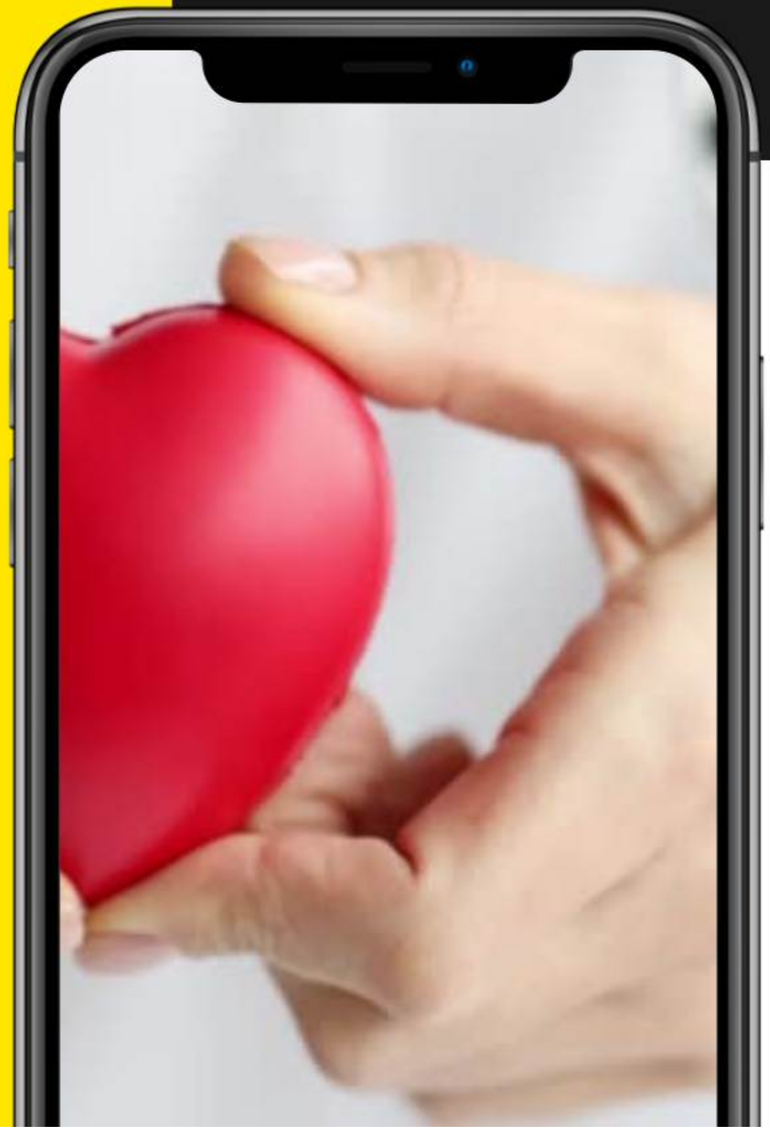
KELOMPOK 4 PRESENT

HEART DISEASE CLASSIFICATION

→ FINAL PITCHING INTERNSHIP
DATA SCIENCE BCC



HEART DISEASEASE CLEVELAND UCI

OUR TEAM

KELOMPOK 4

Rangga Andhito D.
Teknologi Informasi '21

as a
MENTOR

Muhammad Rizqi Azhari
Teknik Informatika '21

as a
MEMBER ONE

M. Hasan Fadhlillah
Teknik Informatika '22

as a
MEMBER TWO

Data & Project Under-standing

FINAL PROJECT DATA SCIENCE BCC

Heart Disease Classifiaction

Dataset

Judul:

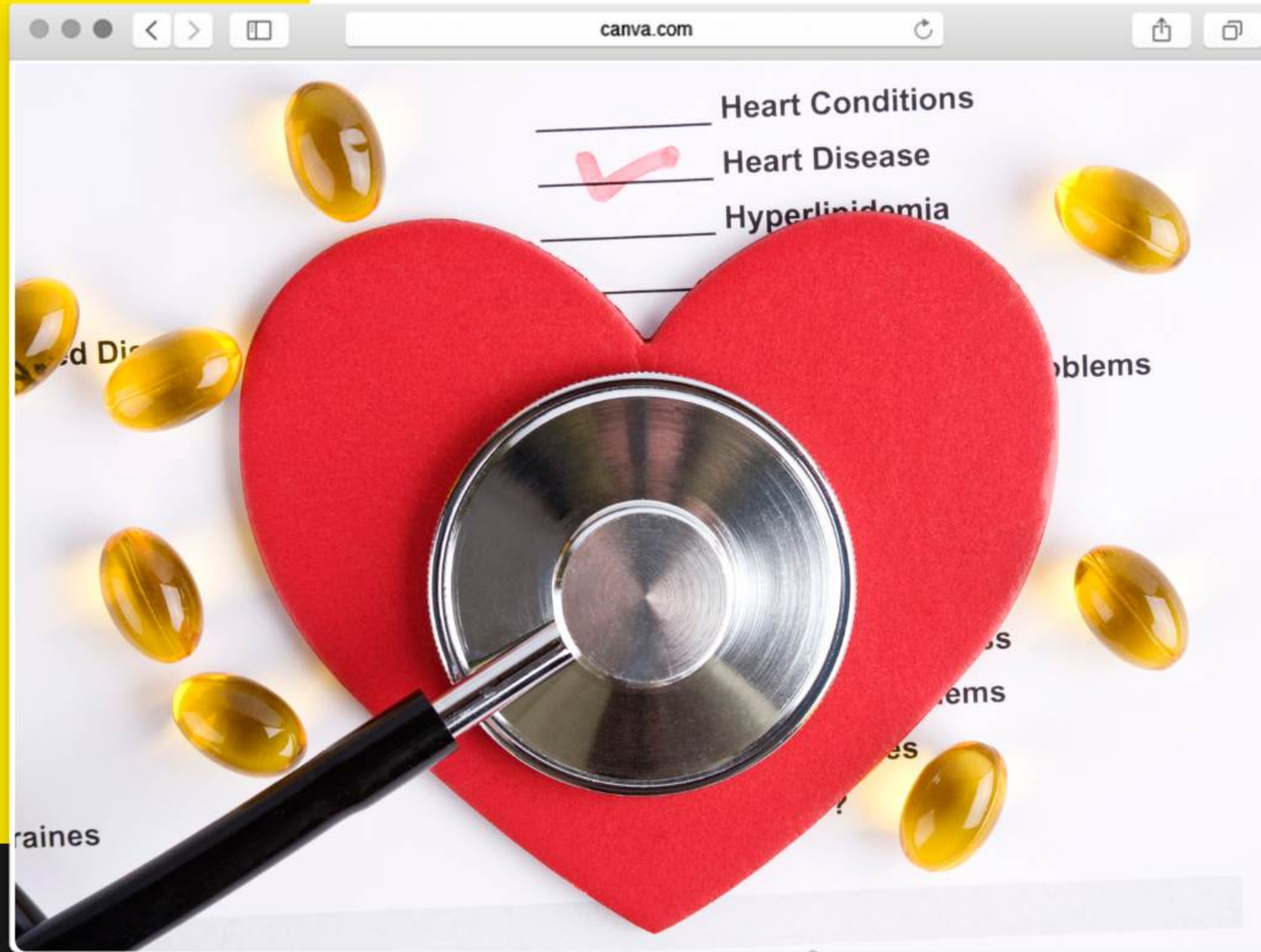
Heart Disease Cleveland UCI

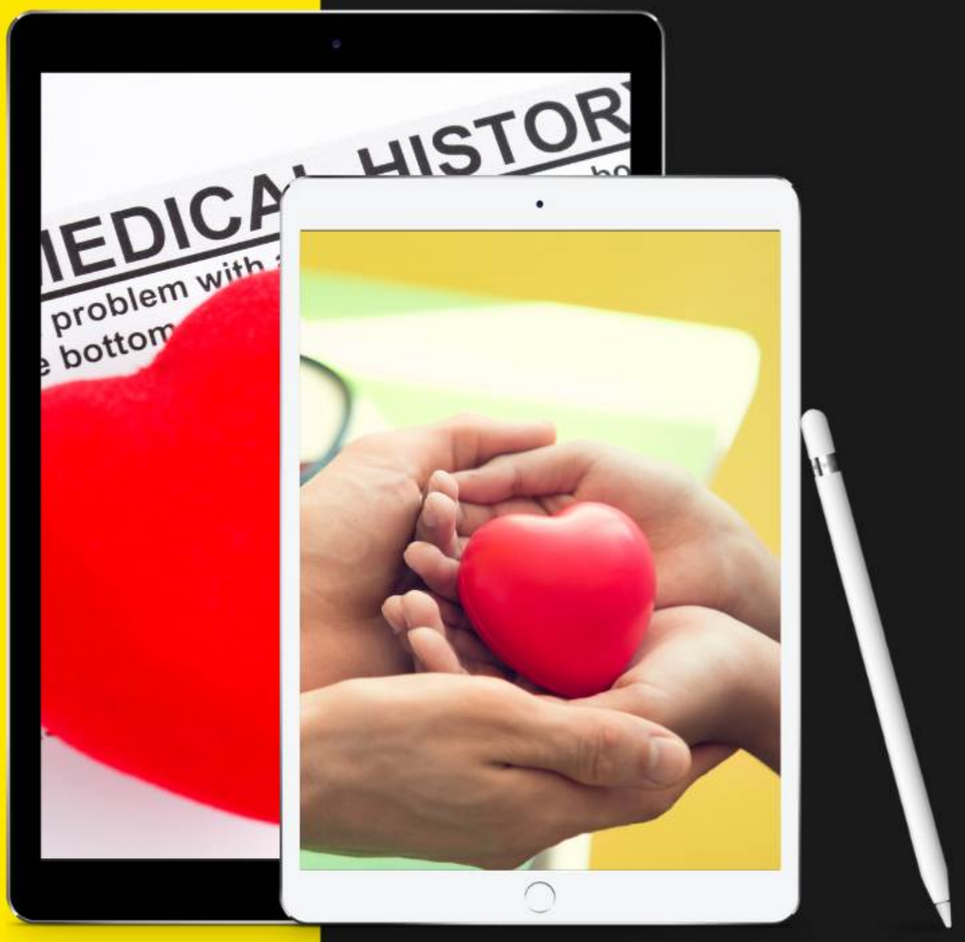
Source:

Kaggle

Link:

https://www.kaggle.com/datasets/cherngs/heart-disease-cleveland-uci





LATAR BELAKANG

Dataset "Heart Disease Cleveland UCI" berisi informasi tentang karakteristik klinis pasien dan diagnosis mereka terkait penyakit jantung. Masalah kesehatan terkait penyakit jantung adalah masalah serius di seluruh dunia, khususnya di negara Cleveland, Amerika Serikat. Penyakit jantung merupakan penyebab utama kematian di banyak negara. Untuk itu, membangun model machine learning yang dapat memprediksi risiko penyakit jantung berdasarkan karakteristik klinis dapat membantu dalam diagnosis dini dan pencegahan penyakit jantung pada populasi yang rentan. Oleh karena itu, analisis dataset ini dapat membantu dokter, peneliti, dan perusahaan asuransi kesehatan untuk memahami faktor-faktor risiko dan membangun model prediksi untuk membantu memperbaiki kesehatan pasien dan mengurangi risiko kesehatan di masa depan.

TUJUAN & MANFAAT

1. Mengidentifikasi faktor-faktor risiko yang menyebabkan penyakit jantung
2. Meningkatkan pemahaman tentang karakteristik klinis pasien yang menyebabkan penyakit jantung
3. Membangun model machine learning yang dapat memprediksi risiko penyakit jantung berdasarkan karakteristik klinis
4. Mendeteksi apakah seseorang menderita penyakit jantung atau tidak.

CONTENT INFO

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
   -- Value 0: typical angina
   -- Value 1: atypical angina
   -- Value 2: non-anginal pain
   -- Value 3: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
   -- Value 0: normal
   -- Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
   -- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
   -- Value 0: upsloping
   -- Value 1: flat
   -- Value 2: downsloping
12. ca: number of major vessels (0-3) colored by flourosopy
13. thal: 0 = normal; 1 = fixed defect; 2 = reversable defect and the label
14. condition: 0 = no disease, 1 = disease

Melihat 5 Data Teratas

```
df_heart.head() # tampilkan 5 data teratas
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	condition
0	69	1	0	160	234	1	2	131	0	0.1	1	1	0	0
1	69	0	0	140	239	0	0	151	0	1.8	0	2	0	0
2	66	0	0	150	226	0	0	114	0	2.6	2	0	0	0
3	65	1	0	138	282	1	2	174	0	1.4	1	1	0	1
4	64	1	0	110	211	0	2	144	1	1.8	1	0	0	0

Melihat Distribusi Data

```
# distribusi data
df_heart.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	condition
count	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000
mean	54.542088	0.676768	2.158249	131.693603	247.350168	0.144781	0.996633	149.599327	0.326599	1.055556	0.602694	0.676768	0.835017	0.461279
std	9.049736	0.468500	0.964859	17.762806	51.997583	0.352474	0.994914	22.941562	0.469761	1.166123	0.618187	0.938965	0.956690	0.499340
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	2.000000	120.000000	211.000000	0.000000	0.000000	133.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	56.000000	1.000000	2.000000	130.000000	243.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	0.000000	0.000000
75%	61.000000	1.000000	3.000000	140.000000	276.000000	0.000000	2.000000	166.000000	1.000000	1.600000	1.000000	1.000000	2.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	3.000000	2.000000	1.000000

Melihat Tipe Data dan Dimensi Data

```
# informasi tipe data di setiap feature
df_heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 297 entries, 0 to 296
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   age        297 non-null    int64  
 1   sex        297 non-null    int64  
 2   cp         297 non-null    int64  
 3   trestbps   297 non-null    int64  
 4   chol       297 non-null    int64  
 5   fbs        297 non-null    int64  
 6   restecg    297 non-null    int64  
 7   thalach    297 non-null    int64  
 8   exang      297 non-null    int64  
 9   oldpeak    297 non-null    float64
 10  slope      297 non-null    int64  
 11  ca         297 non-null    int64  
 12  thal       297 non-null    int64  
 13  condition  297 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 32.6 KB
```

```
# dimensi dataframe
df_heart.shape
```

```
(297, 14)
```

OVERVIEW

Berikut adalah hal-hal yang telah kami lakukan untuk project kami selama 4 minggu internship.

- ✓ Data Cleaning
- ✓ Exploratory Data Analysis (EDA)
- ✓ Scaling / Transformation
- ✓ Encoding
- ✓ ML Modelling

Data Cleaning

FINAL PROJECT DATA SCIENCE BCC

Heart Disease Classifiaction

Mengecek Data Kosong dan Duplikat

```
# mengecek apakah terdapat data kosong
df_heart.isna().sum()
```

```
age          0
sex          0
cp           0
trestbps     0
chol         0
fbs          0
restecg      0
thalach      0
exang        0
oldpeak      0
slope        0
ca           0
thal         0
condition    0
dtype: int64
```

```
# mengecek apakah ada data duplikat
df_heart.duplicated().sum()
```

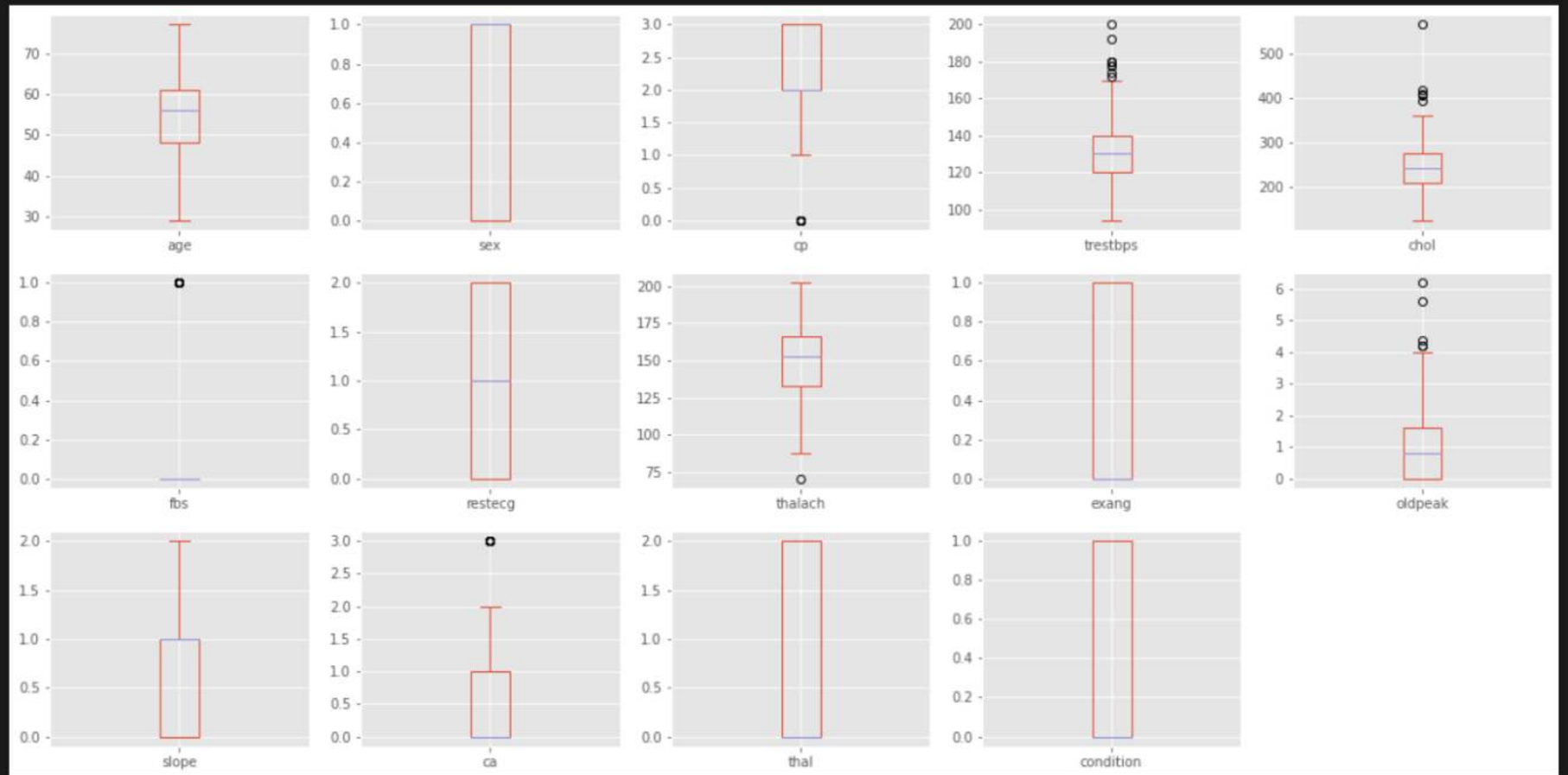
```
0
```

Deteksi Outliers

```
# deteksi outliers menggunakan Isolation Forest
from sklearn.ensemble import IsolationForest

# Isolation Forest
iso = IsolationForest(contamination=0.1, random_state=42)
yhat = iso.fit_predict(df_heart)

# Print columns with outliers
outlier_cols = df_heart.columns[(yhat == -1).any(axis=0)]
print(f"Columns with outliers: {outlier_cols}")

# Print columns without outliers
non_outlier_cols = df_heart.columns[(yhat != -1).all(axis=0)]
print(f"Columns without outliers: {non_outlier_cols}")
```

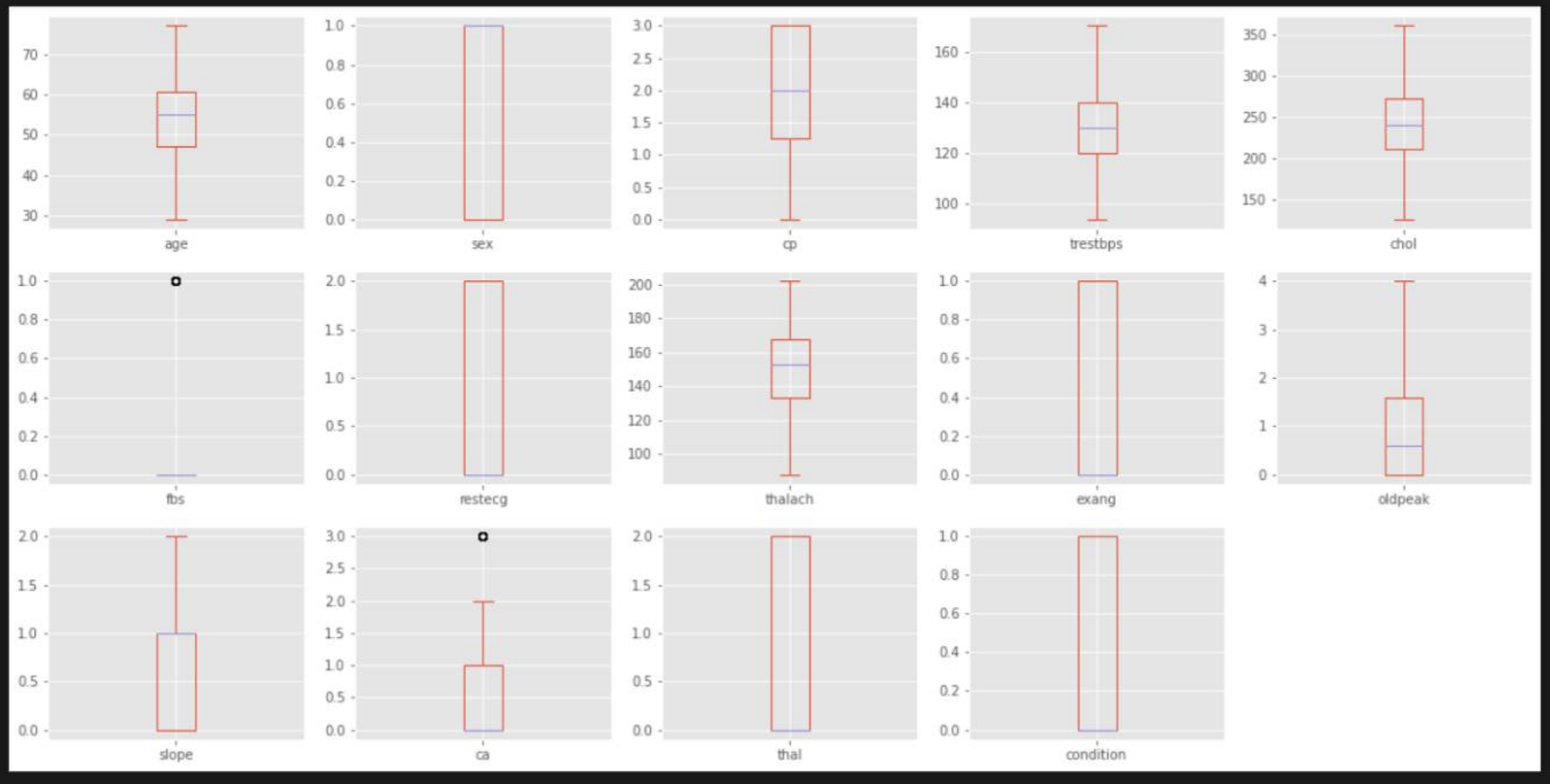
```
Columns with outliers: [['age' 'sex' 'cp' 'trestbps' 'chol' 'fbs' 'restecg' 'thalach' 'exang'
  'oldpeak' 'slope' 'ca' 'thal' 'condition']]
Columns without outliers: []
```

Deteksi Outliers



Sebelum Cleaning

Deteksi Outliers



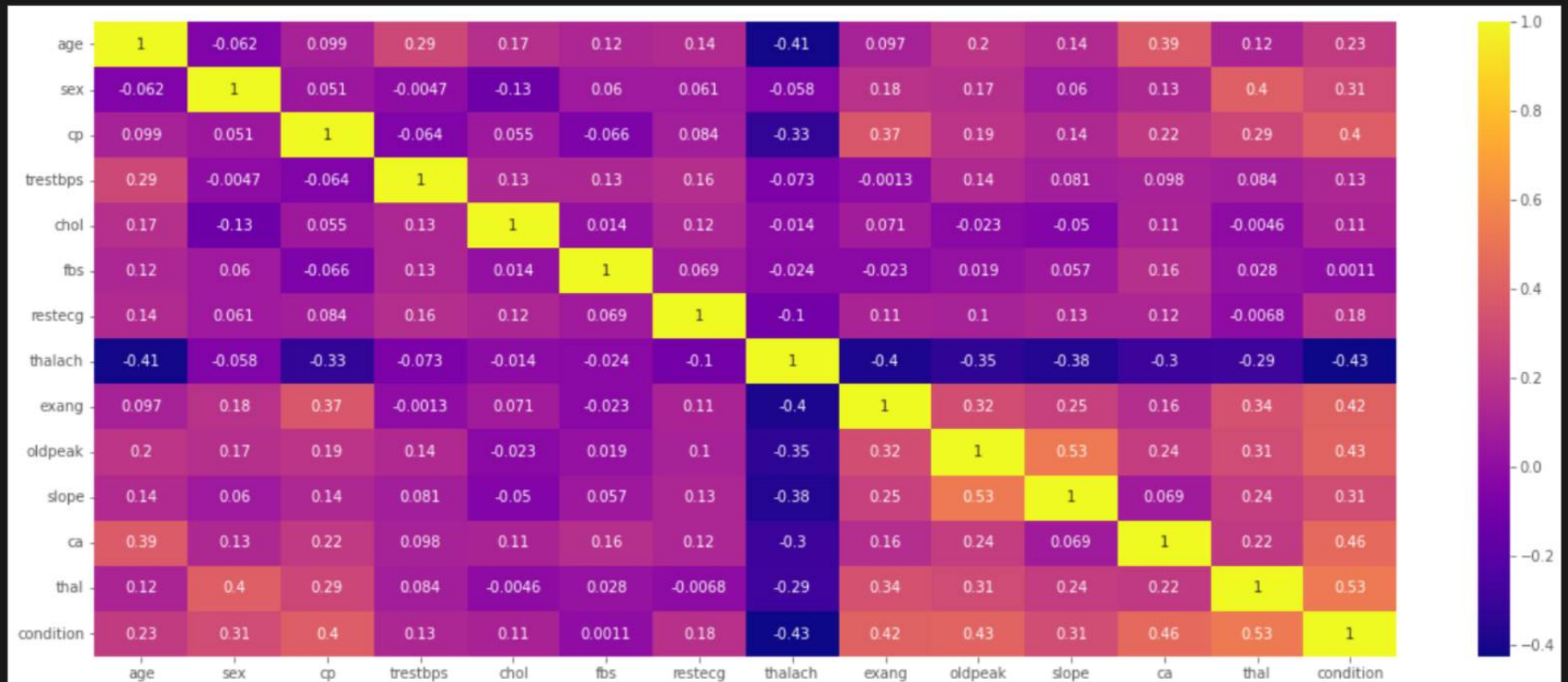
Setelah Cleaning

Exploratory Data Analysis

FINAL PROJECT DATA SCIENCE BCC

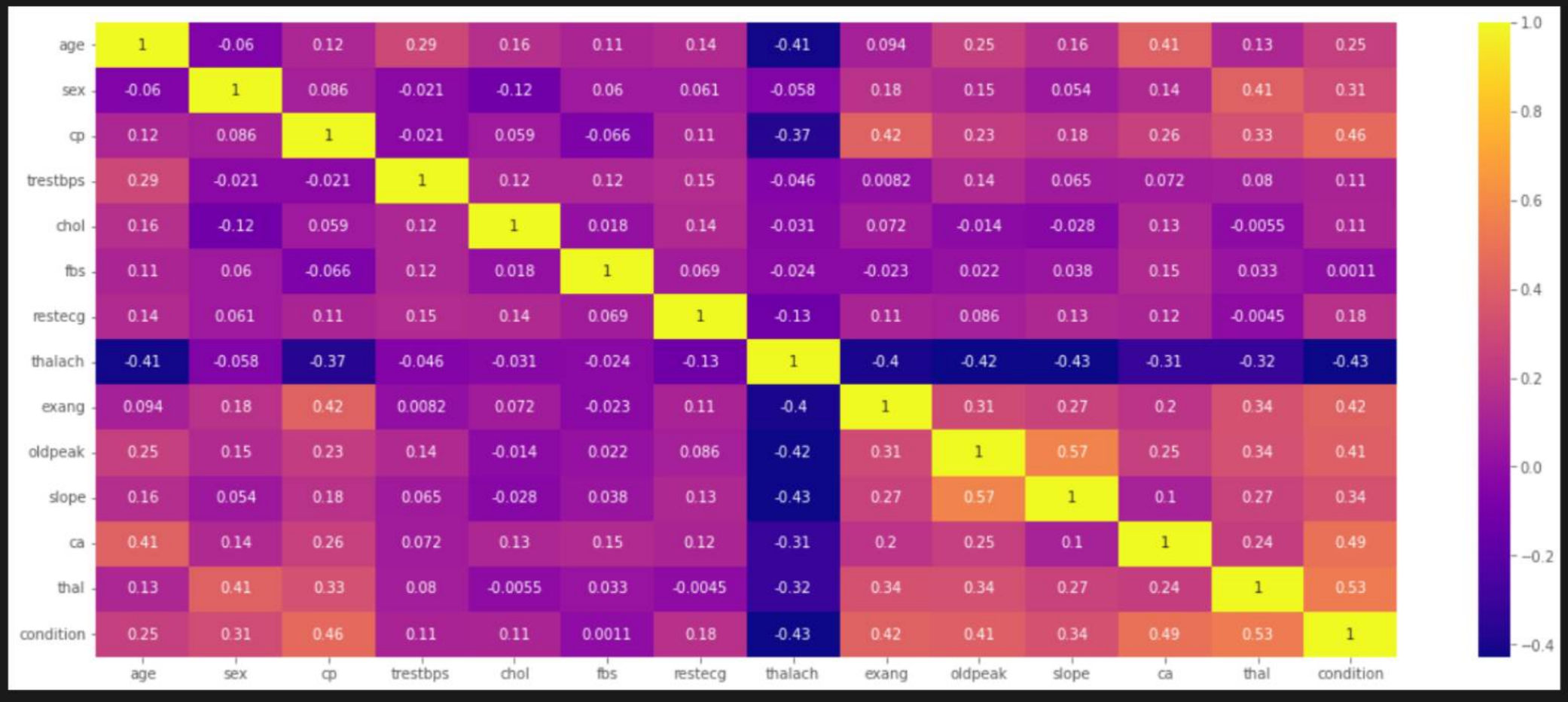
Heart Disease Classifiaction

Melihat Korelasi



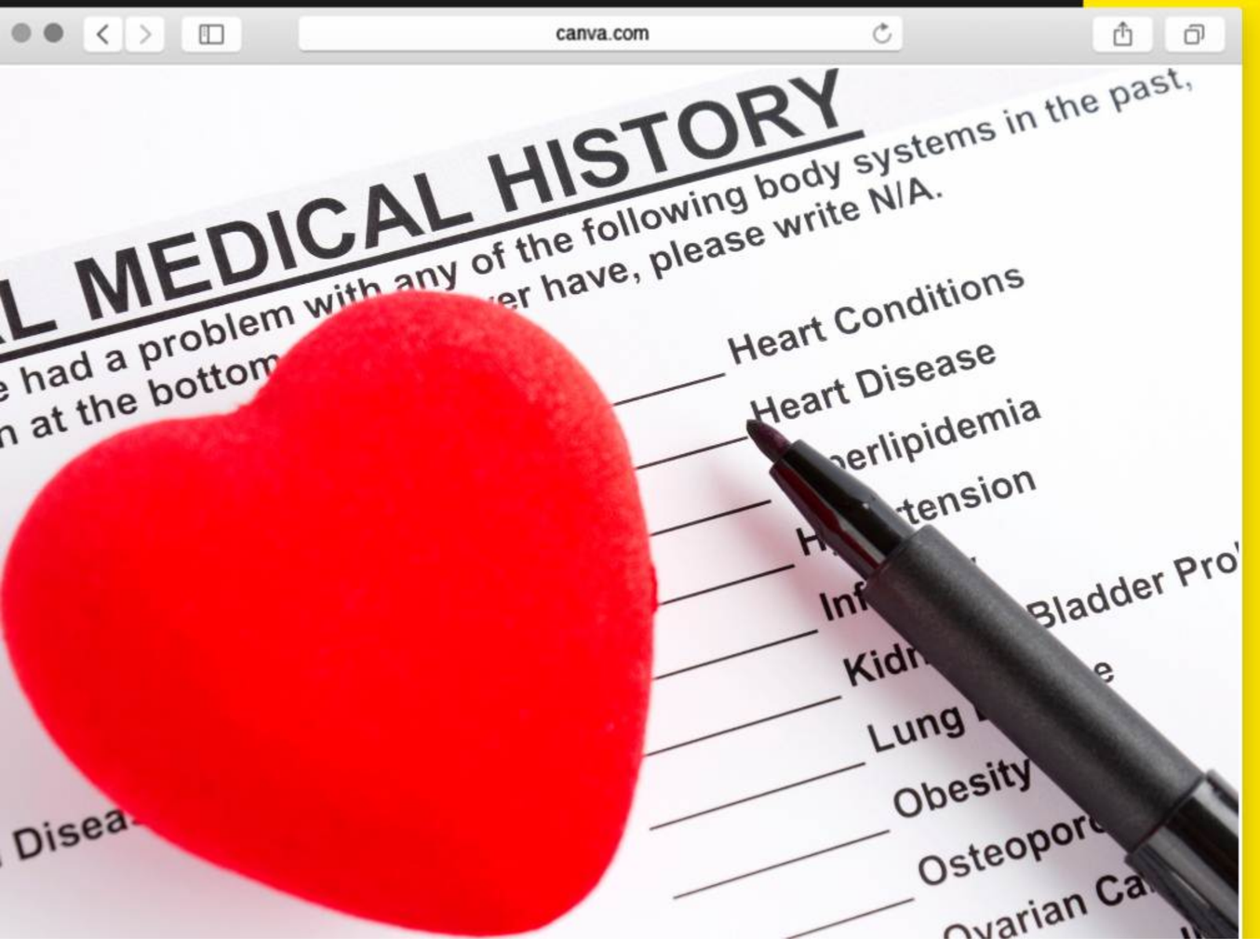
Metode Pearson

Deteksi Outliers



Metode Spearman

Hasil Korelasi

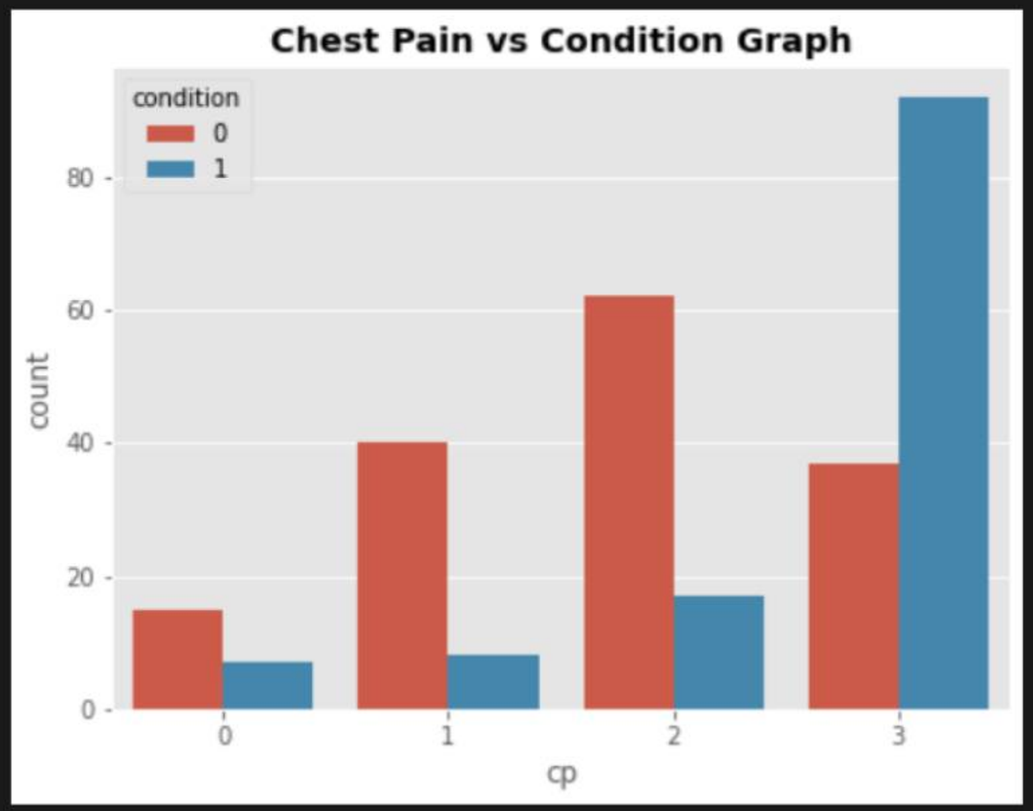


Beberapa fitur yang berkorelasi tinggi:

1. Age dan ca
2. Sex dan thal
3. cp dan exang
4. cp dan condition
5. exang dan condition
6. oldpeak dan slope
7. oldpeak dan condition
8. ca da condition
9. thal dan condition

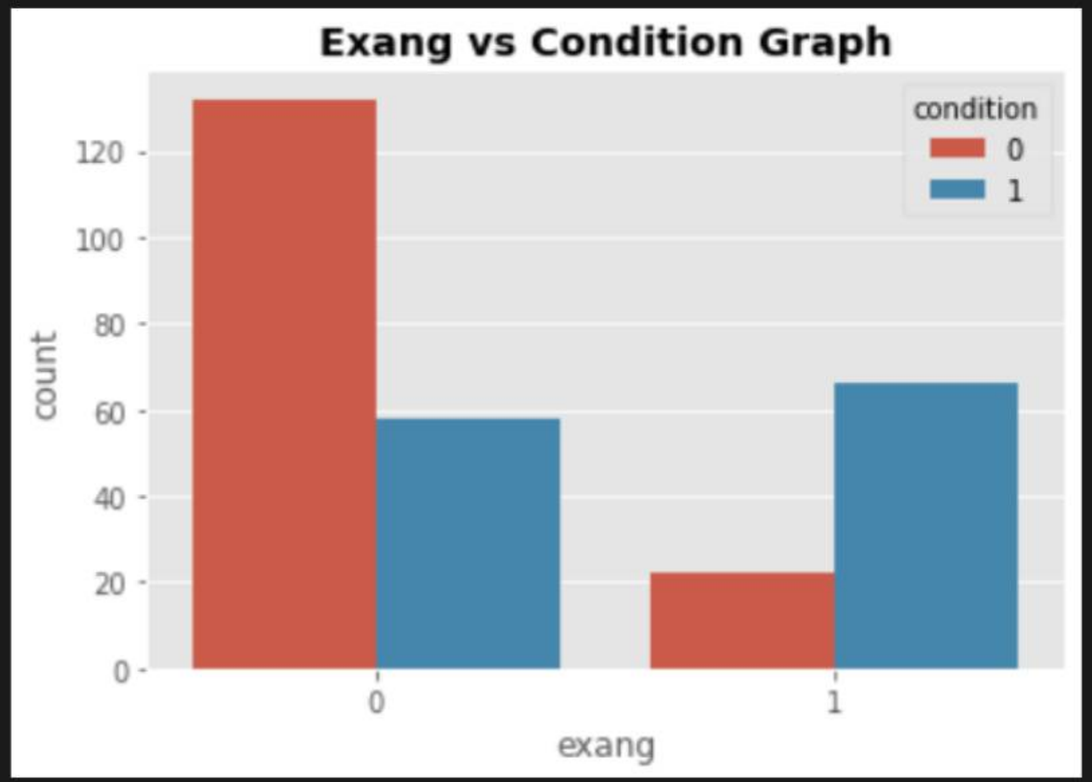
Namun kita hanya akan menganalisis fitur yang berkorelasi langsung dengan condition

CP vs Condition



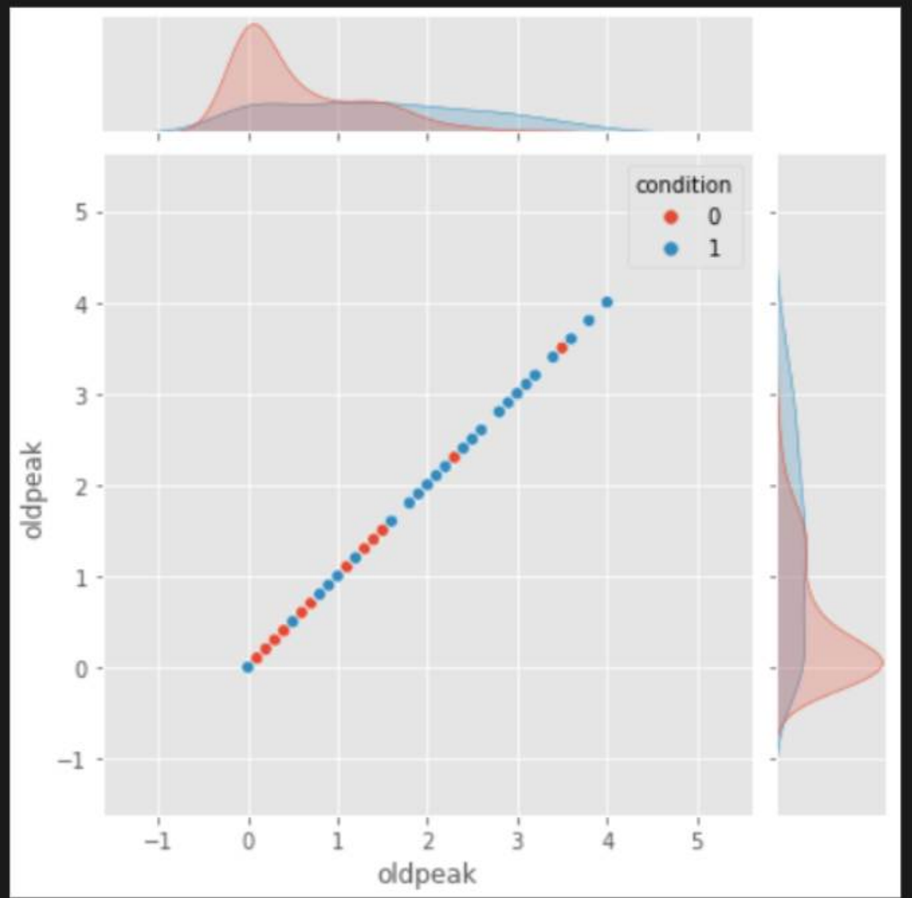
Dapat dilihat bahwa orang yang memiliki chest pain type 3 memiliki kemungkinan besar terkena penyakit jantung.

Exang vs Condition



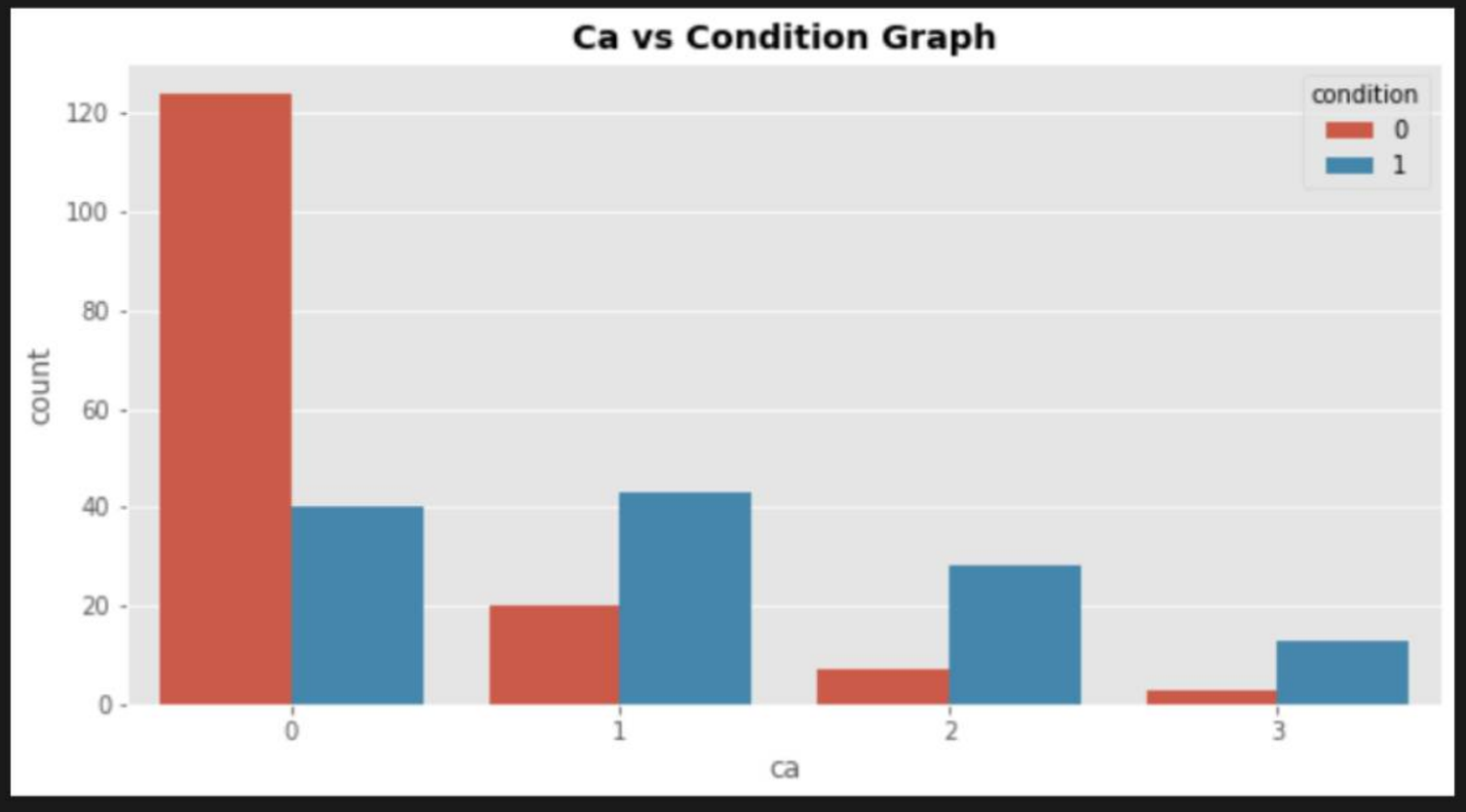
Dapat dilihat bahwa orang yang tidak memiliki exercise induced angina memiliki kemungkinan yang lebh kecil untuk terkena penyakit jantung

Oldpeak vs Condition



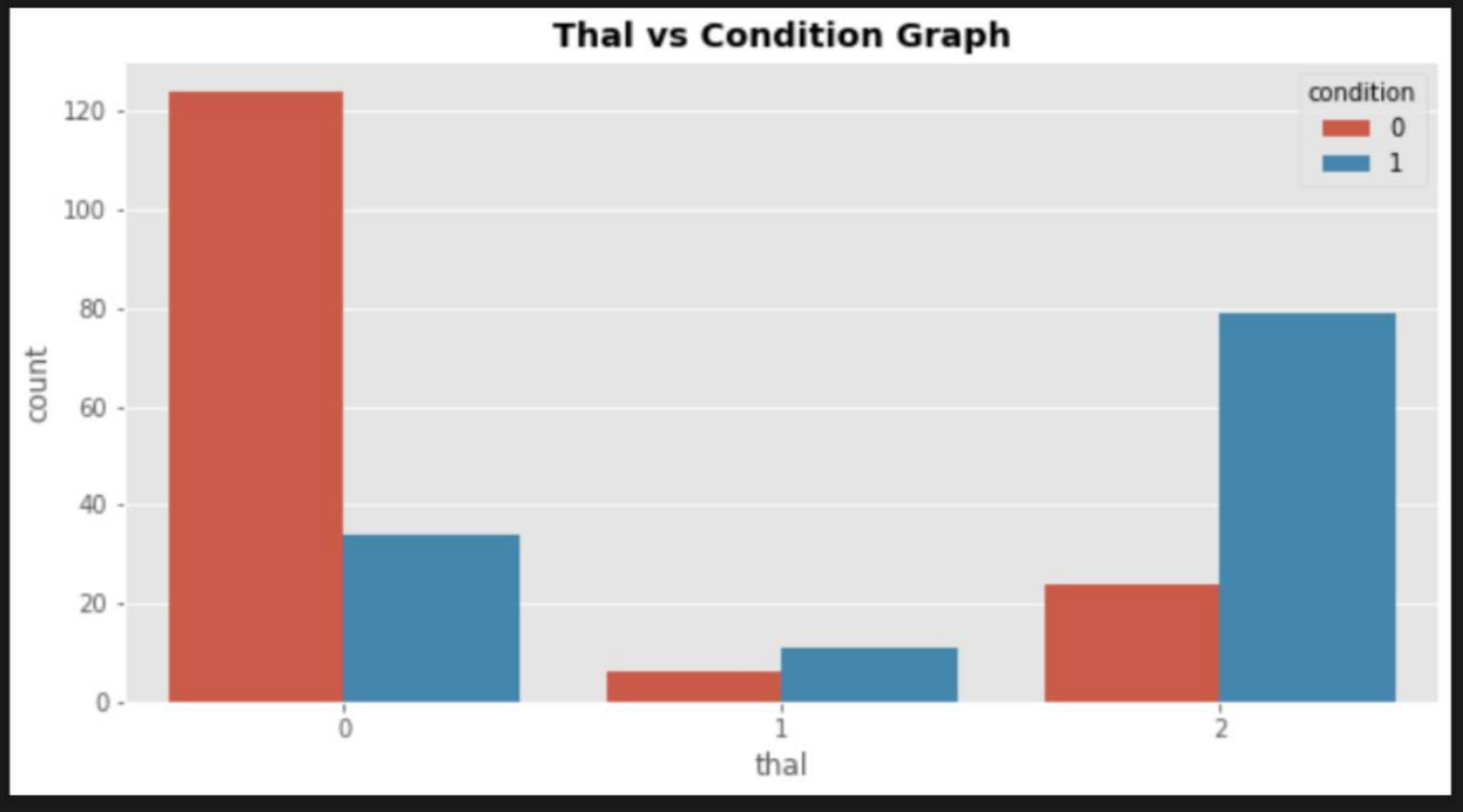
Orang yang memiliki oldpeak lebih dari 2 memiliki kemungkinan lebih besar untuk terkena penyakit jantung

CA vs Condition



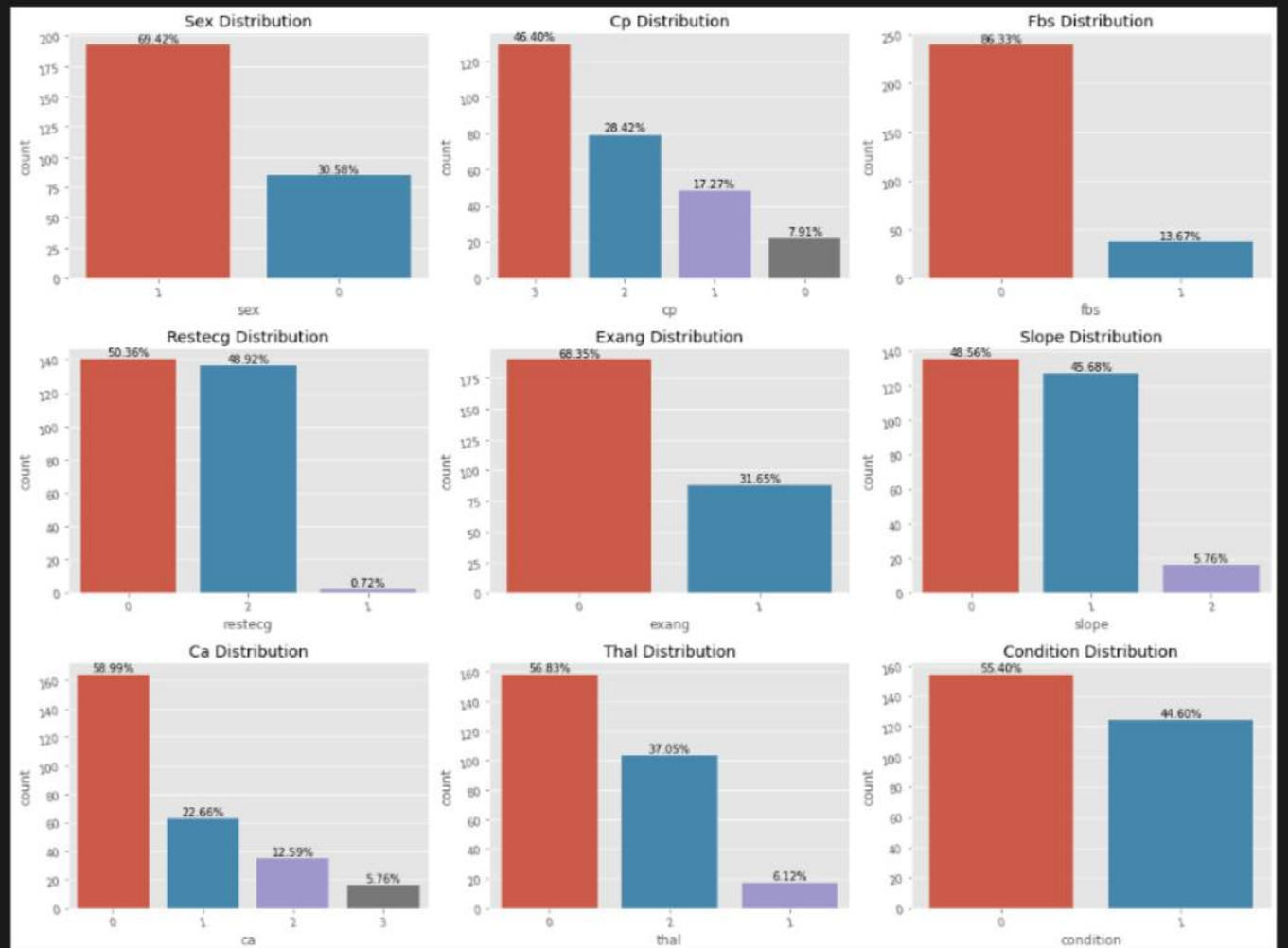
Dapat dilihat bahwa orang yang memiliki ca dengan nilai 0 memiliki kemungkinan lebih kecil untuk terkena penyakit jantung, sementara orang yang memiliki ca dengan nilai 3 memiliki kemungkinan yang besar untuk terkena penyakit jantung

Thal vs Condition



Dapat dilihat bahwa orang yang memiliki thalassemia normal (0) memiliki kemungkinan yang kecil untuk terkena penyakit jantung, sementara orang yang memiliki thalassemia reversable defect (2) memiliki kemungkinan terbesar untuk terkena penyakit jantung.

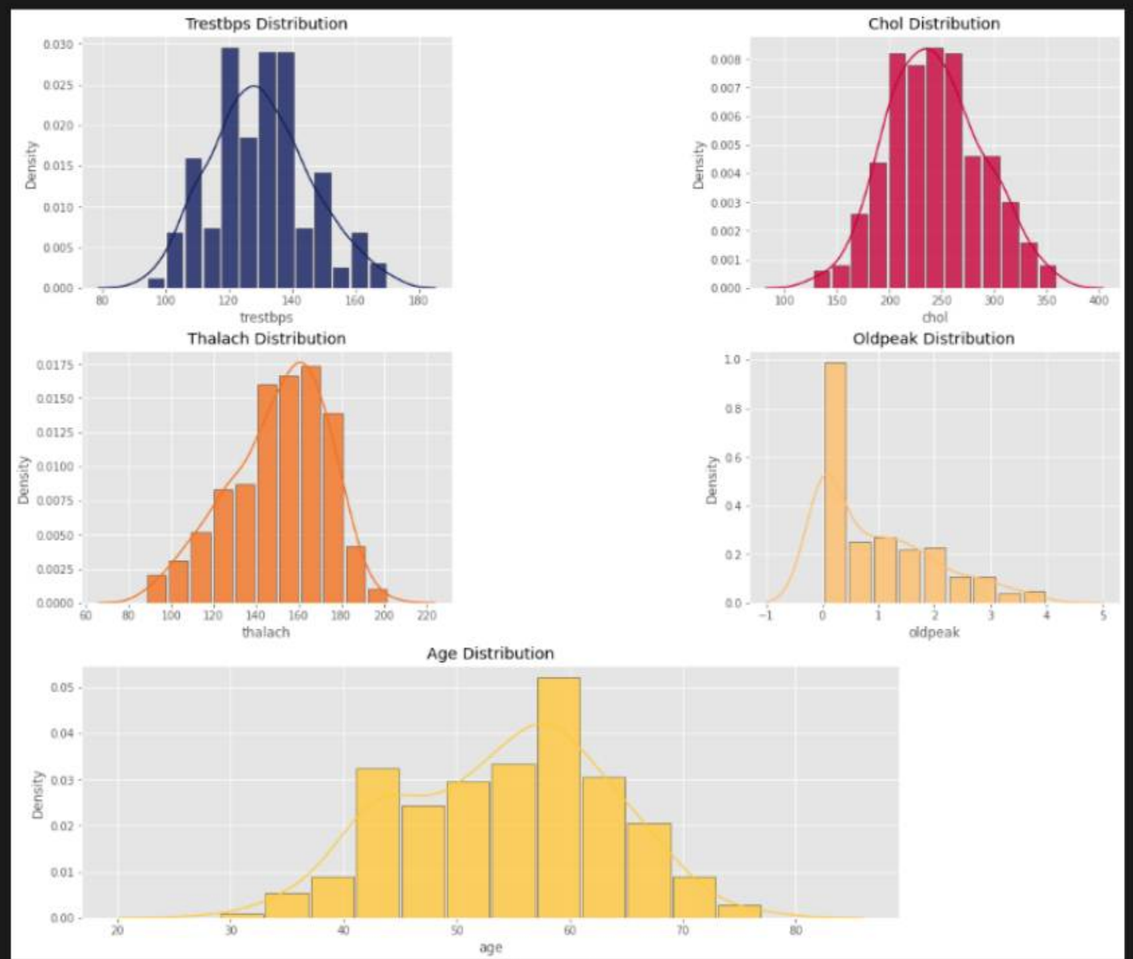
Univariate Analysis



Categorical Data

- Laki-laki lebih banyak dari perempuan.
- Tipe 'chest pain' yang paling umum adalah asymptomatic
- Lebih dari 85% pasien tidak memiliki gula darah puasa yang tinggi.
- Restecg relatif terdistribusi merata
- Lebih dari 67% pasien tidak memiliki exercise induced angina
- Slope lebih terdistribusi antara upsloping dan flat.

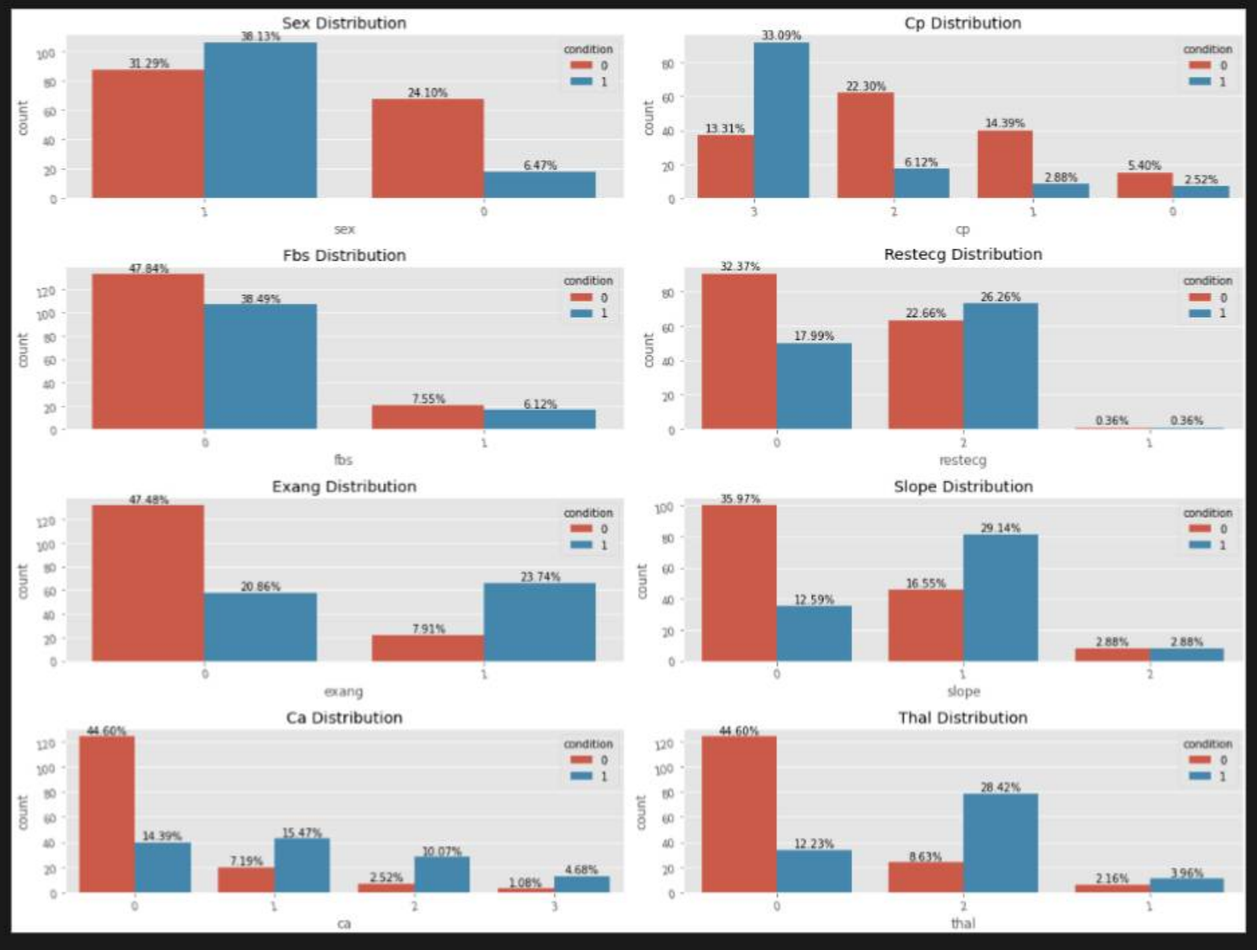
Univariate Analysis



Numerical Data

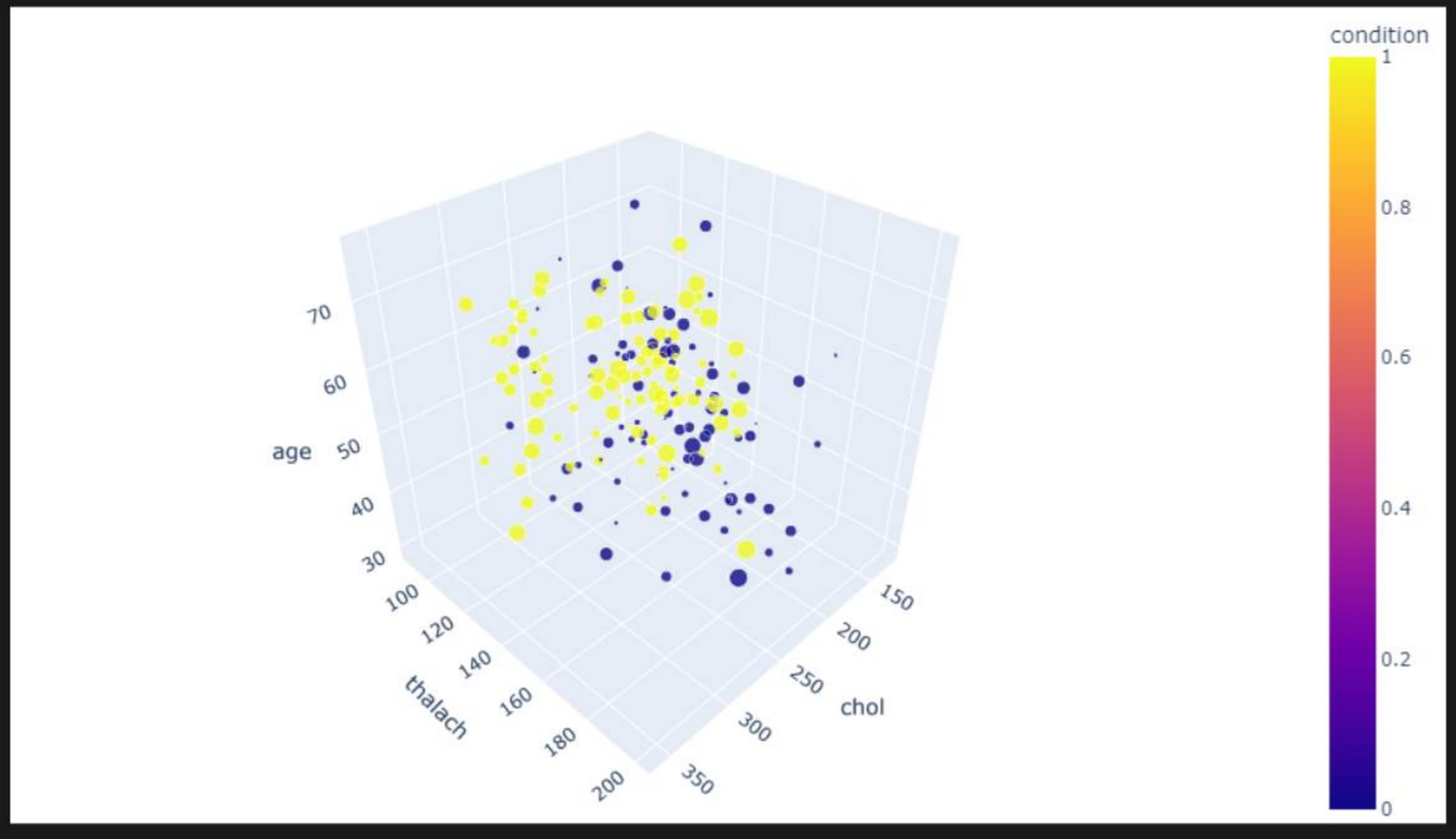
Hampir semua data numerik terdistribusi merata dengan sedikit kecondongan kecuali oldpeak yang lebih condong ke kiri.

Bivariate Analysis



- Laki-laki memiliki kemungkinan lebih tinggi terkena penyakit jantung.
- Asymptomatic chest pain type memiliki jumlah penyakit jantung terbanyak.
- Gula darah tidak berpengaruh pada penyakit jantung.
- RestECG tidak memiliki pengaruh secara langsung terhadap penyakit jantung, namun apabila pasien memiliki ST-T wave yang abnormal maka pasien tersebut memiliki kemungkinan yang sangat tinggi terkena penyakit jantung.
- Exercise induced angina memiliki pengaruh kuat dalam penyakit jantung.
- Pasien yang memiliki flat slope memiliki kemungkinan yang lebih tinggi terkena penyakit jantung.
- Pasien yang memiliki defected thalium memiliki kemungkinan yang lebih tinggi terkena penyakit jantung.

Multivariate Analysis



X axis menunjukkan Cholesterol, Y menunjukkan thalach dan Z axis adalah Age, marker sizes adalah oldpeak dan warna menunjukkan condition.

Scaling / Transfor-mation

FINAL PROJECT DATA SCIENCE BCC

Heart Disease Classifiaction

MinMax Scaling

```
# normalisasi MinMax
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
list_fitur = [df_heart['trestbps'], df_heart['chol'], df_heart['thalach'], df_heart['oldpeak']]
scaled_data = scaler.fit_transform(list_fitur)
df_scaled_mm = pd.DataFrame(scaled_data.T, columns=['trestbps', 'chol', 'thalach', 'oldpeak'])
print(df_scaled_mm.shape)
df_scaled_mm.head()
```

```
(278, 4)
```

	trestbps	chol	thalach	oldpeak
0	0.683625	1.0	0.559641	0.0
1	0.582631	1.0	0.629005	0.0
2	0.659803	1.0	0.498657	0.0
3	0.486814	1.0	0.615110	0.0
4	0.517208	1.0	0.679732	0.0

Data yang sudah dibersihkan outliernya kemudian di scaling menggunakan metode MinMax

Robust Scaling

```
# Scaling menggunakan Robust Scaler
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
list_fitur = [df_heart_copy['trestbps'], df_heart_copy['chol'], df_heart_copy['thalach'], df_heart_copy['oldpeak']]
scaled_data = scaler.fit_transform(list_fitur)
df_scaled_robust = pd.DataFrame(scaled_data.T, columns=['trestbps', 'chol', 'thalach', 'oldpeak'])
print(df_scaled_robust.shape)
df_scaled_robust.head()
```

```
(297, 4)
```

	trestbps	chol	thalach	oldpeak
0	0.180742	1.103147	-0.180742	-1.812403
1	-0.081421	1.384160	0.081421	-2.127313
2	0.217260	1.134581	-0.217260	-1.561859
3	-0.185280	1.296963	0.185280	-1.591354
4	-0.218509	1.079692	0.218509	-1.609254

Data yang masih memiliki outlier kemudian di scaling menggunakan metode Robust Scaler

Data Encoding

FINAL PROJECT DATA SCIENCE BCC

Heart Disease Classifiaction

One Hot Encoding (MinMax)

```
df_heart = df_heart.drop(columns=['trestbps', 'chol', 'thalach', 'oldpeak'], axis=1)
```

```
# encoding fitur categorical menggunakan One Hot Encoding (MinMax)
categorical = ["sex", "cp", "fbs", "restecg", "exang", "slope", "ca", "thal"]
df_mm_encoded = pd.get_dummies(df_heart, columns=categorical)
df_mm_encoded.head()
```

	age	condition	sex_0	sex_1	cp_0	cp_1	cp_2	cp_3	fbs_0	fbs_1	...	slope_0	slope_1	slope_2	ca_0	ca_1	ca_2	ca_3	thal_0	thal_1	thal_2
0	69	0	0	1	1	0	0	0	0	1	...	0	1	0	0	1	0	0	1	0	0
1	69	0	1	0	1	0	0	0	1	0	...	1	0	0	0	0	1	0	1	0	0
2	66	0	1	0	1	0	0	0	1	0	...	0	0	1	1	0	0	0	1	0	0
3	65	1	0	1	1	0	0	0	0	1	...	0	1	0	0	1	0	0	1	0	0
4	64	0	0	1	1	0	0	0	1	0	...	0	1	0	1	0	0	0	1	0	0

5 rows × 25 columns

One Hot Encoding (Robust)

```
df_heart = df_heart.drop(columns=['trestbps', 'chol', 'thalach', 'oldpeak'], axis=1)
```

```
# encoding fitur categorical menggunakan One Hot Encoding (Robust)
categorical = ["sex", "cp", "fbs", "restecg", "exang", "slope", "ca", "thal"]
df_robust_encoded = pd.get_dummies(df_heart_copy, columns=categorical)
df_robust_encoded.head()
```

	age	condition	sex_0	sex_1	cp_0	cp_1	cp_2	cp_3	fbs_0	fbs_1	...	slope_0	slope_1	slope_2	ca_0	ca_1	ca_2	ca_3	thal_0	thal_1	thal_2
0	69	0	0	1	1	0	0	0	0	1	...	0	1	0	0	1	0	0	1	0	0
1	69	0	1	0	1	0	0	0	1	0	...	1	0	0	0	0	1	0	1	0	0
2	66	0	1	0	1	0	0	0	1	0	...	0	0	1	1	0	0	0	1	0	0
3	65	1	0	1	1	0	0	0	0	1	...	0	1	0	0	1	0	0	1	0	0
4	64	0	0	1	1	0	0	0	1	0	...	0	1	0	1	0	0	0	1	0	0

5 rows × 25 columns

Data Merging

FINAL PROJECT DATA SCIENCE BCC

Heart Disease Classifiaction

Data Merging (MinMax)

```
[473] df_heart_mm = pd.concat([df_scaled_mm.reset_index(drop=True), df_mm_encoded.reset_index(drop=True)], axis=1)
      df_heart_robust = pd.concat([df_scaled_robust.reset_index(drop=True), df_robust_encoded.reset_index(drop=True)], axis=1)
```

```
df_heart_mm.head()
```

	trestbps	chol	thalach	oldpeak	age	condition	sex_0	sex_1	cp_0	cp_1	...	slope_0	slope_1	slope_2	ca_0	ca_1	ca_2	ca_3	thal_0	thal_1	thal_2
0	0.683625	1.0	0.559641	0.0	69	0	0	1	1	0	...	0	1	0	0	1	0	0	1	0	0
1	0.582631	1.0	0.629005	0.0	69	0	1	0	1	0	...	1	0	0	0	0	1	0	1	0	0
2	0.659803	1.0	0.498657	0.0	66	0	1	0	1	0	...	0	0	1	1	0	0	0	1	0	0
3	0.486814	1.0	0.615110	0.0	65	1	0	1	1	0	...	0	1	0	0	1	0	0	1	0	0
4	0.517208	1.0	0.679732	0.0	64	0	0	1	1	0	...	0	1	0	1	0	0	0	1	0	0

5 rows × 29 columns

Data Merging (Robust)

```
df_heart_robust.head()
```

	trestbps	chol	thalach	oldpeak	age	condition	sex_0	sex_1	cp_0	cp_1	...	slope_0	slope_1	slope_2	ca_0	ca_1	ca_2	ca_3	thal_0	thal_1	thal_2
0	0.180742	1.103147	-0.180742	-1.812403	69	0	0	1	1	0	...	0	1	0	0	1	0	0	1	0	0
1	-0.081421	1.384160	0.081421	-2.127313	69	0	1	0	1	0	...	1	0	0	0	0	1	0	1	0	0
2	0.217260	1.134581	-0.217260	-1.561859	66	0	1	0	1	0	...	0	0	1	1	0	0	0	1	0	0
3	-0.185280	1.296963	0.185280	-1.591354	65	1	0	1	1	0	...	0	1	0	0	1	0	0	1	0	0
4	-0.218509	1.079692	0.218509	-1.609254	64	0	0	1	1	0	...	0	1	0	1	0	0	0	1	0	0

5 rows × 29 columns

Checking Null Data

```
df_heart_mm.isnull().sum()
```

```
trestbps     0
chol         0
thalach      0
oldpeak      0
age          0
condition    0
sex_0        0
sex_1        0
cp_0         0
cp_1         0
cp_2         0
cp_3         0
fbs_0        0
fbs_1        0
restecg_0    0
restecg_1    0
restecg_2    0
exang_0      0
exang_1      0
slope_0      0
slope_1      0
slope_2      0
ca_0         0
ca_1         0
ca_2         0
ca_3         0
thal_0       0
thal_1       0
thal_2       0
dtype: int64
```

```
df_heart_robust.isnull().sum()
```

```
trestbps     0
chol         0
thalach      0
oldpeak      0
age          0
condition    0
sex_0        0
sex_1        0
cp_0         0
cp_1         0
cp_2         0
cp_3         0
fbs_0        0
fbs_1        0
restecg_0    0
restecg_1    0
restecg_2    0
exang_0      0
exang_1      0
slope_0      0
slope_1      0
slope_2      0
ca_0         0
ca_1         0
ca_2         0
ca_3         0
thal_0       0
thal_1       0
thal_2       0
dtype: int64
```

Data aman
dari null values

Machine Learning Modelling

FINAL PROJECT DATA SCIENCE BCC

Heart Disease Classifiaction

Algoritma yang akan dipakai

FINAL PROJECT DATA SCIENCE BCC

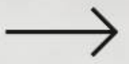
Algoritma :

- KNN
- Logistic Regression
- Decision Tree
- Random Forest
- SVM

Modelling #1

FINAL PROJECT DATA SCIENCE BCC

Splitting Data



```
[479] # Split data into features and target
      X = df_heart.drop(columns=['condition'])
      y = df_heart['condition']
```

```
# row and column data before split
print(df_heart_mm.shape)
print(df_heart_robust.shape)
```

```
(278, 29)
(297, 29)
```

Splitting Data

```
# split the MinMax data and show the rows and columns after the split
X_train_mm, X_test_mm = train_test_split(df_heart_mm, test_size=0.1)
y_train_mm = X_train_mm.pop('condition')
y_test_mm = X_test_mm.pop('condition')
print(X_train_mm.shape)
print(X_test_mm.shape)
```

```
(250, 28)
(28, 28)
```

```
X_test_mm.head()
```

	trestbps	chol	thalach	oldpeak	age	sex_0	sex_1	cp_0	cp_1	cp_2	...	slope_0	slope_1	slope_2	ca_0	ca_1	ca_2	ca_3	thal_0	thal_1	thal_2
196	0.457527	1.0	0.592094	0.0	59	0	1	0	0	0	...	0	1	0	0	1	0	0	0	0	1
29	0.633484	1.0	0.742081	0.0	59	0	1	0	1	0	...	1	0	0	1	0	0	0	1	0	0
15	0.508086	1.0	0.595687	0.0	52	0	1	1	0	0	...	0	1	0	1	0	0	0	0	0	1
227	0.635584	1.0	0.598071	0.0	54	0	1	0	0	0	...	0	1	0	0	1	0	0	0	0	1
26	0.717949	1.0	0.917949	0.0	63	1	0	0	1	0	...	1	0	0	0	0	1	0	1	0	0

5 rows × 28 columns

Splitting Data

```
# split the Robust data and show the rows and columns after the split
X_train_robust, X_test_robust = train_test_split(df_heart_robust, test_size=0.1)
y_train_robust = X_train_robust.pop('condition')
y_test_robust = X_test_robust.pop('condition')
print(X_train_robust.shape)
print(X_test_robust.shape)
```

```
(267, 28)
(30, 28)
```

```
X_test_robust.head()
```

	trestbps	chol	thalach	oldpeak	age	sex_0	sex_1	cp_0	cp_1	cp_2	...	slope_0	slope_1	slope_2	ca_0	ca_1	ca_2	ca_3	thal_0	thal_1	thal_2
208	-0.191732	1.354104	0.191732	-1.495506	59	0	1	0	0	0	...	0	1	0	0	1	0	0	0	0	1
215	-0.140575	1.750799	0.140575	-1.405751	58	1	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
3	-0.185280	1.296963	0.185280	-1.591354	65	0	1	1	0	0	...	0	1	0	0	1	0	0	1	0	0
23	-0.007358	2.185430	0.007358	-1.770419	74	1	0	0	1	0	...	1	0	0	0	1	0	0	1	0	0
158	0.109775	2.117094	-0.109775	-1.224255	70	0	1	0	0	0	...	0	1	0	0	0	0	1	1	0	0

5 rows × 28 columns

Function Confusion Matrix

```
# buat fungsi visualisasi confusion matrix

def plotting_confusion_matrix(model, X_test, y_test, title):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(1, 1, figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', annot_kws={"fontsize":12}, ax=ax)

    # setting title and axis labels
    plt.tick_params(axis='both', which='major', labelsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=12)
    ax.set_ylabel('True Labels', fontsize=12)
    ax.set_title('Confusion Matrix ' + title, fontsize=14)
    ax.xaxis.set_ticklabels(['No Disease', 'Disease'])
    ax.yaxis.set_ticklabels(['No Disease', 'Disease'])
    plt.show()
```

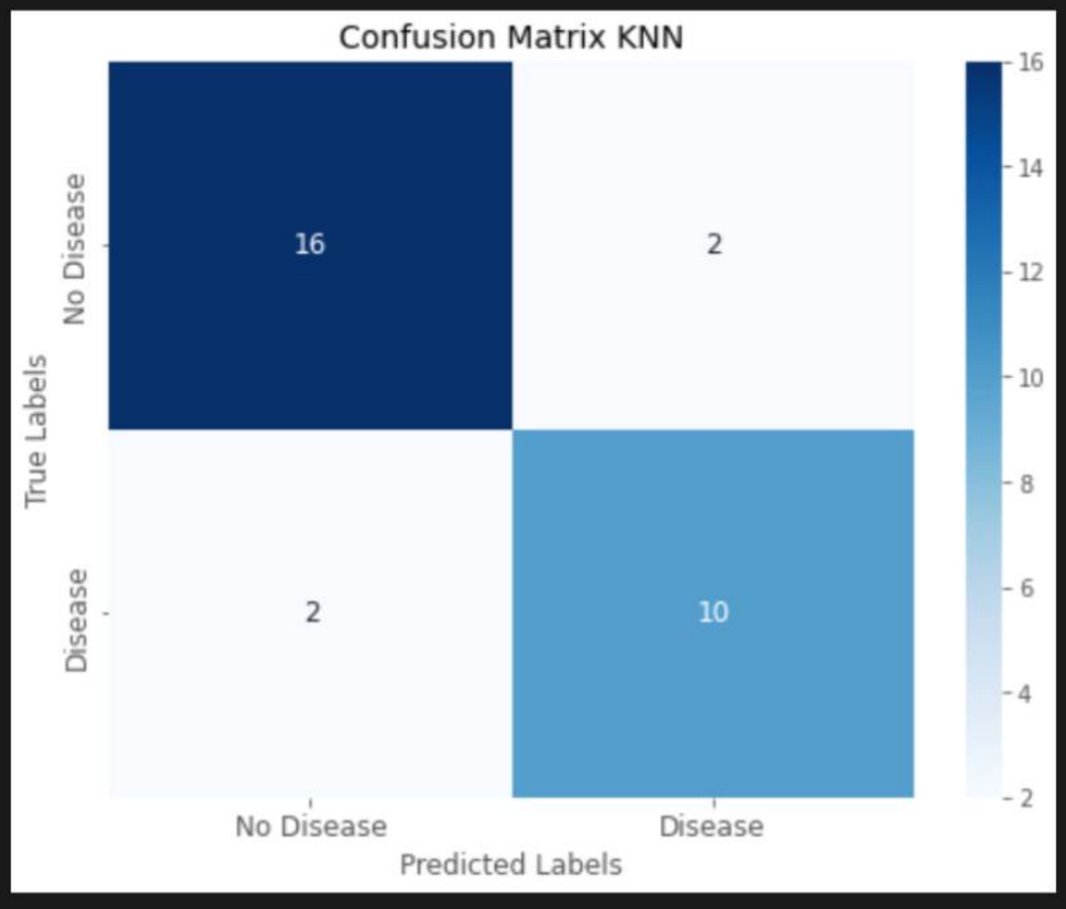
Base Model Scores (Robust)

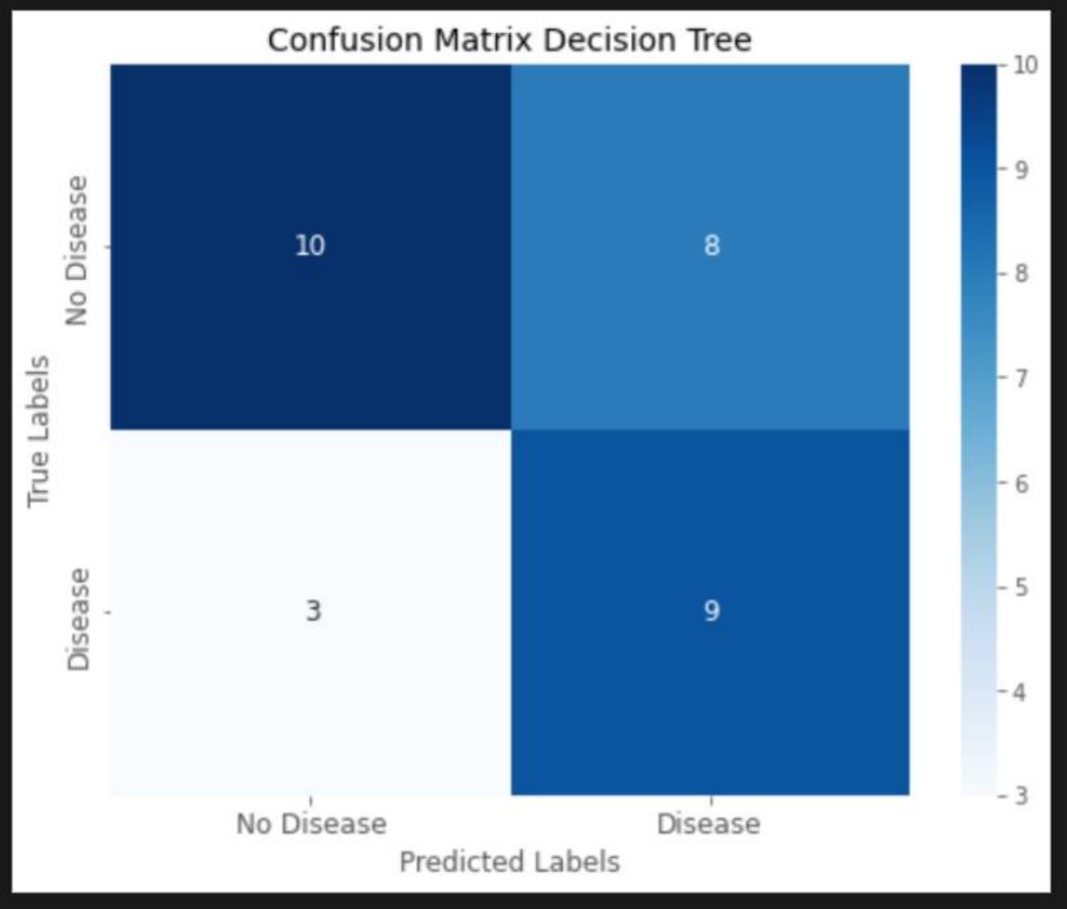
```python
# Recall score for base models
models = [("K-Nearest Neighbor (KNN)", knn),
          ("Decision Tree", dt),
          ("Random Forest", rf),
          ("Support Vector Machine (SVM)", svm),
          ("Logistic Regression", lr)]

print("Recall Scores for Base Models:\n")
for i, (name, model) in enumerate(models):
    y_pred = model.predict(X_test_robust)
    score = recall_score(y_test_robust, y_pred)
    print(f"Model {i+1}: {name:<30} - Recall score = {score:.3f}")
```

```
Recall Scores for Base Models:

Model 1: K-Nearest Neighbor (KNN)       - Recall score = 0.583
Model 2: Decision Tree                  - Recall score = 0.750
Model 3: Random Forest                  - Recall score = 0.750
Model 4: Support Vector Machine (SVM)   - Recall score = 0.667
Model 5: Logistic Regression            - Recall score = 0.750
```

Confusion Matrix

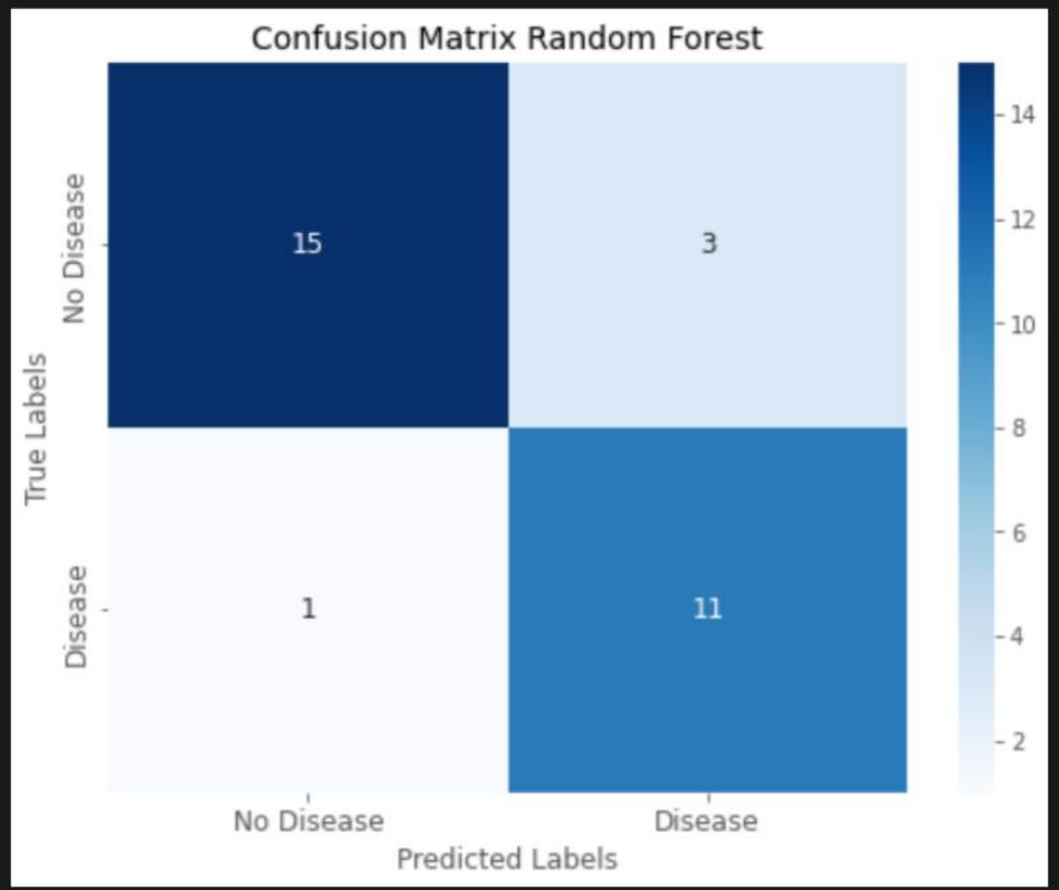
Robust - Base Model

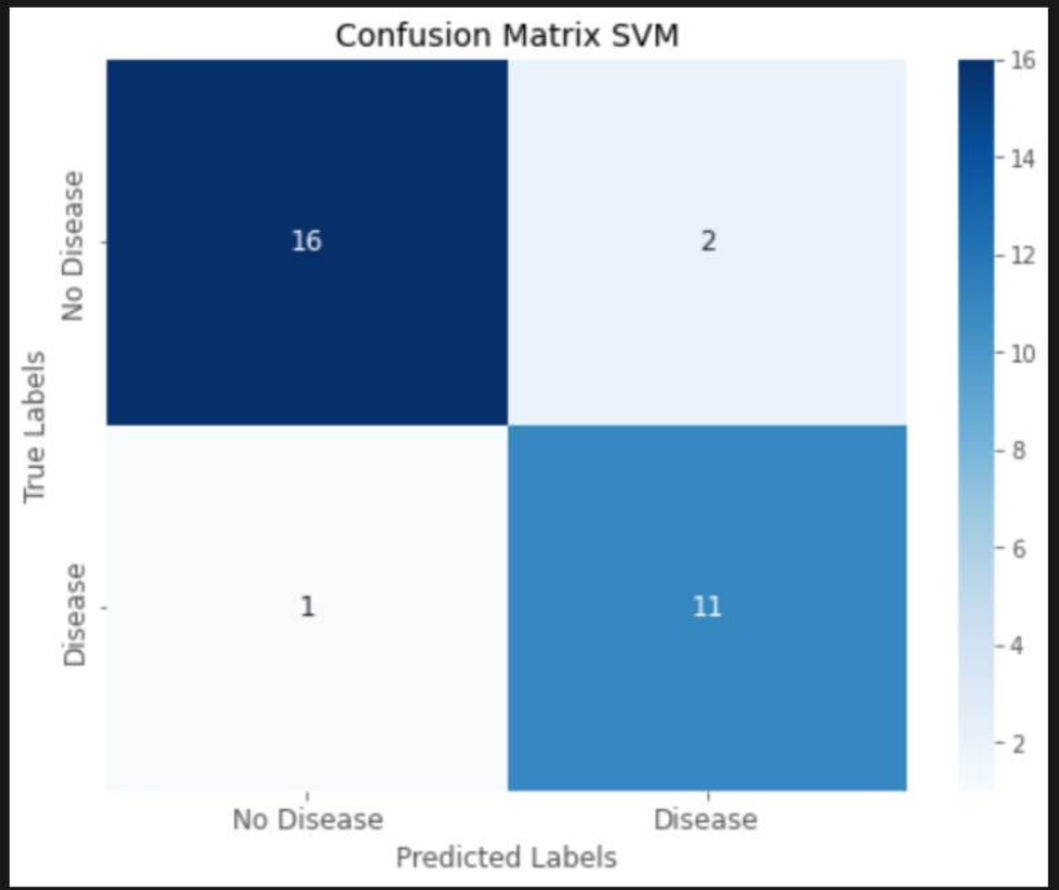




Confusion Matrix

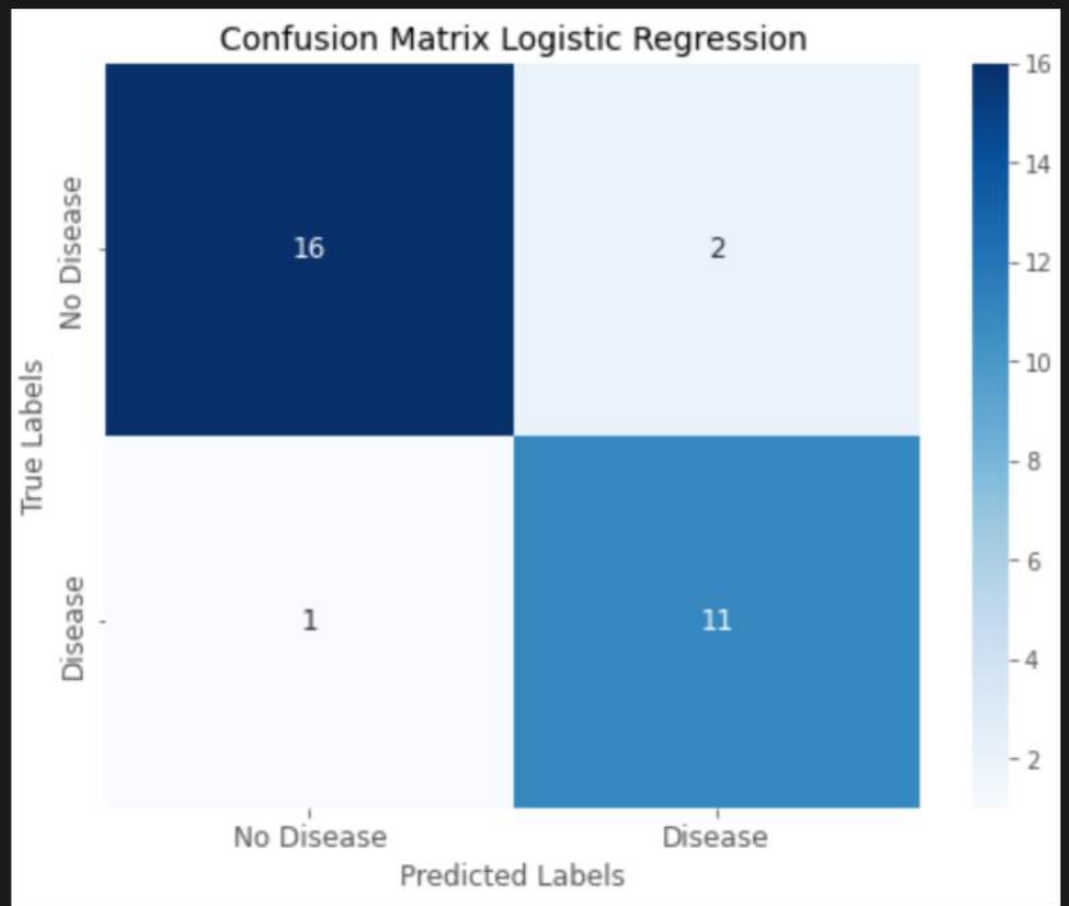
Robust - Base Model





Confusion Matrix

Robust - Base Model



Hyperparameter Tuning with Optuna

```python
def objective_lr(trial):
    C = trial.suggest_loguniform('C', 0.1, 10)
    lr = LogisticRegression(penalty='l2', C=C, solver='lbfgs', max_iter=1000, random_state=42)
    lr.fit(X_train_robust, y_train_robust)
    y_pred = lr.predict(X_test_robust)
    return recall_score(y_test_robust, y_pred)

study_knn = optuna.create_study(direction='maximize')
study_knn.optimize(objective_knn, n_trials=100)

study_dt = optuna.create_study(direction='maximize')
study_dt.optimize(objective_dt, n_trials=100)

study_rf = optuna.create_study(direction='maximize')
study_rf.optimize(objective_rf, n_trials=100)

study_svm = optuna.create_study(direction='maximize')
study_svm.optimize(objective_svm, n_trials=100)

study_lr = optuna.create_study(direction='maximize')
study_lr.optimize(objective_lr, n_trials=100)
```

```
[I 2023-03-17 15:11:05,806] A new study created in memory with name: no-name-95a8d707-11d8-49a0-b752-2140fab8fb24
[I 2023-03-17 15:11:05,825] Trial 0 finished with value: 0.8 and parameters: {'n_neighbors': 1}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:05,852] Trial 1 finished with value: 0.5 and parameters: {'n_neighbors': 19}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:05,876] Trial 2 finished with value: 0.6 and parameters: {'n_neighbors': 7}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:05,897] Trial 3 finished with value: 0.7 and parameters: {'n_neighbors': 11}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:05,912] Trial 4 finished with value: 0.7 and parameters: {'n_neighbors': 3}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:05,929] Trial 5 finished with value: 0.5 and parameters: {'n_neighbors': 16}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:05,950] Trial 6 finished with value: 0.5 and parameters: {'n_neighbors': 6}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:05,973] Trial 7 finished with value: 0.5 and parameters: {'n_neighbors': 13}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:05,992] Trial 8 finished with value: 0.4 and parameters: {'n_neighbors': 8}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:06,012] Trial 9 finished with value: 0.5 and parameters: {'n_neighbors': 12}. Best is trial 0 with value: 0.8.
[I 2023-03-17 15:11:06,039] Trial 10 finished with value: 0.8 and parameters: {'n_neighbors': 1}. Best is trial 0 with value: 0.8.
```

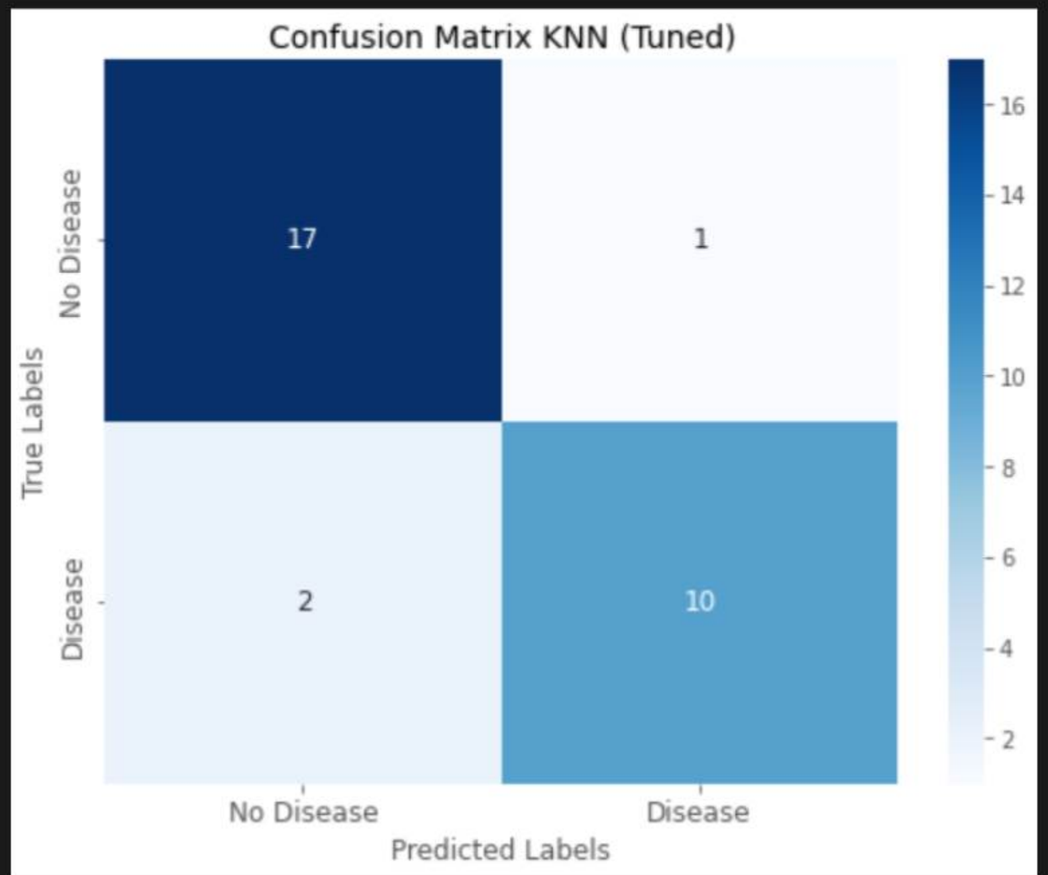
Tuned Model Scores (Robust)

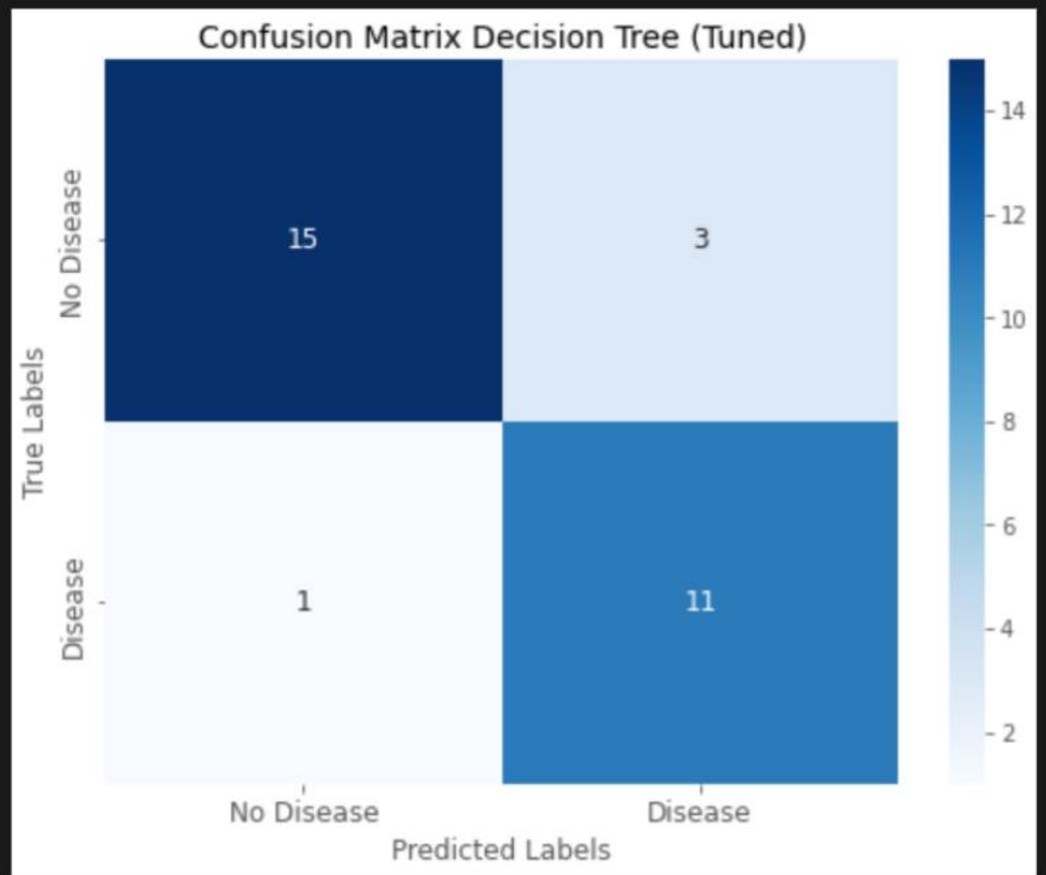
```
# Recall score for tuned models
models_best = [("K-Nearest Neighbor (KNN)", knn_best),
               ("Decision Tree", dt_best),
               ("Random Forest", rf_best),
               ("Support Vector Machine (SVM)", svm_best),
               ("Logistic Regression", lr_best)]

print("Recall score for tuned models:")
for name, model in models_best:
    y_pred = model.predict(X_test_robust)
    score = recall_score(y_test_robust, y_pred)
    model_number = models_best.index((name, model)) + 1
    print(f"Tuned Model {model_number}: {name:<30} - Recall score = {score:.3f}")
```

```
Recall score for tuned models:
Tuned Model 1: K-Nearest Neighbor (KNN)       - Recall score = 0.750
Tuned Model 2: Decision Tree                  - Recall score = 0.750
Tuned Model 3: Random Forest                  - Recall score = 0.750
Tuned Model 4: Support Vector Machine (SVM)   - Recall score = 0.667
Tuned Model 5: Logistic Regression            - Recall score = 0.833
```

Confusion Matrix

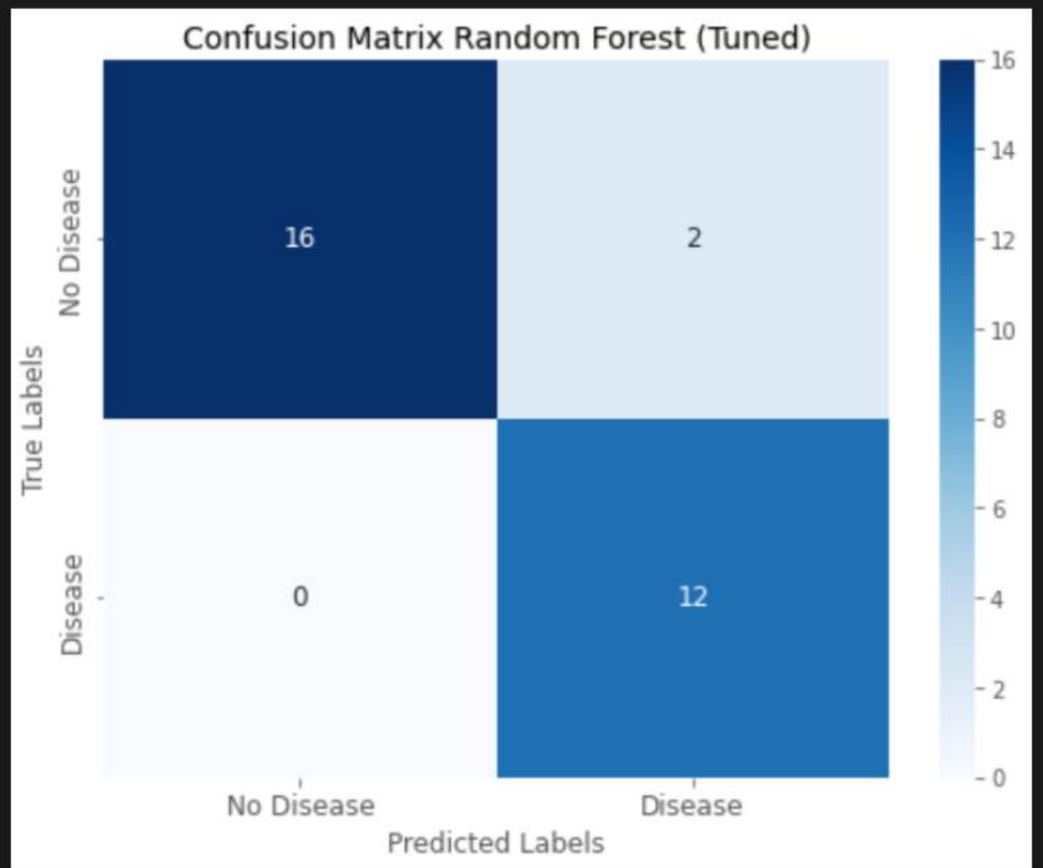
Robust - Tuned Model

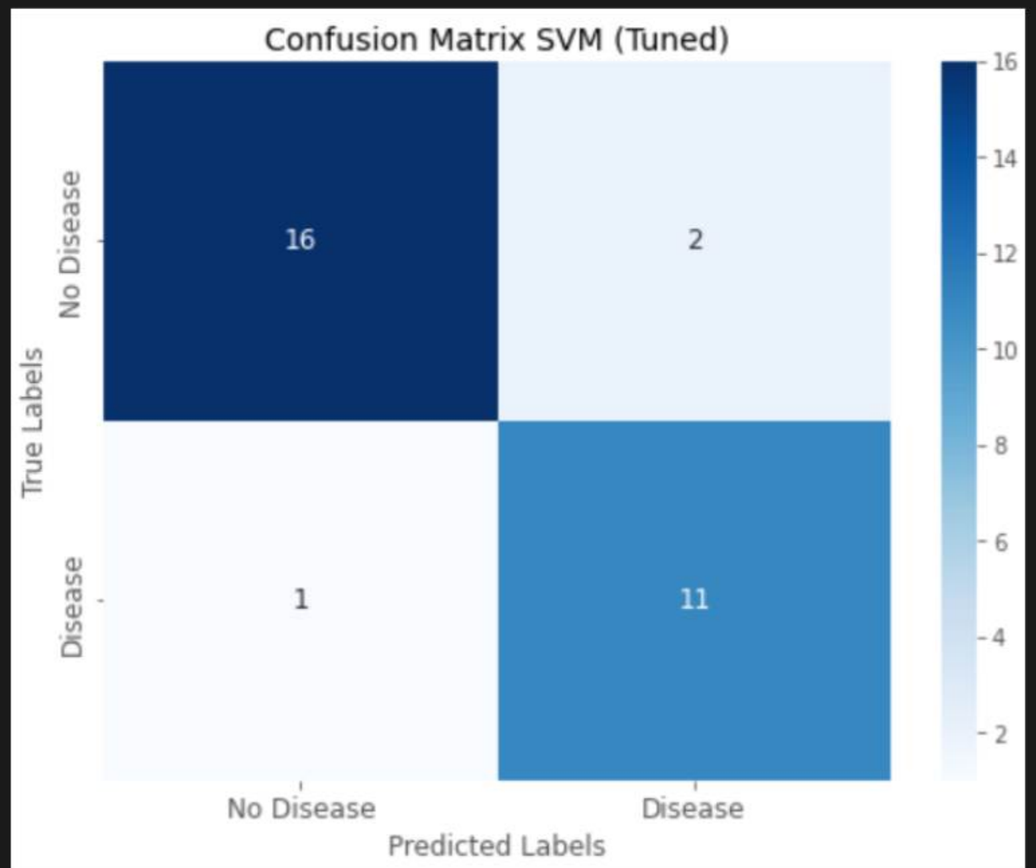




Confusion Matrix

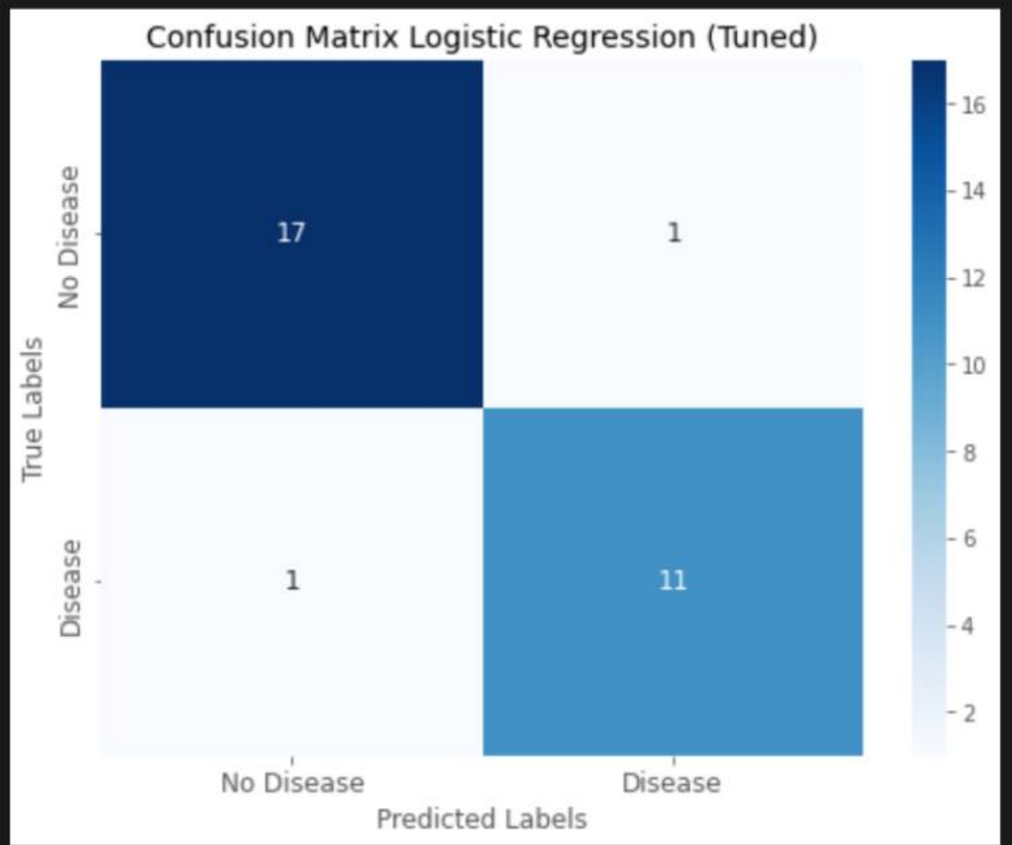
Robust - Tuned Model





Confusion Matrix

Robust - Tuned Model



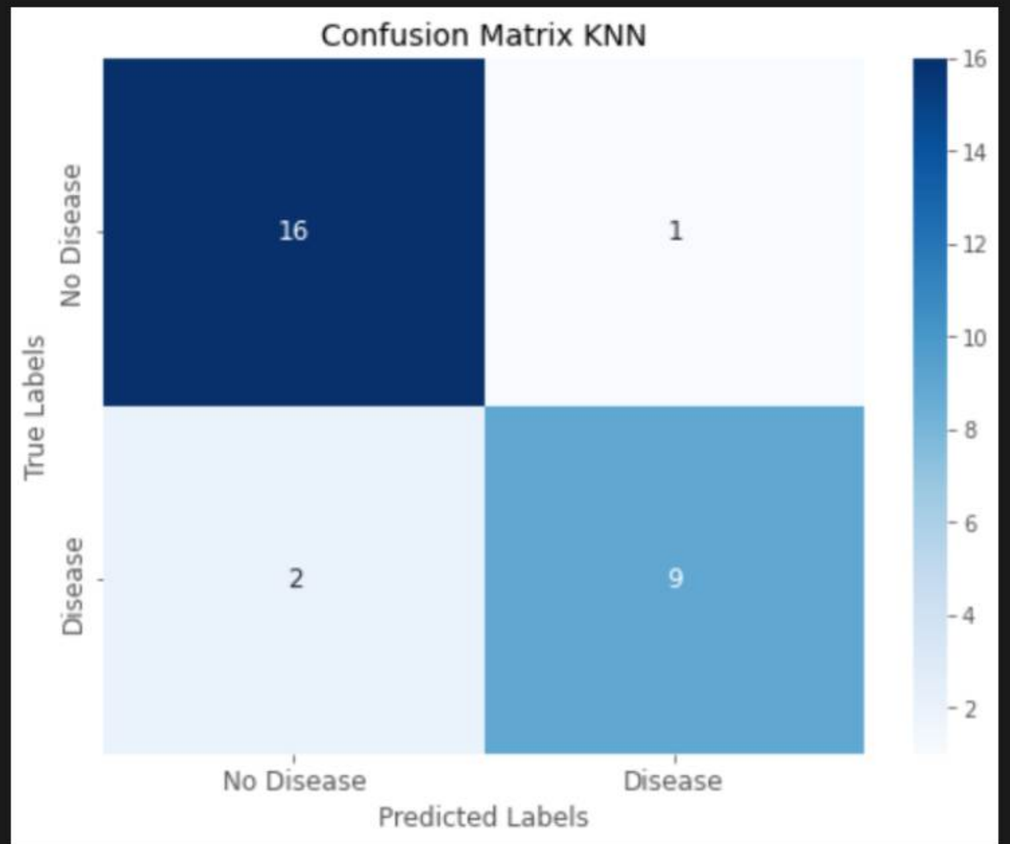
Base Model Scores (MinMax)

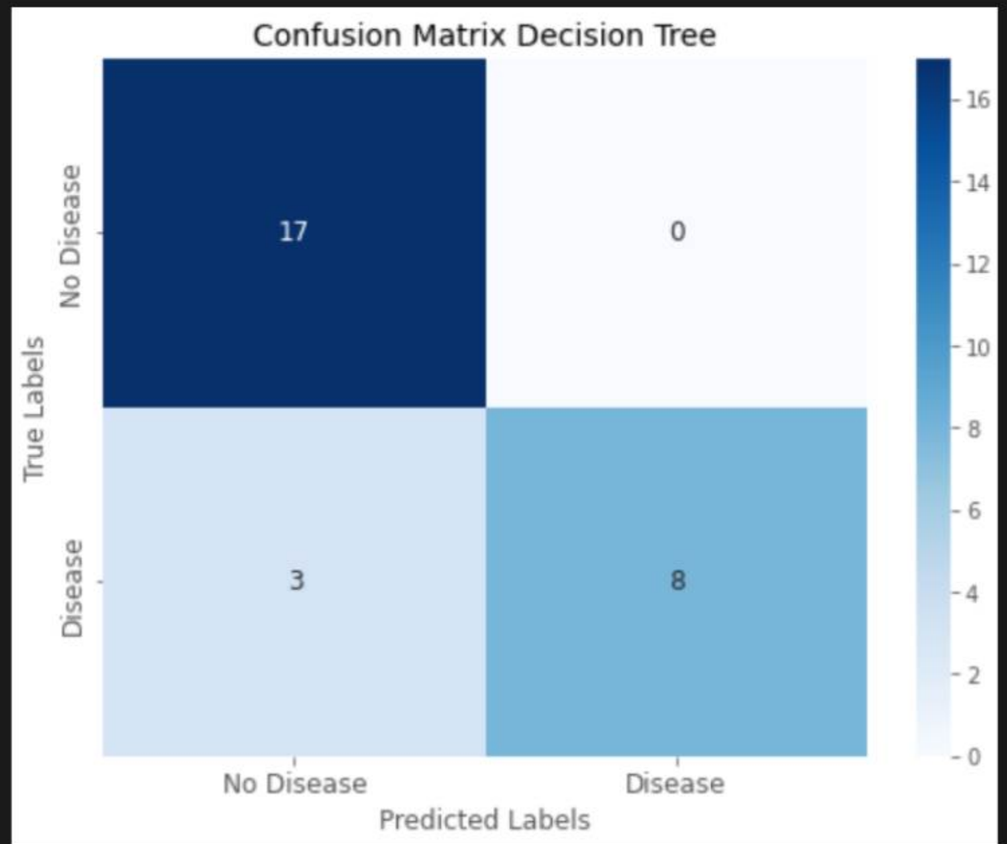
```python
# Recall score for base models
models = [("K-Nearest Neighbor (KNN)", knn),
          ("Decision Tree", dt),
          ("Random Forest", rf),
          ("Support Vector Machine (SVM)", svm),
          ("Logistic Regression", lr)]

print("Recall Scores for Base Models:\n")
for i, (name, model) in enumerate(models):
    y_pred = model.predict(X_test_mm)
    score = recall_score(y_test_mm, y_pred)
    print(f"Model {i+1}: {name:<30} - Recall score = {score:.3f}")
```

```
Recall Scores for Base Models:

Model 1: K-Nearest Neighbor (KNN)       - Recall score = 0.733
Model 2: Decision Tree                  - Recall score = 0.667
Model 3: Random Forest                  - Recall score = 0.667
Model 4: Support Vector Machine (SVM)   - Recall score = 0.600
Model 5: Logistic Regression            - Recall score = 0.600
```

Confusion Matrix

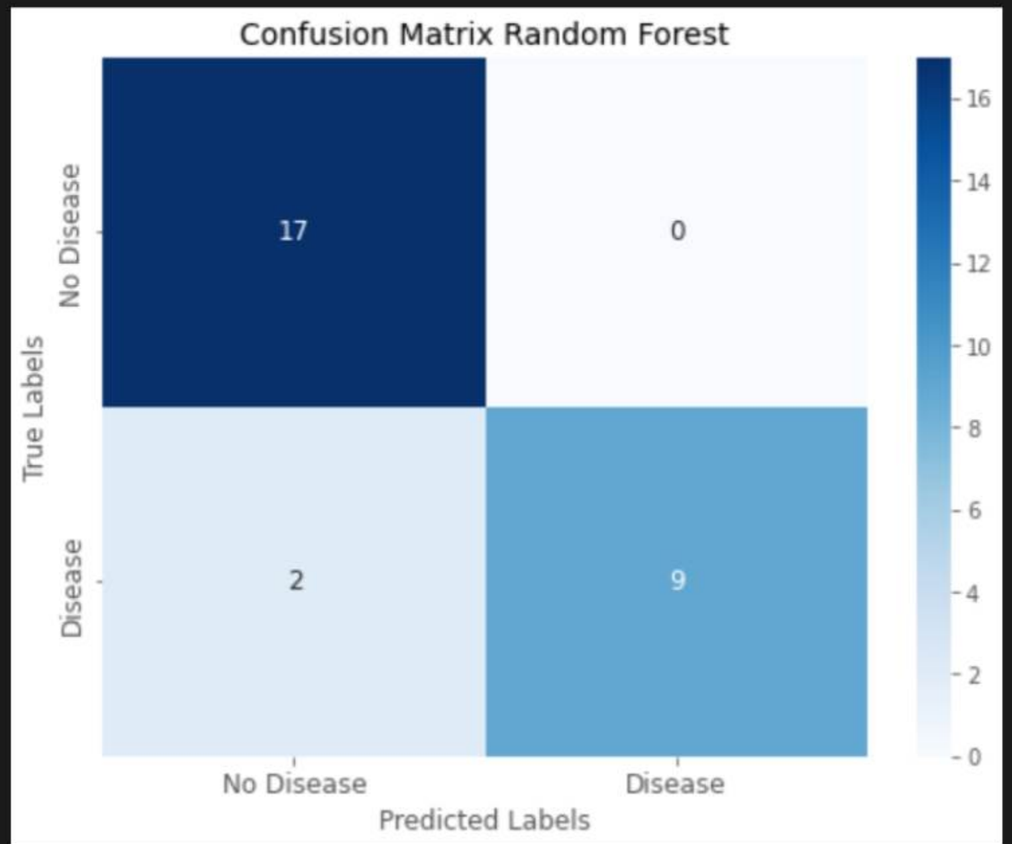
MinMax - Base Model

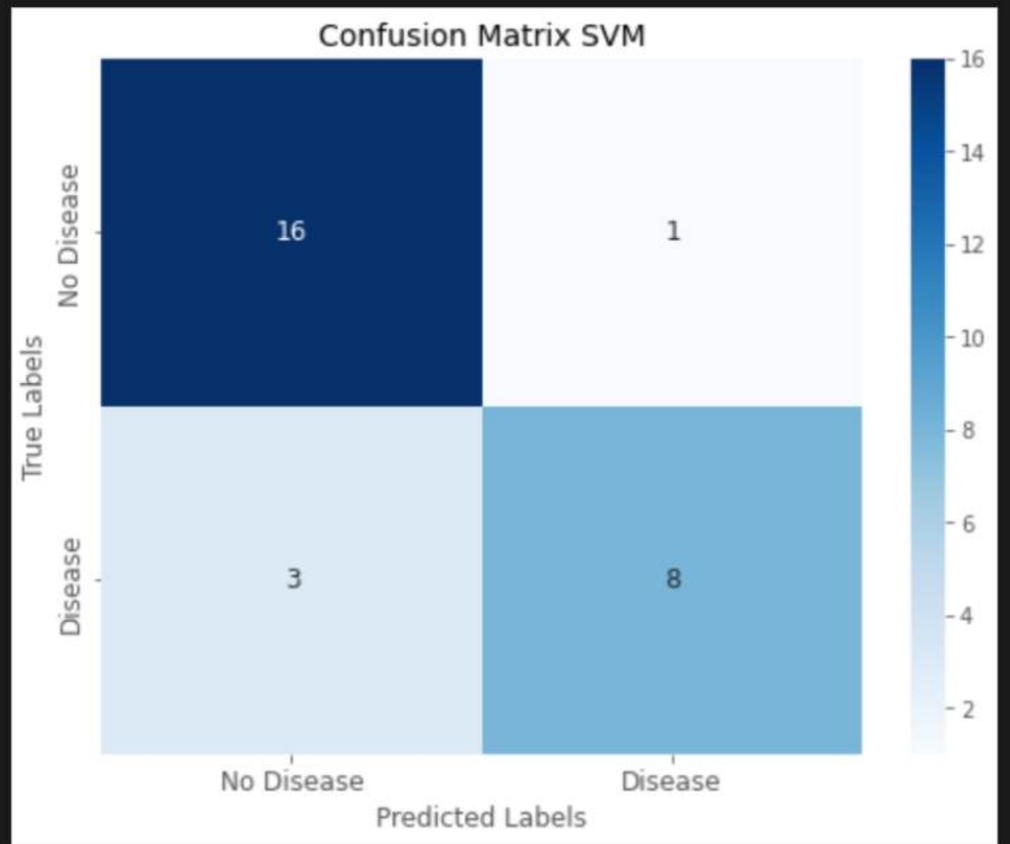




Confusion Matrix

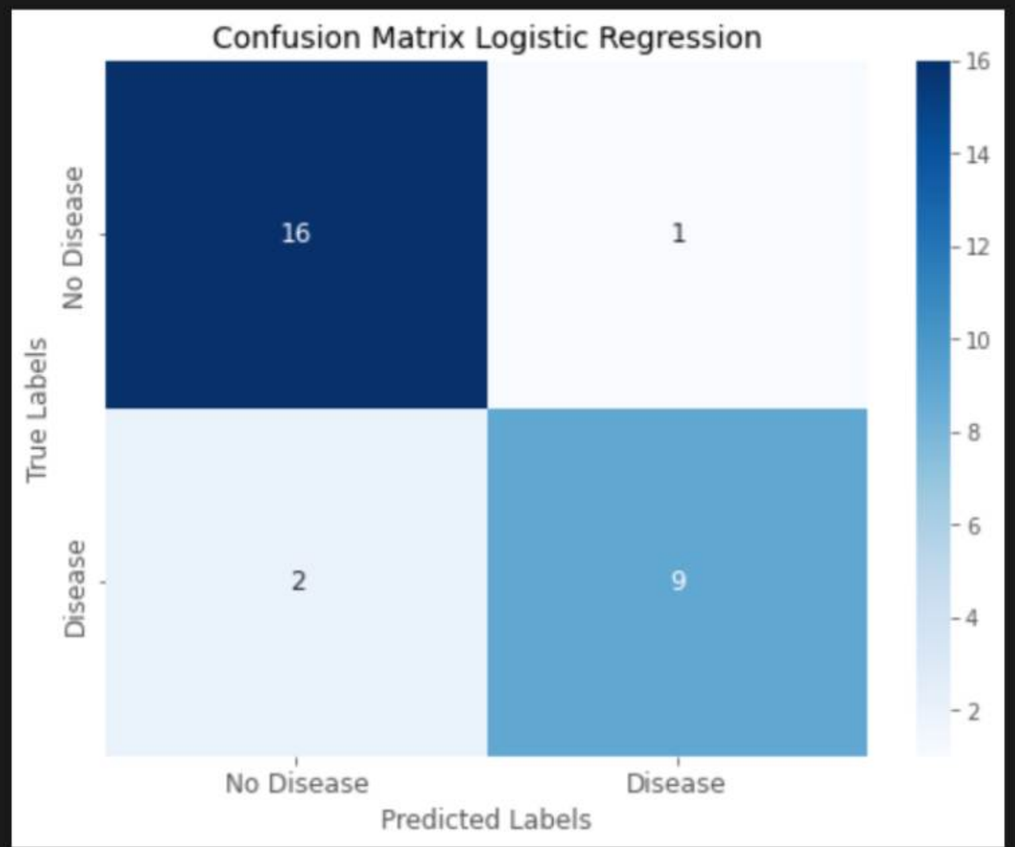
MinMax - Base Model





Confusion Matrix

MinMax - Base Model



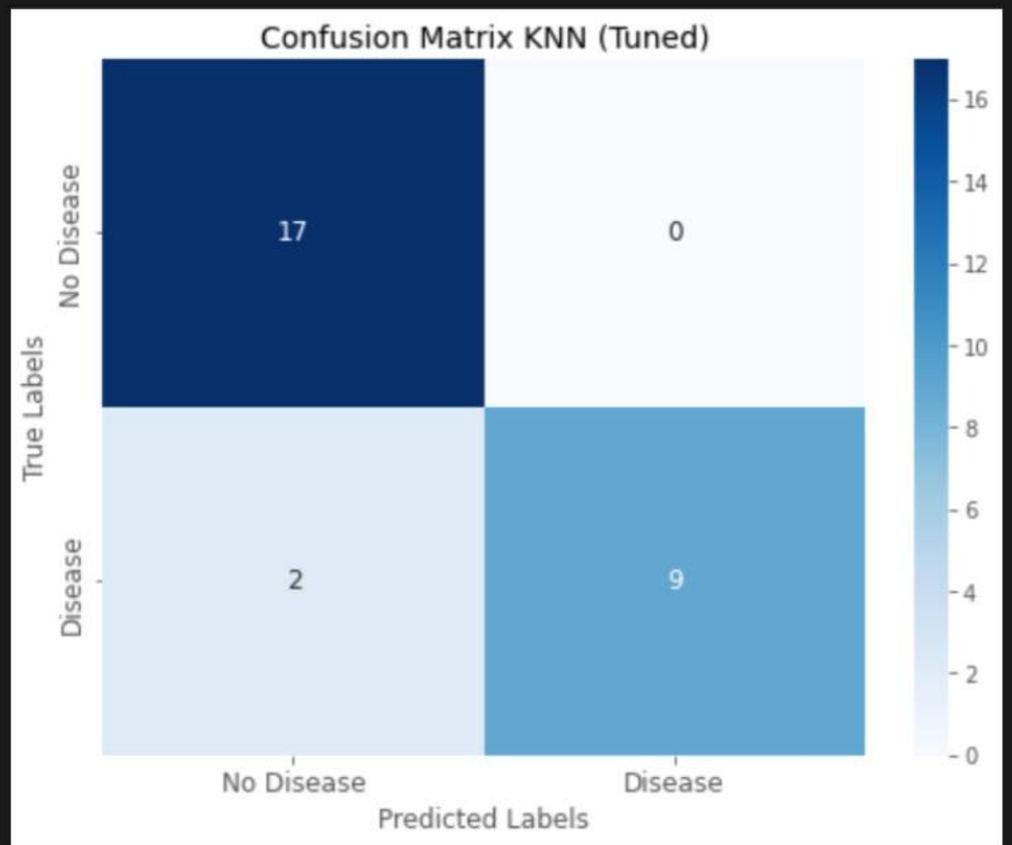
Tuned Model Scores (MinMax)

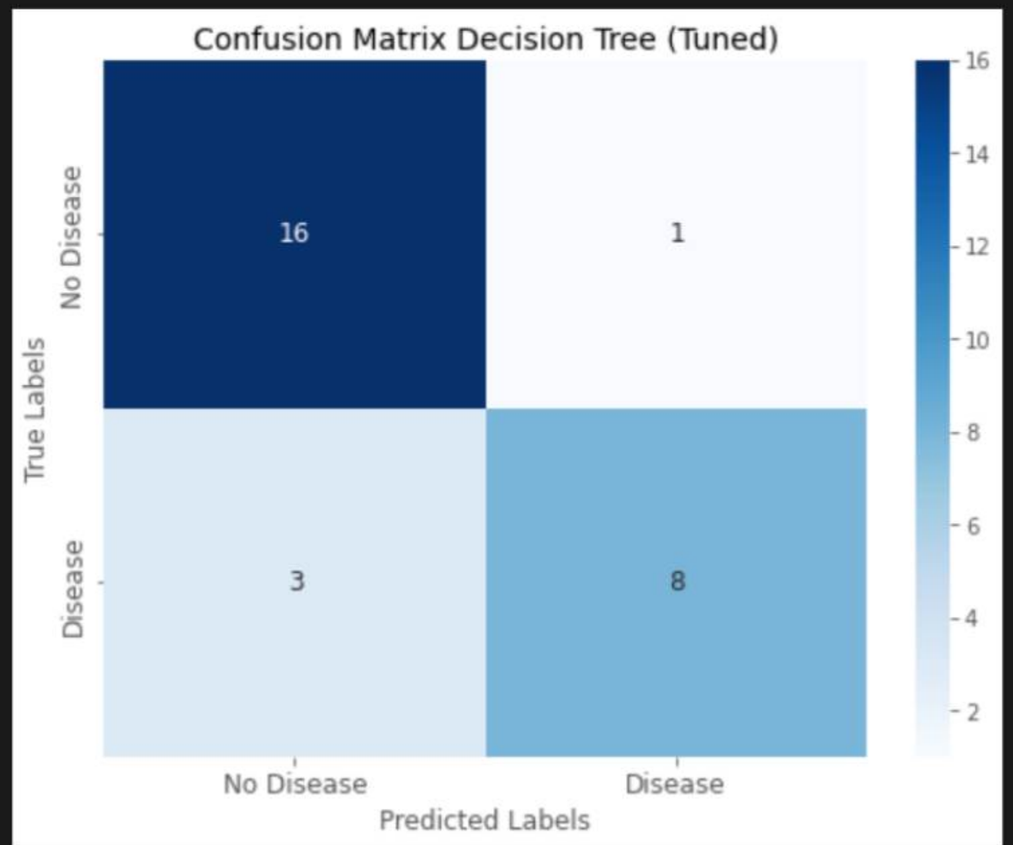
```
# Recall score for tuned models
models_best = [("K-Nearest Neighbor (KNN)", knn_best),
               ("Decision Tree", dt_best),
               ("Random Forest", rf_best),
               ("Support Vector Machine (SVM)", svm_best),
               ("Logistic Regression", lr_best)]

print("Recall score for tuned models:")
for name, model in models_best:
    y_pred = model.predict(X_test_mm)
    score = recall_score(y_test_mm, y_pred)
    model_number = models_best.index((name, model)) + 1
    print(f"Tuned Model {model_number}: {name:<30} - Recall score = {score:.3f}")
```

```
Recall score for tuned models:
Tuned Model 1: K-Nearest Neighbor (KNN)       - Recall score = 0.733
Tuned Model 2: Decision Tree                  - Recall score = 0.733
Tuned Model 3: Random Forest                  - Recall score = 0.800
Tuned Model 4: Support Vector Machine (SVM)   - Recall score = 0.600
Tuned Model 5: Logistic Regression            - Recall score = 0.600
```

Confusion Matrix

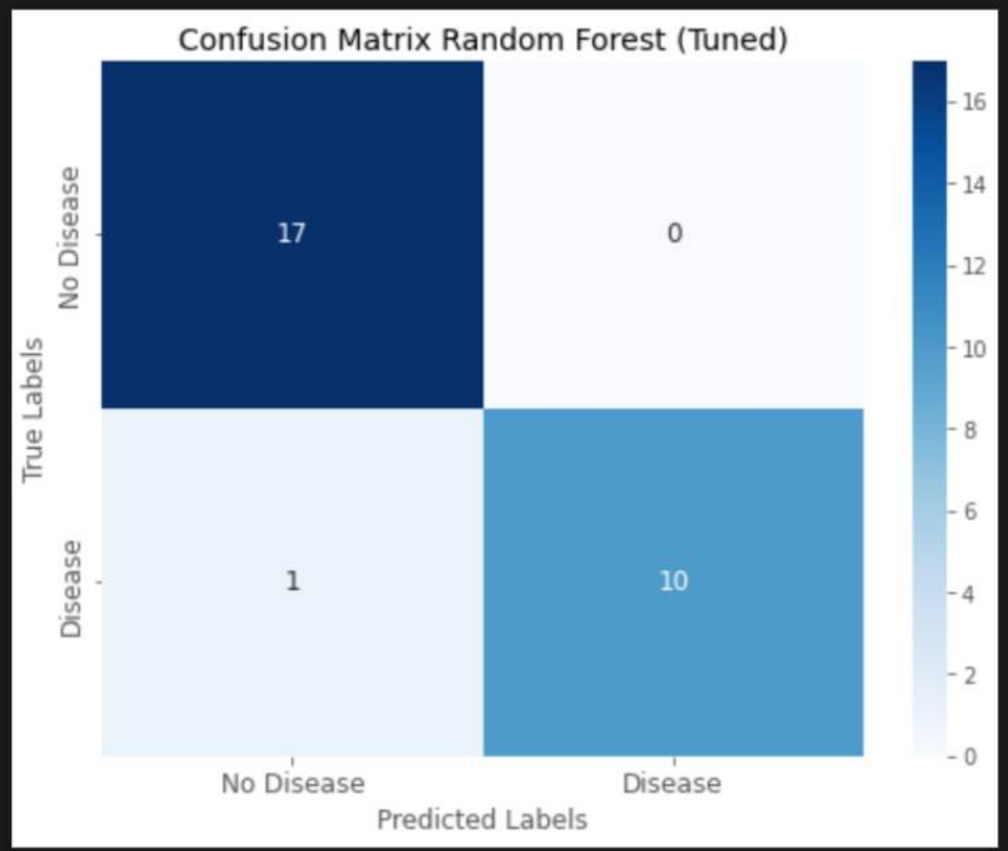
MinMax - Tuned Model

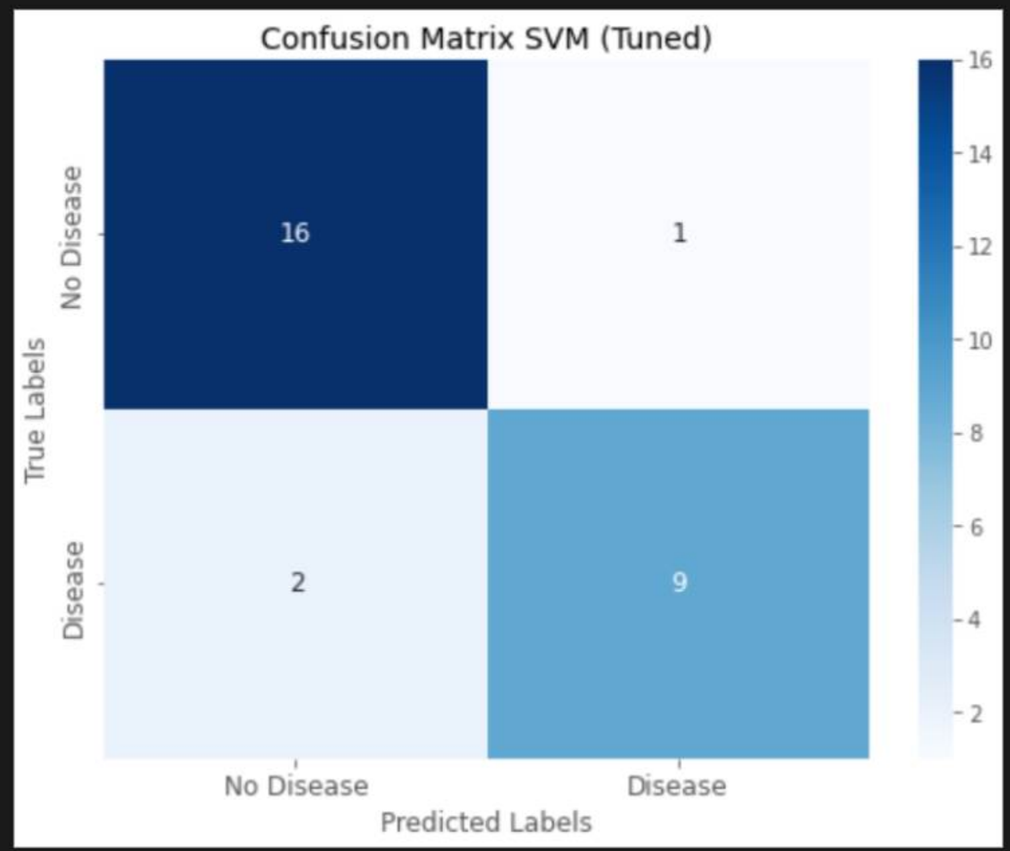




Confusion Matrix

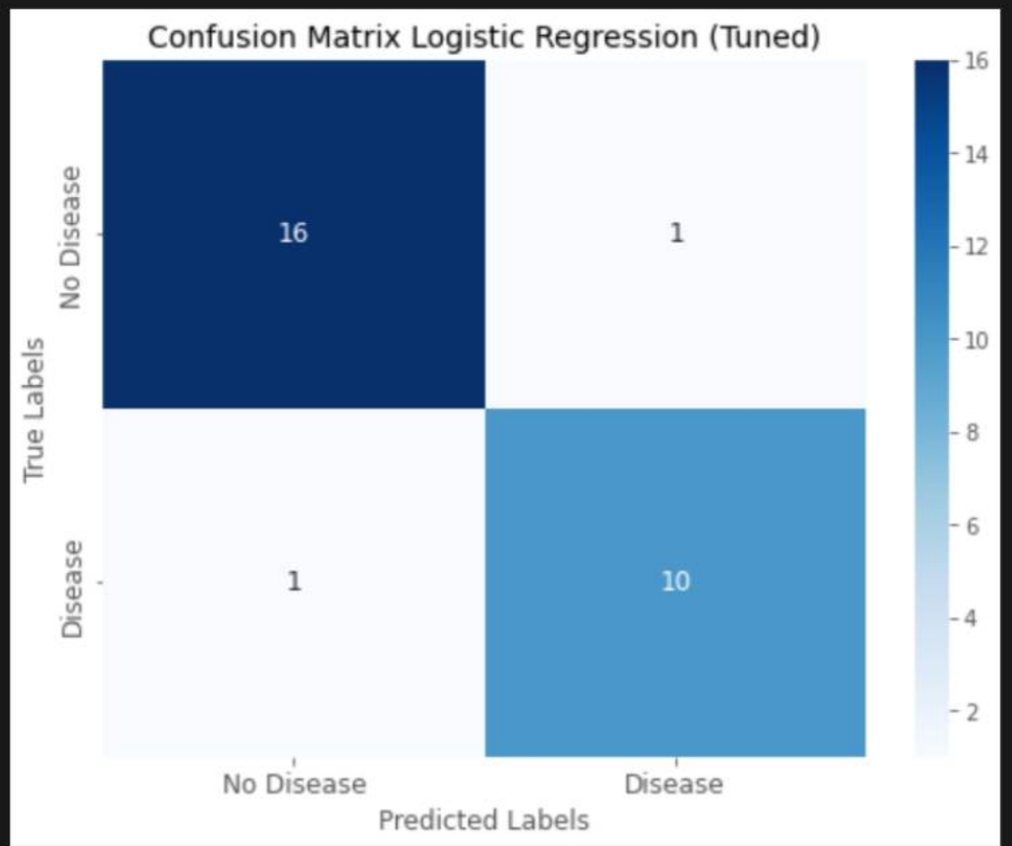
MinMax - Tuned Model





Confusion Matrix

MinMax - Tuned Model



All Score Model

Robust

```
Base Models
K-Nearest Neighbor (KNN)        - Recall score = 0.583
Decision Tree                   - Recall score = 0.750
Random Forest                   - Recall score = 0.750
Support Vector Machine (SVM)    - Recall score = 0.667
Logistic Regression             - Recall score = 0.750

Tuned Models
K-Nearest Neighbor (KNN)        - Recall score = 0.750
Decision Tree                   - Recall score = 0.750
Random Forest                   - Recall score = 0.750
Support Vector Machine (SVM)    - Recall score = 0.667
Logistic Regression             - Recall score = 0.833
```

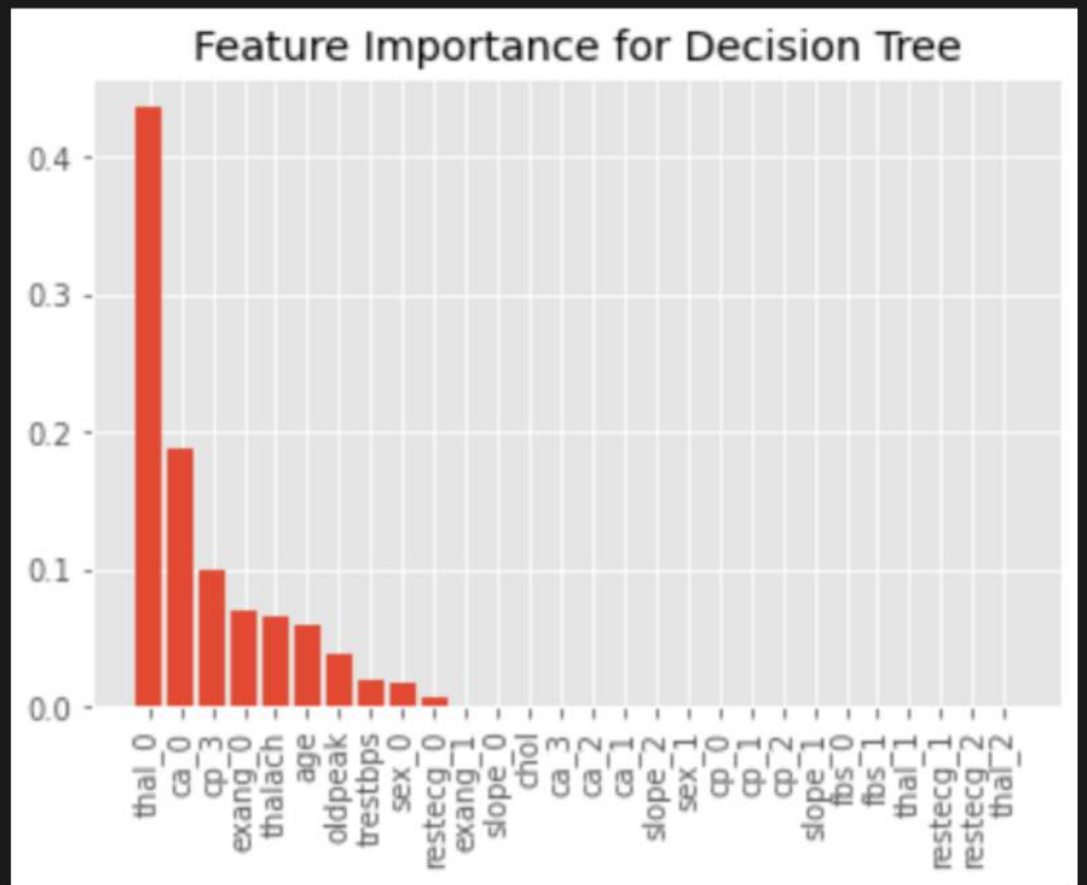
Cross Validation

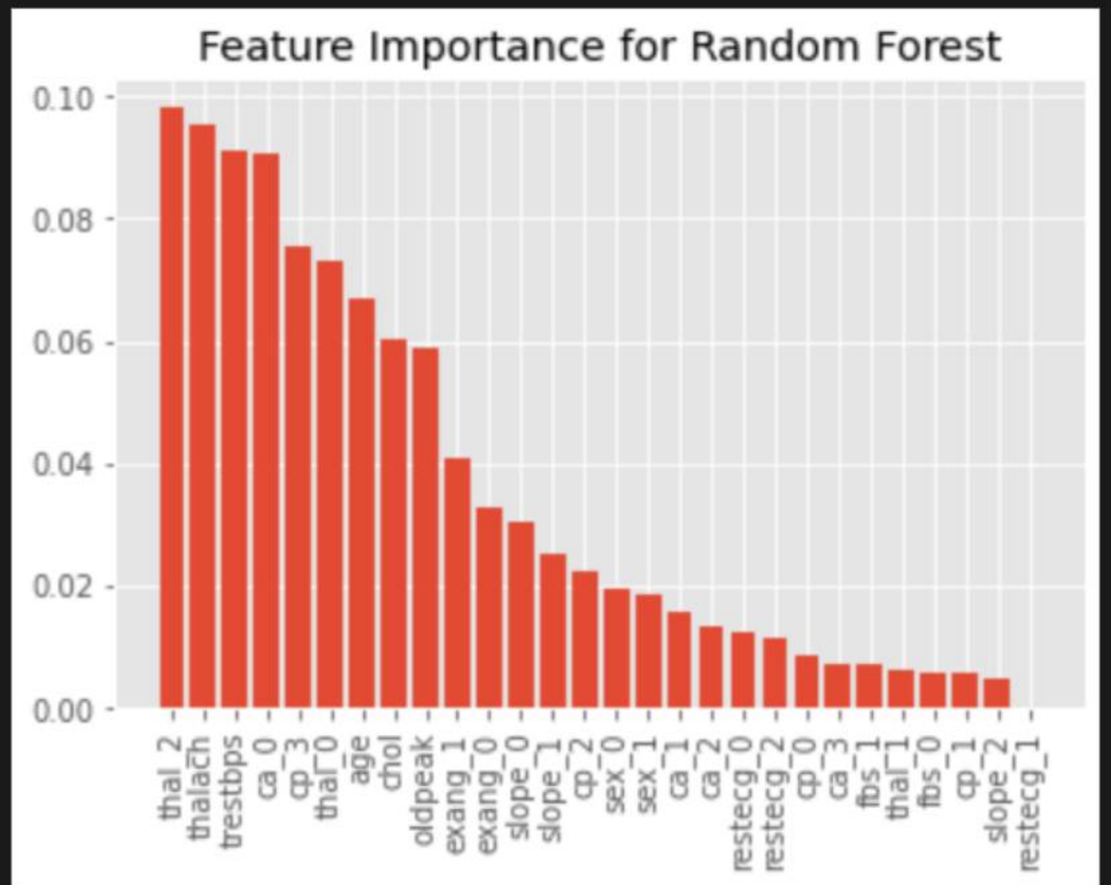
Robust

```
Base Models
K-Nearest Neighbor (KNN)       : Mean Recall score = 0.696
Decision Tree                  : Mean Recall score = 0.704
Random Forest                  : Mean Recall score = 0.784
Support Vector Machine (SVM)   : Mean Recall score = 0.752
Logistic Regression            : Mean Recall score = 0.784

Tuned Models
K-Nearest Neighbor (KNN)       : Mean Recall score = 0.688
Decision Tree                  : Mean Recall score = 0.720
Random Forest                  : Mean Recall score = 0.760
Support Vector Machine (SVM)   : Mean Recall score = 0.768
Logistic Regression            : Mean Recall score = 0.784
```

Feature Importance

Robust





All Score Model

MinMax

```
Base Models
K-Nearest Neighbor (KNN)      : Recall score = 0.733
Decision Tree                 : Recall score = 0.667
Random Forest                 : Recall score = 0.667
Support Vector Machine (SVM)  : Recall score = 0.600
Logistic Regression           : Recall score = 0.600

Tuned Models
K-Nearest Neighbor (KNN)      : Recall score = 0.733
Decision Tree                 : Recall score = 0.733
Random Forest                 : Recall score = 0.800
Support Vector Machine (SVM)  : Recall score = 0.600
Logistic Regression           : Recall score = 0.600
```

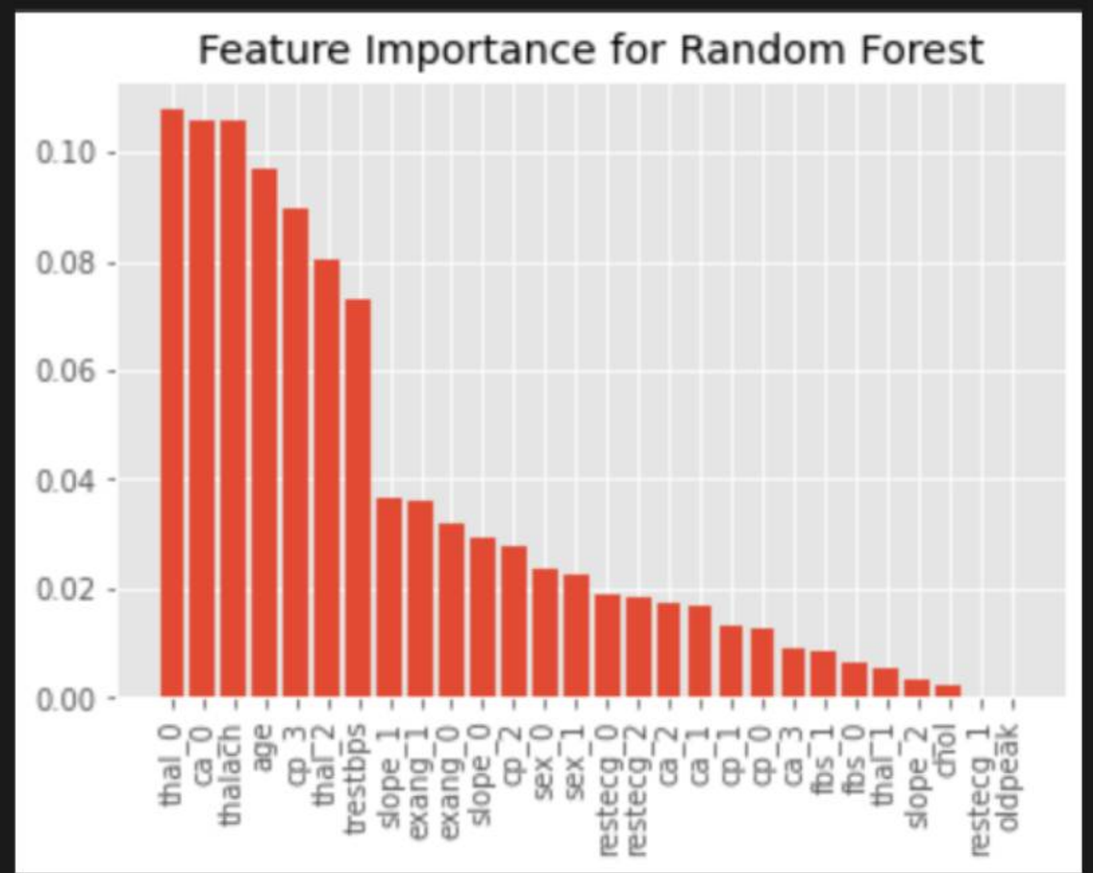
Cross Validation

MinMax

```
Base Models
K-Nearest Neighbor (KNN)       : Mean Recall score = 0.715
Decision Tree                  : Mean Recall score = 0.687
Random Forest                  : Mean Recall score = 0.788
Support Vector Machine (SVM)   : Mean Recall score = 0.797
Logistic Regression            : Mean Recall score = 0.806

Tuned Models
K-Nearest Neighbor (KNN)       : Mean Recall score = 0.715
Decision Tree                  : Mean Recall score = 0.687
Random Forest                  : Mean Recall score = 0.770
Support Vector Machine (SVM)   : Mean Recall score = 0.816
Logistic Regression            : Mean Recall score = 0.797
```

Feature Importance

MinMax



Modelling #2

MEMILIH 10 FITUR TERATAS BERDASARKAN
GABUNGAN FEATURE IMPORTANCE DARI
DECISION TREE DAN RANDOM FOREST

FINAL PROJECT DATA SCIENCE BCC

Feature Selection

Robust

```
# memilih 10 fitur teratas
feature_dt = pd.Series(dt_best.feature_importances_, index=X_train_robust.columns).nlargest(10).index.tolist()
feature_rf = pd.Series(rf_best.feature_importances_, index=X_train_robust.columns).nlargest(10).index.tolist()

selected_cols = [col for col in feature_dt]
for col in feature_rf:
    if col not in selected_cols:
        selected_cols.append(col)

selected_cols
```

```
['cp_3',
 'thal_0',
 'ca_0',
 'age',
 'sex_0',
 'trestbps',
 'chol',
 'thalach',
 'oldpeak',
 'sex_1',
 'thal_2',
 'exang_1',
 'slope_1',
 'slope_0']
```

Feature Selection

MinMax

```
# memilih 10 fitur teatas
feature_dt = pd.Series(dt_best.feature_importances_, index=X_train_mm.columns).nlargest(10).index.tolist()
feature_rf = pd.Series(rf_best.feature_importances_, index=X_train_mm.columns).nlargest(10).index.tolist()

selected_cols = [col for col in feature_dt]
for col in feature_rf:
    if col not in selected_cols:
        selected_cols.append(col)

selected_cols
```

```
['cp_3',
 'thal_0',
 'ca_0',
 'age',
 'sex_0',
 'trestbps',
 'chol',
 'thalach',
 'oldpeak',
 'sex_1',
 'thal_2',
 'exang_1',
 'slope_1',
 'slope_0']
```

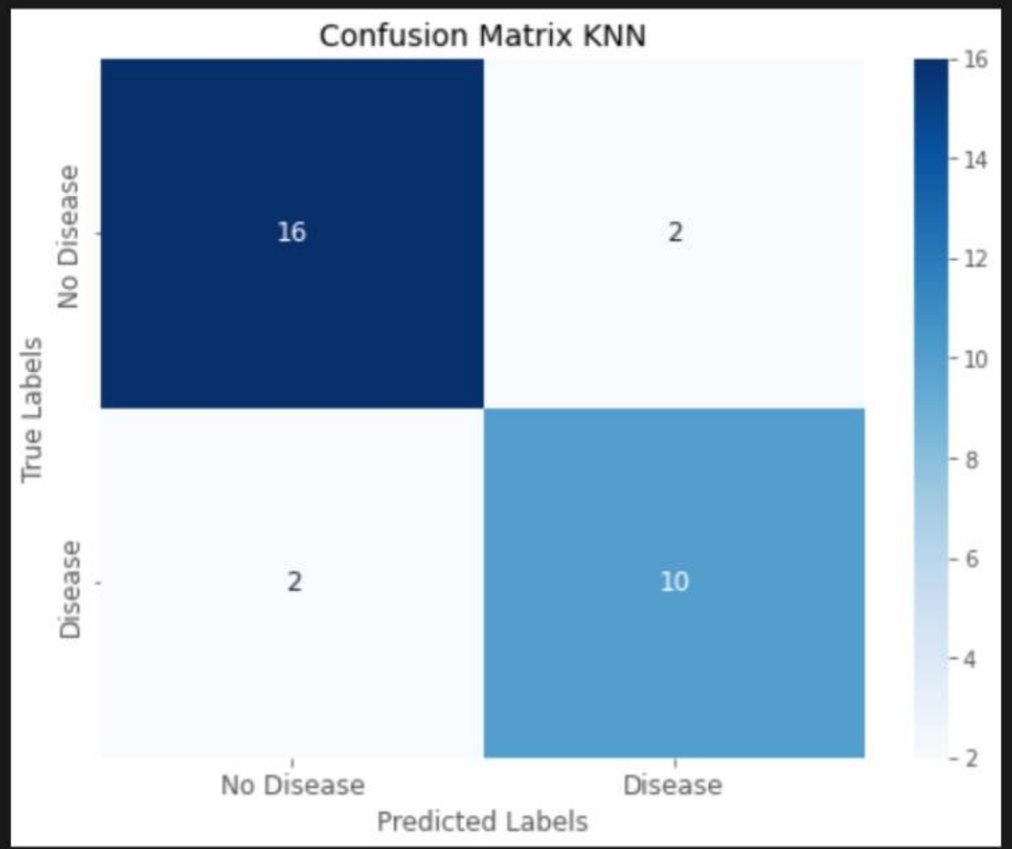
Base Model Scores (Robust) 2

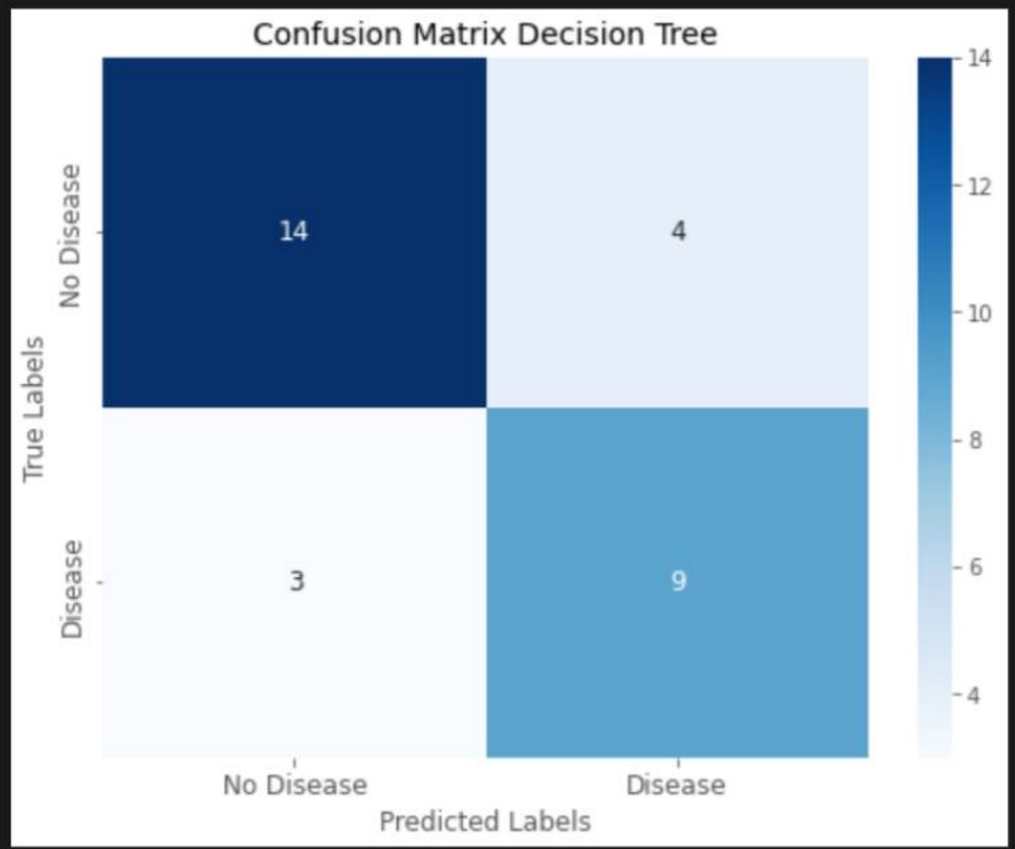
```
# Recall score for base models
models = [("K-Nearest Neighbor (KNN)", knn),
          ("Decision Tree", dt),
          ("Random Forest", rf),
          ("Support Vector Machine (SVM)", svm),
          ("Logistic Regression", lr)]

print("Recall Scores for Base Models:\n")
for i, (name, model) in enumerate(models):
    y_pred = model.predict(X_test_robust2)
    score = recall_score(y_test_robust2, y_pred)
    print(f"Model {i+1}: {name:<30} - Recall score = {score:.3f}")
```

```
Recall Scores for Base Models:

Model 1: K-Nearest Neighbor (KNN)       - Recall score = 0.667
Model 2: Decision Tree                  - Recall score = 0.833
Model 3: Random Forest                  - Recall score = 0.750
Model 4: Support Vector Machine (SVM)   - Recall score = 0.750
Model 5: Logistic Regression            - Recall score = 0.750
```

Confusion Matrix

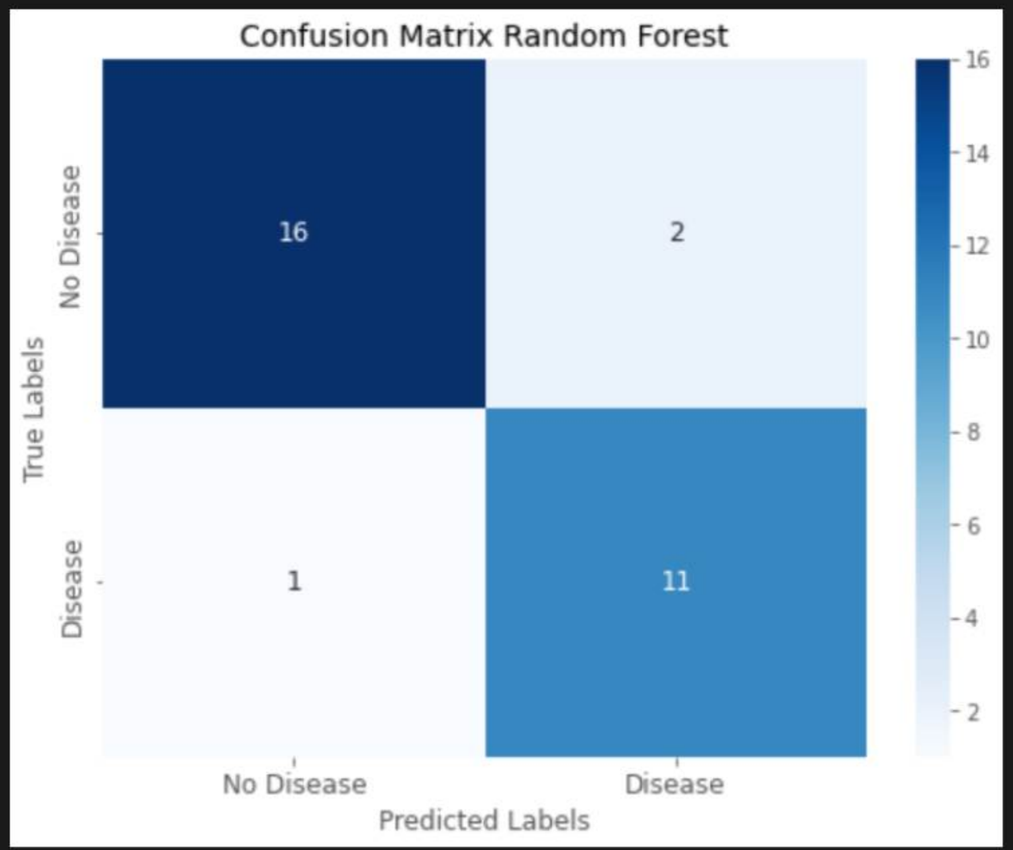
Robust - Base Model

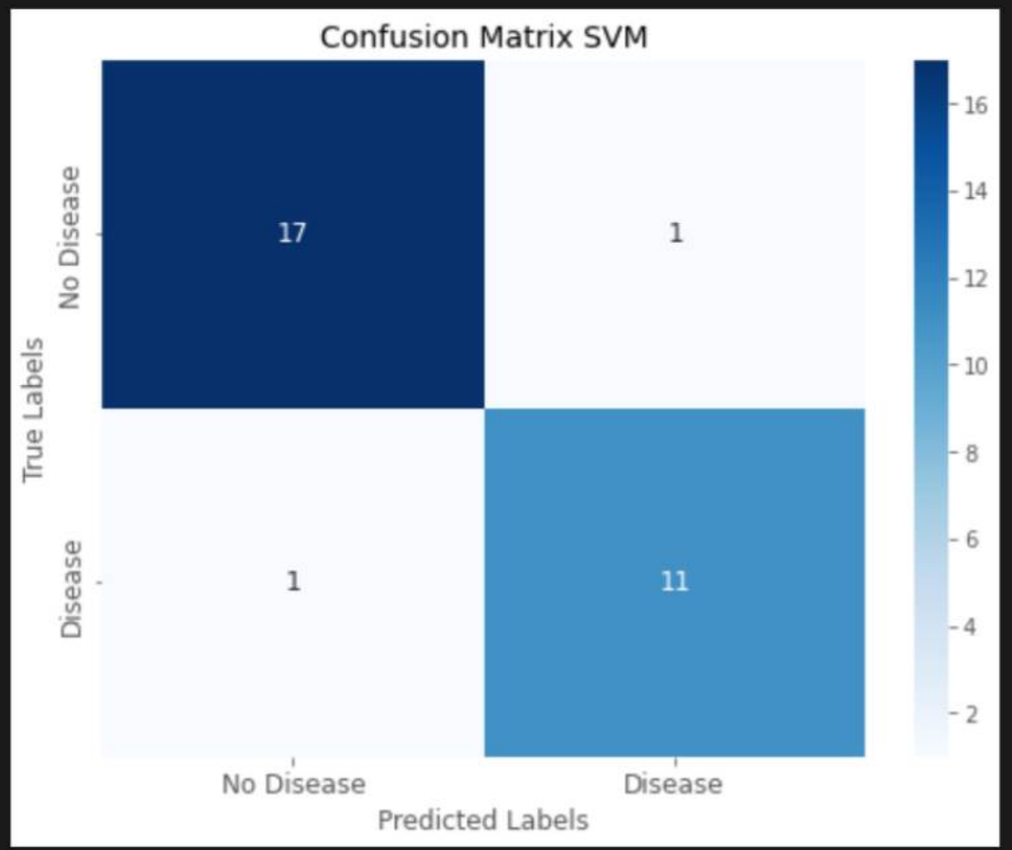




Confusion Matrix

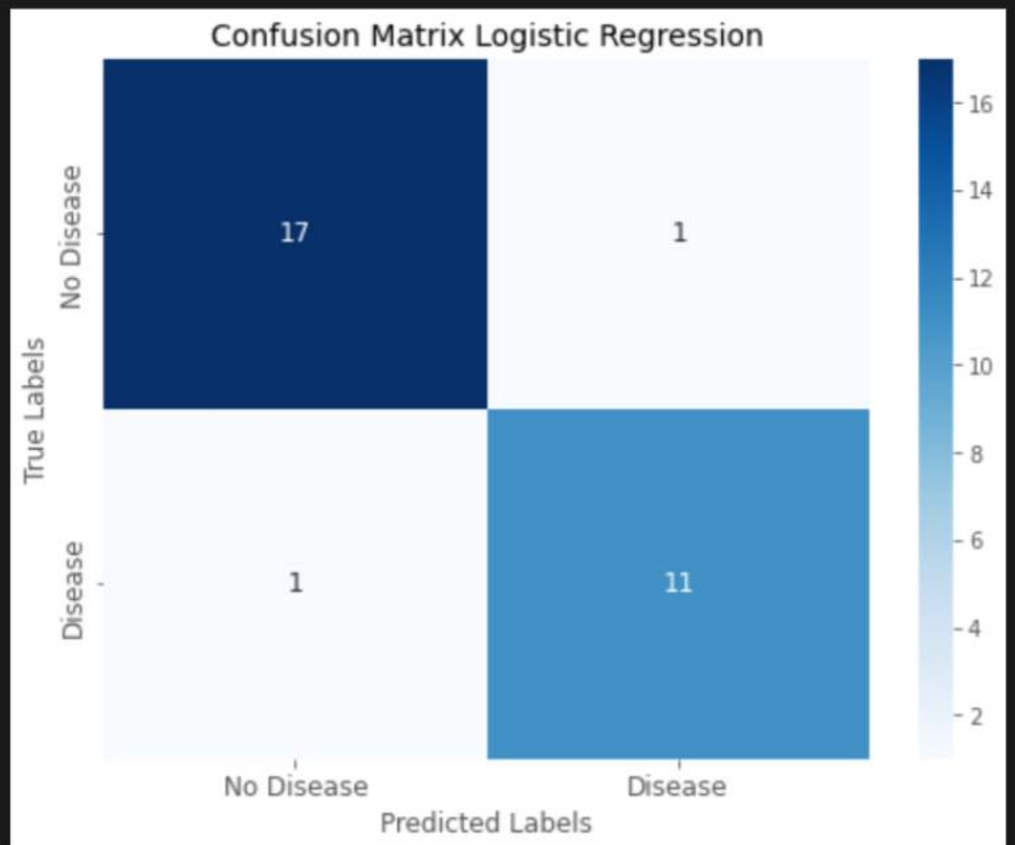
Robust - Base Model





Confusion Matrix

Robust - Base Model



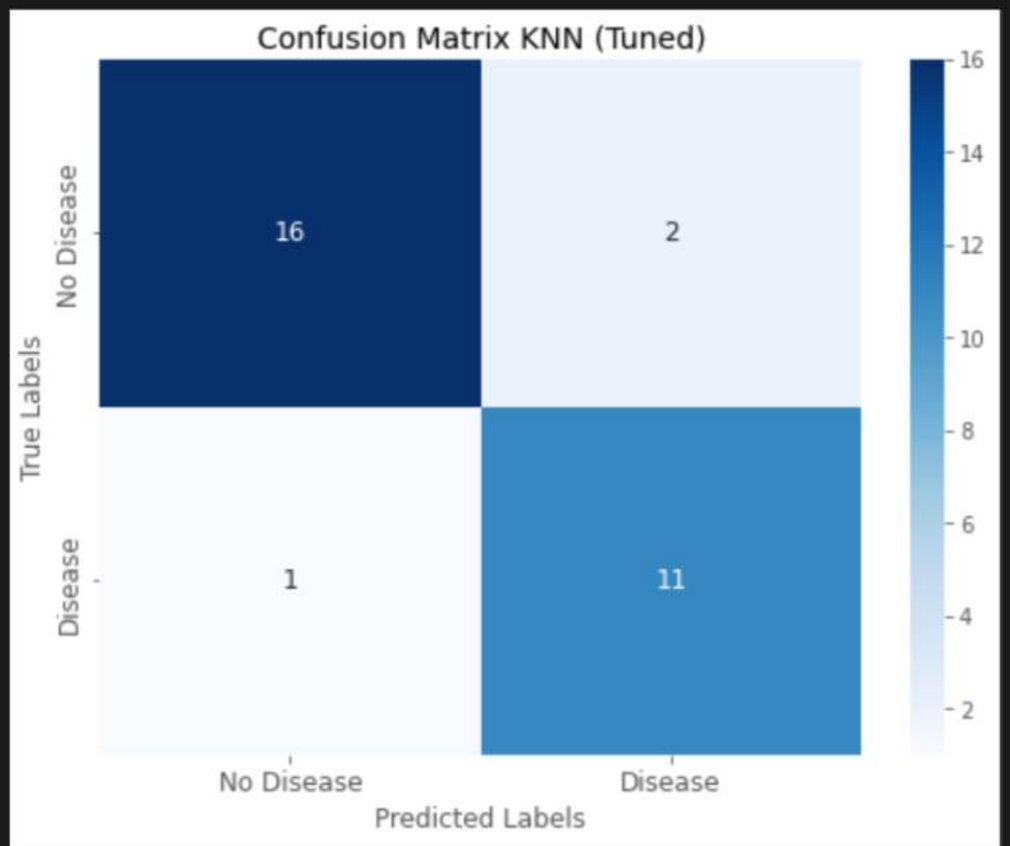
Tuned Model Scores (Robust)

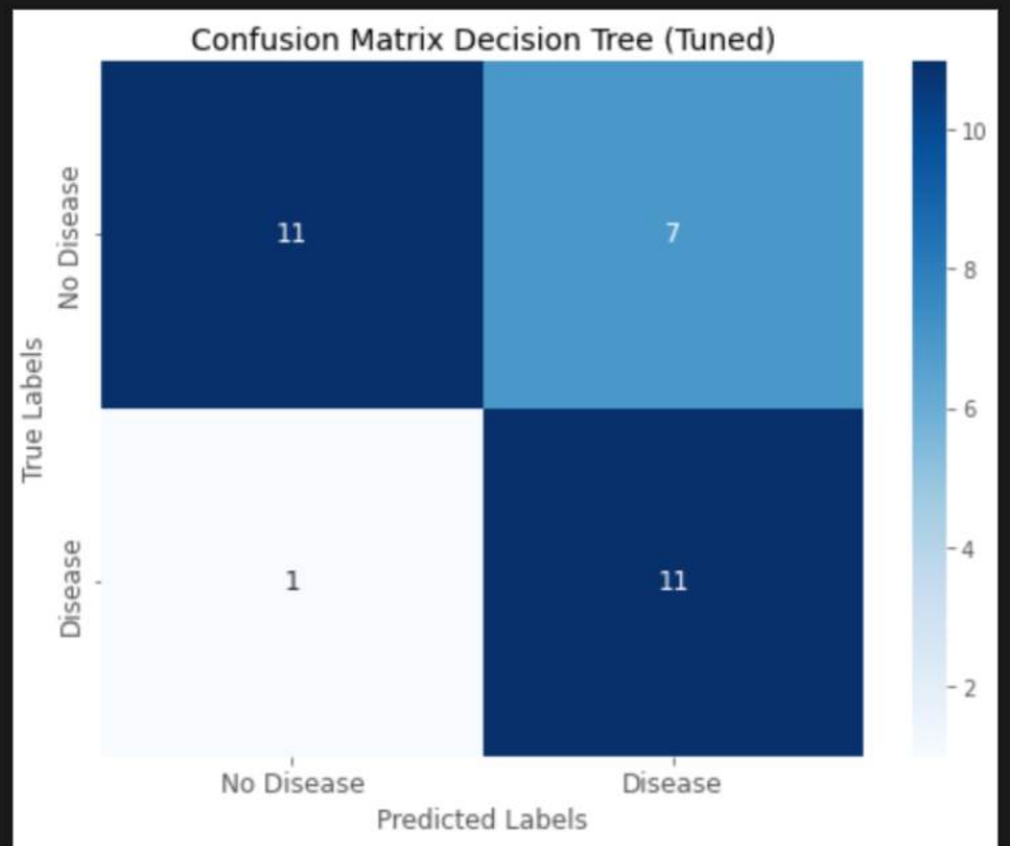
```
# Recall score for tuned models
models_best = [("K-Nearest Neighbor (KNN)", knn_best),
               ("Decision Tree", dt_best),
               ("Random Forest", rf_best),
               ("Support Vector Machine (SVM)", svm_best),
               ("Logistic Regression", lr_best)]

print("Recall score for tuned models:")
for name, model in models_best:
    y_pred = model.predict(X_test_robust2)
    score = recall_score(y_test_robust2, y_pred)
    model_number = models_best.index((name, model)) + 1
    print(f"Tuned Model {model_number}: {name:<30} - Recall score = {score:.3f}")
```

```
Recall score for tuned models:
Tuned Model 1: K-Nearest Neighbor (KNN)       - Recall score = 0.750
Tuned Model 2: Decision Tree                  - Recall score = 0.917
Tuned Model 3: Random Forest                  - Recall score = 0.833
Tuned Model 4: Support Vector Machine (SVM)   - Recall score = 0.750
Tuned Model 5: Logistic Regression            - Recall score = 0.833
```

Confusion Matrix

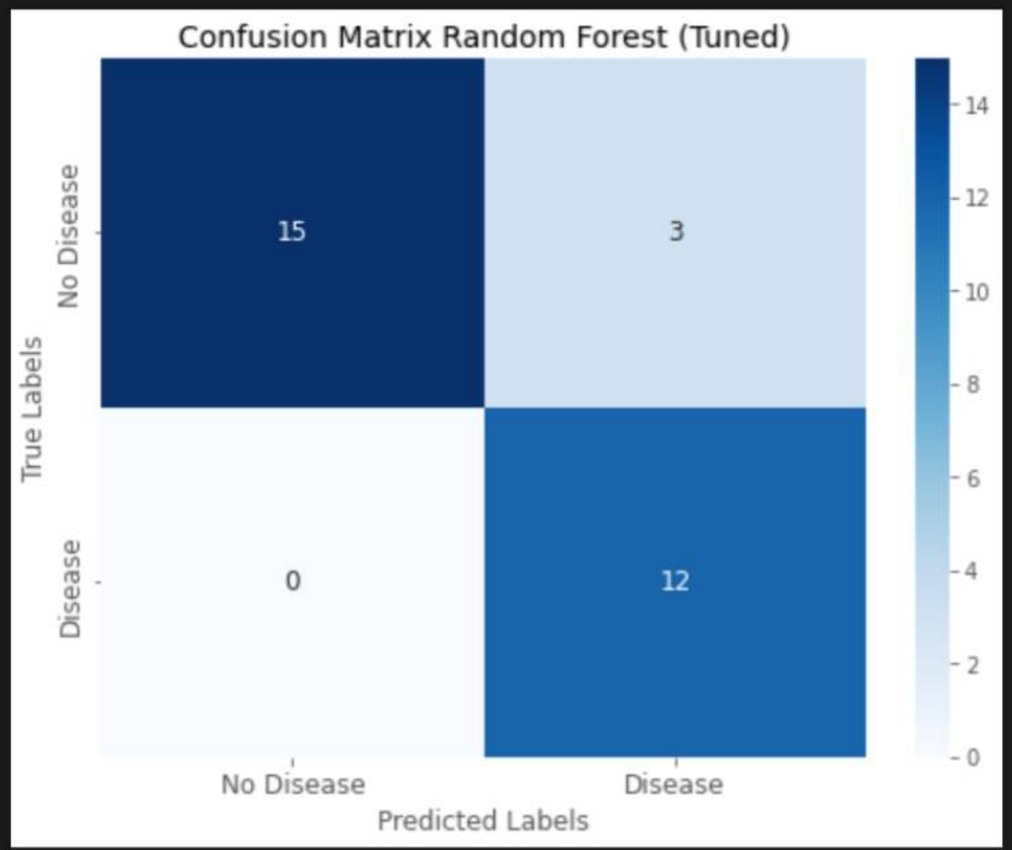
Robust - Tuned Model

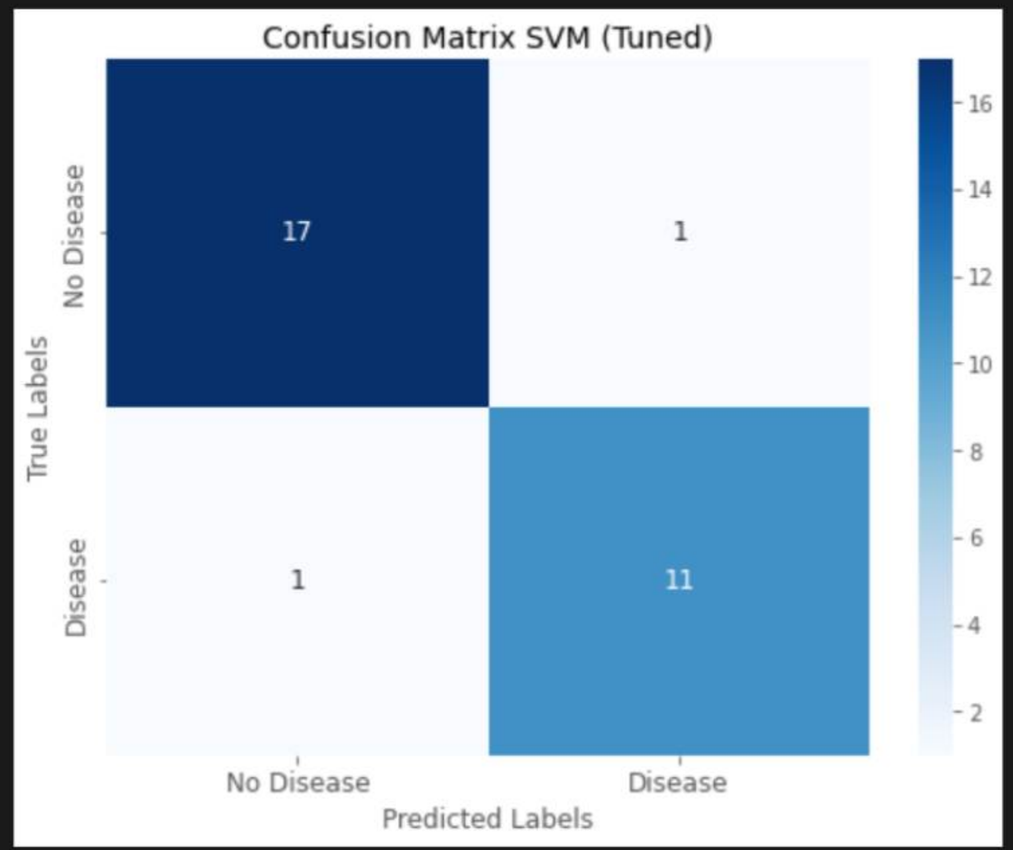




Confusion Matrix

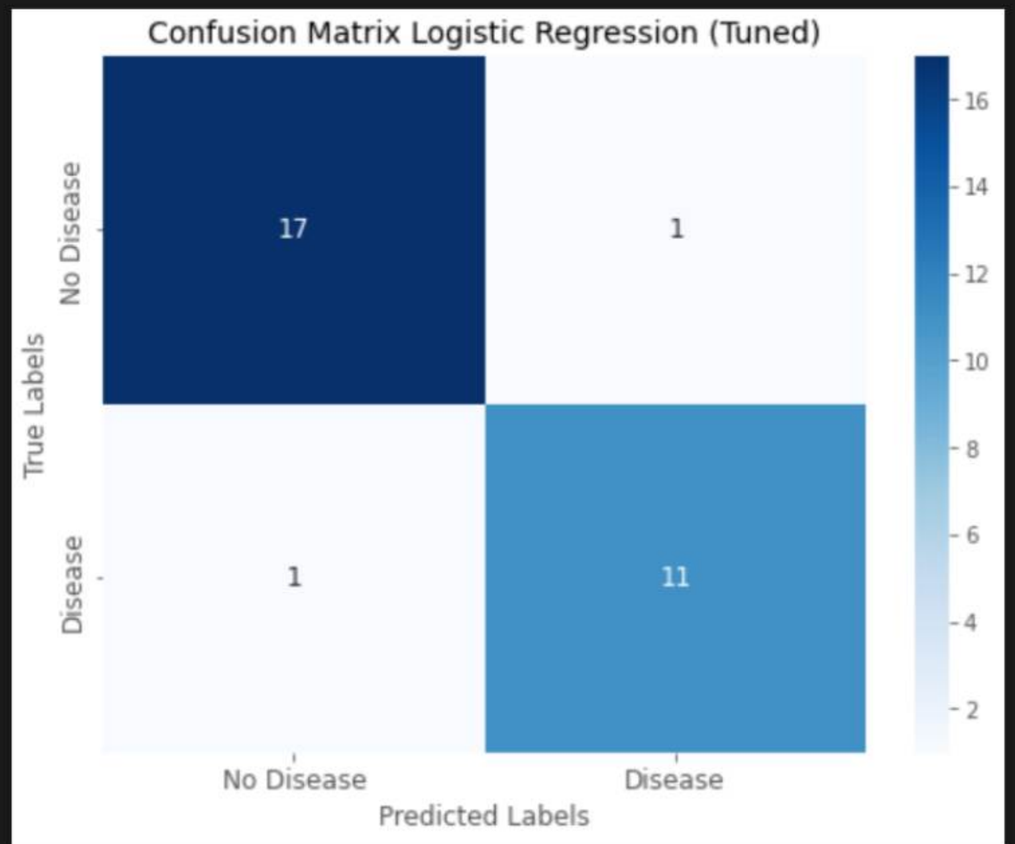
Robust - Tuned Model





Confusion Matrix

Robust - Tuned Model



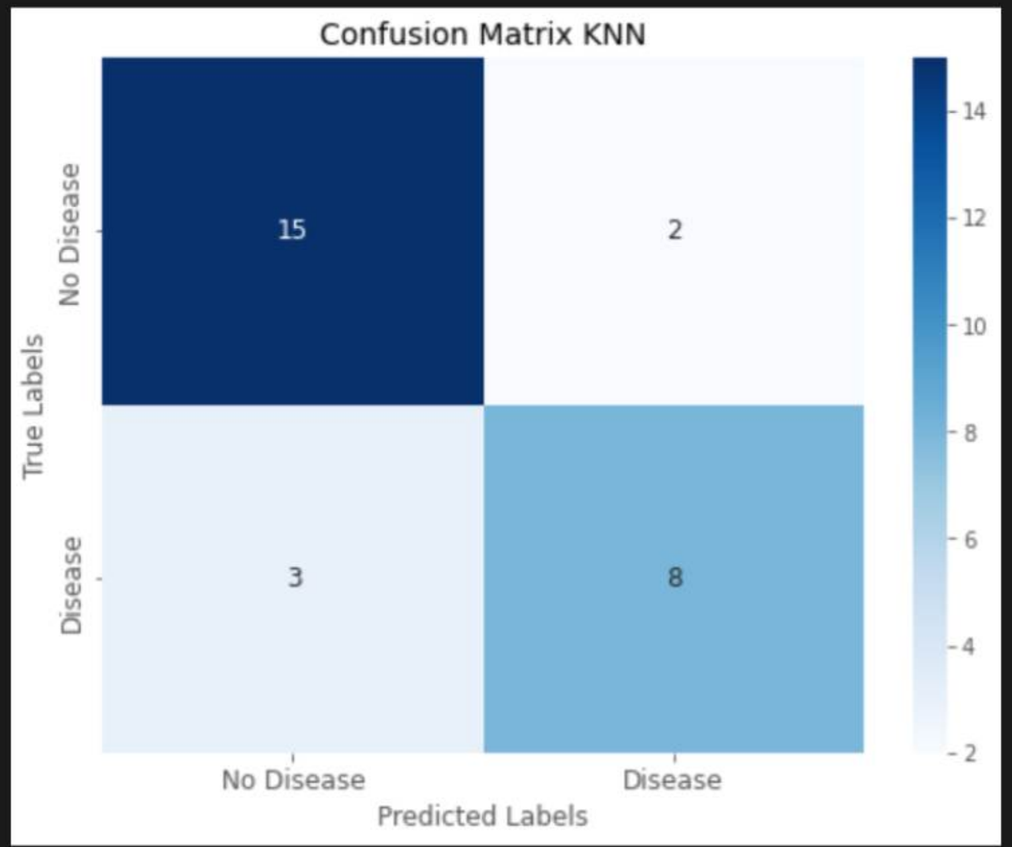
Base Model Scores (MinMax)

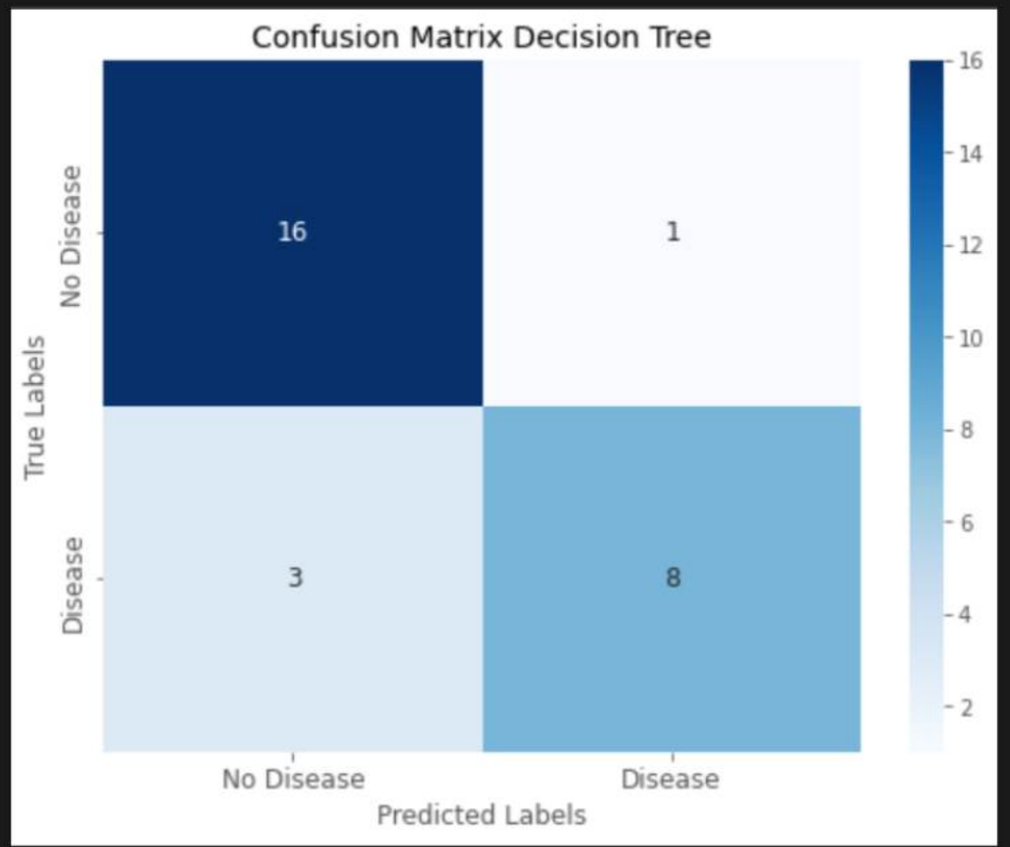
```
# Recall score for base models
models = [("K-Nearest Neighbor (KNN)", knn),
          ("Decision Tree", dt),
          ("Random Forest", rf),
          ("Support Vector Machine (SVM)", svm),
          ("Logistic Regression", lr)]

print("Recall Scores for Base Models:\n")
for i, (name, model) in enumerate(models):
    y_pred = model.predict(X_test_mm2)
    score = recall_score(y_test_mm2, y_pred)
    print(f"Model {i+1}: {name:<30} - Recall score = {score:.3f}")
```

```
Recall Scores for Base Models:

Model 1: K-Nearest Neighbor (KNN)       - Recall score = 0.667
Model 2: Decision Tree                  - Recall score = 0.733
Model 3: Random Forest                  - Recall score = 0.733
Model 4: Support Vector Machine (SVM)   - Recall score = 0.733
Model 5: Logistic Regression            - Recall score = 0.733
```

Confusion Matrix

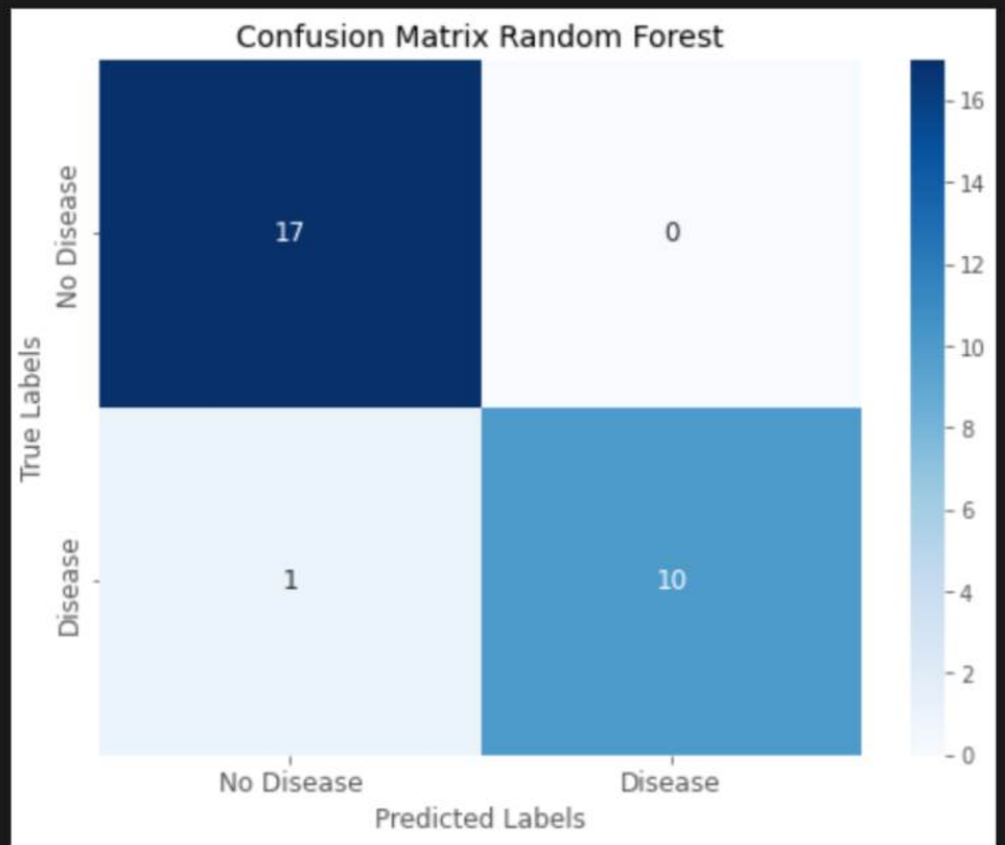
MinMax - Base Model

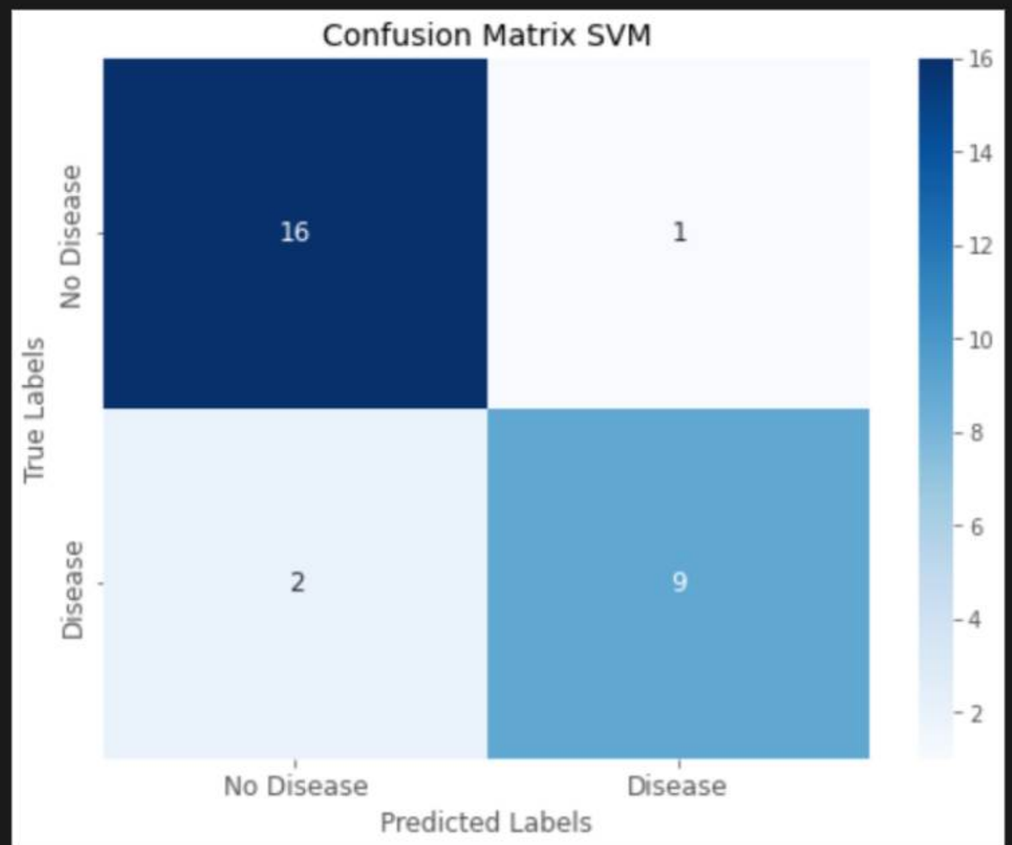




Confusion Matrix

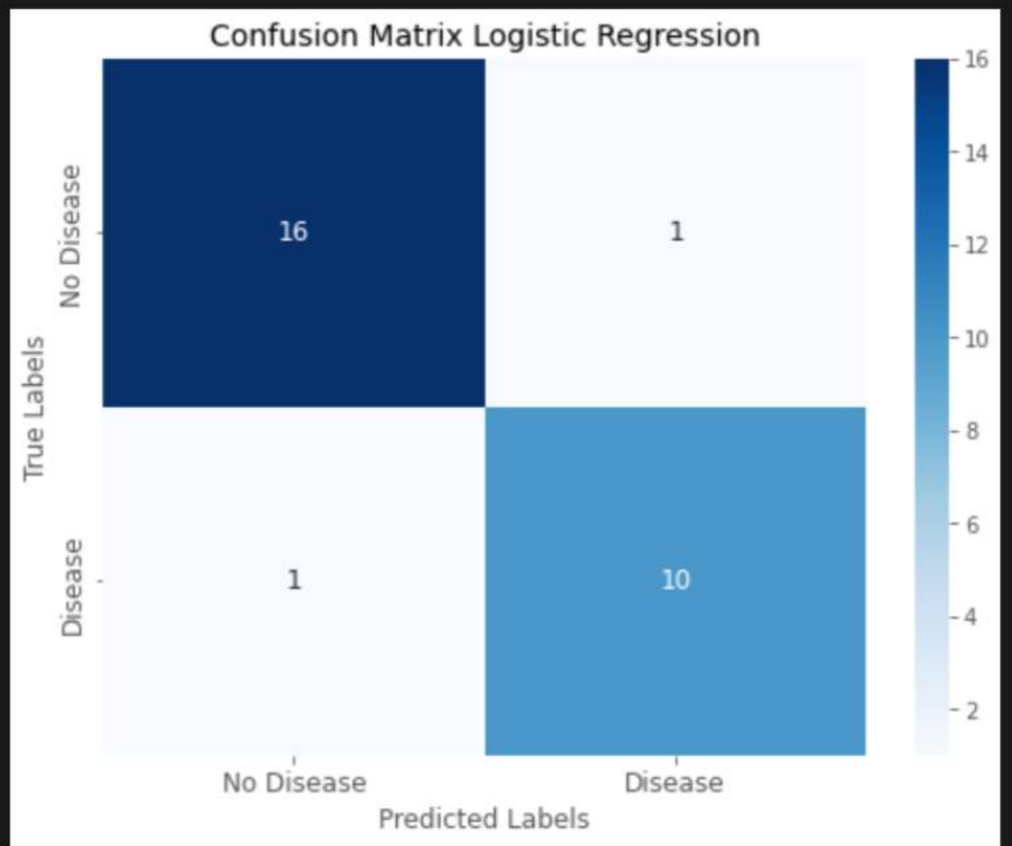
MinMax - Base Model





Confusion Matrix

MinMax - Base Model



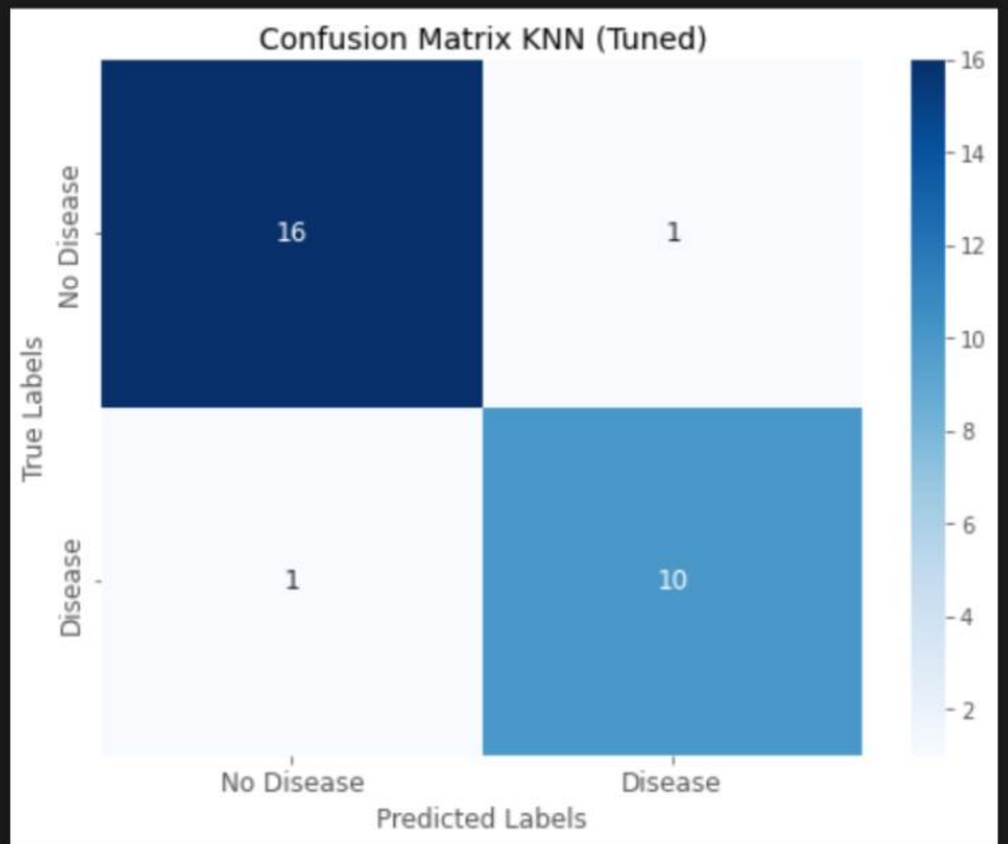
Tuned Model Scores (MinMax)

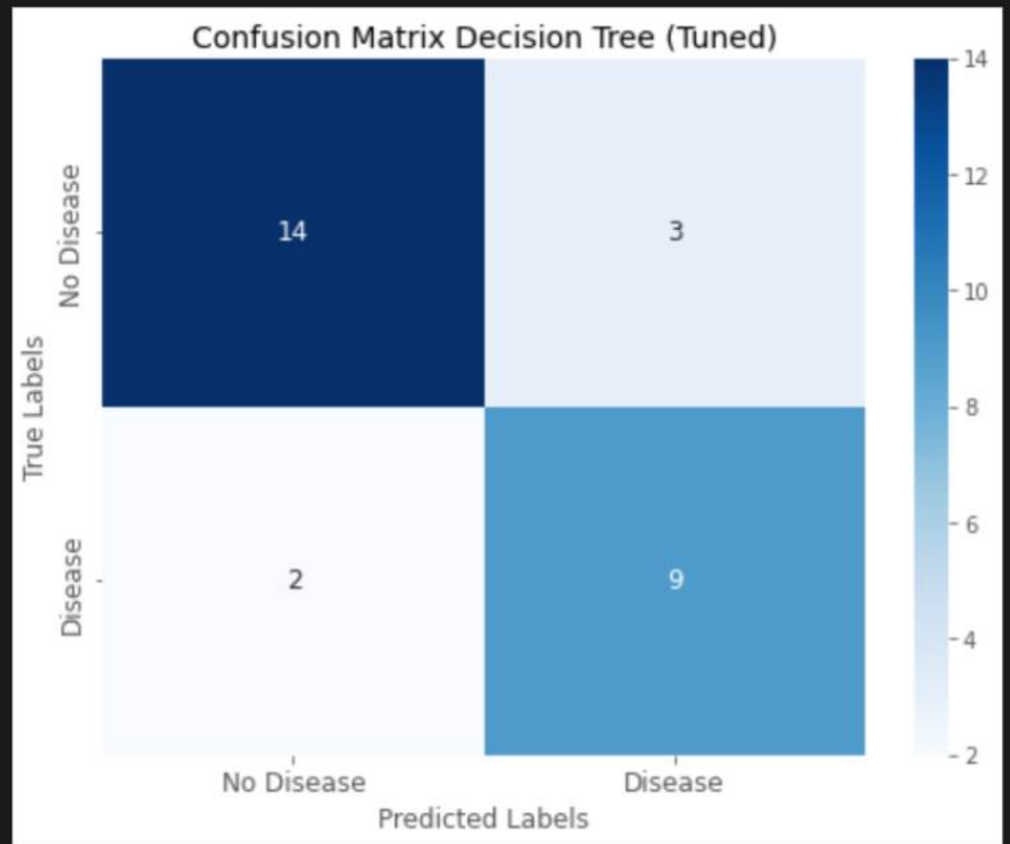
```
# Recall score for tuned models
models_best = [("K-Nearest Neighbor (KNN)", knn_best),
               ("Decision Tree", dt_best),
               ("Random Forest", rf_best),
               ("Support Vector Machine (SVM)", svm_best),
               ("Logistic Regression", lr_best)]

print("Recall score for tuned models:")
for name, model in models_best:
    y_pred = model.predict(X_test_mm2)
    score = recall_score(y_test_mm2, y_pred)
    model_number = models_best.index((name, model)) + 1
    print(f"Tuned Model {model_number}: {name:<30} - Recall score = {score:.3f}")
```

```
Recall score for tuned models:
Tuned Model 1: K-Nearest Neighbor (KNN)       - Recall score = 0.733
Tuned Model 2: Decision Tree                  - Recall score = 0.733
Tuned Model 3: Random Forest                  - Recall score = 0.800
Tuned Model 4: Support Vector Machine (SVM)   - Recall score = 0.733
Tuned Model 5: Logistic Regression            - Recall score = 0.733
```

Confusion Matrix

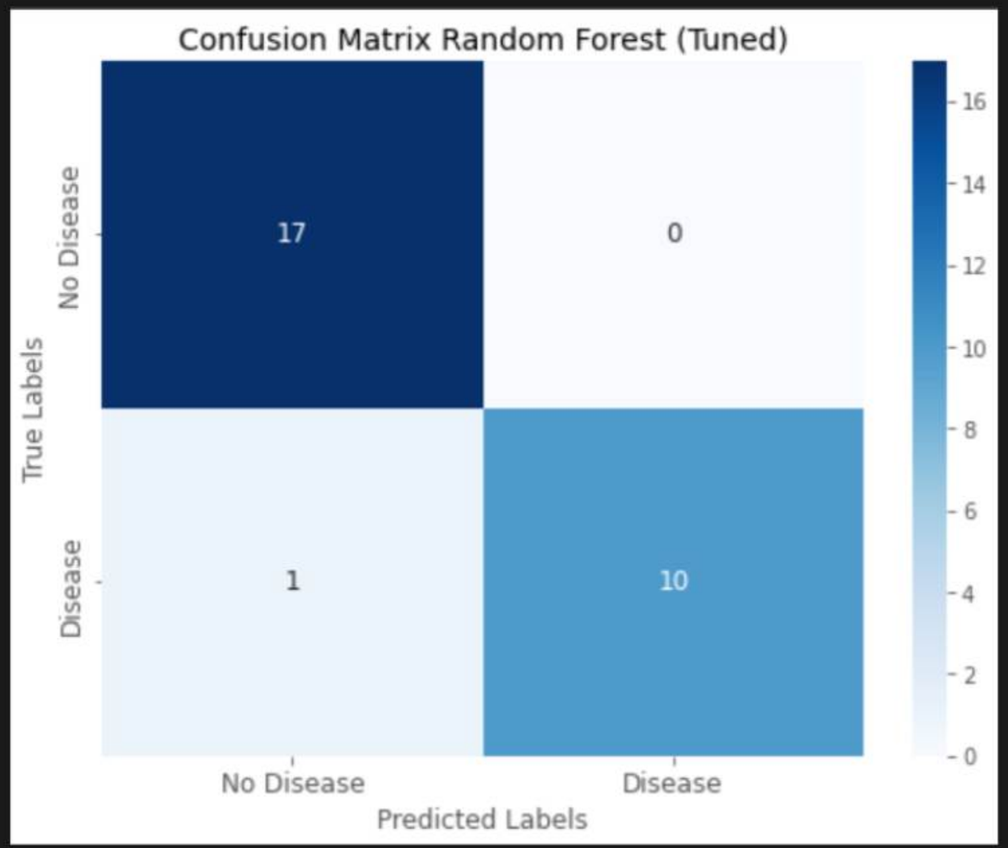
MinMax - Tuned Model

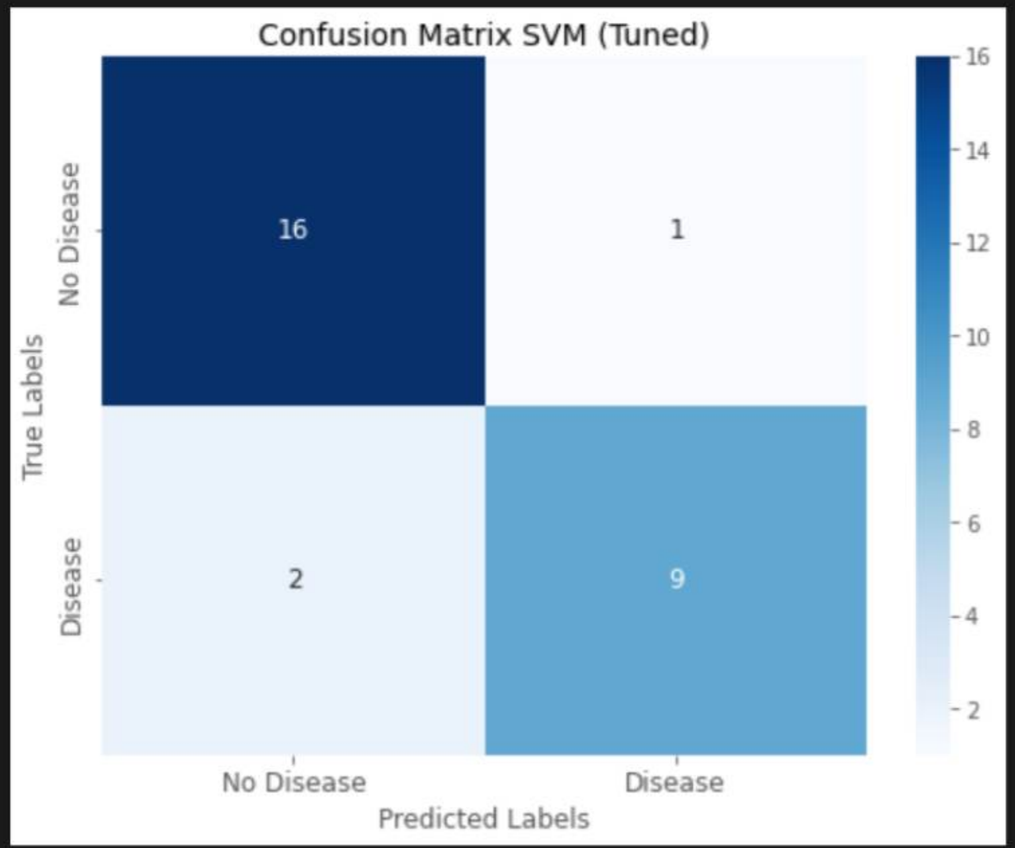




Confusion Matrix

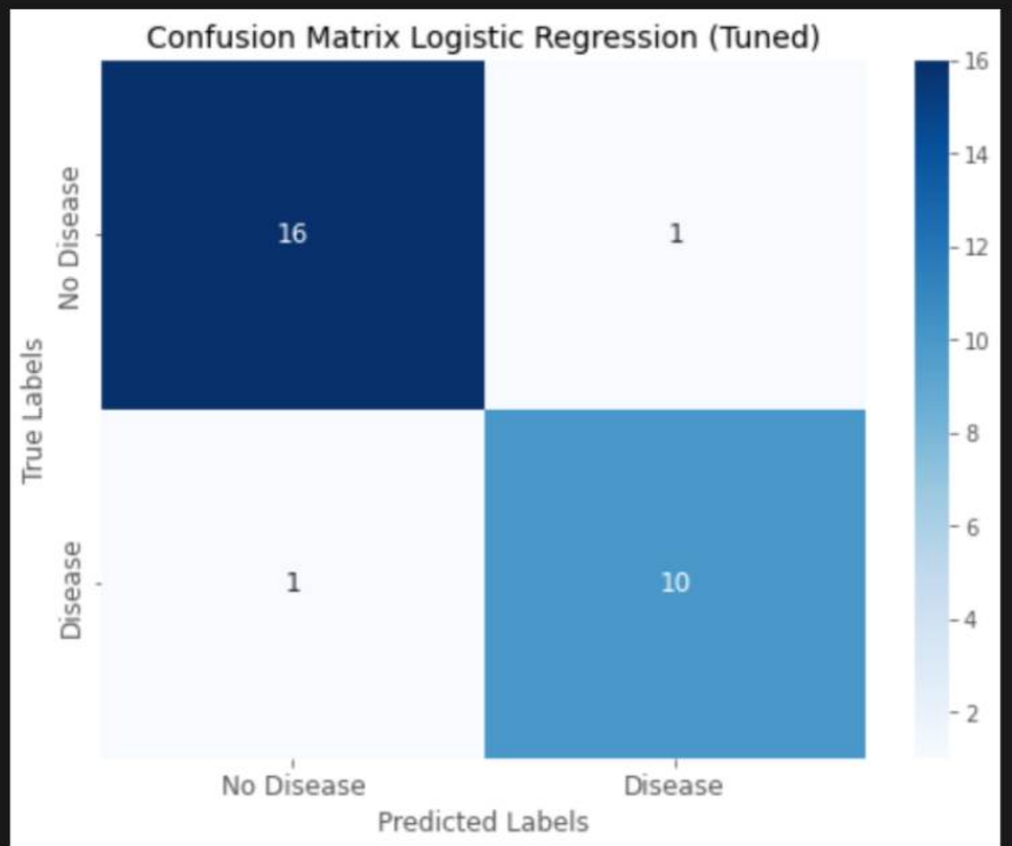
MinMax - Tuned Model





Confusion Matrix

MinMax - Tuned Model



All Score Model

Robust

```
Base Models
K-Nearest Neighbor (KNN)        - Recall score = 0.667
Decision Tree                   - Recall score = 0.833
Random Forest                   - Recall score = 0.750
Support Vector Machine (SVM)    - Recall score = 0.750
Logistic Regression             - Recall score = 0.750

Tuned Models
K-Nearest Neighbor (KNN)        - Recall score = 0.750
Decision Tree                   - Recall score = 0.917
Random Forest                   - Recall score = 0.833
Support Vector Machine (SVM)    - Recall score = 0.750
Logistic Regression             - Recall score = 0.833
```

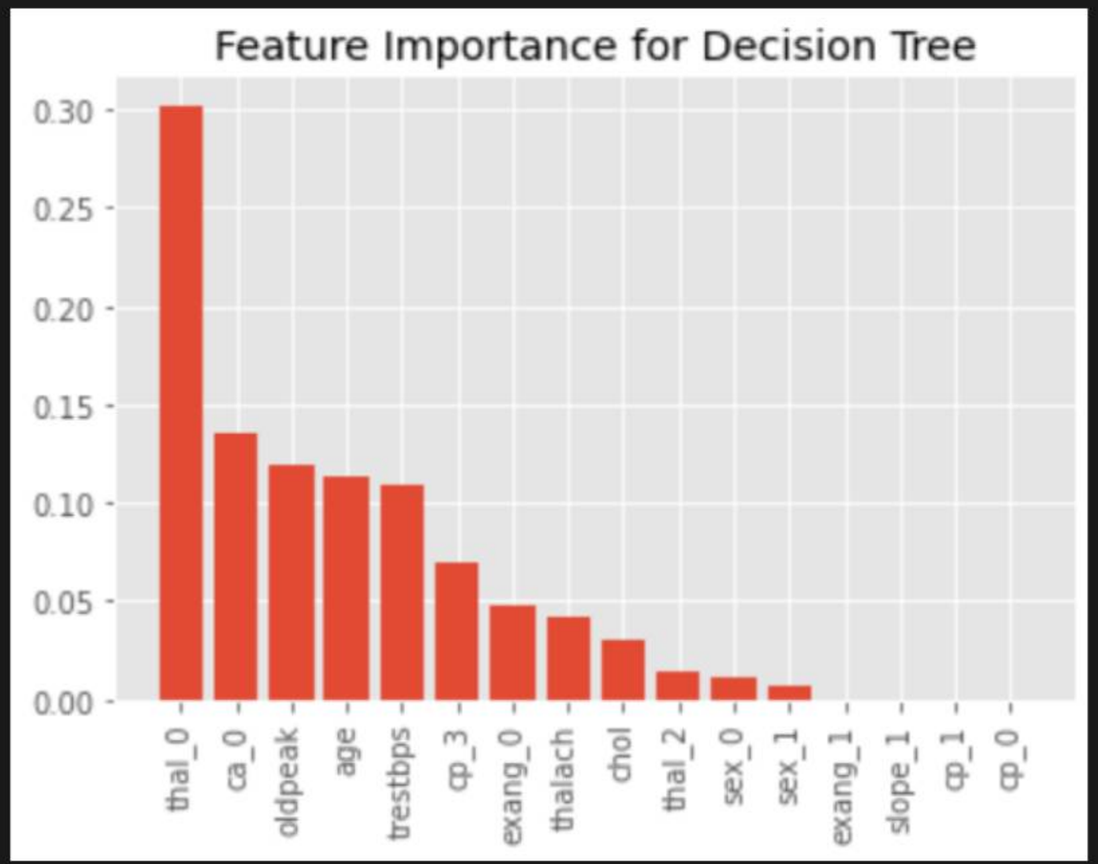
Cross Validation

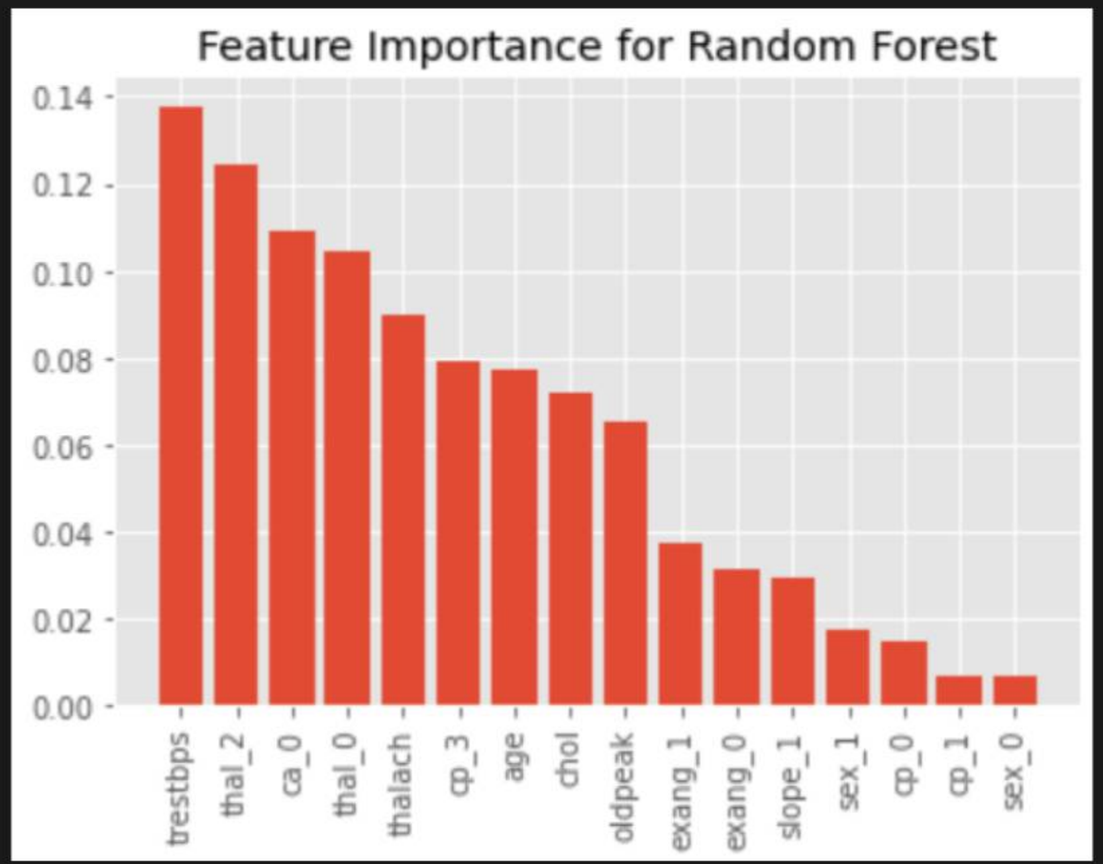
Robust

```
Base Models
K-Nearest Neighbor (KNN)      : Mean Recall score = 0.720
Decision Tree                 : Mean Recall score = 0.656
Random Forest                 : Mean Recall score = 0.768
Support Vector Machine (SVM)  : Mean Recall score = 0.784
Logistic Regression           : Mean Recall score = 0.816

Tuned Models
K-Nearest Neighbor (KNN)      : Mean Recall score = 0.688
Decision Tree                 : Mean Recall score = 0.696
Random Forest                 : Mean Recall score = 0.752
Support Vector Machine (SVM)  : Mean Recall score = 0.768
Logistic Regression           : Mean Recall score = 0.792
```

Feature Importance

Robust





All Score Model

MinMax

```
Base Models
K-Nearest Neighbor (KNN)      : Recall score = 0.667
Decision Tree                 : Recall score = 0.733
Random Forest                 : Recall score = 0.733
Support Vector Machine (SVM)  : Recall score = 0.733
Logistic Regression           : Recall score = 0.733

Tuned Models
K-Nearest Neighbor (KNN)      : Recall score = 0.733
Decision Tree                 : Recall score = 0.733
Random Forest                 : Recall score = 0.800
Support Vector Machine (SVM)  : Recall score = 0.733
Logistic Regression           : Recall score = 0.733
```

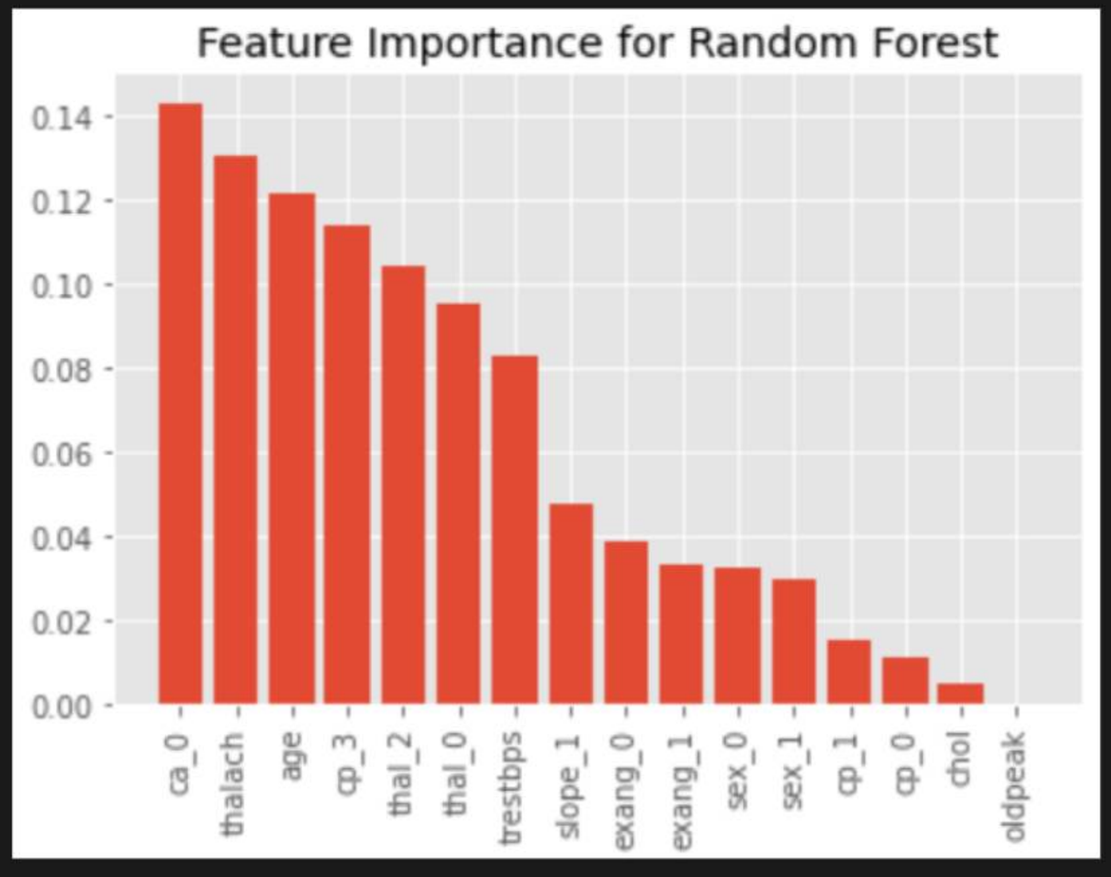
Cross Validation

MinMax

```
Base Models
K-Nearest Neighbor (KNN)       : Mean Recall score = 0.761
Decision Tree                  : Mean Recall score = 0.715
Random Forest                  : Mean Recall score = 0.797
Support Vector Machine (SVM)   : Mean Recall score = 0.788
Logistic Regression            : Mean Recall score = 0.779

Tuned Models
K-Nearest Neighbor (KNN)       : Mean Recall score = 0.733
Decision Tree                  : Mean Recall score = 0.715
Random Forest                  : Mean Recall score = 0.788
Support Vector Machine (SVM)   : Mean Recall score = 0.779
Logistic Regression            : Mean Recall score = 0.779
```

Feature Importance

MinMax



Conclusion

FINAL PROJECT DATA SCIENCE BCC

Heart Disease Classifiaction

Modelling Conclusion

1. Berdasarkan hasil dari serangkaian modelling yang telah dilakukan, pada data yang di transformasi menggunakan Robust Scaler, model yang memberikan hasil terbaik adalah base model Logistic Regression serta dilakukan feature selection dengan rata-rata Recall Score 0.816 atau 81.6%. Sedangkan pada data yang di transformasi menggunakan MinMax Scaler, model yang memberikan hasil terbaik adalah SVM setelah dilakukan hyperparameter tuning menggunakan optuna dan tanpa dilakukan feature selection dengan rata-rata Recall Score 0.816 atau 81.6%.

2. 5 Fitur terpenting menurut Random Forest dengan Robust Scaler adalah thal, ca, age, cp, dan trestbps. Sedangkan 5 fitur terpenting menurut Random Forest dengan MinMax Scaler adalah ca, thal, age, thalach, dan trestbps.

Final Conclusion

1. Berdasarkan hasil analisis diatas, Karakter klinis yang menjadi faktor utama yang menyebabkan penyakit jantung berdasarkan fitur-fitur terpenting menurut Random Forest adalah thalach, thal, age, ca, cp, dan trestbps. Sehingga untuk mengetahui kemungkinan seseorang terkena penyakit jantung dapat dilihat dari kondisi-kondisi diatas.

2. Dari kelima model yang diuji menggunakan dua metode scaling yang berbeda, model Machine Learning yang mampu memberikan hasil prediksi terbaik adalah Logistic Regression menggunakan metode Robust Scaler dan dilakukan feature selection tanpa tuning dengan rata-rata Recall Score 81.6% dan SVM menggunakan MinMax Scaler setelah dilakukan tuning tanpa feature selection dengan rata-rata Recall score 81.6%

TERIMA KASIH