Importing the libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
```

Importing the dataset

```
df = pd.read_csv("/content/drive/MyDrive/MACHINE LEARNING PROJECTS/stroke-data.csv")
```

```
df.head()
```

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

Next steps: Generate code with df | View recommended plots | New interactive sheet

Checking the structure of the data and presence of any null values

```
df.shape
```

```
(5110, 12)
```

```
df.size
```

```
61320
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

```
df.isnull().sum()
```

	0
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

Treating the null values

```
df['bmi'].unique()
```

```
array([36.6,  nan, 32.5, 34.4, 24. , 29. , 27.4, 22.8, 24.2, 29.7, 36.8,
       27.3, 28.2, 30.9, 37.5, 25.8, 37.8, 22.4, 48.9, 26.6, 27.2, 23.5,
       28.3, 44.2, 25.4, 22.2, 30.5, 26.5, 33.7, 23.1, 32. , 29.9, 23.9,
       28.5, 26.4, 20.2, 33.6, 38.6, 39.2, 27.7, 31.4, 36.5, 33.2, 32.8,
       40.4, 25.3, 30.2, 47.5, 20.3, 30. , 28.9, 28.1, 31.1, 21.7, 27. ,
       24.1, 45.9, 44.1, 22.9, 29.1, 32.3, 41.1, 25.6, 29.8, 26.3, 26.2,
       29.4, 24.4, 28. , 28.8, 34.6, 19.4, 30.3, 41.5, 22.6, 56.6, 27.1,
       31.3, 31. , 31.7, 35.8, 28.4, 20.1, 26.7, 38.7, 34.9, 25. , 23.8,
       21.8, 27.5, 24.6, 32.9, 26.1, 31.9, 34.1, 36.9, 37.3, 45.7, 34.2,
       23.6, 22.3, 37.1, 45. , 25.5, 30.8, 37.4, 34.5, 27.9, 29.5, 46. ,
       42.5, 35.5, 26.9, 45.5, 31.5, 33. , 23.4, 30.7, 20.5, 21.5, 40. ,
       28.6, 42.2, 29.6, 35.4, 16.9, 26.8, 39.3, 32.6, 35.9, 21.2, 42.4,
       40.5, 36.7, 29.3, 19.6, 18. , 17.6, 19.1, 50.1, 17.7, 54.6, 35. ,
       22. , 39.4, 19.7, 22.5, 25.2, 41.8, 60.9, 23.7, 24.5, 31.2, 16. ,
       31.6, 25.1, 24.8, 18.3, 20. , 19.5, 36. , 35.3, 40.1, 43.1, 21.4,
       34.3, 27.6, 16.5, 24.3, 25.7, 21.9, 38.4, 25.9, 54.7, 18.6, 24.9,
       48.2, 20.7, 39.5, 23.3, 64.8, 35.1, 43.6, 21. , 47.3, 16.6, 21.6,
       15.5, 35.6, 16.7, 41.9, 16.4, 17.1, 29.2, 37.9, 44.6, 39.6, 40.3,
       41.6, 39. , 23.2, 18.9, 36.1, 36.3, 46.5, 16.8, 46.6, 35.2, 20.9,
       13.8, 31.8, 15.3, 38.2, 45.2, 17. , 49.8, 27.8, 60.2, 23. , 22.1,
       26. , 44.3, 51. , 39.7, 34.7, 21.3, 41.2, 34.8, 19.2, 35.7, 40.8,
       24.7, 19. , 32.4, 34. , 28.7, 32.1, 51.5, 20.4, 30.6, 71.9, 19.3,
       40.9, 17.2, 16.1, 16.2, 40.6, 18.4, 21.1, 42.3, 32.2, 50.2, 17.5,
       18.7, 42.1, 47.8, 20.8, 30.1, 17.3, 36.4, 12. , 36.2, 55.7, 14.4,
       43. , 41.7, 33.8, 43.9, 22.7, 57.5, 37. , 38.5, 16.3, 44. , 32.7,
       54.2, 40.2, 33.3, 17.4, 41.3, 52.3, 14.6, 17.8, 46.1, 33.1, 18.1,
       43.8, 50.3, 38.9, 43.7, 39.9, 15.9, 19.8, 12.3, 78. , 38.3, 41. ,
       42.6, 43.4, 15.1, 20.6, 33.5, 43.2, 30.4, 38. , 33.4, 44.9, 44.7,
       37.6, 39.8, 53.4, 55.2, 42. , 37.2, 42.8, 18.8, 42.9, 14.3, 37.7,
       48.4, 50.6, 46.2, 49.5, 43.3, 33.9, 18.5, 44.5, 45.4, 55. , 54.8,
       19.9, 17.9, 15.6, 52.8, 15.2, 66.8, 55.1, 18.2, 48.5, 55.9, 57.3,
       10.3, 14.1, 15.7, 56. , 44.8, 13.4, 51.8, 38.1, 57.7, 44.4, 38.8,
       49.3, 39.1, 54. , 56.1, 97.6, 53.9, 13.7, 11.5, 41.4, 14.2, 49.4,
       15.4, 45.1, 49.2, 48.7, 53.8, 42.7, 48.8, 52.7, 53.5, 50.5, 15.8,
       45.3, 14.8, 51.9, 63.3, 40.7, 61.2, 48. , 46.8, 48.3, 58.1, 50.4,
       11.3, 12.8, 13.5, 14.5, 15. , 59.7, 47.4, 52.5, 13.2, 52.9, 61.6,
       49.9, 54.3, 47.9, 13. , 13.9, 50.9, 57.2, 64.4, 92. , 50.8, 57.9,
       45.8, 47.6, 14. , 46.4, 46.9, 47.1, 13.3, 48.1, 51.7, 46.3, 54.1,
       14.9])
```

```
df['bmi'].value_counts()
```

bmi	count
28.7	41
28.4	38
26.7	37
27.6	37
26.1	37
...	...
48.7	1
49.2	1
51.0	1
49.4	1
14.9	1

418 rows × 1 columns

```
df['bmi'].mean()
```

```
28.893236911794666
```

```
df['bmi'].fillna(df['bmi'].mean(), inplace=True)
```

```
<ipython-input-11-2a6d4795ba2f>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['bmi'].fillna(df['bmi'].mean(), inplace=True)
```

```
df['bmi'].isnull().sum()
```

```
0
```

```
type('bmi')
```

```
str
```

```
df['bmi'] = pd.to_numeric(df['bmi'], errors='coerce')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                5110 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

1. Gender

```
df['gender'].unique()
```

```
array(['Male', 'Female', 'Other'], dtype=object)
```

```
df['gender'].value_counts()
```

gender	count
Female	2994
Male	2115
Other	1

As there is only 1 value present for the OTHER, we will drop that

```
df = df.drop(df[df['gender'] == 'Other'].index)
```

```
df.head(5)
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_s
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.600000	formerly sr
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	28.893237	never sr
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.500000	never sr
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.400000	sr
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.000000	never sr

Next steps: Generate code with df   View recommended plots   New interactive sheet

```
df['gender'].value_counts()
```

gender	count
Female	2994
Male	2115

Now replacing the Male as 1 and Female as 0

```
df['gender'].replace({'Male': 1, 'Female': 0}, inplace = True)
```

```
<ipython-input-21-a05e76498d29>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['gender'].replace({'Male': 1, 'Female': 0}, inplace = True)
<ipython-input-21-a05e76498d29>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future ve
  df['gender'].replace({'Male': 1, 'Female': 0}, inplace = True)
```

```
df.head(3)
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_s
0	9046	1	67.0	0	1	Yes	Private	Urban	228.69	36.600000	formerly sr
1	51676	0	61.0	0	0	Yes	Self-employed	Rural	202.21	28.893237	never sr
2	31112	1	80.0	0	1	Yes	Private	Rural	105.92	32.500000	never sr

Next steps: Generate code with df   View recommended plots   New interactive sheet

2. Ever_married

```
df['ever_married'].unique()
```

```
array(['Yes', 'No'], dtype=object)
```

```
df['ever_married'].value_counts()
```

	count
ever_married	
Yes	3353
No	1756

Replacing Yes as 1 and No and 0

```
df['ever_married'].replace({'Yes' : 1, 'No' : 0}, inplace = True)
```

```
<ipython-input-25-8391b8ec88f2>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['ever_married'].replace({'Yes' : 1, 'No' : 0}, inplace = True)
<ipython-input-25-8391b8ec88f2>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future ve
  df['ever_married'].replace({'Yes' : 1, 'No' : 0}, inplace = True)
```

```
df.head(3)
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_s
0	9046	1	67.0	0	1	1	Private	Urban	228.69	36.600000	formerly sr
1	51676	0	61.0	0	0	1	Self-employed	Rural	202.21	28.893237	never sr
2	31112	1	80.0	0	1	1	Private	Rural	105.92	32.500000	never sr

Next steps: Generate code with df View recommended plots New interactive sheet

3. Work Type

```
df['work_type'].unique()
```

```
array(['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked'],
      dtype=object)
```

```
df['work_type'].value_counts()
```

	count
work_type	
Private	2924
Self-employed	819
children	687
Govt_job	657
Never_worked	22

4. Residence type

```
df['Residence_type'].unique()
```

```
array(['Urban', 'Rural'], dtype=object)
```

```
df['Residence_type'].value_counts()
```

	count
Residence_type	
Urban	2596
Rural	2513

```
df['Residence_type'].replace({'Urban' : 1, 'Rural' : 0}, inplace = True)
```

```
<ipython-input-31-fadd75f3552d>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Residence_type'].replace({'Urban' : 1, 'Rural' : 0}, inplace = True)
<ipython-input-31-fadd75f3552d>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future ve
  df['Residence_type'].replace({'Urban' : 1, 'Rural' : 0}, inplace = True)
```

```
df.head(3)
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_s
0	9046	1	67.0	0	1	1	Private	1	228.69	36.600000	formerly sr
1	51676	0	61.0	0	0	1	Self-employed	0	202.21	28.893237	never sr
2	31112	1	80.0	0	1	1	Private	0	105.92	32.500000	never sr

Next steps: Generate code with df View recommended plots New interactive sheet

5. Smoking status

```
df['smoking_status'].unique()
```

```
array(['formerly smoked', 'never smoked', 'smokes', 'Unknown'],
      dtype=object)
```

```
df['smoking_status'].value_counts()
```

	count
smoking_status	
never smoked	1892
Unknown	1544
formerly smoked	884
smokes	789

6. ID

I will drop this feature as it has no impact on my target feature.

```
df.drop('id', axis = 1, inplace = True)
```

```
df.head(3)
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	1	67.0	0	1	1	Private	1	228.69	36.600000	formerly smoked
1	0	61.0	0	0	1	Self-employed	0	202.21	28.893237	never smoked

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.dtypes
```

	0
gender	int64
age	float64
hypertension	int64
heart_disease	int64
ever_married	int64
work_type	object
Residence_type	int64
avg_glucose_level	float64
bmi	float64
smoking_status	object
stroke	int64

Target Column (Stroke) Our target column consist of 0 and 1 which result to Binary Classification Problem

```
df['stroke'].value_counts()
```

	count
stroke	
0	4860
1	249

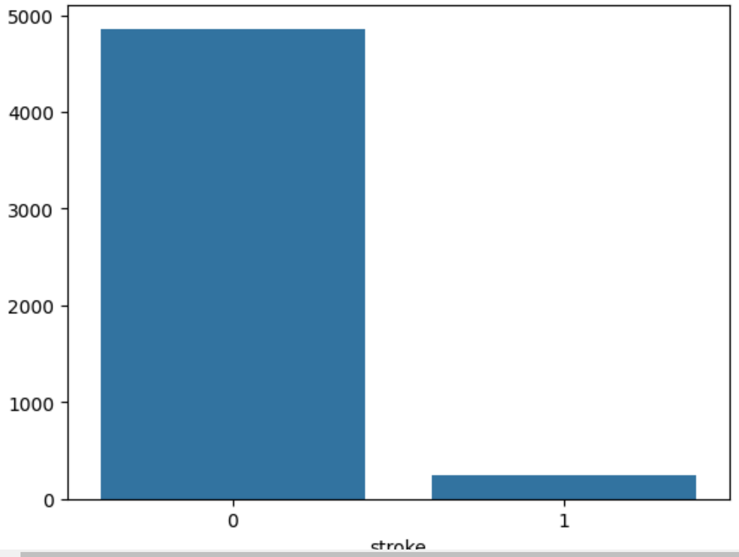
```
df['stroke'].value_counts(normalize = True) * 100
```

	proportion
stroke	
0	95.126248
1	4.873752

The dataset is in imbalance form as the proportion of not having stroke is 95% whreas the proportion of having stroke is only 5%. Hence we need to fix this.

```
stroke_counts = df['stroke'].value_counts()
sns.barplot(x=stroke_counts.index, y=stroke_counts.values)
```

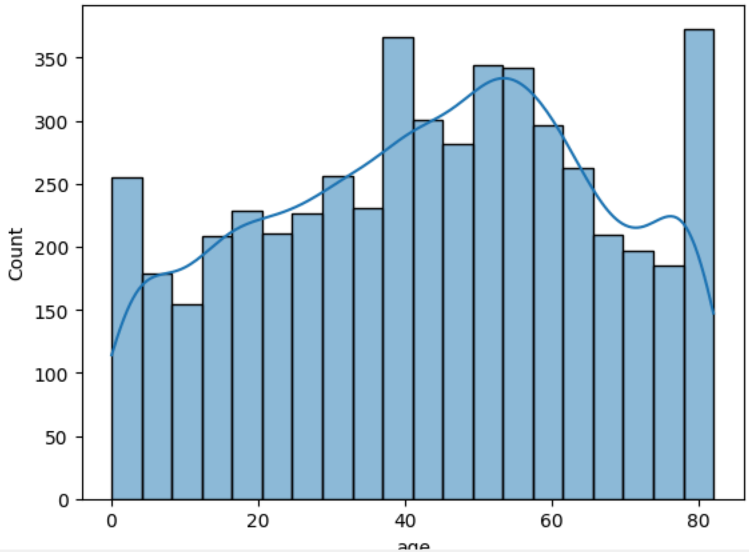
```
<Axes: xlabel='stroke'>
```



We will use ROC metric to fix the imbalance traget column dataset.

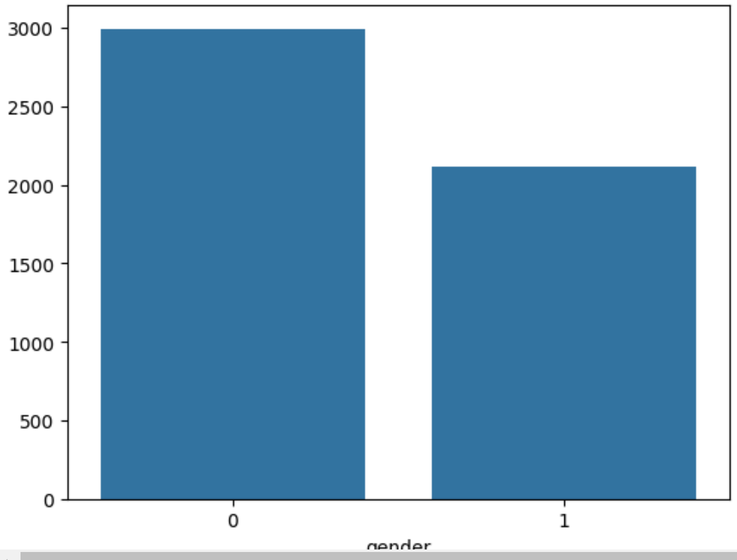
```
sns.histplot(x = df['age'], kde = True)
```

```
<Axes: xlabel='age', ylabel='Count'>
```



```
gender_counts = df['gender'].value_counts()
sns.barplot(x=gender_counts.index, y=gender_counts.values)
```

```
<Axes: xlabel='gender'>
```



Male are having higher chances of getting stroke as compared to female

Dividing the data as per categorical and numerical features

```
categorical_features = [x for x in df.columns if df[x].dtypes == 'O']
```

```
categorical_features
```

```
['work_type', 'smoking_status']
```

```
numerical_features = [x for x in df.columns if df[x].dtypes != 'O']
```

```
numerical_features
```

```
['gender',
 'age',
 'hypertension',
 'heart_disease',
 'ever_married',
 'Residence_type',
 'avg_glucose_level',
 'bmi',
 'stroke']
```

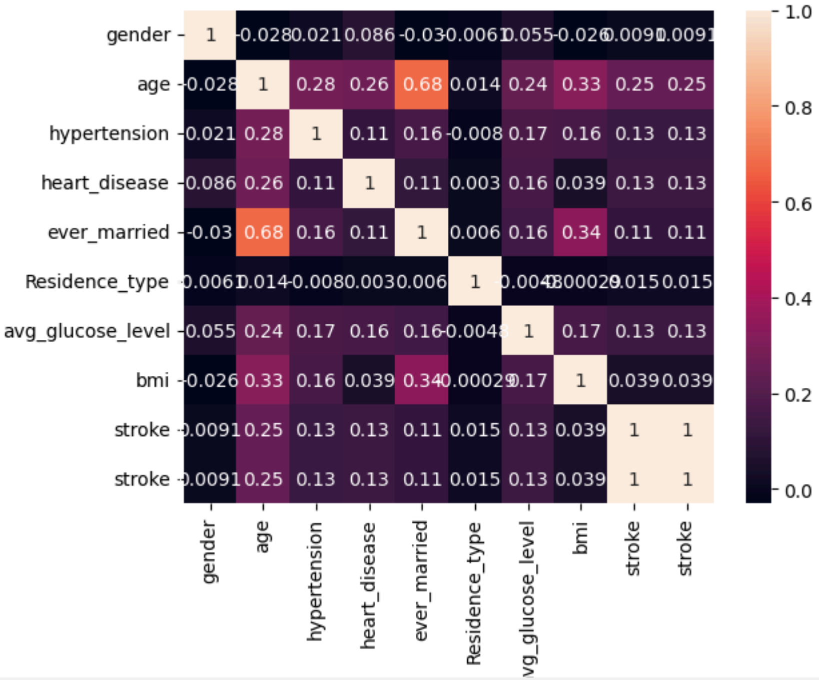
Checking the correlation between the dependent and independent variables

```
df[numerical_features + ['stroke']].corr()
```

	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stro
gender	1.000000	-0.027752	0.021223	0.085685	-0.030171	-0.006105	0.054722	-0.025606	0.0090
age	-0.027752	1.000000	0.276367	0.263777	0.679084	0.014031	0.238323	0.325858	0.2452
hypertension	0.021223	0.276367	1.000000	0.108292	0.164187	-0.007980	0.174540	0.160147	0.1278
heart_disease	0.085685	0.263777	0.108292	1.000000	0.114601	0.003045	0.161907	0.038862	0.1349
ever_married	-0.030171	0.679084	0.164187	0.114601	1.000000	0.005988	0.155329	0.335563	0.1082
Residence_type	-0.006105	0.014031	-0.007980	0.003045	0.005988	1.000000	-0.004783	-0.000288	0.0154
avg_glucose_level	0.054722	0.238323	0.174540	0.161907	0.155329	-0.004783	1.000000	0.168910	0.1319
bmi	-0.025606	0.325858	0.160147	0.038862	0.335563	-0.000288	0.168910	1.000000	0.0389
stroke	0.009081	0.245239	0.127891	0.134905	0.108299	0.015415	0.131991	0.038912	1.0000
stroke	0.009081	0.245239	0.127891	0.134905	0.108299	0.015415	0.131991	0.038912	1.0000

```
sns.heatmap(df[numerical_features + ['stroke']].corr(), annot = True)
```

```
<Axes: >
```



There is no features which are hiving higher correlation with our target column.

```
numerical_features
```

```
['gender',
 'age',
 'hypertension',
 'heart_disease',
 'ever_married',
 'Residence_type',
 'avg_glucose_level',
 'bmi',
 'stroke']
```

```
categorical_features
```

```
['work_type', 'smoking_status']
```

```
df.head(3)
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	1	67.0	0	1	1	Private	1	228.69	36.600000	formerly smoked
1	0	61.0	0	0	1	Self-employed	0	202.21	28.893237	never smoked

Next steps: Generate code with df View recommended plots New interactive sheet

```
df['bmi'].shape
```

```
(5109,)
```

Converting my categorical columns to numerical columns

```
categorical_features
```

```
['work_type', 'smoking_status']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5109 entries, 0 to 5109
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             5109 non-null   int64
 1   age                5109 non-null   float64
 2   hypertension       5109 non-null   int64
 3   heart_disease      5109 non-null   int64
 4   ever_married       5109 non-null   int64
 5   work_type          5109 non-null   object
 6   Residence_type     5109 non-null   int64
 7   avg_glucose_level  5109 non-null   float64
 8   bmi                5109 non-null   float64
 9   smoking_status     5109 non-null   object
 10  stroke             5109 non-null   int64
dtypes: float64(3), int64(6), object(2)
memory usage: 479.0+ KB
```

```
df.isnull().sum()
```

	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

```
# Drop rows with any null values in the relevant columns
df = df.dropna(subset=['work_type', 'smoking_status'])

# Ensure proper data types
df['work_type'] = df['work_type'].astype(str)
df['smoking_status'] = df['smoking_status'].astype(str)

# Create a OneHotEncoder instance
ohe = OneHotEncoder()

# Fit and transform the categorical columns
encoded_features = ohe.fit_transform(df[['work_type', 'smoking_status']])

# Create a DataFrame with the new encoded features
encoded_df = pd.DataFrame(encoded_features.toarray(), columns=ohe.get_feature_names_out(['work_type', 'smoking_status']))

# Concatenate the original DataFrame with the new one, dropping the original columns
df = pd.concat([df.drop(['work_type', 'smoking_status'], axis=1), encoded_df], axis=1)

print(df)
```

```
      gender   age  hypertension  heart_disease  ever_married  Residence_type  \
0        1.0  67.0           0.0            1.0           1.0             1.0
1        0.0  61.0           0.0            0.0           1.0             0.0
2        1.0  80.0           0.0            1.0           1.0             0.0
3        0.0  49.0           0.0            0.0           1.0             1.0
4        0.0  79.0           1.0            0.0           1.0             0.0
...      ...   ...           ...            ...           ...             ...
5106     0.0  81.0           0.0            0.0           1.0             1.0
5107     0.0  35.0           0.0            0.0           1.0             0.0
5108     1.0  51.0           0.0            0.0           1.0             0.0
5109     0.0  44.0           0.0            0.0           1.0             1.0
3116     NaN   NaN           NaN            NaN           NaN             NaN

      avg_glucose_level        bmi  stroke  work_type_Govt_job  \
0                228.69  36.600000     1.0                 0.0
1                202.21  28.893237     1.0                 0.0
2                105.92  32.500000     1.0                 0.0
3                171.23  34.400000     1.0                 0.0
4                174.12  24.000000     1.0                 0.0
...                 ...        ...     ...                 ...
5106             125.20  40.000000     0.0                 0.0
5107              82.99  30.600000     0.0                 0.0
5108             166.29  25.600000     0.0                 1.0
5109              85.28  26.200000     0.0                 NaN
3116                NaN        NaN     NaN                 0.0
```

```
      work_type_Never_worked  work_type_Private  work_type_Self-employed  \
0                        0.0                1.0                      0.0   
1                        0.0                0.0                      1.0   
2                        0.0                1.0                      0.0   
3                        0.0                1.0                      0.0   
4                        0.0                0.0                      1.0   
...                      ...                ...                      ...   
5106                     0.0                0.0                      1.0   
5107                     0.0                1.0                      0.0   
5108                     0.0                0.0                      0.0   
5109                     NaN                NaN                      NaN   
3116                     0.0                0.0                      0.0   

      work_type_children  smoking_status_Unknown  \
0                    0.0                     0.0   
1                    0.0                     0.0   
2                    0.0                     0.0   
3                    0.0                     0.0   
4                    0.0                     0.0   
...                  ...                     ...   
5106                 0.0                     0.0   
5107                 0.0                     0.0   
5108                 0.0                     1.0   
5109                 NaN                     NaN   
3116                 1.0                     1.0   

      smoking_status_formerly smoked  smoking_status_never smoked  \
0                                1.0                          0.0   
1                                0.0                          1.0   
2                                0.0                          1.0   
3                                0.0                          0.0   
4                                0.0                          1.0   
```

```
df = df.drop(df.index[-2:])
```

```
df.isnull().sum()
```

	0
gender	0
age	0
hypertension	0
heart_disease	0
ever_married	0
Residence_type	0
avg_glucose_level	0
bmi	0
stroke	0
work_type_Govt_job	0
work_type_Never_worked	0
work_type_Private	0
work_type_Self-employed	0
work_type_children	0
smoking_status_Unknown	0
smoking_status_formerly smoked	0
smoking_status_never smoked	0
smoking_status_smokes	0

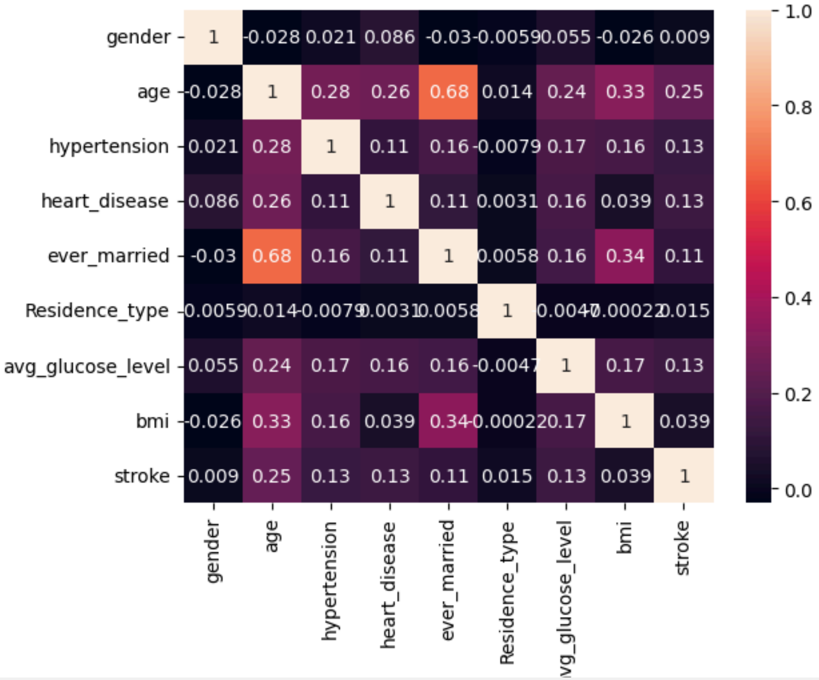
```
df.head(3)
```

	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_job
0	1.0	67.0	0.0	1.0	1.0	1.0	228.69	36.600000	1.0	0.0
1	0.0	61.0	0.0	0.0	1.0	0.0	202.21	28.893237	1.0	0.0
2	1.0	80.0	0.0	1.0	1.0	0.0	105.92	32.500000	1.0	0.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
sns.heatmap(df[numerical_features].corr(), annot = True)
```

```
<Axes: >
```



	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stroke
gender	1	-0.028	0.021	0.086	-0.03	-0.0059	0.055	-0.026	0.009
age	-0.028	1	0.28	0.26	0.68	0.014	0.24	0.33	0.25
hypertension	0.021	0.28	1	0.11	0.16	-0.0079	0.17	0.16	0.13
heart_disease	0.086	0.26	0.11	1	0.11	0.0031	0.16	0.039	0.13
ever_married	-0.03	0.68	0.16	0.11	1	0.0058	0.16	0.34	0.11
Residence_type	-0.0059	0.014	-0.0079	0.0031	0.0058	1	-0.0047	-0.00022	0.015
avg_glucose_level	0.055	0.24	0.17	0.16	0.16	-0.0047	1	0.17	0.13
bmi	-0.026	0.33	0.16	0.039	0.34	-0.00022	0.17	1	0.039
stroke	0.009	0.25	0.13	0.13	0.11	0.015	0.13	0.039	1

There is no correlation hence we can go for model prediction

```
# Define features and target
X = df.drop('stroke', axis=1)
y = df['stroke']
```

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

As our target column is imabalnce we are fixing this using SMOTE

```
# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Check the class distribution after SMOTE
print("Original training set shape:", X_train.shape, "Class distribution:", y_train.value_counts())
print("Resampled training set shape:", X_train_resampled.shape, "Class distribution:", pd.Series(y_train_resampled).value_counts())
```

```
Original training set shape: (4086, 17) Class distribution: stroke
0.0    3899
1.0     187
Name: count, dtype: int64
Resampled training set shape: (7798, 17) Class distribution: stroke
0.0    3899
1.0    3899
Name: count, dtype: int64
```

```
df.head(3)
```

	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_job
0	1.0	67.0	0.0	1.0	1.0	1.0	228.69	36.600000	1.0	0.0
1	0.0	61.0	0.0	0.0	1.0	0.0	202.21	28.893237	1.0	0.0
2	1.0	80.0	0.0	1.0	1.0	0.0	105.92	32.500000	1.0	0.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

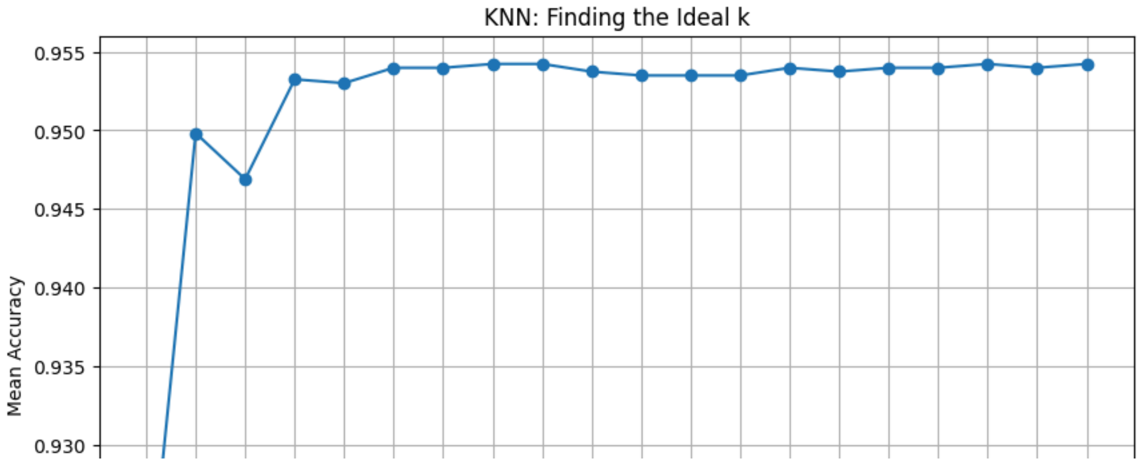
```
▾    RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Make predictions
y_pred = model.predict(X_test)


# Evaluate model
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[958   2]
 [ 62   0]]
              precision    recall  f1-score   support

         0.0       0.94      1.00      0.97       960
         1.0       0.00      0.00      0.00        62

    accuracy                           0.94      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.88      0.94      0.91      1022
```

Implementing KNN

```
# List to store the accuracy for different values of k
k_values = range(1, 21)  # Test k from 1 to 20
accuracy = []


# Loop to find the ideal k
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    # Perform 5-fold cross-validation and get the mean accuracy
    cv_scores = cross_val_score(knn, X_train, y_train, cv=5)
    accuracy.append(cv_scores.mean())


# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy, marker='o')
plt.title('KNN: Finding the Ideal k')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Accuracy')
plt.xticks(k_values)
plt.grid()
plt.show()
```


KNN: Finding the Ideal k
Mean Accuracy