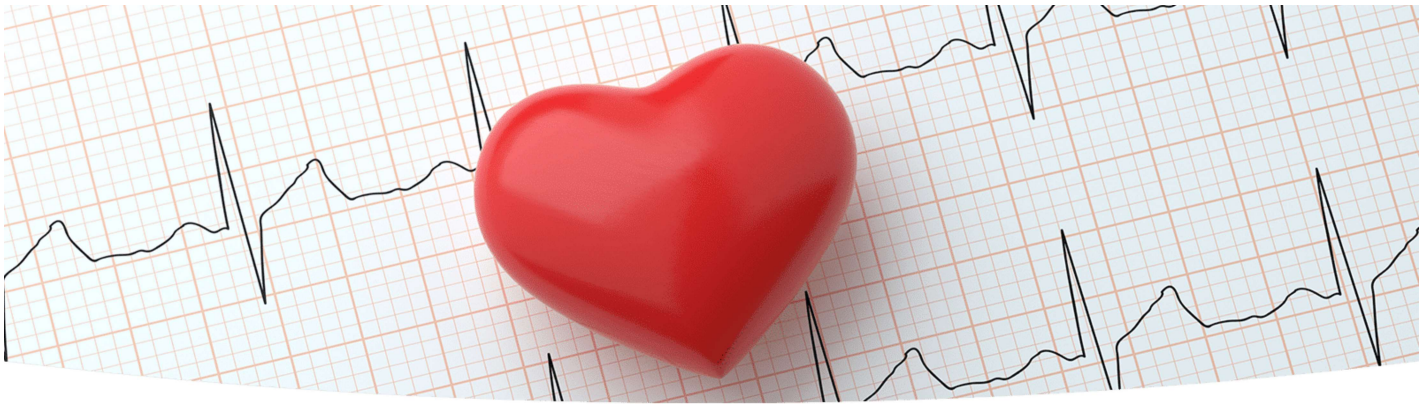
Heart Disease Detection



This notebook is a continuation of our previous notebook, in which we explored the Personal Key Indicators of Heart Disease dataset in order to gain insights about heart disease and its causses. In the this notebook we carry on with building a machine learning model to identify heart dissease patients

Table of contents

Refresh

```
!pip install pywaffle
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting pywaffle
  Downloading pywaffle-1.1.0-py2.py3-none-any.whl (30 kB)
Collecting fontawesomefree
  Downloading fontawesomefree-6.2.0-py3-none-any.whl (25.1 MB)
     |████████████████████████████████| 25.1 MB 10.8 MB/s
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from
Installing collected packages: fontawesomefree, pywaffle
Successfully installed fontawesomefree-6.2.0 pywaffle-1.1.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting catboost
  Downloading catboost-1.1-cp37-none-manylinux1_x86_64.whl (76.8 MB)
     |████████████████████████████████| 76.8 MB 15 kB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from cat
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from ca
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from catbo
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-packages
Installing collected packages: catboost
Successfully installed catboost-1.1
```

```
import shutil
from os import path

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

```
import plotly.express as px
from pywaffle.waffle import Waffle

import scipy
from scipy.stats import zscore
from imblearn.pipeline import Pipeline, make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder, StandardScaler
from sklearn.feature_selection import f_classif, chi2
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler, TomekLinks
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, VotingClassifier
from catboost import CatBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from xgboost import XGBClassifier
from sklearn.tree  import DecisionTreeClassifier

from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc


%matplotlib inline
sns.set_style("darkgrid")

colors6 = sns.color_palette(['#1337f5', '#E80000', '#0f1e41', '#fd523e', '#404e5c', '#c9bbaa'
colors2 = sns.color_palette(['#1337f5', '#E80000'], 2)
colors1 = sns.color_palette(['#1337f5'], 1)
```

1 Data Reading

```
if path.exists('./heart-disease'):
    shutil.rmtree('./heart-disease')

!git clone https://github.com/lemonpudding-datasets/heart-disease.git

shutil.move("./heart-disease/heart_2020_cleaned.csv", "./heart_2020_cleaned.csv")
shutil.rmtree('./heart-disease')
```

```
Cloning into 'heart-disease'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

```
df = pd.read_csv("heart_2020_cleaned.csv")
df.head()
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth
0	No	16.60	Yes	No	No	3.0	30.0
1	No	20.34	No	No	Yes	0.0	0.0
2	No	26.58	Yes	No	No	20.0	30.0
3	No	24.21	No	No	No	0.0	0.0
4	No	23.71	No	No	No	28.0	0.0

2 Data Cleaning

```
num_cols = df.select_dtypes(exclude='object').columns
obj_cols = df.select_dtypes(include='object').columns[1:]
```

2.1 Missing Data

For missing data we should first check if any values are null/NaN. Then we should check the unique values of every column (especially categorical) in search of any abnormal values such as "???".

2.1.1 Null Values

```
df.isna().sum()
```

```
HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
```

```
Diabetic           0
PhysicalActivity   0
GenHealth          0
SleepTime          0
Asthma             0
KidneyDisease      0
SkinCancer         0
dtype: int64
```

We Have no null values in the dataset.

2.1.2 Unusual Values

```
print(f"Unique Values for categorical columns:")
for col in df.select_dtypes(include='object'):
  print(f"  - {col}: {df[col].unique()}\n")
```

```
Unique Values for categorical columns:
  - HeartDisease: ['No' 'Yes']

  - Smoking: ['Yes' 'No']

  - AlcoholDrinking: ['No' 'Yes']

  - Stroke: ['No' 'Yes']

  - DiffWalking: ['No' 'Yes']

  - Sex: ['Female' 'Male']

  - AgeCategory: ['55-59' '80 or older' '65-69' '75-79' '40-44' '70-74' '60-64' '50-54'
 '45-49' '18-24' '35-39' '30-34' '25-29']

  - Race: ['White' 'Black' 'Asian' 'American Indian/Alaskan Native' 'Other'
 'Hispanic']

  - Diabetic: ['Yes' 'No' 'No, borderline diabetes' 'Yes (during pregnancy)']

  - PhysicalActivity: ['Yes' 'No']

  - GenHealth: ['Very good' 'Fair' 'Good' 'Poor' 'Excellent']

  - Asthma: ['Yes' 'No']

  - KidneyDisease: ['No' 'Yes']

  - SkinCancer: ['Yes' 'No']
```

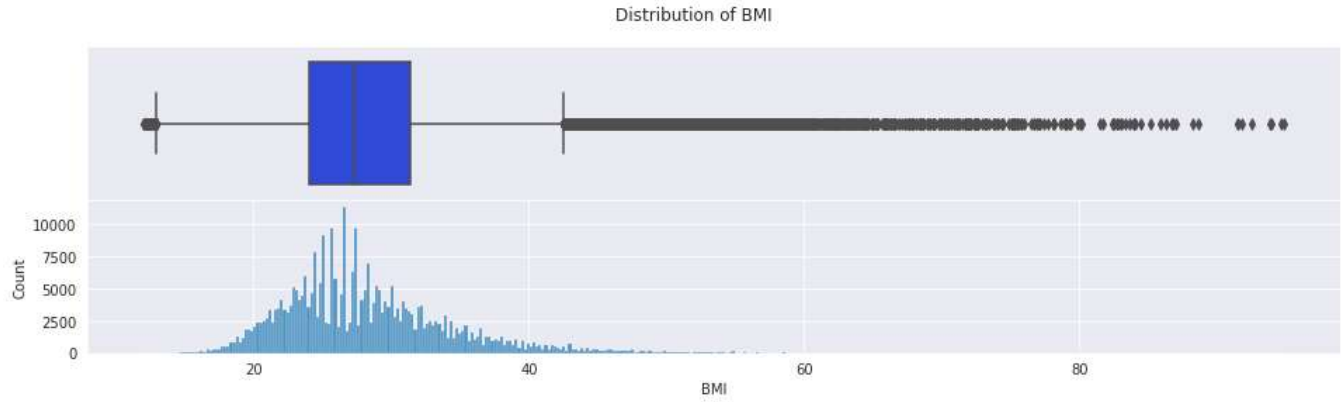
We can see for each column that all the values are valid and there are no missing values.

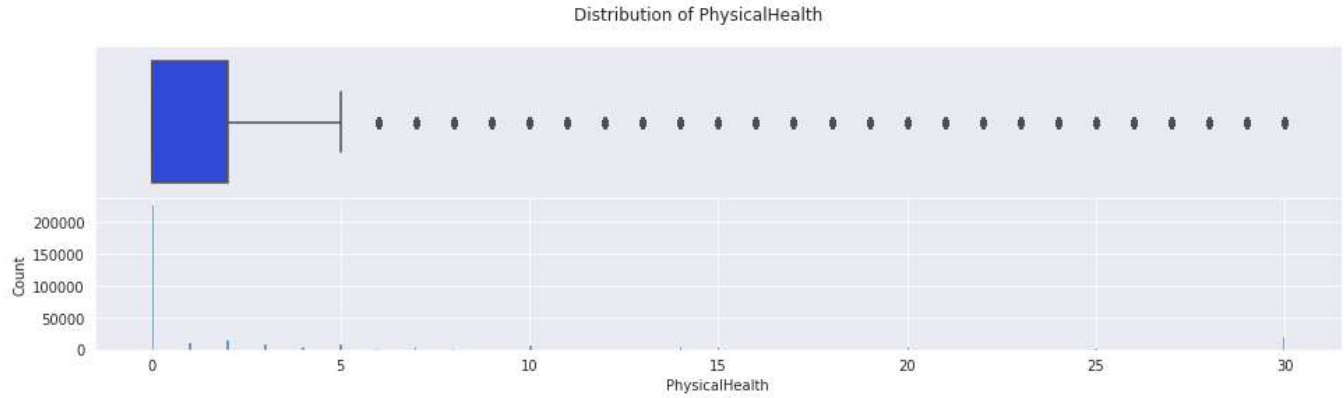
2.2 Outliers

2.2.1 Univariate Outlier Detection

We should explore outliers only in the numerical variables as there is no notion of uniariate outliers in categorical variables.

```
numerical = df.select_dtypes(exclude='object')
n = len(numerical)

for col in numerical:
  fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(16, 4))
  plt.suptitle(f"Distribution of {col}")
  sns.boxplot(data=df, x=col, ax=ax1, palette=colors1)
  ax1.set_xlabel(None)
  ax1.get_xaxis().set_ticks([])
  sns.histplot(data=df, x=col, ax=ax2, palette=colors1)
  plt.subplots_adjust(hspace=0)
  print("\n")
  plt.savefig(f"{col}.png")
  plt.show()
```





We can see that all numerical variables are skewed and contain outliers. Let's examine the IQR outliers more closely.

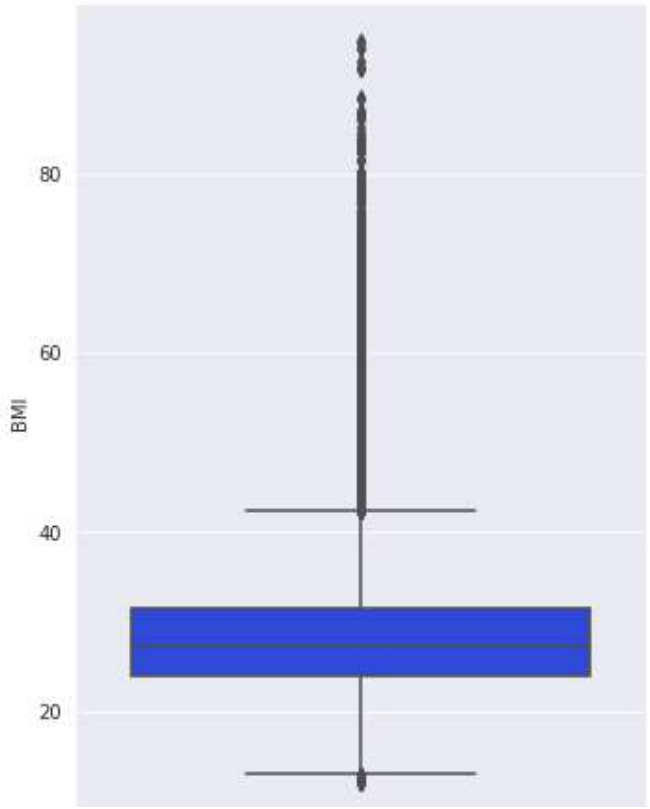
```
for col in numerical:
  Q1, Q3 = df[col].quantile([0.25,0.75])
  IQR = Q3 - Q1
  right = Q3 + 1.5 * IQR
  left = Q1 - 1.5 * IQR
  n = len(df.loc[(df[col] < left) | (df[col] > right)])
  print(f"{col}:\n\tOutlier Num = {n}\n\tOutlier Percentage = {n*100/len(df):.2f}%\n")
```

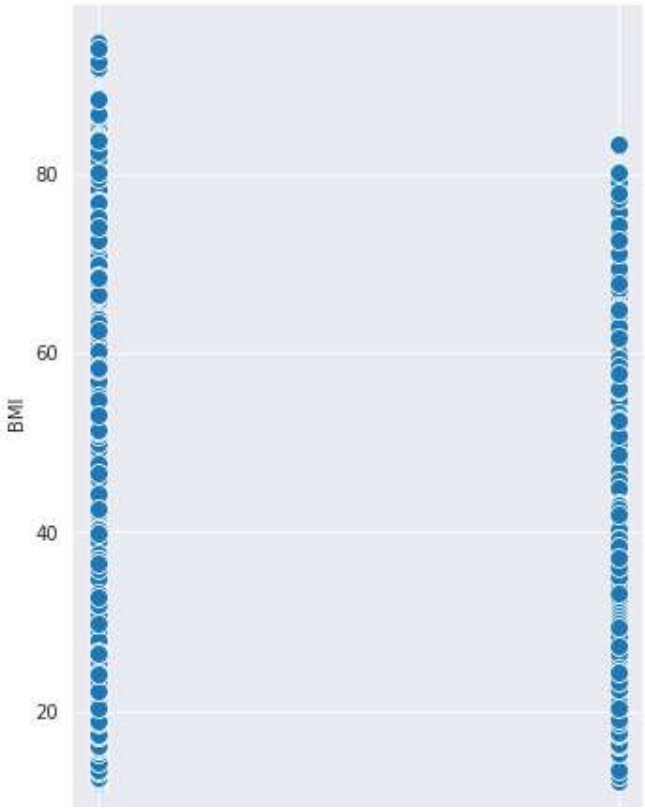
```
    BMI:
            Outlier Num = 10396
            Outlier Percentage = 3.25%

    PhysicalHealth:
            Outlier Num = 47146
            Outlier Percentage = 14.74%
```

```
        MentalHealth:
                Outlier Num = 51576
                Outlier Percentage = 16.13%

        SleepTime:
                Outlier Num = 4543
                Outlier Percentage = 1.42%
```

2.2.2 Multivariate Outlier Detection

```
num_cols = df.select_dtypes(exclude='object').columns
for col in num_cols:
    fig, ax =plt.subplots(1,2, figsize=(12,8))
    sns.boxplot(data=df, y=col, ax=ax[0], palette=colors1)
    sns.scatterplot(data=df,x = 'HeartDisease', s = 100, y=col, ax=ax[1], palette=colors1)
    plt.savefig(f"{col}.png")
    plt.show()
```





2.2.3 Outlier Removal

```
zs = df.copy()
for col in num_cols:
  zs[col] = zscore(zs[col])
outscores = np.abs(zs[num_cols])
no_outliers = (outscores<=3.3).all(axis=1)
df2 = df[no_outliers]
```

2.3 Categorical Encoding & Feature Scaling

```
trans = make_column_transformer(
    (OrdinalEncoder(), obj_cols),
    (StandardScaler(), num_cols),
    remainder='passthrough'
)

trans
```

```
     ColumnTransformer(remainder='passthrough',
                       transformers=[('ordinalencoder', OrdinalEncoder(),
                                      Index(['Smoking', 'AlcoholDrinking', 'Stroke',
     'DiffWalking', 'Sex',
            'AgeCategory', 'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth',
            'Asthma', 'KidneyDisease', 'SkinCancer'],
```

```
            dtype='object')),
                                    ('standardscaler', StandardScaler(),
                                     Index(['BMI', 'PhysicalHealth', 'MentalHealth',
       'SleepTime'], dtype='object'))])
```

2.4 Train-Test Split

```
X = df2.drop(columns=['HeartDisease'])
y =  df2['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

The test set contains 73,600 observations.

3 Modeling

Helpers

Show code

```
# Let's make function to make using different algorithms easy.
def try_model(transformation, sampler, model, X_train, y_train, X_test, y_test):
    model_pipeline = Pipeline(steps=[('transform', transformation),
                                     ('sample', sampler),
                                     ('model', model)])

    model_pipeline.fit(X_train, y_train)
    y_pred = model_pipeline.predict(X_test)

    print(f'Training Accuracy : {model_pipeline.score(X_train, y_train)}')
    print(f'Test Accuracy : {model_pipeline.score(X_test, y_test)}')

    return y_pred, classification_report(y_test, y_pred)
```

```
def f_importances(coef, names, top=-1):
    imp = coef
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)
```

```
    plt.figure(figsize=(16, 8), dpi=80)
    sns.barplot(x=list(imp[::-1][0:top]), y=list(names[::-1][0:top]), palette='RdYlGn')
    plt.yticks(range(top), names[::-1][0:top])
    plt.title("Feature Importance");
    plt.show()


def plot_roc_auc(model, X_test, y_test, y_pred=None):
  if(y_pred is None):
    y_pred =  model.decision_function(X_test)

  fpr, tpr, thresholds = roc_curve(y_test, y_pred)

  roc_auc = auc(fpr, tpr)

  plt.title('Receiver Operating Characteristic')
  plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
  plt.legend(loc = 'lower right')
  plt.plot([0, 1], [0, 1],'r--')
  plt.axis('tight')
  plt.ylabel('True Positive Rate')
  plt.xlabel('False Positive Rate')
  return fpr, tpr, thresholds


model_metrics = {
           'Model':[],
           'Sampling Method':[],
           'Heart Disease Precision':[],
           'No Heart Disease Precision':[],
           'Heart Disease Recall':[],
           'No Heart Disease Recall':[],
           'Heart Disease F1':[],
           'No Heart Disease F1':[],
           'Accuracy':[]}
```

3.1 Logistic Regression

3.1.1 Model

```
y_pred, report = try_model(trans,
                           RandomUnderSampler(random_state = 42),
                           LogisticRegression(C = 10),
                           X_train,
                           y_train,
```

```
                          X_test,
                          y_test)

    Training Accuracy : 0.7304088296701555
    Test Accuracy : 0.7322318990231111
```

3.1.2 Evaluation

3.1.2.1 Classification Report

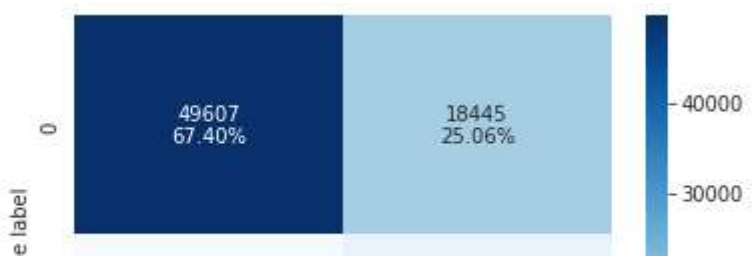
```
print(report)
```

```
                  precision    recall  f1-score   support

              No       0.98      0.73      0.83     68052
             Yes       0.19      0.77      0.30      5549

        accuracy                           0.73     73601
       macro avg       0.58      0.75      0.57     73601
    weighted avg       0.92      0.73      0.79     73601
```

```
model_metrics['Model'].append("Logistic Regression")
model_metrics['Sampling Method'].append("SMOTE")
model_metrics['Heart Disease Precision'].append(0.19)
model_metrics['No Heart Disease Precision'].append(0.98)
model_metrics['Heart Disease Recall'].append(0.78)
model_metrics['No Heart Disease Recall'].append(0.72)
model_metrics['Heart Disease F1'].append(0.30)
model_metrics['No Heart Disease F1'].append(0.83)
model_metrics['Accuracy'].append(0.73)
```

3.1.2.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
make_confusion_matrix(cm)
```



3.2 Decision Tree

3.2.1 Model

```
dt = DecisionTreeClassifier(max_features=17, max_depth=10)

y_pred, report = try_model(trans,
                           SMOTE(random_state = 42),
                           dt,
                           X_train,
                           y_train,
                           X_test,
                           y_test)
```

3.2.2 Evaluation

3.2.2.1 Classification Report

```
print(report)


model_metrics['Model'].append("Decision Tree")
model_metrics['Sampling Method'].append("SMOTE")
model_metrics['Heart Disease Precision'].append(0.22)
model_metrics['No Heart Disease Precision'].append(0.96)
model_metrics['Heart Disease Recall'].append(0.63)
model_metrics['No Heart Disease Recall'].append(0.82)
model_metrics['Heart Disease F1'].append(0.32)
model_metrics['No Heart Disease F1'].append(0.88)
model_metrics['Accuracy'].append(0.80)
```

3.2.2.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
make_confusion_matrix(cm)
```

3.2.3 Feature Importances

```
features_names = df2.columns.drop('HeartDisease').to_list()
f_importances(abs(dt.feature_importances_), features_names, top=16)
```

3.3 Random Forest

3.3.1 Model

```
rf = RandomForestClassifier(max_features=17, max_depth=10)

y_pred, report = try_model(trans,
                           SMOTE(random_state = 42),
                           rf,
                           X_train,
                           y_train,
                           X_test,
                           y_test)
```

3.3.2 Evaluation

3.3.2.1 Classification Report

```
print(report)
```

```
model_metrics['Model'].append("Random Forest")
model_metrics['Sampling Method'].append("SMOTE")
model_metrics['Heart Disease Precision'].append(0.22)
model_metrics['No Heart Disease Precision'].append(0.96)
model_metrics['Heart Disease Recall'].append(0.63)
model_metrics['No Heart Disease Recall'].append(0.82)
model_metrics['Heart Disease F1'].append(0.33)
model_metrics['No Heart Disease F1'].append(0.89)
model_metrics['Accuracy'].append(0.81)
```

3.3.2.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
```

```
make_confusion_matrix(cm)
```

3.3.3 Feature Importances

```
f_importances(abs(rf.feature_importances_), features_names, top=16)
```

3.4 KNN

3.4.1 Hyperparameter Tuning

```
err = []
k = list(range(1, 21))

for i in k:

  neigh_undersample = Pipeline(steps=[('transform', trans),
                                      ('sample', RandomUnderSampler(random_state = 42)),
                                      ('KNN', KNeighborsClassifier(n_neighbors=i))])

  res = cross_val_score(neigh_undersample, X_train.iloc[:20000], y_train.iloc[:20000], scorin

  err.append(1 - res.mean())


plt.figure(figsize=(16, 6), dpi=80)
fig = sns.lineplot(x=k, y=err, palette=colors1);
fig.set_title("KNN Validation F1 Error as K Increases")
fig.set_xlabel("Number of Neighbors")
fig.set_ylabel("Validation Error")
fig.set_xticks(k);
```

The error fluctuates and it does look like it is less with even K. However to avoid ties we will set the number of neighbors to 19.

3.4.2 Model

```
y_pred, report = try_model(trans,
                           RandomUnderSampler(random_state = 42),
                           KNeighborsClassifier(n_neighbors=19),
                           X_train,
                           y_train,
```

```
                                X_test,
                                y_test)
```

3.4.3 Evaluation

3.4.3.1 Classification Report

```
print(report)


model_metrics['Model'].append("KNN")
model_metrics['Sampling Method'].append("Random Undersampling")
model_metrics['Heart Disease Precision'].append(0.19)
model_metrics['No Heart Disease Precision'].append(0.97)
model_metrics['Heart Disease Recall'].append(0.76)
model_metrics['No Heart Disease Recall'].append(0.73)
model_metrics['Heart Disease F1'].append(0.30)
model_metrics['No Heart Disease F1'].append(0.83)
model_metrics['Accuracy'].append(0.73)
```

3.4.3.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
make_confusion_matrix(cm)
```

The false positive rate is really high. However in our application (Heart disease prediction), it is much more desirable than false negatives.

3.5 SVM

3.5.1 Hyperparameter Tuning

```
svm_undersample = Pipeline(steps=[('transform', trans),
                                  ('sample', RandomUnderSampler(random_state = 42)),
                                  ('SVM', SVC())])

svm_dist1 = {
    "SVM__C": scipy.stats.expon(scale=.1),
    "SVM__gamma": scipy.stats.expon(scale=.01),
    "SVM__kernel": ["rbf"],
}
```

```
svm_dist2 = {
    "SVM__C": scipy.stats.expon(scale=1),
    "SVM__gamma": scipy.stats.expon(scale=.01),
    "SVM__kernel": ["rbf"],
}

svm_dist3 = {
    "SVM__C": scipy.stats.expon(scale=10),
    "SVM__gamma": scipy.stats.expon(scale=.01),
    "SVM__kernel": ["rbf"],
}

svm_dist4 = {
    "SVM__C": scipy.stats.expon(scale=100),
    "SVM__gamma": scipy.stats.expon(scale=.01),
    "SVM__kernel": ["rbf"],
}

svm_dist5 = {
    "SVM__C": scipy.stats.expon(scale=1000),
    "SVM__gamma": scipy.stats.expon(scale=.01),
    "SVM__kernel": ["rbf"],
}

svm_dist6 = {
    "SVM__C": scipy.stats.expon(scale=.1),
    "SVM__gamma": scipy.stats.expon(scale=1),
    "SVM__kernel": ["rbf"],
}

svm_dist7 = {
    "SVM__C": scipy.stats.expon(scale=1),
    "SVM__gamma": scipy.stats.expon(scale=1),
    "SVM__kernel": ["rbf"],
}

svm_dist8 = {
    "SVM__C": scipy.stats.expon(scale=10),
    "SVM__gamma": scipy.stats.expon(scale=1),
    "SVM__kernel": ["rbf"],
}

svm_dist9 = {
    "SVM__C": scipy.stats.expon(scale=100),
    "SVM__gamma": scipy.stats.expon(scale=1),
    "SVM__kernel": ["rbf"],
}

svm_dist10 = {
    "SVM__C": scipy.stats.expon(scale=1000),
```

```
    "SVM__gamma": scipy.stats.expon(scale=1),
    "SVM__kernel": ["rbf"],
}

svm_dist11 = {
    "SVM__C": scipy.stats.expon(scale=.1),
    "SVM__gamma": scipy.stats.expon(scale=10),
    "SVM__kernel": ["rbf"],
}

svm_dist12 = {
    "SVM__C": scipy.stats.expon(scale=1),
    "SVM__gamma": scipy.stats.expon(scale=10),
    "SVM__kernel": ["rbf"],
}

svm_dist13 = {
    "SVM__C": scipy.stats.expon(scale=10),
    "SVM__gamma": scipy.stats.expon(scale=10),
    "SVM__kernel": ["rbf"],
}

svm_dist14 = {
    "SVM__C": scipy.stats.expon(scale=100),
    "SVM__gamma": scipy.stats.expon(scale=10),
    "SVM__kernel": ["rbf"],
}

svm_dist15 = {
    "SVM__C": scipy.stats.expon(scale=1000),
    "SVM__gamma": scipy.stats.expon(scale=10),
    "SVM__kernel": ["rbf"],
}

svm_dist16 = {
    "SVM__C": scipy.stats.expon(scale=10),
    "SVM__degree": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "SVM__kernel": ["poly"],
}

distributions = [svm_dist1, svm_dist2, svm_dist3, svm_dist4, svm_dist5,
                 svm_dist6, svm_dist7, svm_dist8, svm_dist9, svm_dist10,
                 svm_dist11, svm_dist12, svm_dist13, svm_dist14, svm_dist15,
                 svm_dist16]

clf = RandomizedSearchCV(svm_undersample, distributions, random_state=42, scoring='f1_micro',

clf.fit(X_train[:10000], y_train[:10000])

clf.best_params_
```

3.5.2 Model

```
svm = SVC(C=0.05, gamma=0.477)

y_pred, report = try_model(trans,
                           RandomUnderSampler(random_state = 42),
                           svm,
                           X_train,
                           y_train,
                           X_test,
                           y_test)
```

3.5.3 Evaluation

3.5.3.1 Classification Report

```
print(report)
```

```
model_metrics['Model'].append("SVM")
model_metrics['Sampling Method'].append("Random Undersampling")
model_metrics['Heart Disease Precision'].append(0.14)
model_metrics['No Heart Disease Precision'].append(0.98)
model_metrics['Heart Disease Recall'].append(0.87)
model_metrics['No Heart Disease Recall'].append(0.56)
model_metrics['Heart Disease F1'].append(0.24)
model_metrics['No Heart Disease F1'].append(0.72)
model_metrics['Accuracy'].append(0.59)
```

3.5.3.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
make_confusion_matrix(cm)
```

3.5.3.3 ROC Curve

```
svm_pipeline = Pipeline(steps=[('transform', trans),
                               ('sample', RandomUnderSampler(random_state = 42)),
                               ('model', svm)])
```

```
fpr, tpr, thresholds = plot_roc_auc(svm_pipeline, X_test, y_test.map({'No':0,
```

```
top_left = np.sqrt((1 - tpr)**2 + tpr**2)
threshold_df = pd.DataFrame({"Threshold":thresholds, "TPR":tpr, "FPR":fpr, "Distance":top_lef
threshold_df = threshold_df.sort_values("Distance")
threshold_df
```

The optimal threshold for classification seems to be around 0.66. Meaning, any sample decision function that is greater than or equal 0.66 is classified as 1 (heart disease), and otherwise 0 (normal).

3.6 AdaBoost

3.6.1 Model

```
y_pred, report = try_model(trans,
                           RandomUnderSampler(random_state = 42),
                           AdaBoostClassifier(base_estimator=DecisionTreeClassifier()),
                           X_train,
                           y_train,
                           X_test,
                           y_test)
```

3.6.2 Evaluation

3.6.2.1 Classification Report

```
print(report)
```

```
model_metrics['Model'].append("AdaBoost")
model_metrics['Sampling Method'].append("Random Undersampling")
model_metrics['Heart Disease Precision'].append(0.17)
model_metrics['No Heart Disease Precision'].append(0.97)
model_metrics['Heart Disease Recall'].append(0.73)
model_metrics['No Heart Disease Recall'].append(0.70)
model_metrics['Heart Disease F1'].append(0.81)
model_metrics['No Heart Disease F1'].append(0.27)
model_metrics['Accuracy'].append(0.70)
```

3.6.2.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
make_confusion_matrix(cm)
```

3.7 XGBoost

3.7.1 Model

```
y_pred, report = try_model(trans,
                           RandomUnderSampler(random_state = 42),
                           XGBClassifier(),
                           X_train,
                           y_train,
                           X_test,
                           y_test)
```

3.7.2 Evaluation

3.7.2.1 Classification Report

```
print(report)
```

```
model_metrics['Model'].append("XGBoost")
model_metrics['Sampling Method'].append("Random Undersampling")
model_metrics['Heart Disease Precision'].append(0.19)
model_metrics['No Heart Disease Precision'].append(0.98)
model_metrics['Heart Disease Recall'].append(0.80)
model_metrics['No Heart Disease Recall'].append(0.73)
model_metrics['Heart Disease F1'].append(0.31)
model_metrics['No Heart Disease F1'].append(0.84)
model_metrics['Accuracy'].append(0.73)
```

3.7.2.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
make_confusion_matrix(cm)
```

3.8 CatBoost

3.8.1 Model

```
y_pred, report = try_model(trans,
                           SMOTE(random_state = 42),
                           CatBoostClassifier(),
                           X_train,
                           y_train,
                           X_test,
                           y_test)
```

3.8.2 Evaluation

3.8.2.1 Classification Report

```
print(report)
```

```
model_metrics['Model'].append("CatBoost")
model_metrics['Sampling Method'].append("SMOTE")
model_metrics['Heart Disease Precision'].append(0.28)
model_metrics['No Heart Disease Precision'].append(0.94)
model_metrics['Heart Disease Recall'].append(0.21)
model_metrics['No Heart Disease Recall'].append(0.96)
model_metrics['Heart Disease F1'].append(0.24)
model_metrics['No Heart Disease F1'].append(0.95)
model_metrics['Accuracy'].append(0.90)
```

3.8.2.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
make_confusion_matrix(cm)
```

3.9 Voting

3.5.1 Model

```
voting = VotingClassifier(estimators=[('lr', LogisticRegression(C=1000)),
                                      ('dt', DecisionTreeClassifier(max_features=17,max_depth
                                      ('knn', KNeighborsClassifier(n_neighbors=19)),
```

```
                                        ('svm', SVC(C=0.05, gamma=0.477)),
                                        ('xgb', XGBClassifier(learning_rate= 0.5, max_depth=1,
                                        voting='hard')

y_pred, report = try_model(trans,
                           RandomUnderSampler(random_state = 42),
                           voting,
                           X_train,
                           y_train,
                           X_test,
                           y_test)
```

3.5.2 Evaluation

3.5.2.1 Classification Report

```
print(report)
```

```
model_metrics['Model'].append("Voting Classifier")
model_metrics['Sampling Method'].append("Random Undersampling")
model_metrics['Heart Disease Precision'].append(0.19)
model_metrics['No Heart Disease Precision'].append(0.98)
model_metrics['Heart Disease Recall'].append(0.81)
model_metrics['No Heart Disease Recall'].append(0.71)
model_metrics['Heart Disease F1'].append(0.30)
model_metrics['No Heart Disease F1'].append(0.82)
model_metrics['Accuracy'].append(0.72)
```

3.5.2.2 Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
make_confusion_matrix(cm)
```

3.10 Model Comparison

In our application, reducing false negatives is paramount. It could be the difference between life and death. On the other hand, false positives are not as important, as further tests to the patient would reveal the misdiagnosis. Therefore, we are looking for high recall for the heart diesease class. Below is a comparison of the models side by side using varoius metrics.

```
comparison = pd.DataFrame(model_metrics)
```

```
comparison = comparison.set_index('Model')
comparison
```

3.10.1 Highlights

- The CatBoost model with SMOTE yielded the highest percision (28%) for the heart disease class (i.e. percentage of heart disease classifications that were actually correct). They also yielded the best over all accuracy. However, this is at the expense of heart disease recall being the lowest (21%) (i.e. only 21% of heart disease patients correctly identified).
- The SVM model with random undersampling returned the highest recall (87%) for the heart diease class, however it has the lowest precision for heart diease (14%) (i.e. out of all heart disease classifications 14% were correct).
- Random forest and decision tree models with SMOTE had a moderate recall for the heart disease class (63%) and a good recall for the no heart disease class (82%). However it still suffers from low precision in the heart disease class.

In our application we care the most about identifying heart diease in order to take proactive action in saving the patient. Therefore, using the SVM model to select patients for further tests would be the best course of action for medical experts.

4 Conclusion

In 2019, around 17.9 million people died from Cardiovascular diseases. **85%** of them were caused by heart diseases. In order to identify the causes of heart disease, we investigated the Personal Key Indicators of Heart Disease which had 17 indicators of heart disease of 319,795 surveyed individuals in the U.S. During our investigation we identified that age is a major factor in heart disease. Furhtermore, heart diesease is more prominent in smokers (~12%), kidney disease (~30%), strole victims (~48%), skin cancer patients (~18%),people who have difficulty walking (~18%), and diabetics (~25%). Finally, after experimenting with different models we concluded that SVMs with random undersampling yield the best recall for the heart diseas class (87%), CatBoost with SMOTE yields the best recall for the no heart disease class (96%), and Decision tree/Random Forest had a compromise. An application of our model is to be used by medical experts in selecting the patients suspected of heart disease in order to conduct further testing on them.