
0.00050011,
        0.00019064, 0.00027769, 0.00016248, 0.00114481]),
 'mean_score_time': array([0.09102678, 0.09610238, 0.11113396, 0.07794733,
0.07878141,
        0.08365622, 0.07859507, 0.08171954, 0.07590122]),
 'std_score_time': array([0.00214686, 0.00207809, 0.0140908 , 0.00306418,
0.00275461,
        0.00215417, 0.00226467, 0.00593353, 0.0037656 ]),
 'param_leaf_size': masked_array(data=[20, 20, 20, 30, 30, 30, 40, 40, 40],
             mask=[False, False, False, False, False, False, False, False,
                   False],
        fill_value='?',
             dtype=object),
 'param_n_neighbors': masked_array(data=[3, 5, 7, 3, 5, 7, 3, 5, 7],
             mask=[False, False, False, False, False, False, False, False,
                   False],
        fill_value='?',
             dtype=object),
 'params': [{'leaf_size': 20, 'n_neighbors': 3},
  {'leaf_size': 20, 'n_neighbors': 5},
  {'leaf_size': 20, 'n_neighbors': 7},
  {'leaf_size': 30, 'n_neighbors': 3},
  {'leaf_size': 30, 'n_neighbors': 5},
  {'leaf_size': 30, 'n_neighbors': 7},
  {'leaf_size': 40, 'n_neighbors': 3},
  {'leaf_size': 40, 'n_neighbors': 5},
  {'leaf_size': 40, 'n_neighbors': 7}],
 'split0_test_score': array([0.5531463 , 0.59285785, 0.62179361, 0.5531463 ,
0.59285785,
        0.62179361, 0.5531463 , 0.59285785, 0.62179361]),
 'split1_test_score': array([0.64386792, 0.66441813, 0.67211209, 0.64386792,
0.66441813,
```

```
        0.67211209, 0.64386792, 0.66441813, 0.67211209]),
 'split2_test_score': array([0.58921032, 0.6439339 , 0.62297137, 0.58921032,
0.6439339 ,
        0.62297137, 0.58921032, 0.6439339 , 0.62297137]),
 'split3_test_score': array([0.56837973, 0.6072536 , 0.66814553, 0.56837973,
0.6072536 ,
        0.66814553, 0.56837973, 0.6072536 , 0.66814553]),
 'split4_test_score': array([0.59786047, 0.61406094, 0.62115385, 0.59786047,
0.61406094,
        0.62115385, 0.59786047, 0.61406094, 0.62115385]),
 'mean_test_score': array([0.59047976, 0.62449863, 0.64123548, 0.59047976,
0.62449863,
        0.64123548, 0.59047976, 0.62449863, 0.64123548]),
 'std_test_score': array([0.03095337, 0.02600034, 0.02363182, 0.03095337,
0.02600034,
        0.02363182, 0.03095337, 0.02600034, 0.02363182]),
 'rank_test_score': array([7, 4, 1, 7, 4, 1, 7, 4, 1], dtype=int32),
 'split0_train_score': array([0.91728351, 0.87579922, 0.84687763, 0.91728351,
0.87579922,
        0.84687763, 0.91728351, 0.87579922, 0.84687763]),
 'split1_train_score': array([0.90760622, 0.86177029, 0.84109272, 0.90760622,
0.86177029,
        0.84109272, 0.90760622, 0.86177029, 0.84109272]),
 'split2_train_score': array([0.91776892, 0.87055731, 0.84152184, 0.91776892,
0.87055731,
        0.84152184, 0.91776892, 0.87055731, 0.84152184]),
 'split3_train_score': array([0.91575807, 0.86615399, 0.84398609, 0.91575807,
0.86615399,
        0.84398609, 0.91575807, 0.86615399, 0.84398609]),
 'split4_train_score': array([0.91360959, 0.87290322, 0.84824004, 0.91360959,
0.87290322,
        0.84824004, 0.91360959, 0.87290322, 0.84824004]),
 'mean_train_score': array([0.91440526, 0.8694368 , 0.84434367, 0.91440526,
0.8694368 ,
        0.84434367, 0.91440526, 0.8694368 , 0.84434367]),
 'std_train_score': array([0.00369564, 0.00496597, 0.00283767, 0.00369564,
0.00496597,
        0.00283767, 0.00369564, 0.00496597, 0.00283767])}
```

[181]:
```
knn.best_estimator_
```

[181]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=20, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

[182]:
```
knn_pred = knn.best_estimator_.predict(X_tests)
knn_prob = knn.best_estimator_.predict_proba(X_tests)
```

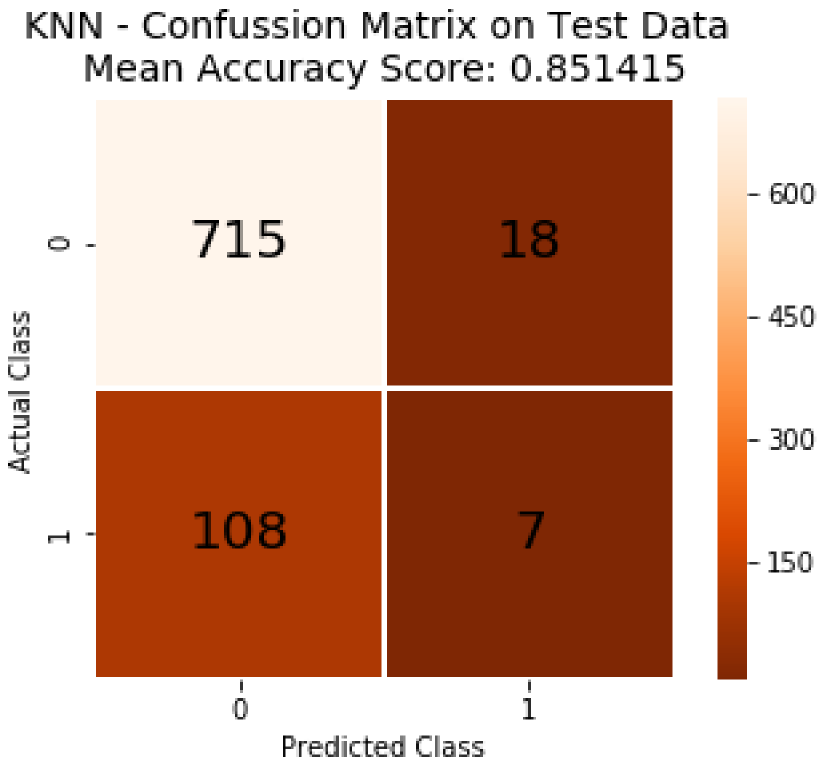
```
knn_prob
```

[182]:
```
array([[1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       …,
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.85714286, 0.14285714]])
```

[183]:
```
knn_matrix = metrics.confusion_matrix(y_test, knn_pred)
knn_matrix
```

[183]:
```
array([[715,  18],
       [108,   7]])
```

[184]:
```
knn_test = knn.best_estimator_.score(X_tests, y_test)

knn_matrix = metrics.confusion_matrix(y_test, knn_pred)
knn_cm = pd.DataFrame(knn_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (knn_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

knn_title = 'KNN - Confussion Matrix on Test Data \nMean Accuracy Score: {0:
 ↪2f}'.format(knn_test)
plt.title(knn_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



[185]:
```python
print("", classification_report(y_test, knn_pred, target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years      0.87      0.98      0.92       733
  Risky in 10 years      0.28      0.06      0.10       115

           accuracy                          0.85       848
          macro avg      0.57      0.52      0.51       848
       weighted avg      0.79      0.85      0.81       848
```

[186]:
```python
acc_knn = accuracy_score(y_test, knn_pred)
print("KNN accuracy:", acc_knn)
```

```
KNN accuracy: 0.8514150943396226
```

[187]:
```python
error_knn = 1-acc_knn
error_knn
```

[187]: 0.1485849056603774

[188]:
```
knn_probs = knn.best_estimator_.predict_proba(X_tests)[:,1]
print(roc_auc_score(y_test, knn_probs))
```

```
0.595242896968978
```

11.0.1 For training set:

[189]:
```
knn_pred_tr = knn.best_estimator_.predict(X_trains)
knn_prob_tr = knn.best_estimator_.predict_proba(X_trains)
knn_prob_tr
```

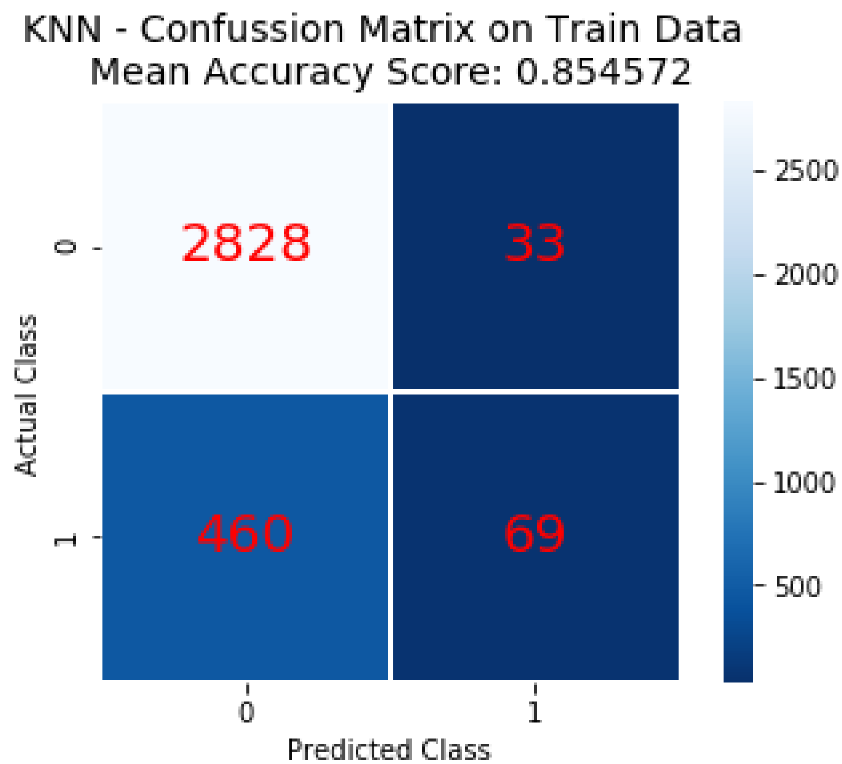
[189]:
```
array([[0.85714286, 0.14285714],
       [0.85714286, 0.14285714],
       [1.        , 0.        ],
       …,
       [0.42857143, 0.57142857],
       [1.        , 0.        ],
       [1.        , 0.        ]])
```

[190]:
```
knn_matrix_tr = metrics.confusion_matrix(y_train, knn_pred_tr)
knn_matrix_tr
```

[190]:
```
array([[2828,   33],
       [ 460,   69]])
```

[191]:
```
knn_train = knn.best_estimator_.score(X_trains, y_train)

knn_matrix = metrics.confusion_matrix(y_train, knn_pred_tr)

knn_cm_tr = pd.DataFrame(knn_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (knn_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
↪5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

knn_title = 'KNN - Confussion Matrix on Train Data \nMean Accuracy Score: {0:
↪2f}'.format(knn_train)
plt.title(knn_title, size = 14)
# plt.figure(figsize=(16, 26))
```

```
plt.show;
```

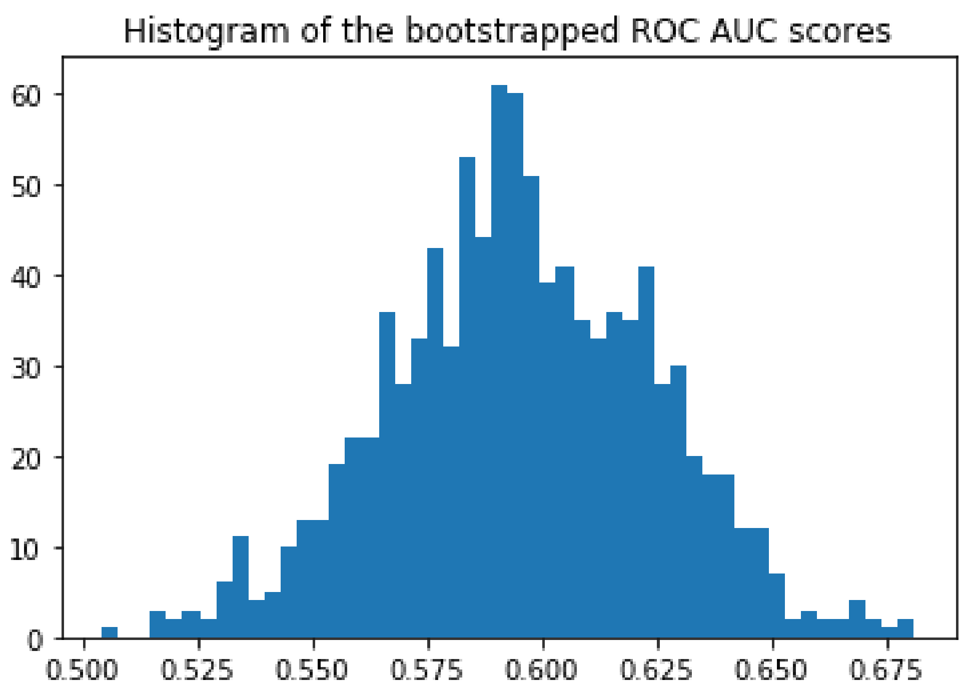


[192]:

```
y_pred = knn_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
```

```
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()
sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.5952
```




```
Confidence interval for the score: [0.5482 - 0.6422]
```

[193]:
```
alpha = .95
y_pred = knn_probs
y_true = y_test_v2
```

```
auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.5952428969689779
AUC COV: 0.0008321927812735917
95% AUC CI: [0.53870234 0.65178345]
```

[ ]:

```

```

12 Support Vector Machine

[194]:

```
from sklearn.svm import SVC
seed(0)
params4 = {'degree' : [2,3,4],
           'C': [0.5,1.0,1.5,2.0,2.5],
           'tol':[1e-5,1e-3,1e-1],
           'max_iter':[-1,-2,-3]}
svc = SVC(random_state=0,probability=True)
svc = GridSearchCV(svc, cv=5, param_grid=params4, scoring = 'roc_auc',refit =␣
 ↪True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

svc.fit(X_trains, y_train)
svc.cv_results_
```

```
Fitting 5 folds for each of 135 candidates, totalling 675 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    5.2s
[Parallel(n_jobs=-1)]: Done 198 tasks      | elapsed:   11.5s
[Parallel(n_jobs=-1)]: Done 340 tasks      | elapsed:   16.8s
```

```
[Parallel(n_jobs=-1)]: Done 534 tasks      | elapsed:   27.7s
[Parallel(n_jobs=-1)]: Done 652 out of 675 | elapsed:   33.1s remaining:    1.2s
[Parallel(n_jobs=-1)]: Done 675 out of 675 | elapsed:   35.8s finished
/usr/local/lib/python3.7/site-packages/sklearn/utils/validation.py:724:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[194]:
```
{'mean_fit_time': array([1.6430851 , 1.32595725, 0.75211272, 0.00830755,
0.00708532,
        0.00793896, 0.00672722, 0.00800772, 0.00696988, 1.65427618,
        1.33386455, 0.79126811, 0.00761147, 0.00729547, 0.00693927,
        0.00706992, 0.00760274, 0.0082715 , 1.84108286, 1.49798107,
        0.83724113, 0.00684819, 0.0069313 , 0.00682526, 0.00684443,
        0.00707517, 0.00705266, 1.92636771, 1.56961594, 0.96944757,
        0.00784702, 0.00730886, 0.00731134, 0.00761909, 0.00711923,
        0.0074254 , 1.9130188 , 1.5422008 , 0.97861357, 0.00715365,
        0.00768843, 0.0070425 , 0.00808721, 0.00771117, 0.00908237,
        1.93496904, 1.5502666 , 0.95974221, 0.00723181, 0.00719104,
        0.00765829, 0.00759192, 0.00772309, 0.0076055 , 2.04185619,
        1.58932443, 1.07688565, 0.00748119, 0.00771365, 0.00760064,
        0.00750141, 0.0080493 , 0.00758719, 2.08480768, 1.62337322,
        1.07169557, 0.00752177, 0.0079072 , 0.00744886, 0.00713735,
        0.0074131 , 0.00741754, 2.06221352, 1.62698522, 1.06578321,
        0.00705795, 0.00866446, 0.00805378, 0.00844035, 0.0089551 ,
        0.00857449, 2.08964376, 1.66066399, 1.15852423, 0.00757113,
        0.00739689, 0.00806298, 0.00768819, 0.00756493, 0.00795603,
        2.16588745, 1.68270507, 1.1607944 , 0.00800796, 0.00768576,
        0.00739827, 0.00826998, 0.00843039, 0.00731144, 2.10631199,
        1.6787787 , 1.16217556, 0.00745959, 0.00741858, 0.00772562,
        0.00760093, 0.00796094, 0.00811472, 2.19450927, 1.71344576,
        1.17613769, 0.00802121, 0.0077848 , 0.00842266, 0.00786119,
        0.00733128, 0.00813999, 2.15442266, 1.6890182 , 1.21509047,
        0.0074821 , 0.0071723 , 0.00821428, 0.00782366, 0.00824919,
        0.00796609, 1.64544163, 1.29366064, 1.05099344, 0.0072536 ,
        0.00788922, 0.0082819 , 0.00894146, 0.00788832, 0.00827861]),
 'std_fit_time': array([5.15530903e-02, 3.27813306e-02, 5.97741650e-03,
1.35868269e-03,
        4.55490693e-04, 2.26906406e-03, 3.89409088e-04, 3.01201314e-03,
        2.38827184e-04, 5.74594365e-02, 2.78795447e-02, 2.95975160e-02,
        9.86839555e-04, 9.25505002e-04, 4.57458608e-04, 7.96624536e-04,
        1.41138288e-03, 1.04337721e-03, 6.53974642e-02, 1.53588945e-02,
        4.10393587e-02, 2.99513698e-04, 2.25686831e-04, 3.12341748e-04,
        1.32640773e-04, 2.33550858e-04, 3.25979113e-04, 6.26793651e-02,
        3.00775512e-02, 1.44213899e-02, 4.60426622e-04, 4.14671109e-04,
        1.92752243e-04, 9.39455687e-04, 1.88360842e-04, 2.21866268e-04,
```

```
       5.45702092e-02, 3.51418099e-02, 2.41807873e-02, 9.64173224e-05,
       4.53577035e-04, 2.00941159e-04, 1.07506853e-03, 6.25556978e-04,
       1.21192761e-03, 5.20947476e-02, 4.23181394e-02, 7.27042547e-03,
       3.96543285e-04, 4.07130078e-04, 3.39909294e-04, 3.14618819e-04,
       3.84287234e-04, 1.87866300e-04, 5.73668151e-02, 3.55518897e-02,
       3.02796509e-02, 2.61943912e-04, 8.41675770e-04, 2.41026614e-04,
       3.46797879e-04, 1.14996933e-03, 3.49574911e-04, 7.86945957e-02,
       2.26940978e-02, 2.23044261e-02, 3.00183349e-04, 4.50374385e-04,
       3.71557664e-04, 1.12953192e-04, 4.31242188e-04, 7.11669380e-04,
       5.69980091e-02, 3.76813290e-02, 7.51059826e-03, 1.73277033e-04,
       1.33348457e-03, 5.31451816e-04, 6.42129296e-04, 6.96095674e-04,
       4.80439266e-04, 8.12836799e-02, 4.08010685e-02, 2.65624428e-02,
       5.17995595e-04, 8.70387050e-05, 7.95537080e-04, 1.02113383e-04,
       1.92307640e-04, 3.37798154e-04, 9.77620956e-02, 5.86468574e-02,
       1.89968022e-02, 4.34605259e-04, 4.90583723e-04, 1.72581169e-04,
       9.08871630e-04, 6.16531667e-04, 2.44093151e-04, 6.09229571e-02,
       4.23334842e-02, 1.05120982e-02, 1.98900513e-04, 3.10284845e-04,
       4.44033174e-04, 5.26057131e-04, 1.97441023e-04, 2.77072450e-04,
       8.06675528e-02, 7.07569555e-02, 1.02284640e-02, 9.99697234e-04,
       7.43154488e-04, 1.20414633e-03, 7.07475265e-04, 5.52087214e-04,
       4.87319005e-04, 7.07444281e-02, 4.17955303e-02, 4.06038983e-02,
       2.79264167e-04, 1.19319140e-04, 1.29299698e-03, 4.06201127e-04,
       1.15695924e-03, 5.75343056e-04, 2.76578309e-01, 2.73849331e-01,
       1.70413699e-01, 8.48251294e-04, 5.31625048e-04, 5.98867614e-04,
       4.10623119e-04, 2.26265638e-04, 6.02784618e-04]),
 'mean_score_time': array([0.03366528, 0.0339355 , 0.02828231, 0.00294347,
0.00309381,
       0.00301647, 0.0036694 , 0.0031271 , 0.00317507, 0.03626199,
       0.03331523, 0.03092694, 0.0030746 , 0.00318961, 0.00309439,
       0.00313816, 0.00339565, 0.00368533, 0.03831029, 0.03821735,
       0.03129883, 0.00294995, 0.00290885, 0.00316215, 0.00302205,
       0.00333118, 0.00338974, 0.03754649, 0.03915534, 0.03575387,
       0.00343194, 0.00294399, 0.0031774 , 0.00329723, 0.00337377,
       0.00331182, 0.03813076, 0.04141483, 0.03532038, 0.00314956,
       0.00330338, 0.00321965, 0.00333533, 0.00328026, 0.00344572,
       0.03788934, 0.0369082 , 0.03414307, 0.00293164, 0.00319409,
       0.00364733, 0.00323167, 0.00347986, 0.00328927, 0.04010549,
       0.04003115, 0.03768778, 0.00302577, 0.00303245, 0.00358524,
       0.00317864, 0.00315046, 0.00370536, 0.03745446, 0.03762197,
       0.036376  , 0.00387635, 0.00346041, 0.0031682 , 0.00303321,
       0.00305386, 0.00307221, 0.03816609, 0.03773685, 0.03611131,
       0.00296278, 0.00309114, 0.00379767, 0.00379496, 0.00376883,
       0.00388842, 0.03797321, 0.03839073, 0.03962264, 0.00310555,
       0.00385385, 0.00329318, 0.00327501, 0.00359201, 0.00474162,
       0.03980336, 0.03850317, 0.03675399, 0.00324097, 0.00340943,
       0.00462813, 0.00365853, 0.0041604 , 0.0032423 , 0.0379416 ,
       0.03802876, 0.03712173, 0.00319057, 0.00337386, 0.00374641,
```

```
        0.00410862, 0.00425625, 0.00365028, 0.03812528, 0.03783526,
        0.03892298, 0.00338769, 0.00391283, 0.00342607, 0.00340362,
        0.00378556, 0.00324736, 0.04029894, 0.03821921, 0.03933206,
        0.00307574, 0.00307927, 0.00383377, 0.00421329, 0.00365601,
        0.00320954, 0.0254416 , 0.02791257, 0.03154726, 0.00308175,
        0.00379143, 0.00391045, 0.00364389, 0.00395279, 0.00423098]),
 'std_score_time': array([5.56081692e-04, 6.99879392e-04, 7.70117145e-04,
1.61939782e-04,
        2.16434311e-04, 1.85510615e-04, 1.11283391e-03, 3.38103807e-04,
        4.25051915e-04, 3.03333812e-03, 6.69264307e-04, 4.65477202e-03,
        4.29186406e-04, 6.04429374e-04, 2.39666031e-04, 5.79335377e-04,
        9.34477727e-04, 9.23924366e-04, 4.81115862e-04, 9.86177422e-04,
        1.90538963e-03, 1.89194599e-04, 9.40497200e-05, 5.58272852e-04,
        2.96135324e-04, 8.02762784e-04, 6.62139632e-04, 8.69336424e-04,
        1.58812807e-03, 3.03372364e-03, 3.11417000e-04, 6.15082969e-05,
        2.05043748e-04, 2.74536467e-04, 2.32440904e-04, 9.77437551e-05,
        2.21230432e-03, 5.42847972e-03, 1.82897669e-03, 1.99065745e-04,
        1.71187900e-04, 9.94412311e-05, 3.46614816e-04, 1.60726210e-04,
        3.76629901e-04, 1.49944365e-03, 4.56550012e-04, 5.77628941e-04,
        5.17236236e-05, 2.17144543e-04, 7.34395374e-04, 2.38022504e-04,
        1.54047526e-04, 1.00891734e-04, 2.27020154e-03, 2.91571500e-03,
        2.16605738e-03, 5.70756351e-05, 9.04770079e-05, 3.73388542e-04,
        1.64407644e-04, 4.10614663e-05, 6.35600297e-04, 4.39149435e-04,
        1.68606380e-03, 2.06513408e-03, 7.52387020e-04, 1.47264500e-04,
        1.96426040e-04, 4.37668980e-05, 9.64986232e-05, 1.27446664e-04,
        1.25424159e-03, 6.90177690e-04, 1.52220619e-03, 4.05433300e-05,
        5.09815416e-05, 6.84786183e-04, 6.53443650e-04, 2.08164549e-04,
        4.30386293e-04, 1.22052231e-03, 1.08193066e-03, 2.86648538e-03,
        1.42284613e-04, 1.24033891e-03, 1.24113632e-04, 1.02637866e-04,
        5.85039747e-04, 1.53244407e-03, 4.46710678e-03, 1.66929457e-03,
        1.12857682e-03, 2.45483188e-04, 3.41478747e-04, 1.99290831e-03,
        5.81059428e-04, 2.15628488e-03, 1.57197410e-04, 4.23211060e-04,
        6.35648514e-04, 5.35104556e-04, 2.80281474e-04, 5.72055340e-04,
        1.01024240e-03, 1.49964882e-03, 7.13520632e-04, 5.77897474e-04,
        1.28764888e-03, 4.26213169e-04, 7.81980976e-04, 4.39744672e-04,
        6.81587281e-04, 2.45597569e-04, 2.82855502e-04, 7.88755114e-04,
        3.62167682e-04, 2.54489654e-03, 9.14436164e-04, 2.76863936e-03,
        6.14851146e-05, 1.23761851e-04, 1.04843381e-03, 1.07256737e-03,
        6.37100352e-04, 2.53737656e-04, 3.18689419e-03, 5.46489791e-03,
        2.55748164e-03, 4.33873116e-04, 1.11509761e-03, 8.03145782e-04,
        3.20572256e-04, 5.75118580e-04, 5.47465378e-04]),
 'param_C': masked_array(data=[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
0.5,
                   0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
                   0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                   1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                   1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.5,
```

```
                  1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5,
                  1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5,
                  1.5, 1.5, 1.5, 1.5, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0,
                  2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0,
                  2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.5, 2.5,
                  2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5,
                  2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5,
                  2.5, 2.5, 2.5],
            mask=[False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False],
      fill_value='?',
           dtype=object),
 'param_degree': masked_array(data=[2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3,
3, 3, 3,
                  4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                  3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4,
                  2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                  4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                  3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4,
                  2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                  4, 4, 4, 4, 4, 4, 4, 4, 4],
            mask=[False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
```

```
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_max_iter': masked_array(data=[-1, -1, -1, -2, -2, -2, -3, -3, -3, -1,
-1, -1, -2, -2,
                   -2, -3, -3, -3, -1, -1, -1, -2, -2, -2, -3, -3, -3, -1,
                   -1, -1, -2, -2, -2, -3, -3, -3, -1, -1, -1, -2, -2, -2,
                   -3, -3, -3, -1, -1, -1, -2, -2, -2, -3, -3, -3, -1, -1,
                   -1, -2, -2, -2, -3, -3, -3, -1, -1, -1, -2, -2, -2, -3,
                   -3, -3, -1, -1, -1, -2, -2, -2, -3, -3, -3, -1, -1, -1,
                   -2, -2, -2, -3, -3, -3, -1, -1, -1, -2, -2, -2, -3, -3,
                   -3, -1, -1, -1, -2, -2, -2, -3, -3, -3, -1, -1, -1, -2,
                   -2, -2, -3, -3, -3, -1, -1, -1, -2, -2, -2, -3, -3, -3,
                   -1, -1, -1, -2, -2, -2, -3, -3, -3],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_tol': masked_array(data=[1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
0.001,
                   0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                   0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                   1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                   0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                   0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                   1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                   0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
```

```
                    0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                    1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                    0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                    0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                    1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                    0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                    0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                    1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                    0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.5, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
  {'C': 0.5, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
  {'C': 0.5, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
  {'C': 0.5, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
  {'C': 0.5, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
  {'C': 0.5, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
  {'C': 0.5, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
  {'C': 0.5, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
  {'C': 0.5, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
  {'C': 0.5, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
  {'C': 0.5, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
  {'C': 0.5, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
  {'C': 0.5, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
  {'C': 0.5, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
  {'C': 0.5, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
  {'C': 0.5, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
  {'C': 0.5, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
  {'C': 0.5, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
  {'C': 0.5, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
```

```
{'C': 0.5, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
{'C': 0.5, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
{'C': 0.5, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
{'C': 0.5, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
{'C': 0.5, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
{'C': 0.5, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
{'C': 0.5, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
{'C': 0.5, 'degree': 4, 'max_iter': -3, 'tol': 0.1},
{'C': 1.0, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.0, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
{'C': 1.0, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
{'C': 1.0, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.0, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
{'C': 1.0, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
{'C': 1.0, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.0, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
{'C': 1.0, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
{'C': 1.0, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.0, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
{'C': 1.0, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
{'C': 1.0, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.0, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
{'C': 1.0, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
{'C': 1.0, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.0, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
{'C': 1.0, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
{'C': 1.0, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.0, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
{'C': 1.0, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
{'C': 1.0, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.0, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
{'C': 1.0, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
{'C': 1.0, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.0, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
{'C': 1.0, 'degree': 4, 'max_iter': -3, 'tol': 0.1},
{'C': 1.5, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.5, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
{'C': 1.5, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
{'C': 1.5, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.5, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
{'C': 1.5, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
{'C': 1.5, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.5, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
{'C': 1.5, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
{'C': 1.5, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.5, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
{'C': 1.5, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
```

```
{'C': 1.5, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.5, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
{'C': 1.5, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
{'C': 1.5, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.5, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
{'C': 1.5, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
{'C': 1.5, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.5, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
{'C': 1.5, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
{'C': 1.5, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.5, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
{'C': 1.5, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
{'C': 1.5, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.5, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
{'C': 1.5, 'degree': 4, 'max_iter': -3, 'tol': 0.1},
{'C': 2.0, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.0, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
{'C': 2.0, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
{'C': 2.0, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
{'C': 2.0, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
{'C': 2.0, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
{'C': 2.0, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
{'C': 2.0, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
{'C': 2.0, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
{'C': 2.0, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.0, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
{'C': 2.0, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
{'C': 2.0, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
{'C': 2.0, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
{'C': 2.0, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
{'C': 2.0, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
{'C': 2.0, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
{'C': 2.0, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
{'C': 2.0, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.0, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
{'C': 2.0, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
{'C': 2.0, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
{'C': 2.0, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
{'C': 2.0, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
{'C': 2.0, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
{'C': 2.0, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
{'C': 2.0, 'degree': 4, 'max_iter': -3, 'tol': 0.1},
{'C': 2.5, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.5, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
{'C': 2.5, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
{'C': 2.5, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
{'C': 2.5, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
```

```
 {'C': 2.5, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
 {'C': 2.5, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
 {'C': 2.5, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
 {'C': 2.5, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
 {'C': 2.5, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
 {'C': 2.5, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
 {'C': 2.5, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
 {'C': 2.5, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
 {'C': 2.5, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
 {'C': 2.5, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
 {'C': 2.5, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
 {'C': 2.5, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
 {'C': 2.5, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
 {'C': 2.5, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
 {'C': 2.5, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
 {'C': 2.5, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
 {'C': 2.5, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
 {'C': 2.5, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
 {'C': 2.5, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
 {'C': 2.5, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
 {'C': 2.5, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
 {'C': 2.5, 'degree': 4, 'max_iter': -3, 'tol': 0.1}],
'split0_test_score': array([0.58735882, 0.58737528, 0.58887352, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.58735882,
       0.58737528, 0.58887352, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.58735882, 0.58737528,
       0.58887352, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.58760578, 0.58747407, 0.59369752,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.58760578, 0.58747407, 0.59369752, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.58760578, 0.58747407, 0.59369752, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.58801739,
       0.58781982, 0.59508051, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.58801739, 0.58781982,
       0.59508051, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.58801739, 0.58781982, 0.59508051,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.58854424, 0.58805031, 0.58635451, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.58854424, 0.58805031, 0.58635451, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.58854424,
       0.58805031, 0.58635451, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.58839606, 0.58842899,
       0.58650268, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.58839606, 0.58842899, 0.58650268,
```

```
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.58839606, 0.58842899, 0.58650268, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split1_test_score': array([0.62707811, 0.62603906, 0.62671527, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.62707811,
       0.62603906, 0.62671527, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.62707811, 0.62603906,
       0.62671527, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.62529687, 0.62514844, 0.63261974,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.62529687, 0.62514844, 0.63261974, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.62529687, 0.62514844, 0.63261974, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.62104169,
       0.62109117, 0.62481858, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.62104169, 0.62109117,
       0.62481858, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.62104169, 0.62109117, 0.62481858,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.61827088, 0.61803998, 0.60952962, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.61827088, 0.61803998, 0.60952962, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.61827088,
       0.61803998, 0.60952962, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.61569798, 0.61543409,
       0.612152  , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.61569798, 0.61543409, 0.612152  ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.61569798, 0.61543409, 0.612152  , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split2_test_score': array([0.60753398, 0.60825966, 0.59381185, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.60753398,
       0.60825966, 0.59381185, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.60753398, 0.60825966,
       0.59381185, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.60136562, 0.60115121, 0.59033184,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.60136562, 0.60115121, 0.59033184, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.60136562, 0.60115121, 0.59033184, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.59366341,
       0.59386133, 0.59810001, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.59366341, 0.59386133,
       0.59810001, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.59366341, 0.59386133, 0.59810001,
```

```
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.59163478, 0.59168426, 0.58258016, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.59163478, 0.59168426, 0.58258016, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.59163478,
       0.59168426, 0.58258016, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.58962264, 0.58950719,
       0.58658794, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.58962264, 0.58950719, 0.58658794,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.58962264, 0.58950719, 0.58658794, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split3_test_score': array([0.63245481, 0.63233936, 0.64922813, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.63245481,
       0.63233936, 0.64922813, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.63245481, 0.63233936,
       0.64922813, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.63446695, 0.634335  , 0.62742446,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.63446695, 0.634335  , 0.62742446, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.63446695, 0.634335  , 0.62742446, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.6338897 ,
       0.63408761, 0.64240005, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.6338897 , 0.63408761,
       0.64240005, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.6338897 , 0.63408761, 0.64240005,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.63161367, 0.63174561, 0.62806769, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.63161367, 0.63174561, 0.62806769, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.63161367,
       0.63174561, 0.62806769, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.62778731, 0.62810067,
       0.63459889, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.62778731, 0.62810067, 0.63459889,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.62778731, 0.62810067, 0.63459889, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split4_test_score': array([0.56390276, 0.56403596, 0.54004329, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.56390276,
       0.56403596, 0.54004329, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.56390276, 0.56403596,
       0.54004329, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.56476856, 0.56521812, 0.56200466,
```

```
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.56476856, 0.56521812, 0.56200466, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.56476856, 0.56521812, 0.56200466, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.56440226,
       0.56403596, 0.56315351, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.56440226, 0.56403596,
       0.56315351, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.56440226, 0.56403596, 0.56315351,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.56748252, 0.56731602, 0.56871462, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.56748252, 0.56731602, 0.56871462, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.56748252,
       0.56731602, 0.56871462, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.57151182, 0.57169497,
       0.56591742, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.57151182, 0.57169497, 0.56591742,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.57151182, 0.57169497, 0.56591742, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'mean_test_score': array([0.60367262, 0.60361675, 0.59974882, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.60367262,
       0.60361675, 0.59974882, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.60367262, 0.60361675,
       0.59974882, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.60270749, 0.60267193, 0.60122499,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.60270749, 0.60267193, 0.60122499, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.60270749, 0.60267193, 0.60122499, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.60020986,
       0.60018619, 0.60471995, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.60020986, 0.60018619,
       0.60471995, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.60020986, 0.60018619, 0.60471995,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.59951543, 0.59937335, 0.59505452, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.59951543, 0.59937335, 0.59505452, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.59951543,
       0.59937335, 0.59505452, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.59860814, 0.59863812,
       0.59715786, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.59860814, 0.59863812, 0.59715786,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
```

```
       0.5       , 0.59860814, 0.59863812, 0.59715786, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'std_test_score': array([0.02544399, 0.02521034, 0.03711759, 0.        , 0.
,
       0.        , 0.        , 0.        , 0.        , 0.02544399,
       0.02521034, 0.03711759, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.02544399, 0.02521034,
       0.03711759, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.02524433, 0.02506827, 0.02601313,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.02524433, 0.02506827, 0.02601313, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.02524433, 0.02506827, 0.02601313, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.02465896,
       0.02483704, 0.02715086, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.02465896, 0.02483704,
       0.02715086, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.02465896, 0.02483704, 0.02715086,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.02275657, 0.02284769, 0.02108827, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.02275657, 0.02284769, 0.02108827, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.02275657,
       0.02284769, 0.02108827, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.02030662, 0.02031139,
       0.02377648, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.02030662, 0.02031139, 0.02377648,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.02030662, 0.02031139, 0.02377648, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ]),
 'rank_test_score': array([ 4,  7, 25, 46, 46, 46, 46, 46, 46,  4,  7, 25, 46,
46, 46, 46, 46,
       46,  4,  7, 25, 46, 46, 46, 46, 46, 46, 10, 13, 16, 46, 46, 46, 46,
       46, 46, 10, 13, 16, 46, 46, 46, 46, 46, 46, 10, 13, 16, 46, 46, 46,
       46, 46, 46, 19, 22,  1, 46, 46, 46, 46, 46, 46, 19, 22,  1, 46, 46,
       46, 46, 46, 46, 19, 22,  1, 46, 46, 46, 46, 46, 46, 28, 31, 43, 46,
       46, 46, 46, 46, 46, 28, 31, 43, 46, 46, 46, 46, 46, 46, 28, 31, 43,
       46, 46, 46, 46, 46, 46, 37, 34, 40, 46, 46, 46, 46, 46, 46, 37, 34,
       40, 46, 46, 46, 46, 46, 46, 37, 34, 40, 46, 46, 46, 46, 46, 46],
      dtype=int32),
 'split0_train_score': array([0.86895551, 0.87081845, 0.82444949, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86895551,
       0.87081845, 0.82444949, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.86895551, 0.87081845,
       0.82444949, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.87127722, 0.87193953, 0.82411368,
```

```
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.87127722, 0.87193953, 0.82411368, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.87127722, 0.87193953, 0.82411368, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.87509713,
       0.87513536, 0.85327188, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.87509713, 0.87513536,
       0.85327188, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.87509713, 0.87513536, 0.85327188,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.88227302, 0.8828289 , 0.86736845, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.88227302, 0.8828289 , 0.86736845, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.88227302,
       0.8828289 , 0.86736845, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.88631921, 0.88653309,
       0.8722371 , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.88631921, 0.88653309, 0.8722371 ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.88631921, 0.88653309, 0.8722371 , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split1_train_score': array([0.86129985, 0.861525  , 0.80146698, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86129985,
       0.861525  , 0.80146698, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.86129985, 0.861525  ,
       0.80146698, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.86288623, 0.86225932, 0.8327875 ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.86288623, 0.86225932, 0.8327875 , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.86288623, 0.86225932, 0.8327875 , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86589269,
       0.86551366, 0.84856447, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.86589269, 0.86551366,
       0.84856447, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.86589269, 0.86551366, 0.84856447,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.87150593, 0.87129937, 0.85471579, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.87150593, 0.87129937, 0.85471579, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.87150593,
       0.87129937, 0.85471579, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.87762627, 0.87665854,
       0.86988651, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.87762627, 0.87665854, 0.86988651,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
```

```
       0.5       , 0.87762627, 0.87665854, 0.86988651, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split2_train_score': array([0.86583382, 0.86491154, 0.79607166, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86583382,
       0.86491154, 0.79607166, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.86583382, 0.86491154,
       0.79607166, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.86834661, 0.86838792, 0.83773562,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.86834661, 0.86838792, 0.83773562, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.86834661, 0.86838792, 0.83773562, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.87300761,
       0.87339181, 0.8617202 , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.87300761, 0.87339181,
       0.8617202 , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.87300761, 0.87339181, 0.8617202 ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.87803422, 0.8782005 , 0.87181163, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.87803422, 0.8782005 , 0.87181163, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.87803422,
       0.8782005 , 0.87181163, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.88233064, 0.88225215,
       0.87811581, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.88233064, 0.88225215, 0.87811581,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.88233064, 0.88225215, 0.87811581, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split3_train_score': array([0.86851392, 0.86761746, 0.80669085, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86851392,
       0.86761746, 0.80669085, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.86851392, 0.86761746,
       0.80669085, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.87198618, 0.87310573, 0.85800008,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.87198618, 0.87310573, 0.85800008, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.87198618, 0.87310573, 0.85800008, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.87763143,
       0.87714912, 0.8544328 , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.87763143, 0.87714912,
       0.8544328 , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.87763143, 0.87714912, 0.8544328 ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
```

```
       0.5       , 0.88047162, 0.88142592, 0.86761333, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.88047162, 0.88142592, 0.86761333, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.88047162,
       0.88142592, 0.86761333, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.88424545, 0.8838788 ,
       0.87870141, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.88424545, 0.8838788 , 0.87870141,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.88424545, 0.8838788 , 0.87870141, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split4_train_score': array([0.86001859, 0.86087688, 0.79641971, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86001859,
       0.86087688, 0.79641971, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.86001859, 0.86087688,
       0.79641971, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.86232556, 0.86283868, 0.83936505,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.86232556, 0.86283868, 0.83936505, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.86232556, 0.86283868, 0.83936505, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86687665,
       0.86629656, 0.86189075, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.86687665, 0.86629656,
       0.86189075, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.86687665, 0.86629656, 0.86189075,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.86936188, 0.86909605, 0.86087482, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.86936188, 0.86909605, 0.86087482, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86936188,
       0.86909605, 0.86087482, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.87014392, 0.87037163,
       0.86337343, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.87014392, 0.87037163, 0.86337343,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.87014392, 0.87037163, 0.86337343, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'mean_train_score': array([0.86492434, 0.86514987, 0.80501974, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.86492434,
       0.86514987, 0.80501974, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.86492434, 0.86514987,
       0.80501974, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.86736436, 0.86770623, 0.83840039,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
```

```
       0.5       , 0.86736436, 0.86770623, 0.83840039, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.86736436, 0.86770623, 0.83840039, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.8717011 ,
       0.8714973 , 0.85597602, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.8717011 , 0.8714973 ,
       0.85597602, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.8717011 , 0.8714973 , 0.85597602,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.87632933, 0.87657015, 0.8644768 , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.87632933, 0.87657015, 0.8644768 , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.87632933,
       0.87657015, 0.8644768 , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.8801331 , 0.87993884,
       0.87246285, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.8801331 , 0.87993884, 0.87246285,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.8801331 , 0.87993884, 0.87246285, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'std_train_score': array([0.00366513, 0.00373301, 0.01045905, 0.        , 0.
,
       0.        , 0.        , 0.        , 0.        , 0.00366513,
       0.00373301, 0.01045905, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00366513, 0.00373301,
       0.01045905, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.00407626, 0.00449226, 0.01114571,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.00407626, 0.00449226, 0.01114571, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00407626, 0.00449226, 0.01114571, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.00459177,
       0.00472481, 0.00514985, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00459177, 0.00472481,
       0.00514985, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.00459177, 0.00472481, 0.00514985,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.00504387, 0.00545989, 0.00600418, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00504387, 0.00545989, 0.00600418, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.00504387,
       0.00545989, 0.00600418, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00576327, 0.00577323,
       0.00566006, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.00576327, 0.00577323, 0.00566006,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.00576327, 0.00577323, 0.00566006, 0.        ,
```

```
         0.         , 0.         , 0.         , 0.         , 0.        ])}
```

[195]:
```
svc.best_estimator_
```

[195]:
```
SVC(C=1.5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=2, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=True, random_state=0, shrinking=True,
    tol=0.1, verbose=False)
```

[196]:
```
svc_pred = svc.best_estimator_.predict(X_tests)
svc_prob = svc.best_estimator_.predict_proba(X_tests)
svc_prob
```

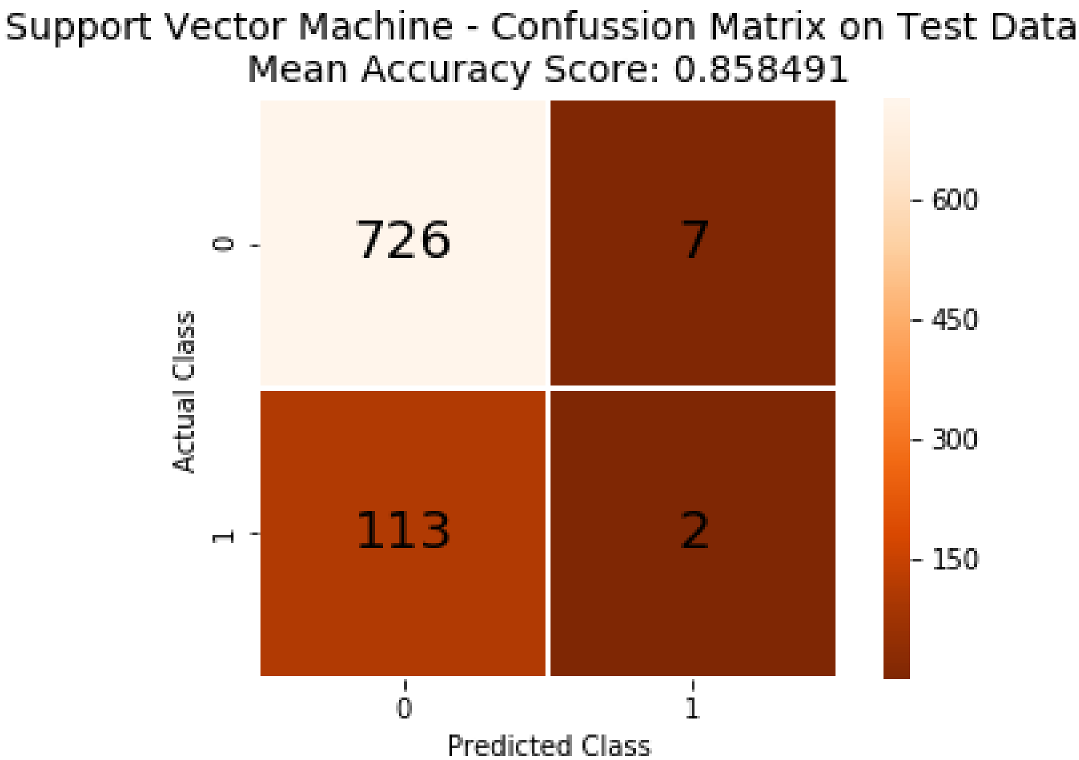
[196]:
```
array([[0.85686222, 0.14313778],
       [0.85551473, 0.14448527],
       [0.87157328, 0.12842672],
       …,
       [0.85650704, 0.14349296],
       [0.86263254, 0.13736746],
       [0.85457594, 0.14542406]])
```

[197]:
```
svc_matrix = metrics.confusion_matrix(y_test, svc_pred)
svc_matrix
```

[197]:
```
array([[726,   7],
       [113,   2]])
```

[198]:
```
svc_test = svc.best_estimator_.score(X_tests, y_test)

svc_matrix = metrics.confusion_matrix(y_test, svc_pred)

svc_cm = pd.DataFrame(svc_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (svc_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

svc_title = 'Support Vector Machine - Confussion Matrix on Test Data \nMean
 ↪Accuracy Score: {0:2f}'.format(svc_test)
plt.title(svc_title, size = 14)
```

```
# plt.figure(figsize=(16, 26))
plt.show;
```



[199]:
```
print("", classification_report(y_test, svc_pred, target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years       0.87      0.99      0.92       733
  Risky in 10 years       0.22      0.02      0.03       115

           accuracy                           0.86       848
          macro avg       0.54      0.50      0.48       848
       weighted avg       0.78      0.86      0.80       848
```

[200]:
```
acc_svc = accuracy_score(y_test, svc_pred)
print("Support Vector Machine accuracy:", acc_svc)
```

```
Support Vector Machine accuracy: 0.8584905660377359
```

[201]:
```
error_svc = 1-acc_svc
error_svc
```

[201]: 0.14150943396226412

[202]:
```
svc_probs = svc.best_estimator_.predict_proba(X_tests)[:,1]
print(roc_auc_score(y_test, svc_probs))
```

```
0.6050180912272376
```

12.0.1 For training set:

[203]:
```
svc_pred_tr = svc.best_estimator_.predict(X_trains)
svc_prob_tr = svc.best_estimator_.predict_proba(X_trains)
svc_prob_tr
```

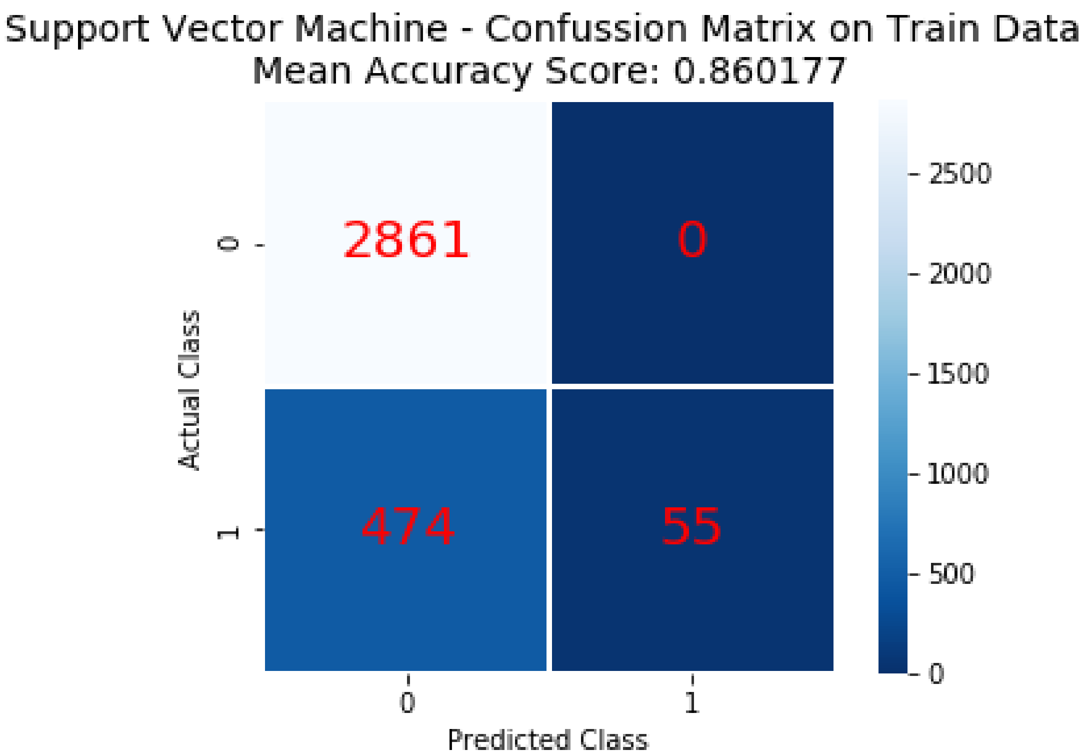
[203]:
```
array([[0.85562618, 0.14437382],
       [0.80583632, 0.19416368],
       [0.86189295, 0.13810705],
       …,
       [0.86254203, 0.13745797],
       [0.86172249, 0.13827751],
       [0.85708549, 0.14291451]])
```

[204]:
```
svc_matrix_tr = metrics.confusion_matrix(y_train, svc_pred_tr)
svc_matrix_tr
```

[204]:
```
array([[2861,    0],
       [ 474,   55]])
```

[205]:
```
svc_train = svc.best_estimator_.score(X_trains, y_train)

svc_matrix = metrics.confusion_matrix(y_train, svc_pred_tr)

svc_cm_tr = pd.DataFrame(svc_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (svc_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
 ↪5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

svc_title = 'Support Vector Machine - Confussion Matrix on Train Data \nMean␣
 ↪Accuracy Score: {0:2f}'.format(svc_train)
```

```
plt.title(svc_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



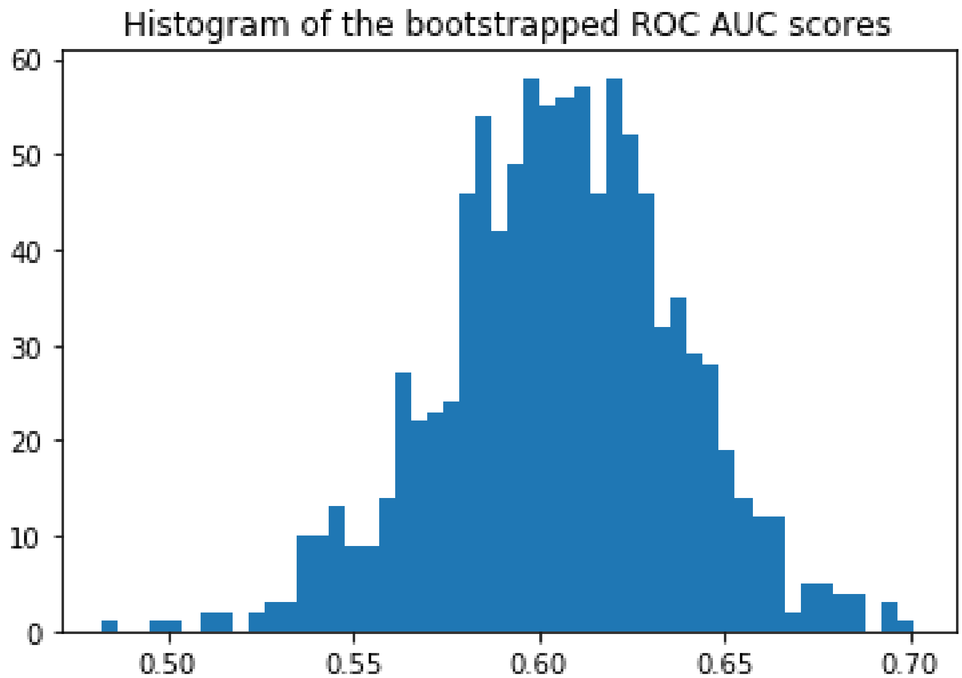
[206]:
```
print("", classification_report(y_train, svc_pred_tr,␣
↪target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years       0.86      1.00      0.92      2861
  Risky in 10 years       1.00      0.10      0.19       529

           accuracy                           0.86      3390
          macro avg       0.93      0.55      0.56      3390
       weighted avg       0.88      0.86      0.81      3390
```

[207]:
```
y_pred = svc_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))
```

```python
n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.6050
```




```
Confidence interval for the score: [0.5508 - 0.6572]
```

[208]:
```
alpha = .95
y_pred = svc_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.6050180912272376
```

```
AUC COV: 0.0010130072142994942
95% AUC CI: [0.5426368  0.66739938]
```

[ ]:

13 AdaBoost

[209]:
```
from sklearn.ensemble import AdaBoostClassifier
params5 = {'n_estimators':list(range(100, 1000, 200)),
           'learning_rate':[i/10.0 for i in range(1,10,3)]
           }

# create AdaBoostClassifier model
ada = AdaBoostClassifier(algorithm='SAMME.R', random_state=42)

# create gridsearch object with various combinations of parameters
ada = GridSearchCV(ada, params5, cv = 5,
                   refit = True,
                   n_jobs=-1, verbose = 5)

ada.fit(X_train, y_train)
ada.cv_results_
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:   27.6s
[Parallel(n_jobs=-1)]: Done  68 out of  75 | elapsed:   43.2s remaining:    4.4s
[Parallel(n_jobs=-1)]: Done  75 out of  75 | elapsed:   47.0s finished
/usr/local/lib/python3.7/site-packages/sklearn/utils/validation.py:724:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[209]:
```
{'mean_fit_time': array([ 0.96570077,  2.9743022 ,  5.29625077,  7.8960597 ,
10.75794311,
         1.166009  ,  3.68204541,  6.31340799,  9.17034268, 12.04604778,
         1.40285521,  3.99481645,  6.86136665,  8.84434986,  8.99486394]),
 'std_fit_time': array([0.00729729, 0.01508428, 0.1494108 , 0.16209918,
0.19594145,
        0.01450642, 0.11054166, 0.02383834, 0.12593654, 0.10008126,
        0.13866862, 0.05712087, 0.09384066, 0.30559129, 0.62561053]),
 'mean_score_time': array([0.11316004, 0.36205149, 0.64695463, 0.9455893 ,
1.27889209,
        0.12820139, 0.42560201, 0.72134862, 1.12053103, 1.42895179,
```

```
       0.15473967, 0.41867132, 0.7403975 , 0.78567972, 0.61816659]),
 'std_score_time': array([0.00315797, 0.00600744, 0.04015234, 0.07357845,
0.04352772,
       0.00635117, 0.04007826, 0.03024043, 0.1097754 , 0.08435108,
       0.01780479, 0.04623695, 0.05975559, 0.19250361, 0.06729245]),
 'param_learning_rate': masked_array(data=[0.1, 0.1, 0.1, 0.1, 0.1, 0.4, 0.4,
0.4, 0.4, 0.4, 0.7,
                   0.7, 0.7, 0.7, 0.7],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_n_estimators': masked_array(data=[100, 300, 500, 700, 900, 100, 300,
500, 700, 900, 100,
                   300, 500, 700, 900],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'learning_rate': 0.1, 'n_estimators': 100},
  {'learning_rate': 0.1, 'n_estimators': 300},
  {'learning_rate': 0.1, 'n_estimators': 500},
  {'learning_rate': 0.1, 'n_estimators': 700},
  {'learning_rate': 0.1, 'n_estimators': 900},
  {'learning_rate': 0.4, 'n_estimators': 100},
  {'learning_rate': 0.4, 'n_estimators': 300},
  {'learning_rate': 0.4, 'n_estimators': 500},
  {'learning_rate': 0.4, 'n_estimators': 700},
  {'learning_rate': 0.4, 'n_estimators': 900},
  {'learning_rate': 0.7, 'n_estimators': 100},
  {'learning_rate': 0.7, 'n_estimators': 300},
  {'learning_rate': 0.7, 'n_estimators': 500},
  {'learning_rate': 0.7, 'n_estimators': 700},
  {'learning_rate': 0.7, 'n_estimators': 900}],
 'split0_test_score': array([0.84977909, 0.84683358, 0.84388807, 0.84094256,
0.84094256,
       0.84094256, 0.84094256, 0.83946981, 0.83357879, 0.83357879,
       0.84388807, 0.83505155, 0.83357879, 0.82621502, 0.82179676]),
 'split1_test_score': array([0.84660767, 0.85103245, 0.8539823 , 0.85693215,
0.85545723,
       0.85250737, 0.85840708, 0.85988201, 0.85545723, 0.85545723,
       0.85250737, 0.8539823 , 0.85250737, 0.84955752, 0.84955752]),
 'split2_test_score': array([0.84660767, 0.84955752, 0.85693215, 0.85693215,
0.8539823 ,
       0.85103245, 0.8480826 , 0.8480826 , 0.8480826 , 0.84660767,
       0.84660767, 0.84218289, 0.84365782, 0.84218289, 0.83628319]),
 'split3_test_score': array([0.84365782, 0.83628319, 0.83775811, 0.83775811,
```

```
0.83480826,
        0.84070796, 0.83185841, 0.83185841, 0.83333333, 0.83038348,
        0.83185841, 0.83480826, 0.83185841, 0.82448378, 0.82300885]),
 'split4_test_score': array([0.8478582 , 0.84342688, 0.84638109, 0.84047267,
0.84194978,
        0.84490399, 0.84342688, 0.84194978, 0.83604136, 0.83604136,
        0.84342688, 0.83604136, 0.83308715, 0.83161004, 0.82717873]),
 'mean_test_score': array([0.84690265, 0.84542773, 0.84778761, 0.84660767,
0.84542773,
        0.8460177 , 0.84454277, 0.84424779, 0.84129794, 0.84041298,
        0.84365782, 0.84041298, 0.83893805, 0.83480826, 0.83156342]),
 'std_test_score': array([0.00199503, 0.00525486, 0.0069252 , 0.00849974,
0.0079846 ,
        0.00494955, 0.00871627, 0.00938901, 0.00891309, 0.00928604,
        0.00672828, 0.00730126, 0.00799338, 0.00961964, 0.01033412]),
 'rank_test_score': array([ 2,  5,  1,  3,  5,  4,  7,  8, 10, 11,  9, 11, 13,
14, 15],
       dtype=int32)}
```

[210]:
```
ada.best_estimator_
```

[210]:
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=0.1,
                   n_estimators=500, random_state=42)
```

[211]:
```
ada_pred = ada.best_estimator_.predict(X_test)
ada_prob = ada.best_estimator_.predict_proba(X_test)
ada_prob
```

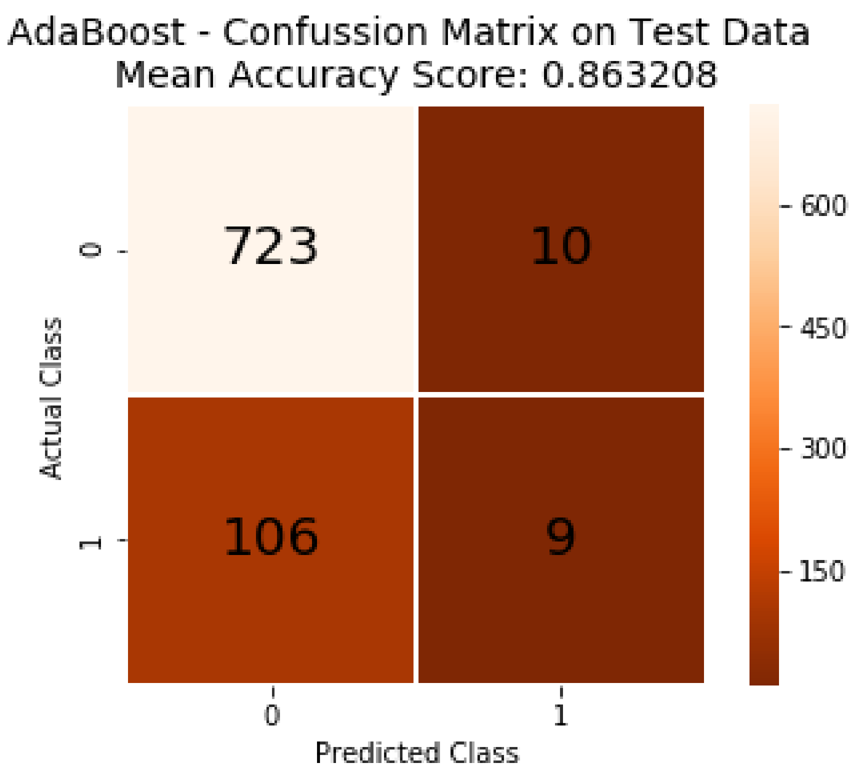
[211]:
```
array([[0.50877679, 0.49122321],
       [0.51479466, 0.48520534],
       [0.51709261, 0.48290739],
       …,
       [0.50815869, 0.49184131],
       [0.50682996, 0.49317004],
       [0.5116839 , 0.4883161 ]])
```

[212]:
```
ada_matrix = metrics.confusion_matrix(y_test, ada_pred)
ada_matrix
```

[212]:
```
array([[723,  10],
       [106,   9]])
```

[213]:
```
ada_test = ada.best_estimator_.score(X_test, y_test)

ada_matrix = metrics.confusion_matrix(y_test, ada_pred)

ada_cm = pd.DataFrame(ada_matrix, range(2), range(2))
```

```
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (ada_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

ada_title = 'AdaBoost - Confussion Matrix on Test Data \nMean Accuracy Score:␣
↪{0:2f}'.format(ada_test)
plt.title(ada_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



[214]:
```
print("", classification_report(y_test, ada_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support
```

```
No risk in 10 years       0.87      0.99      0.93       733
  Risky in 10 years       0.47      0.08      0.13       115

           accuracy                           0.86       848
          macro avg       0.67      0.53      0.53       848
       weighted avg       0.82      0.86      0.82       848
```

[215]:
```
acc_ada = accuracy_score(y_test, ada_pred)
print("AdaBoost accuracy:", acc_ada)
```

```
AdaBoost accuracy: 0.8632075471698113
```

[216]:
```
error_ada = 1-acc_ada
error_ada
```

[216]:
```
0.1367924528301887
```

[217]:
```
ada_probs = ada.best_estimator_.predict_proba(X_test)[:,1]
print(roc_auc_score(y_test, ada_probs))
```

```
0.7042173319888486
```

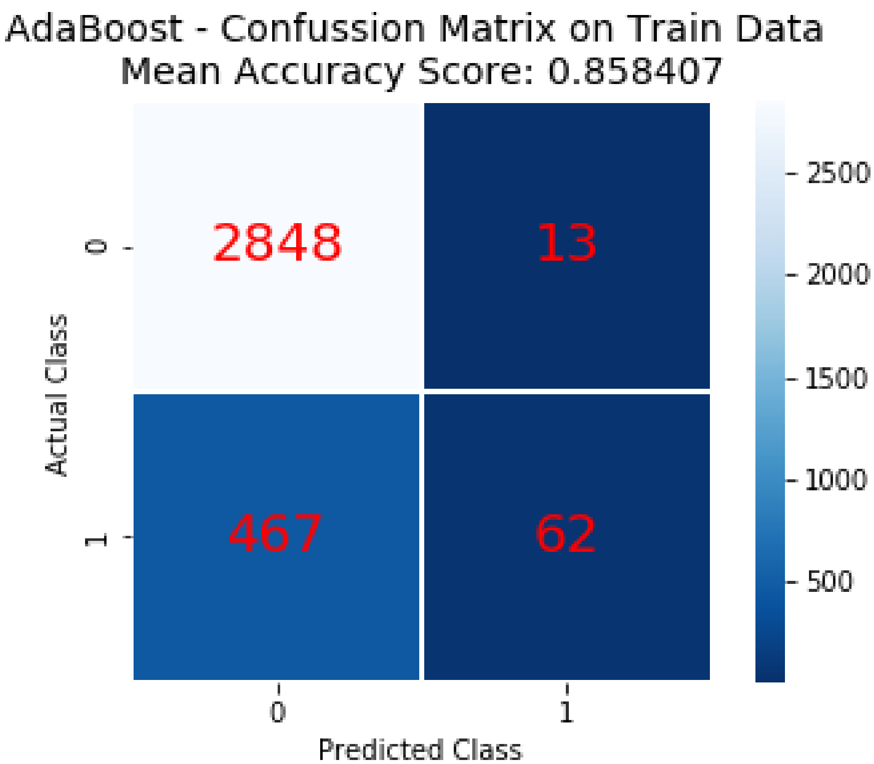
13.0.1 For training set:

[218]:
```
ada_pred_tr = ada.best_estimator_.predict(X_train)
ada_prob_tr = ada.best_estimator_.predict_proba(X_train)
ada_prob_tr
```

[218]:
```
array([[0.51194418, 0.48805582],
       [0.51267724, 0.48732276],
       [0.51247983, 0.48752017],
       …,
       [0.50812342, 0.49187658],
       [0.51184831, 0.48815169],
       [0.5202282 , 0.4797718 ]])
```

[219]:
```
ada_train = ada.best_estimator_.score(X_train, y_train)
ada_matrix = metrics.confusion_matrix(y_train, ada_pred_tr)

ada_cm_tr = pd.DataFrame(ada_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (ada_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
↪5,annot_kws=akws)
```

```
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

ada_title = 'AdaBoost - Confussion Matrix on Train Data \nMean Accuracy Score:␣
↪{0:2f}'.format(ada_train)
plt.title(ada_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



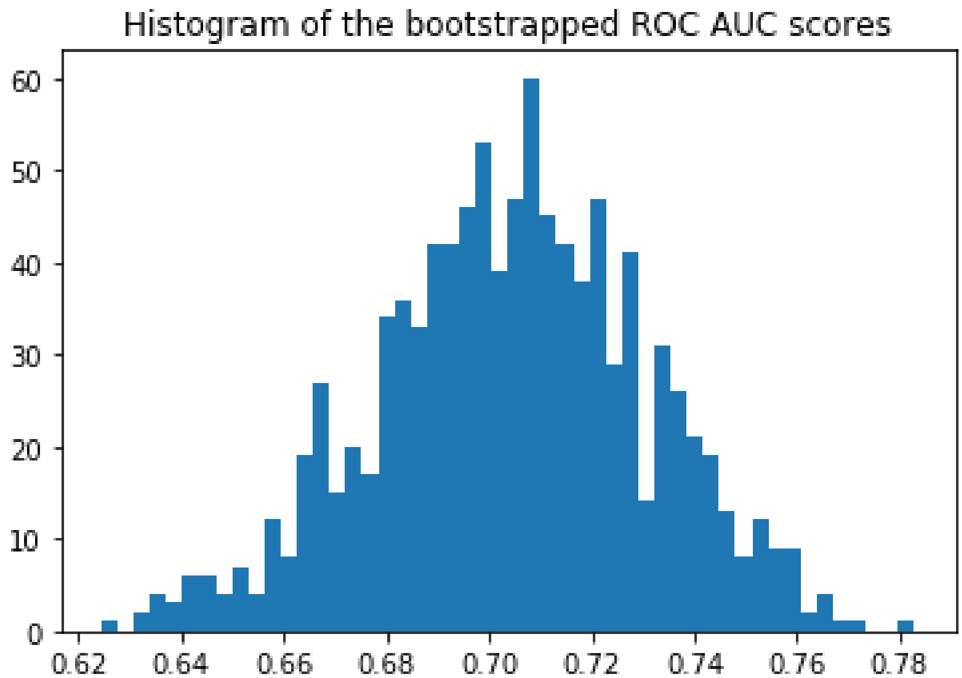
[220]:
```
print("", classification_report(y_train, ada_pred_tr,␣
↪target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years       0.86      1.00      0.92      2861
  Risky in 10 years       0.83      0.12      0.21       529

           accuracy                           0.86      3390
          macro avg       0.84      0.56      0.56      3390
```

```
weighted avg       0.85      0.86      0.81      3390
```

[221]:
```
y_pred = ada_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.7042
```




```
Confidence interval for the score: [0.6594 - 0.7474]
```

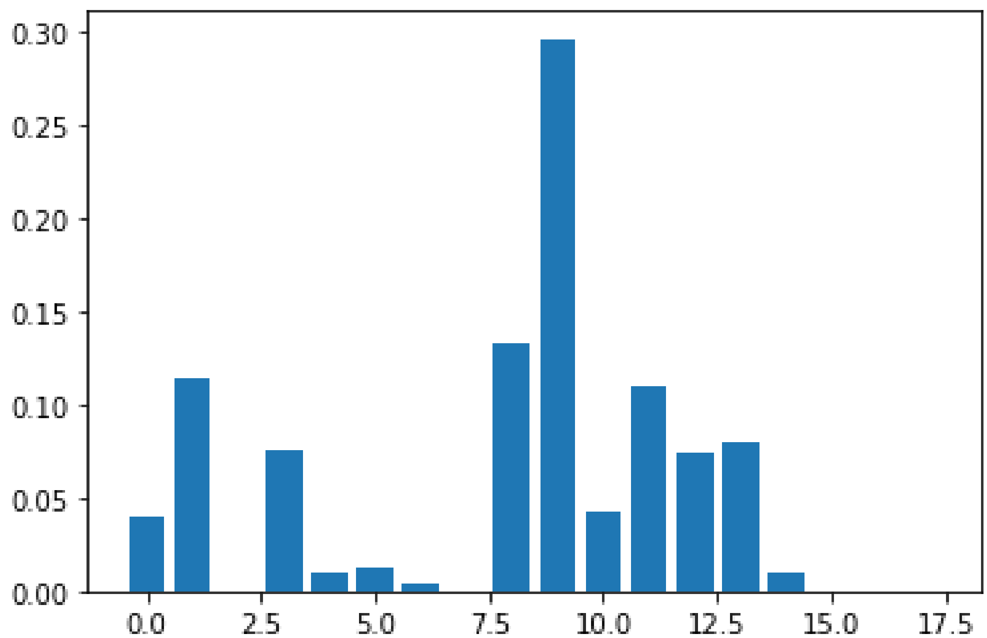
[222]:
```python
alpha = .95
y_pred = ada_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.7042173319888486
```

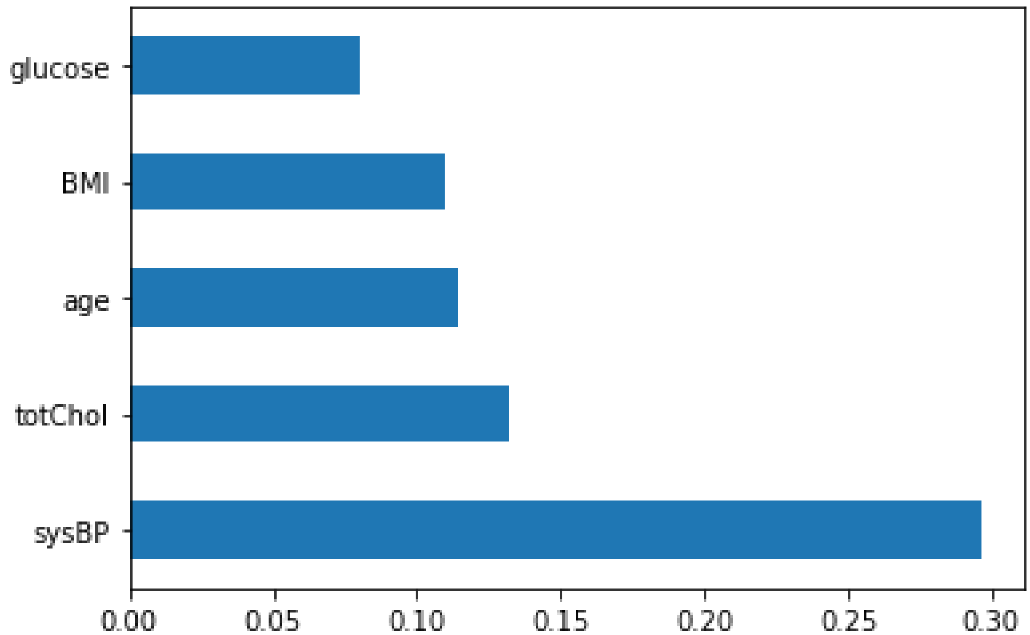
```
AUC COV: 0.00072496717284027743
95% AUC CI: [0.65144488 0.75698978]
```

[223]:
```
print(ada.best_estimator_.feature_importances_)
pyplot.bar(range(len(ada.best_estimator_.feature_importances_)), ada.
 ↪best_estimator_.feature_importances_)
pyplot.show()
```

```
[0.04  0.114 0.    0.076 0.01  0.012 0.004 0.    0.132 0.296 0.042 0.11
 0.074 0.08  0.01  0.    0.    0.   ]
```



[224]:
```
feat_importances_ada = pd.Series(ada.best_estimator_.feature_importances_,␣
 ↪index=X_train.columns)
feat_importances_ada.nlargest(5).plot(kind='barh')
pyplot.show()
```



14 GradientBoosting

[225]:
```
from sklearn.ensemble import GradientBoostingClassifier

# create a dictionary of parameters
params6 = {'n_estimators':list(range(100, 1000, 200)),
           'learning_rate':[i/10.0 for i in range(1,11,2)],
           'max_depth':[1,2,3]
           }

# create AdaBoostClassifier model
gb = GradientBoostingClassifier(random_state=1)

# create gridsearch object with various combinations of parameters
gb = GridSearchCV(gb, params6, cv = 5,
                  refit = True,
                  n_jobs=-1, verbose = 5)

gb.fit(X_train, y_train)
gb.cv_results_
```

```
Fitting 5 folds for each of 75 candidates, totalling 375 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
```

```
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed:   16.5s
[Parallel(n_jobs=-1)]: Done 264 tasks      | elapsed:   31.2s
[Parallel(n_jobs=-1)]: Done 375 out of 375 | elapsed:   47.4s finished
/usr/local/lib/python3.7/site-
packages/sklearn/ensemble/gradient_boosting.py:1450: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[225]:
```
{'mean_fit_time': array([0.15932169, 0.51340604, 0.83827376, 1.10519271,
1.34034624,
        0.26741924, 0.70358062, 1.03688641, 1.5052496 , 1.93516083,
        0.35458431, 0.99735265, 1.65355182, 2.43619099, 3.26404276,
        0.15755653, 0.43520694, 0.68633413, 0.98180213, 1.47697859,
        0.33247085, 0.90852251, 1.50417185, 1.78113742, 2.08511586,
        0.36102152, 1.06007137, 1.80281372, 2.85695381, 3.82170115,
        0.18600793, 0.57247295, 0.81294279, 1.06086202, 1.54588971,
        0.33936734, 1.02573857, 1.72435565, 2.18204236, 2.48226957,
        0.42915139, 1.16524863, 1.90141315, 2.65990334, 3.53348656,
        0.14902892, 0.42939124, 0.71822357, 1.09126358, 1.44154525,
        0.28175373, 0.87885818, 1.60759006, 2.09863353, 2.46082029,
        0.41032624, 1.2267354 , 2.07013812, 3.42723351, 5.11536651,
        0.20697284, 0.84601502, 1.42862177, 1.45463967, 1.74582486,
        0.30271635, 1.08499827, 1.70638828, 1.9892416 , 2.37889857,
        0.39186654, 1.21237221, 1.9871244 , 2.48078418, 2.31568527]),
 'std_fit_time': array([0.00932541, 0.01862688, 0.00523729, 0.00969145,
0.05407049,
        0.00320971, 0.02244226, 0.02104429, 0.01117783, 0.03241585,
        0.05411071, 0.00599617, 0.01384934, 0.02229602, 0.08351019,
        0.01832026, 0.00561193, 0.01349506, 0.03093726, 0.05456189,
        0.04080836, 0.03550175, 0.12692456, 0.06480497, 0.04904323,
        0.00703947, 0.00869911, 0.01647918, 0.03440889, 0.08169081,
        0.04116687, 0.00760689, 0.02637518, 0.01935946, 0.07594568,
        0.04118855, 0.00725483, 0.02057306, 0.04453428, 0.11740404,
        0.04279674, 0.00787189, 0.00588774, 0.01598807, 0.05386302,
        0.00550055, 0.0046857 , 0.0175974 , 0.03016694, 0.04264409,
        0.00457029, 0.04365303, 0.0256948 , 0.05221093, 0.08615889,
        0.007263  , 0.05160618, 0.04901945, 0.01935512, 0.10785388,
        0.01829631, 0.20853611, 0.04723601, 0.11206006, 0.0485247 ,
        0.01445916, 0.01896841, 0.1074698 , 0.10575348, 0.13936591,
        0.01741626, 0.10743542, 0.22072054, 0.23278165, 0.30960406]),
 'mean_score_time': array([0.00744805, 0.00916553, 0.01100001, 0.00820446,
0.00853496,
        0.00577703, 0.00608716, 0.00734153, 0.00938401, 0.01115155,
        0.0044611 , 0.00705876, 0.00935316, 0.01559248, 0.01830459,
        0.00430937, 0.0057672 , 0.00593667, 0.00867639, 0.01101942,
```

```
       0.00831604, 0.00756283, 0.00947242, 0.01103773, 0.01320806,
       0.00495963, 0.00843182, 0.01025901, 0.01693745, 0.01809564,
       0.00592623, 0.00834098, 0.00693426, 0.01018686, 0.01466064,
       0.00762806, 0.00977039, 0.0112227 , 0.01103096, 0.01343818,
       0.00586638, 0.00818334, 0.01102428, 0.01335292, 0.01808167,
       0.00504689, 0.00614691, 0.00733757, 0.00796304, 0.01012211,
       0.00499778, 0.01248446, 0.01517501, 0.01518536, 0.01498241,
       0.00521007, 0.00848484, 0.01343236, 0.02395706, 0.02365994,
       0.00454726, 0.00745502, 0.01248918, 0.01213865, 0.01312609,
       0.00661225, 0.00929027, 0.01218262, 0.01117978, 0.01272974,
       0.00516658, 0.00972438, 0.01083155, 0.01022677, 0.00887823]),
 'std_score_time': array([0.00673419, 0.00378984, 0.00265367, 0.00125148,
0.00219411,
       0.00207801, 0.00071409, 0.00039618, 0.00168016, 0.00078405,
       0.00032012, 0.00103873, 0.00089124, 0.0059284 , 0.00567625,
       0.00077401, 0.00051479, 0.00044606, 0.00239484, 0.00305683,
       0.00439829, 0.00173066, 0.00166166, 0.00111226, 0.00178781,
       0.00022671, 0.00116555, 0.00078191, 0.0035154 , 0.00269004,
       0.00149775, 0.00421694, 0.00109258, 0.00279101, 0.00648322,
       0.00385055, 0.00283196, 0.00286915, 0.00111944, 0.00126616,
       0.00132168, 0.00102209, 0.00095618, 0.00029636, 0.00133468,
       0.0018938 , 0.00110186, 0.00116855, 0.00021689, 0.00104608,
       0.00053116, 0.00672503, 0.00336201, 0.00261463, 0.0013424 ,
       0.00034567, 0.0009441 , 0.00300777, 0.01090853, 0.00550804,
       0.00036978, 0.00308088, 0.00639168, 0.00371629, 0.00284051,
       0.00201865, 0.00200141, 0.00211525, 0.00199792, 0.00074285,
       0.00051302, 0.0014475 , 0.00136875, 0.00086275, 0.00146473]),
 'param_learning_rate': masked_array(data=[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
0.1, 0.1, 0.1, 0.1,
                   0.1, 0.1, 0.1, 0.1, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3,
                   0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.5, 0.5, 0.5,
                   0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
                   0.5, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7,
                   0.7, 0.7, 0.7, 0.7, 0.7, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9,
                   0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object),
```

```
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3,
3, 1, 1, 1,
                    1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 2,
                    2, 2, 2, 2, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 2, 2, 2, 2,
                    2, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3,
                    3, 3, 3],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[100, 300, 500, 700, 900, 100, 300,
500, 700, 900, 100,
                    300, 500, 700, 900, 100, 300, 500, 700, 900, 100, 300,
                    500, 700, 900, 100, 300, 500, 700, 900, 100, 300, 500,
                    700, 900, 100, 300, 500, 700, 900, 100, 300, 500, 700,
                    900, 100, 300, 500, 700, 900, 100, 300, 500, 700, 900,
                    100, 300, 500, 700, 900, 100, 300, 500, 700, 900, 100,
                    300, 500, 700, 900, 100, 300, 500, 700, 900],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100},
  {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 300},
  {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500},
  {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700},
  {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 900},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 300},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 500},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700},
```

```
{'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 900},
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100},
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300},
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500},
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700},
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 900},
{'learning_rate': 0.3, 'max_depth': 1, 'n_estimators': 100},
{'learning_rate': 0.3, 'max_depth': 1, 'n_estimators': 300},
{'learning_rate': 0.3, 'max_depth': 1, 'n_estimators': 500},
{'learning_rate': 0.3, 'max_depth': 1, 'n_estimators': 700},
{'learning_rate': 0.3, 'max_depth': 1, 'n_estimators': 900},
{'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 100},
{'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 300},
{'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 500},
{'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 700},
{'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 900},
{'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 100},
{'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 300},
{'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 500},
{'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 700},
{'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 900},
{'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 100},
{'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 300},
{'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 500},
{'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 700},
{'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 900},
{'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 100},
{'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 300},
{'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 500},
{'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 700},
{'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 900},
{'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 100},
{'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 300},
{'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 500},
{'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 700},
{'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 900},
{'learning_rate': 0.7, 'max_depth': 1, 'n_estimators': 100},
{'learning_rate': 0.7, 'max_depth': 1, 'n_estimators': 300},
{'learning_rate': 0.7, 'max_depth': 1, 'n_estimators': 500},
{'learning_rate': 0.7, 'max_depth': 1, 'n_estimators': 700},
{'learning_rate': 0.7, 'max_depth': 1, 'n_estimators': 900},
{'learning_rate': 0.7, 'max_depth': 2, 'n_estimators': 100},
{'learning_rate': 0.7, 'max_depth': 2, 'n_estimators': 300},
{'learning_rate': 0.7, 'max_depth': 2, 'n_estimators': 500},
{'learning_rate': 0.7, 'max_depth': 2, 'n_estimators': 700},
{'learning_rate': 0.7, 'max_depth': 2, 'n_estimators': 900},
{'learning_rate': 0.7, 'max_depth': 3, 'n_estimators': 100},
```

```
  {'learning_rate': 0.7, 'max_depth': 3, 'n_estimators': 300},
  {'learning_rate': 0.7, 'max_depth': 3, 'n_estimators': 500},
  {'learning_rate': 0.7, 'max_depth': 3, 'n_estimators': 700},
  {'learning_rate': 0.7, 'max_depth': 3, 'n_estimators': 900},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 100},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 300},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 500},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 700},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 900},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 300},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 500},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 700},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 900},
  {'learning_rate': 0.9, 'max_depth': 3, 'n_estimators': 100},
  {'learning_rate': 0.9, 'max_depth': 3, 'n_estimators': 300},
  {'learning_rate': 0.9, 'max_depth': 3, 'n_estimators': 500},
  {'learning_rate': 0.9, 'max_depth': 3, 'n_estimators': 700},
  {'learning_rate': 0.9, 'max_depth': 3, 'n_estimators': 900}],
 'split0_test_score': array([0.84977909, 0.84683358, 0.84830633, 0.84977909,
0.84830633,
        0.84388807, 0.83063328, 0.83210604, 0.82768778, 0.82179676,
        0.84388807, 0.83210604, 0.82621502, 0.81885125, 0.81443299,
        0.84683358, 0.84830633, 0.84241532, 0.83799705, 0.83799705,
        0.83505155, 0.81590574, 0.80559647, 0.80265096, 0.79086892,
        0.82768778, 0.81590574, 0.81443299, 0.81443299, 0.81296024,
        0.84241532, 0.83799705, 0.83505155, 0.8365243 , 0.83210604,
        0.82621502, 0.81296024, 0.79528719, 0.78939617, 0.78792342,
        0.7982327 , 0.78645066, 0.79381443, 0.79086892, 0.79086892,
        0.84241532, 0.8365243 , 0.8365243 , 0.83063328, 0.83357879,
        0.82474227, 0.7982327 , 0.78055965, 0.77466863, 0.77908689,
        0.77761414, 0.77908689, 0.78497791, 0.79675994, 0.79234168,
        0.83946981, 0.83357879, 0.82916053, 0.82916053, 0.82326951,
        0.80265096, 0.77319588, 0.76435935, 0.76877761, 0.75994109,
        0.77908689, 0.79381443, 0.79528719, 0.78939617, 0.79234168]),
 'split1_test_score': array([0.84660767, 0.8539823 , 0.85545723, 0.85545723,
0.85545723,
        0.84513274, 0.84070796, 0.84070796, 0.83628319, 0.83775811,
        0.84218289, 0.82890855, 0.820059  , 0.81710914, 0.81415929,
        0.85103245, 0.8539823 , 0.85545723, 0.85545723, 0.8539823 ,
        0.8480826 , 0.83480826, 0.82448378, 0.81710914, 0.820059  ,
        0.84070796, 0.82448378, 0.81858407, 0.81415929, 0.81710914,
        0.85545723, 0.8539823 , 0.85545723, 0.84955752, 0.84660767,
        0.83480826, 0.820059  , 0.81710914, 0.80530973, 0.80678466,
        0.82300885, 0.81858407, 0.81563422, 0.820059  , 0.81563422,
        0.8539823 , 0.8539823 , 0.85103245, 0.8480826 , 0.84955752,
        0.83185841, 0.820059  , 0.81268437, 0.80530973, 0.81120944,
```

```
        0.80678466, 0.79941003, 0.80678466, 0.82153392, 0.81563422,
        0.8539823 , 0.8480826 , 0.84955752, 0.8480826 , 0.84513274,
        0.81710914, 0.79646018, 0.79646018, 0.79646018, 0.79646018,
        0.81120944, 0.80235988, 0.81415929, 0.80973451, 0.80825959]),
 'split2_test_score': array([0.84513274, 0.85250737, 0.8539823 , 0.85250737,
0.85103245,
        0.8480826 , 0.84955752, 0.85103245, 0.84955752, 0.84218289,
        0.85250737, 0.84365782, 0.84660767, 0.84365782, 0.84513274,
        0.8539823 , 0.85103245, 0.8539823 , 0.85250737, 0.85103245,
        0.8480826 , 0.84070796, 0.82890855, 0.82448378, 0.83038348,
        0.85103245, 0.83628319, 0.83185841, 0.82743363, 0.83333333,
        0.85250737, 0.85693215, 0.85545723, 0.85250737, 0.85250737,
        0.84218289, 0.84365782, 0.83038348, 0.8259587 , 0.82448378,
        0.83185841, 0.81710914, 0.81710914, 0.81563422, 0.8259587 ,
        0.85250737, 0.8539823 , 0.85250737, 0.85103245, 0.8480826 ,
        0.83480826, 0.80973451, 0.82448378, 0.81710914, 0.81858407,
        0.79646018, 0.80383481, 0.81563422, 0.82153392, 0.82153392,
        0.84955752, 0.85250737, 0.85103245, 0.8480826 , 0.84660767,
        0.81710914, 0.81120944, 0.80383481, 0.80530973, 0.80678466,
        0.80530973, 0.81120944, 0.80235988, 0.81268437, 0.81268437]),
 'split3_test_score': array([0.84218289, 0.83923304, 0.83628319, 0.83333333,
0.82890855,
        0.83480826, 0.83480826, 0.83333333, 0.83333333, 0.83038348,
        0.83480826, 0.83628319, 0.8259587 , 0.82448378, 0.82153392,
        0.83628319, 0.83038348, 0.82743363, 0.82743363, 0.83038348,
        0.83038348, 0.82153392, 0.80825959, 0.79351032, 0.79498525,
        0.81858407, 0.81415929, 0.80678466, 0.80973451, 0.81563422,
        0.83185841, 0.82890855, 0.83333333, 0.82890855, 0.8259587 ,
        0.82890855, 0.80383481, 0.79351032, 0.79941003, 0.79056047,
        0.80678466, 0.80530973, 0.79941003, 0.80235988, 0.7920354 ,
        0.83185841, 0.82890855, 0.82448378, 0.82300885, 0.820059  ,
        0.82890855, 0.80088496, 0.78466077, 0.78761062, 0.78761062,
        0.79351032, 0.78761062, 0.7920354 , 0.78908555, 0.79056047,
        0.82890855, 0.8259587 , 0.81858407, 0.81858407, 0.81858407,
        0.81120944, 0.7920354 , 0.78023599, 0.79056047, 0.78466077,
        0.81120944, 0.79351032, 0.7920354 , 0.78023599, 0.78613569]),
 'split4_test_score': array([0.8478582 , 0.84194978, 0.84342688, 0.84047267,
0.84194978,
        0.84047267, 0.83604136, 0.82865583, 0.82570162, 0.82274742,
        0.83899557, 0.81979321, 0.81093058, 0.80649926, 0.79763663,
        0.84047267, 0.83751846, 0.83751846, 0.84342688, 0.84194978,
        0.83604136, 0.816839  , 0.80649926, 0.80206795, 0.788774  ,
        0.81979321, 0.79911374, 0.78138848, 0.79025111, 0.78138848,
        0.84047267, 0.84194978, 0.84342688, 0.84047267, 0.83604136,
        0.82422452, 0.79615953, 0.79468242, 0.77991137, 0.79025111,
        0.79615953, 0.7872969 , 0.78434269, 0.79025111, 0.78434269,
        0.83751846, 0.84194978, 0.83751846, 0.82570162, 0.82422452,
```

```
        0.82570162, 0.79025111, 0.78286558, 0.78286558, 0.77991137,
        0.80502216, 0.77695716, 0.77991137, 0.77548006, 0.78581979,
        0.83899557, 0.83456425, 0.83161004, 0.82274742, 0.82127031,
        0.81536189, 0.81240768, 0.80797637, 0.80206795, 0.79911374,
        0.79025111, 0.79025111, 0.79320532, 0.79025111, 0.788774  ]),
 'mean_test_score': array([0.84631268, 0.84690265, 0.84749263, 0.84631268,
0.84513274,
        0.84247788, 0.83834808, 0.83716814, 0.83451327, 0.83097345,
        0.84247788, 0.83215339, 0.8259587 , 0.82212389, 0.81858407,
        0.84572271, 0.84424779, 0.84336283, 0.84336283, 0.84306785,
        0.83952802, 0.8259587 , 0.81474926, 0.8079646 , 0.80501475,
        0.83156342, 0.8179941 , 0.81061947, 0.81120944, 0.8120944 ,
        0.84454277, 0.8439528 , 0.84454277, 0.84159292, 0.83864307,
        0.83126844, 0.81533923, 0.80619469, 0.8       , 0.8       ,
        0.81120944, 0.80294985, 0.8020649 , 0.80383481, 0.80176991,
        0.84365782, 0.84306785, 0.84041298, 0.83569322, 0.83510324,
        0.82920354, 0.80383481, 0.79705015, 0.79351032, 0.79528024,
        0.79587021, 0.78938053, 0.79587021, 0.80088496, 0.80117994,
        0.84218289, 0.83893805, 0.8359882 , 0.83333333, 0.83097345,
        0.81268437, 0.79705015, 0.79056047, 0.79262537, 0.78938053,
        0.79941003, 0.79823009, 0.79941003, 0.79646018, 0.79764012]),
 'std_test_score': array([0.00256723, 0.00574207, 0.007048  , 0.00820426,
0.00921769,
        0.00454512, 0.00646018, 0.00796832, 0.00842664, 0.00804904,
        0.00589239, 0.00790553, 0.01171976, 0.01223864, 0.01541763,
        0.0065464 , 0.00888509, 0.01046702, 0.01012398, 0.00860758,
        0.00724143, 0.00999691, 0.00989163, 0.01121096, 0.01693674,
        0.01252602, 0.01226069, 0.01671519, 0.0120259 , 0.01691735,
        0.00853847, 0.01034324, 0.00954318, 0.00860738, 0.00965507,
        0.00651679, 0.01630679, 0.01494391, 0.01560744, 0.01396819,
        0.01399996, 0.01391231, 0.01264348, 0.01230636, 0.0160894 ,
        0.00852429, 0.00982716, 0.01035661, 0.01161901, 0.0120351 ,
        0.00376101, 0.01022162, 0.01802095, 0.01548944, 0.01645658,
        0.01041588, 0.01070399, 0.01339831, 0.01818507, 0.01449   ,
        0.00880153, 0.0098395 , 0.01248347, 0.01250565, 0.01226264,
        0.0054646 , 0.01436212, 0.01617397, 0.0129495 , 0.0163576 ,
        0.01273311, 0.00762748, 0.00819868, 0.01257876, 0.01075231]),
 'rank_test_score': array([ 3,  2,  1,  3,  6, 16, 24, 25, 29, 34, 16, 31, 37,
39, 40,  5,  9,
        12, 12, 14, 21, 37, 43, 49, 51, 32, 41, 48, 46, 45,  7, 10,  7, 19,
        23, 33, 42, 50, 59, 59, 46, 54, 55, 52, 56, 11, 14, 20, 27, 28, 36,
        52, 65, 71, 70, 68, 74, 68, 58, 57, 18, 22, 26, 30, 34, 44, 65, 73,
        72, 74, 61, 63, 61, 67, 64], dtype=int32)}
```

[226]:
```
gb.best_estimator_
```

[226]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=1,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=500,
                           n_iter_no_change=None, presort='auto',
                           random_state=1, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

[227]:
```
gb_pred = gb.best_estimator_.predict(X_test)
gb_prob = gb.best_estimator_.predict_proba(X_test)
gb_prob
```

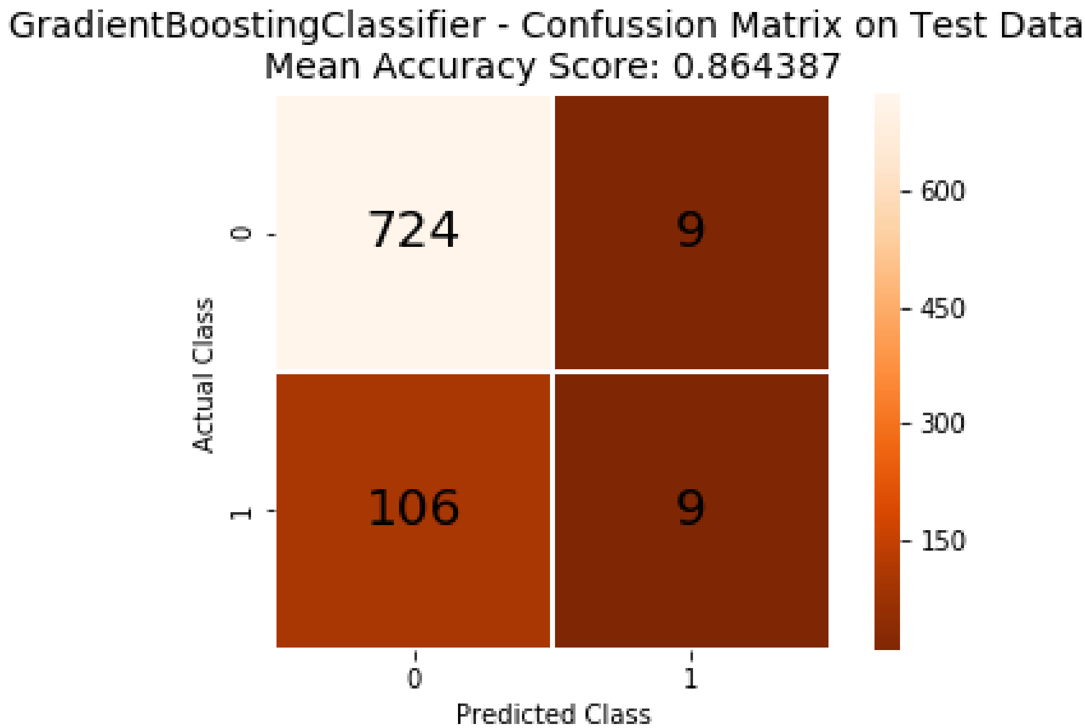
[227]:
```
array([[0.8544359 , 0.1455641 ],
       [0.95091299, 0.04908701],
       [0.9523758 , 0.0476242 ],
       …,
       [0.82844521, 0.17155479],
       [0.71349173, 0.28650827],
       [0.9136693 , 0.0863307 ]])
```

[228]:
```
gb_matrix = metrics.confusion_matrix(y_test, gb_pred)
gb_matrix
```

[228]:
```
array([[724,   9],
       [106,   9]])
```

[229]:
```
gb_test = gb.best_estimator_.score(X_test, y_test)
gb_matrix = metrics.confusion_matrix(y_test, gb_pred)
gb_cm = pd.DataFrame(gb_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (gb_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

gb_title = 'GradientBoostingClassifier - Confussion Matrix on Test Data \nMean␣
 ↪Accuracy Score: {0:2f}'.format(gb_test)
plt.title(gb_title, size = 14)
```

```
# plt.figure(figsize=(16, 26))
plt.show;
```



[230]:
```
print("", classification_report(y_test, gb_pred, target_names=target_names))
```

```
                      precision    recall  f1-score   support

No risk in 10 years       0.87      0.99      0.93       733
  Risky in 10 years       0.50      0.08      0.14       115

           accuracy                           0.86       848
          macro avg       0.69      0.53      0.53       848
       weighted avg       0.82      0.86      0.82       848
```

[231]:
```
acc_gb = accuracy_score(y_test, gb_pred)
print("Gradient Boosting Classifier accuracy:", acc_gb)
```

```
Gradient Boosting Classifier accuracy: 0.8643867924528302
```

[232]:
```
error_gb = 1-acc_gb
error_gb
```

[232]: 0.13561320754716977

[233]:
```
gb_probs = gb.best_estimator_.predict_proba(X_test)[:,1]
print(roc_auc_score(y_test, gb_probs))
```

```
0.7127172430156
```

14.0.1 For training set:

[234]:
```
gb_pred_tr = gb.best_estimator_.predict(X_train)
gb_prob_tr = gb.best_estimator_.predict_proba(X_train)
gb_prob_tr
```

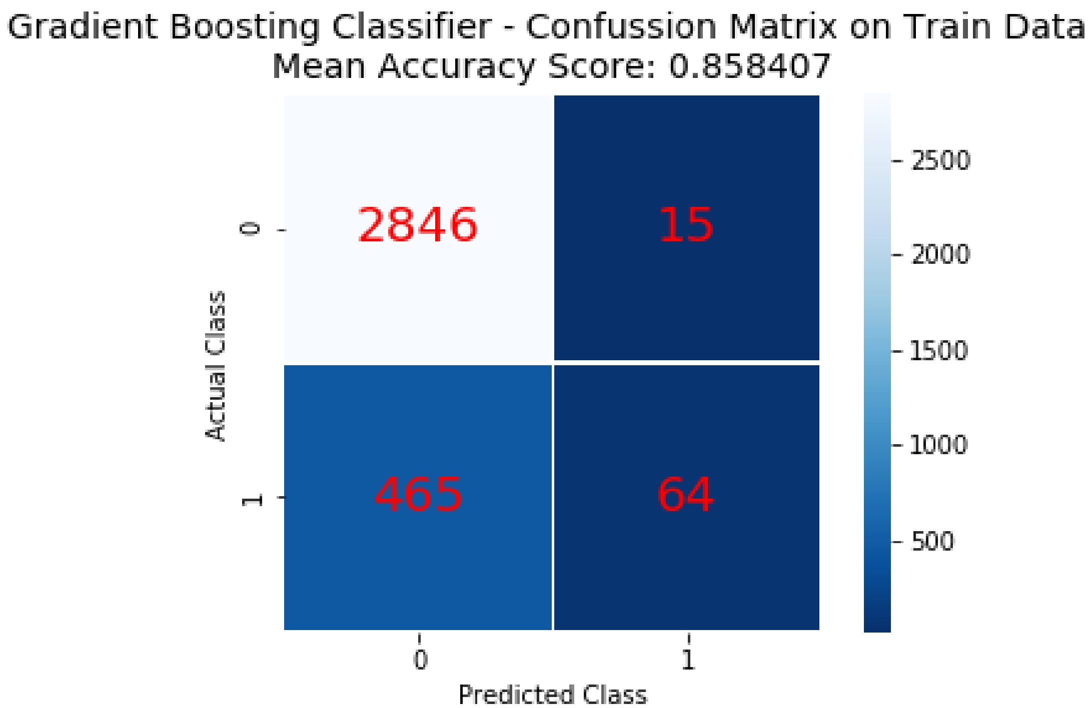
[234]:
```
array([[0.91540188, 0.08459812],
       [0.91767221, 0.08232779],
       [0.92364845, 0.07635155],
       …,
       [0.84344379, 0.15655621],
       [0.90897754, 0.09102246],
       [0.94775825, 0.05224175]])
```

[235]:
```
gb_matrix_tr = metrics.confusion_matrix(y_train, gb_pred_tr)
gb_matrix_tr
```

[235]:
```
array([[2846,   15],
       [ 465,   64]])
```

[236]:
```
gb_train = gb.best_estimator_.score(X_train, y_train)

gb_matrix = metrics.confusion_matrix(y_train, gb_pred_tr)

gb_cm_tr = pd.DataFrame(gb_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (gb_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
 ↪5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

gb_title = 'Gradient Boosting Classifier - Confussion Matrix on Train Data␣
 ↪\nMean Accuracy Score: {0:2f}'.format(gb_train)
```

```
plt.title(gb_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



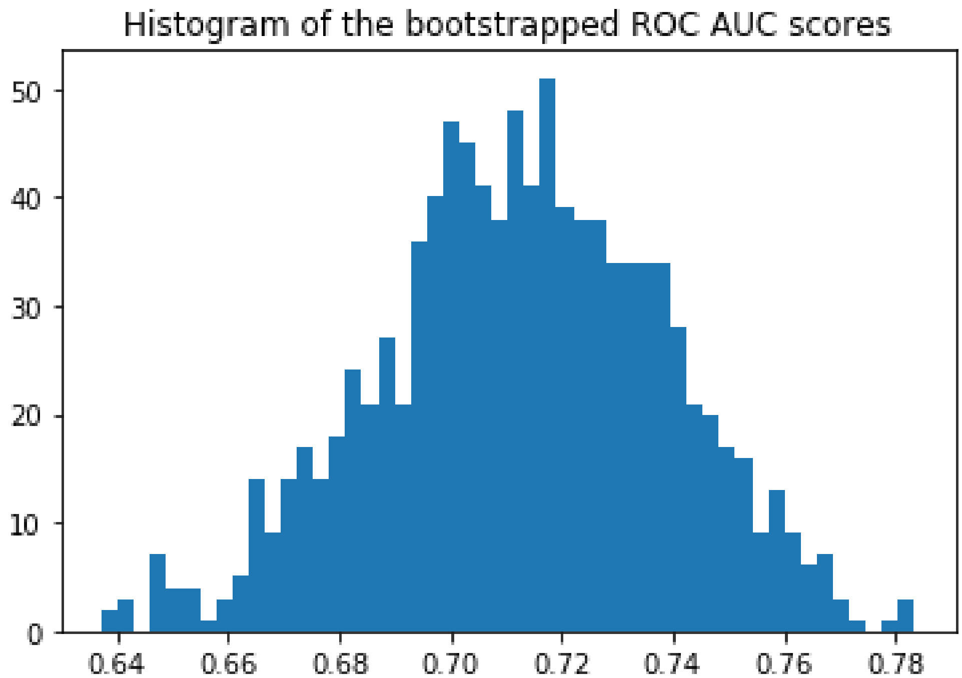
[237]:
```
print("", classification_report(y_train, gb_pred_tr, target_names=target_names))
```

```
                     precision    recall  f1-score   support

No risk in 10 years       0.86      0.99      0.92      2861
  Risky in 10 years       0.81      0.12      0.21       529

           accuracy                           0.86      3390
          macro avg       0.83      0.56      0.57      3390
       weighted avg       0.85      0.86      0.81      3390
```

[238]:
```
y_pred = gb_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []
```

```python
rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.7127
```




```
Confidence interval for the score: [0.6689 - 0.7554]
```

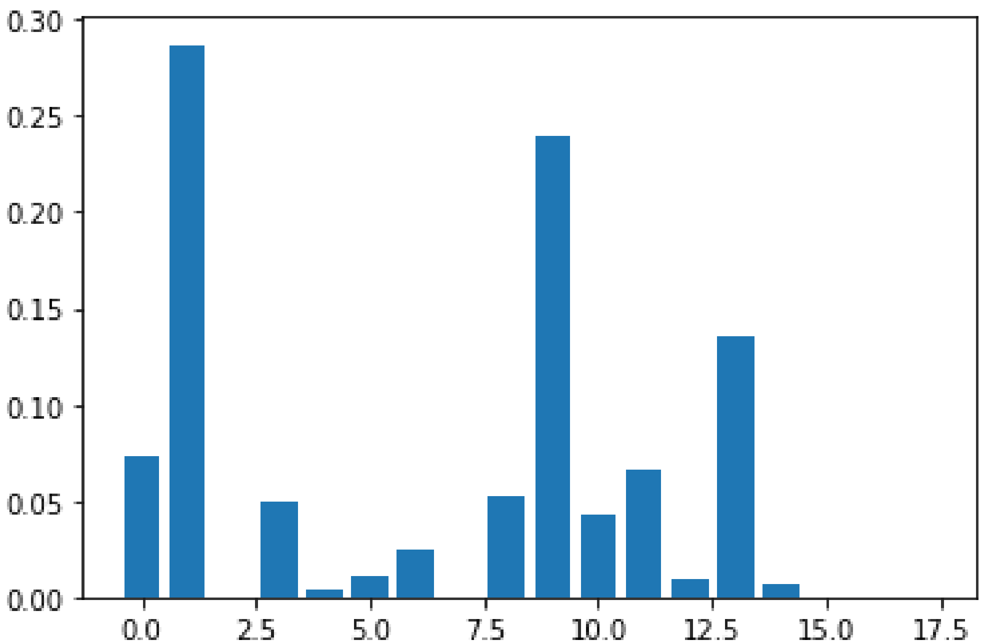
[239]:
```
alpha = .95
y_pred = gb_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.7127172430155999
```

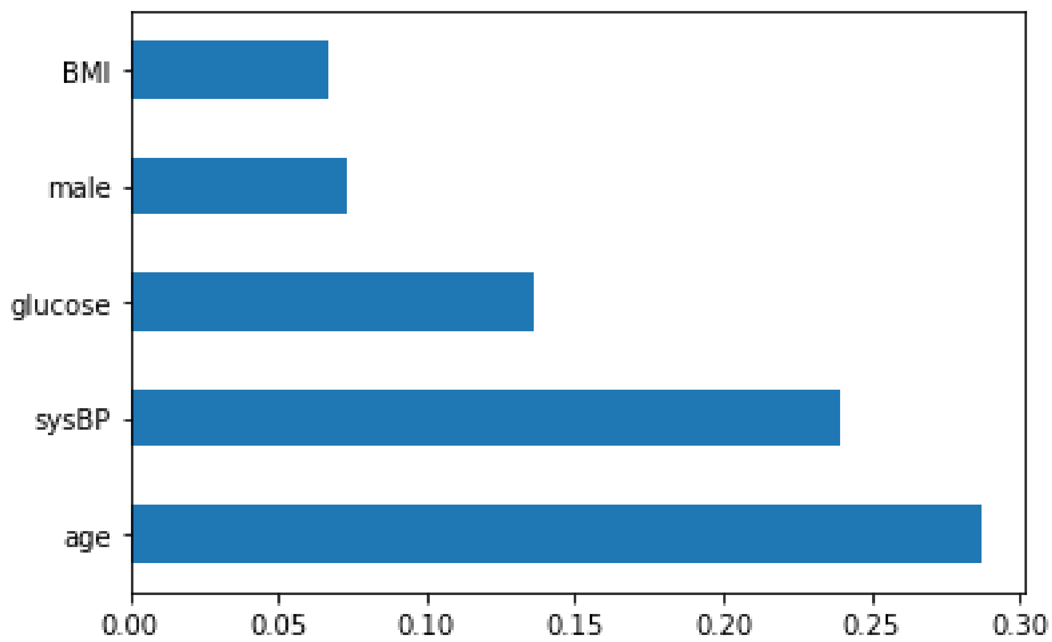
```
AUC COV: 0.0007039269793256396
95% AUC CI: [0.66071622 0.76471827]
```

[240]:
```
print(gb.best_estimator_.feature_importances_)
pyplot.bar(range(len(gb.best_estimator_.feature_importances_)), gb.
 ↪best_estimator_.feature_importances_)
pyplot.show()
```

```
[0.07320772 0.28689989 0.         0.04942735 0.00348583 0.01013333
 0.02396908 0.         0.05248233 0.23961114 0.04251996 0.06676472
 0.00913074 0.13614182 0.00622608 0.         0.         0.        ]
```



[241]:
```
feat_importances_gb = pd.Series(gb.best_estimator_.feature_importances_,␣
 ↪index=X_train.columns)
feat_importances_gb.nlargest(5).plot(kind='barh')
pyplot.show()
```



15 XGBoost

[242]:
```
X_train.dtypes
```

[242]:
```
male                  int8
age                  int64
currentSmoker       object
cigsPerDay         float64
BPMeds              object
prevalentStroke     object
prevalentHyp        object
diabetes            object
totChol            float64
sysBP              float64
diaBP              float64
BMI                float64
heartRate          float64
glucose            float64
1                    uint8
2                    uint8
3                    uint8
4                    uint8
dtype: object
```

[243]:
```
X_train_xgb = X_train.copy()
X_test_xgb = X_test.copy()
```

[244]:
```
X_train_xgb['currentSmoker'] = X_train_xgb['currentSmoker'].apply(int)
X_train_xgb['BPMeds'] = X_train_xgb['BPMeds'].apply(int)
X_train_xgb['prevalentStroke'] = X_train_xgb['prevalentStroke'].apply(int)
X_train_xgb['prevalentHyp'] = X_train_xgb['prevalentHyp'].apply(int)
X_train_xgb['diabetes'] = X_train_xgb['diabetes'].apply(int)

X_test_xgb['currentSmoker'] = X_test_xgb['currentSmoker'].apply(int)
X_test_xgb['BPMeds'] = X_test_xgb['BPMeds'].apply(int)
X_test_xgb['prevalentStroke'] = X_test_xgb['prevalentStroke'].apply(int)
X_test_xgb['prevalentHyp'] = X_test_xgb['prevalentHyp'].apply(int)
X_test_xgb['diabetes'] = X_test_xgb['diabetes'].apply(int)
```

[245]:
```
from xgboost.sklearn import XGBClassifier
# create a dictionary of parameters using range(start, stop but not including,␣
↪step)
params7 = {'n_estimators': list(range(100, 1100, 300)),
           'learning_rate':[i/10.0 for i in range(1,11,4)],
           'max_depth':[1,2],
#             'gamma': [i/4 for i in range(0,20,1)]
          }

# create XGBoost model
xgb = XGBClassifier(random_state = 1)

# create randomizedsearchCV object with various combinations of parameters
xgb = GridSearchCV(xgb, params7, cv = 5,
                   refit = True,
                   n_jobs=-1, verbose = 5)

xgb.fit(X_train_xgb, y_train)
xgb.cv_results_
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    9.0s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:   20.5s finished
/usr/local/lib/python3.7/site-packages/sklearn/preprocessing/label.py:219:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/site-packages/sklearn/preprocessing/label.py:252:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
```

```
ravel().
  y = column_or_1d(y, warn=True)
```

[245]:
```
{'mean_fit_time': array([0.260603  , 1.09295635, 1.84497924, 2.59585052,
0.32896075,
        1.3467576 , 2.36528506, 3.53867755, 0.27087879, 1.09671917,
        2.21313467, 3.1153204 , 0.47617645, 1.53220892, 2.46041994,
        3.57345562, 0.27446666, 1.10995016, 1.99847832, 2.87910342,
        0.36073437, 1.53775702, 2.4301096 , 2.76862206]),
 'std_fit_time': array([0.01342605, 0.02447204, 0.01981906, 0.03666803,
0.01541151,
        0.01504484, 0.01986879, 0.0222688 , 0.00251561, 0.03641868,
        0.02214971, 0.03437687, 0.00927825, 0.01884433, 0.03741645,
        0.01482621, 0.00315911, 0.00897173, 0.00534643, 0.02557538,
        0.00516977, 0.00603459, 0.17766013, 0.09265213]),
 'mean_score_time': array([0.00515561, 0.00722599, 0.00896654, 0.01349001,
0.00350246,
        0.01060696, 0.01414299, 0.02288766, 0.00438466, 0.00607777,
        0.01185169, 0.01167564, 0.00578146, 0.00823379, 0.01362171,
        0.01880012, 0.00321822, 0.00671625, 0.00861578, 0.01158376,
        0.00378733, 0.00899043, 0.01354976, 0.01410108]),
 'std_score_time': array([2.52942417e-03, 2.21066129e-03, 2.74201956e-04,
2.37103812e-03,
        3.22866457e-04, 2.57612197e-03, 1.18864602e-03, 3.90578221e-03,
        1.10571867e-03, 1.01188845e-04, 1.48299622e-03, 1.98673983e-04,
        3.59469194e-03, 5.00331486e-05, 4.31018475e-04, 2.77551366e-04,
        2.77695215e-04, 1.58732938e-03, 3.52595282e-04, 2.22766379e-04,
        9.54398119e-05, 3.73850051e-04, 2.81337213e-04, 4.40980190e-04]),
 'param_learning_rate': masked_array(data=[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
0.1, 0.5, 0.5, 0.5,
                    0.5, 0.5, 0.5, 0.5, 0.5, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9,
                    0.9, 0.9],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2,
2, 2, 1, 1,
                    1, 1, 2, 2, 2, 2],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[100, 400, 700, 1000, 100, 400, 700,
1000, 100, 400,
```

```
                   700, 1000, 100, 400, 700, 1000, 100, 400, 700, 1000,
                   100, 400, 700, 1000],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100},
  {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 400},
  {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700},
  {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 1000},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 400},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 1000},
  {'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 100},
  {'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 400},
  {'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 700},
  {'learning_rate': 0.5, 'max_depth': 1, 'n_estimators': 1000},
  {'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 400},
  {'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 700},
  {'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 1000},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 100},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 400},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 700},
  {'learning_rate': 0.9, 'max_depth': 1, 'n_estimators': 1000},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 400},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 700},
  {'learning_rate': 0.9, 'max_depth': 2, 'n_estimators': 1000}],
 'split0_test_score': array([0.84977909, 0.84977909, 0.85125184, 0.85272459,
0.84683358,
        0.84094256, 0.83505155, 0.82916053, 0.84683358, 0.84536082,
        0.84683358, 0.84094256, 0.83063328, 0.80559647, 0.79381443,
        0.78645066, 0.84683358, 0.83210604, 0.8365243 , 0.82179676,
        0.81885125, 0.77761414, 0.77614138, 0.77025037]),
 'split1_test_score': array([0.84513274, 0.85250737, 0.8539823 , 0.8539823 ,
0.84365782,
        0.83923304, 0.84070796, 0.83775811, 0.85693215, 0.85545723,
        0.84955752, 0.8480826 , 0.82448378, 0.820059  , 0.81563422,
        0.80235988, 0.8539823 , 0.84660767, 0.84365782, 0.84070796,
        0.82448378, 0.80530973, 0.78908555, 0.78466077]),
 'split2_test_score': array([0.84513274, 0.8539823 , 0.85545723, 0.85545723,
0.8480826 ,
        0.8539823 , 0.85693215, 0.85103245, 0.85693215, 0.8539823 ,
        0.8539823 , 0.85103245, 0.8480826 , 0.83185841, 0.83333333,
```

```
        0.8259587 , 0.85103245, 0.85103245, 0.84955752, 0.84660767,
        0.83038348, 0.81710914, 0.80973451, 0.80678466]),
 'split3_test_score': array([0.84365782, 0.84070796, 0.84070796, 0.83480826,
0.84218289,
        0.83923304, 0.83333333, 0.83480826, 0.83775811, 0.83333333,
        0.83185841, 0.82743363, 0.82743363, 0.81710914, 0.79941003,
        0.7979351 , 0.83038348, 0.82448378, 0.82448378, 0.82153392,
        0.82153392, 0.80530973, 0.79498525, 0.78318584]),
 'split4_test_score': array([0.8478582 , 0.84490399, 0.84342688, 0.84342688,
0.84047267,
        0.83899557, 0.83013294, 0.82717873, 0.84638109, 0.83899557,
        0.84047267, 0.83604136, 0.82422452, 0.80649926, 0.7872969 ,
        0.78286558, 0.84047267, 0.83308715, 0.83013294, 0.83161004,
        0.80502216, 0.77548006, 0.76218612, 0.77991137]),
 'mean_test_score': array([0.84631268, 0.84837758, 0.84896755, 0.8480826 ,
0.84424779,
        0.84247788, 0.83923304, 0.8359882 , 0.84896755, 0.84542773,
        0.84454277, 0.84070796, 0.83097345, 0.81622419, 0.80589971,
        0.79911504, 0.84454277, 0.83746313, 0.83687316, 0.83244838,
        0.820059  , 0.79616519, 0.78643068, 0.78495575]),
 'std_test_score': array([0.00220234, 0.00492557, 0.00585429, 0.0078565 ,
0.00283705,
        0.00579446, 0.00949148, 0.00842653, 0.00726308, 0.00850132,
        0.00770639, 0.00847335, 0.00886528, 0.0096692 , 0.01662109,
        0.0152104 , 0.00840836, 0.00983945, 0.00901007, 0.01002155,
        0.00843306, 0.01659992, 0.01622501, 0.01201413]),
 'rank_test_score': array([ 5,  3,  1,  4,  9, 10, 12, 15,  1,  6,  7, 11, 17,
19, 20, 21,  7,
        13, 14, 16, 18, 22, 23, 24], dtype=int32)}
```

[246]:
```
xgb.best_estimator_
```

[246]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=1,
              min_child_weight=1, missing=None, n_estimators=700, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=1,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

[247]:
```
xgb_pred = xgb.best_estimator_.predict(X_test_xgb)
xgb_prob = xgb.best_estimator_.predict_proba(X_test_xgb)
xgb_prob
```

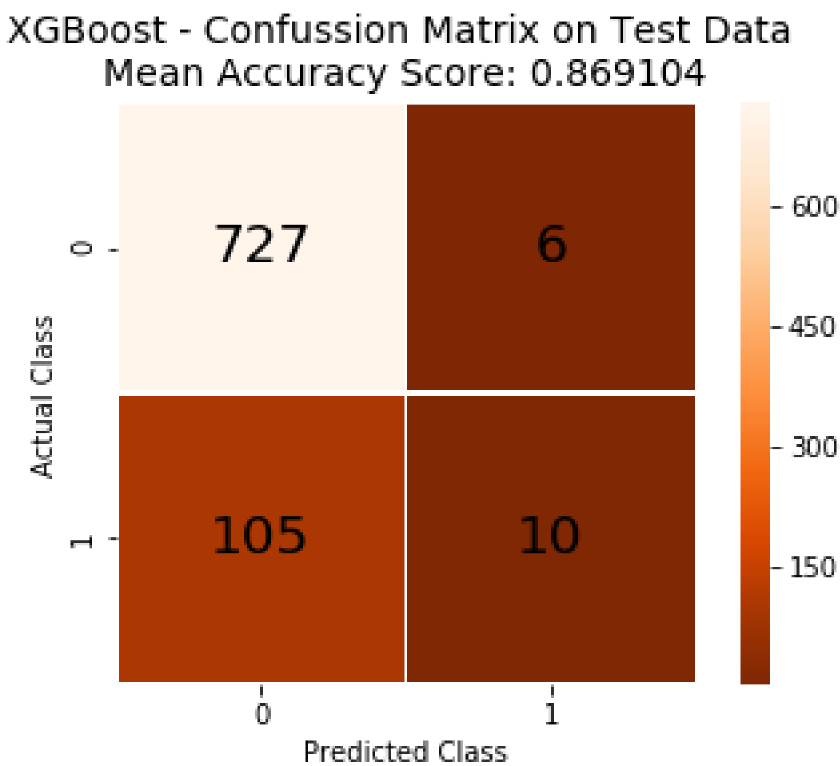
[247]:
```
array([[0.84977704, 0.15022297],
       [0.951557  , 0.04844299],
       [0.95299846, 0.04700153],
```

```
       …,
       [0.8036161 , 0.19638388],
       [0.8124214 , 0.18757862],
       [0.9151857 , 0.08481433]], dtype=float32)
```

[248]:
```
xgb_matrix = metrics.confusion_matrix(y_test, xgb_pred)
xgb_matrix
```

[248]:
```
array([[727,   6],
       [105,  10]])
```

[249]:
```
xgb_test = xgb.best_estimator_.score(X_test_xgb, y_test)

xgb_matrix = metrics.confusion_matrix(y_test, xgb_pred)

xgb_cm = pd.DataFrame(xgb_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (xgb_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

xgb_title = 'XGBoost - Confussion Matrix on Test Data \nMean Accuracy Score: {0:
 ↪2f}'.format(xgb_test)
plt.title(xgb_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



[250]:
```
print("", classification_report(y_test, xgb_pred, target_names=target_names))
```

```
                   precision    recall  f1-score   support

No risk in 10 years       0.87      0.99      0.93       733
  Risky in 10 years       0.62      0.09      0.15       115

           accuracy                           0.87       848
          macro avg       0.75      0.54      0.54       848
       weighted avg       0.84      0.87      0.82       848
```

[251]:
```
acc_xgb = accuracy_score(y_test, xgb_pred)
print("XGBoost accuracy:", acc_xgb)
```

```
XGBoost accuracy: 0.8691037735849056
```

[252]:
```
error_xgb = 1-acc_xgb
error_xgb
```

[252]: 0.13089622641509435

[253]:
```
xgb_probs = xgb.best_estimator_.predict_proba(X_test_xgb)[:,1]
print(roc_auc_score(y_test, xgb_probs))
```

```
0.7155465923245744
```

15.0.1 For training set:

[254]:
```
xgb_pred_tr = xgb.best_estimator_.predict(X_train_xgb)
xgb_prob_tr = xgb.best_estimator_.predict_proba(X_train_xgb)
xgb_prob_tr
```

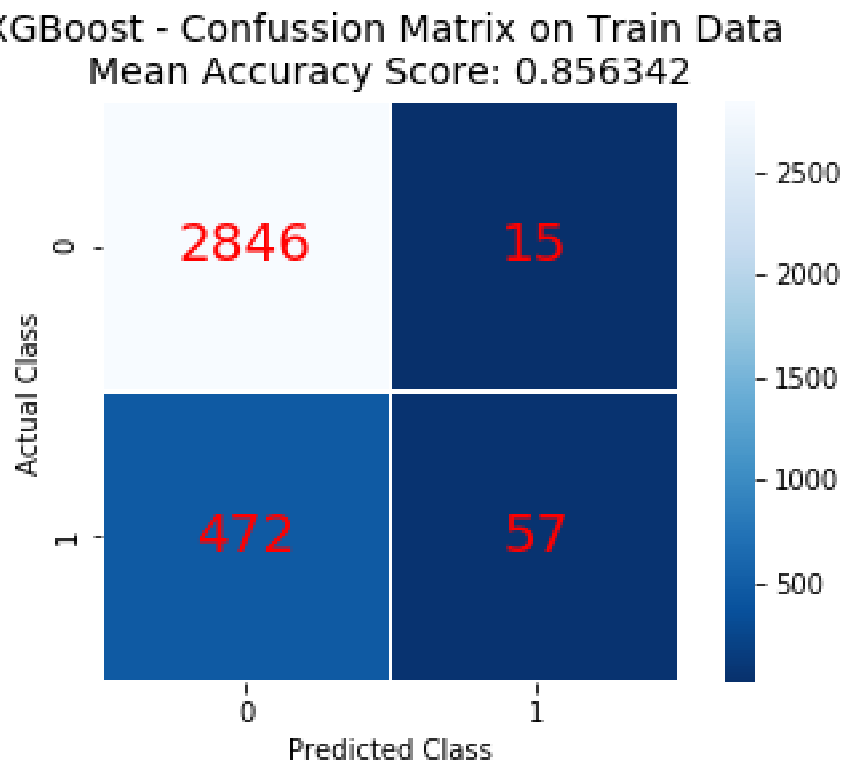
[254]:
```
array([[0.91783243, 0.08216758],
       [0.92377293, 0.07622706],
       [0.92486244, 0.07513757],
       …,
       [0.8580538 , 0.14194618],
       [0.9166989 , 0.08330115],
       [0.9624737 , 0.03752631]], dtype=float32)
```

[255]:
```
xgb_matrix_tr = metrics.confusion_matrix(y_train, xgb_pred_tr)
xgb_matrix_tr
```

[255]:
```
array([[2846,   15],
       [ 472,   57]])
```

[356]:
```
xgb_train = xgb.best_estimator_.score(X_train_xgb, y_train)

xgb_matrix = metrics.confusion_matrix(y_train, xgb_pred_tr)

xgb_cm_tr = pd.DataFrame(xgb_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (xgb_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
↪5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

xgb_title = 'XGBoost - Confussion Matrix on Train Data \nMean Accuracy Score:␣
↪{0:2f}'.format(xgb_train)
plt.title(xgb_title, size = 14)
# plt.figure(figsize=(16, 26))
```

```
plt.show;
```



[257]:

```
print("", classification_report(y_train, xgb_pred_tr,␣
↪target_names=target_names))
```

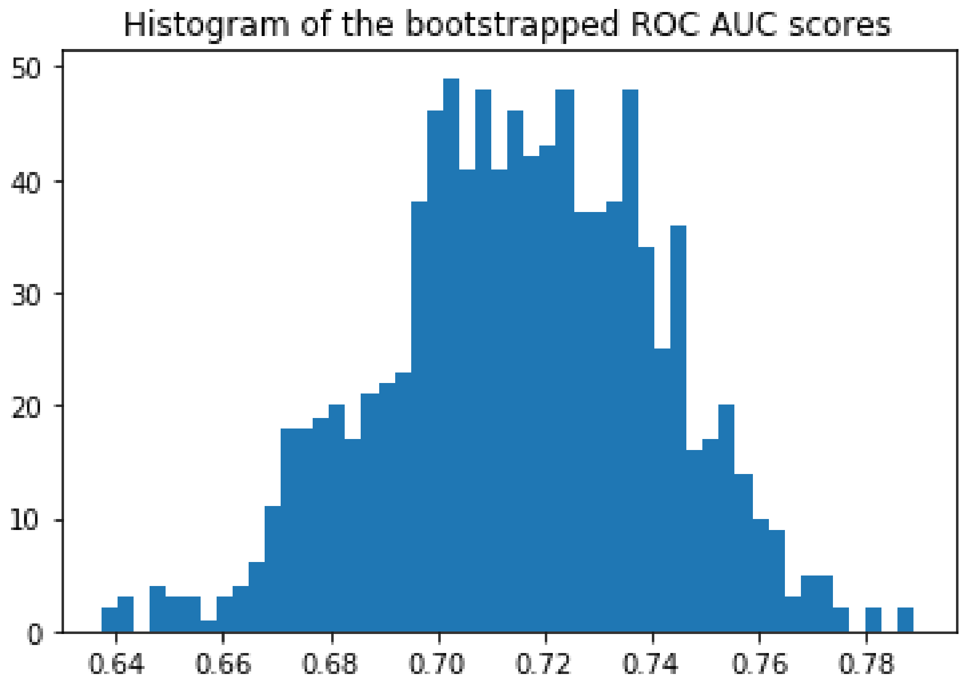
```
                    precision    recall  f1-score   support

No risk in 10 years       0.86      0.99      0.92      2861
  Risky in 10 years       0.79      0.11      0.19       529

           accuracy                           0.86      3390
          macro avg       0.82      0.55      0.56      3390
       weighted avg       0.85      0.86      0.81      3390
```

[258]:

```
y_pred = xgb_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
```

```python
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.7155
```




```
Confidence interval for the score: [0.6732 - 0.7561]
```

[259]:
```
alpha = .95
y_pred = xgb_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

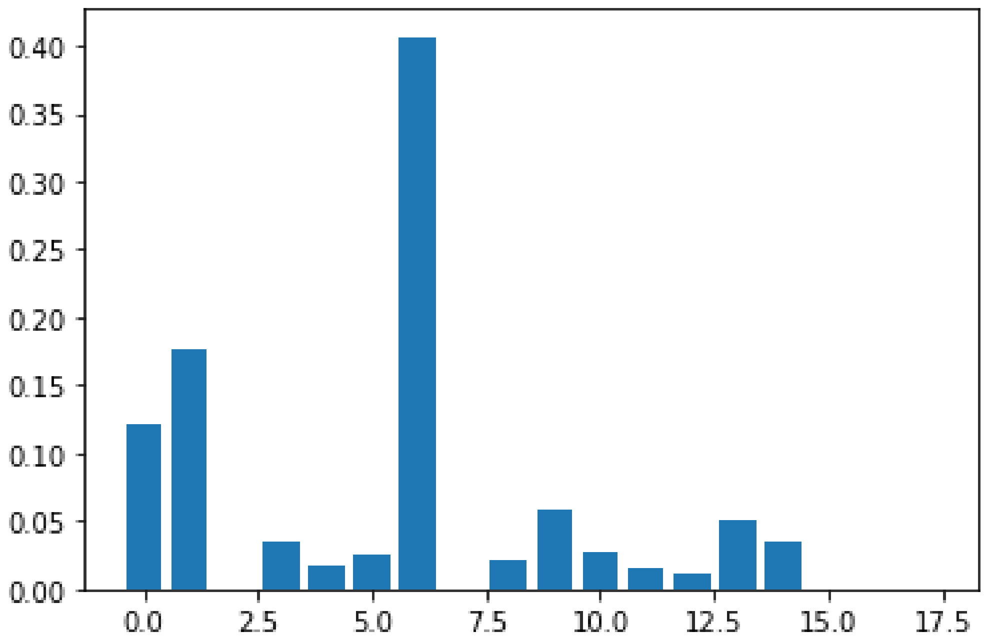
```
AUC: 0.7155465923245744
```

```
AUC COV: 0.0006809455165142485
95% AUC CI: [0.66440146 0.76669172]
```

[260]:

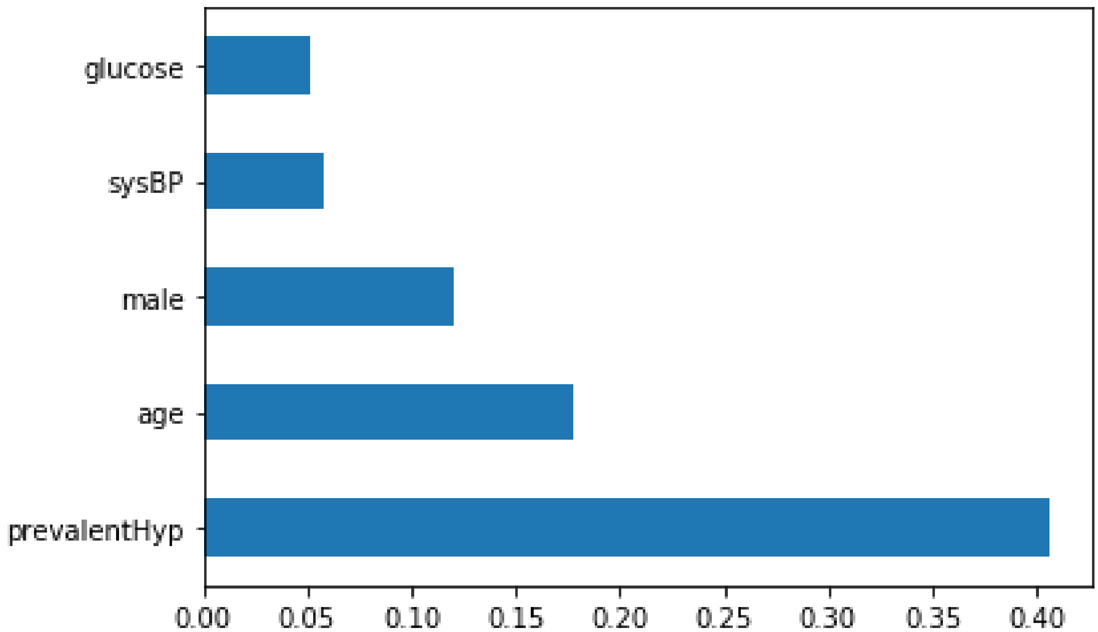
```
print(xgb.best_estimator_.feature_importances_)
pyplot.bar(range(len(xgb.best_estimator_.feature_importances_)), xgb.
 ↪best_estimator_.feature_importances_)
pyplot.show()
```

```
[0.1206473  0.1772953  0.         0.03535922 0.01786305 0.02476247
 0.4066664  0.         0.02030996 0.05846637 0.02616968 0.01455139
 0.01199612 0.05129574 0.03461698 0.         0.         0.        ]
```



[261]:

```
feat_importances_xgb = pd.Series(xgb.best_estimator_.feature_importances_,␣
 ↪index=X_train.columns)
feat_importances_xgb.nlargest(5).plot(kind='barh')
pyplot.show()
```



[ ]:

16 Artificial Neural Network

[262]:
```
df[df['TenYearCHD']==0].shape
```

[262]: (3594, 19)

[263]:
```
df[df['TenYearCHD']==1].shape
```

[263]: (644, 19)

[264]:
```
from imblearn.over_sampling import SMOTE
smt = SMOTE()
X_train_ann = X_train.copy()
X_test_ann = X_test.copy()
y_train_ann = y_train.copy()
y_test_ann = y_test.copy()
X_train_ann, y_train_ann = smt.fit_sample(X_train_ann, y_train_ann)
```

```
Using TensorFlow backend.
/usr/local/lib/python3.7/site-packages/sklearn/utils/validation.py:724:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
```

```
ravel().
  y = column_or_1d(y, warn=True)
```

16.0.1 To test for the SMOTE

[265]:
```
y_train_ann.shape
```

[265]: (5722,)

[266]:
```
y_train[y_train['TenYearCHD']==0].shape
```

[266]: (2861, 1)

[267]:
```
y_train[y_train['TenYearCHD']==1].shape
```

[267]: (529, 1)

[268]:
```
X_train.shape
```

[268]: (3390, 18)

[269]:
```
y_train_ann = pd.DataFrame(y_train_ann)
y_train_ann.columns = ['TenYearCHD']
y_train_ann[y_train_ann['TenYearCHD']==0].shape
```

[269]: (2861, 1)

[270]:
```
y_train_ann[y_train_ann['TenYearCHD']==1].shape
```

[270]: (2861, 1)

[271]:
```
X_train_ann.shape
```

[271]: (5722, 18)

[272]:
```
# from sklearn.preprocessing import StandardScaler
# scaler = StandardScaler()
# Fit only to the training data
scaler = scaler.fit(X_train_ann)
X_train_anns = scaler.transform(X_train_ann)
X_test_anns = scaler.transform(X_test_ann)
```

[273]:
```
from sklearn.neural_network import MLPClassifier
seed(1)
# skf = StratifiedKFold(n_splits=5)
params8 = {'alpha' : [0.0001,0.01],
           'power_t': [0.5,0.75],
```

```
        'max_iter': [200,250]}
mlp = MLPClassifier(solver='lbfgs', random_state=1)
mlp = GridSearchCV(mlp, cv=5, param_grid=params8, scoring = 'roc_auc',refit =␣
↪True,
                  n_jobs=-1, verbose = 5, return_train_score=True)

mlp.fit(X_train_anns, y_train_ann)
mlp.cv_results_
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 out of  40 | elapsed:   23.4s remaining:   12.6s
[Parallel(n_jobs=-1)]: Done  35 out of  40 | elapsed:   26.4s remaining:    3.8s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   28.5s finished
/usr/local/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:921:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[273]:
```
{'mean_fit_time': array([7.06076951, 7.14142642, 8.78435416, 8.66917658,
6.9881465 ,
        7.03834085, 8.37243543, 4.9870924 ]),
 'std_fit_time': array([0.04252977, 0.0576656 , 0.0998577 , 0.03139759,
0.0460937 ,
        0.10180893, 0.03818451, 1.69337437]),
 'mean_score_time': array([0.01264338, 0.0097466 , 0.01141415, 0.01123695,
0.01073775,
        0.01049471, 0.00621152, 0.00406189]),
 'std_score_time': array([0.00333593, 0.00038653, 0.00117778, 0.00155474,
0.00097223,
        0.00074526, 0.00184749, 0.00177847]),
 'param_alpha': masked_array(data=[0.0001, 0.0001, 0.0001, 0.0001, 0.01, 0.01,
0.01, 0.01],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_max_iter': masked_array(data=[200, 200, 250, 250, 200, 200, 250, 250],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_power_t': masked_array(data=[0.5, 0.75, 0.5, 0.75, 0.5, 0.75, 0.5,
0.75],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
```

```
 'params': [{'alpha': 0.0001, 'max_iter': 200, 'power_t': 0.5},
  {'alpha': 0.0001, 'max_iter': 200, 'power_t': 0.75},
  {'alpha': 0.0001, 'max_iter': 250, 'power_t': 0.5},
  {'alpha': 0.0001, 'max_iter': 250, 'power_t': 0.75},
  {'alpha': 0.01, 'max_iter': 200, 'power_t': 0.5},
  {'alpha': 0.01, 'max_iter': 200, 'power_t': 0.75},
  {'alpha': 0.01, 'max_iter': 250, 'power_t': 0.5},
  {'alpha': 0.01, 'max_iter': 250, 'power_t': 0.75}],
 'split0_test_score': array([0.78238748, 0.78238748, 0.77835342, 0.77835342,
0.79359118,
        0.79359118, 0.79959888, 0.79959888]),
 'split1_test_score': array([0.98047123, 0.98047123, 0.98003111, 0.98003111,
0.98069741,
        0.98069741, 0.98151499, 0.98151499]),
 'split2_test_score': array([0.98079063, 0.98079063, 0.98120782, 0.98120782,
0.98067143,
        0.98067143, 0.98006626, 0.98006626]),
 'split3_test_score': array([0.97618771, 0.97618771, 0.97835163, 0.97835163,
0.97581025,
        0.97581025, 0.97601044, 0.97601044]),
 'split4_test_score': array([0.98155167, 0.98155167, 0.98251748, 0.98251748,
0.98205597,
        0.98205597, 0.98253276, 0.98253276]),
 'mean_test_score': array([0.94022256, 0.94022256, 0.94003576, 0.94003576,
0.94251318,
        0.94251318, 0.94389421, 0.94389421]),
 'std_test_score': array([0.07900868, 0.07900868, 0.08092342, 0.08092342,
0.07455633,
        0.07455633, 0.07224479, 0.07224479]),
 'rank_test_score': array([5, 5, 7, 7, 3, 3, 1, 1], dtype=int32),
 'split0_train_score': array([1.        , 1.        , 1.        , 1.        ,
0.99999274,
        0.99999274, 1.        , 1.        ]),
 'split1_train_score': array([0.99691708, 0.99691708, 0.99896326, 0.99896326,
0.99758928,
        0.99758928, 0.99927951, 0.99927951]),
 'split2_train_score': array([0.99762077, 0.99762077, 0.99927856, 0.99927856,
0.9974784 ,
        0.9974784 , 0.9993591 , 0.9993591 ]),
 'split3_train_score': array([0.99737514, 0.99737514, 0.99930643, 0.99930643,
0.9966413 ,
        0.9966413 , 0.99883787, 0.99883787]),
 'split4_train_score': array([0.99698732, 0.99698732, 0.99897911, 0.99897911,
0.99680143,
        0.99680143, 0.99904018, 0.99904018]),
 'mean_train_score': array([0.99778006, 0.99778006, 0.99930547, 0.99930547,
0.99770063,
```

```
        0.99770063, 0.99930333, 0.99930333]),
 'std_train_score': array([0.00113935, 0.00113935, 0.00037596, 0.00037596,
0.00120385,
        0.00120385, 0.00039368, 0.00039368])}
```

[274]:
```python
mlp.best_estimator_
```

[274]:
```
MLPClassifier(activation='relu', alpha=0.01, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=250, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=1, shuffle=True, solver='lbfgs', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

[275]:
```python
mlp_pred = mlp.best_estimator_.predict(X_test_anns)
mlp_prob = mlp.best_estimator_.predict_proba(X_test_anns)
mlp_prob
```

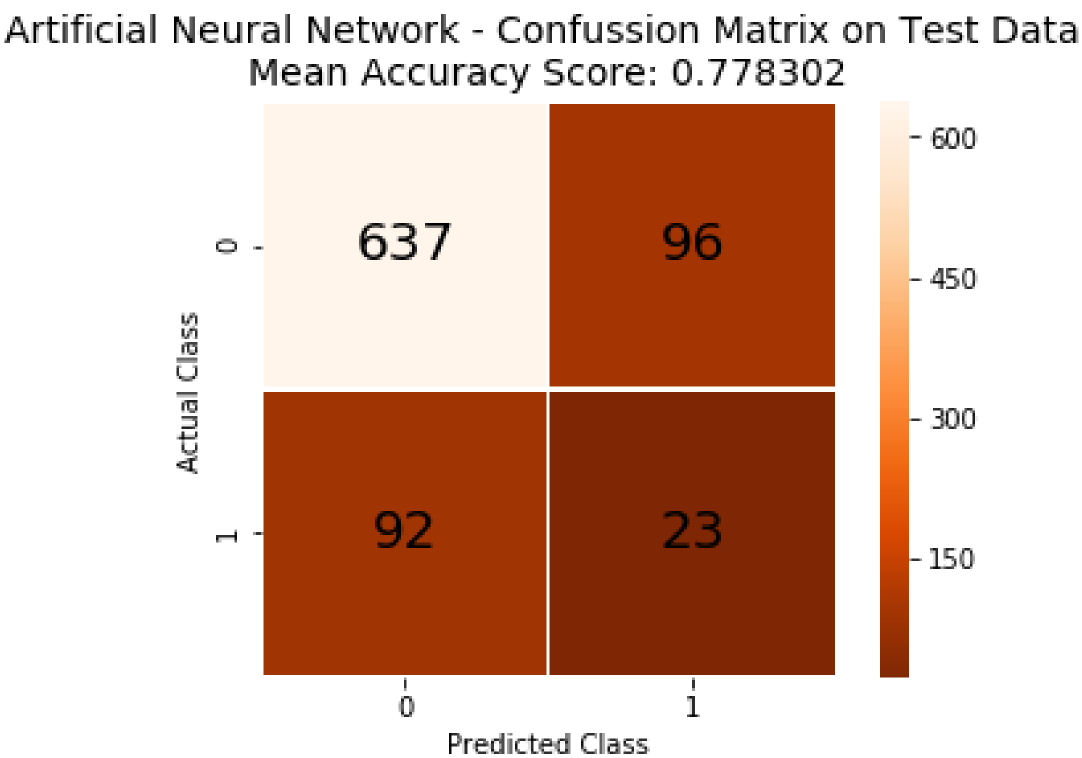
[275]:
```
array([[6.68254652e-01, 3.31745348e-01],
       [9.61147098e-01, 3.88529025e-02],
       [1.00000000e+00, 3.56061839e-15],
       …,
       [9.99987438e-01, 1.25621189e-05],
       [9.78984709e-01, 2.10152910e-02],
       [9.97576219e-01, 2.42378142e-03]])
```

[276]:
```python
mlp_matrix = metrics.confusion_matrix(y_test_ann, mlp_pred)
mlp_matrix
```

[276]:
```
array([[637,  96],
       [ 92,  23]])
```

[277]:
```python
mlp_test = mlp.best_estimator_.score(X_test_anns, y_test_ann)

mlp_matrix = metrics.confusion_matrix(y_test_ann, mlp_pred)

mlp_cm = pd.DataFrame(mlp_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (mlp_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

```
plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

mlp_title = 'Artificial Neural Network - Confussion Matrix on Test Data \nMean␣
↪Accuracy Score: {0:2f}'.format(mlp_test)
plt.title(mlp_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



[278]:
```
print("", classification_report(y_test_ann, mlp_pred,␣
↪target_names=target_names))
```

```
                   precision    recall  f1-score   support

No risk in 10 years       0.87      0.87      0.87       733
  Risky in 10 years       0.19      0.20      0.20       115

           accuracy                           0.78       848
          macro avg       0.53      0.53      0.53       848
       weighted avg       0.78      0.78      0.78       848
```

[279]:
```
acc_mlp = accuracy_score(y_test_ann, mlp_pred)
print("Artificial Neural Network accuracy:", acc_mlp)
```

```
Artificial Neural Network accuracy: 0.7783018867924528
```

[280]:
```
error_mlp = 1-acc_mlp
error_mlp
```

[280]:
```
0.22169811320754718
```

[281]:
```
mlp_probs = mlp.best_estimator_.predict_proba(X_test_anns)[:,1]
print(roc_auc_score(y_test_ann, mlp_probs))
```

```
0.5232101548134527
```

16.0.2 For training set:

[282]:
```
mlp_pred_tr = mlp.best_estimator_.predict(X_train_anns)
mlp_prob_tr = mlp.best_estimator_.predict_proba(X_train_anns)
mlp_prob_tr
```

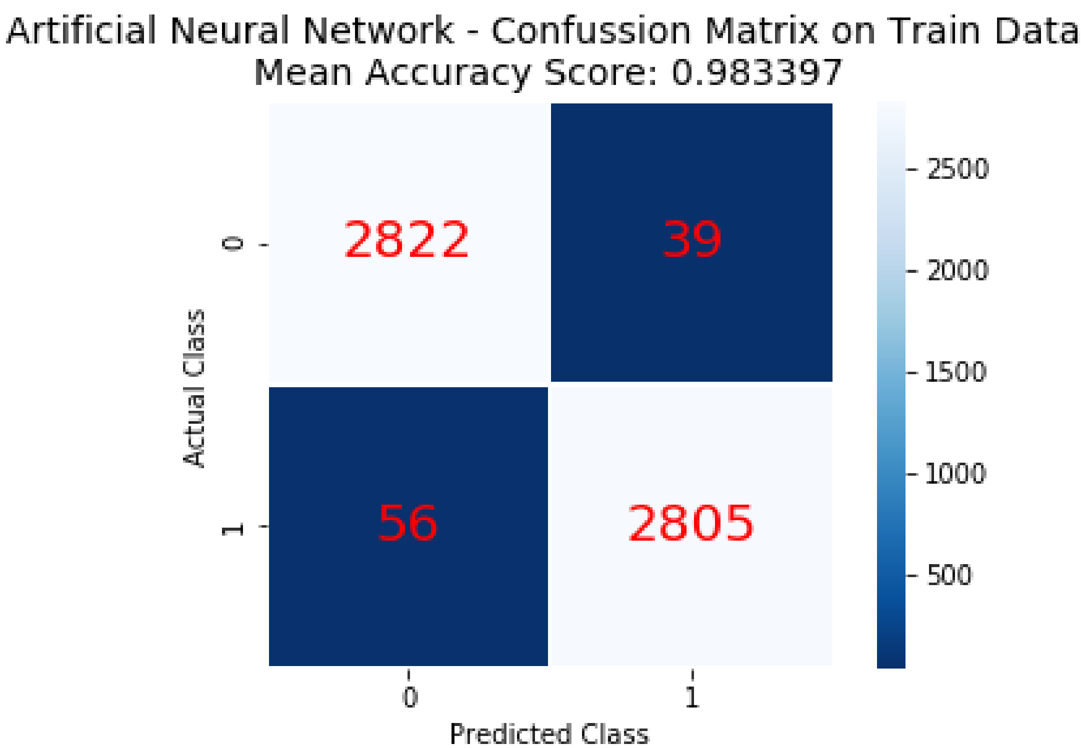
[282]:
```
array([[9.98338668e-01, 1.66133201e-03],
       [3.08207186e-01, 6.91792814e-01],
       [9.98273597e-01, 1.72640254e-03],
       …,
       [1.36327839e-01, 8.63672161e-01],
       [7.75957076e-12, 1.00000000e+00],
       [0.00000000e+00, 1.00000000e+00]])
```

[283]:
```
mlp_matrix_tr = metrics.confusion_matrix(y_train_ann, mlp_pred_tr)
mlp_matrix_tr
```

[283]:
```
array([[2822,   39],
       [  56, 2805]])
```

[284]:
```
mlp_train = mlp.best_estimator_.score(X_train_anns, y_train_ann)

mlp_matrix = metrics.confusion_matrix(y_train_ann, mlp_pred_tr)

mlp_cm_tr = pd.DataFrame(mlp_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (mlp_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
 ↪5,annot_kws=akws)
```

```
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

mlp_title = 'Artificial Neural Network - Confussion Matrix on Train Data \nMean␣
↪Accuracy Score: {0:2f}'.format(mlp_train)
plt.title(mlp_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



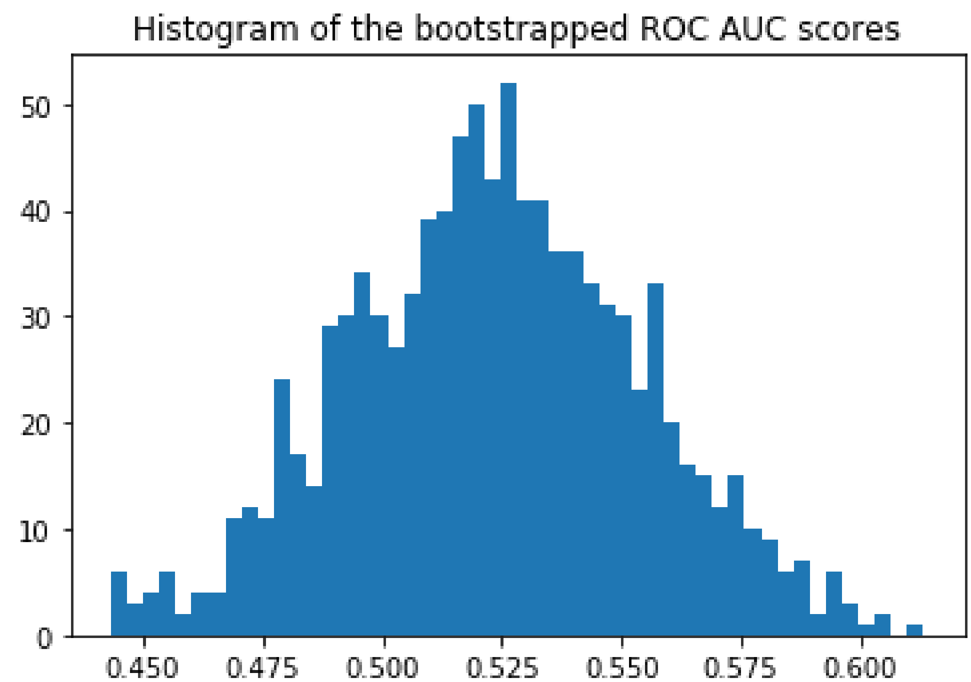
[285]:
```
print("", classification_report(y_train_ann, mlp_pred_tr,␣
↪target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years       0.98      0.99      0.98      2861
  Risky in 10 years       0.99      0.98      0.98      2861

           accuracy                           0.98      5722
          macro avg       0.98      0.98      0.98      5722
       weighted avg       0.98      0.98      0.98      5722
```

[286]:

```python
y_pred = mlp_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.5232
```




```
Confidence interval for the score: [0.4737 - 0.5754]
```

[287]:
```
alpha = .95
y_pred = mlp_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.5232101548134528
```

```
AUC COV: 0.0009674056783003154
95% AUC CI: [0.46224911 0.5841712 ]
```

17 Bagging Classifier

[289]:
```
from sklearn.ensemble import BaggingClassifier
seed(1)
params9 = {'n_estimators' : [10,250,500],
           'max_samples': [1,2,3],
           'max_features': [1,2,3]}
bg = BaggingClassifier(random_state=0)
bg = GridSearchCV(bg, cv=5, param_grid=params9, scoring = 'roc_auc',refit =␣
 ↪True,
                  n_jobs=-1, verbose = 5, return_train_score=True)

bg.fit(X_train, y_train)
bg.cv_results_
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done 135 out of 135 | elapsed:   14.3s finished
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/bagging.py:623:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[289]:
```
{'mean_fit_time': array([0.04484706, 0.72290874, 1.46731582, 0.03910427,
0.71862798,
        1.44076791, 0.03893161, 0.73467422, 1.47788825, 0.05005646,
        0.7934576 , 1.46597652, 0.04321842, 0.69078479, 1.26879425,
        0.03361645, 0.62356362, 1.32698855, 0.03710866, 0.77036166,
        1.56274529, 0.04236307, 0.77810402, 1.48561015, 0.03795991,
        0.7337718 , 1.12977209]),
 'std_fit_time': array([0.00378684, 0.01122412, 0.01101424, 0.00253033,
0.0088934 ,
        0.01180028, 0.00323655, 0.0069623 , 0.04061906, 0.00423101,
        0.03921366, 0.0391275 , 0.00636319, 0.00460621, 0.03994948,
        0.00176702, 0.00259025, 0.05322291, 0.00482043, 0.02846782,
        0.01442894, 0.00387562, 0.01033031, 0.03007344, 0.00134103,
        0.02527591, 0.08220997]),
 'mean_score_time': array([0.01021256, 0.07981405, 0.13589292, 0.00825076,
0.08541961,
        0.13275394, 0.01099052, 0.05828094, 0.13445516, 0.00937548,
```

```
       0.09336901, 0.13875685, 0.01099172, 0.08322902, 0.13876443,
       0.00715327, 0.07170157, 0.16893706, 0.00751667, 0.09845896,
       0.18409214, 0.00868087, 0.09555802, 0.18018813, 0.00824857,
       0.09146318, 0.09141593]),
 'std_score_time': array([0.00294167, 0.00554998, 0.00290347, 0.00189018,
0.01587274,
       0.0166047 , 0.00261527, 0.00262133, 0.0112499 , 0.00128698,
       0.00324441, 0.00672005, 0.00259179, 0.00551209, 0.00164237,
       0.00010047, 0.00069929, 0.00392772, 0.00012075, 0.0013499 ,
       0.00602621, 0.00014651, 0.00270116, 0.03298741, 0.00012225,
       0.00072456, 0.00074793]),
 'param_max_features': masked_array(data=[1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2,
                   3, 3, 3, 3, 3, 3, 3, 3, 3],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object),
 'param_max_samples': masked_array(data=[1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 1, 1, 2,
2, 2, 3, 3, 3,
                   1, 1, 1, 2, 2, 2, 3, 3, 3],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object),
 'param_n_estimators': masked_array(data=[10, 250, 500, 10, 250, 500, 10, 250,
500, 10, 250, 500,
                   10, 250, 500, 10, 250, 500, 10, 250, 500, 10, 250, 500,
                   10, 250, 500],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'max_features': 1, 'max_samples': 1, 'n_estimators': 10},
  {'max_features': 1, 'max_samples': 1, 'n_estimators': 250},
  {'max_features': 1, 'max_samples': 1, 'n_estimators': 500},
  {'max_features': 1, 'max_samples': 2, 'n_estimators': 10},
  {'max_features': 1, 'max_samples': 2, 'n_estimators': 250},
  {'max_features': 1, 'max_samples': 2, 'n_estimators': 500},
  {'max_features': 1, 'max_samples': 3, 'n_estimators': 10},
  {'max_features': 1, 'max_samples': 3, 'n_estimators': 250},
```

```
 {'max_features': 1, 'max_samples': 3, 'n_estimators': 500},
 {'max_features': 2, 'max_samples': 1, 'n_estimators': 10},
 {'max_features': 2, 'max_samples': 1, 'n_estimators': 250},
 {'max_features': 2, 'max_samples': 1, 'n_estimators': 500},
 {'max_features': 2, 'max_samples': 2, 'n_estimators': 10},
 {'max_features': 2, 'max_samples': 2, 'n_estimators': 250},
 {'max_features': 2, 'max_samples': 2, 'n_estimators': 500},
 {'max_features': 2, 'max_samples': 3, 'n_estimators': 10},
 {'max_features': 2, 'max_samples': 3, 'n_estimators': 250},
 {'max_features': 2, 'max_samples': 3, 'n_estimators': 500},
 {'max_features': 3, 'max_samples': 1, 'n_estimators': 10},
 {'max_features': 3, 'max_samples': 1, 'n_estimators': 250},
 {'max_features': 3, 'max_samples': 1, 'n_estimators': 500},
 {'max_features': 3, 'max_samples': 2, 'n_estimators': 10},
 {'max_features': 3, 'max_samples': 2, 'n_estimators': 250},
 {'max_features': 3, 'max_samples': 2, 'n_estimators': 500},
 {'max_features': 3, 'max_samples': 3, 'n_estimators': 10},
 {'max_features': 3, 'max_samples': 3, 'n_estimators': 250},
 {'max_features': 3, 'max_samples': 3, 'n_estimators': 500}],
'split0_test_score': array([0.5       , 0.5       , 0.5       , 0.49903685,
0.56162534,
       0.62702098, 0.54336659, 0.58455168, 0.62564622, 0.5       ,
       0.5       , 0.5       , 0.52594751, 0.628618  , 0.68414337,
       0.58927689, 0.6272844 , 0.67597715, 0.5       , 0.5       ,
       0.5       , 0.42015707, 0.6618509 , 0.67547499, 0.47884356,
       0.68248049, 0.69868945]),
'split1_test_score': array([0.5       , 0.5       , 0.5       , 0.46073031,
0.52895336,
       0.62839755, 0.50806505, 0.65870332, 0.6964969 , 0.5       ,
       0.5       , 0.5       , 0.4705601 , 0.60031831, 0.62243535,
       0.62953556, 0.69646391, 0.67432214, 0.5       , 0.5       ,
       0.5       , 0.51234497, 0.6481231 , 0.64484101, 0.63929938,
       0.73187426, 0.72033745]),
'split2_test_score': array([0.5       , 0.5       , 0.5       , 0.5       ,
0.53548456,
       0.58888871, 0.56209592, 0.63067357, 0.62801821, 0.5       ,
       0.5       , 0.5       , 0.50074218, 0.59044729, 0.53654011,
       0.55389068, 0.65711176, 0.65074548, 0.5       , 0.5       ,
       0.5       , 0.50074218, 0.64738092, 0.60508477, 0.55389068,
       0.65154539, 0.63515141]),
'split3_test_score': array([0.5       , 0.5       , 0.5       , 0.5       ,
0.52394775,
       0.55395666, 0.55266196, 0.60707217, 0.62260028, 0.5       ,
       0.5       , 0.5       , 0.51314487, 0.5338765 , 0.50832069,
       0.59312739, 0.68017713, 0.65227108, 0.5       , 0.5       ,
       0.5       , 0.51314487, 0.62357336, 0.61009863, 0.59312739,
       0.67181521, 0.65801062]),
```

```
 'split4_test_score': array([0.5       , 0.5       , 0.5       , 0.48458208,
0.53912754,
       0.56872294, 0.56056444, 0.57452547, 0.63443223, 0.5       ,
       0.5       , 0.5       , 0.53145188, 0.55370463, 0.51568432,
       0.57701465, 0.61058941, 0.63830336, 0.5       , 0.5       ,
       0.5       , 0.53145188, 0.5962704 , 0.58409091, 0.54729437,
       0.63095238, 0.65512821]),
 'mean_test_score': array([0.5       , 0.5       , 0.5       , 0.48887411,
0.53783435,
       0.59341456, 0.54534572, 0.6111082 , 0.64143618, 0.5       ,
       0.5       , 0.5       , 0.50836769, 0.58141504, 0.57347446,
       0.58857265, 0.65433025, 0.65833496, 0.5       , 0.5       ,
       0.5       , 0.49553536, 0.63545908, 0.62394502, 0.56247089,
       0.67374875, 0.67347628]),
 'std_test_score': array([0.        , 0.        , 0.        , 0.01524121,
0.01300575,
       0.03013653, 0.01979462, 0.03068224, 0.02780356, 0.        ,
       0.        , 0.        , 0.02168863, 0.0337549 , 0.06871212,
       0.02463912, 0.03191877, 0.01457628, 0.        , 0.        ,
       0.        , 0.03898238, 0.02312929, 0.03233987, 0.05319754,
       0.03400143, 0.03123059]),
 'rank_test_score': array([17, 17, 17, 27, 15,  9, 14,  8,  5, 17, 17, 17, 16,
11, 12, 10,  4,
        3, 17, 17, 17, 26,  6,  7, 13,  1,  2], dtype=int32),
 'split0_train_score': array([0.5       , 0.5       , 0.5       , 0.5346437 ,
0.56625791,
       0.60003213, 0.54748022, 0.58377143, 0.60073267, 0.5       ,
       0.5       , 0.5       , 0.53137761, 0.62979219, 0.64572484,
       0.56562247, 0.6118752 , 0.63737363, 0.5       , 0.5       ,
       0.5       , 0.5021817 , 0.63783033, 0.65480707, 0.53049728,
       0.66672246, 0.68162445]),
 'split1_train_score': array([0.5       , 0.5       , 0.5       , 0.45808043,
0.53799754,
       0.6019936 , 0.488568  , 0.63536215, 0.65559408, 0.5       ,
       0.5       , 0.5       , 0.45798386, 0.60744108, 0.60728616,
       0.58187219, 0.66778931, 0.66143815, 0.5       , 0.5       ,
       0.5       , 0.47359661, 0.64248275, 0.63412022, 0.57653367,
       0.69976669, 0.6953169 ]),
 'split2_train_score': array([0.5       , 0.5       , 0.5       , 0.5       ,
0.54855631,
       0.62974169, 0.53603626, 0.62556662, 0.65825972, 0.5       ,
       0.5       , 0.5       , 0.50356314, 0.59792026, 0.58587427,
       0.5593423 , 0.69041835, 0.67168398, 0.5       , 0.5       ,
       0.5       , 0.50356314, 0.68937162, 0.6501528 , 0.5593423 ,
       0.68915576, 0.67432587]),
 'split3_train_score': array([0.5       , 0.5       , 0.5       , 0.5       ,
0.49971133,
```

```
        0.6057979 , 0.54977036, 0.60058952, 0.66371546, 0.5       ,
        0.5       , 0.5       , 0.50045391, 0.49061448, 0.55524262,
        0.55815407, 0.67402688, 0.69218237, 0.5       , 0.5       ,
        0.5       , 0.50045391, 0.56886621, 0.64323256, 0.55815407,
        0.66152955, 0.69389371]),
 'split4_train_score': array([0.5       , 0.5       , 0.5       , 0.45438551,
0.53730413,
        0.63466528, 0.51347142, 0.54929544, 0.66685316, 0.5       ,
        0.5       , 0.5       , 0.56625669, 0.58663563, 0.58731721,
        0.57905065, 0.62777269, 0.68529606, 0.5       , 0.5       ,
        0.5       , 0.56625669, 0.61225807, 0.64905011, 0.59164936,
        0.64982958, 0.71173197]),
 'mean_train_score': array([0.5       , 0.5       , 0.5       , 0.48942193,
0.53796544,
        0.61444612, 0.52706525, 0.59891703, 0.64903102, 0.5       ,
        0.5       , 0.5       , 0.51192704, 0.58248073, 0.59628902,
        0.56880833, 0.65437649, 0.66959484, 0.5       , 0.5       ,
        0.5       , 0.50921041, 0.6301618 , 0.64627255, 0.56323533,
        0.67340081, 0.69137858]),
 'std_train_score': array([0.        , 0.        , 0.        , 0.02992871,
0.02180113,
        0.01469958, 0.02314996, 0.03076666, 0.02447135, 0.        ,
        0.        , 0.        , 0.03589287, 0.04807586, 0.02979465,
        0.00988775, 0.02959452, 0.01931749, 0.        , 0.        ,
        0.        , 0.03059605, 0.03948191, 0.00710676, 0.02047905,
        0.01835488, 0.01282213])}
```

[290]:
```
bg.best_estimator_
```

[290]:
```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
                  max_features=3, max_samples=3, n_estimators=250, n_jobs=None,
                  oob_score=False, random_state=0, verbose=0, warm_start=False)
```

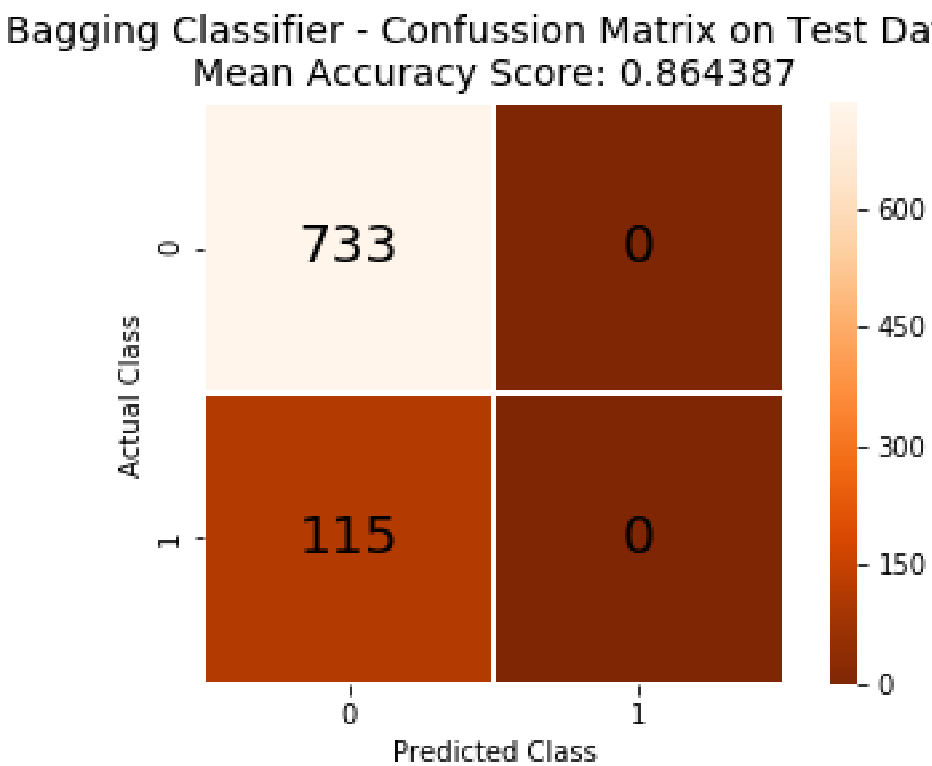
[291]:
```
bg_pred = bg.best_estimator_.predict(X_test)
bg_prob = bg.best_estimator_.predict_proba(X_test)
bg_prob
```

[291]:
```
array([[0.816, 0.184],
       [0.814, 0.186],
       [0.838, 0.162],
       …,
       [0.864, 0.136],
       [0.822, 0.178],
       [0.846, 0.154]])
```

[292]:
```
bg_matrix = metrics.confusion_matrix(y_test, bg_pred)
bg_matrix
```

[292]:
```
array([[733,   0],
       [115,   0]])
```

[293]:
```
import matplotlib.pyplot as plt
bg_test = bg.best_estimator_.score(X_test, y_test)

bg_matrix = metrics.confusion_matrix(y_test, bg_pred)

bg_cm = pd.DataFrame(bg_matrix, range(2), range(2))
# plt.figure(figsize=(5, 8))
fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'black','fontsize':'20'}
ax = sns.heatmap (bg_cm, fmt='d',
                  cmap='Oranges_r', annot=True, square =␣
 ↪True,ax=ax,linewidths=0.5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

bg_title = 'Bagging Classifier - Confussion Matrix on Test Data \nMean Accuracy␣
 ↪Score: {0:2f}'.format(bg_test)
plt.title(bg_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



[294]:
```
print("", classification_report(y_test, bg_pred, target_names=target_names))
```

```
                    precision    recall  f1-score   support

No risk in 10 years      0.86      1.00      0.93       733
  Risky in 10 years      0.00      0.00      0.00       115

           accuracy                          0.86       848
          macro avg      0.43      0.50      0.46       848
       weighted avg      0.75      0.86      0.80       848
```

```
/usr/local/lib/python3.7/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

[295]:
```
acc_bg = accuracy_score(y_test, bg_pred)
print("Bagging Classifier accuracy:", acc_bg)
```

```
Bagging Classifier accuracy: 0.8643867924528302
```

[296]:
```
error_bg = 1-acc_bg
error_bg
```

[296]: 0.13561320754716977

[297]:
```
bg_probs = bg.best_estimator_.predict_proba(X_test)[:,1]
print(roc_auc_score(y_test, bg_probs))
```

```
0.5892816893054155
```

17.0.1 For training set:

[298]:
```
bg_pred_tr = bg.best_estimator_.predict(X_train)
bg_prob_tr = bg.best_estimator_.predict_proba(X_train)
bg_prob_tr
```

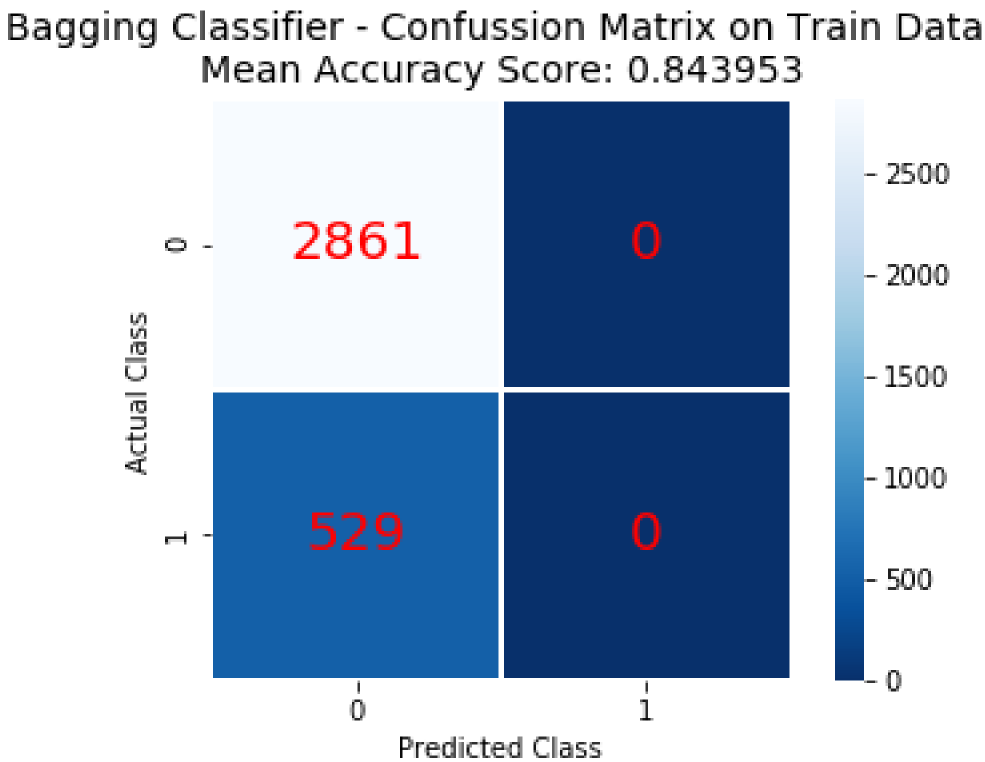
[298]:
```
array([[0.848, 0.152],
       [0.78 , 0.22 ],
       [0.854, 0.146],
       …,
       [0.848, 0.152],
       [0.848, 0.152],
       [0.8  , 0.2  ]])
```

[299]:
```
bg_matrix_tr = metrics.confusion_matrix(y_train, bg_pred_tr)
bg_matrix_tr
```

[299]:
```
array([[2861,    0],
       [ 529,    0]])
```

[300]:
```
bg_train = bg.best_estimator_.score(X_train, y_train)

bg_matrix = metrics.confusion_matrix(y_train, bg_pred_tr)

bg_cm_tr = pd.DataFrame(bg_matrix, range(2), range(2))

fig, ax = plt.subplots(figsize=(6,4))
akws = {"ha": 'center',"va": 'center','c':'red','fontsize':'20'}
ax = sns.heatmap (bg_cm_tr, fmt='d',
                  cmap='Blues_r', annot=True, square = True,ax=ax,linewidths=0.
 ↪5,annot_kws=akws)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')
```

```
bg_title = 'Bagging Classifier - Confussion Matrix on Train Data \nMean␣
↪Accuracy Score: {0:2f}'.format(bg_train)
plt.title(bg_title, size = 14)
# plt.figure(figsize=(16, 26))
plt.show;
```



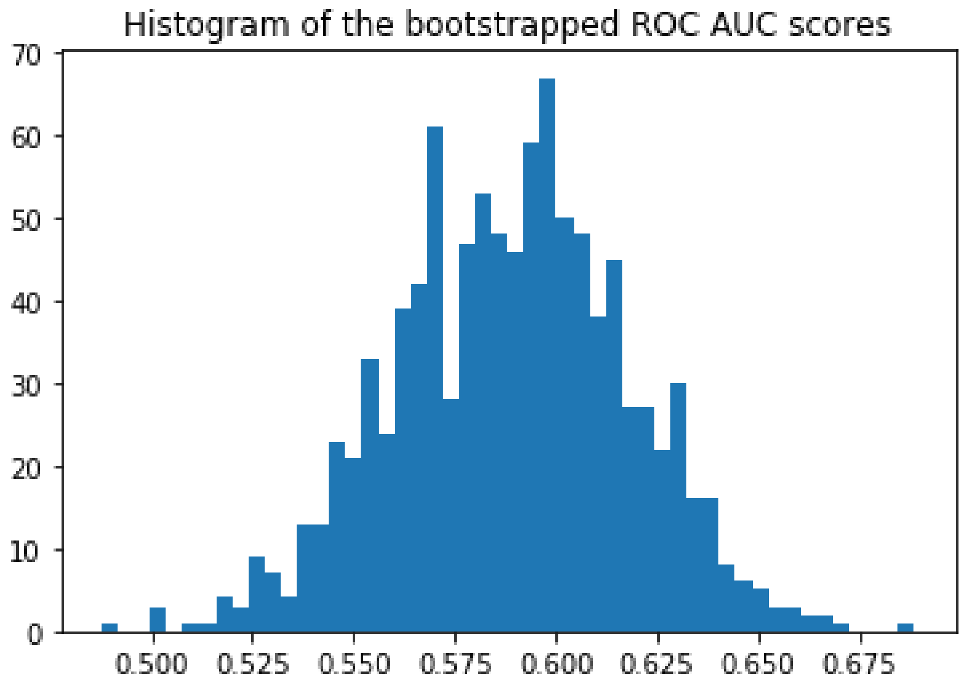
[301]:
```
print("", classification_report(y_train, bg_pred_tr, target_names=target_names))
```

```
                   precision    recall  f1-score   support

No risk in 10 years       0.84      1.00      0.92      2861
  Risky in 10 years       0.00      0.00      0.00       529

           accuracy                           0.84      3390
          macro avg       0.42      0.50      0.46      3390
       weighted avg       0.71      0.84      0.77      3390
```

```
/usr/local/lib/python3.7/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

[302]:
```python
y_pred = bg_probs
y_true = y_test_v

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.5893
```




```
Confidence interval for the score: [0.5410 - 0.6354]
```

[303]:
```
alpha = .95
y_pred = bg_probs
y_true = y_test_v2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.5892816893054155
```

```
AUC COV: 0.0008449165367543924
95% AUC CI: [0.53231054 0.64625284]
```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```