Data Mining

D209: Task 1

Shantel Johnson

April 6, 2023

Contents

[Section A 3](#_Toc131705927)

[Part 1: Research Question 3](#_Toc131705928)

[Part 2: Analytical Goal 3](#_Toc131705929)

[Section B 3](#_Toc131705930)

[Part 1: Classification Method 3](#_Toc131705931)

[Part 2: Assumption 4](#_Toc131705932)

[Part 3: Packages 4](#_Toc131705933)

[Section C 4](#_Toc131705934)

[Part 1: Preprocessing Goal 4](#_Toc131705935)

[Part 2: Variables 4](#_Toc131705936)

[Part 3: Steps 4](#_Toc131705937)

[Part 4: Preprocessing Output 5](#_Toc131705938)

[Section D 5](#_Toc131705939)

[Part 1: Data Partition Output 5](#_Toc131705940)

[Part 2: Analysis 5](#_Toc131705941)

[Part 3: Code 5](#_Toc131705942)

[Section E 5](#_Toc131705943)

[Part 1: Model Evaluation 5](#_Toc131705944)

[Part 2: Results 6](#_Toc131705945)

[Part 3: Limitation 6](#_Toc131705946)

[Part 4: Recommendation 6](#_Toc131705947)

[Section F 6](#_Toc131705948)

[Part 1: Data Preparation Code 6](#_Toc131705949)

[Data Cleaning 9](#_Toc131705950)

[Data Preparation 10](#_Toc131705951)

[Data Transformation 12](#_Toc131705952)

[Output 14](#_Toc131705953)

[Part 2: Classification Model Code 14](#_Toc131705954)

[Naive Bayes Classifier 14](#_Toc131705955)

[Section G 18](#_Toc131705956)

[Part 1: Demonstration 18](#_Toc131705957)

[Section H 18](#_Toc131705958)

[Part 1: Web Sources 18](#_Toc131705959)

[Part 2: References 18](#_Toc131705960)

# Section A

## Part 1: Research Question

The dataset consists of AnyTelecom’s observations on customer demographics, their telecom services, and whether they have churned (dropped AnyTelecom’s services within the past month). As the project’s data analyst, it is my job to answer the research question: Which customers are at risk of churn? To answer this question, I will use Naïve Bayes classification.

## Part 2: Analytical Goal

The goal of this project is to build a machine learning model that assigns a hypothetical customer to either “at risk” or “not at risk” of churn. The information for this hypothetical customer is included in the following table:

|  |  |
| --- | --- |
| **Variable** | **Value** |
| Area | Suburban |
| Children | 2 |
| Age | 42 |
| Income | 53,000 |
| Marital | Divorced |
| Gender | Female |
| Email | 3 |
| Contract | One year |
| Port\_modem | Yes |
| Tablet | Yes |
| InternetService | DSL |
| Phone | Yes |
| Multiple | Yes |
| OnlineSecurity | Yes |
| OnlineBackup | Yes |
| DeviceProtection | Yes |
| TechSupport | Yes |
| StreamingTV | Yes |
| StreamingMovies | Yes |
| PaperlessBilling | Yes |
| PaymentMethod | Credit Card (automatic) |
| Tenure | 7.5 |
| MonthlyCharge | 245 |
| Bandwidth\_GB\_Year | 1000 |

# Section B

## Part 1: Classification Method

Naïve Bayes classification uses the probability of observing predictor variables to estimate the probability of observing the outcome Y=i, given a set of predictor variables. The expected outcome is a binary response that assigns a record to the class with the highest probability for a given set of predictor values (Bruce, Bruce, & Gedeck, 2019).

## Part 2: Assumption

One assumption of Naïve Bayes classification is that the method only works with categorical variables. To use continuous variables with Naïve Bayes, they must first be converted to categorical variables (Bruce, Bruce, & Gedeck, 2019).

## Part 3: Packages

The following libraries will be used in support of the analysis:

|  |  |  |
| --- | --- | --- |
| **Library** | **Module** | **Purpose** |
| pandas | DataFrame | Data manipulation & analysis |
| pandas | cut | Feature transformation |
| scipy | stats | Calculate z-scores |
| sklearn | model\_selection | Partition data |
| sklearn | naïve\_bayes | Classification model |
| sklearn | metrics | Evaluate model |

# Section C

## Part 1: Preprocessing Goal

Recall in Section B, Part 2 that continuous variables must be converted to categorical variables for use in a Naïve Bayes classifier. One preprocessing goal is to convert continuous predictor variables into categorical variables using the technique of binning.

## Part 2: Variables

Data descriptions for all predictor variables used in the analysis, including whether a particular variable is considered continuous or categorical, can be viewed in Section F, Part 1.

## Part 3: Steps

The data was prepared according to the following steps:

1. Data quality issues were identified.
   1. Find duplicates using the pandas.DataFrame.duplicated() function.
   2. Find missing values using the pandas.DataFrame.isnull().sum() function.
   3. Find outliers by applying the scipy.stats.zscore() function on continuous variables in the dataset.
2. Data quality issues were remediated.
   1. Replace outliers using pandas.DataFrame.replace() function.
3. Continuous variables were binned and converted to categorical variables using the pandas.cut() function.
4. All predictor variables were encoded.
   1. Binary variables were encoded with ordinal encoding using the pandas.DataFrame.replace() function.
   2. All other variables were encoded with one-hot encoding using the pandas.DataFrame.get\_dummies() function.

To view the associated data preparation code, refer to Section F, Part 1.

## Part 4: Preprocessing Output

Refer to the attachment “churn prepped1.csv” to view the output of the data cleaning process.

# Section D

## Part 1: Data Partition Output

Refer to the attachments “train\_data1.csv” and “test\_data1.csv” to view the output of partitioning the data.

## Part 2: Analysis

As stated in Section A, Part 1, the dataset was analyzed using Naïve Bayes classification. The analysis was conducted according to the following steps:

1. The data was partitioned into train and test sets using the sklearn.model\_selction.train\_test\_split() function.
2. The Naïve Bayes classifier was fit with the predictor and target variables in the training set using the sklearn.naive\_bayes.MultinomialNB() function.
3. The model was evaluated.
   1. The model’s accuracy was calculated with the predictor and target variables in the test set using the sklearn.metrics.accuracy\_score() function.
   2. A receiver operating characteristic (ROC) curve was plotted using the sklearn.metrics.roc\_curve() function.
   3. The model’s area under the curve (AUC) value was calculated using the sklearn.metrics.auc() function.

## Part 3: Code

Refer to Section F, Part 2 to view the code associated with the analysis classification.

# Section E

## Part 1: Model Evaluation

The Naïve Bayes classification model was evaluated for accuracy and AUC. According to the output, the model’s accuracy was approximately 0.875, meaning that 87.5% of the predicted values matched the true values for Churn. Likewise, the AUC was approximately 0.926; this means that the model is effective at classifying 1s (Churn = 1, or Churn = Yes) correctly.

## Part 2: Results

Since the model assigns most customers to the correct class, we can reasonably assume that the model will assign our hypothetical customer to the correct class as well. Recall that the information for the hypothetical customer was defined in Section A, Part 2. After plugging this information into the model, the result is that the hypothetical customer is classified as being at risk of churn. Refer to Section F, Part 2 for more.

## Part 3: Limitation

As mentioned in Section C, Part 3, all continuous predictor variables were binned and converted to categorical variables. Aside from Children, four bins were used to convert each continuous variable. It is possible that increasing or decreasing the number of bins for each continuous variable could impact the accuracy of the Naïve Bayes classifier.

## Part 4: Recommendation

While the accuracy of the model is relatively high at 87.5%, there is still room for improvement. I recommend conducting feature selection on the full dataset to determine which predictor variables best improve the model’s accuracy and include them in the next iteration of the Naïve Bayes classifier.

# Section F

## Part 1: Data Preparation Code

In [1]:

# suppress warnings

import warnings

warnings.filterwarnings('ignore')

In [2]:

# read churn data into DataFrame using pandas

import pandas as pd

df = pd.read\_csv('source\_output/churn\_clean.csv', header='infer')

In [3]:

# select relevant variables

df = df[['Area',

'Children',

'Age',

'Income',

'Marital',

'Gender',

'Email',

'Contract',

'Port\_modem',

'Tablet',

'InternetService',

'Phone',

'Multiple',

'OnlineSecurity',

'OnlineBackup',

'DeviceProtection',

'TechSupport',

'StreamingTV',

'StreamingMovies',

'PaperlessBilling',

'PaymentMethod',

'Tenure',

'MonthlyCharge',

'Bandwidth\_GB\_Year',

'Churn']]

In [4]:

df.head()

Out[4]:

|  | **Area** | **Children** | **Age** | **Income** | **Marital** | **Gender** | **Email** | **Contract** | **Port\_modem** | **Tablet** | **...** | **DeviceProtection** | **TechSupport** | **StreamingTV** | **StreamingMovies** | **PaperlessBilling** | **PaymentMethod** | **Tenure** | **MonthlyCharge** | **Bandwidth\_GB\_Year** | **Churn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Urban | 0 | 68 | 28561.99 | Widowed | Male | 10 | One year | Yes | Yes | ... | No | No | No | Yes | Yes | Credit Card (automatic) | 6.795513 | 172.455519 | 904.536110 | No |
| **1** | Urban | 1 | 27 | 21704.77 | Married | Female | 12 | Month-to-month | No | Yes | ... | No | No | Yes | Yes | Yes | Bank Transfer(automatic) | 1.156681 | 242.632554 | 800.982766 | Yes |
| **2** | Urban | 4 | 50 | 9609.57 | Widowed | Female | 9 | Two Year | Yes | No | ... | No | No | No | Yes | Yes | Credit Card (automatic) | 15.754144 | 159.947583 | 2054.706961 | No |
| **3** | Suburban | 1 | 48 | 18925.23 | Married | Male | 15 | Two Year | No | No | ... | No | No | Yes | No | Yes | Mailed Check | 17.087227 | 119.956840 | 2164.579412 | No |
| **4** | Suburban | 0 | 83 | 40074.19 | Separated | Male | 16 | Month-to-month | Yes | No | ... | No | Yes | Yes | No | No | Mailed Check | 1.670972 | 149.948316 | 271.493436 | Yes |

5 rows × 25 columns

In [5]:

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Data columns (total 25 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Area 10000 non-null object

1 Children 10000 non-null int64

2 Age 10000 non-null int64

3 Income 10000 non-null float64

4 Marital 10000 non-null object

5 Gender 10000 non-null object

6 Email 10000 non-null int64

7 Contract 10000 non-null object

8 Port\_modem 10000 non-null object

9 Tablet 10000 non-null object

10 InternetService 10000 non-null object

11 Phone 10000 non-null object

12 Multiple 10000 non-null object

13 OnlineSecurity 10000 non-null object

14 OnlineBackup 10000 non-null object

15 DeviceProtection 10000 non-null object

16 TechSupport 10000 non-null object

17 StreamingTV 10000 non-null object

18 StreamingMovies 10000 non-null object

19 PaperlessBilling 10000 non-null object

20 PaymentMethod 10000 non-null object

21 Tenure 10000 non-null float64

22 MonthlyCharge 10000 non-null float64

23 Bandwidth\_GB\_Year 10000 non-null float64

24 Churn 10000 non-null object

dtypes: float64(4), int64(3), object(18)

memory usage: 1.9+ MB

### Data Cleaning[¶](" \l "Data-Cleaning)

#### Duplicates[¶](#Duplicates)

In [6]:

# check for duplication

df[df.duplicated()]

Out[6]:

|  | **Area** | **Children** | **Age** | **Income** | **Marital** | **Gender** | **Email** | **Contract** | **Port\_modem** | **Tablet** | **...** | **DeviceProtection** | **TechSupport** | **StreamingTV** | **StreamingMovies** | **PaperlessBilling** | **PaymentMethod** | **Tenure** | **MonthlyCharge** | **Bandwidth\_GB\_Year** | **Churn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

0 rows × 25 columns

#### Missing Values[¶](#Missing-Values)

In [7]:

# check for missing values

df.isnull().sum()

Out[7]:

Area 0

Children 0

Age 0

Income 0

Marital 0

Gender 0

Email 0

Contract 0

Port\_modem 0

Tablet 0

InternetService 0

Phone 0

Multiple 0

OnlineSecurity 0

OnlineBackup 0

DeviceProtection 0

TechSupport 0

StreamingTV 0

StreamingMovies 0

PaperlessBilling 0

PaymentMethod 0

Tenure 0

MonthlyCharge 0

Bandwidth\_GB\_Year 0

Churn 0

dtype: int64

#### Outliers[¶](#Outliers)

In [8]:

# check for outliers

# import scipy.stats to calculate z-scores

from scipy import stats

outliers = df.select\_dtypes(include='number')

outliers = stats.zscore(outliers)

outliers[outliers.abs() >= 3].count()

Out[8]:

Children 191

Age 0

Income 145

Email 12

Tenure 0

MonthlyCharge 0

Bandwidth\_GB\_Year 0

dtype: int64

In [9]:

# children, income, and email features contain outliers

Children\_outliers = outliers[outliers.Children.abs() >= 3].index

Income\_outliers = outliers[outliers.Income.abs() >= 3].index

Email\_outliers = outliers[outliers.Email.abs() >= 3].index

In [10]:

# replace outlier values with median values

df.Children.iloc[Children\_outliers] = df.Children.median()

df.Income.iloc[Income\_outliers] = df.Income.median()

df.Email.iloc[Email\_outliers] = df.Email.median()

### Data Preparation[¶](" \l "Data-Preparation)

#### Data Description: Categorical Variables[¶](#Data-Description:-Categorical-Variables)

In [11]:

df.select\_dtypes(exclude='number').describe()

Out[11]:

|  | **Area** | **Marital** | **Gender** | **Contract** | **Port\_modem** | **Tablet** | **InternetService** | **Phone** | **Multiple** | **OnlineSecurity** | **OnlineBackup** | **DeviceProtection** | **TechSupport** | **StreamingTV** | **StreamingMovies** | **PaperlessBilling** | **PaymentMethod** | **Churn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 |
| **unique** | 3 | 5 | 3 | 3 | 2 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 4 | 2 |
| **top** | Suburban | Divorced | Female | Month-to-month | No | No | Fiber Optic | Yes | No | No | No | No | No | No | No | Yes | Electronic Check | No |
| **freq** | 3346 | 2092 | 5025 | 5456 | 5166 | 7009 | 4408 | 9067 | 5392 | 6424 | 5494 | 5614 | 6250 | 5071 | 5110 | 5882 | 3398 | 7350 |

#### Data Description: Continuous Variables[¶](#Data-Description:-Continuous-Variables)

In [12]:

df.select\_dtypes(include='number').describe()

Out[12]:

|  | **Children** | **Age** | **Income** | **Email** | **Tenure** | **MonthlyCharge** | **Bandwidth\_GB\_Year** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 |
| **mean** | 1.925000 | 53.078400 | 38151.157582 | 12.022200 | 34.526188 | 172.624816 | 3392.341550 |
| **std** | 1.883383 | 20.698882 | 24850.801431 | 3.004565 | 26.443063 | 42.943094 | 2185.294852 |
| **min** | 0.000000 | 18.000000 | 348.670000 | 3.000000 | 1.000259 | 79.978860 | 155.506715 |
| **25%** | 0.000000 | 35.000000 | 19224.717500 | 10.000000 | 7.917694 | 139.979239 | 1236.470827 |
| **50%** | 1.000000 | 53.000000 | 33169.742500 | 12.000000 | 35.430507 | 167.484700 | 3279.536903 |
| **75%** | 3.000000 | 71.000000 | 51669.637500 | 14.000000 | 61.479795 | 200.734725 | 5586.141370 |
| **max** | 8.000000 | 89.000000 | 124025.100000 | 21.000000 | 71.999280 | 290.160419 | 7158.981530 |

### Data Transformation[¶](" \l "Data-Transformation)

#### Data Transformation: Numerical Variables[¶](#Data-Transformation:-Numerical-Variable)

In [13]:

# transform numerical variables to categorical variables

df.Children = pd.cut(df['Children'].array,bins=[0,1,3,8],labels=['children\_1', 'children\_2', 'children\_3'])

df.Age = pd.cut(df['Age'].array,bins=[18,35,53,71,89],labels=['age\_1', 'age\_2', 'age\_3', 'age\_4'])

df.Income = pd.cut(df['Income'].array,bins=[340,19200,33100,51700,124000],labels=['income\_1', 'income\_2', 'income\_3', 'income\_4'])

df.Email = pd.cut(df['Email'].array,bins=[3,10,12,14,21],labels=['email\_1', 'email\_2', 'email\_3', 'email\_4'])

df.Tenure = pd.cut(df['Tenure'].array,bins=[1,8,35,61,71],labels=['tenure\_1', 'tenure\_2', 'tenure\_3', 'tenure\_4'])

df.MonthlyCharge = pd.cut(df['MonthlyCharge'].array,bins=[79,139,167,200,290],labels=['charge\_1', 'charge\_2', 'charge\_3', 'charge\_4'])

df.Bandwidth\_GB\_Year = pd.cut(df['Bandwidth\_GB\_Year'].array,bins=[150,1230,3280,5590,7160],labels=['bw\_1', 'bw\_2', 'bw\_3', 'bw\_4'])

#### Data Transformation: Categorical Variables[¶](#Data-Transformation:-Categorical-Variab)

In [14]:

# perform nominal encoding

df = pd.get\_dummies(df, columns=['Area',

'Marital',

'Gender',

'InternetService',

'PaymentMethod',

'Bandwidth\_GB\_Year',

'MonthlyCharge',

'Tenure',

'Email',

'Income',

'Age',

'Children',

'Contract'])

In [15]:

# perform ordinal encoding

df.replace('No', 0, inplace=True)

df.replace('Yes', 1, inplace=True)

In [16]:

df.head(5)

Out[16]:

|  | **Port\_modem** | **Tablet** | **Phone** | **Multiple** | **OnlineSecurity** | **OnlineBackup** | **DeviceProtection** | **TechSupport** | **StreamingTV** | **StreamingMovies** | **...** | **Age\_age\_1** | **Age\_age\_2** | **Age\_age\_3** | **Age\_age\_4** | **Children\_children\_1** | **Children\_children\_2** | **Children\_children\_3** | **Contract\_Month-to-month** | **Contract\_One year** | **Contract\_Two Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **1** | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | ... | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **2** | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **3** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| **4** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

5 rows × 60 columns

### Output[¶](" \l "Output)

In [17]:

df.to\_csv("source\_output/churn\_prepped1.csv")

In [ ]:

## Part 2: Classification Model Code

### Naive Bayes Classifier[¶](" \l "Naive-Bayes-Classifier)

In [1]:

# read prepped data using pandas

import pandas as pd

df = pd.read\_csv("source\_output/churn\_prepped1.csv", header='infer', index\_col=0)

df.head(5)

Out[1]:

|  | **Port\_modem** | **Tablet** | **Phone** | **Multiple** | **OnlineSecurity** | **OnlineBackup** | **DeviceProtection** | **TechSupport** | **StreamingTV** | **StreamingMovies** | **...** | **Age\_age\_1** | **Age\_age\_2** | **Age\_age\_3** | **Age\_age\_4** | **Children\_children\_1** | **Children\_children\_2** | **Children\_children\_3** | **Contract\_Month-to-month** | **Contract\_One year** | **Contract\_Two Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **1** | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | ... | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **2** | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **3** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| **4** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

5 rows × 60 columns

In [2]:

# partition data

from sklearn.model\_selection import train\_test\_split

X = df.drop(columns='Churn')

y = df['Churn']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

#### Output[¶](#Output)

In [3]:

X\_train.to\_csv("source\_output/train\_data1.csv")

X\_test.to\_csv("source\_output/test\_data1.csv")

#### Model[¶](#Model)

In [4]:

# assign variables and fit model

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB(alpha=0.01, fit\_prior=True)

clf.fit(X\_train, y\_train)

Out[4]:

MultinomialNB(alpha=0.01)

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.   
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

MultinomialNB

MultinomialNB(alpha=0.01)

#### Evaluation[¶](#Evaluation)

##### Accuracy[¶](#Accuracy)

In [5]:

# get accuracy score

from sklearn.metrics import (accuracy\_score, roc\_curve, auc)

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_pred, y\_test)

accuracy

Out[5]:

0.875

##### ROC Curve[¶](#ROC-Curve)

In [6]:

# plot ROC curve

fpr, tpr, thresholds = roc\_curve(y, clf.predict\_proba(X)[:,1], pos\_label=1)

roc\_df = pd.DataFrame({'recall':tpr, 'specificity':1-fpr})

ax = roc\_df.plot(x='specificity', y='recall', figsize=(4,4), legend=False)

ax.set\_ylim(0, 1)

ax.set\_xlim(1, 0)

ax.plot((1, 0), (0, 1))

ax.set\_xlabel('specificity')

ax.set\_ylabel('recall')

Out[6]:

Text(0, 0.5, 'recall')

Chart, line chart

Description automatically generated

##### AUC[¶](#AUC)

In [7]:

# calculate auc

auc(fpr, tpr)

Out[7]:

0.9256672570915159

#### Hypothetical Customer[¶](#Hypothetical-Customer)

In [16]:

# read hypothetical customer data into DataFrame

df = pd.read\_csv('source\_output/hypothetical\_customer.csv', header='infer')

In [17]:

# predict classification for hypothetical customer

clf.predict(df)

Out[17]:

array([1], dtype=int64)

In [ ]:

# Section G

## Part 1: Demonstration

To view a walkthrough demonstration of the code and programming environment, view the following Panopto link: <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=b12decfa-9488-4fe4-9830-afdd00330c52>

# Section H

## Part 1: Web Sources

Gommers, et al. (2023). doi:10.5281/zenodo.7655153

Grisel, et al. (2022). doi:10.5281/zenodo.6543413

The pandas development team. (2023). doi:10.5281/zenodo.7741580

## Part 2: References

Bruce, P., Bruce, A., & Gedeck, P. (2019). *Practical Statistics for Data Scientists : 50+ Essential Concepts Using R and Python* (2 ed.). O'Reilly Media, Incorporated. Retrieved April 2023