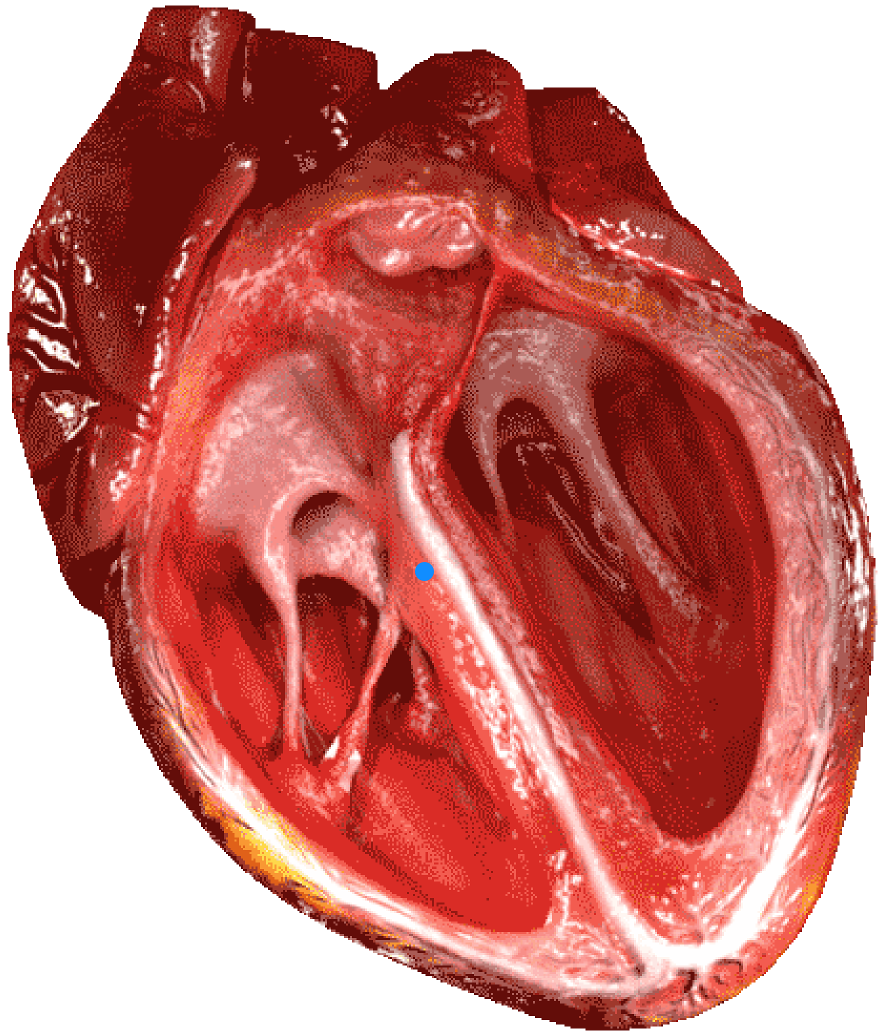
SUMMATIVE CAPSTONE PROJECT

by Noel Rodrigues

Will I Get A Heart Attack?

1. Heart attack is the major cause of morbidity and mortality
2. This project attempts to reduce the effort and time by automating the risk prediction with the help of a binary classifier.
3. Various machine learning models were used to build the classifier.
4. One of the resulting algorithms gave an accuracy score of 84%.



2. OBJECTIVE

To predict whether a patient is at risk of a heart attack.

This is a binary outcome.

Negative (-) = 0, patient is NOT at risk
Positive (+) = 1, patient is at risk

TARGET AUDIENCE:
Heart Centres / Cardiologists

3. METHODOLOGY

DATA SOURCE:
Kaggle
https://www.kaggle.com/nareshbhat/health-care-data-set-on-heart-attack-possibility

PREDICTION METRIC:
Classification Model

ML MODELS:
Logistic Regression
Random Forest
Decision Tree

TOOLS:
Pandas, Numpy, Seaborn and Scikit Learn via Jupyter Notebook IDE

MS SQL

Power BI

4. PROCESS FLOW

4.1 Imports & Data Cleaning

4.2 Understanding the Data

4.3 Exporatory Data Analysis (EDA)

4.4 Preprocessing the Data

4.5 Final Preprocessed Data

4.1 Imports & Data Cleaning

Connect to SQL view

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
# Connect to SQl view
import pyodbc
conn = pyodbc.connect('Driver={SQL Server};'
                      'Server=WSAMZN-DAVSQS3E;'
                      'Database=NoelHasAHeartAttack;'
                      'Trusted_Connection=yes;')

data = pd.read_sql_query('SELECT * FROM heart_attack_data_cleaned', conn)
cursor = conn.cursor()
```

data.head(15)

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	num
28	1	2	130	132	0	2	185	0	0.00	0
29	1	2	140	167	0	0	170	0	0.00	0
29	1	2	120	243	0	0	160	0	0.00	0
30	0	1	170	237	0	1	170	0	0.00	0
31	0	2	100	219	0	1	150	0	0.00	0
31	1	4	120	270	0	0	153	1	1.50	1
32	0	2	105	198	0	0	165	0	0.00	0
32	1	2	110	225	0	0	184	0	0.00	0
32	1	2	125	254	0	0	155	0	0.00	0
32	1	4	118	529	0	0	130	0	0.00	1
33	0	4	100	246	0	0	150	1	1.00	1
33	1	3	120	298	0	0	185	0	0.00	0
34	1	1	140	156	0	0	180	0	0.00	1
34	0	2	130	161	0	0	190	0	0.00	0

data.tail(15)

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	num
66	1	4	140	280	0	0	94	1	1.00	1
65	1	4	170	263	1	0	112	1	2.00	1
65	1	4	130	275	0	1	115	1	1.00	1
65	1	4	140	306	1	0	87	1	1.50	1
63	1	4	150	223	0	0	115	0	0.00	1
62	0	1	160	193	0	0	116	0	0.00	0
62	1	2	140	271	0	0	152	0	1.00	0
61	1	4	125	292	0	1	115	1	0.00	0
61	0	4	130	294	0	1	120	1	1.00	0
60	1	3	120	246	0	2	135	0	0.00	0
60	1	4	100	248	0	0	125	0	1.00	1
59	1	4	130	164	0	0	125	0	0.00	1
59	0	2	130	188	0	0	124	0	1.00	0
59	1	3	180	213	0	0	100	0	0.00	0

4.2 Understanding the Dataset

1. **age**: #
2. **sex**: (binary, 1 = male, 0 = female)
3. **cp**: Chest Pain (ordinal, 1 = typical angina, 2 = atypical angina, 3 = non-anginal pain, 4 = asymptomatic)
4. **trestbps**: Resting Blood Pressure (#, normal blood pressure = > 120/80 mm)
5. **chol**: Serum Cholesterol (#, normal serum cholesterol = > 200 mg/dL)
6. **fps**: Fasting Blood Sugar (binary, > 120 mg/dl, 1 = true; 0 = false]
7. **restecg**: Resting Electrocardiographic Results (0 = normal, 1 = abnormal, 2 = ventricular hypertrophy)
8. **thalach**: Maximum Heart Rate Achieved (#)
9. **exang**: Exercise-induced Angina (binary, 1 = yes, 0 = no)
10. **oldpeak**: ST Depression induced by Exercise Relative to Rest (#)
11. **slope**: The Slope of the Peak Exercise ST Segment (ordinal, 1 = upsloping, 2 = flat, 3 = downsloping)
12. **ca**: Number of Major Vessels Coloured by Fluoroscopy (ordinal, 0-3,)
13. **thal**: Maximum Heart Rate Achieved (Ordinal, 3 = normal, 6 = fixed defect, 7 = reversable defect]
14. **num**: Diagnosis of Heart Disease (binary, 0 = < 50% diameter narrowing, 1 = > 50% diameter narrowing)

```
data.dtypes
```

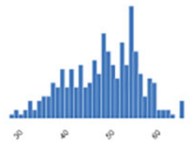
```
age            int64
sex            int64
cp             int64
trestbps       int64
chol           int64
fbs            int64
restecg        int64
thalach        int64
exang          int64
oldpeak      float64
num            int64
dtype: object
```

4.3 EDA

age
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	38
Distinct (%)	12.9%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	47.82653061

Minimum	28
Maximum	66
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	2.4 KiB



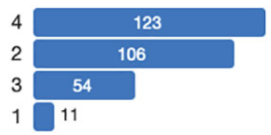
sex
Categorical

Distinct	2
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Memory size	2.4 KiB

cp
Categorical

HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION

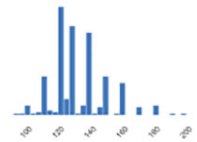
Distinct	4
Distinct (%)	1.4%
Missing	0
Missing (%)	0.0%
Memory size	2.4 KiB



trestbps
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	31
Distinct (%)	10.5%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	132.6428571

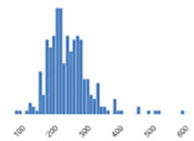
Minimum	92
Maximum	200
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	2.4 KiB



chol
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	153
Distinct (%)	52.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	248.1734694

Minimum	85
Maximum	603
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	2.4 KiB



fbs
Categorical

Distinct	2
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Memory size	2.4 KiB

4.3 EDA

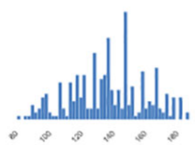
restecg
Categorical

Distinct	3
Distinct (%)	1.0%
Missing	0
Missing (%)	0.0%
Memory size	2.4 KiB

thalach
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	71
Distinct (%)	24.1%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	139.0986395

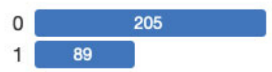
Minimum	82
Maximum	190
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	2.4 KiB



exang
Categorical

HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION

Distinct	2
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Memory size	2.4 KiB

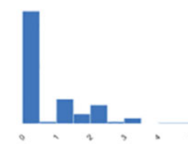


oldpeak
Real number ($\mathbb{R}_{\geq 0}$)

HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
ZEROS

Distinct	10
Distinct (%)	3.4%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	0.5860544218

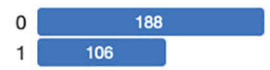
Minimum	0
Maximum	5
Zeros	189
Zeros (%)	64.3%
Negative	0
Negative (%)	0.0%
Memory size	2.4 KiB



num
Categorical

HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION

Distinct	2
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Memory size	2.4 KiB



4.3 EDA

oldpeak (ST depression induced by exercise relative to rest), **exang** (exercise-induced angina), and **cp** (chest pain) have the most correlation with **num** (diagnosis of heart disease)

4.4 Preprocessing the Data

Scaling a few attributes using normal scaler, to allow better algorithm convergence.

4.5 Final Preprocessed Data

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	target
0	28	1	2	-0.150199	-1.742241	0	2	1.951972	-0.658898	0.0	0
1	29	1	2	-0.718518	-0.077586	0	0	0.888838	-0.658898	0.0	0
2	29	1	2	0.418120	-1.217350	0	0	1.314091	-0.658898	0.0	0
3	30	0	1	2.123077	-0.167567	0	1	1.314091	-0.658898	0.0	0
4	31	0	2	-1.855155	-0.437511	0	1	0.463584	-0.658898	0.0	0
5	32	0	2	-1.570996	-0.752446	0	0	1.101465	-0.658898	0.0	0
6	32	1	2	-1.286836	-0.347530	0	0	1.909446	-0.658898	0.0	0
7	32	1	2	-0.434358	0.087380	0	0	0.676211	-0.658898	0.0	0
8	33	1	3	-0.718518	0.747243	0	0	1.951972	-0.658898	0.0	0
9	34	0	2	-0.150199	-1.307331	0	0	2.164598	-0.658898	0.0	0

5 DATA PREPARATION

5.1 Splitting the Data into Training and Testing Datasets

```
y = data['target']
X = data.drop('target',axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state = 0)

#80% Train and 20% Test Data
```

```
# Checking if the data is equally splitted or not, otherwise it will cause data imbalance.
from collections import Counter

print(y_test.unique())
Counter(y_train)
```

```
[1 0]

Counter({0: 155, 1: 80})
```

5.2 Training the Models

Using these Machine Learning models:

1. Logistic Regression
2. Random Forest
3. Decision Tree

```
trained_records = []
```

```
model = LogisticRegression()
```

```
import time

start = time.time()

# Train model
model.fit(X_train,y_train)
train_time = time.time() - start

# Test model
y_pred = model.predict(X_test)

# Get model name
model_name = model.__class__.__name__

# Results
test_precision = precision_score(y_test, y_pred, pos_label=1)
test_recall = recall_score(y_test, y_pred, pos_label=1)
test_f1score = f1_score(y_test, y_pred, pos_label=1)
test_acc_score = accuracy_score(y_test, y_pred)

# Print Accuracy Scores
print("Accuracy of Logistic Regression: {:.3f}".format(test_acc_score*100),'%\n')
print(classification_report(y_test,y_pred))

# Store record into trained dist
trained_records.append({
    'model': model_name,
    'train_time': train_time,
    'r2': test_precision,
    'recall': test_recall,
    'f1score': test_f1score,
    'acc_score': test_acc_score
})
trained_records
```

5.2 Training the Models

1. LOGISTIC REGRESSION

Accuracy of Logistic Regression: 84.746 %

	precision	recall	f1-score	support
0	0.85	0.88	0.87	33
1	0.84	0.81	0.82	26
accuracy			0.85	59
macro avg	0.85	0.84	0.84	59
weighted avg	0.85	0.85	0.85	59

2. RANDOM FOREST

Accuracy of Random Forest: 83.051 %

	precision	recall	f1-score	support
0	0.81	0.91	0.86	33
1	0.86	0.73	0.79	26
accuracy			0.83	59
macro avg	0.84	0.82	0.82	59
weighted avg	0.83	0.83	0.83	59

3. DECISION TREE

Accuracy of Decision Tree: 81.356 %

	precision	recall	f1-score	support
0	0.78	0.94	0.85	33
1	0.89	0.65	0.76	26
accuracy			0.81	59
macro avg	0.83	0.80	0.80	59
weighted avg	0.83	0.81	0.81	59

5.3 Evaluation of Models

```
trained_records
```

```
trained_records = pd.DataFrame(trained_records,columns=['model','train_time','r2','recall','f1score','acc_score'])
trained_records
```

```
# Store results into SQL

trained_insert = '''
    insert into params
    values (?, GETDATE(), ?, ?, ?, ?)
'''

for rec in trained_records.iterrows():
    values = (
        rec[1]['model'],
        rec[1]['train_time'],
        rec[1]['r2'],
        rec[1]['recall'],
        rec[1]['f1score'],
        rec[1]['acc_score'])
    cursor.execute(trained_insert, values)

conn.commit()
# cursor.close()
```

	model	train_time	r2	recall	f1score	acc_score
0	LogisticRegression	0.031517	0.840000	0.807692	0.823529	0.847458
1	RandomForestClassifier	0.125773	0.863636	0.730769	0.791667	0.830508
2	DecisionTreeClassifier	0.003075	0.904762	0.730769	0.808511	0.847458

5.4 Test Best Model

From the above results, Logistic Regression has the highest Accuracy Score of 84%. So we will use Logistic Regression to predict the test set.

```
best_model = DecisionTreeClassifier()
best_model.fit(X_train,y_train)
y_pred = best_model.predict(X_test)
```

```
test_pred = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
test_pred = test_pred.rename_axis('number')
test_pred
```

```
# Insert best test model record into SQL
testPred_insert = '''
    INSERT INTO test_pred
    values(?, GETDATE(), ?, ?)
'''

for rec in test_pred.iterrows():
    values = (rec[0], rec[1]['Actual'], rec[1]['Predicted'])
    cursor.execute(testPred_insert, values)
conn.commit()
# cursor.close()
```

number	Actual	Predicted
216	1	0
212	1	0
45	0	0
230	1	1
22	0	0
239	1	1
184	0	1
199	1	1
59	0	0
73	0	0
15	0	0
12	0	0
288	1	1
129	0	0
139	0	0
263	1	1
89	0	1
144	0	0
124	0	0
157	0	0
118	0	1
207	1	1
74	0	0
210	1	0
213	1	0
284	1	1
101	0	0
8	0	0
245	1	1
276	1	1
111	0	0
153	0	0
264	1	1
176	0	0

6. CONCLUSION

1. Logistic Regression gives the best accuracy compared to other models.
2. Exercise-induced angina and chest pain are major symptoms of heart attack.
3. Based on these parameters and my cardiological report, I am NOT at risk of suffering a heart attack today.

ACKNOWLEDGEMENT

1. A.M. Aditya
2. Gouri Choudhury
3. Fredi Tantri
4. Tongwei Wang
5. Adrian Sim
6. Billy Chua
7. Christopher Simon
8. Tay An Ting
9. Hui Ching
10. Kenny Lim
11. Lindawati Hendrawan
12. Michelle Michael
13. Nor Aza Hussain
14. Zaleha Ali