



AP EDA: Framingham

📄 Course #	5100 - Intro AI
📁 Task	Class Lecture
➤ Parent Class	🧠 *Intro to AI Syllabus, 🧠 🔥🔥🔥 AI FINAL PROJECT 🔥🔥🔥
➤ Individual Class	🧠 AP XGBoost, 🧠 Random Forest Exploratory Data Analysis (EDA)
🔗 URL	https://rajagopalvenkat.com/teaching/resources/AI/
🔦 Progress	In progress
➤ *Algorithms, Formulas & Equations DB*	*Artificial Intelligence, Artificial Intelligence Tutorial AI Tutorial - GeeksforGeeks
📅 Week	10
☰ Office Hours	<p>Khoury Office Hours App for all office hours (online and in-person) for efficient queue management this semester.</p> <p>Instructor - Raj</p> <ul style="list-style-type: none">• In-person, Meserve 303, Tuesday, 9:30 am - 11:30 am.• To schedule appointments outside of office hours, visit my appointments page.• If you decide to swing by on a whim and my office door is open, feel free to bug me
☑ Reviewed?	<input type="checkbox"/>
📎 Textbook File	https://artint.info/3e/html/ArtInt3e.html

Masters CS Northeastern DB*

📁 Task	📅 Week	🍏 Class	🔦 Progress	📅 Date	📅 Due Date	📄 Rea
Syllabus	1	🧠 Anaconda Setup	Submitted!			
Homework	1	🧠 Problem Set 0: Background	Submitted!		@January 17, 2025	

"Framingham" heart disease dataset includes over 4,240 records, 16 columns and 15 attributes. The goal of the dataset is to predict whether the patient has 10-year risk of future (CHD) coronary heart disease

Heart Failure Attributes

1. Age: age of the patient [years]
2. Sex: sex of the patient [M: Male, F: Female]
3. ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4. RestingBP: resting blood pressure [mm Hg]
5. Cholesterol: serum cholesterol [mm/dl]
6. FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
7. RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
8. MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
9. ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
10. Oldpeak: oldpeak = ST [Numeric value measured in depression]
11. ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
12. HeartDisease: output class [1: heart disease, 0: Normal]

sex	
age	
education	
currentSmoker	
cigsPerDay	
BPMeds	
prevalentStroke	
prevalentHyp	
diabetes	
totChol	
sysBP	
diaBP	
BMI	
heartRate	
glucose	
TenYearCHD	

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

Framingham *number of missing values

Table 2. Criteria for heart failure.

MAJOR

1. Paroxysmal nocturnal dyspnea
2. Neck-vein distension (not counting supine position)
3. Rales in presence of unexplained dyspnea
4. Cardiomegaly and pulmonary hilar congestion (by X-ray in absence of left to right shunt), or increasing heart size.
5. Acute pulmonary edema described in hospital records
6. Ventricular gallop
7. Increased venous pressure (>16 cm H₂O from right atrium)
8. Circulation time (>24 seconds, arm to tongue)
9. Hepato-jugular reflux
10. Autopsy shows pulmonary edema, visceral congestion, cardiomegaly

MINOR

1. Ankle edema
2. Night cough
3. Dyspnea on ordinary exertion
4. Hepatomegaly
5. Pleural effusion
6. Decreased vital capacity by one-third from maximum records
7. Tachycardia (≥ 120 beats per minute)

[Open in a new tab](#)

Minor or Major: Weight loss (≥ 4.5 Kg) in 5 days, in response to HF therapy.

▼ XGBoost v1 Results @March 11, 2025 11:55 AM

```
# 4. Train default XGBoost classifier model
```

```
#Set up basic parameters
```

```
params = {  
    "objective": "binary:logistic", # Binary classification  
    "eval_metric": "auc", # AUC is good for classification  
    "seed": 42  
}
```

```
# Train the model
```

```
model = xgb.train(params, dtrain, num_boost_round=100)
```

```
# Save model
```

```
model.save_model("xgboost_model.json")
```

```
----
```

```
# Define XGBoost classifier
```

```
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric="auc")
```

```
# Define hyperparameter grid
```

```
param_grid = {  
    "n_estimators": [30, 40, 50, 75, 100, 150, 200, 300], # Number of trees  
    "max_depth": [3, 5, 7, 8, 9, 10, 11, 12], # Tree depth  
    "learning_rate": [0.01, 0.1, 0.2], # Step size shrinkage  
    "subsample": [0.8, 1.0], # Percentage of samples used per tree  
    "colsample_bytree": [0.8, 1.0] # Percentage of features used per tree  
}
```

```
# Run GridSearchCV
```

```
grid_search = GridSearchCV(xgb_clf, param_grid, cv=3, scoring="roc_auc", verbose=2, n_jobs=-1)  
grid_search.fit(X_train, y_train)
```

```
# Best parameters
```

```
print("Best parameters:", grid_search.best_params_)
```

```
# Save best model
```

```
best_model = grid_search.best_estimator_  
best_model.save_model("best_xgboost_model.json")
```

```
#Fitting 3 folds for each of 768 candidates, totalling 2304 fits
```

```
Best parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300, 'subsample': 0.8}
```

```
Accuracy: 0.8573113207547169
```

```
AUC Score: 0.7036613400616765
```

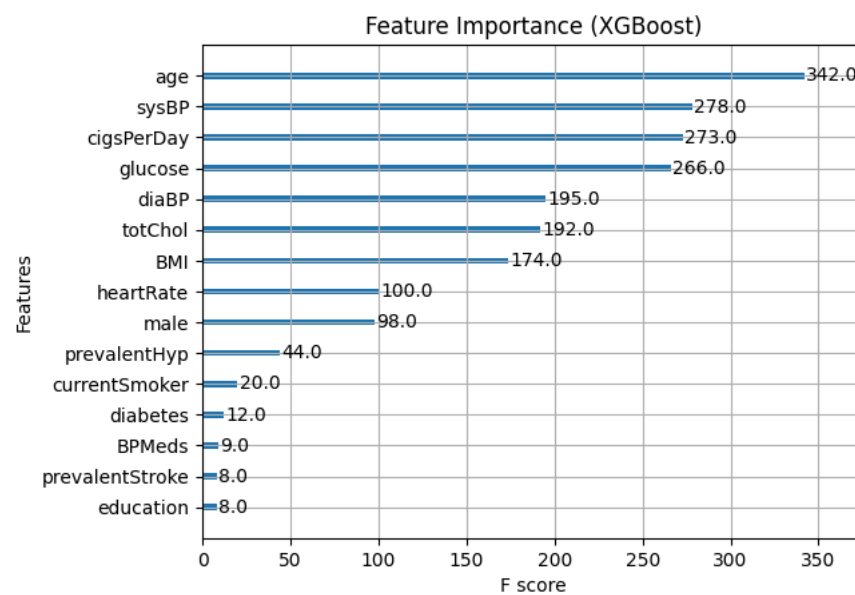
```
Confusion Matrix:
```

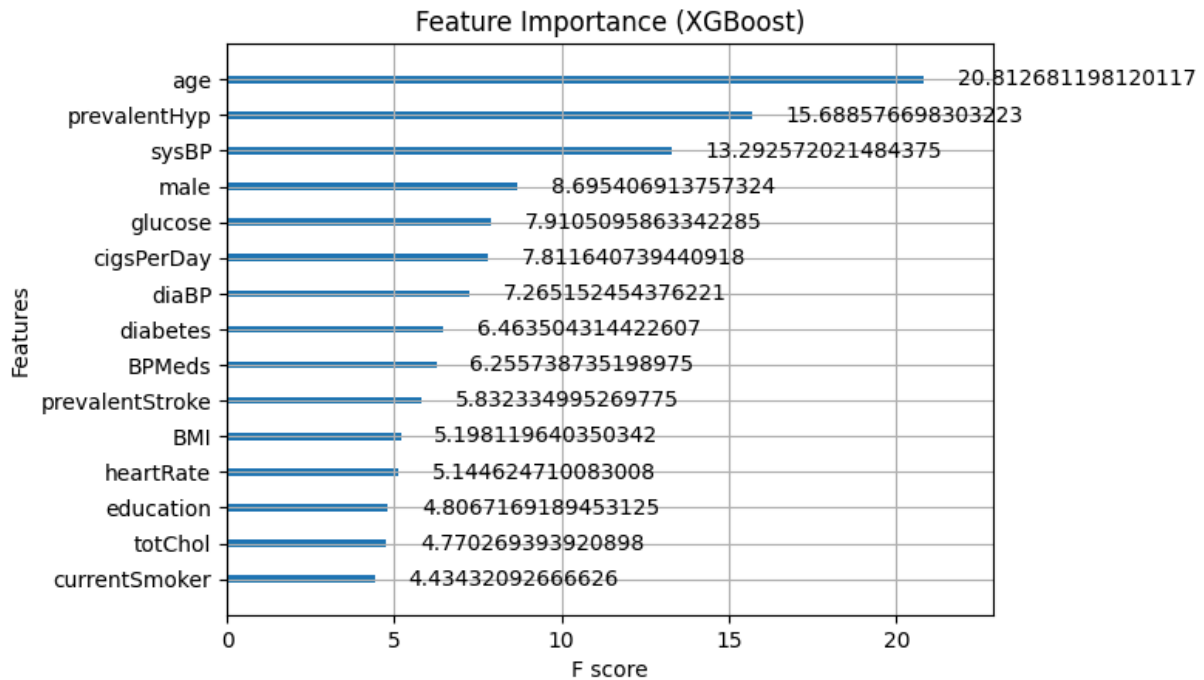
```
[[725 0]
 [121 2]]
```

Plot

```
# Plot feature importance using built-in XGBoost method
xgb.plot_importance(best_model, importance_type="weight") # "gain", "cover", or "weight"
plt.title("Feature Importance (XGBoost)")
plt.show()
```

- **"weight"** : Number of times a feature appears in trees.
- **"gain"** : Improvement in performance when a feature is used.
- **"cover"** : Coverage of samples a feature influences.





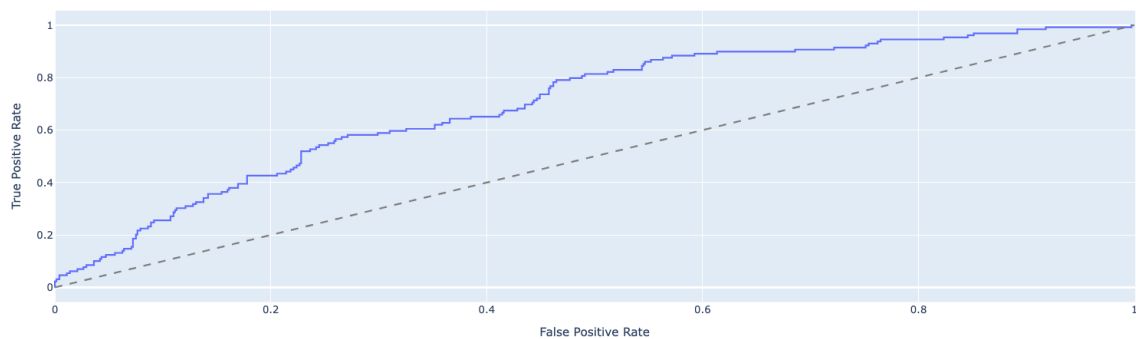
▼ Logistic Regression Results @March 10, 2025

Logistic Regression Classification Report:

	precision	recall	f1-score	support
0	0.85	0.99	0.92	719
1	0.47	0.05	0.10	129
accuracy			0.85	848
macro avg	0.66	0.52	0.51	848
weighted avg	0.79	0.85	0.79	848

Logistic Regression ROC-AUC: 0.6995935353796724

Logistic Regression ROC Curve (AUC = 0.6996)



▼ Decision Tree Results @March 10, 2025

(3)

```

Decision Tree Classification Report:
              precision    recall  f1-score   support

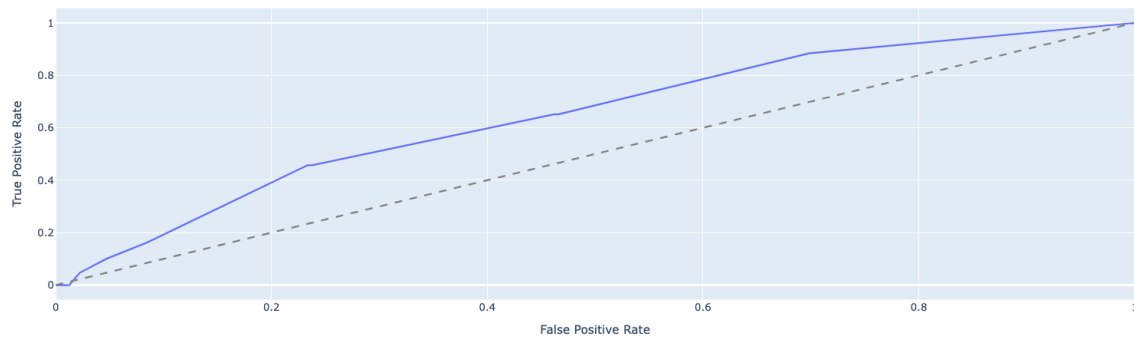
     0       0.85         0.98         0.91         719
     1       0.27         0.05         0.08         129

 accuracy          0.56          0.51          0.49         848
 macro avg          0.56          0.51          0.49         848
 weighted avg       0.76          0.84          0.78         848

Decision Tree ROC-AUC: 0.6444135373203523

```

Decision Tree ROC Curve (AUC = 0.6444)



▼ Framingham Dataset

Info	Result
Total rows	4240
df.shape	(4240, 16)

▼ Heart Failure Attributes

1. Age: age of the patient [years]
2. Sex: sex of the patient [M: Male, F: Female]
3. ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4. RestingBP: resting blood pressure [mm Hg]
5. Cholesterol: serum cholesterol [mm/dl]
6. FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
7. RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
8. MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
9. ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
10. Oldpeak: oldpeak = ST [Numeric value measured in depression]
11. ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
12. HeartDisease: output class [1: heart disease, 0: Normal]

▼ *After converting all features to numeric

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

Index:	4240	entries	
Data	columns	(total	
#	Column	Non-Null	
---	-----	-----	
0	male	4240	Int64
1	age	4240	Int64
2	education	4240	Int64
3	currentSmoker	4240	Int64
4	cigsPerDay	4240	Int64
5	BPMeds	4240	Int64
6	prevalentStroke	4240	Int64
7	prevalentHyp	4240	Int64
8	diabetes	4240	Int64
9	totChol	4240	Int64
10	sysBP	4240	Float64
11	diaBP	4240	Float64
12	BMI	4240	Float64
13	heartRate	4240	Int64
14	glucose	4240	Int64
15	TenYearCHD	4240	Int64
dtypes:	Float64(3)		
memory	usage:	629.4	

df.info()	#	Column	Non-Null	Count	Dtype
	0	male	4240	non-null	int64
	1	age	4240	non-null	int64
	2	education	4135	non-null	float64
	3	currentSmoker	4240	non-null	int64
	4	cigsPerDay	4211	non-null	float64
	5	BPMeds	4187	non-null	float64
	6	prevalentStroke	4240	non-null	int64
	7	prevalentHyp	4240	non-null	int64
	8	diabetes	4240	non-null	int64
	9	totChol	4190	non-null	float64
	10	sysBP	4240	non-null	float64
	11	diaBP	4240	non-null	float64
	12	BMI	4221	non-null	float64
	13	heartRate	4239	non-null	float64
	14	glucose	3852	non-null	float64
	15	TenYearCHD	4240	non-null	int64
dtypes:	float64(9),	int64(7)			
memory	usage:	530.1	KB		


```

root
|-- male: string (nullable = true)
|-- age: string (nullable = true)
|-- education: string (nullable = true)
|-- currentSmoker: string (nullable = true)
|-- cigsPerDay: string (nullable = true)
|-- BPMeds: string (nullable = true)
|-- prevalentStroke: string (nullable = true)
|-- prevalentHyp: string (nullable = true)
|-- diabetes: string (nullable = true)
|-- totChol: string (nullable = true)
|-- sysBP: string (nullable = true)
|-- diaBP: string (nullable = true)
|-- BMI: string (nullable = true)
|-- heartRate: string (nullable = true)
|-- glucose: string (nullable = true)
|-- TenYearCHD: string (nullable = true)

```

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('ggplot')
pd.set_option('max_columns', 200)

```

```
[5] 1 df.info()
```

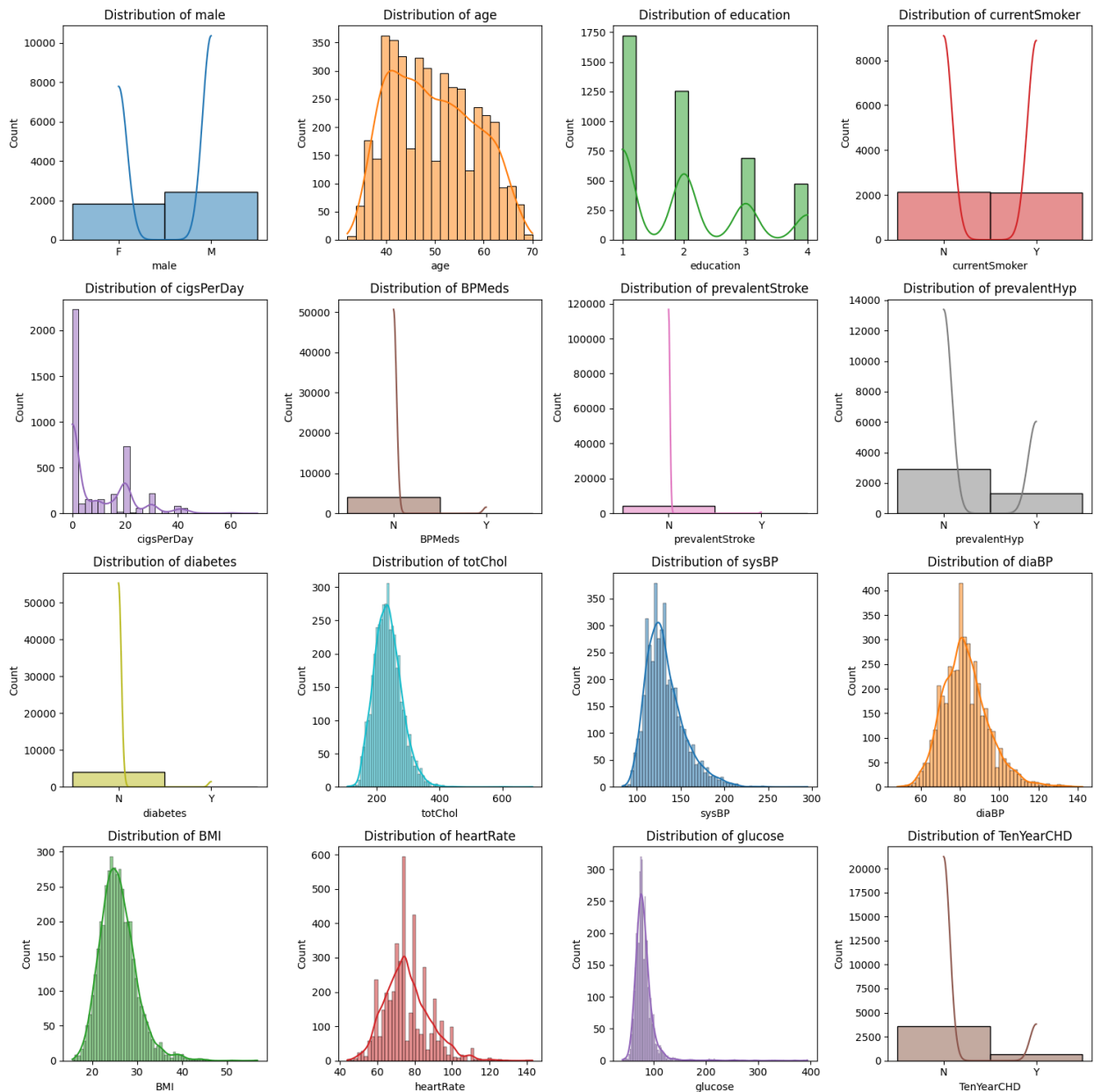
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4240 entries, 0 to 4239
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   male                  4240 non-null  int64
1   age                   4240 non-null  int64
2   education             4135 non-null  float64
3   currentSmoker         4240 non-null  int64
4   cigsPerDay            4211 non-null  float64
5   BPMeds                4187 non-null  float64
6   prevalentStroke       4240 non-null  int64
7   prevalentHyp          4240 non-null  int64
8   diabetes              4240 non-null  int64
9   totChol               4190 non-null  float64
10  sysBP                 4240 non-null  float64
11  diaBP                 4240 non-null  float64
12  BMI                   4221 non-null  float64
13  heartRate             4239 non-null  float64
14  glucose               3852 non-null  float64
15  TenYearCHD            4240 non-null  int64
dtypes: float64(9), int64(7)
memory usage: 530.1 KB

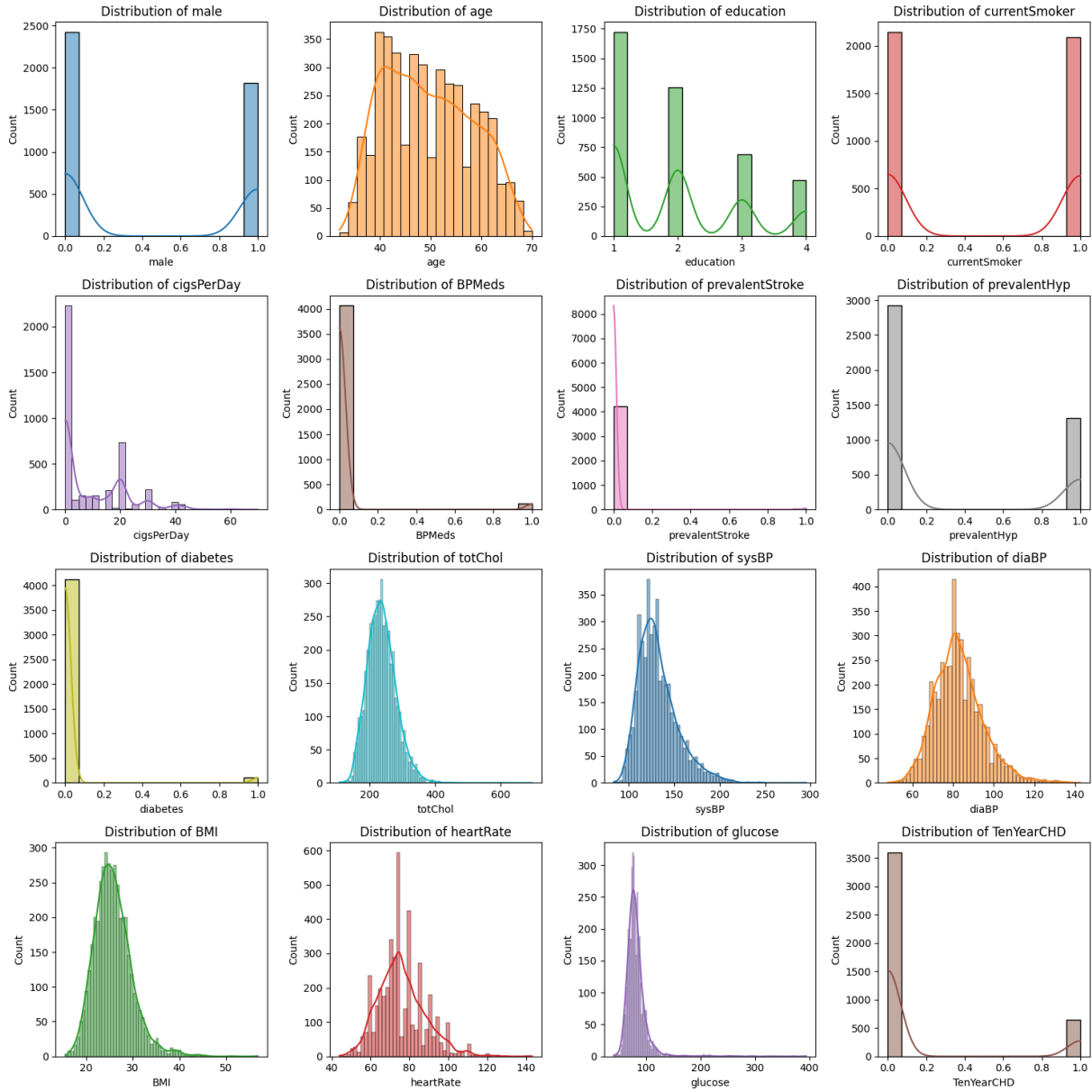
```

'male', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds',
 'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
 'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD'

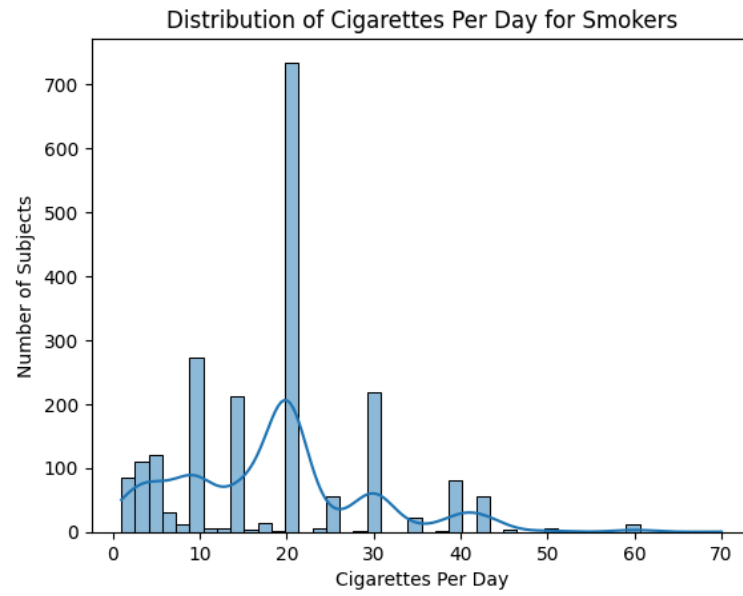
df_visualize



▼ df original



▼ Smokers: 2005



```

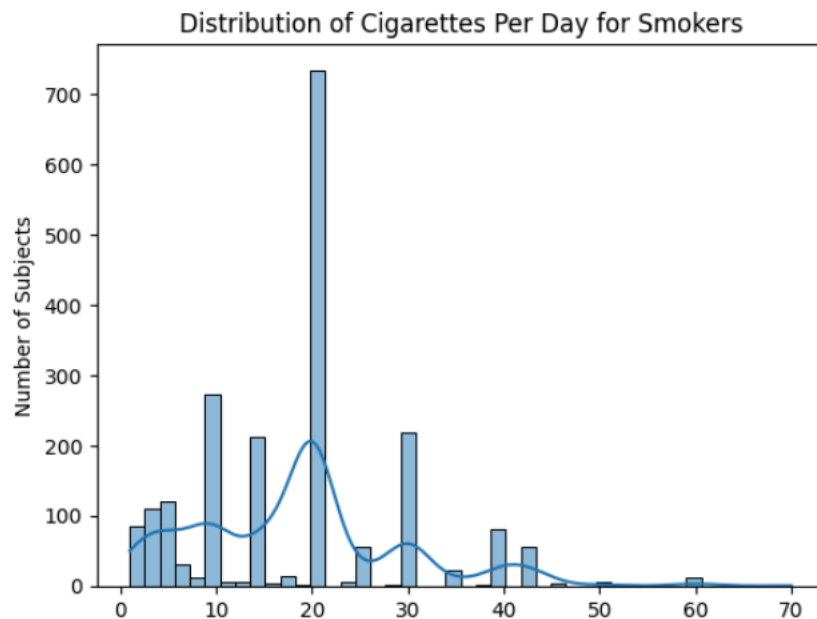
1 df_smoker = df_visualize[df_visualize['currentSmoker'] == 'Y']
2 df_smoker.info()
3
4 # Plot the distribution of cigsPerDay for those records
5 sns.histplot(df_smoker['cigsPerDay'], kde=True)
6 plt.title("Distribution of Cigarettes Per Day for Smokers")
7 plt.xlabel("Cigarettes Per Day")
8 plt.ylabel("Number of Subjects")
9 plt.show()

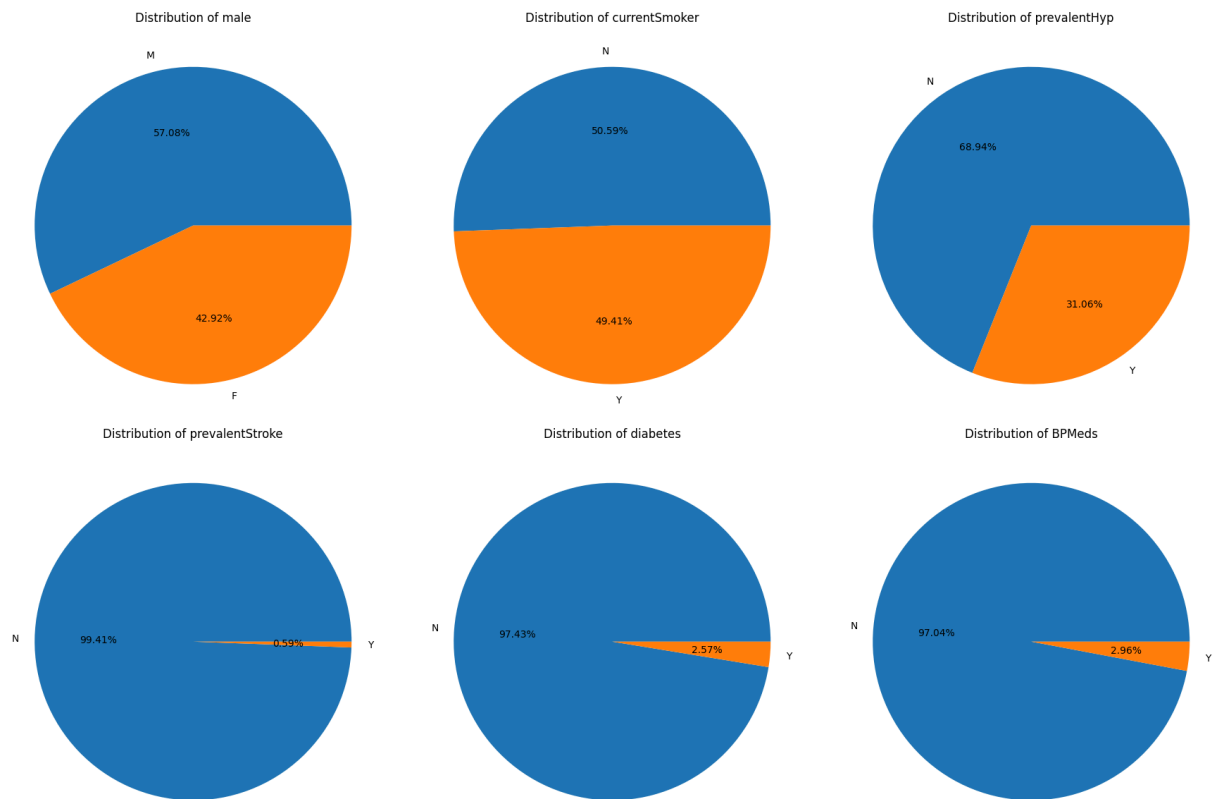
```

```

<class 'pandas.core.frame.DataFrame'>
Index: 2095 entries, 2 to 4239
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   male                  2095 non-null   object
1   age                   2095 non-null   int64
2   education             2046 non-null   float64
3   currentSmoker         2095 non-null   object
4   cigsPerDay            2066 non-null   float64
5   BPMeds               2072 non-null   float64
6   prevalentStroke       2095 non-null   object
7   prevalentHyp          2095 non-null   object
8   diabetes              2095 non-null   object
9   totChol               2064 non-null   float64
10  sysBP                 2095 non-null   float64
11  diaBP                 2095 non-null   float64
12  BMI                   2088 non-null   float64
13  heartRate             2094 non-null   float64
14  glucose               1890 non-null   float64
15  TenYearCHD           2095 non-null   object
dtypes: float64(9), int64(1), object(6)
memory usage: 278.2+ KB

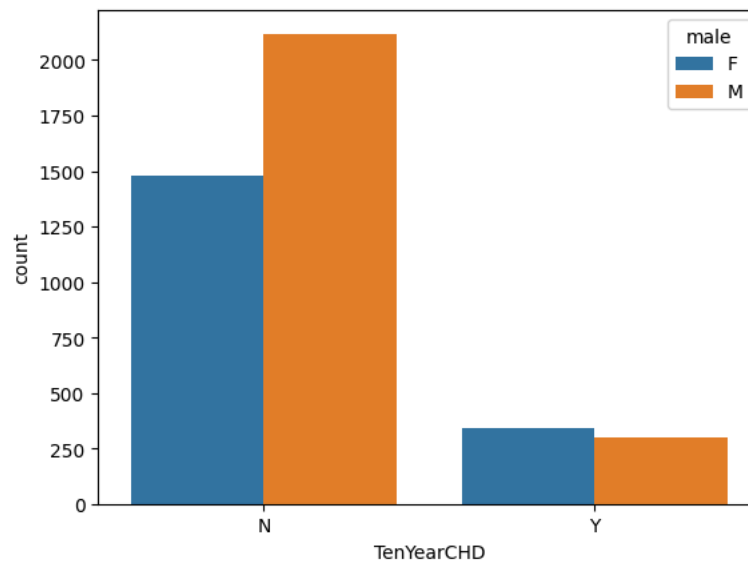
```

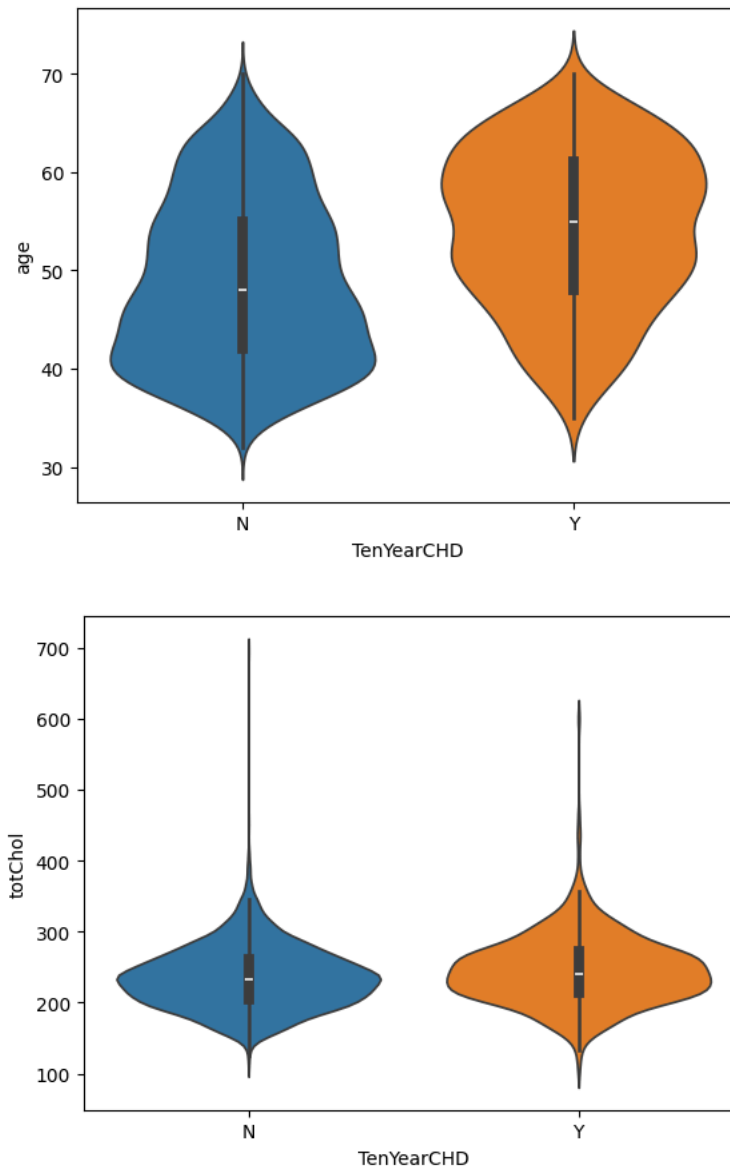




Violin Plot

```
sns.violinplot(x=df_visualize["age"])
sns.countplot(x=df_visualize["TenYearCHD"],hue=df_visualize["male"])
```





Logistic Regression

1. Data Encoding

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

# Encode all columns with LabelEncoder (this converts categorical values to numeric)
df = df.apply(LabelEncoder().fit_transform)

# Split data into features (X) and target (y)
```

```
X = df.drop(columns=['TenYearCHD']) # All columns except the target
y = df['TenYearCHD'] # Target column
```

2. Logistic Regression Model

```
# Split the data into train and test sets (stratified to preserve target class proportions)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize the features for logistic regression (helps the model converge)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3. Training, Predicting & Evaluating Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_curve, auc
import plotly.express as px

# Initialize and train logistic regression with increased iterations if needed
log_model = LogisticRegression(max_iter=1000, random_state=42)
log_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_log = log_model.predict(X_test_scaled)

# Print the classification report
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_log))

# ROC-AUC Calculation for Logistic Regression
fpr, tpr, _ = roc_curve(y_test, log_model.predict_proba(X_test_scaled)[:, 1])
roc_auc = auc(fpr, tpr)
print("Logistic Regression ROC-AUC:", roc_auc)

# Plot ROC Curve using Plotly Express
roc_data = pd.DataFrame({'False Positive Rate': fpr, 'True Positive Rate': tpr})
fig = px.line(roc_data, x='False Positive Rate', y='True Positive Rate',
               title=f'Logistic Regression ROC Curve (AUC = {roc_auc:.4f})',
               labels={'False Positive Rate': 'False Positive Rate', 'True Positive Rate': 'True Positive Rate'})
# Add the reference line for a random classifier
fig.add_shape(type='line', x0=0, y0=0, x1=1, y1=1, line=dict(dash='dash', color='grey'))
fig.show()
```

```
Logistic Regression Classification Report:
precision  recall  f1-score  support
```


0	0.85	0.99	0.92	719
1	0.47	0.05	0.10	129
accuracy			0.85	848
macro avg	0.66	0.52	0.51	848
weighted avg	0.79	0.85	0.79	848

Logistic Regression ROC-AUC: 0.6995935353796724

```

Logistic Regression Classification Report:
precision    recall  f1-score   support

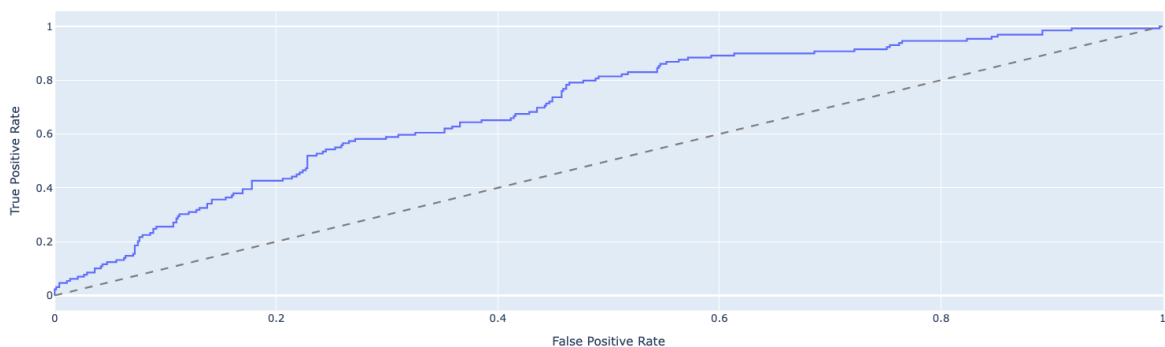
     0       0.85       0.99       0.92       719
     1       0.47       0.05       0.10       129

 accuracy          0.85          848
 macro avg          0.66          0.52          0.51          848
 weighted avg          0.79          0.85          0.79          848

```

Logistic Regression ROC-AUC: 0.6995935353796724

Logistic Regression ROC Curve (AUC = 0.6996)



Decision Tree

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, roc_curve, auc

# Initialize and train the Decision Tree Classifier (limiting max_depth to control overfitting)
dt_model = DecisionTreeClassifier(max_depth=4, random_state=42)
dt_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt_model.predict(X_test)

# Print the classification report
print("\nDecision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))

# ROC-AUC Calculation for Decision Tree (if predict_proba is available)
if hasattr(dt_model, "predict_proba"):
    fpr_dt, tpr_dt, _ = roc_curve(y_test, dt_model.predict_proba(X_test)[:, 1])
    roc_auc_dt = auc(fpr_dt, tpr_dt)
    print("Decision Tree ROC-AUC:", roc_auc_dt)

```

```
# Optional: Plot ROC Curve using Plotly Express for Decision Tree
roc_data_dt = pd.DataFrame({'False Positive Rate': fpr_dt, 'True Positive Rate': tpr_dt})
fig_dt = px.line(roc_data_dt, x='False Positive Rate', y='True Positive Rate',
                  title=f'Decision Tree ROC Curve (AUC = {roc_auc_dt:.4f})',
                  labels={'False Positive Rate': 'False Positive Rate', 'True Positive Rate': 'True Positive Rate'})
fig_dt.add_shape(type='line', x0=0, y0=0, x1=1, y1=1, line=dict(dash='dash', color='grey'))
fig_dt.show()
```



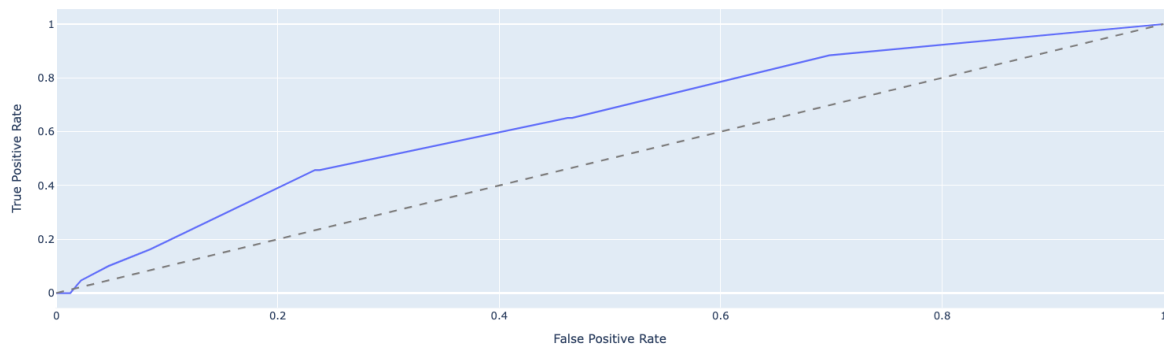
```
Decision Tree Classification Report:
      precision    recall  f1-score   support

     0       0.85       0.98       0.91       719
     1       0.27       0.05       0.08       129

 accuracy          0.84      848
 macro avg       0.56       0.51       0.49      848
 weighted avg    0.76       0.84       0.78      848
```

Decision Tree ROC-AUC: 0.6444135373203523

Decision Tree ROC Curve (AUC = 0.6444)



```
Decision Tree Classification Report:
      precision    recall  f1-score   support

     0       0.85       0.98       0.91       719
     1       0.27       0.05       0.08       129

 accuracy          0.84      848
 macro avg       0.56       0.51       0.49      848
 weighted avg    0.76       0.84       0.78      848
```

Decision Tree ROC-AUC: 0.6444135373203523