STROKE CLASSIFICATION

1.0 IMPORT LIBRARIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

2.0 DATA EXTRACTION,TRANSFORM AND LOAD (ETL) AND EXPLORATORY

- Data extracted manually form Kaggle
- Transform and Load by Pandas(row and columns)

```
df = pd.read_csv('stroke_data.csv')
df
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_
0	Male	58.0	1	0	Yes	Private	U
1	Female	70.0	0	0	Yes	Private	F
2	Female	52.0	0	0	Yes	Private	U
3	Female	75.0	0	1	Yes	Self-employed	F
4	Female	32.0	0	0	Yes	Private	F
...	...	...	...	...	...	...	
29060	Female	10.0	0	0	No	children	U
29061	Female	56.0	0	0	Yes	Govt_job	U
29062	Female	82.0	1	0	Yes	Private	U
29063	Male	40.0	0	0	Yes	Private	U
29064	Female	82.0	0	0	Yes	Private	U

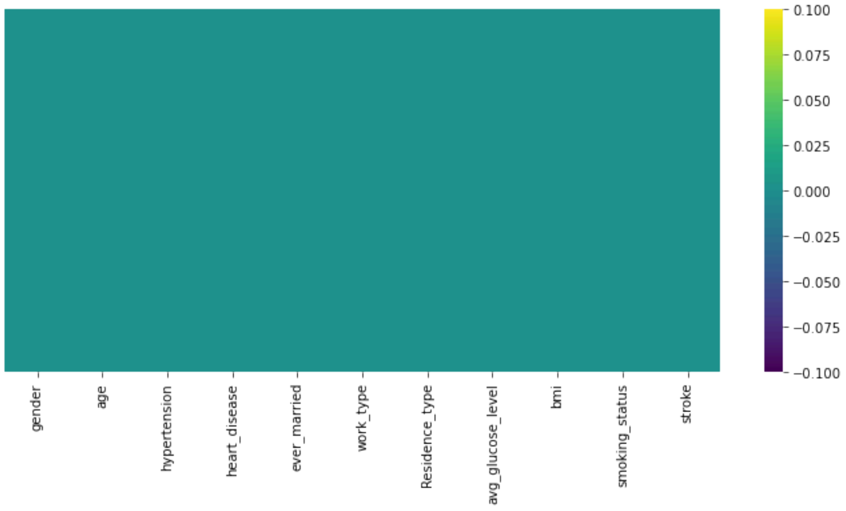
3.0 EXPLORATORY DATA ANALYSIS

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29065 entries, 0 to 29064
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             29065 non-null  object
 1   age                29065 non-null  float64
 2   hypertension       29065 non-null  int64
 3   heart_disease      29065 non-null  int64
 4   ever_married       29065 non-null  object
 5   work_type          29065 non-null  object
 6   Residence_type     29065 non-null  object
 7   avg_glucose_level  29065 non-null  float64
 8   bmi                29065 non-null  float64
 9   smoking_status     29065 non-null  object
 10  stroke             29065 non-null  int64
dtypes: float64(3), int64(3), object(5)
memory usage: 2.4+ MB
```

```
plt.figure(figsize=(12,5))
sns.heatmap(df.isnull(), yticklabels=False, cbar =True, cmap='viridis')
```

```
<AxesSubplot:>
```

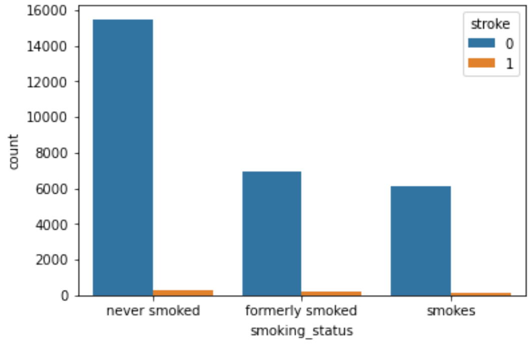



```
df.isnull().any()
```

```
gender               False
age                  False
hypertension         False
heart_disease        False
ever_married         False
work_type            False
Residence_type       False
avg_glucose_level    False
bmi                  False
smoking_status       False
stroke               False
dtype: bool
```

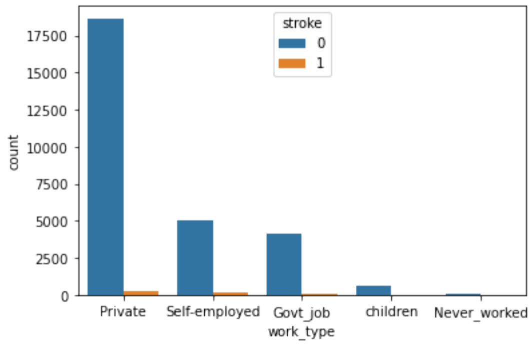
```
sns.countplot(x= 'smoking_status', hue ='stroke', data=df )
```

```
<AxesSubplot:xlabel='smoking_status', ylabel='count'>
```



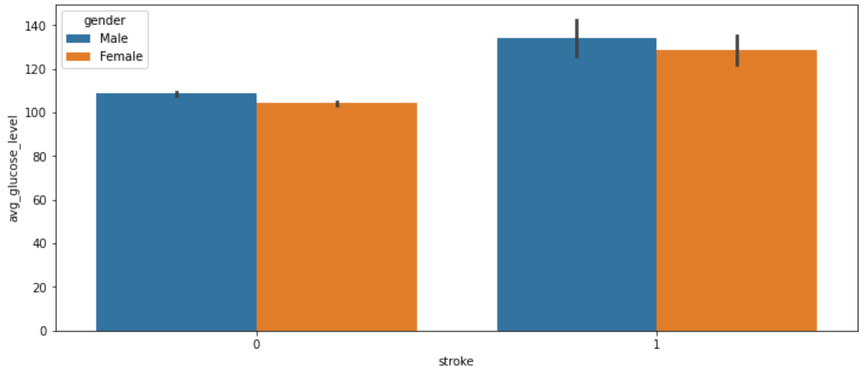

```
sns.countplot(x= 'work_type', hue ='stroke', data=df )
```

```
<AxesSubplot:xlabel='work_type', ylabel='count'>
```



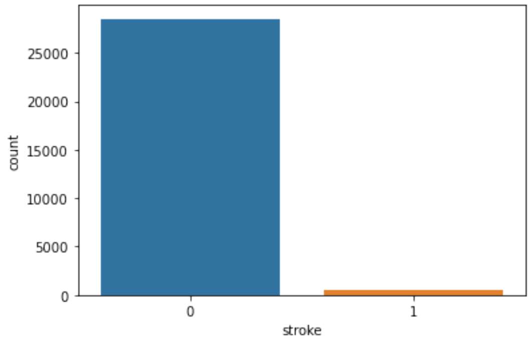

```
# plt.plot(,, color = 'red')
plt.figure(figsize=(12,5))

sns.barplot(x=df['stroke'], y=df['avg_glucose_level'], hue=df['gender'] )
```

<AxesSubplot:xlabel='stroke', ylabel='avg_glucose_level'>




```
sns.countplot(x= 'stroke',data=df )
```

<AxesSubplot:xlabel='stroke', ylabel='count'>




```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29065 entries, 0 to 29064
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             29065 non-null  object 
 1   age                29065 non-null  float64
 2   hypertension       29065 non-null  int64  
 3   heart_disease      29065 non-null  int64  
 4   ever_married       29065 non-null  object 
 5   work_type          29065 non-null  object 
 6   Residence_type     29065 non-null  object 
 7   avg_glucose_level  29065 non-null  float64
 8   bmi                29065 non-null  float64
 9   smoking_status     29065 non-null  object 
 10  stroke             29065 non-null  int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 2.4+ MB
```

TREAT THE Imbalance Dataset

- count_class_0, count_class_1 = df.stroke.value_counts()
- count_class_0, count_class_1

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

TO USE UNDER SAMPLE

- df_class_0_under = df_class_0.sample(count_class_1)
- df_test_under = pd.concat([df_class_0_under, df_class_1], axis=0)
- df_test_under
- df_test_under.stroke.value_counts()

TO USE OVERSAMPLE

+count_class_0, count_class_1 `# Using 10% of your`

- n_10_per = int(count_class_0 * 0.30)
- print('30% of OVERSAMPLE DATA ====> ', n_10_per)
- df_class_0_over = df_class_0.sample(n_10_per)
- df_class_1_over = df_class_1.sample(n_10_per, replace=True)
- df_sample_overr = pd.concat([df_class_0_over, df_class_1_over], axis=0)
- df_sample_overr

`Imbalance dataset late treated using SMOTE library`

4.0 TRANSFORMATION

```
# Import Principal Component Analysis
#from sklearn.decomposition import PCA
```

Double-click (or enter) to edit

```
Start coding or generate with AI.
```

```
# Print all object types column
for col in df:
    if df[col].dtypes == 'object':
        print(col)
```

```
gender
ever_married
work_type
Residence_type
smoking_status
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Double-click (or enter) to edit

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
#TODO:  Check for Object type column, get there dummies values, reduce there demensionality to 1,
df_transformed = pd.DataFrame()
for col in df:
    # Check for obj type
    if df[col].dtypes == 'object':
        #  generate dummies values fpr the column
        dummy_val = pd.get_dummies(df[col], drop_first=True )
        # Add the new column df_transformed
        df_transformed = pd.concat([df_transformed, dummy_val], axis=1)
df_transformed
```

	Male	Yes	Never_worked	Private	Self-employed	children	Urban	never smoked	smokes
0	1	1	0	1	0	0	1	1	0
1	0	1	0	1	0	0	0	0	0
2	0	1	0	1	0	0	1	0	0
3	0	1	0	0	1	0	0	1	0
4	0	1	0	1	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...
29060	0	0	0	0	0	1	1	1	0
29061	0	1	0	0	0	0	1	0	0
29062	0	1	0	1	0	0	1	0	0
29063	1	1	0	1	0	0	1	1	0
29064	0	1	0	1	0	0	1	1	0

29065 rows × 9 columns

```
 #pd.concat([df_transformed, pd.get_dummies(df_test_under['gender'])], axis=1, )
#pd.concat([df_transformed, pd.get_dummies(df_test_over['gender'])], axis=1 )
```

```
# Set the index of df_transformed to match the index of df_clean_outlier

#df_transformed.index = df_sample_over.index
df_transformed.index = df.index
```

```
df_transformed.tail(3)
```

	Male	Yes	Never_worked	Private	Self-employed	children	Urban	never smoked	smokes
29062	0	1	0	1	0	0	1	0	0
29063	1	1	0	1	0	0	1	1	0
29064	0	1	0	1	0	0	1	1	0

```
# concanteenate newly generate and origin dataset
df_clean_transformed = pd.concat([df, df_transformed], axis=1)
df_clean_transformed
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_
0	Male	58.0	1	0	Yes	Private	U
1	Female	70.0	0	0	Yes	Private	F
2	Female	52.0	0	0	Yes	Private	U
3	Female	75.0	0	1	Yes	Self-employed	F
4	Female	32.0	0	0	Yes	Private	F
...	...	...	...	...	...	...	
29060	Female	10.0	0	0	No	children	U
29061	Female	56.0	0	0	Yes	Govt_job	U
29062	Female	82.0	1	0	Yes	Private	U
29063	Male	40.0	0	0	Yes	Private	U
29064	Female	82.0	0	0	Yes	Private	U

29065 rows × 20 columns

```
# Drop the object type column
df_clean_transformed.drop(['gender', 'work_type', 'Residence_type', 'ever_married', 'smoking_status'], axis=1, inplace=True)
df_clean_transformed.sample(4)
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	Male	Yes
16199	76.0	0	0	69.19	21.2	0	1	1
12642	35.0	0	0	140.00	32.4	0	1	0
854	71.0	0	0	151.30	26.3	0	0	1

Using SMOTE from imbalanced-learn to balance the DataSet

```
!pip install imbalanced-learn
from imblearn.over_sampling import SMOTE
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: imbalanced-learn in c:\users\ola\appdata\roaming\python\python39\site-packages (0.11.0)
Requirement already satisfied: scipy>=1.5.0 in c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (1.7.3)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (1.0.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\ola\appdata\roaming\python\python39\site-packages (from imbalanced-learn)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (2.2.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\ola\appdata\roaming\python\python39\site-packages (from imbalanced-learn)
```

```
df_clean_transformed.shape
```

```
(29065, 15)
```

```
count_class_0, count_class_1 = df.stroke.value_counts()
count_class_0, count_class_1
```

```
(28517, 548)
```

```
# Use 20% of Oversample data
n_20_per = int(count_class_0 * 0.20)

print( '20% of OVERSAMPLE DATA ====> ', n_20_per)
df_class_0 = df_clean_transformed[df_clean_transformed['stroke'] == 0]
df_class_1 = df_clean_transformed[df_clean_transformed['stroke'] == 1]

# Select 20% of the sample
df_class_0_over = df_class_0.sample(n_20_per)

# Concatenate class 1 and 0
df_clean_transformed_sample = pd.concat([df_class_0_over, df_class_1], axis=0)
df_clean_transformed_sample
```

```
20% of OVERSAMPLE DATA ====>  5703
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	Male	Yes
15342	52.0	0	1	157.90	38.0	0	1	1
18426	26.0	0	0	96.85	20.4	0	0	1
26028	62.0	0	0	94.89	31.2	0	1	1
25322	63.0	0	0	116.46	36.7	0	0	1
9104	61.0	0	0	190.35	34.3	0	0	1
...	...	...	...	...	...	...	...	...
28863	79.0	0	1	88.29	36.0	1	1	1
28891	76.0	0	0	93.38	26.7	1	1	1
28910	56.0	0	0	83.27	32.9	1	0	1
29004	80.0	0	0	75.91	26.7	1	0	1
29014	62.0	1	1	77.97	31.5	1	1	1

```
X = df_clean_transformed_sample.drop('stroke', axis=1)  # independent column
y = df_clean_transformed_sample['stroke']
```

```
y.value_counts()
```

```
0    5703
1     548
Name: stroke, dtype: int64
```

```
X.shape   # dependant column
```

```
(6251, 14)
```

```
# Balance the dataset using SMOTE and increase the dataset
smote = SMOTE(sampling_strategy = 'minority')
X_sm, y_sm = smote.fit_resample(X,y)
y_sm.value_counts()
```

```
0    5703
1    5703
Name: stroke, dtype: int64
```

```
X_sm.shape
```

```
(11406, 14)
```

5.0 MODEL AND EVALUATION

- Let's start by splitting our data into a training set and test set

Splitting the dataset into the Training set and Test set

```
df_clean_transformed.shape
```

```
(29065, 15)
```

```
from sklearn.model_selection import train_test_split
```

```
Start coding or generate with AI.
```

```
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=10, stratify=y)
X_train, X_test, y_train, y_test = train_test_split(X_sm, y_sm, test_size=0.20, random_state=10, stratify=y_sm)
```

```
X_test.head(17)
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	Male	Yes
2113	31.000000	0	0	97.330000	30.300000	0	1
3177	77.000000	0	0	81.390000	37.200000	0	1
4196	76.000000	0	0	77.670000	40.500000	0	1
974	61.000000	0	0	74.930000	42.600000	0	1
8024	64.429281	0	0	149.578809	28.355169	1	1
11310	77.576308	0	0	116.515218	27.650799	0	1
11188	78.086590	0	0	93.521540	22.591341	1	1
5952	67.000000	0	0	58.050000	31.300000	1	1
9683	71.377975	0	0	101.992015	29.140178	1	0
2102	57.000000	0	0	64.460000	30.300000	1	1
9718	78.105684	0	0	65.423979	22.195116	0	1
4005	55.000000	0	0	81.250000	27.400000	1	1
5501	56.000000	0	0	82.840000	28.600000	1	1
5063	64.000000	1	0	210.810000	28.400000	0	1
11263	80.000000	0	0	64.661847	43.660545	0	0
9694	79.273879	0	0	188.238193	26.065594	0	1

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

Train using LOGISTIC_REGRESSION model on the Training set

```
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0)
```

Making the Confusion Matrix

```
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[932 209]
 [220 921]]
0.8120070113935145
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.82      0.81      1141
           1       0.82      0.81      0.81      1141

    accuracy                           0.81      2282
   macro avg       0.81      0.81      0.81      2282
weighted avg       0.81      0.81      0.81      2282
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Train using K-NEAREST_NEIGHBORS model on the Training set

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn_classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

Making the Confusion Matrix

```
knn_y_pred = knn_classifier.predict(X_test)
knn_cm = confusion_matrix(y_test, knn_y_pred)
print(knn_cm)
accuracy_score(y_test, knn_y_pred)
```

```
[[ 949  192]
 [ 106 1035]]
0.8694127957931639
```

```
Start coding or generate with AI.
```

```
print(classification_report(y_test, knn_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.83      0.86      1141
           1       0.84      0.91      0.87      1141
```

```
    accuracy                           0.87      2282
   macro avg       0.87      0.87      0.87      2282
weighted avg       0.87      0.87      0.87      2282
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Train using SUPPORT_VECTOR_MACHINE model on the Training set

```
from sklearn.svm import SVC
svc_classifier = SVC(kernel = 'linear', random_state = 42)
svc_classifier.fit(X_train, y_train)
```

```
SVC(kernel='linear', random_state=42)
```

Making the CONFUSION MATRIX

```
svc_y_pred = svc_classifier.predict(X_test)
svc_cm = confusion_matrix(y_test, svc_y_pred)
print(svc_cm)
accuracy_score(y_test, svc_y_pred)
```

```
[[934 207]
 [216 925]]
0.8146362839614374
```

```
print(classification_report(y_test, svc_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.82      0.82      1141
           1       0.82      0.81      0.81      1141

    accuracy                           0.81      2282
   macro avg       0.81      0.81      0.81      2282
weighted avg       0.81      0.81      0.81      2282
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Train using KERNEL_SVM model on the Training set

```
from sklearn.svm import SVC
kernel_classifier = SVC(kernel ='rbf', random_state = 40)
kernel_classifier.fit(X_train, y_train)
```

```
SVC(random_state=40)
```

Making the CONFUSION MATRIX

```
kernel_y_pred = kernel_classifier.predict(X_test)
kernel_cm = confusion_matrix(y_test, kernel_y_pred)
print(kernel_cm)
accuracy_score(y_test, kernel_y_pred)
```

```
[[940 201]
 [159 982]]
0.8422436459246275
```

```
print(classification_report(y_test, kernel_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.82      0.84      1141
           1       0.83      0.86      0.85      1141
```

```
    accuracy                           0.84      2282
   macro avg       0.84      0.84      0.84      2282
weighted avg       0.84      0.84      0.84      2282
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Train using GAUSSIAN_NB model on the Training set

```
from sklearn.naive_bayes import GaussianNB
gaussian_classifier = GaussianNB()
gaussian_classifier.fit(X_train, y_train)
```

```
GaussianNB()
```

Making the CONFUSION MATRIX

```
gaussian_y_pred = gaussian_classifier.predict(X_test)
gaussian_cm = confusion_matrix(y_test, gaussian_y_pred)
print(gaussian_cm)
accuracy_score(y_test, gaussian_y_pred)
```

```
[[  39 1102]
 [   0 1141]]
0.5170902716914987
```

```
print(classification_report(y_test, gaussian_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.03      0.07      1141
           1       0.51      1.00      0.67      1141

    accuracy                           0.52      2282
   macro avg       0.75      0.52      0.37      2282
weighted avg       0.75      0.52      0.37      2282
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Train using DECISION TREE model on the Training set

```
from sklearn.tree import DecisionTreeClassifier
decision_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 2)
decision_classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=2)
```

Making the CONFUSION MATRIX

```
decision_y_pred = decision_classifier.predict(X_test)
decision_cm = confusion_matrix(y_test, decision_y_pred)
print(decision_cm)
accuracy_score(y_test, decision_y_pred)
```

```
[[ 971  170]
 [ 115 1026]]
0.8751095530236634
```

```
print(classification_report(y_test, decision_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.85      0.87      1141
           1       0.86      0.90      0.88      1141

    accuracy                           0.88      2282
```

```
   macro avg       0.88      0.88      0.88      2282
weighted avg       0.88      0.88      0.88      2282
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Train using RANDOM FOREST model on the Training set

```
from sklearn.ensemble import RandomForestClassifier
random_classifier = RandomForestClassifier(n_estimators=10, criterion = 'entropy', random_state = 42)
random_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=42)
```

Making the CONFUSION MATRIX

```
random_y_pred = random_classifier.predict(X_test)
random_cm = confusion_matrix(y_test, random_y_pred)
print(random_cm)
accuracy_score(y_test, random_y_pred)
```

```
[[1028  113]
 [ 120 1021]]
0.8978965819456617
```

```
print(classification_report(y_test, random_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      1141
```