



```
# CST383 Project
**Team members:**
Justin Thomas
Jordan Bienz
Michael Janes

## Project Description
Stroke predication from classification dataset
```

CST383 Project

Team members:

Justin Thomas
Jordan Bienz
Michael Janes

Project Description

Stroke predication from classification dataset

Read the Data

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

from sklearn import svm
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures, MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from sklearn.metrics import roc_curve, RocCurveDisplay, confusion_matrix, plot_confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, precision_recall_curve, PrecisionRecallDisplay

url = "https://raw.githubusercontent.com/JustinThomasCSUMB/cst383project/main/healthcare-data"
df = pd.read_csv(url)
```

Initial Exploration

Saved successfully!



- There is 5,110 total initial entries.

How many NA values are in the data?

- There are 201 rows that have NA values, all of the NA values are in the BMI column. These rows make up ~4% of the total rows

Does the dataset contain much obviously bad data?

What are the types of the columns?

```
shape = df.shape
print("Data Shape:", shape)
print('\nInfo:', df.info())
print('\nDescribe:', df.describe())
```

```
Data Shape: (5110, 12)
<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   int64
1   gender                5110 non-null   object
2   age                   5110 non-null   float64
3   hypertension          5110 non-null   int64
4   heart_disease         5110 non-null   int64
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   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Info: None

```
Describe:
count    5110.000000    5110.000000    ...    4909.000000    5110.000000
mean     36517.829354    43.226614    ...     28.893237     0.048728
std      21161.721625    22.612647    ...     7.854067     0.215320
min        67.000000     0.080000    ...    10.300000     0.000000
25%     17741.250000    25.000000    ...    23.500000     0.000000
50%     36932.000000    45.000000    ...    28.100000     0.000000
75%     54682.000000    61.000000    ...    33.100000     0.000000
max      72940.000000    82.000000    ...    97.600000     1.000000
```

[8 rows x 7 columns]

```
print("\nTotal null values in column")
print(df.isnull().sum()) # % of null values
```

```
print("\nTotal null values")
print(df.isnull().sum()) #
```

```
Percent of values null per column
id          0.000000
gender      0.000000
age         0.000000
hypertension 0.000000
heart_disease 0.000000
ever_married 0.000000
work_type   0.000000
Residence_type 0.000000
avg_glucose_level 0.000000
bmi         3.933464
smoking_status 0.000000
stroke      0.000000
dtype: float64
```

```
Total null values
id          0
gender      0
age         0
hypertension 0
heart_disease 0
ever_married 0
work_type   0
Residence_type 0
avg_glucose_level 0
bmi        201
smoking_status 0
stroke      0
dtype: int64
```

```
print(df.head(10))
```

	id	gender	age	...	bmi	smoking_status	stroke
0	9046	Male	67.0	...	36.6	formerly smoked	1
1	51676	Female	61.0	...	NaN	never smoked	1
2	31112	Male	80.0	...	32.5	never smoked	1
3	60182	Female	49.0	...	34.4	smokes	1
4	1665	Female	79.0	...	24.0	never smoked	1
5	56669	Male	81.0	...	29.0	formerly smoked	1
6	53882	Male	74.0	...	27.4	never smoked	1
7	10434	Female	69.0	...	22.8	never smoked	1
8	27419	Female	59.0	...	NaN	Unknown	1
9	60491	Female	78.0	...	24.2	Unknown	1

```
[10 rows x 12 columns]
```

```
possible_columns = ['gender','age','hypertension', 'heart_disease', 'avg_glucose_level', 'bmi', 'smoking_status', 'stroke']
print("Useful Predictors:\n", possible_columns)
```

Saved successfully!

×

```
possible_columns = ['gender','age','hypertension', 'heart_disease', 'avg_glucose_level', 'bmi', 'smoking_status', 'stroke']
```

▼ Initial Preprocessing and Cleaning

remove columns with lots of missing data

remove columns that are useless

remove columns that are not relevant to what you want to do

remove other missing data

```
df = df.dropna() #drop all na/null values
df = df.drop(columns=['ever_married','work_type','Residence_type','id']) # remove useless col
print("Data with removed columns:\n", df.head(5))
```

Data with removed columns:

	gender	age	hypertension	...	bmi	smoking_status	stroke
0	Male	67.0	0	...	36.6	formerly smoked	1
2	Male	80.0	0	...	32.5	never smoked	1
3	Female	49.0	0	...	34.4	smokes	1
4	Female	79.0	1	...	24.0	never smoked	1
5	Male	81.0	0	...	29.0	formerly smoked	1

[5 rows x 8 columns]

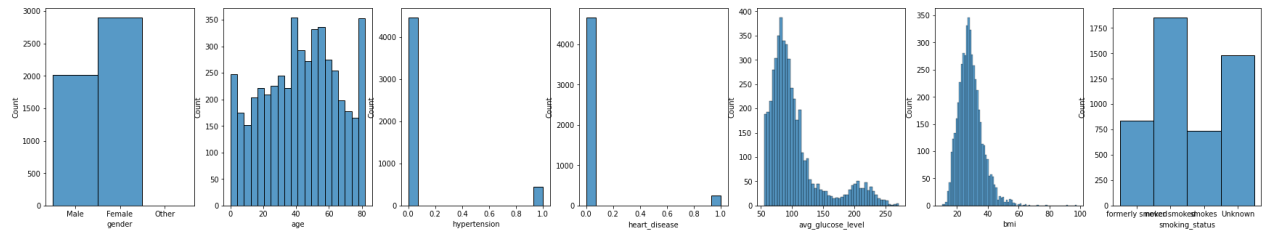
▼ Exploration and Visualization

- histograms of single numeric variables
- bar plots of value counts of single categorical variables
- grid of scatter plots (numeric variables)
- violin/bar plots for categorical/numeric variable pairs
- three-variable plots, such as scatterplots with color or shape of points as a
- third variable, or grouped bar plots
- plots of data over time (if applicable)

```
# histograms of single numeric variables
fig, ax = plt.subplots(1, len(possible_columns), figsize=[30,5])
for i,p in enumerate(possible_columns):
    sns.histplot(data=df, x=p, ax=ax[i])
```

Saved successfully!





bar plots of value counts of single categorical variables

```
plt.figure()
gender = df['gender'].value_counts()
sns.barplot(x=gender.index, y=gender)
plt.title("Gender of Participants")
plt.xlabel('Gender')
plt.ylabel('Total Participants')

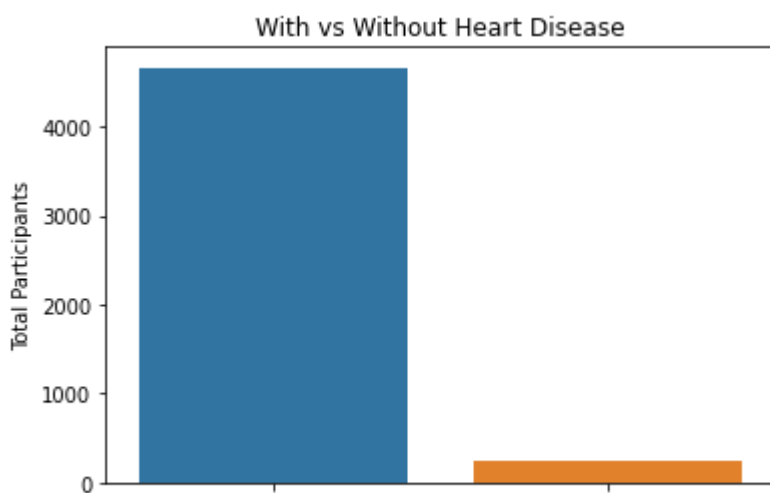
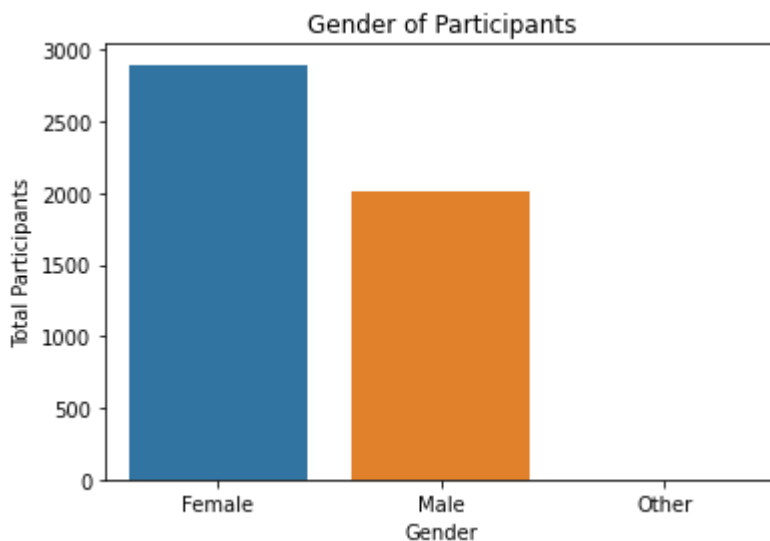
plt.figure()
heartDisease = df['heart_disease'].value_counts()
sns.barplot(x=heartDisease.index, y=heartDisease)
plt.title("With vs Without Heart Disease")
plt.xlabel('Has Heart Disease')
plt.ylabel('Total Participants')

plt.figure()
smoker = df['smoking_status'].value_counts()
sns.barplot(x=smoker.index, y=smoker)
plt.title("Smoking Status")
plt.xlabel('Status')
plt.ylabel('Total Participants')
```

Saved successfully!



```
Text(0, 0.5, 'Total Participants')
```



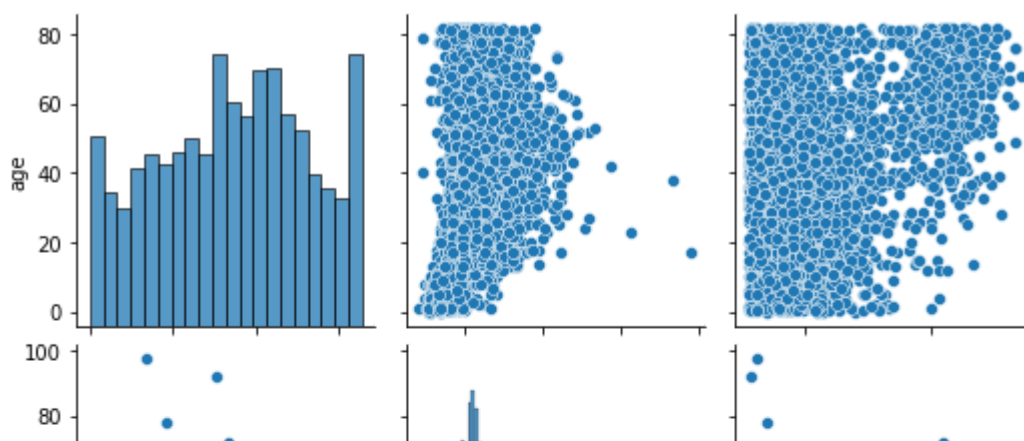
```
# scatter and histogram (pair plot) of numerical data  
plt.figure()  
scatNumDf = df[['age', 'bmi', 'avg_glucose_level']]  
sns.pairplot(scatNumDf)
```

Saved successfully!



```
<seaborn.axisgrid.PairGrid at 0x7fd023d35850>
```

```
<Figure size 432x288 with 0 Axes>
```



```
# Violin plot
plt.figure()
sns.violinplot(x='gender', y='age', data=df)
plt.title('Gender and Age Comparison')

plt.figure()
sns.violinplot(x='smoking_status', y='age', data=df)
plt.title('Smoking and Age Comparison')
```

Saved successfully!



```
Text(0.5, 1.0, 'Smoking and Age Comparison')
```

Gender and Age Comparison



```
# Three-variable scatter plot
```

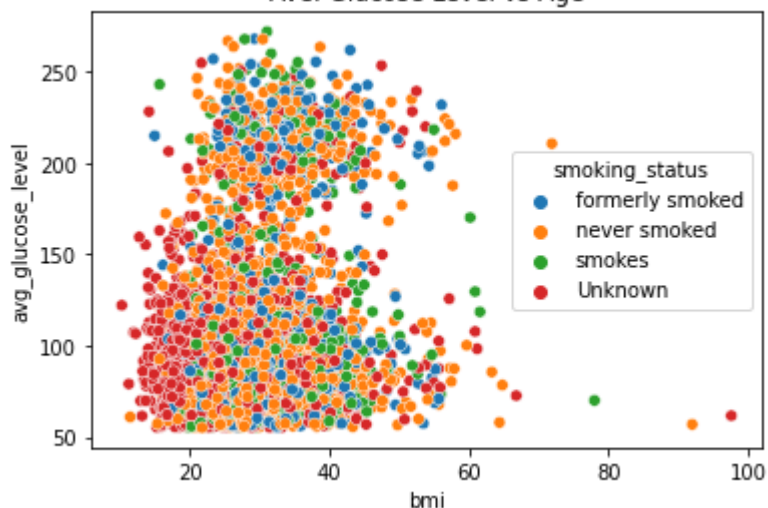
```
plt.figure()
```

```
sns.scatterplot(x='bmi', y='avg_glucose_level', hue='smoking_status', data=df)
```

```
plt.title('Ave. Glucose Level vs Age')
```

```
Text(0.5, 1.0, 'Ave. Glucose Level vs Age')
```

Ave. Glucose Level vs Age



▼ Final Preprocessing and Cleaning

- convert categorical to numeric data
- scale data

```
# clean up
```

```
# only compare male and female
```

```
df = df[df.gender != 'Other']
```

```
#replace smoking strings with numerical representation, replace gender with binary representa
```

```
df = df.replace({'never smoked':0,'formerly smoked':1,'smokes':2,'Unknown':3,'Male':1,'Female
```

```
dfdum = pd.get_dummies(df, drop_first=True)
```

```
#normalize data
```

```
scaler = MinMaxScaler()
```

```
ageScaled = scaler.fit_transform(df[['age']])
```

```
glucScaled = scaler.fit_transform(df[['avg_glucose_level']])
```

```
bmiScaled = scaler.fit_transform(df[['bmi']])
```

```
dfnorm = df.copy()
```

Saved successfully!

```
dfnorm['avg_glucose_level'] = glucscaled
```


Machine learning

- accuracy
- confusion matrix
- precision/recall
- ROC curve, precision/recall curve (if predictions are probabilities)
- MSE, RMSE
- R-squared statistic (usually computed on training data)
- predicted/actual scatterplot
- grid search to tune hyperparameters
- feature selection (such as forward feature selection)
- learning curve
- create training and test sets
- train model and make predictions
- assess results (classification case)
- assess result (regression case)
- tuning
- cross-validation can be used in both assessment and tuning
- assess bias/variance

```
# mse and rmse helper
def rmse(predicted, actual):
    return np.sqrt(((predicted - actual)**2).mean())
```

```
def mse(predicted, actual):
    return np.square(actual - predicted).mean()
```

```
# training data predictions
predictions = ['gender','age','hypertension', 'heart_disease', 'avg_glucose_level', 'bmi', 's
target = ['stroke']
xdum = dfdum[predictions].values
ydum = dfdum[target].values
```

```
target = ['stroke']
y = dfnorm[target].values
```

```
#prediction 1 all columns
pred1Cols = ['gender','age','hypertension', 'heart_disease', 'avg_glucose_level', 'bmi', 'smo
```

Saved successfully!

```
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y, test_size=0.30, random_state=0
```

```
clf = make_pipeline(StandardScaler(), LogisticRegression(random_state=0))
```

```

clf.fit(x1_test,y1_test)

# RocCurveDsisplay
plt.figure()
y1_score = clf.decision_function(x1_test)
fpr, tpr, _ = roc_curve(y1_test, y1_score, pos_label=clf.classes_[1])
rocauc1 = roc_auc_score(y1_test, y1_score)
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=rocauc1, estimator_name='AUC Score').
plt.title('ROC Curve Test Data - All Columns')

# precision recall display
plt.figure()
prec, recall, _ = precision_recall_curve(y1_test, y1_score, pos_label=clf.classes_[1])
aps1 = average_precision_score(y1_test, y1_score)
pr_display = PrecisionRecallDisplay(precision=prec, recall=recall, average_precision=aps1, es
plt.title('Precision Recall Test Data - All Columns')

# confusion matrix
plt.figure()
y1pred = clf.predict(x1_test)
cm = confusion_matrix(y1_test,y1pred)
cm_display=ConfusionMatrixDisplay(cm, display_labels=['']).plot()
plt.title('Conf. Matrix Test Data - All Columns')

# results
# accuracy
accuracy1 = accuracy_score(y1_test, y1pred)
mse1 = mse(y1pred, y1_test)
rmse1 = rmse(y1pred, y1_test)
r21 = r2_score(y1_test, y1pred)

print('Prediction 1 contains data from all columns in the dataset.')
print('Accuracy for Prediction 1: {:.2f}'.format(accuracy1))
print('MSE for Prediction 1: {:.2f}'.format(mse1))
print('RMSE for Prediction 1: {:.2f}'.format(rmse1))
print('R Squared for Prediction 1: {:.2f}'.format(r21))

# -----

#prediction 2 age, hypertension, heart_disease, and smoking_status
pred2Cols = ['age','hypertension','heart_disease','smoking_status']
x2 = dfnorm[pred2Cols].values

x2_train, x2_test, y2_train, y2_test = train_test_split(x2, y, test_size=0.30, random_state=0)

clf2 = make_pipeline(StandardScaler(), LogisticRegression(random_state=0))

plt.figure()
y2_score = clf2.decision_function(x2_test)
fpr2, tpr2, _ = roc_curve(y2_test, y2_score, pos_label=clf.classes_[1])

```

Saved successfully!



```

roc_auc2 = roc_auc_score(y2_test, y2_score)
roc_display = RocCurveDisplay(fpr=fpr2, tpr=tpr2, roc_auc=roc_auc2, estimator_name='AUC Score')
plt.title('ROC Curve Test Data - age, hypertension, heart_disease, smoking_status')

# precision recall display
plt.figure()
prec2, recall2, _ = precision_recall_curve(y2_test, y2_score, pos_label=clf.classes_[1])
aps2 = average_precision_score(y2_test, y2_score)
pr_display2 = PrecisionRecallDisplay(precision=prec2, recall=recall2, average_precision=aps2,
plt.title('Precision Recall Test Data - age, hypertension, heart_disease, smoking_status')

# confusion matrix
plt.figure()
y2pred = clf2.predict(x2_test)
cm2 = confusion_matrix(y2_test, y1pred)
cm_display2 = ConfusionMatrixDisplay(cm2, display_labels=['']).plot()
plt.title('Conf. Matrix Test Data - age, hypertension, heart_disease, smoking_status')

# results
# accuracy
accuracy2 = accuracy_score(y2_test, y2pred)
mse2 = mse(y2pred, y2_test)
rmse2 = rmse(y2pred, y2_test)
r22 = r2_score(y2_test, y2pred)

print('Prediction 2 contains data from age, hypertension, heart_disease, and smoking_status c
print('Accuracy for Prediction 2: {:.2f}'.format(accuracy2))
print('MSE for Prediction 2: {:.2f}'.format(mse2))
print('RMSE for Prediction 2: {:.2f}'.format(rmse2))
print('R Squared for Prediction 2: {:.2f}'.format(r22))

# -----

#prediction 3 bmi, avg_glucose_level, gender
pred3Cols = ['bmi', 'avg_glucose_level', 'gender']
x3 = dfnorm[pred3Cols].values

x3_train, x3_test, y3_train, y3_test = train_test_split(x3, y, test_size=0.30, random_state=0)

clf3 = make_pipeline(StandardScaler(), LogisticRegression(random_state=0))
clf3.fit(x3_test, y3_test)

# RocCurveDisplay
plt.figure()
y3_score = clf3.decision_function(x3_test)
fpr3, tpr3, _ = roc_curve(y3_test, y3_score, pos_label=clf.classes_[1])
roc_auc3 = roc_auc_score(y3_test, y3_score)
roc_display3 = RocCurveDisplay(fpr=fpr3, tpr=tpr3, roc_auc=roc_auc3, estimator_name='AUC Score')
plt.title('ROC Curve Test Data - bmi, avg_glucose_level, gender')

# precision recall display

```

Saved successfully!



```
plt.figure()
prec3, recall3, _ = precision_recall_curve(y3_test, y3_score, pos_label=clf.classes_[1])
aps3 = average_precision_score(y3_test, y3_score)
pr_display3 = PrecisionRecallDisplay(precision=prec3, recall=recall3, average_precision=aps3,
plt.title('Precision Recall Test Data - bmi, avg_glucose_level, gender')

# confusion matrix
plt.figure()
y3pred = clf3.predict(x3_test)
cm3 = confusion_matrix(y3_test,y3pred)
cm_display3 = ConfusionMatrixDisplay(cm3, display_labels=['']).plot()
plt.title('Conf. Matrix Test Data - bmi, avg_glucose_level, gender')

# results
# accuracy
accuracy3 = accuracy_score(y3_test, y3pred)
mse3 = mse(y3pred, y3_test)
rmse3 = rmse(y3pred, y3_test)
r23 = r2_score(y3_test, y3pred)

print('Prediction 3 contains data from bmi, avg_glucose_level, and gender columns in the data
print('Accuracy for Prediction 3: {:.2f}'.format(accuracy3))
print('MSE for Prediction 3: {:.2f}'.format(mse3))
print('RMSE for Prediction 3: {:.2f}'.format(rmse3))
print('R Squared for Prediction 3: {:.2f}'.format(r23))

# -----
```

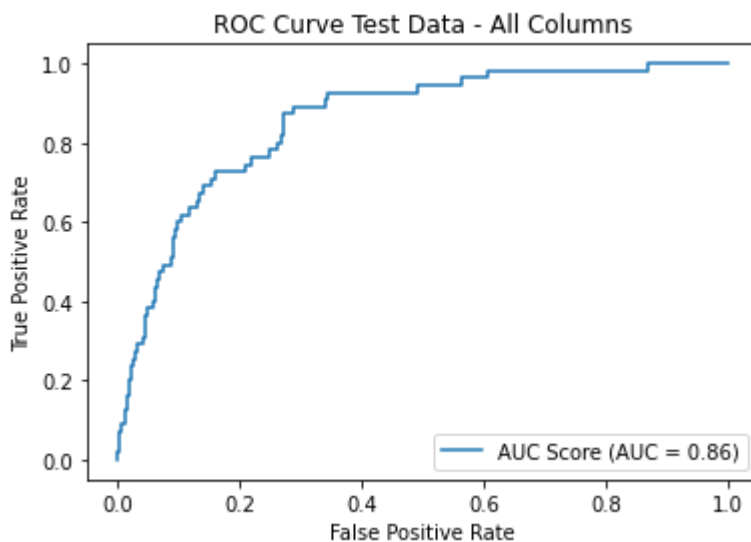
Saved successfully!



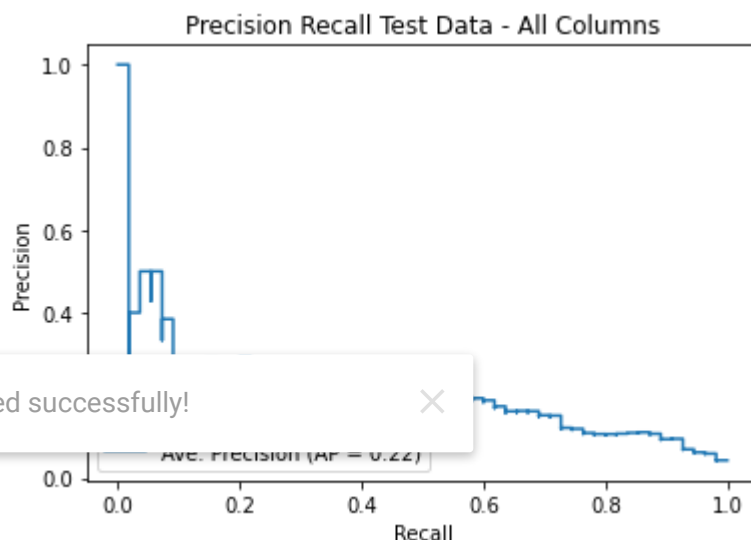
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConver
y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConver
y = column_or_1d(y, warn=True)
Prediction 1 contains data from all columns in the dataset.
Accuracy for Prediction 1: 0.96
MSE for Prediction 1: 0.04
RMSE for Prediction 1: 0.19
R Squared for Prediction 1: -0.02
Prediction 2 contains data from age, hypertension, heart_disease, and smoking_stat
Accuracy for Prediction 2: 0.96
MSE for Prediction 2: 0.04
RMSE for Prediction 2: 0.19
R Squared for Prediction 2: -0.04
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConver
y = column_or_1d(y, warn=True)
Prediction 3 contains data from bmi, avg_glucose_level, and gender columns in the
Accuracy for Prediction 3: 0.96
MSE for Prediction 3: 0.04
RMSE for Prediction 3: 0.19
R Squared for Prediction 3: -0.04
<Figure size 432x288 with 0 Axes>

```

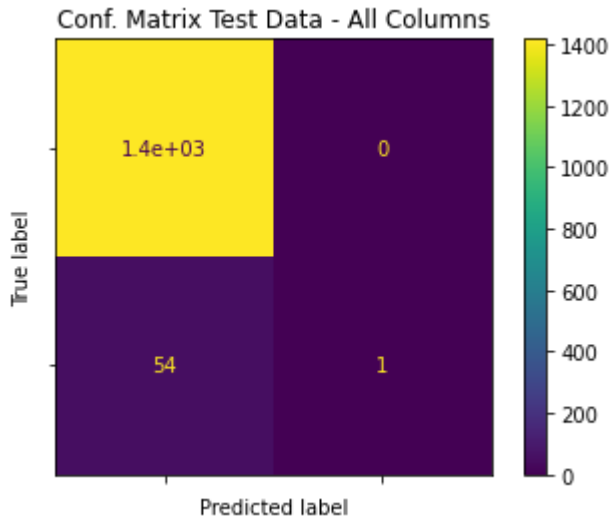


<Figure size 432x288 with 0 Axes>



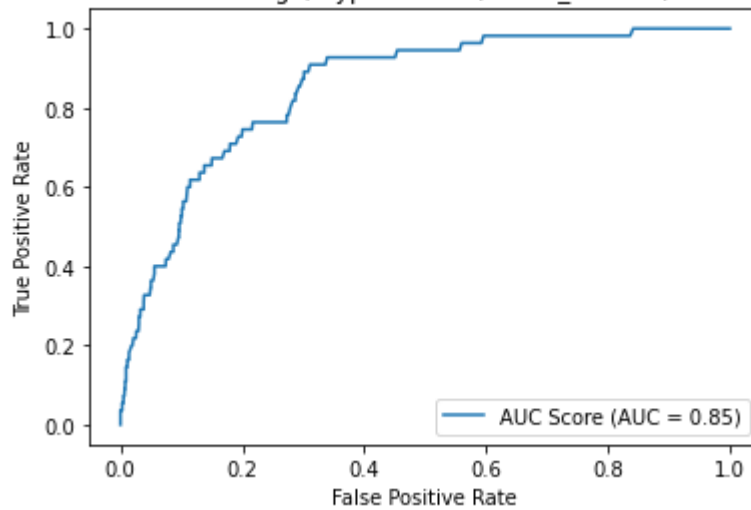
Saved successfully!

<Figure size 432x288 with 0 Axes>



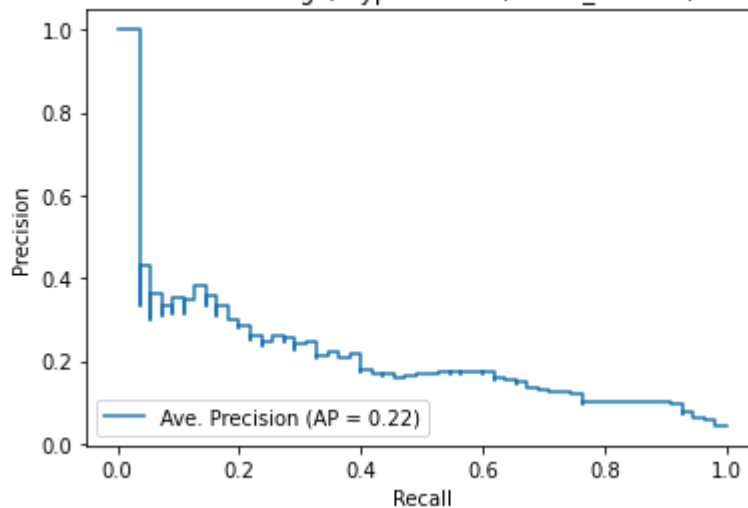
<Figure size 432x288 with 0 Axes>

ROC Curve Test Data - age, hypertension, heart_disease, smoking_status



<Figure size 432x288 with 0 Axes>

Precision Recall Test Data - age, hypertension, heart_disease, smoking_status

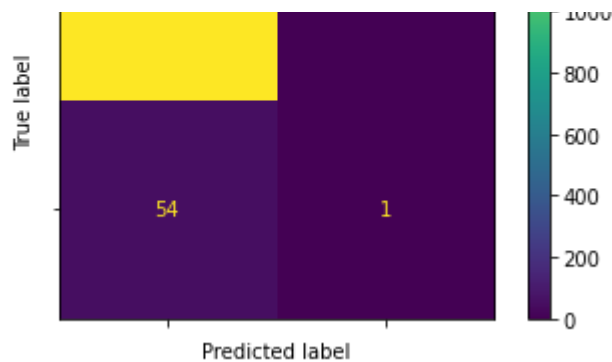


Saved successfully!

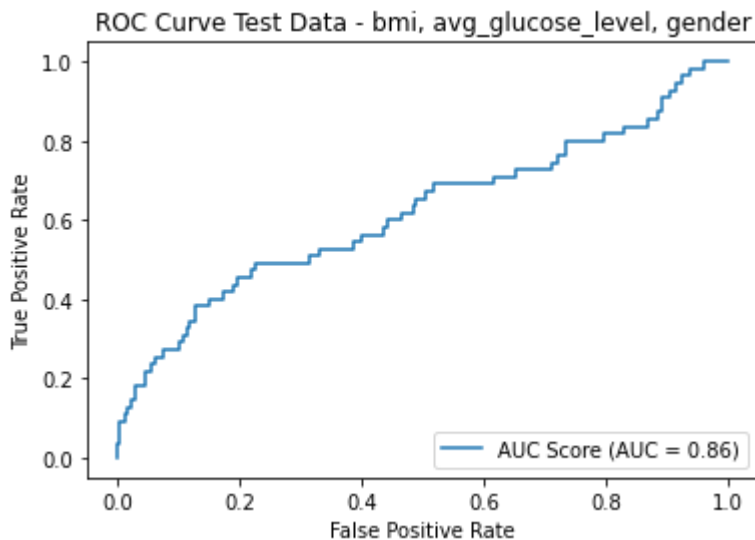
<Figure size 432x288 with 0 Axes>

Precision, heart_disease, smoking_status

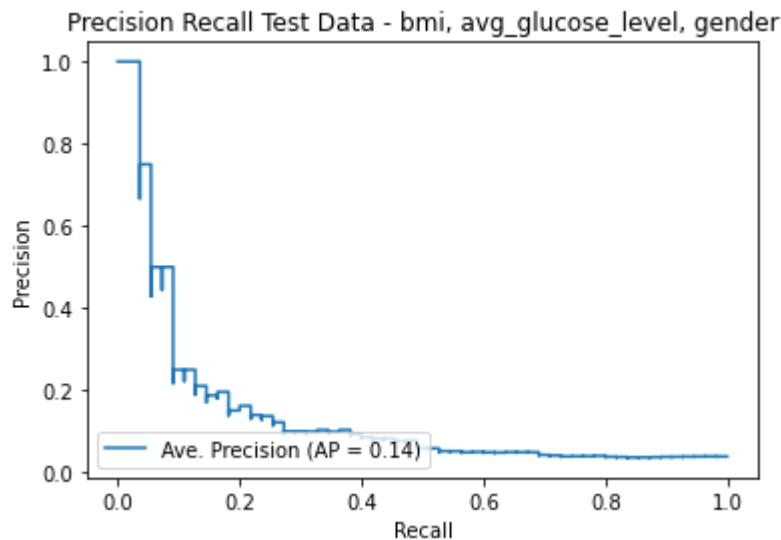




<Figure size 432x288 with 0 Axes>

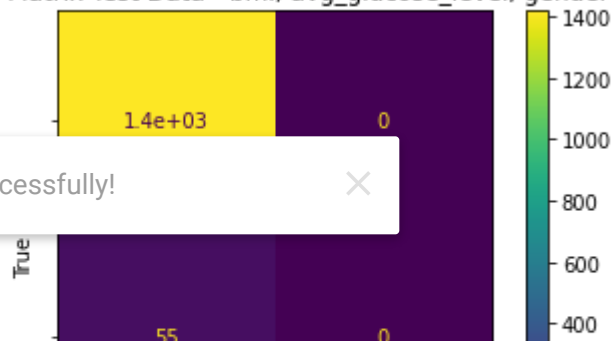


<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

Conf. Matrix Test Data - bmi, avg_glucose_level, gender



Saved successfully!



✓ 2s completed at 7:14 PM



Saved successfully!

