

Research Question: Can 9 classifiers be built with 75% or more accuracy with the given hospital dataset?

Data: The dataset for this analysis is publicly available information provided by Data.gov. The original data set contained 5110 rows and 12 columns.

The data set includes the following variables: "id", "gender", "age", "hypertension", "heart_disease", "ever_married", "work_type", "Residence_type", "avg_glucose_level", "bmi", "smoking_status", "stroke".

There is no information that would make the hospitals associated with this analysis identifiable.

Limitations: The dataset is limited to the patient population included in the dataset. There are no delimitations to this study.

```
In [1]: #imports
import numpy as np
import pandas as pd

import os

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.patches import Patch
from matplotlib.lines import Line2D
plt.rc("font", size=14)
get_ipython().run_line_magic('matplotlib', 'inline')

import seaborn as sns

import scipy

from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, r
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from imblearn.over_sampling import SMOTE

from colorama import Fore, Back, Style
```

```
In [2]: #package/Library versions
print("pandas version " + pd.__version__)
```

```

print("numpy version " + np.__version__)
print("scipy version " + scipy.__version__)
print("matplotlib version " + matplotlib.__version__)
print("seaborn version " + sns.__version__)

pandas version 1.4.2
numpy version 1.21.5
scipy version 1.7.3
matplotlib version 3.5.1
seaborn version 0.11.2

```

In [3]:

```
#ignore future warnings
import warnings
warnings.filterwarnings('ignore')
```

In [4]:

```
#Load data into pandas Dataframe
df = pd.read_csv("stroke_dataset.csv")
```

In [5]:

```
#view top of data
df.head()
```

Out[5]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level
0	9046	Male	67.0	0	1	Yes	Private	Urban	117.0
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	117.0
2	31112	Male	80.0	0	1	Yes	Private	Rural	117.0
3	60182	Female	49.0	0	0	Yes	Private	Urban	117.0
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	117.0

In [6]:

```
df.shape
```

Out[6]:

```
(5110, 12)
```

In [7]:

```
#check for duplicates
df.duplicated()
```

Out[7]:

```

0      False
1      False
2      False
3      False
4      False
...
5105    False
5106    False
5107    False
5108    False
5109    False
Length: 5110, dtype: bool

```

In [8]:

```
#check for null values
print(df.isnull().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
dtype: int64
```

```
In [9]: #impute null 'bmi' values with mean
df["bmi"] = df["bmi"].fillna(df["bmi"].mean())
```

```
In [10]: #confirm imputation worked
print(df.isnull().sum())
```

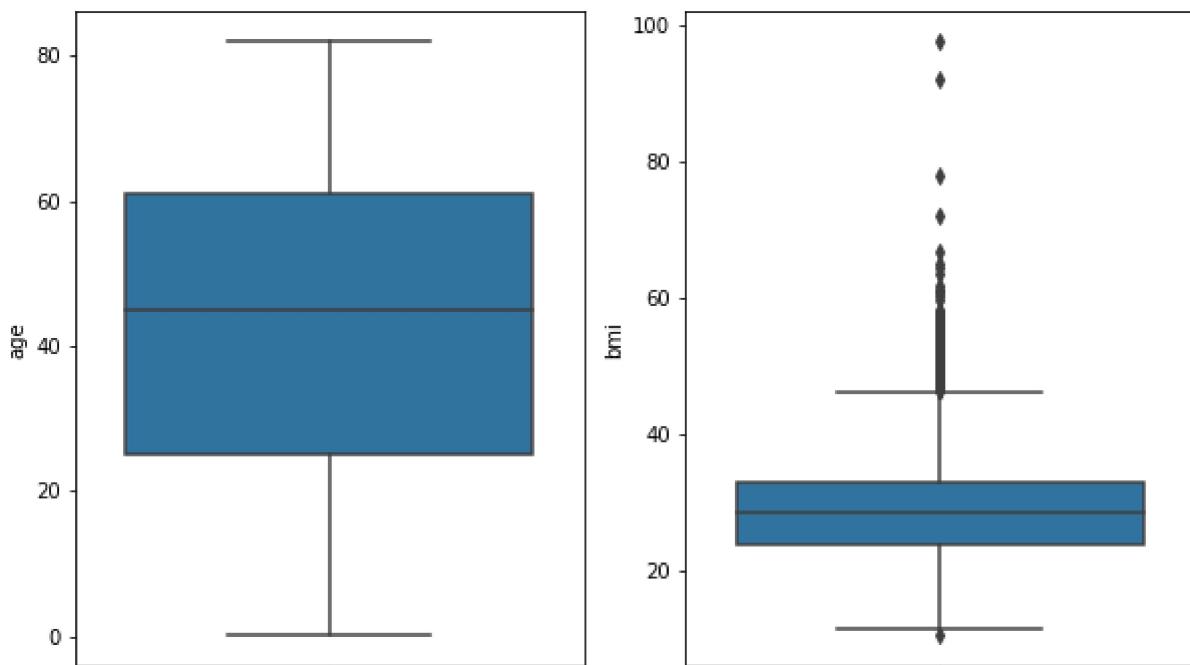
```
id          0
gender      0
age         0
hypertension 0
heart_disease 0
ever_married 0
work_type    0
Residence_type 0
avg_glucose_level 0
bmi         0
smoking_status 0
stroke       0
dtype: int64
```

```
In [11]: #check for outliers in age and bmi
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (10, 6) , squeeze=True)

sns.boxplot(data=df,y=df['age'],palette='tab10' , ax=axes[0])
sns.boxplot(data=df,y=df['bmi'],palette='tab10' , ax=axes[1])

plt.show
```

```
Out[11]: <function matplotlib.pyplot.show(close=None, block=None)>
```



```
In [12]: #address outliers in 'bmi'

#display rows with 'bmi' > 70
display(df[df['bmi'] > 70])

#drop rows greater than 70
df.drop(df.index[df['bmi'] > 70], inplace=True)

#reset index of dataframe
df = df.reset_index(drop = True)
```

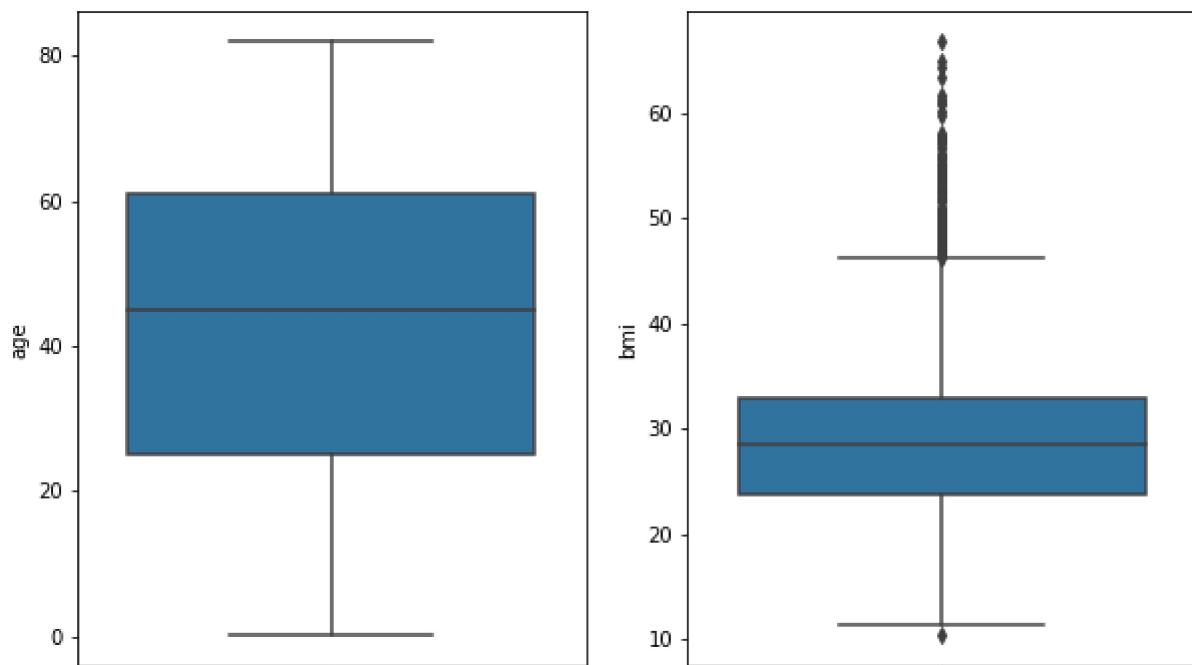
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_bmi
544	545	Male	42.0	0	0	Yes	Private	Rural	
928	41097	Female	23.0	1	0	No	Private	Urban	
2128	56420	Male	17.0	1	0	No	Private	Rural	
4209	51856	Male	38.0	1	0	Yes	Private	Rural	

```
In [13]: #viewing outliers after addressing outliers in 'bmi'
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (10, 6) , squeeze=True)

sns.boxplot(data=df,y=df['age'],palette='tab10' , ax=axes[0])
sns.boxplot(data=df,y=df['bmi'],palette='tab10' , ax=axes[1])

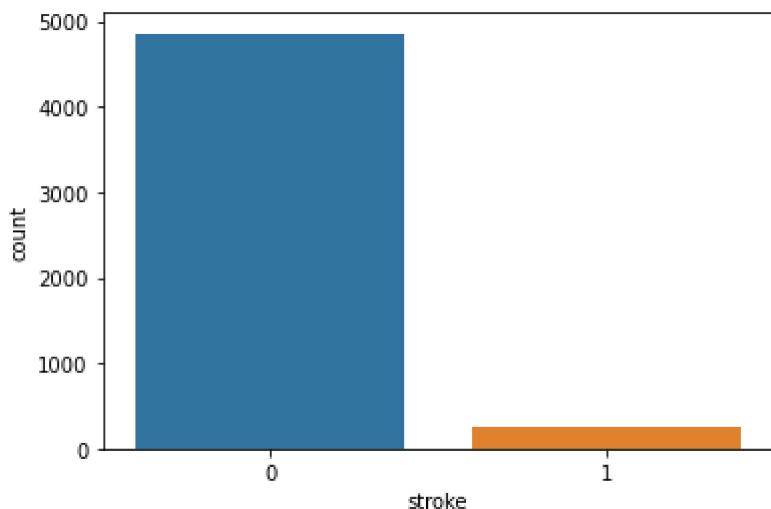
plt.show
```

Out[13]: <function matplotlib.pyplot.show(close=None, block=None)>



```
In [14]: #delete id column as it's not necessary for analysis
df = df.drop("id",axis=1)
```

```
In [15]: #checking dataset balance
sns.countplot(df['stroke'], label="Count")
plt.show()
```



Result: data is not balanced

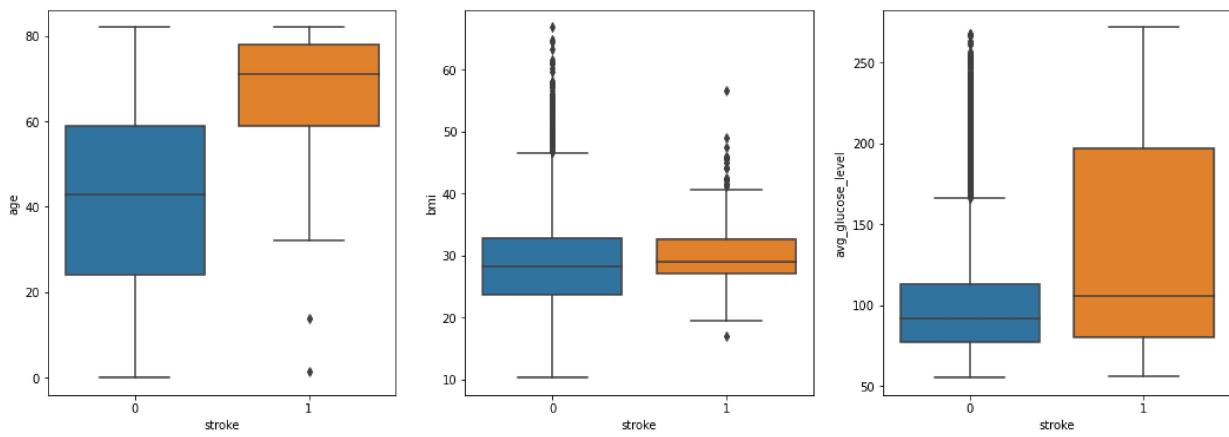
Data Visualization

```
In [16]: #boxplot
fig, axes = plt.subplots(nrows = 1, ncols = 3, figsize = (18, 6) , squeeze=True)

sns.boxplot(data=df,y=df['age'],x=df['stroke'],palette='tab10' , ax=axes[0])
sns.boxplot(data=df,y=df['bmi'],x=df['stroke'],palette='tab10' , ax=axes[1])
sns.boxplot(data=df,y=df['avg_glucose_level'],x=df['stroke'],palette='tab10' , ax=axes[2])

plt.show
```

Out[16]: <function matplotlib.pyplot.show(close=None, block=None)>



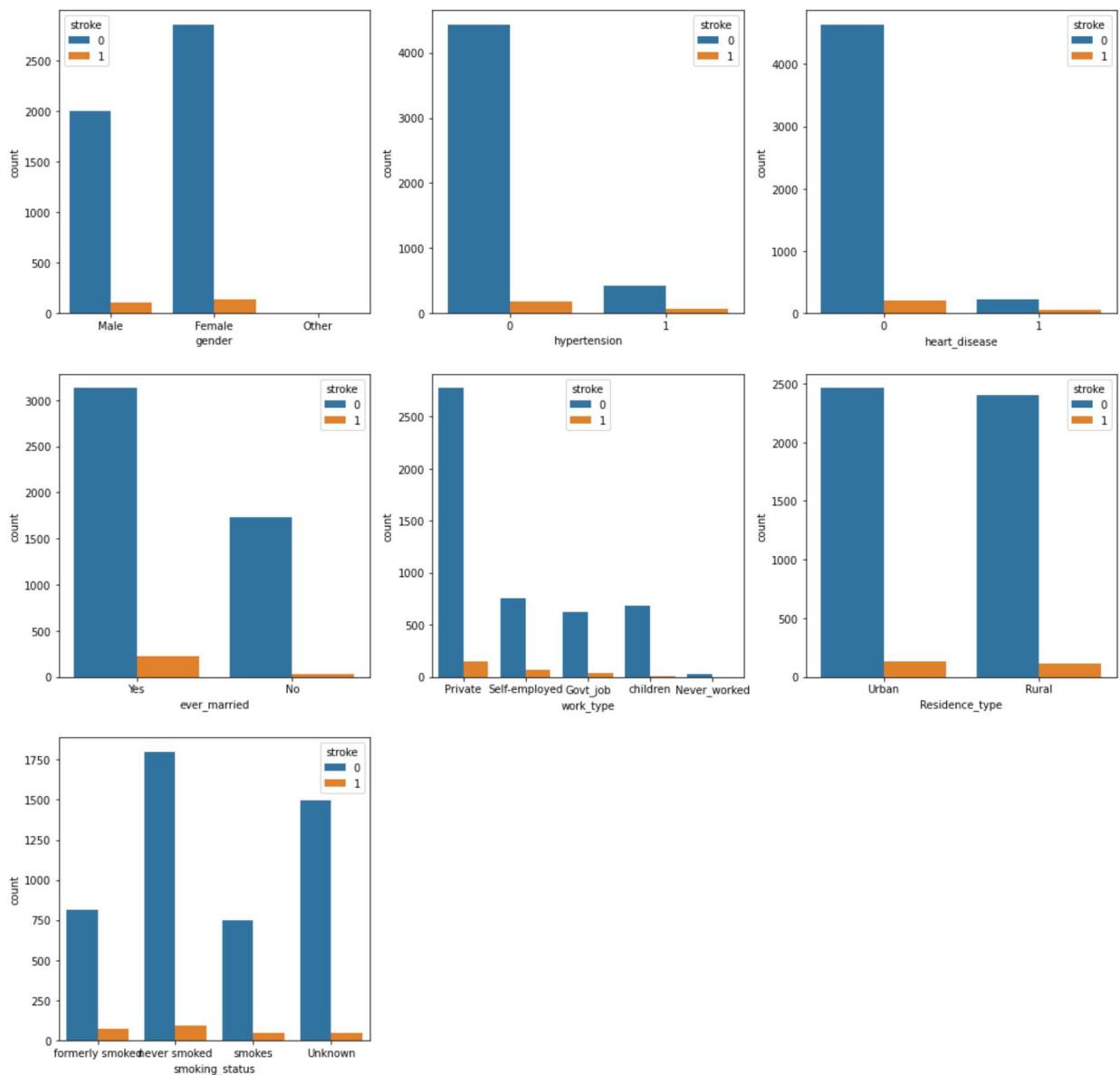
In [17]: #histograms

```
fig, axes = plt.subplots(nrows = 3, ncols = 3, figsize = (18, 18))
fig.delaxes( ax=axes[2,1])
fig.delaxes( ax=axes[2,2])
sns.countplot(x="gender", hue='stroke', palette='tab10', data=df , ax=axes[0,0])
sns.countplot(x="hypertension", hue='stroke', palette="tab10", data=df , ax=axes[0,1])
sns.countplot(x="heart_disease", hue='stroke', palette="tab10", data=df , ax=axes[0,2])

sns.countplot(x="ever_married", hue='stroke', palette="tab10", data=df , ax=axes[1,0])
sns.countplot(x="work_type", hue='stroke', palette="tab10", data=df , ax=axes[1,1])
sns.countplot(x="Residence_type", hue='stroke', palette="tab10", data=df , ax=axes[1,2])

sns.countplot(x="smoking_status", hue='stroke', palette="tab10", data=df , ax=axes[2,0])
plt.show()
```

Stroke Data



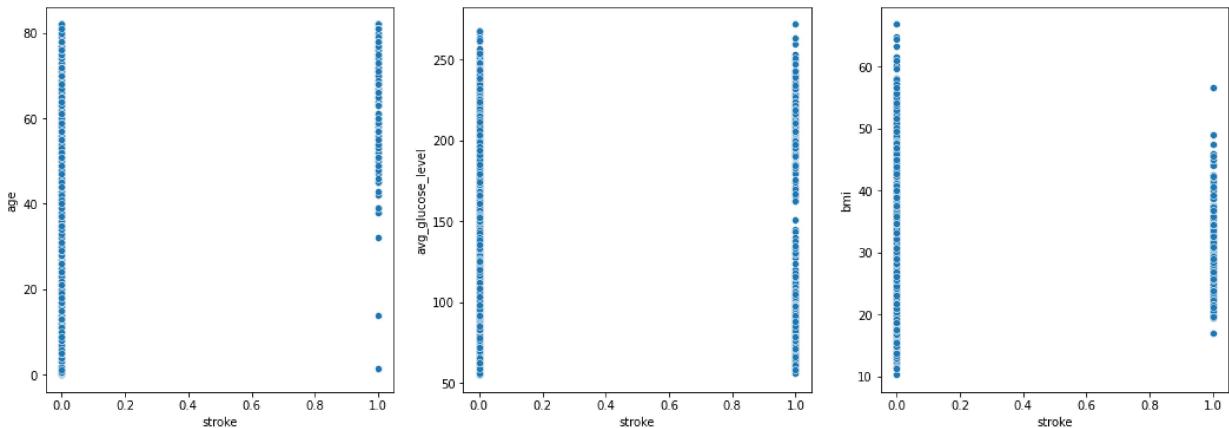
```
In [18]: #scatterplots
fig, axes = plt.subplots(nrows = 1, ncols = 3, figsize = (18, 6) , squeeze=True)

sns.scatterplot(data=df, x=df['stroke'], y=df['age'],palette='tab10', ax=axes[0])
sns.scatterplot(data=df, x=df['stroke'], y=df['avg_glucose_level'],palette='tab10', ax=axes[1])
sns.scatterplot(data=df, x=df['stroke'], y=df['bmi'],palette='tab10',ax=axes[2] )

plt.show
```

```
Out[18]: <function matplotlib.pyplot.show(close=None, block=None)>
```

Stroke Data



Model building

```
In [19]: #categorical columns in the training data
object_cols = [col for col in df.columns if df[col].dtype == "object"]

print('Categorical columns that will be ordinal encoded:', object_cols)
```

Categorical columns that will be ordinal encoded: ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']

```
In [20]: #get number of unique entries in each column with categorical data
object_nunique = list(map(lambda col: df[col].nunique(), object_cols))

d = dict(zip(object_cols, object_nunique))

#print number of unique entries by column, in ascending order
sorted(d.items(), key=lambda x: x[1])
```

```
Out[20]: [('ever_married', 2),
 ('Residence_type', 2),
 ('gender', 3),
 ('smoking_status', 4),
 ('work_type', 5)]
```

```
In [21]: ordinal_encoder = OrdinalEncoder()
df[object_cols] = ordinal_encoder.fit_transform(df[object_cols])
```

```
In [22]: #oversampling the data

smote = SMOTE()
test_df = df[['gender', 'age', 'hypertension', 'heart_disease', 'work_type', 'avg_glucose_level']]
train_df = df.drop(index=test_df.index)

X_test, y_test = test_df[['gender', 'age', 'hypertension', 'heart_disease', 'work_type']]
X_train, y_train = train_df[['gender', 'age', 'hypertension', 'heart_disease', 'work_type']]

X_train, y_train = smote.fit_resample(X_train, y_train)
upsampled_df = X_train.assign(Stroke = y_train)

X_test, y_test = smote.fit_resample(X_test, y_test)
up_test_df = X_test.assign(Stroke = y_test)
```

```
In [23]: fig, axes = plt.subplots(nrows=1, ncols=3, dpi=100, figsize=(15, 5))
```

```
df.stroke.value_counts().plot(kind='bar', color='tab:red', title='Stroke - Before Upsampling')
upsampled_df.Stroke.value_counts().plot(kind='bar', color='tab:green', title='Stroke(Train set) - After Upsampling')
up_test_df.Stroke.value_counts().plot(kind='bar', color='tab:blue', title='Stroke (Test set) - After Upsampling')
```



In [24]: `#performance comparison visualization`

```
def score_vis(score):

    names = ['Naive Bayes', 'SVM', 'Decision Tree', 'Random Forest', 'Logistic Regression']

    plt.rcParams['figure.figsize']=20,8
    ax = sns.barplot(x=names, y=score, palette = "plasma", saturation = 2.0)
    plt.xlabel('Model', fontsize = 20 )
    plt.ylabel('Accuracy(%)', fontsize = 20)
    plt.title('Model Comparison - Test set', fontsize = 20)
    plt.xticks(fontsize = 12, horizontalalignment = 'center', rotation = 8)
    plt.yticks(fontsize = 12)
    for i in ax.patches:
        width, height = i.get_width(), i.get_height()
        x, y = i.get_xy()
        ax.annotate(f'{round(height,2)}%', (x + width/2, y + height*1.02), ha='center')
    plt.show()
```

In [25]: `def trainer(X_train, y_train, X_test, y_test):`

```
models= [['Naive Bayes', GaussianNB()],
         ['SVM', SVC()],
         ['Decision Tree', DecisionTreeClassifier()],
         ['Random Forest', RandomForestClassifier()],
         ['Logistic Regression', LogisticRegression(max_iter=200)],
         ['AdaBoost', AdaBoostClassifier()],
         ['XGBoost', XGBClassifier()],
         ['CatBoost', CatBoostClassifier(logging_level='Silent')],
         ['KNN', KNeighborsClassifier()]]
```

```
scores = []

for model_name, model in models:

    model = model
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    cm_model = confusion_matrix(y_test, pred)
    scores.append(accuracy_score(y_test, model.predict(X_test)))
```

```
print(Back.YELLOW + Fore.BLACK + Style.BRIGHT + model_name)
print(Back.RESET)
print(cm_model)
print('\n' + Fore.BLUE + 'Training Acc. : ' + Fore.GREEN + str(round(accuracy))
print(Fore.BLUE + 'Validation Acc.: ' + Fore.RED + str(round(accuracy_score(y_
print(Fore.CYAN + classification_report(y_test, pred)))
print('\n' + Fore.BLACK + Back.WHITE + '*****')

return scores
```

In [26]: *#results*
scores = trainer(X_train, y_train, X_test, y_test)

Naive Bayes

```
[[662 297]
 [113 846]]
```

Training Acc. : 78.78%
Validation Acc.: 78.62%

	precision	recall	f1-score	support
0	0.85	0.69	0.76	959
1	0.74	0.88	0.80	959
accuracy			0.79	1918
macro avg	0.80	0.79	0.78	1918
weighted avg	0.80	0.79	0.78	1918

```
*****
```

SVM

```
[[666 293]
 [224 735]]
```

Training Acc. : 76.85%
Validation Acc.: 73.04%

	precision	recall	f1-score	support
0	0.75	0.69	0.72	959
1	0.71	0.77	0.74	959
accuracy			0.73	1918
macro avg	0.73	0.73	0.73	1918
weighted avg	0.73	0.73	0.73	1918

```
*****
```

Decision Tree

```
[[891 68]
 [229 730]]
```

Training Acc. : 100.0%
Validation Acc.: 84.52%

	precision	recall	f1-score	support
0	0.80	0.93	0.86	959
1	0.91	0.76	0.83	959
accuracy			0.85	1918
macro avg	0.86	0.85	0.84	1918
weighted avg	0.86	0.85	0.84	1918

```
*****
```

Random Forest

```
[[927  32]
 [235 724]]
```

Training Acc. : 100.0%
Validation Acc.: 86.08%

	precision	recall	f1-score	support
0	0.80	0.97	0.87	959
1	0.96	0.75	0.84	959
accuracy			0.86	1918
macro avg	0.88	0.86	0.86	1918
weighted avg	0.88	0.86	0.86	1918

```
*****
```

Logistic Regression

```
[[706 253]
 [152 807]]
```

Training Acc. : 76.91%
Validation Acc.: 78.88%

	precision	recall	f1-score	support
0	0.82	0.74	0.78	959
1	0.76	0.84	0.80	959
accuracy			0.79	1918
macro avg	0.79	0.79	0.79	1918
weighted avg	0.79	0.79	0.79	1918

```
*****
```

AdaBoost

```
[[874  85]
 [148 811]]
```

Training Acc. : 89.24%
Validation Acc.: 87.85%

	precision	recall	f1-score	support
0	0.86	0.91	0.88	959
1	0.91	0.85	0.87	959
accuracy			0.88	1918
macro avg	0.88	0.88	0.88	1918
weighted avg	0.88	0.88	0.88	1918

```
*****
```

XGBoost

```
[[932 27]
 [190 769]]
```

Training Acc. : 99.55%
Validation Acc.: 88.69%

	precision	recall	f1-score	support
0	0.83	0.97	0.90	959
1	0.97	0.80	0.88	959
accuracy			0.89	1918
macro avg	0.90	0.89	0.89	1918
weighted avg	0.90	0.89	0.89	1918

```
*****
```

CatBoost

```
[[932 27]
 [204 755]]
```

Training Acc. : 98.74%
Validation Acc.: 87.96%

	precision	recall	f1-score	support
0	0.82	0.97	0.89	959
1	0.97	0.79	0.87	959
accuracy			0.88	1918
macro avg	0.89	0.88	0.88	1918
weighted avg	0.89	0.88	0.88	1918

```
*****
```

KNN

```
[[779 180]
 [395 564]]
```

Training Acc. : 91.94%
Validation Acc.: 70.02%

	precision	recall	f1-score	support
0	0.66	0.81	0.73	959
1	0.76	0.59	0.66	959
accuracy			0.70	1918
macro avg	0.71	0.70	0.70	1918
weighted avg	0.71	0.70	0.70	1918

```
*****
```

