In [1]:
```
# Assignment: DSC550 Week10
# Name: Bezawada, Sashidhar
# Date: 2023-05-20
# 8.2 Term Project: Term Project Milestone 3: Model Building and Evaluation
```

In [2]:
```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:
```
#Load the dataset as a Pandas data frame
df_brainstroke= pd.read_csv("datasets/week6/brain_stroke.csv")
```

In [4]:
```
#Display the first ten rows of data
df_brainstroke.head(10)
```

Out[4]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_g
0	9046	Male	67.0	0	1	Yes	Private	Urban	
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	
2	31112	Male	80.0	0	1	Yes	Private	Rural	
3	60182	Female	49.0	0	0	Yes	Private	Urban	
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	
5	56669	Male	81.0	0	0	Yes	Private	Urban	
6	53882	Male	74.0	1	1	Yes	Private	Rural	
7	10434	Female	69.0	0	0	No	Private	Urban	
8	27419	Female	59.0	0	0	Yes	Private	Rural	
9	60491	Female	78.0	0	0	Yes	Private	Urban	

In [5]:
```
#Find the information in the data frame.
df_brainstroke.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
```

In [6]:
```
df_brainstroke['gender'].unique()
```
Out[6]:
```
array(['Male', 'Female', 'Other'], dtype=object)
```
In [7]:
```
df_brainstroke['hypertension'].unique()
```
Out[7]:
```
array([0, 1], dtype=int64)
```
In [8]:
```
df_brainstroke['heart_disease'].unique()
```
Out[8]:
```
array([1, 0], dtype=int64)
```
In [9]:
```
df_brainstroke['ever_married'].unique()
```
Out[9]:
```
array(['Yes', 'No'], dtype=object)
```
In [10]:
```
df_brainstroke['work_type'].unique()
```
Out[10]:
```
array(['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked'],
      dtype=object)
```
In [11]:
```
df_brainstroke['stroke'].unique()
```
Out[11]:
```
array([1, 0], dtype=int64)
```

MILESTONE 2

Data Transformation

1. Dropping Columns/Features

Dropping column Id as this a unique number given to each record (can be patient id or random generated number or serial number). It has no significance in decision making.

In [12]:
```
df_brainstroke = df_brainstroke.drop('id',axis=1)
```

2. Replacing values in a cloumn

Changing work type 'Children' as 'Student' as Children is not work type and Changing formatting for other work types. Changing binary values to string values of Yes and No instead of 1 and 0 respectively for hypertension and heart_disease.

In [14]:
```
df_brainstroke['work_type'].unique()
```

Out[14]:
```
array(['Private', 'Self Employed', 'Government', 'Student',
       'Never_worked'], dtype=object)
```

In ...:
```
#Changing binary values to string values of Yes and No instead of 1 and 0 respect
df_brainstroke['hypertension'] = df_brainstroke['hypertension'].replace([1], 'Yes
df_brainstroke['hypertension'] = df_brainstroke['hypertension'].replace([0], 'No'

df_brainstroke['heart_disease'] = df_brainstroke['heart_disease'].replace([1], 'Y
df_brainstroke['heart_disease'] = df_brainstroke['heart_disease'].replace([0], 'N
```

In [16]:
```
df_brainstroke['hypertension'].unique()
```

Out[16]:
```
array(['No', 'Yes'], dtype=object)
```

In [17]:
```
df_brainstroke['heart_disease'].unique()
```

Out[17]:
```
array(['Yes', 'No'], dtype=object)
```

In [18]:
```
df_brainstroke['stroke'].unique()
```

Out[18]:
```
array([1, 0], dtype=int64)
```

In [19]:
```
df_brainstroke['smoking_status'].unique()
```

Out[19]:
```
array(['formerly smoked', 'never smoked', 'smokes', 'Unknown'],
      dtype=object)
```

3. Renaming Column Name

Changing column name stroke to brain_stroke as this is more meaningful

In [20]:
```
df_brainstroke = df_brainstroke.rename(columns={'stroke': 'brain_stroke'})
```

In [21]:
```
df_brainstroke.head(10)
```

Out[21]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_gluc
0	Male	67.0	No	Yes	Yes	Private	Urban	
1	Female	61.0	No	No	Yes	Self Employed	Rural	
2	Male	80.0	No	Yes	Yes	Private	Rural	
3	Female	49.0	No	No	Yes	Private	Urban	
4	Female	79.0	Yes	No	Yes	Self Employed	Rural	
5	Male	81.0	No	No	Yes	Private	Urban	
6	Male	74.0	Yes	Yes	Yes	Private	Rural	
7	Female	69.0	No	No	No	Private	Urban	
8	Female	59.0	No	No	Yes	Private	Rural	
9	Female	78.0	No	No	Yes	Private	Urban	

In [22]:
```
df_brainstroke.dtypes
```

Out[22]:
```
gender                object
age                  float64
hypertension          object
heart_disease         object
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
brain_stroke           int64
dtype: object
```

4. Transform features

As per BMI chart on National Heart, Lung and Blood Institute website (https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmi_tbl2.html), the maximum BMI is 54. So, any value above 54 is unrelaistic. So, replace values over 54 with 54 considering the person is obese.

In [23]:
```
np.max(df_brainstroke['bmi'])
```

Out[23]:
```
97.6
```

In [24]:
```
max_bmi = 54.0
df_brainstroke['bmi'][df_brainstroke['bmi']>max_bmi] = max_bmi
```

```
C:\Users\sashi_000\AppData\Local\Temp\ipykernel_15068\849835967.py:2: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_brainstroke['bmi'][df_brainstroke['bmi']>max_bmi] = max_bmi
```

In [25]:
```
np.max(df_brainstroke['bmi'])
```

Out[25]:
```
54.0
```

5. Engineer new useful features.

Adding new column stage_of_life based on the age as we can decide which age group has an impact

In [...
```
df_brainstroke['stage_of_life'] = np.where(df_brainstroke.age <= 2, 'Infant',
                                  np.where(df_brainstroke.age < 5, 'Toddler',
                                  np.where(df_brainstroke.age < 13, 'Child',
                                  np.where(df_brainstroke.age < 20, 'Teen',
                                  np.where(df_brainstroke.age < 40, 'Adult',
                                  np.where(df_brainstroke.age < 60, 'Middle Age A
```

In [27]:
```
df_brainstroke.head(10)
```

Out[27]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose
0	Male	67.0	No	Yes	Yes	Private	Urban	2
1	Female	61.0	No	No	Yes	Self Employed	Rural	2
2	Male	80.0	No	Yes	Yes	Private	Rural	1
3	Female	49.0	No	No	Yes	Private	Urban	1
4	Female	79.0	Yes	No	Yes	Self Employed	Rural	1
5	Male	81.0	No	No	Yes	Private	Urban	1
6	Male	74.0	Yes	Yes	Yes	Private	Rural	
7	Female	69.0	No	No	No	Private	Urban	
8	Female	59.0	No	No	Yes	Private	Rural	
9	Female	78.0	No	No	Yes	Private	Urban	

I am adding one more column 'weight_status' as bmi might have different values and difficult guage details on that. So, weight_status will categorize them into 3 different categories.Instead of bmi, I feel weight_status will be best suited for analysis.

In [...

```
df_brainstroke['weight_status'] = np.where(df_brainstroke.bmi < 18.5, 'Underweig
                                  np.where(df_brainstroke.bmi < 25.0, 'Healthy we
                                  np.where(df_brainstroke.bmi < 30, 'Overweight',
```

In [29]:

```
df_brainstroke.head(10)
```

Out[29]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose
0	Male	67.0	No	Yes	Yes	Private	Urban	2
1	Female	61.0	No	No	Yes	Self Employed	Rural	2
2	Male	80.0	No	Yes	Yes	Private	Rural	1
3	Female	49.0	No	No	Yes	Private	Urban	1
4	Female	79.0	Yes	No	Yes	Self Employed	Rural	1
5	Male	81.0	No	No	Yes	Private	Urban	1
6	Male	74.0	Yes	Yes	Yes	Private	Rural	
7	Female	69.0	No	No	No	Private	Urban	
8	Female	59.0	No	No	Yes	Private	Rural	
9	Female	78.0	No	No	Yes	Private	Urban	

6. Replace Null Values

In [30]:
```
df_brainstroke.isnull().sum()
```

Out[30]:
```
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
brain_stroke           0
stage_of_life          0
weight_status          0
dtype: int64
```

As there are 201 null values for bmi, if we remove them, it will reduce dataset drastically. So, I am replacing null values with median.

In [3...
```
df_brainstroke['bmi'] = df_brainstroke['bmi'].fillna(df_brainstroke['bmi'].medi
```

In [32]:
```
df_brainstroke.isnull().sum()
```

Out[32]:
```
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
brain_stroke         0
stage_of_life        0
weight_status        0
dtype: int64
```

7. String Values case change

Finally, changing string values to UPPER CASE to avoid same values due to different cases falling into 2 different categories

In [33]:
```
df_brainstroke = df_brainstroke.apply(lambda x: x.astype(str).str.upper())
```

In [34]:
```
np.sort(df_brainstroke['bmi'].unique())
```

```
Out[34]:array(['10.3', '11.3', '11.5', '12.0', '12.3', '12.8', '13.0', '13.2',
               '13.3', '13.4', '13.5', '13.7', '13.8', '13.9', '14.0', '14.1',
               '14.2', '14.3', '14.4', '14.5', '14.6', '14.8', '14.9', '15.0',
               '15.1', '15.2', '15.3', '15.4', '15.5', '15.6', '15.7', '15.8',
               '15.9', '16.0', '16.1', '16.2', '16.3', '16.4', '16.5', '16.6',
               '16.7', '16.8', '16.9', '17.0', '17.1', '17.2', '17.3', '17.4',
               '17.5', '17.6', '17.7', '17.8', '17.9', '18.0', '18.1', '18.2',
               '18.3', '18.4', '18.5', '18.6', '18.7', '18.8', '18.9', '19.0',
               '19.1', '19.2', '19.3', '19.4', '19.5', '19.6', '19.7', '19.8',
               '19.9', '20.0', '20.1', '20.2', '20.3', '20.4', '20.5', '20.6',
               '20.7', '20.8', '20.9', '21.0', '21.1', '21.2', '21.3', '21.4',
               '21.5', '21.6', '21.7', '21.8', '21.9', '22.0', '22.1', '22.2',
               '22.3', '22.4', '22.5', '22.6', '22.7', '22.8', '22.9', '23.0',
               '23.1', '23.2', '23.3', '23.4', '23.5', '23.6', '23.7', '23.8',
               '23.9', '24.0', '24.1', '24.2', '24.3', '24.4', '24.5', '24.6',
               '24.7', '24.8', '24.9', '25.0', '25.1', '25.2', '25.3', '25.4',
               '25.5', '25.6', '25.7', '25.8', '25.9', '26.0', '26.1', '26.2',
               '26.3', '26.4', '26.5', '26.6', '26.7', '26.8', '26.9', '27.0',
               '27.1', '27.2', '27.3', '27.4', '27.5', '27.6', '27.7', '27.8',
               '27.9', '28.0', '28.1', '28.2', '28.3', '28.4', '28.5', '28.6',
               '28.7', '28.8', '28.9', '29.0', '29.1', '29.2', '29.3', '29.4',
               '29.5', '29.6', '29.7', '29.8', '29.9', '30.0', '30.1', '30.2',
               '30.3', '30.4', '30.5', '30.6', '30.7', '30.8', '30.9', '31.0',
               '31.1', '31.2', '31.3', '31.4', '31.5', '31.6', '31.7', '31.8',
               '31.9', '32.0', '32.1', '32.2', '32.3', '32.4', '32.5', '32.6',
               '32.7', '32.8', '32.9', '33.0', '33.1', '33.2', '33.3', '33.4',
               '33.5', '33.6', '33.7', '33.8', '33.9', '34.0', '34.1', '34.2',
               '34.3', '34.4', '34.5', '34.6', '34.7', '34.8', '34.9', '35.0',
               '35.1', '35.2', '35.3', '35.4', '35.5', '35.6', '35.7', '35.8',
               '35.9', '36.0', '36.1', '36.2', '36.3', '36.4', '36.5', '36.6',
               '36.7', '36.8', '36.9', '37.0', '37.1', '37.2', '37.3', '37.4',
               '37.5', '37.6', '37.7', '37.8', '37.9', '38.0', '38.1', '38.2',
               '38.3', '38.4', '38.5', '38.6', '38.7', '38.8', '38.9', '39.0',
               '39.1', '39.2', '39.3', '39.4', '39.5', '39.6', '39.7', '39.8',
               '39.9', '40.0', '40.1', '40.2', '40.3', '40.4', '40.5', '40.6',
               '40.7', '40.8', '40.9', '41.0', '41.1', '41.2', '41.3', '41.4',
               '41.5', '41.6', '41.7', '41.8', '41.9', '42.0', '42.1', '42.2',
               '42.3', '42.4', '42.5', '42.6', '42.7', '42.8', '42.9', '43.0',
               '43.1', '43.2', '43.3', '43.4', '43.6', '43.7', '43.8', '43.9',
               '44.0', '44.1', '44.2', '44.3', '44.4', '44.5', '44.6', '44.7',
               '44.8', '44.9', '45.0', '45.1', '45.2', '45.3', '45.4', '45.5',
               '45.7', '45.8', '45.9', '46.0', '46.1', '46.2', '46.3', '46.4',
               '46.5', '46.6', '46.8', '46.9', '47.1', '47.3', '47.4', '47.5',
               '47.6', '47.8', '47.9', '48.0', '48.1', '48.2', '48.3', '48.4',
               '48.5', '48.7', '48.8', '48.9', '49.2', '49.3', '49.4', '49.5',
               '49.8', '49.9', '50.1', '50.2', '50.3', '50.4', '50.5', '50.6',
               '50.8', '50.9', '51.0', '51.5', '51.7', '51.8', '51.9', '52.3',
               '52.5', '52.7', '52.8', '52.9', '53.4', '53.5', '53.8', '53.9',
               '54.0'], dtype=object)
```

Final dataset after Data preparation:

```
In [35]: df_brainstroke
```

Out[35]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_
0	MALE	67.0	NO	YES	YES	PRIVATE	URBAN	
1	FEMALE	61.0	NO	NO	YES	SELF EMPLOYED	RURAL	
2	MALE	80.0	NO	YES	YES	PRIVATE	RURAL	
3	FEMALE	49.0	NO	NO	YES	PRIVATE	URBAN	
4	FEMALE	79.0	YES	NO	YES	SELF EMPLOYED	RURAL	
...	...	...	...	...	...	...	...	
5105	FEMALE	80.0	YES	NO	YES	PRIVATE	URBAN	
5106	FEMALE	81.0	NO	NO	YES	SELF EMPLOYED	URBAN	
5107	FEMALE	35.0	NO	NO	YES	SELF EMPLOYED	RURAL	
5108	MALE	51.0	NO	NO	YES	PRIVATE	RURAL	
5109	FEMALE	44.0	NO	NO	YES	GOVERNMENT	URBAN	

5110 rows × 13 columns

In [...
```
df_brainstroke = df_brainstroke.astype({'age': 'float', 'avg_glucose_level': 'fl
```

In [37]:
```
df_brainstroke.dtypes
```

Out[37]:
```
gender                object
age                  float64
hypertension          object
heart_disease         object
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
brain_stroke          object
stage_of_life         object
weight_status         object
dtype: object
```

In [38]:
```
df_brainstroke
```

Out[38]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_
0	MALE	67.0	NO	YES	YES	PRIVATE	URBAN	
1	FEMALE	61.0	NO	NO	YES	SELF EMPLOYED	RURAL	
2	MALE	80.0	NO	YES	YES	PRIVATE	RURAL	
3	FEMALE	49.0	NO	NO	YES	PRIVATE	URBAN	
4	FEMALE	79.0	YES	NO	YES	SELF EMPLOYED	RURAL	
...	...	...	...	...	...	...	...	
5105	FEMALE	80.0	YES	NO	YES	PRIVATE	URBAN	
5106	FEMALE	81.0	NO	NO	YES	SELF EMPLOYED	URBAN	
5107	FEMALE	35.0	NO	NO	YES	SELF EMPLOYED	RURAL	
5108	MALE	51.0	NO	NO	YES	PRIVATE	RURAL	
5109	FEMALE	44.0	NO	NO	YES	GOVERNMENT	URBAN	

5110 rows × 13 columns

In ...
```
df_brain_one_hot = pd.get_dummies(data=df_brainstroke,columns=['gender','hyperten
                                                                 'work_type','Residence_t
                                                                 'weight_status'])
```

In [40]:
```
df_brain_one_hot
```

Out[40]:

	age	avg_glucose_level	bmi	brain_stroke	gender_FEMALE	gender_MALE	gender_OTHER	hy
0	67.0	228.69	36.6	1	0	1	0	
1	61.0	202.21	28.1	1	1	0	0	
2	80.0	105.92	32.5	1	0	1	0	
3	49.0	171.23	34.4	1	1	0	0	
4	79.0	174.12	24.0	1	1	0	0	
...	...	...	...	...	...	...	...	
5105	80.0	83.75	28.1	0	1	0	0	
5106	81.0	125.20	40.0	0	1	0	0	
5107	35.0	82.99	30.6	0	1	0	0	
5108	51.0	166.29	25.6	0	0	1	0	
5109	44.0	85.28	26.2	0	1	0	0	

5110 rows × 35 columns

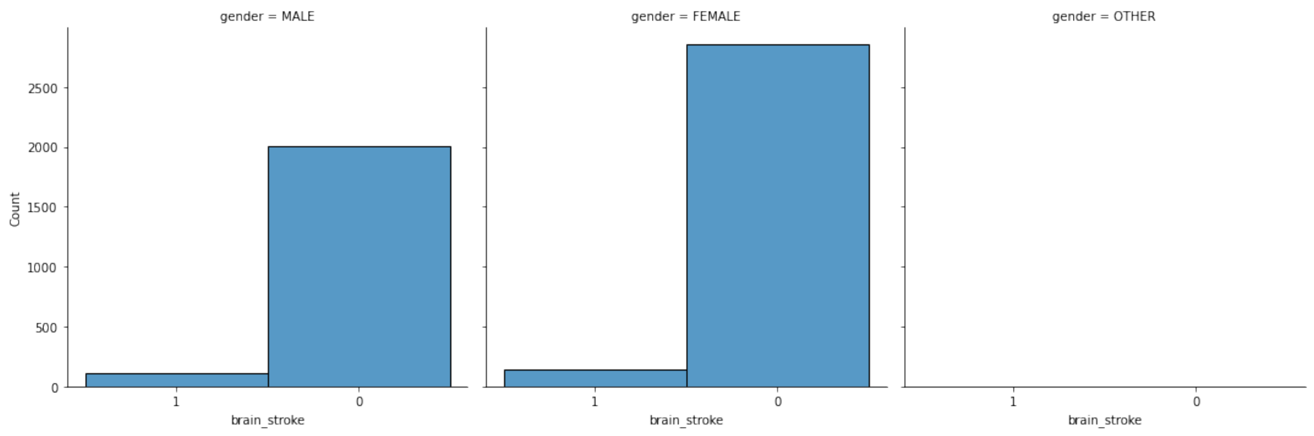
In [41]:
```
df_brain_one_hot.dtypes
```

```
Out[41]:age                              float64
        avg_glucose_level                float64
        bmi                              float64
        brain_stroke                      object
        gender_FEMALE                      uint8
        gender_MALE                        uint8
        gender_OTHER                       uint8
        hypertension_NO                    uint8
        hypertension_YES                   uint8
        heart_disease_NO                   uint8
        heart_disease_YES                  uint8
        ever_married_NO                    uint8
        ever_married_YES                   uint8
        work_type_GOVERNMENT               uint8
        work_type_NEVER_WORKED             uint8
        work_type_PRIVATE                  uint8
        work_type_SELF EMPLOYED            uint8
        work_type_STUDENT                  uint8
        Residence_type_RURAL               uint8
        Residence_type_URBAN               uint8
        smoking_status_FORMERLY SMOKED     uint8
        smoking_status_NEVER SMOKED        uint8
        smoking_status_SMOKES              uint8
        smoking_status_UNKNOWN             uint8
        stage_of_life_ADULT                uint8
        stage_of_life_CHILD                uint8
        stage_of_life_INFANT               uint8
        stage_of_life_MIDDLE AGE ADULT     uint8
        stage_of_life_SENIOR ADULT         uint8
        stage_of_life_TEEN                 uint8
        stage_of_life_TODDLER              uint8
        weight_status_HEALTHY WEIGHT       uint8
        weight_status_OBESITY              uint8
        weight_status_OVERWEIGHT           uint8
        weight_status_UNDERWEIGHT          uint8
        dtype: object
```

Data Visualization

```
In [42]: '''plt.figure()
         plt.hist(df.loc[df_brainstroke['brain_stroke']=='Yes','gender'],align='mid')
         plt.xlabel('Gender')
         plt.ylabel('Count of Brain Strokes')
         plt.title('Comparing Brain Stroke in patients based on the gender category')
         plt.show()'''
         import seaborn as sns
         sns.displot(df_brainstroke, x="brain_stroke", col="gender")
```

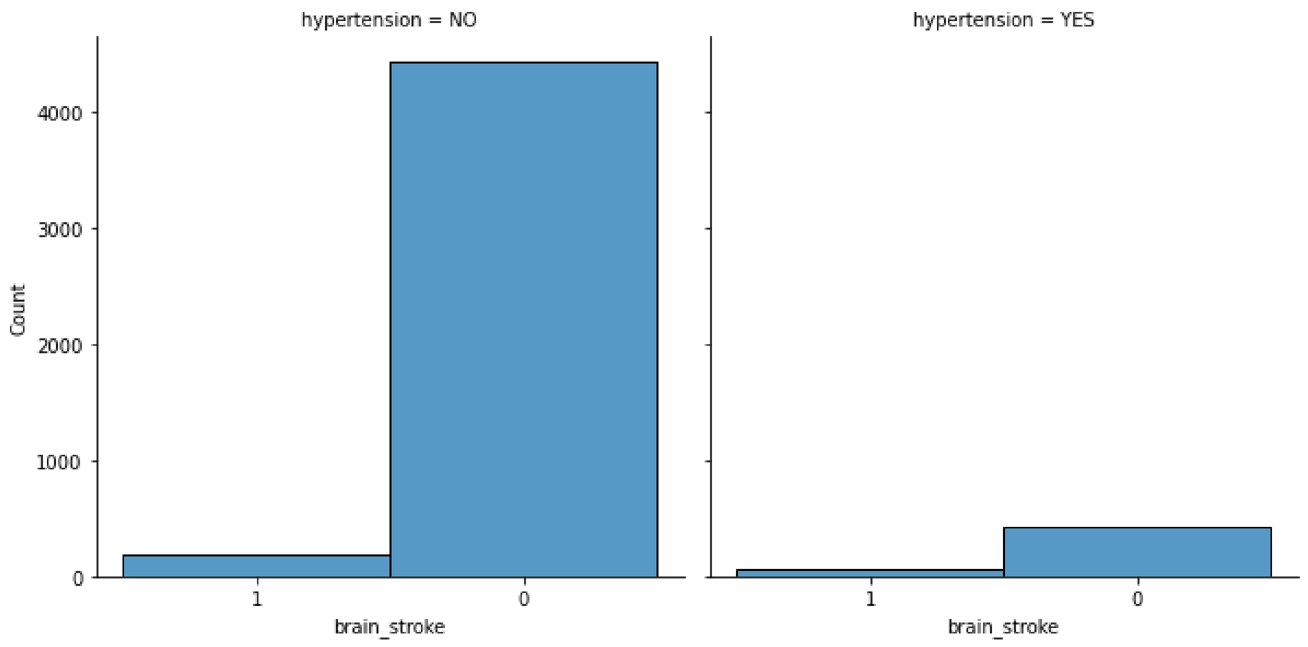
```
Out[42]:<seaborn.axisgrid.FacetGrid at 0x19daa4447f0>
```



In [4...

```
'''plt.figure()
plt.hist(df.loc[df_brainstroke['brain_stroke']=='Yes','hypertension'],align='mi
plt.xlabel('Hypertension')
plt.ylabel('Count of Brain Strokes')
plt.title('1.   Comparing Brain Stroke in patients based on hypertension')
plt.show()'''
import seaborn as sns
sns.displot(df_brainstroke, x="brain_stroke", col="hypertension")
```

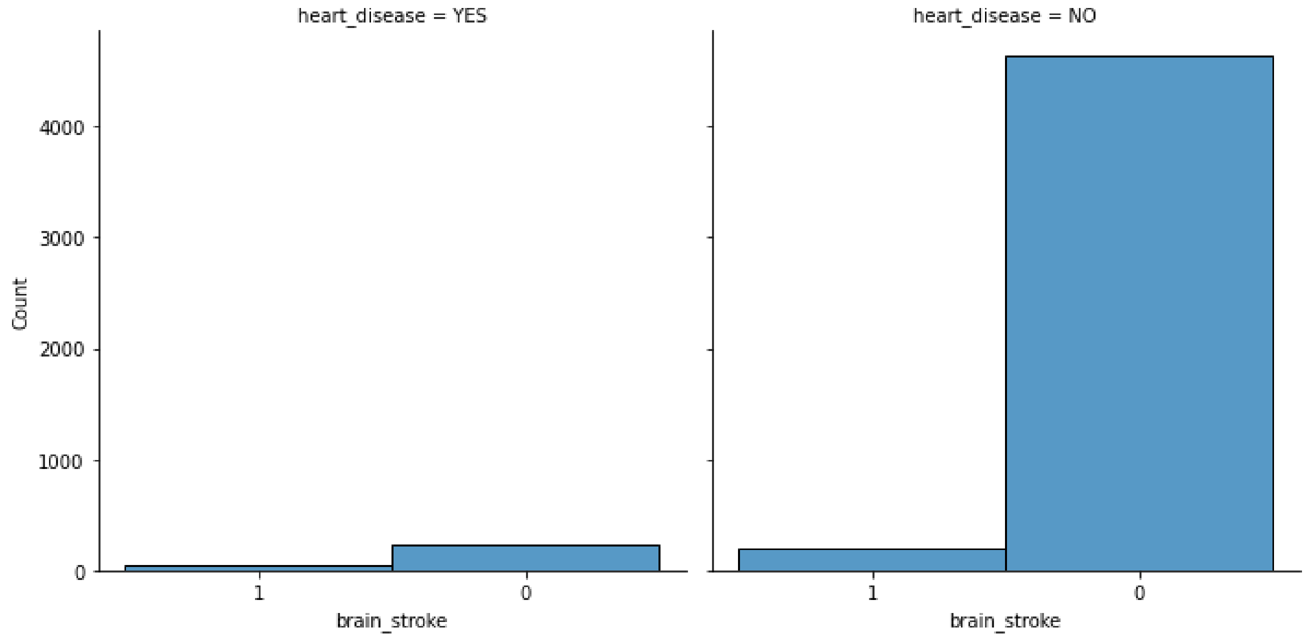
Out[43]:<seaborn.axisgrid.FacetGrid at 0x19daa4442b0>



In [4...

```
'''plt.figure()
plt.hist(df.loc[df_brainstroke['brain_stroke']=='Yes','heart_disease'],align='m
plt.xlabel('Heart Disease')
plt.ylabel('Count of Brain Strokes')
plt.title('1.   Comparing Brain Stroke in patients based on heart disease')
plt.show()'''
import seaborn as sns
sns.displot(df_brainstroke, x="brain_stroke", col="heart_disease")
```

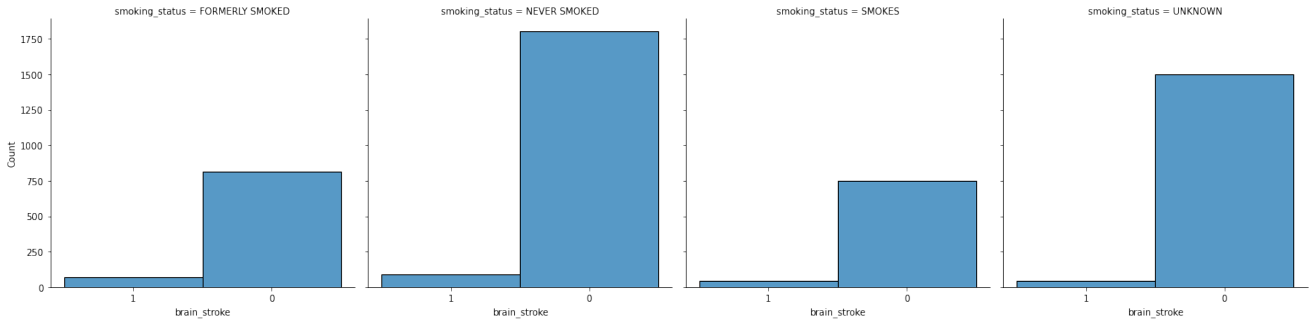
Out[44]:<seaborn.axisgrid.FacetGrid at 0x19dab90d970>



As per analysis, With heart disease history have high chances of getting Brain stroke as the grph shows heart disease is directly proportional to Brain Stroke.

In [4…
```
'''plt.figure()
plt.hist(df.loc[df_brainstroke['brain_stroke']=='Yes','smoking_status'],align='r
plt.xlabel('Smoking Status')
plt.ylabel('Count of Brain Strokes')
plt.title('Comparing Brain Stroke in patients based on smoking status')
plt.show()'''
import seaborn as sns
sns.displot(df_brainstroke, x="brain_stroke", col="smoking_status")
```

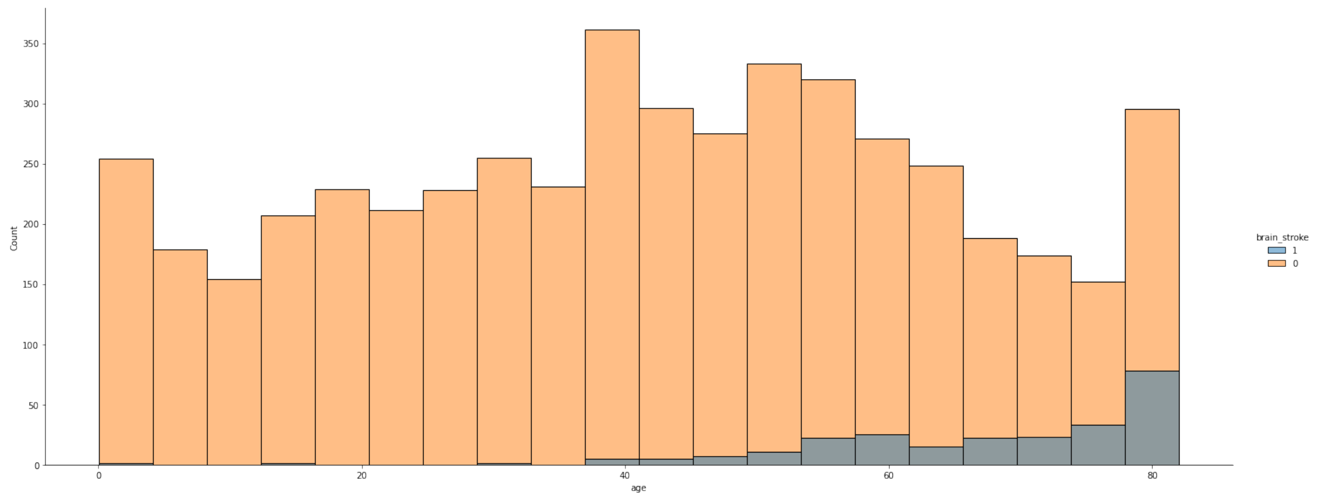
Out[45]:
```
<seaborn.axisgrid.FacetGrid at 0x19daba425b0>
```



As per graphical representation, we can confirm that non-smokers have very little chances of getting Brain stroke

In [46]:
```
'''plt.rcParams["figure.figsize"] = (20,8)
plt.hist(df.loc[df_brainstroke['brain_stroke']!='No','age'])
plt.xlabel('Age')
plt.ylabel('Count of Brain Strokes')
plt.title('Comparing Brain Stroke in patients based on age')
plt.show()'''
sns.displot(df_brainstroke, x="age", hue="brain_stroke",height=8,aspect=20/8)
```

Out[46]:
```
<seaborn.axisgrid.FacetGrid at 0x19dab925730>
```



As per the grapical analysis, people over the age of 40 have high chances of getting Brain stroke

MILESTONE 3

Building a model and evaluating

In [47]:
```
#importing libraries
# Core libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Sklearn processing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Sklearn regression algorithms
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

# Sklearn regression model evaluation function
from sklearn.metrics import mean_absolute_error

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder

from sklearn.datasets import make_classification
```

In [48]:
```
cat_columns = df_brain_one_hot.select_dtypes(include='object').keys()
cat_columns
```

Out[48]:
```
Index(['brain_stroke'], dtype='object')
```

In [4...
```
# creating instance of one-hot-encoder
enc = OneHotEncoder(handle_unknown='ignore')
```

Out[49]:

	0	1
0	0.0	1.0
1	0.0	1.0
2	0.0	1.0
3	0.0	1.0
4	0.0	1.0
...	...	...
5105	1.0	0.0
5106	1.0	0.0
5107	1.0	0.0
5108	1.0	0.0
5109	1.0	0.0

5110 rows × 2 columns

In [50]:

```
# merge with main df df_brainstroke on key values
df_brainstroke = df_brainstroke.join(enc_df)
df_brainstroke
```

Out[50]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_
0	MALE	67.0	NO	YES	YES	PRIVATE	URBAN	
1	FEMALE	61.0	NO	NO	YES	SELF EMPLOYED	RURAL	
2	MALE	80.0	NO	YES	YES	PRIVATE	RURAL	
3	FEMALE	49.0	NO	NO	YES	PRIVATE	URBAN	
4	FEMALE	79.0	YES	NO	YES	SELF EMPLOYED	RURAL	
...	...	...	...	...	...	...	...	
5105	FEMALE	80.0	YES	NO	YES	PRIVATE	URBAN	
5106	FEMALE	81.0	NO	NO	YES	SELF EMPLOYED	URBAN	
5107	FEMALE	35.0	NO	NO	YES	SELF EMPLOYED	RURAL	
5108	MALE	51.0	NO	NO	YES	PRIVATE	RURAL	
5109	FEMALE	44.0	NO	NO	YES	GOVERNMENT	URBAN	

5110 rows × 15 columns

In...
```
df_brainstroke['gender'] = df_brainstroke['gender'].map(dict(zip(['MALE','FEMALE',
df_brainstroke['hypertension'] = df_brainstroke['hypertension'].map(dict(zip(['YES
df_brainstroke['heart_disease'] = df_brainstroke['heart_disease'].map(dict(zip(['Y
df_brainstroke['ever_married'] = df_brainstroke['ever_married'].map(dict(zip(['YES
df_brainstroke['work_type'] = df_brainstroke['work_type'].map(dict(zip(['PRIVATE',
       'NEVER_WORKED'],[0,1,2,3,4])))
df_brainstroke['Residence_type'] = df_brainstroke['Residence_type'].map(dict(zip([
df_brainstroke['smoking_status'] = df_brainstroke['smoking_status'].map(dict(zip([
df_brainstroke['stage_of_life'] = df_brainstroke['stage_of_life'].map(dict(zip(['S
       'TODDLER', 'CHILD'],[0,1,2,3,4,5,6])))
df_brainstroke['weight_status'] = df_brainstroke['weight_status'].map(dict(zip(['C
```

In [52]:
```
#df_brain = df_brain.dropna()
y = df_brain_one_hot["brain_stroke"]
X = df_brain_one_hot.drop('brain_stroke', axis=1)
```

In [53]:
```
# Split dataset into random train and test subsets:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [54]:
```
#pip install imbalanced-learn
```

In [56]:
```
# summarize the new class distribution
from collections import Counter
counter = Counter(y_train)
print(counter)
counter = Counter(y_test)
print(counter)
counter = Counter(y_res)
print(counter)
```

```
Counter({'0': 3895, '1': 193})
Counter({'0': 966, '1': 56})
Counter({'0': 3895, '1': 3895})
```

In [57]:
```
# Standardize features by removing mean and scaling to unit variance:
scaler = StandardScaler()
scaler.fit(X_train)
```

Out[57]:
```
StandardScaler()
```

In [58]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
X_res = scaler.fit_transform(X_res)
```

2. KNN classifier:

In [59]:
```
# Use the KNN classifier to fit data:
classifier = KNeighborsClassifier()
classifier.fit(X_res, y_res)
```

Out[59]:
```
KNeighborsClassifier()
```

In [60]:
```
# Predict y data with classifier:
y_predict = classifier.predict(X_test)
```

In [61]:
```
# Print results:
print(confusion_matrix(y_test, y_predict))
print(classification_report(y_test, y_predict))
```

```
[[937  29]
 [ 49   7]]
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       966
           1       0.19      0.12      0.15        56

    accuracy                           0.92      1022
   macro avg       0.57      0.55      0.56      1022
weighted avg       0.91      0.92      0.92      1022
```

In [62]:
```
knn_score = accuracy_score(y_test, y_predict, normalize=False)
knn_score
```

Out[62]:
```
944
```

In [63]:
```
# print the shapes of the new X objects
print(X_train.shape)
print(X_test.shape)
```

```
(4088, 34)
(1022, 34)
```

In [64]:
```
# print the shapes of the new y objects
print(y_train.shape)
print(y_test.shape)
```

```
(4088,)
(1022,)
```

In [65]:
```
#Importing Libraries
from sklearn.linear_model import LogisticRegression
```

In [66]:
```
#train the model on the training set
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

Out[66]:
```
LogisticRegression()
```

In [67]:
```
import sklearn.metrics as metrics
#make predictions on the testing set
y_pred = logreg.predict(X_test)

print(accuracy_score(y_test, y_pred))
```

```
0.9452054794520548
```

In [68]:
```
#Repeat for KNN with K=5:
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.9442270058708415
```

In [69]:
```
#Repeat for KNN with K=1:
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.9442270058708415
```

In [70]:
```
# try K=1 through K=35 and record testing accuracy
k_range = range(1, 36)

# We can create Python dictionary using [] or dict()
scores = []

# We use a loop through the range 1 to 36
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

print("Accuracy Score of KNN Classifier:  " + str(scores))
```

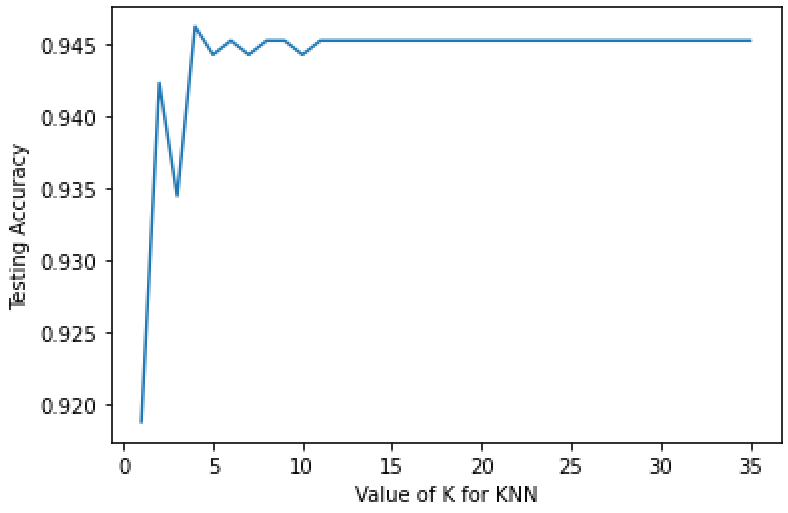
```
Accuracy Score of KNN Classifier:  [0.9187866927592955, 0.9422700587084148, 0.9344422
700587084, 0.9461839530332681, 0.9442270058708415, 0.9452054794520548, 0.944227005870
8415, 0.9452054794520548, 0.9452054794520548, 0.9442270058708415, 0.9452054794520548,
0.9452054794520548, 0.9452054794520548, 0.9452054794520548, 0.9452054794520548, 0.945
2054794520548, 0.9452054794520548, 0.9452054794520548, 0.9452054794520548, 0.94520547
94520548, 0.9452054794520548, 0.9452054794520548, 0.9452054794520548, 0.9452054794520
548, 0.9452054794520548, 0.9452054794520548, 0.9452054794520548, 0.9452054794520548,
0.9452054794520548, 0.9452054794520548, 0.9452054794520548, 0.9452054794520548, 0.945
2054794520548, 0.9452054794520548, 0.9452054794520548]
```

In [71]:
```python
# import Matplotlib (scientific plotting library)
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

Out[71]:
```
Text(0, 0.5, 'Testing Accuracy')
```



Summary:
As per the classification regression and KNeighborsRegressoion after evaluating the accuracy of the training model we got the output around 0.95

Decision Tree Classifier:

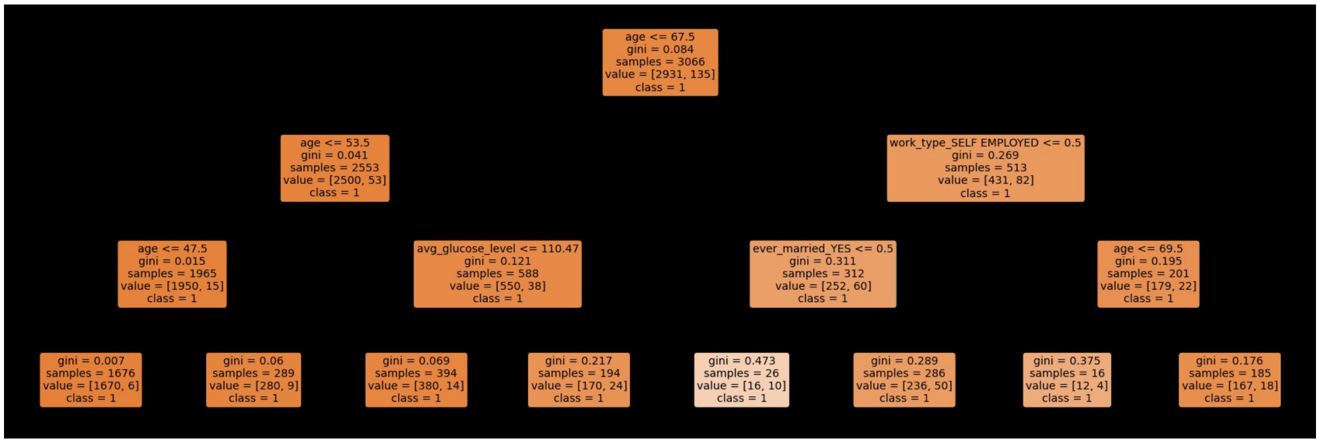
In [72]:
```python
#Import Libraries
from sklearn.tree import DecisionTreeClassifier
```

In [73]:
```python
#save the feature name and target variables
feature_names = X.columns
labels = y.unique()
#split the dataset
from sklearn.model_selection import train_test_split
X_train, test_x, y_train, test_lab = train_test_split(X,y,
```

In [74]:
```
clf = DecisionTreeClassifier(max_depth =3, random_state = 42)
clf.fit(X_train, y_train)
```

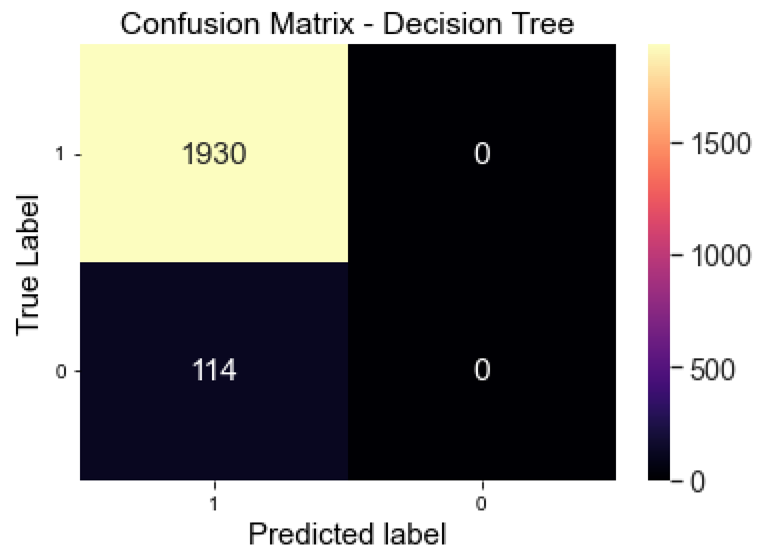
Out[74]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In [75]:
```
#import relevant packages
from sklearn import tree
import matplotlib.pyplot as plt
#plt the figure, setting a black background
plt.figure(figsize=(30,10), facecolor ='k')
#create the tree plot
a = tree.plot_tree(clf,
                   #use the feature names stored
                   feature_names = feature_names,
                   #use the class names stored
                   class_names = labels,
                   rounded = True,
                   filled = True,
                   fontsize=14)
#show the plot
plt.show()
```



In [76]:
```
test_pred_decision_tree = clf.predict(test_x)
```

In [77]:
```
#import the relevant packages
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
#get the confusion matrix
confusion_matrix = metrics.confusion_matrix(test_lab,
                                            test_pred_decision_tree)
#turn this into a dataframe
matrix_df = pd.DataFrame(confusion_matrix)
#plot the result
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10,7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
#set axis titles
ax.set_title('Confusion Matrix - Decision Tree')
ax.set_xlabel("Predicted label", fontsize =15)
```



```
<Figure size 720x504 with 0 Axes>
```

In [... `print("Accuracy Score of Decision Tree Classifier:  " + str(accuracy_score(test_`

```
Accuracy Score of Decision Tree Classifier:  0.9442270058708415
```

Random Forest Classifier

In [...
```
#Split the data
training, testing, training_labels, testing_labels = train_test_split(X, y, test_
```

In [8...
```
# Normalize the data
sc = StandardScaler()
normed_train_data = pd.DataFrame(sc.fit_transform(training), columns = X.column
normed_test_data = pd.DataFrame(sc.fit_transform(testing), columns = X.columns)
```

In [81]:
```
#Building Random Forest model

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

clf=RandomForestClassifier()
clf.fit(training, training_labels)
```

Out[81]:
```
RandomForestClassifier()
```

In [82]:
```
preds = clf.predict(testing)
```

In [83]:
```
print(clf.score(testing, testing_labels))
```

```
0.9374021909233177
```

In [... `print("Accuracy Score of Random Forest Classifier:  " + str(accuracy_score(testi`

```
Accuracy Score of Random Forest Classifier:  0.9374021909233177
```

In [85]:
```
confusion_matrix(testing_labels, preds)
```

Out[85]:
```
array([[1197,    1],
       [  79,    1]], dtype=int64)
```

In [ ]: