Project Kaggle

By Kelly & Krista Yuda

Problem Background

Menurut WHO, Stroke merupakan penyumbang kematian ke 2 di dunia dan menyumbang angka kematian 11%. Salah satu cara menurunkan kematian akibat stroke adalah dengan menggunakan early detection system untuk memprediksi apakah ada yang berpotensi mengalami stroke sehingga dapat diketahui lebih awal & mendapatkan perawatan.

Business Understanding

01. Problem Background

Bagaimana cara *early detection* terhadap penyakit stroke?

02. Goals

Mengetahui pasien yang berpotensi mengalami stroke sebelum kejadian

03. Objective

Membuat sistem untuk mendeteksi potensi seseorang akan mengalami stroke

04. Analytic Approach

Predictive Analytics:
Membuat model yang bisa membantu memprediksi apakah seseorang akan mengalami stroke/tidak.

Data Collection

To solve this problem, we will use dataset from Kaggle: https://www.kaggle.com/fedesoriano/stroke-prediction-dataset

Understand Dataset

Column	Detail Data	Data Type
ID	Unique identifier	Integer
Gender	“Male”/”Female”/”Other”	String
Age	Integer >0	Positive Integer
Hypertension	1 or 0	Boolean
Heart_disease	1 or 0	Boolean
Ever_Married	Yes or No	Binary string
Work_Type	"children", "Govt_jov", "Never_worked", "Private" or "Self-employed"	String

Understand Dataset

Column	Detail Data	Data Type
Residence_type	“Urban”/”Rural”	String
Avg_glucose_level	Positive decimals	Float
BMI	Positive decimals	Float
Smoking_status	"formerly smoked", "never smoked", "smokes" or "Unknown"	String
Stroke	1 or 0	Boolean

What we're doing

Week 1

Understanding dataset & business goals

Week 2

Making timeline

Week 3

Handling missing data
Data visualization

Week 4

Data cleaning
One hot encoder

What we're doing

Week 5

Standardization data
Splitting dataset

Week 6

Handling minority data

Week 7

Building model
Evaluate model using right parameters

Week 8

Hyperparameter tuning & conclusion

Converting Data

BMI Index

BMI Category		Female	Male
1	Kurus	< 17 kg/m2	< 18 kg/m2
2	Normal	17 - 23 kg/ m2	18 - 25 kg/ m2
3	Kegemukan	23 - 27 kg/m2	25 - 27 kg/m2
4	Obesitas	> 27 kg/m	> 27 kg/m

Source : https://www.kemkes.go.id/index.php?txtKeyword=status+gizi&act=search-by-map&pgnumber=0&charindex=&strucid=1280&fullcontent=1&C-ALL=1

Converting Data

BMI

Converting from each gender

```
#changing BMI to categorical
df['Female_BMI'] = pd.cut(df['bmi'][df['gender']=='Female'], bins=[0, 17, 23, 27,99], labels=['Kurus', 'Normal', 'Kegemukan','Obesitas'])
df['Male_BMI'] = pd.cut(df['bmi'][df['gender']=='Male'], bins=[0, 18, 25, 27,99], labels=['Kurus', 'Normal', 'Kegemukan','Obesitas'])
df.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	Female_BMI	Male_BMI
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1	NaN	Obesitas
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1	NaN	NaN
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1	NaN	Obesitas
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1	Obesitas	NaN
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1	Kegemukan	NaN

Converting Data

BMI

Combining 2 result

```
df['Male_BMI'].fillna(value=df['Female_BMI'], inplace=True)
df=df.drop(['Female_BMI'], axis=1)
df = df.rename(columns={'Male_BMI': 'converted_bmi'})
df.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	converted_bmi
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1	Obesitas
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1	NaN
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1	Obesitas
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1	Obesitas
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1	Kegemukan

Handling Missing Value

BMI & converted_bmi

Only BMI & converted_bmi that has missing value
Inserting with mean of data, inserting 'converted_bmi' with "unknown"
Meanwhile bmi column will be drop

```
df.isna().sum()

id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
converted_bmi        0
```

```
#insert bmi nan value with unknown
df['converted_bmi'] = df['converted_bmi'].cat.add_categories('Unknown')
df['converted_bmi'] = df['converted_bmi'].fillna('Unknown')
df.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	converted_bmi
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1	Obesitas
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1	Unknown
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1	Obesitas
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1	Obesitas
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1	Kegemukan

Cleaning Data

```
from collections import Counter
print(Counter(df['gender']))
print(Counter(df['hypertension']))
print(Counter(df['heart_disease']))
print(Counter(df['ever_married']))
print(Counter(df['work_type']))
print(Counter(df['Residence_type']))
print(Counter(df['smoking_status']))
```

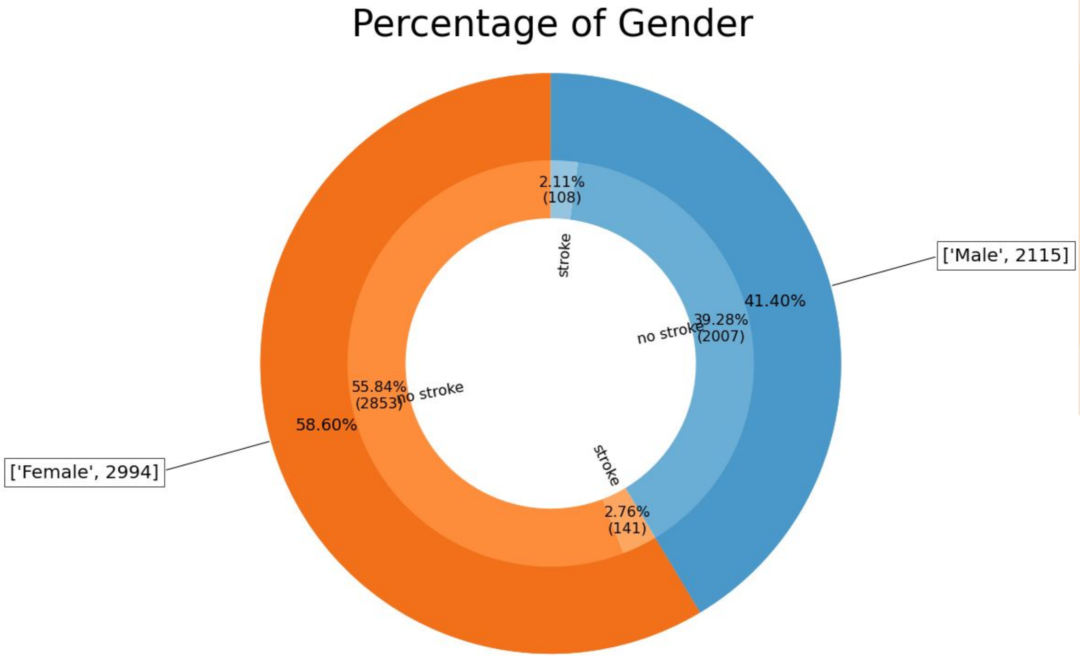
```
Counter({'Female': 2994, 'Male': 2115, 'Other': 1})
Counter({0: 4612, 1: 498})
Counter({0: 4834, 1: 276})
Counter({'Yes': 3353, 'No': 1757})
Counter({'Private': 2925, 'Self-employed': 819, 'children': 687, 'Govt_job': 657, 'Never_worked': 22})
Counter({'Urban': 2596, 'Rural': 2514})
Counter({'never smoked': 1892, 'Unknown': 1544, 'formerly smoked': 885, 'smokes': 789})
```

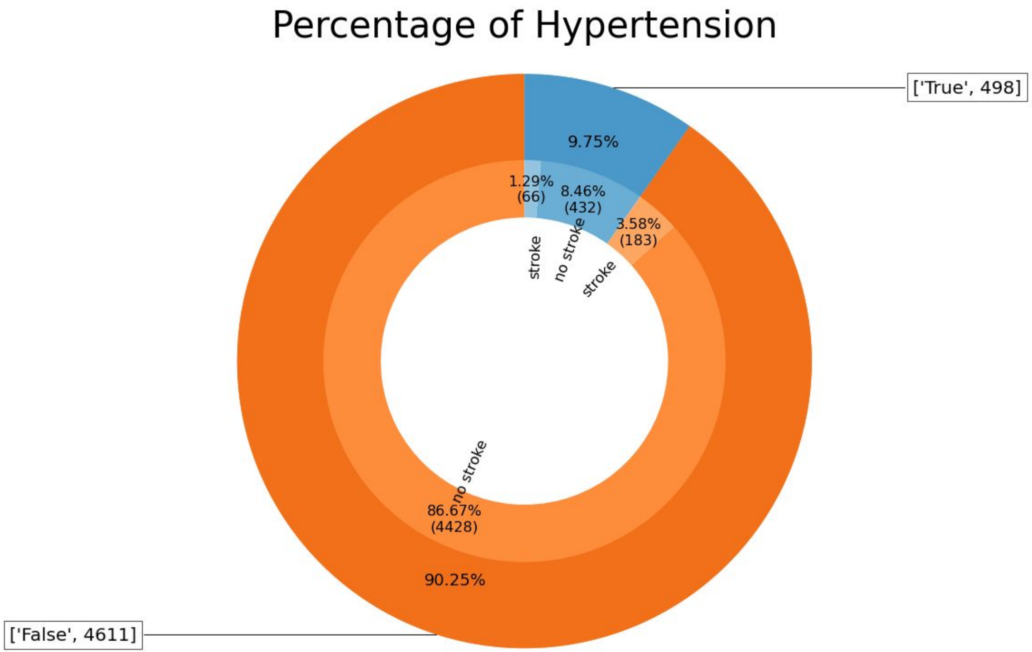
Gender

Gender has one outlier: other with only 1 data
Take it out

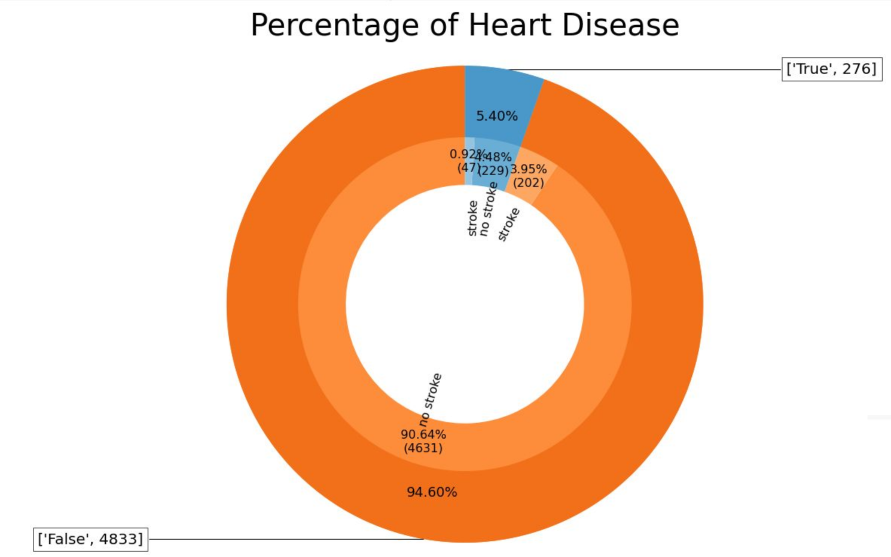
```
#drop other from gender
other = df[df['gender'] == 'Other'].index
df.drop(other, axis=0, inplace= True)
```

Data composition
Gender and Hypertension



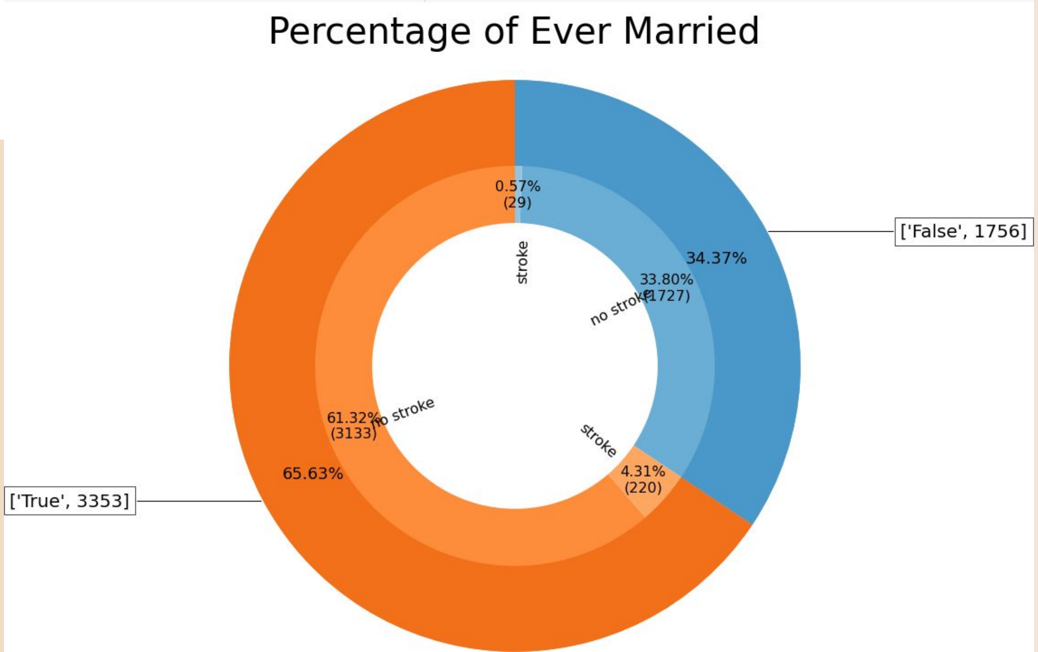


- ★ Overall gender has balance value (60% Female 40% Male) although data stroke and no stroke is not balance
- ★ Neither hypertension variable nor stroke has balance value



Data composition

Heart Disease and Marriage Status

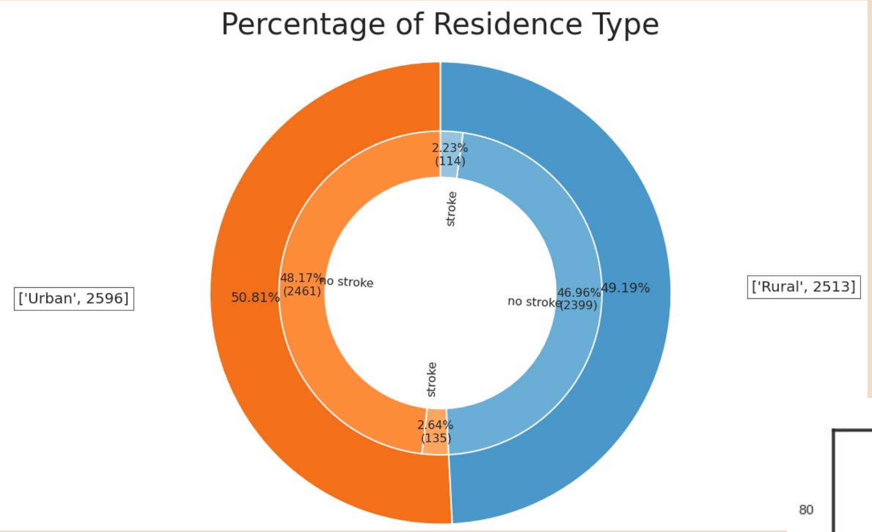


★ Neither heart disease variable nor stroke has balance value

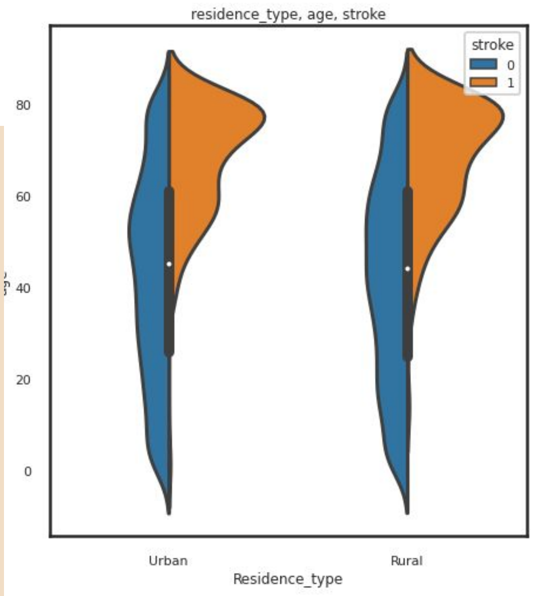
★ Overall ever married has balance value (65% Married 35% Not Married) although data stroke and no stroke is not balance

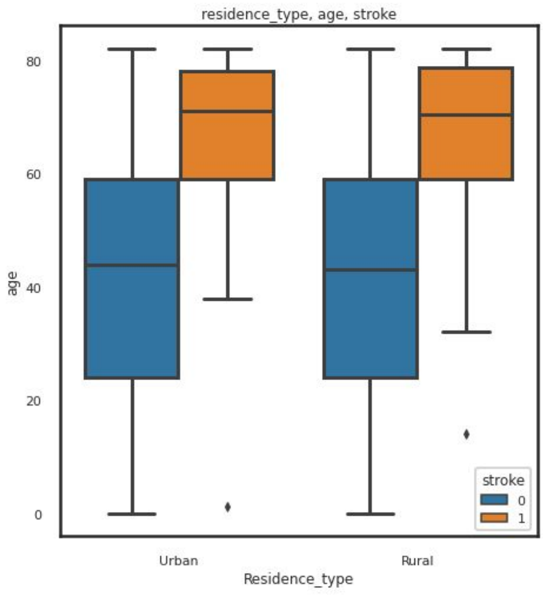
Data composition

Residence Type



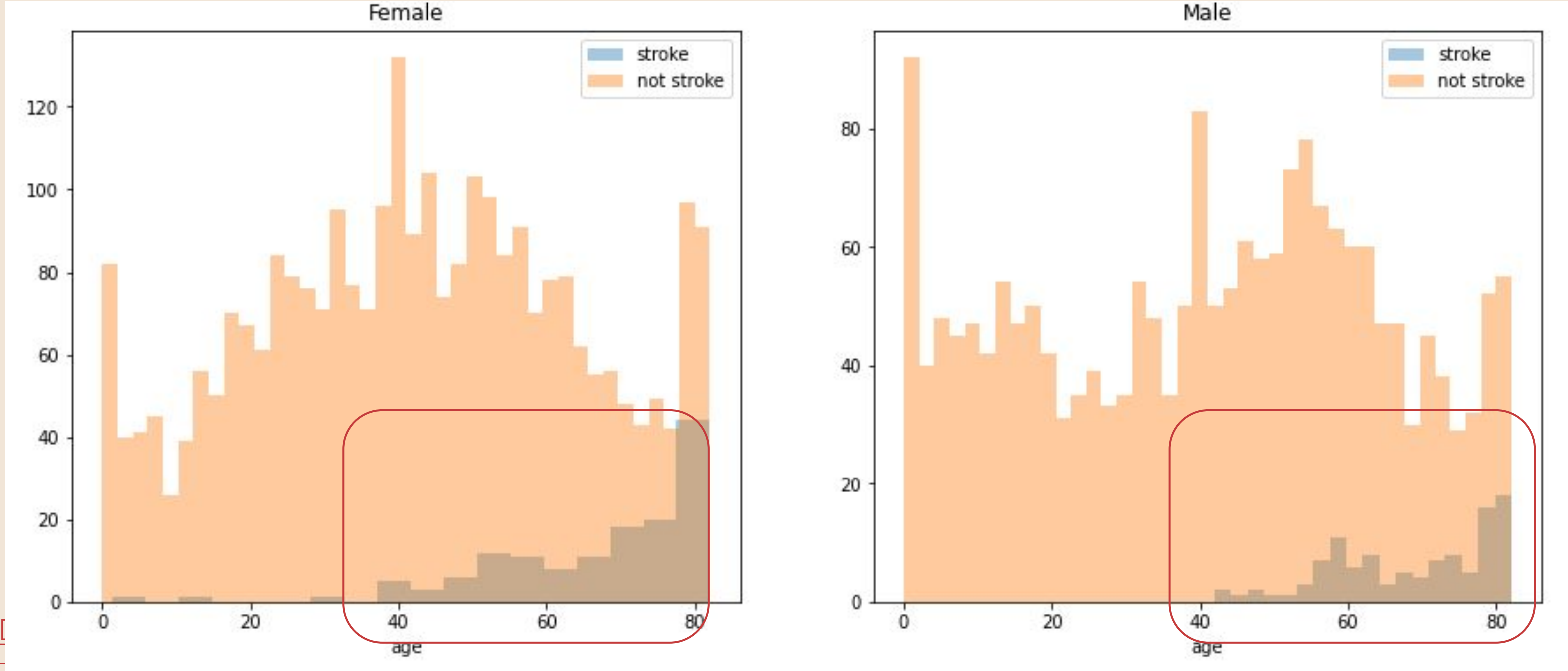
- ★ Residence has balanced value 50%: 50% and has stratified data value of stroke no stroke value
- ★ We can conclude that a person whether live in urban or rural they have same probability in stroke with the same age





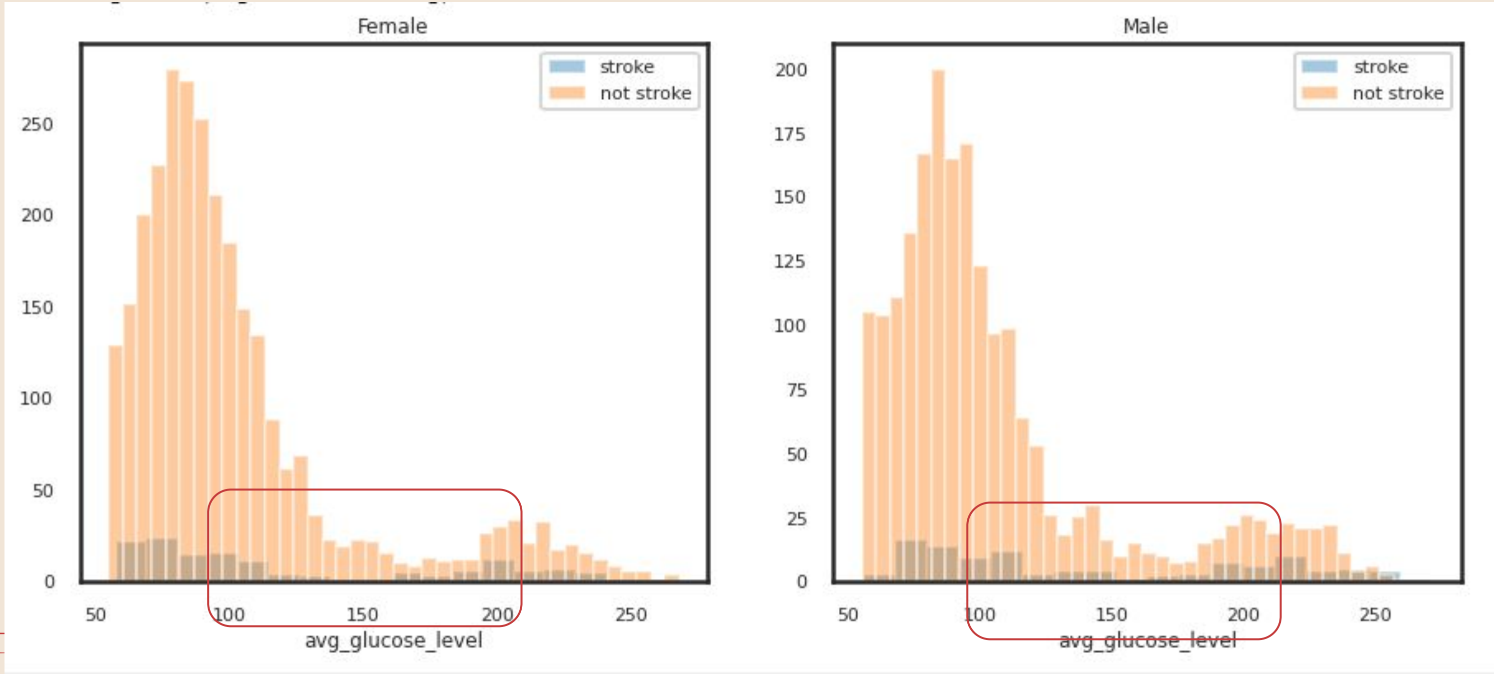
Distribution data

Insights: the older a person, the more possibility they get stroke disease for both gender



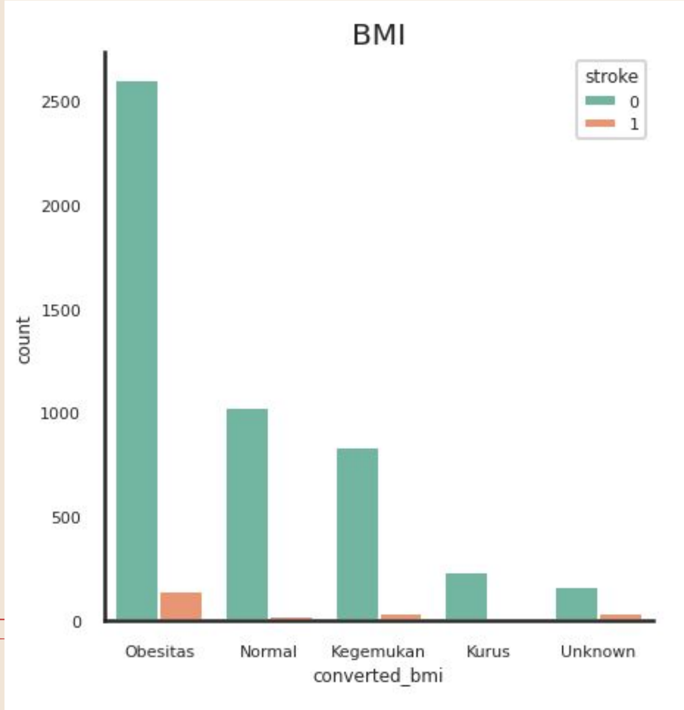
Distribution data

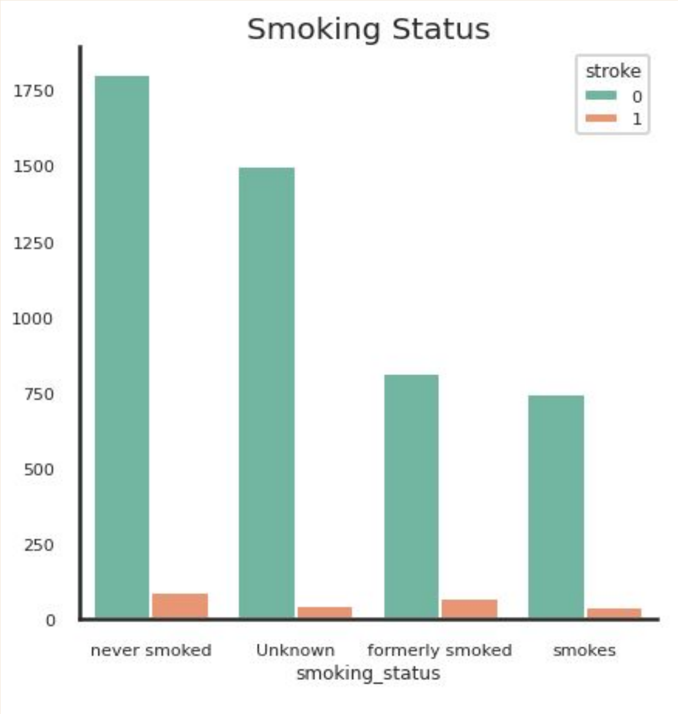
Insights: for both gender, the chance of getting stroke is lower if you have avg glucose level between 100-200



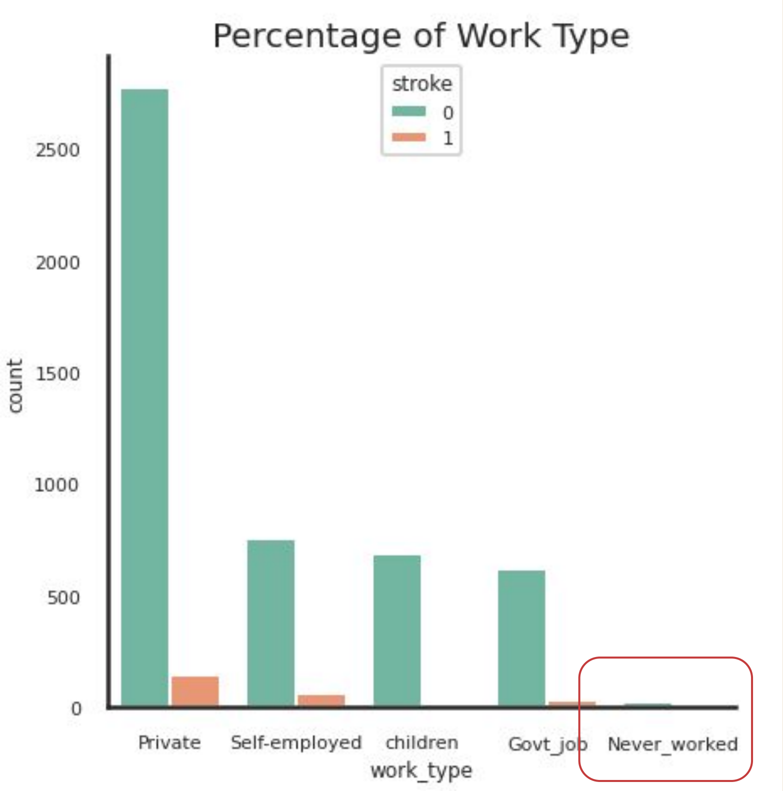
Distribution data

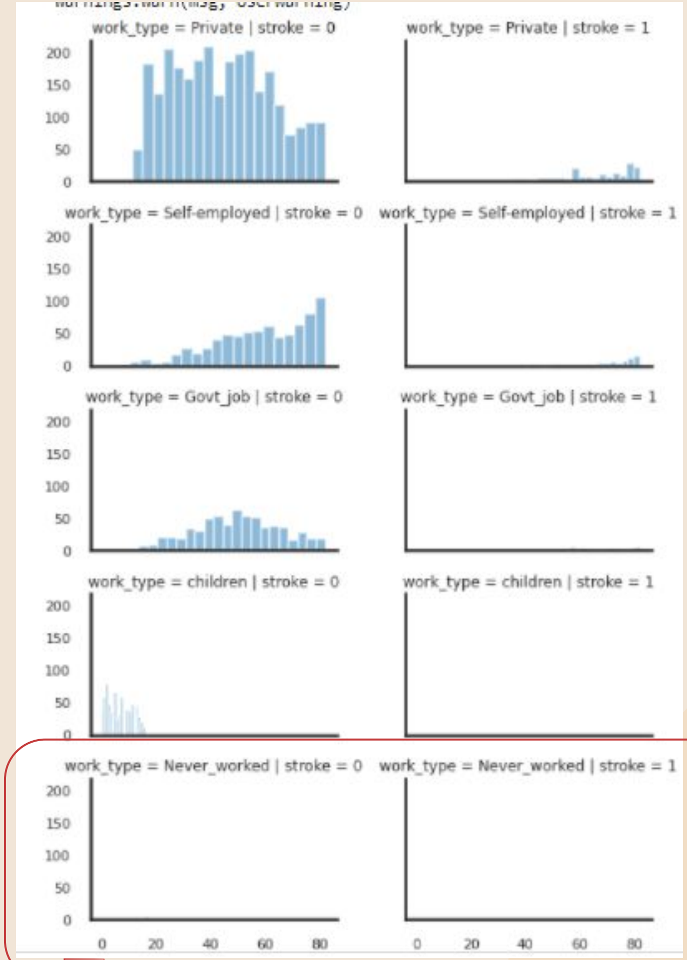
- ★ If you’re categorized “normal” or “kurus” you’ll have lower chance of getting stroke
- ★ In our data “never smoked” is the highest volume, but it has the same number of stroke as “formerly stroke” while it is only half of the “never smoked”.





Data Composition
Work Type





Cleaning Data

```
from collections import Counter
print(Counter(df['gender']))
print(Counter(df['hypertension']))
print(Counter(df['heart_disease']))
print(Counter(df['ever_married']))
print(Counter(df['work_type']))
print(Counter(df['Residence_type']))
print(Counter(df['smoking_status']))
```

```
Counter({'Female': 2994, 'Male': 2115, 'Other': 1})
Counter({0: 4612, 1: 498})
Counter({0: 4834, 1: 276})
Counter({'Yes': 3353, 'No': 1757})
Counter({'Private': 2925, 'Self-employed': 819, 'children': 687, 'Govt_job': 657, 'Never_worked': 22})
Counter({'Urban': 2596, 'Rural': 2514})
Counter({'never smoked': 1892, 'Unknown': 1544, 'formerly smoked': 885, 'smokes': 789})
```

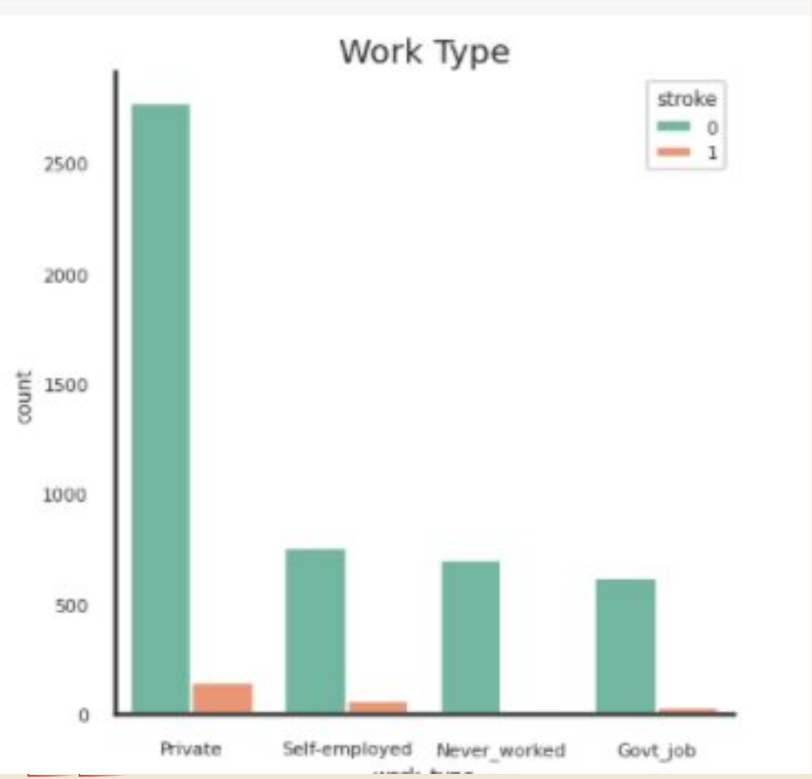
work_type

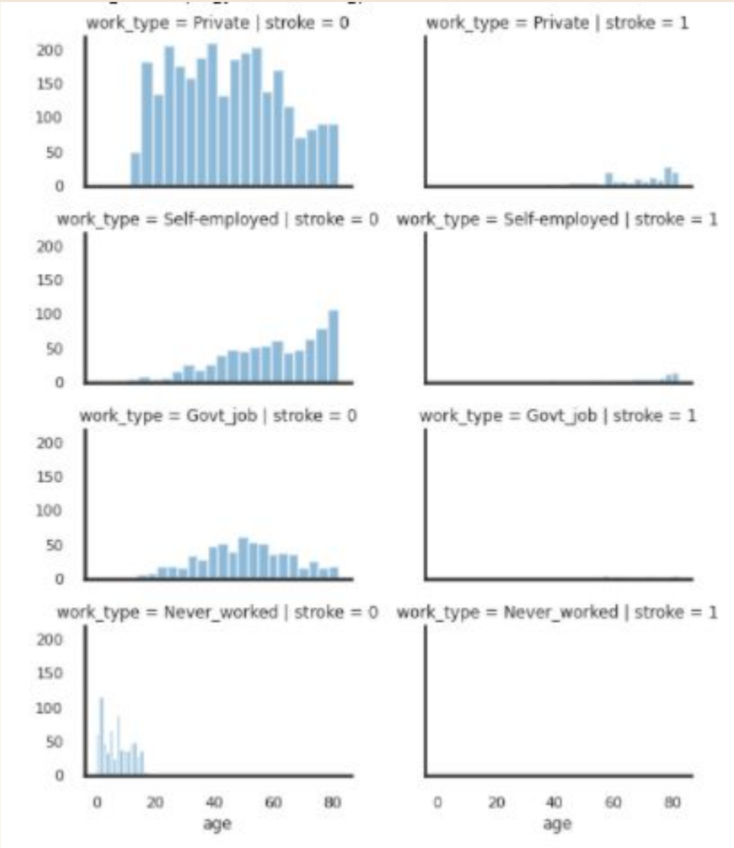
work_type children has same meaning with never_worked, so we change children to never_worked

```
#replace value children menjadi Never_worked
df.replace(to_replace='children',value='Never_worked',inplace=True)
```

Data Composition

Work Type





Cleaning Data

```
#check data
df.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.60000	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	28.89456	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.50000	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.40000	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.00000	never smoked	1

id & bmi

delete id & bmi column

```
df.drop(columns=['id','bmi'],inplace=True)
```

Cleaning Data

```
#changing text to binary
df['gender']=df['gender'].map({'Male':0,'Female':1})
# df['smoking_status'] = df['smoking_status'].map({'formerly smoked':0, 'never smoked':1, 'smokes':2, 'Unknown':3})
df['ever_married']= df['ever_married'].map({'No':0,'Yes':1})
df['Residence_type']=df['Residence_type'].map({'Urban':0,'Rural':1})
```

Change string to int type from columns : gender ,every_married & Residence_type

String	Int
Male	0
female	1

String	Int
No	0
Yes	1

String	Int
Urban	0
Rudal	1

One Hot Encoder

Transforming Data Categorical to Binary

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.60000	formerly smoked	1
1	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	28.89456	never smoked	1
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.50000	never smoked	1
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.40000	smokes	1
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.00000	never smoked	1

ever_married	Residence_type	avg_glucose_level	stroke	work_type_Govt_job	work_type_Never_worked	work_type_Private	work_type_Self-employed	smoking_status_Unknown	smoking_status_formerly smoked	smoking_status_never smoked	smoking_status
1	0	228.69	1	0	0	1	0	0	1	0	
1	1	202.21	1	0	0	0	1	0	0	1	
1	1	105.92	1	0	0	1	0	0	0	1	
1	0	171.23	1	0	0	1	0	0	0	0	
1	1	174.12	1	0	0	0	1	0	0	1	

Cleaning Data

Delete columns unknown

```
df= df.drop(['smoking_status_Unknown', 'converted_bmi_Unknown'], axis=1)
df.head()
```

After one hot encoder data , we delete columns 'smoking_status_Unknown' and 'converted_bmi_Unknown'
Because in actual we don’t know the smoking_status and converted_bmi

Status of “smoking_status_unknown” will be:

Smoking_status_formerly	Smoking_status_never_smoked	Smoking_status_smokes
0	0	0

Standardization Data

Data standardization is about making sure that data is internally consistent; that is, each data type has the same content and format.

```
#standarization data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
scaled_data= pd.DataFrame(scaled_data,columns = df.columns)
scaled_data
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Female	gender_Male	ever_married_No	ever_married_Yes	work_type_Gc
0	1.051242	-0.328637	4.184599	2.706450	1.001034e+00	4.417926	-1.189791	1.189791	-0.723678	0.723678	-0
1	0.785889	-0.328637	-0.238972	2.121652	1.384627e-15	4.417926	0.840484	-0.840484	-0.723678	0.723678	-0
2	1.626174	-0.328637	4.184599	-0.004867	4.683922e-01	4.417926	-1.189791	1.189791	-0.723678	0.723678	-0
3	0.255182	-0.328637	-0.238972	1.437473	7.152261e-01	4.417926	0.840484	-0.840484	-0.723678	0.723678	-0
4	1.581949	3.042866	-0.238972	1.501297	-6.358651e-01	4.417926	0.840484	-0.840484	-0.723678	0.723678	-0

Data Processing

Separate Future Data and Target data :

```
#separate future data and target data
X=df.drop(['stroke'],1)
y=df['stroke']
```

Spitting Data set :

```
#splitting dataset
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.2, stratify=y, random_state=0)
```

handling imbalanced dataset :

```
#handling imbalance dataset
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_smote, y_smote = smote.fit_resample(X_train,y_train)
```

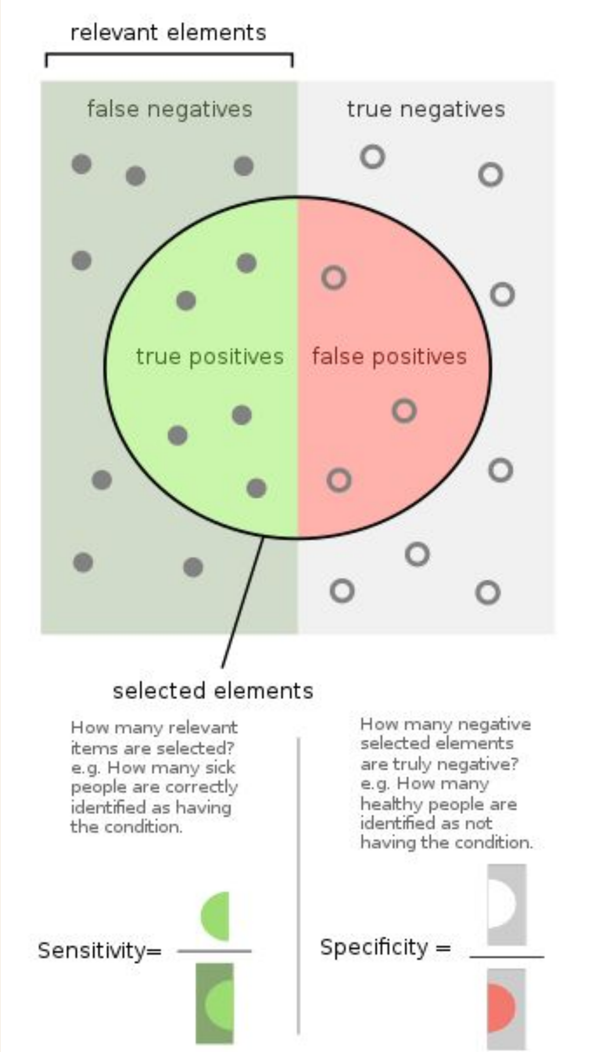
Evaluation Metric

On this case, accuracy is less important than sensitivity & specificity

Sensitivity/ True Positive Rate (TPR) is how much true positives we can get from all positive patient

Specificity/ True Negative Rate (TNR) is how much true negatives we can get from all negative patient

The highlight is we should detect true positives as much as we can
But if we're trying to increase sensitivity, the model will mark positive as many as possible, then the false positives will increase, hence specificity will be decrease (trade-off)



Evaluation Metric

Example:
Total patient checked daily in a hospital = 1000
True positives stroke = 50
True negatives stroke = 950

Sensitivity: 90%
- True positives: 45
- False negatives: 5

Specificity : 60%
- True negatives: 570
- False positives: 380

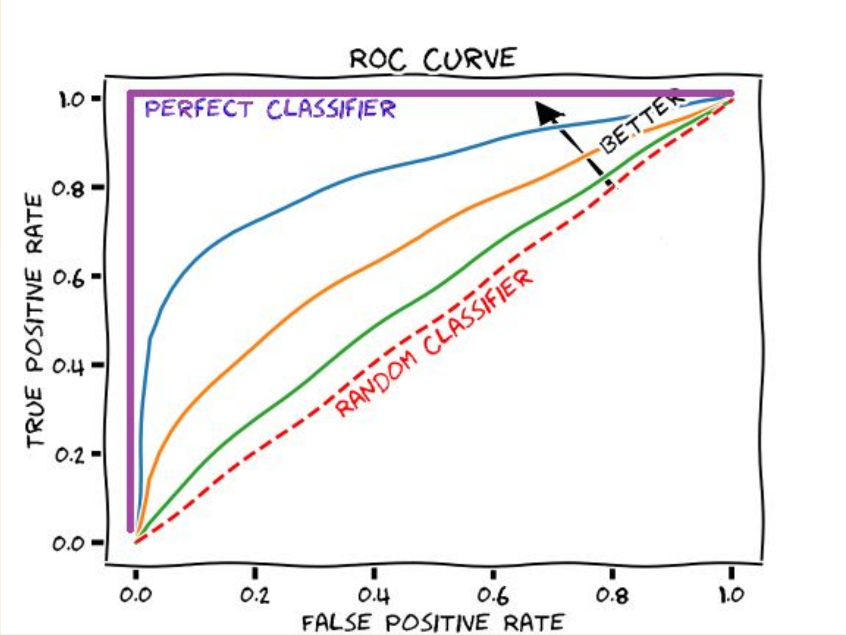
425 person will be hospitalized/ getting treatment

This is bad situation since the real patient will get less treatment due to the capacity

Evaluation Metric

So we'll use **ROC score** instead to find best model

The higher the ROC score, the model is better at differentiating true and false data



Model

Model Naïve bayes

Sensitivity: 90.0
Specificity: 50.82
Accuracy: 52.74
ROC score train: 83.84
ROC score test: 78.45

Model Random forest

Sensitivity: 2.0
Specificity: 99.69
Accuracy: 94.91
ROC score train: 100.0
ROC score test: 74.9

Model Logistic Regression

Sensitivity: 68.0
Specificity: 75.41
Accuracy: 75.05
ROC score train: 87.5
ROC score test: 81.38

Model

Model KNeigboars

Sensitivity: 56.00
Specificity: 77.26
Accuracy: 76.22
ROC score train: 99.19
ROC score test: 70.56

Model Decision tree

Sensitivity: 8.0
Specificity: 93.72
Accuracy: 89.53
ROC score train: 100.0
ROC score test: 50.86

Model SVC

Sensitivity: 70.0
Specificity: 70.88
Accuracy: 70.84
ROC score train: 85.51
ROC score test: 79.62

Choosing Model

Choose the 2 best Model

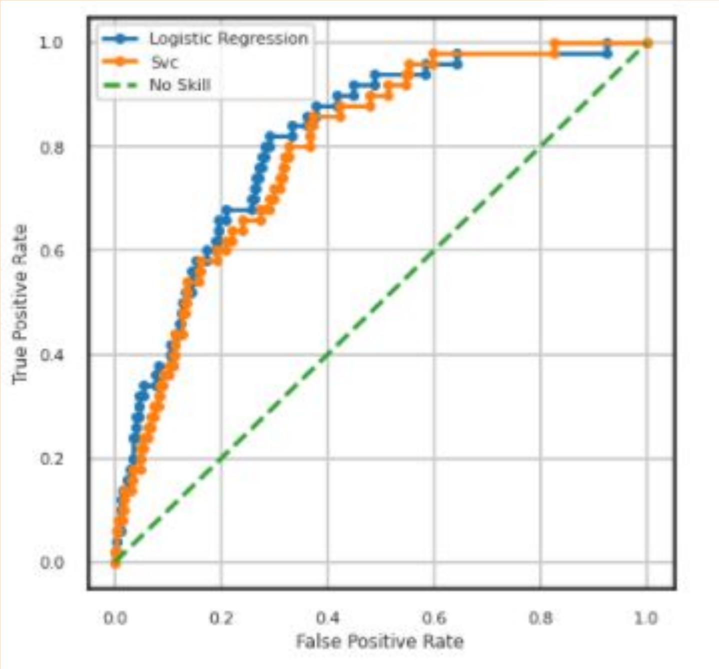
Model :
- Logistic Regression
- Model SVC

Next Process

Hyperparameter Tuning Model

	Model	ROC_Score_test	ROC_Score_train	Sensitivity	Specificity	Accuracy
2	model_logistic_regression	81.38	87.50	68.0	75.41	75.05
5	model_svc	79.62	85.51	70.0	70.88	70.84
0	model_naive_bayes	78.45	83.84	90.0	50.82	52.74
1	model_random_forest	74.90	100.00	2.0	99.69	94.91
3	model_knn	70.56	99.19	56.0	77.26	76.22
4	model_Desicion tree	50.86	100.00	8.0	93.72	89.53

Graph ROC Best Model



Hyperparameter Tuning Model

Hyperparameter Model LogisticRegression

Before:
Sensitivity: 68.0
Specificity: 75.41
Accuracy: 75.05
ROC score train: 87.5
ROC score test: 81.38

After:
Sensitivity: 68.0
Specificity: 75.51
Accuracy: 75.15
ROC score train: 87.54
ROC score test: 81.38

Hyperparameter Model SVC

Before:
Sensitivity: 70.0
Specificity: 70.88
Accuracy: 70.84
ROC score train: 85.51
ROC score test: 79.62

After:
Sensitivity: 18.0
Specificity: 93.42
Accuracy: 89.73
ROC score train: 99.63
ROC score test: 65.22

Conclusion

Model

Hence, our model is **logistic regression**, with model:

LogisticRegression(C=78.47599703514607, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='auto', n_jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm_start=False)

Getting: ROC train score 87.54%, ROC test score 81.38%, sensitivity 68.0%, specificity 75.51% and accuracy 75.15%

Confusion Matrix

	precision	recall	f1-score	support
0	0.98	0.76	0.85	972
1	0.12	0.68	0.21	50
accuracy			0.75	1022
macro avg	0.55	0.72	0.53	1022
weighted avg	0.94	0.75	0.82	1022


Thankyou!