Configuration Manual

MSc Research Project
Data Analytics

Livia Anthony Pereira
Student ID: 22158294

School of Computing
National College of Ireland

Supervisor: Arjun Chikkankod

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……………………………Livia Anthony Pereira…………………………………………………

Student ID: ……………………………………x22158294………………………..……………………..

Programme: …….…Data Analytics……………………………… **Year:** ………2023…………..

Module: …………Research Project……………………,,,,,,,…………………………………..………

Lecturer: ………………………..Arjun Chikkankod…………………………………………………

Submission Due Date: ………………14/08/2023…………………………………………………………….………

Project Title: Comparative Modeling of Stroke Prediction Using Advanced Machine Learning Techniques

Word Count: ……………1003……… **Page Count:** ………..14……………………..……..………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ………………..**Livia Anthony Pereira**……………………….………………

Date: ……………14/12/2023…………………………………………….………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Livia Anthony Pereira
Student ID: 22158294

1 Introduction

This Manual contains information on how to run and set up the implementation code for the ongoing study endeavour. This paper offers particular details about the hardware of the computer as well as the applications that need to be used. Users can use the XGBoost, CatBoost, Plotly, and Imblearn models to create summaries of research publications by following the procedures below.

2 System Specification

2.1 Hardware Specification

Following are the hardware specifications of the system that was used to develop the project:

Processor: Intel Core i5 – 1135G7
RAM: 8GB DDR4
Storage: 500GB
Operating System: Windows 11

2.2 Software Specification

The Jupyter Lab a web-based platform was used to train and evaluate the models and its specification was the following:

Processor: Intel Core i5

3 Software Tools

Following are the software tools that were used to implement the project:

3.1 Python

To construct the project, the Python programming language was utilized. Python was chosen mostly because of its useful packages for deep learning models, dataset preparation, and visualization. I installed Python from the home page. The official Python website's download page is displayed in Figure 1.

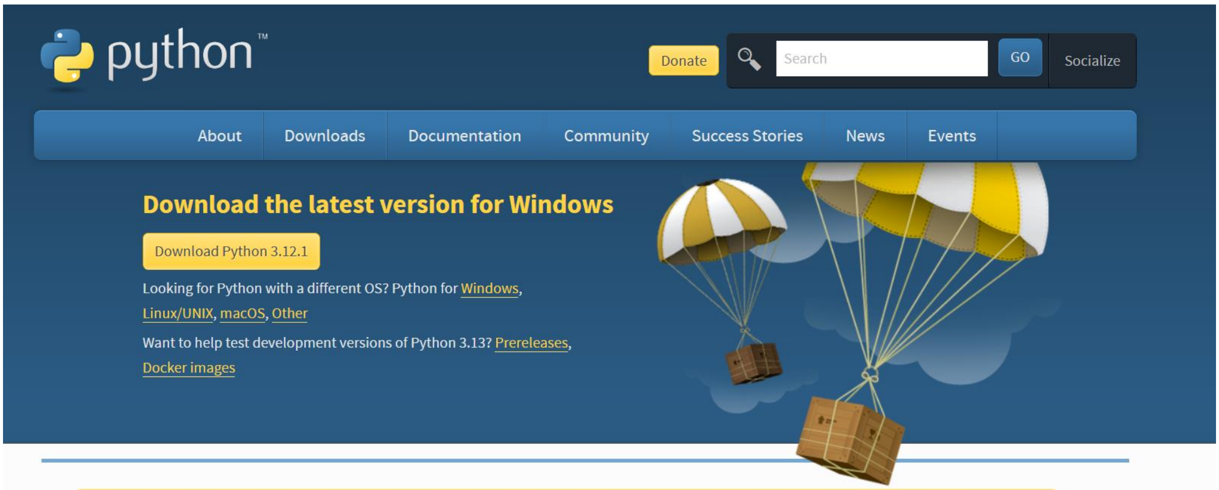


Figure 1: Download page of Python's official website

3.2 Jupyter Notebook

The Jupyter Notebook was utilized as a compiler to run the code because it enables users to implement all of the code in a single location and run the code in sections, such as cells, so that viewers can easily examine each code's output. Figure 2 illustrates the download page for Jupyter Notebook, which may be obtained from its official website

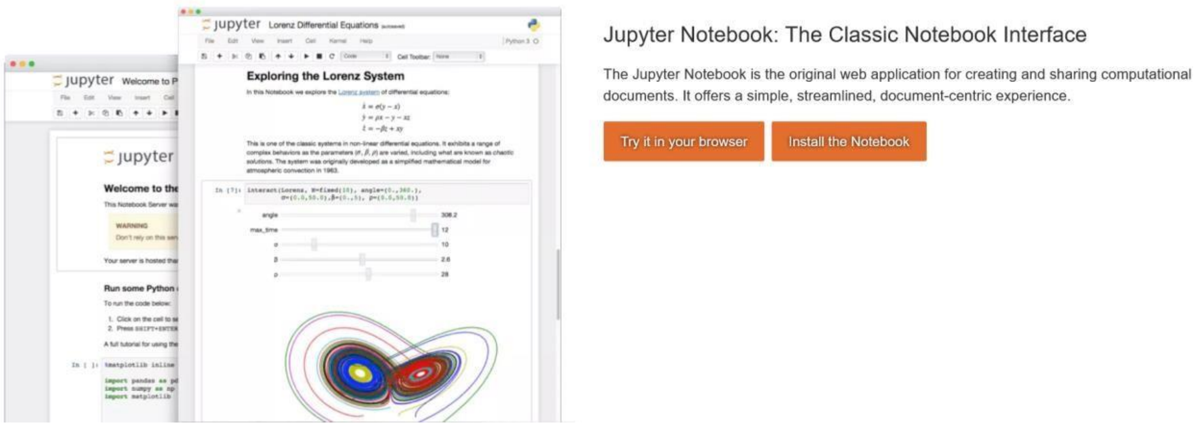


Figure 2: Download page of Jupyter Notebook's official website

4 Project Implementation

Following are the Python packages which were installed using pip and used to implement the project:

- Seaborn
- Pandas
- Numpy
- Matplotlib
- Scikit-Learn

- SMOTETomek
- IMBLearn

- Datasets

```
#Python libraries
#Importing Classic libraries for data ingestion, load, transform and data manipulation
import numpy as np
import pandas as pd
# Plots   (Libraries for plotting the graphs / visualisation)
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
pd.set_option("display.max_rows",None)
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.impute import SimpleImputer
# Modeling Libraries (Libraries for model intialising and evaluation)
from sklearn import preprocessing   # importing preprocessing libraries
from collections import Counter
from imblearn.combine import SMOTETomek
from sklearn.preprocessing import MinMaxScaler, StandardScaler   # importimg scaling libraries
from sklearn.linear_model import LogisticRegression   # importing Logistic Regression
from sklearn.ensemble import RandomForestClassifier   # importing random forest model
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report  # evalution libraries
from sklearn.model_selection import train_test_split   # importing train test split
from sklearn.metrics import precision_score, recall_score, precision_recall_curve,f1_score, fbeta_score,accuracy_score  # evalut
from imblearn.over_sampling import SMOTE  # importing smote for equi sampling of data
from xgboost import XGBClassifier   # importing xgb classifier
from sklearn.ensemble import RandomForestClassifier       #importing ML models
from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import label_binarize
%matplotlib inline
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
from sklearn.model_selection import cross_validate
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve,confusion_matrix,precision_score,recall_score,roc_auc_score,accuracy_score,classification_r
```

Figure 3: Necessary Libraries and Packages

The code generates some descriptive statistics, removes an extra column, then reads a CSV file into a pandas dataframe.as can be seen in Figure 4:

```
In [2]: df = pd.read_csv('Stroke Dataset.csv')      # reading the data from csv file
        df.shape          # shape of data

Out[2]: (43400, 12)

In [3]: df.head()      # printing the some rows of data from start

Out[3]:
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	30669	Male	3.0	0	0	No	children	Rural	95.12	18.0	NaN	0
1	30468	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	16523	Female	8.0	0	0	No	Private	Urban	110.89	17.6	NaN	0
3	56543	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	46136	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	NaN	0

```
In [4]: df = df.drop(['id'],axis=1)    # since id is nothing but index therefore dropping this unnecessary column
        df.shape          # shape of data after dropping column

Out[4]: (43400, 11)

In [5]: df.describe()     # getting statistics of data

Out[5]:
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	43400.000000	43400.000000	43400.000000	43400.000000	41938.000000	43400.000000
mean	42.217894	0.093571	0.047512	104.482750	28.605038	0.018041
std	22.519649	0.291235	0.212733	43.111751	7.770020	0.133103
min	0.080000	0.000000	0.000000	55.000000	10.100000	0.000000
25%	24.000000	0.000000	0.000000	77.540000	23.200000	0.000000
50%	44.000000	0.000000	0.000000	91.580000	27.700000	0.000000
75%	60.000000	0.000000	0.000000	112.070000	32.900000	0.000000
max	82.000000	1.000000	1.000000	291.050000	97.600000	1.000000

Figure 4: Loading and Checking the Dataset

To be able to compare the null percentages before and after imputation, the code in the image identifies columns in a pandas DataFrame which contain null values, imputes those values, and then generates a violin plot.which can be seen in the following Figure 5:

```
In [7]: df = df.drop_duplicates()         # drooping duplicates if any
        df.shape  # shape of data aftre dropping duplicates
```

```
Out[7]: (43400, 11)
```

```
In [8]: # Calculate null percentage for each column
        null_percentage = (df.isnull().sum() / len(df)) * 100
        print('Null Percentage in Each Column \n', null_percentage)
```

```
Null Percentage in Each Column
 gender                0.000000
age                   0.000000
hypertension          0.000000
heart_disease         0.000000
ever_married          0.000000
work_type             0.000000
Residence_type        0.000000
avg_glucose_level     0.000000
bmi                   3.368664
smoking_status       30.626728
stroke                0.000000
dtype: float64
```

```
In [9]: null_percentage_df = null_percentage.reset_index()
        null_percentage_df.columns = ['Columns', 'Null_Percentage']
```

```
In [10]: # plotting the graph to show columns having null/nan
         dark_palette = px.colors.qualitative.Dark24
         fig = px.box(null_percentage_df, y='Null_Percentage', x='Columns', color='Columns',title="Null Percentage Vs Columns",color_discr
         fig.show()
```

```
In [11]: # Identifying columns with null values
         columns_with_null = df.columns[df.isnull().any()]

         # Creating a DataFrame with columns 'Columns' and 'Null_Percentage'
         null_percentage_df = pd.DataFrame({'Columns': columns_with_null, 'Null_Percentage': (df[columns_with_null].isnull().sum() / len(d

         # Imputing null values using SimpleImputer with strategy='constant' and fill_value='random'
         simple_imputer = SimpleImputer(strategy='most_frequent')
         df[columns_with_null] = simple_imputer.fit_transform(df[columns_with_null])

         # Verifying that there are no null values after imputation
         null_percentage_imputed_df = pd.DataFrame({'Columns': columns_with_null, 'Null_Percentage': (df[columns_with_null].isnull().sum()

         # Create a violin plot to compare null percentages before and after imputation
         fig = px.violin(
             null_percentage_df,
             y='Null_Percentage',
             x='Columns',
             title="Null in Columns Before Imputation",
             color_discrete_sequence=px.colors.qualitative.Dark24
         )

         fig2 = px.violin(
             null_percentage_imputed_df,
             y='Null_Percentage',
             x='Columns',
             title="Nulls in Columns After Imputation",
             color_discrete_sequence=px.colors.qualitative.Dark24
         )

         fig.show()
         fig2.show()
```

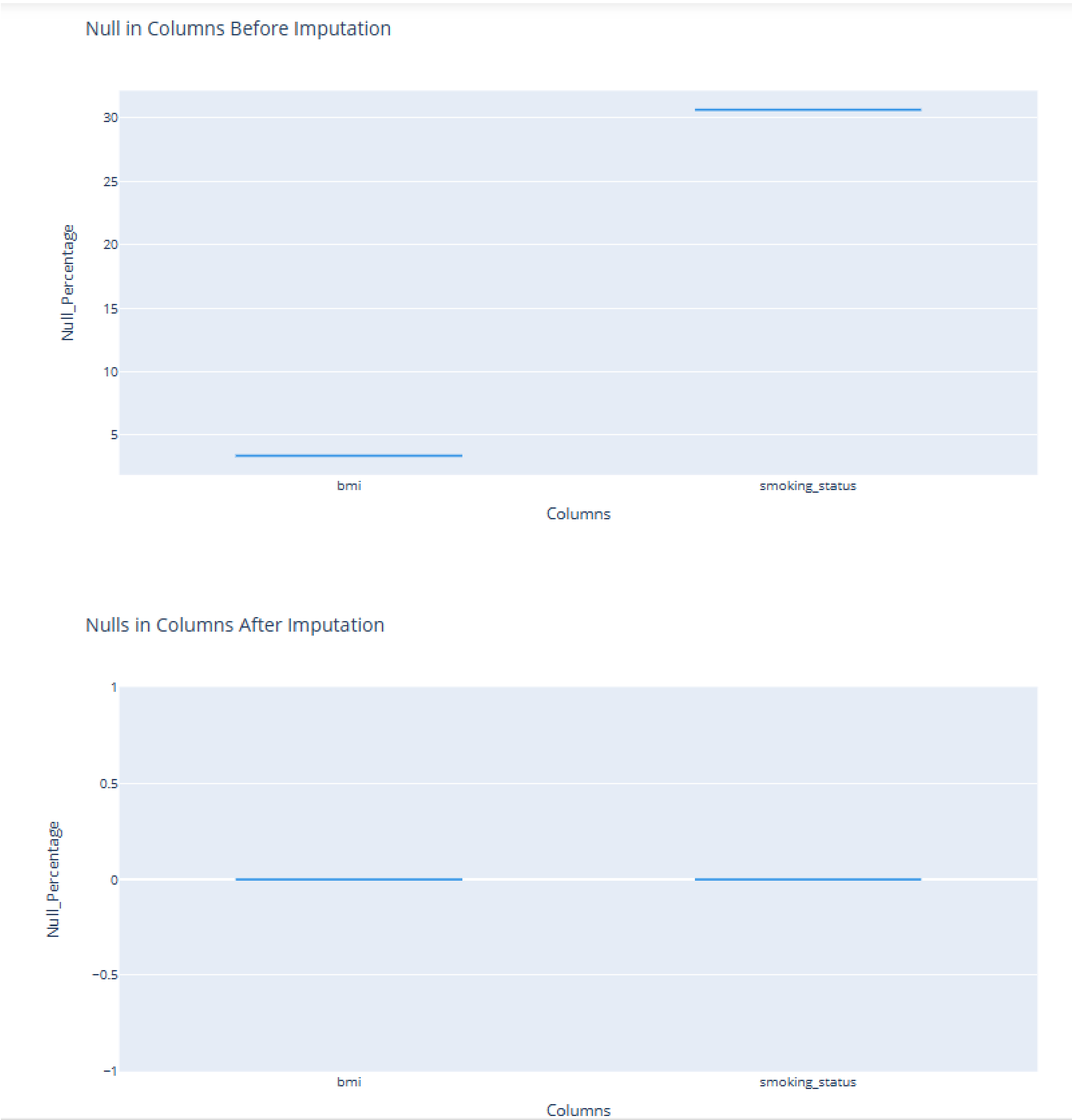
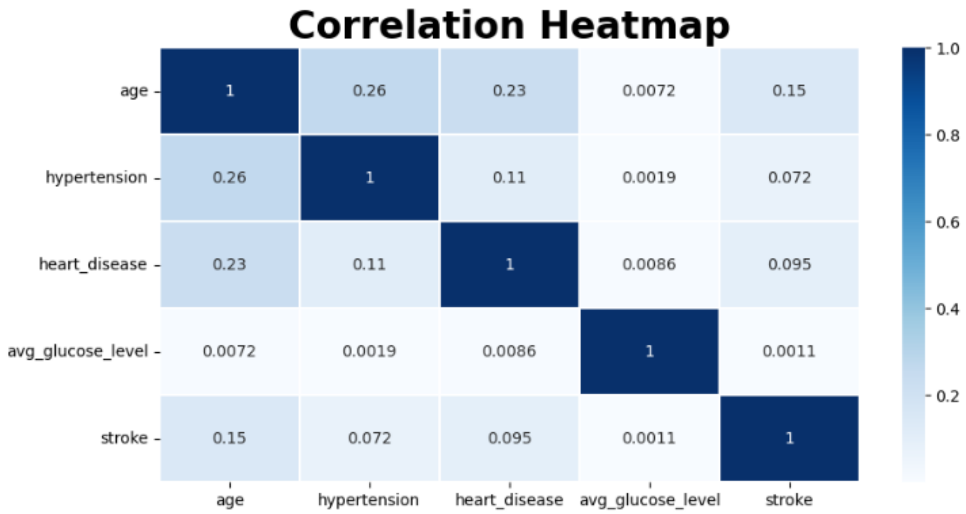


Figure 5: Preprocessing of the Dataset

The code in the image calculates the correlation matrix of a pandas DataFrame and plots a heatmap to visualize the correlations as shown in figure 6

Feature Engineering

```
In [30]:
corr = df.corr(method='pearson')
fig, ax = plt.subplots(figsize=(10, 5))

# Plotting correlation matrix
sns.heatmap(corr, annot=True, cmap="Blues", linewidth=0.2)
plt.title("Correlation Heatmap", fontsize=24, weight="bold")
plt.show()
```




```
In [31]:
from sklearn.preprocessing import LabelEncoder   # importing label encoder and min max scalar
label_enc = LabelEncoder()
# Label encode categorical columns
categorical_columns = ['gender','ever_married','work_type', 'Residence_type', 'smoking_status']  # Update with your categorical
label_encoder = LabelEncoder()
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])
```

Figure 6: Feature Engineering

Figure 7 shows the calculation of the correlation matrix between the target variable and all other variables.

Method 1: Correlation Analysis and Feature Selection using Correlation

```
In [32]: target_variable = 'stroke'

         # Set the correlation threshold
         correlation_threshold = 0.02  # adjust this threshold as needed

         # Calculate Pearson correlation coefficients
         correlation_matrix = df.corr(method='pearson')[[target_variable]]

         print(correlation_matrix)

         # Filter columns based on correlation threshold and exclude the target variable
         selected_columns = correlation_matrix[correlation_matrix.abs() > correlation_threshold].dropna().index.tolist()
         selected_columns = [col for col in selected_columns if col != target_variable]

         # Print selected columns
         print("Selected Columns for Prediction:")
         print(selected_columns)
```

```
                     stroke
gender             0.011312
age                0.148659
hypertension       0.072342
heart_disease      0.095164
ever_married       0.067436
work_type         -0.006624
Residence_type     0.007142
avg_glucose_level  0.001117
smoking_status    -0.025916
stroke             1.000000
Selected Columns for Prediction:
['age', 'hypertension', 'heart_disease', 'ever_married', 'smoking_status']
```

```
In [33]: #dropping failure column from data
         label = df['stroke']
         unique_classes= label.unique()
         df.drop(['stroke'],inplace=True,axis=1)
```

```
In [34]: sm=SMOTE(random_state=43)                          # balancing imbalanced labels using smote oversampling
         new_features, new_labels = sm.fit_resample(df, label)
         print("The number of classes before fit {}".format(Counter(label)))
         print("The number of classes after fit {}".format(Counter(new_labels)))
         new_features2 = new_features.copy()
         new_labels2 = new_labels.copy()
```

```
The number of classes before fit Counter({0: 37072, 1: 525})
The number of classes after fit Counter({0: 37072, 1: 37072})
```

Figure 7: Correlation Analysis and Feature Selection

4.1 Implementation of the KNN Model

An import statement that imports the KNeighborsClassifier class from the scikit-learn library.

1. in Figure 8. The creation of an instance of the KNeighborsClassifier class with setting the number of neighbors to 5, which means that the KNN algorithm will use the 5 most similar training examples to predict the label of a new data point.

```
In [39]: from sklearn.neighbors import KNeighborsClassifier       # importing K-Nearest Neighbour
         neigh= KNeighborsClassifier(n_neighbors=5)              # intialising algorithm
         neigh.fit(x_train, y_train)          # fitting training data into algorithm
         y_pred1 =  neigh.predict(x_test)
```

```
In [40]: print(confusion_matrix(y_test,y_pred1))          # evaluation on different metrics
         print(accuracy_score(y_test,y_pred1))
         print(classification_report(y_test,y_pred1))
         acc_neigh= accuracy_score(y_test,y_pred1)
         pre_neigh= precision_score(y_test,y_pred1,average='micro')
         re_neigh= recall_score(y_test,y_pred1,average='micro')
         f_neigh=f1_score(y_test,y_pred1,average='micro')
```

```
[[6964  487]
 [ 748 6630]]
0.9167172432395981
              precision    recall  f1-score   support

           0       0.90      0.93      0.92      7451
           1       0.93      0.90      0.91      7378

    accuracy                           0.92     14829
   macro avg       0.92      0.92      0.92     14829
weighted avg       0.92      0.92      0.92     14829
```

Figure 8: KNN Model

In Figure 9 shows a ROC curve for a BRF algorithm, with an AUC of 0.9840. This indicates that the algorithm is very good at distinguishing between positive and negative cases:

Plotting ROC_AUC Plot for BRF Algorithm

```
In [45]: y_probs2 = brf.predict_proba(x_test)[:, 1]

         # Calculate ROC curve
         fpr, tpr, thresholds = roc_curve(y_test, y_probs2)

         # Calculate AUC-ROC score
         roc_auc2 = roc_auc_score(y_test, y_probs2)
         fig = go.Figure()

         # Add ROC curve trace
         fig.add_trace(go.Scatter(x=fpr, y=tpr, mode='lines', name=f'AUC = {roc_auc2:.4f}', line=dict(color='orange', width=4)))


         fig.add_shape(type='line', line=dict(dash='dashdot'), x0=0, x1=1, y0=0, y1=1)


         fig.update_layout(
             title='Receiver Operating Characteristic (ROC) Curve',
             xaxis=dict(title='False Positive Rate (FPR)'),
             yaxis=dict(title='True Positive Rate (TPR)'),
             showlegend=True,
             #legend=dict(x=0.9, y=0.2),
             width=900,  # Customize width
             height=400,  # Customize height
         )

         fig.show()
```

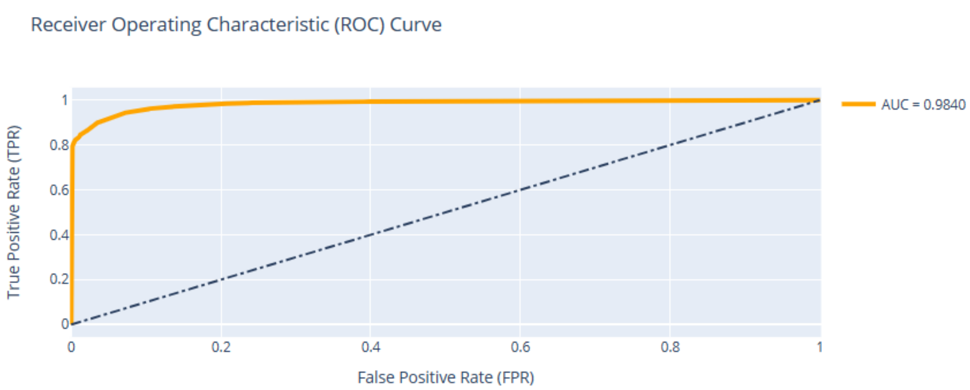


Figure 9: ROC_AUC Plot for BRF

Figure 10 shows the split of the dataset into train and test sets.

```
x_train, x_test, y_train, y_test = train_test_split(scaled_df1,new_labels, test_size =0.2,random_state = 43,shuffle=True)
```

Figure 10: Splitting Dataset

4.2 Implementation of the CatBoost Model

In Figure 11 shows CatBoost algorithm being used to train a machine learning model. The model has an accuracy of 93%.

CatBoost Algorithm

```
In [48]: from catboost import CatBoostClassifier       # importing CatBoostClassifier
         cb = CatBoostClassifier()         # intialising algorithm
         cb.fit(x_train, y_train)                    # fitting training data into algorithm
         y_pred3 = cb.predict(x_test)                          # prediction on test data
```

```
Learning rate set to 0.058895
0:      learn: 0.6577420        total: 185ms    remaining: 3m 4s
1:      learn: 0.6264938        total: 214ms    remaining: 1m 46s
2:      learn: 0.5951542        total: 246ms    remaining: 1m 21s
3:      learn: 0.5710607        total: 272ms    remaining: 1m 7s
4:      learn: 0.5517598        total: 301ms    remaining: 59.9s
5:      learn: 0.5318166        total: 329ms    remaining: 54.4s
6:      learn: 0.5156861        total: 355ms    remaining: 50.4s
7:      learn: 0.5036045        total: 383ms    remaining: 47.5s
8:      learn: 0.4939120        total: 412ms    remaining: 45.4s
9:      learn: 0.4855053        total: 450ms    remaining: 44.5s
10:     learn: 0.4767839        total: 487ms    remaining: 43.8s
11:     learn: 0.4686432        total: 525ms    remaining: 43.2s
12:     learn: 0.4625136        total: 559ms    remaining: 42.4s
13:     learn: 0.4552737        total: 589ms    remaining: 41.4s
14:     learn: 0.4504611        total: 618ms    remaining: 40.5s
15:     learn: 0.4454825        total: 646ms    remaining: 39.7s
16:     learn: 0.4400219        total: 682ms    remaining: 39.4s
17:     learn: 0.4371675        total: 712ms    remaining: 38.8s
```

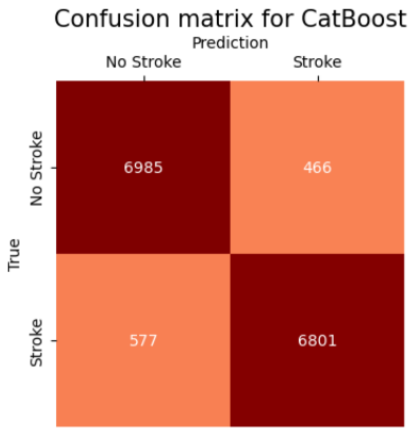
```
In [49]: print(confusion_matrix(y_test,y_pred3))          # evaluation on different metrics
         print(accuracy_score(y_test,y_pred3))
         print(classification_report(y_test,y_pred3))
         acc_cat= accuracy_score(y_test,y_pred3)
         pre_cat= precision_score(y_test,y_pred3,average='micro')
         re_cat= recall_score(y_test,y_pred3,average='micro')
         f_cat=f1_score(y_test,y_pred3,average='micro')
```

```
[[6985  466]
 [ 577 6801]]
0.9296648459100412
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      7451
           1       0.94      0.92      0.93      7378

    accuracy                           0.93     14829
   macro avg       0.93      0.93      0.93     14829
weighted avg       0.93      0.93      0.93     14829
```

Figure 11: CatBoost Model

The confusion matrix shows the performance of a CatBoost model on a stroke prediction task. The model has an accuracy of 93% which is shown in figure 12.

In [51]:
```
plt.figure(figsize = (4,4))  # plottting confusion matrix
true= ['No Stroke','Stroke']
class_label = ['No Stroke','Stroke']
fig = sns.heatmap(confusion_matrix(y_test, y_pred3),cmap= "OrRd", fmt='d',annot=True,cbar = False,
            center = True,xticklabels= class_label,yticklabels= class_label,annot_kws = {'size':10})
fig.set_xlabel("Prediction",fontsize=10)
fig.xaxis.set_label_position('top')
fig.set_ylabel("True",fontsize=10)
fig.xaxis.tick_top()
plt.title('Confusion matrix for CatBoost', fontsize = 15); # title with fontsize 20
```



In [52]:
```
accuracy3=accuracy_score(y_test,y_pred3)
precision3 = precision_score(y_test, y_pred3, average = 'weighted')
recall3 = recall_score(y_test, y_pred3, average = 'weighted')
f1_score3 = f1_score(y_test, y_pred3, average = 'weighted')
auc_score3 = roc_auc3
print("Precision: %s, Recall: %s, f1_score: %s, accuracy: %s" %(precision3, recall3, f1_score3, accuracy3))
```

```
Precision: 0.9297559953062087, Recall: 0.9296648459100412, f1_score: 0.9296583122522437, accuracy: 0.9296648459100412
```

Figure 12: Confusion Matrix for CatBoost

Figure 13, shows a table comparing the performance of algorithms based on Method 1 and Method 2. Method 2 outperforms Method 1 on all metrics.

Evaluation and Comparison of Algorithms Based on Method 1 and Method2

```
In [86]: data_C = {'Precision':[precision1,precision2,precision3,precision4,precision5,precision6,precision7,precision8], # data for metri
                 'Recall':[recall1,recall2,recall3,recall4,recall5,recall6,recall7,recall8],
                 'Accuracy':[accuracy1,accuracy2,accuracy3,accuracy4,accuracy5,accuracy6,accuracy7,accuracy8],
                 'F1_score':[f1_score1,f1_score2,f1_score3,f1_score4,f1_score5,f1_score6,f1_score7,f1_score8],
                   'Auc_Score':[auc_score1,auc_score2,auc_score3,auc_score4,auc_score5,auc_score6,auc_score7,auc_score8],
                 'Algo':['KNN1','BRF1','Catboost1','Xgboost1','KNN2','BRF2','Catboost3','Xgboost4'] }
         data_C = pd.DataFrame(data_C)         # making dataframe of data
```

```
In [87]: data_C
```

Out[87]:

	Precision	Recall	Accuracy	F1_score	Auc_Score	Algo
0	0.917220	0.916717	0.916717	0.916684	0.963315	KNN1
1	0.934966	0.934925	0.934925	0.934922	0.983982	BRF1
2	0.929756	0.929665	0.929665	0.929658	0.982777	Catboost1
3	0.875145	0.874503	0.874503	0.874434	0.952615	Xgboost1
4	0.933959	0.927372	0.927372	0.927119	0.968278	KNN2
5	0.975808	0.975723	0.975723	0.975723	0.997350	BRF2
6	0.976401	0.976398	0.976398	0.976397	0.982777	Catboost3
7	0.945998	0.945917	0.945917	0.945916	0.990083	Xgboost4

Figure 13: Evaluation and Comparison

In Figure 14 shows a bar plot comparing the accuracy of four machine learning algorithms on two datasets. On both datasets, XgBoost outperforms the other algorithms.

Comparison of Algorithms and Methods Based on Accuracy

```
In [88]: #Accuracy Plot
         np.random.seed(42)
         # Customizable colors for bars
         custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']
         random_x= np.random.randint(1,101,100)
         random_y= np.random.randint(1,101,100)

         x = ['Method1', 'Method2']

         plot = go.Figure(data=[go.Bar(
             name = 'K-Nearest Neighbour',
             x = x,
             y = [accuracy1,accuracy5],width=[0.1,0.1],text=[data_C['Accuracy'].loc[0],data_C['Accuracy'].loc[4]], textposition='auto',tex
             marker_color=custom_colors[0]
            ),
                                 go.Bar(
             name = 'Balanced Random Forest',
             x = x,
             y = [accuracy2,accuracy6],width=[0.1,0.1],text=[data_C['Accuracy'].loc[1],data_C['Accuracy'].loc[5]], textposition='auto',tex
                                 marker_color=custom_colors[1]
            ),
                                 go.Bar(
             name = 'CatBoost',
             x = x,
             y = [accuracy3,accuracy7],width=[0.1,0.1],text=[data_C['Accuracy'].loc[2],data_C['Accuracy'].loc[6]], textposition='auto',tex
                                 marker_color=custom_colors[2]
            ),
                                 go.Bar(
             name = 'XgBoost',
             x = x,
             y = [accuracy4,accuracy8],width=[0.1,0.1],text=[data_C['Accuracy'].loc[3],data_C['Accuracy'].loc[7]], textposition='auto',tex
                                 marker_color=custom_colors[3]
            ),
         ])

         plot.update_traces(textposition='inside')
         tick_vals = np.arange(0, 1.1, 0.1)
         plot.update_yaxes(tickvals=tick_vals, ticktext=[f'{val:0.1%}' for val in tick_vals])
         plot.update_layout(uniformtext_minsize=8, uniformtext_mode='hide', title = 'Accuracy Comparison')
         plot.show()
```

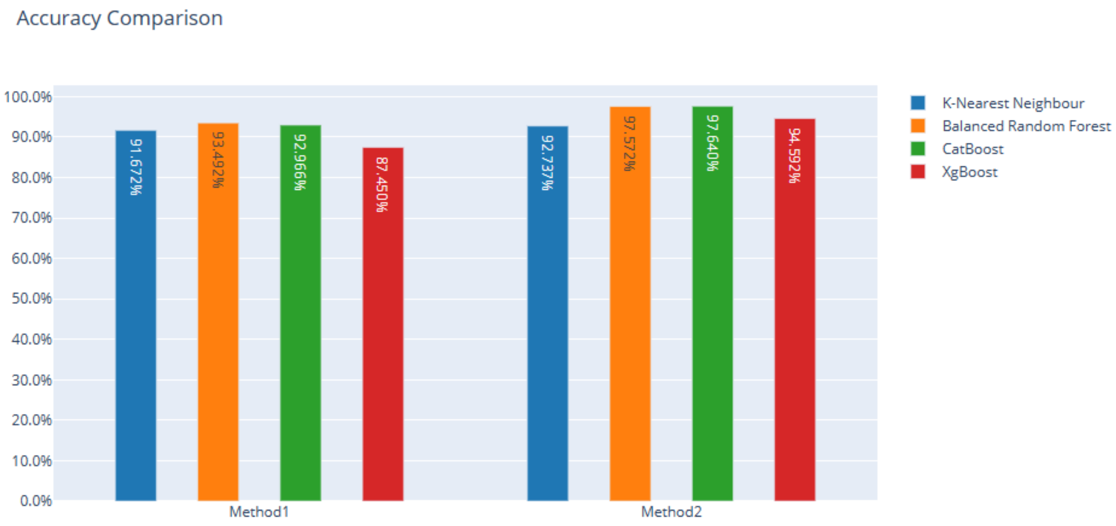


Figure 14: Comparison based on Accuracy

In Figure 15, The image shows a comparison of algorithms and methods based on AUC score. Method 1 and Method 2 have the highest AUC scores, while Method 3 and Method 4 have the lowest.

Comparison of Algorithms and Methods Based on Auc_Score

In [91]:
```python
# Auc_Score Comparison
np.random.seed(42)
# Customizable colors for bars
custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']
random_x= np.random.randint(1,101,100)
random_y= np.random.randint(1,101,100)

x = ['Method1', 'Method2']

plot = go.Figure(data=[go.Bar(
    name = 'K-Nearest Neighbour',
    x = x,
    y = [auc_score1,auc_score5],width=[0.1,0.1],text=[data_C['Auc_Score'].loc[0],data_C['Auc_Score'].loc[4]], textposition='auto'
    marker_color=custom_colors[0]
   ),
                       go.Bar(
    name = 'Balanced Random Forest',
    x = x,
    y = [auc_score2,auc_score6],width=[0.1,0.1],text=[data_C['Auc_Score'].loc[1],data_C['Auc_Score'].loc[5]], textposition='auto'
                           marker_color=custom_colors[1]
   ),
                       go.Bar(
    name = 'CatBoost',
    x = x,
    y = [auc_score3,auc_score7],width=[0.1,0.1],text=[data_C['Auc_Score'].loc[2],data_C['Auc_Score'].loc[6]], textposition='auto'
                           marker_color=custom_colors[2]
   ),
                       go.Bar(
    name = 'XgBoost',
    x = x,
    y = [auc_score4,auc_score8],width=[0.1,0.1],text=[data_C['Auc_Score'].loc[3],data_C['Auc_Score'].loc[7]], textposition='auto'
                           marker_color=custom_colors[3]
   ),
])

plot.update_traces(textposition='inside')
tick_vals = np.arange(0, 1.1, 0.1)
plot.update_yaxes(tickvals=tick_vals, ticktext=[f'{val:.1f}' for val in tick_vals])
plot.update_layout(uniformtext_minsize=8, uniformtext_mode='hide', title = 'Auc_Score Comparison')
plot.show()
```

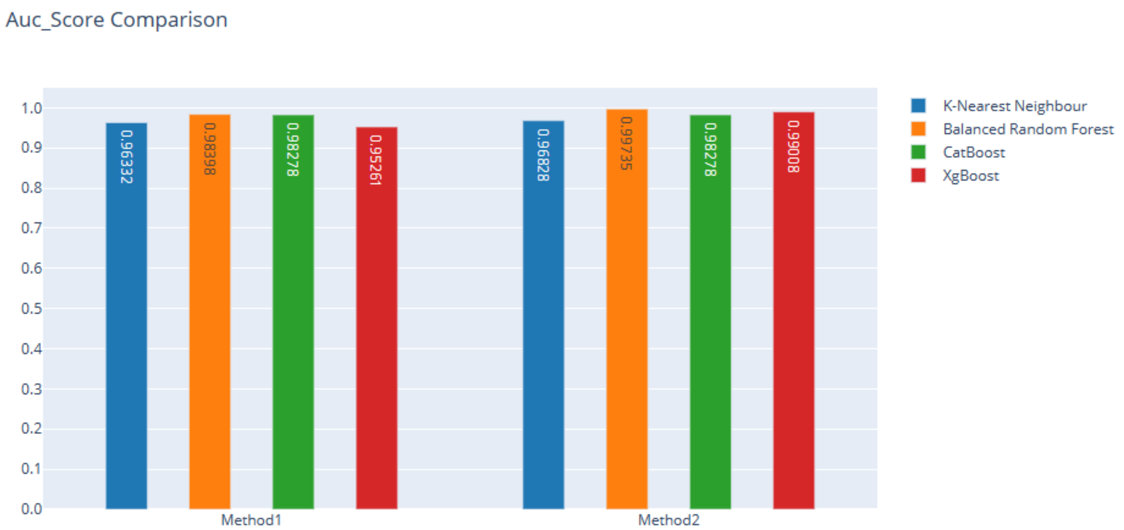


Figure 15: Comparison based on Auc_Score