```
import sys
!{sys.executable} -m pip install -U pandas-profiling[notebook]
!jupyter nbextension enable --py widgetsnbextension
!pip install matplotlib
!pip install graphviz
```

```
Requirement already satisfied: pandas-profiling[notebook] in /usr/local/lib/python3.7/di
WARNING: pandas-profiling 1.4.1 does not provide the extra 'notebook'
Requirement already satisfied: jinja2>=2.8 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: matplotlib>=1.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-package
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: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (f
Enabling notebook extension jupyter-js-widgets/extension...
      - Validating: OK
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (3.2
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
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: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (0.10
```

```
from google.colab import files
uploaded = files.upload()
```

Choose files stroke preprocessed.arff

- **stroke preprocessed.arff**(n/a) - 368681 bytes, last modified: 06/03/2022 - 100% done

```
Saving stroke preprocessed.arff to stroke preprocessed (1).arff
```

```
import pandas as pd
from scipy.io import arff
import numpy as np


data_file = "stroke preprocessed.arff"
data = arff.loadarff(data_file)


df = pd.DataFrame(data[0])
for col in df.columns:
  if df[col].dtype == 'object':
    # Ensure data isn't read as bytes but rather as strings from file
    df[col] = df[col].str.decode('utf-8')
```

```
# Examine data types
print(df.dtypes)
```

```
"id"                   float64
"gender"                object
"age"                  float64
"hypertension"          object
"heart_disease"         object
"ever_married"          object
"work_type"             object
"residence_type"        object
"avg_glucose_level"    float64
"bmi"                  float64
"smoking_status"        object
"stroke"                object
dtype: object
```

```
# Display first 10 rows
df.head(10)
```

```
# Examine meta info about data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
```

```
---  ------                --------------  -----
 0   "id"                  5110 non-null   float64
 1   "gender"              5110 non-null   object
 2   "age"                 5110 non-null   float64
 3   "hypertension"        5110 non-null   object
 4   "heart_disease"       5110 non-null   object
 5   "ever_married"        5110 non-null   object
 6   "work_type"           5110 non-null   object
 7   "residence_type"      5110 non-null   object
 8   "avg_glucose_level"   5110 non-null   float64
 9   "bmi"                 5110 non-null   float64
 10  "smoking_status"      5110 non-null   object
 11  "stroke"              5110 non-null   object
dtypes: float64(4), object(8)
memory usage: 479.2+ KB
```

```
# The original 201 null values were all from bmi column, and they have been replaced by place
# Convert the 5000 values back into null values
df = df.replace(5000.0, np.nan)
```

```
# Check head of dataset again
df.head(10)
```

```
# Check structure of data types to ensure bmi remains float
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   "id"                 5110 non-null   float64
 1   "gender"             5110 non-null   object 
 2   "age"                5110 non-null   float64
 3   "hypertension"       5110 non-null   object 
 4   "heart_disease"      5110 non-null   object 
 5   "ever_married"       5110 non-null   object 
 6   "work_type"          5110 non-null   object 
 7   "residence_type"     5110 non-null   object 
 8   "avg_glucose_level"  5110 non-null   float64
 9   "bmi"                4909 non-null   float64
 10  "smoking_status"     5110 non-null   object 
 11  "stroke"             5110 non-null   object 
dtypes: float64(4), object(8)
memory usage: 479.2+ KB
```

```
# Remove records with NAs from dataset
df_noNA = df
df_noNA = df_noNA.dropna()
df_noNA.head(10)
```

```
# Modify 'stroke' column so that 0 is coded as "No Stroke" and 1 is coded as "Stroke"
df_noNA.loc[df_noNA['"stroke"'] == '0', '"stroke"'] = "No Stroke"
```

```
df_noNA.loc[df_noNA['"stroke"'] == '1', '"stroke"'] = "Stroke"
df_noNA.head(10)
```

```
# See if there are any extreme values in numeric data
df_noNA.describe()
```

```
# Create list of categorical columns
cat_cols = ['"gender"', '"hypertension"', '"heart_disease"', '"ever_married"', '"work_type"',
```

```
# Create copy of a data frame in memory w/ a different name
df_dummy = df_noNA.copy()
# Convert only categorical feature into dummy/one-hot features
df_dummy = pd.get_dummies(df_noNA, columns = cat_cols, prefix = cat_cols)
# Print dataset
df_dummy
```

```
# Create train test set split
from sklearn.model_selection import train_test_split


# Set class name as "stroke", all else will be used as features
class_col_name = '"stroke"'
dummy_feature_names = df_dummy.columns[df_dummy.columns != class_col_name]
# 70% training, 30% test set split
x_train, x_test, y_train, y_test = train_test_split(df_dummy.loc[:, dummy_feature_names], df_
```

```
# Naive Bayes modeling
from sklearn.naive_bayes import MultinomialNB



# Create Multinomial NB Classifier
nb = MultinomialNB()


# Train model using training sets
nb.fit(x_train, y_train)
```

```
     MultinomialNB()
```

```
# Predict response for test dataset
y_pred = nb.predict(x_test)


# Print Naive Bayes output
print("Number of features used: ", nb.n_features_)
print("Classes: ", nb.classes_)
print("Number of records for classes: ", nb.class_count_)
print("Log prior probability for classes: ", nb.class_log_prior_)
print("Log conditional probability for each feature given a class: ", nb.feature_log_prob_)
```

```
     Number of features used:  24
     Classes:  ['No Stroke' 'Stroke']
     Number of records for classes:  [3291.  145.]
     Log prior probability for classes:  [-0.04311653 -3.16532954]
     Log conditional probability for each feature given a class:  [[-4.86773845e-03 -6.79502
       -1.10563808e+01 -1.14237629e+01 -1.79362509e+01 -1.06200350e+01
       -1.29804239e+01 -1.05742404e+01 -1.36665535e+01 -1.15561284e+01
       -1.09735341e+01 -1.25702749e+01 -1.57390263e+01 -1.10803154e+01
       -1.24452492e+01 -1.24852125e+01 -1.12629529e+01 -1.11845648e+01
       -1.17028211e+01 -1.23527546e+01 -1.15025073e+01 -1.24167920e+01]
      [-6.14395678e-03 -6.36670554e+00 -5.66931236e+00 -7.16182860e+00
       -1.11140899e+01 -1.14296068e+01 -1.55567412e+01 -1.09027808e+01
       -1.18190715e+01 -1.07776177e+01 -1.22245366e+01 -1.28486910e+01
       -1.06739392e+01 -1.24656987e+01 -1.55567412e+01 -1.10458817e+01
       -1.20013931e+01 -1.55567412e+01 -1.13670864e+01 -1.11622920e+01
       -1.25610089e+01 -1.17955410e+01 -1.15494080e+01 -1.21227540e+01]]
     /usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:103: FutureWarning:
       warnings.warn(msg, category=FutureWarning)
```

```
# Confusion matrix and Evaluation metrics
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred)
print("Confusion Matrix")
print(cf)
tn, fp, fn, tp = cf.ravel()
print("TP: ", tp, ", FP: ", fp, ", TN: ", tn, ", FN: ", fn)
```

```
Confusion Matrix
[[872 537]
 [ 21  43]]
TP:  43 , FP:  537 , TN:  872 , FN:  21
```

```
# Classifier report
from sklearn.metrics import classification_report
from sklearn import metrics

print(classification_report(y_test, y_pred))
```

```
                   precision    recall  f1-score   support

        No Stroke       0.98      0.62      0.76      1409
           Stroke       0.07      0.67      0.13        64

         accuracy                           0.62      1473
        macro avg       0.53      0.65      0.45      1473
     weighted avg       0.94      0.62      0.73      1473
```

```
# Decision tree on dummy encoded data
from sklearn import tree
clf = tree.DecisionTreeClassifier(max_depth = 5) # 5 levels set
clf = clf.fit(x_train, y_train)

import graphviz
# Obtain unique class values to show on tree
class_values = df_dummy[class_col_name]. unique()
print("class names: ", class_values)
dot_data = tree.export_graphviz(clf, out_file = None, feature_names = dummy_feature_names, cl
```

```
class names:  ['Stroke' 'No Stroke']
```

```
# Draw graph
graph = graphviz.Source(dot_data, format = "png")
# Graph won't display
```

```
# Perform prediction on test set
y_pred = clf.predict(x_test)
```

```
# Get decision tree report
from sklearn.metrics import classification_report
from sklearn import metrics
print(classification_report(y_test, y_pred))
```

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

```
              precision    recall  f1-score   support

    No Stroke       0.96      0.99      0.97      1409
       Stroke       0.12      0.03      0.05        64

     accuracy                           0.95      1473
    macro avg       0.54      0.51      0.51      1473
 weighted avg       0.92      0.95      0.93      1473
```