Import all necessary open source libraries

In [1]:
```python
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
import statsmodels.api as sm
```

In [2]:
```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages (0.10.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages (from imbalanced-learn) (2.1.0)
Requirement already satisfied: joblib>=1.1.1 in /Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in /Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages (from imbalanced-learn) (1.2.1)
Requirement already satisfied: scipy>=1.3.2 in /Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages (from imbalanced-learn) (1.6.2)
Requirement already satisfied: numpy>=1.17.3 in /Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages (from imbalanced-learn) (1.20.1)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
conda install -c conda-forge imbalanced-learn
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.14.0
  latest version: 23.1.0

Please update conda by running

    $ conda update -n base -c conda-forge conda



# All requested packages already installed.

Retrieving notices: ...working... done

Note: you may need to restart the kernel to use updated packages.
```

Import the CSV data set

In [4]:
```python
stroke_dataset= pd.read_csv("healthcare-dataset-stroke-data.csv")
stroke_dataset
stroke_dataset.describe()
```

Out[4]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

In [5]:
```python
stroke_dataset= pd.read_csv("healthcare-dataset-stroke-data.csv")
print(stroke_dataset.columns)
```

```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

In [ ]:

Data Cleansing

The stroke_dataset is imported into the Juypter along with all the libraries. For the first step of this project I have done data cleaning. Seleveral steps to data cleaning are:

1. First I have fixed the data type of each column.
2. Secondly, I checked if the data set had any null values.
3. In the third step, I handled missing valus by delecting the records.
4. In the fourth step, I checked if there are any duplicate value especially in the id column.
5. I also delected the column 'ever_married' due to multiple missing values.

In [6]:
```python
Step1 = stroke_dataset.drop('ever_married', axis=1)
print(Step1)

convert_dict = {'id': int,
                'gender': str,
                'age': int,
                'hypertension': int,
                'heart_disease': int,
                'work_type': str,
                'Residence_type': str,
                'avg_glucose_level': float,
                'bmi': float,
                'smoking_status': str,
                'stroke': int}
Step2 = Step1.astype(convert_dict)
print(Step2.dtypes)
```

```
         id  gender   age  hypertension  heart_disease      work_type  \
0      9046    Male  67.0             0              1        Private
1     51676  Female  61.0             0              0  Self-employed
2     31112    Male  80.0             0              1        Private
3     60182  Female  49.0             0              0        Private
4      1665  Female  79.0             1              0  Self-employed
...     ...     ...   ...           ...            ...            ...
5105  18234  Female  80.0             1              0        Private
5106  44873  Female  81.0             0              0  Self-employed
5107  19723  Female  35.0             0              0  Self-employed
5108  37544    Male  51.0             0              0        Private
5109  44679  Female  44.0             0              0       Govt_job

     Residence_type  avg_glucose_level   bmi   smoking_status  stroke
0             Urban             228.69  36.6  formerly smoked       1
1             Rural             202.21   NaN     never smoked       1
2             Rural             105.92  32.5     never smoked       1
3             Urban             171.23  34.4           smokes       1
4             Rural             174.12  24.0     never smoked       1
...             ...                ...   ...              ...     ...
5105          Urban              83.75   NaN     never smoked       0
5106          Urban             125.20  40.0     never smoked       0
5107          Rural              82.99  30.6     never smoked       0
5108          Rural             166.29  25.6  formerly smoked       0
5109          Urban              85.28  26.2          Unknown       0

[5110 rows x 11 columns]
id                     int64
gender                object
age                    int64
hypertension           int64
heart_disease          int64
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

In [7]:
```python
Step2.isnull()
Step2.isnull().any()
Step2.isnull().sum()
```

Out[7]:
```
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

In [8]:
```python
Stroke_df = Step2[Step2['bmi'].notnull()]
print(Stroke_df)
```

```
         id  gender  age  hypertension  heart_disease      work_type  \
0      9046    Male   67             0              1        Private
2     31112    Male   80             0              1        Private
3     60182  Female   49             0              0        Private
4      1665  Female   79             1              0  Self-employed
5     56669    Male   81             0              0        Private
...     ...     ...  ...           ...            ...            ...
5104  14180  Female   13             0              0       children
5106  44873  Female   81             0              0  Self-employed
5107  19723  Female   35             0              0  Self-employed
5108  37544    Male   51             0              0        Private
5109  44679  Female   44             0              0       Govt_job

     Residence_type  avg_glucose_level   bmi   smoking_status  stroke
0             Urban             228.69  36.6  formerly smoked       1
2             Rural             105.92  32.5     never smoked       1
3             Urban             171.23  34.4           smokes       1
4             Rural             174.12  24.0     never smoked       1
5             Urban             186.21  29.0  formerly smoked       1
...             ...                ...   ...              ...     ...
5104          Rural             103.08  18.6          Unknown       0
5106          Urban             125.20  40.0     never smoked       0
5107          Rural              82.99  30.6     never smoked       0
5108          Rural             166.29  25.6  formerly smoked       0
5109          Urban              85.28  26.2          Unknown       0

[4909 rows x 11 columns]
```

Calculate IQR by finding Q1 and Q2

In second step check if there are any outliers

In [9]:
```python
Q1 = Stroke_df.quantile(0.25)
Q3 = Stroke_df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
id                   36615.0
age                     35.0
hypertension             0.0
heart_disease            0.0
avg_glucose_level       36.5
bmi                      9.6
stroke                   0.0
dtype: float64
```

In [10]:
```python
Lower_Limit = Q1 - (1.5 * IQR)
Upper_Limit = Q3 + (1.5 * IQR)
Outliers = Stroke_df [ ((Stroke_df < Lower_Limit) | ((Stroke_df > Upper_Limit))).any(axis=1) ]
print(Outliers)
```

```
         id  gender  age  hypertension  heart_disease      work_type  \
0      9046    Male   67             0              1        Private
2     31112    Male   80             0              1        Private
3     60182  Female   49             0              0        Private
4      1665  Female   79             1              0  Self-employed
5     56669    Male   81             0              0        Private
...     ...     ...  ...           ...            ...            ...
5064  66684    Male   70             0              0  Self-employed
5076  39935  Female   34             0              0        Private
5088  22190  Female   64             1              0  Self-employed
5091   6369    Male   59             1              0        Private
5100  68398    Male   82             1              0  Self-employed

     Residence_type  avg_glucose_level   bmi   smoking_status  stroke
0             Urban             228.69  36.6  formerly smoked       1
2             Rural             105.92  32.5     never smoked       1
3             Urban             171.23  34.4           smokes       1
4             Rural             174.12  24.0     never smoked       1
5             Urban             186.21  29.0  formerly smoked       1
...             ...                ...   ...              ...     ...
5064          Rural             193.88  24.3          Unknown       0
5076          Rural             174.37  23.0     never smoked       0
5088          Urban              76.89  30.2          Unknown       0
5091          Rural              95.05  30.9     never smoked       0
5100          Rural              71.97  28.3     never smoked       0

[1154 rows x 11 columns]
<ipython-input-10-bb8bdedf4a34>:3: FutureWarning: Automatic reindexing on DataFrame vs Series comparisons is deprecated and will raise ValueError in a future version. Do `left, right = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  Outliers = Stroke_df [ ((Stroke_df < Lower_Limit) | ((Stroke_df > Upper_Limit))).any(axis=1) ]
<ipython-input-10-bb8bdedf4a34>:3: FutureWarning: Automatic reindexing on DataFrame vs Series comparisons is deprecated and will raise ValueError in a future version. Do `left, right = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  Outliers = Stroke_df [ ((Stroke_df < Lower_Limit) | ((Stroke_df > Upper_Limit))).any(axis=1) ]
```

In [11]:
```python
Stroke_df.isnull().sum()
Stroke_df.shape
Stroke_df.duplicated(['id'])
```

Out[11]:
```
0       False
2       False
3       False
4       False
5       False
        ...
5104    False
5106    False
5107    False
5108    False
5109    False
Length: 4909, dtype: bool
```

Select columns bmi, age and avg_glucose_level and create box plots

In [12]:
```python
Num_columns = Stroke_df[["bmi", "age", "avg_glucose_level"]]
print(Num_columns)
```

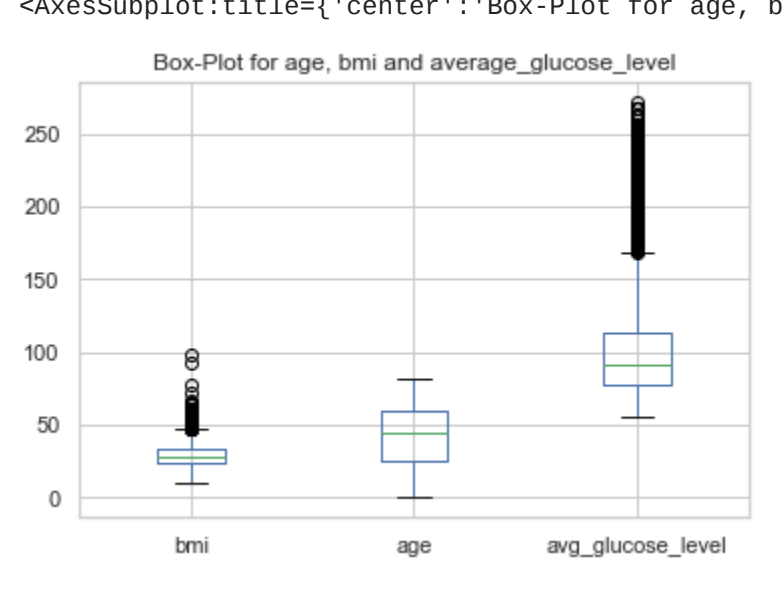
```
       bmi  age  avg_glucose_level
0     36.6   67             228.69
2     32.5   80             105.92
3     34.4   49             171.23
4     24.0   79             174.12
5     29.0   81             186.21
...    ...  ...                ...
5104  18.6   13             103.08
5106  40.0   81             125.20
5107  30.6   35              82.99
5108  25.6   51             166.29
5109  26.2   44              85.28

[4909 rows x 3 columns]
```

Visualize the data

1. Created box plots to show the data distribution.
2. Box-Plot for age, bmi and average_glucose_level

In [13]:
```python
Num_columns.plot(kind = "box", title = "Box-Plot for age, bmi and average_glucose_level")
```

Out[13]:
```
<AxesSubplot:title={'center':'Box-Plot for age, bmi and average_glucose_level'}>
```



1. The bmi in above box plot is above 25. This could be related to stroke and bmi above 25 is considered over weight.
2. The age is between 25 and 70. The age shows adult category therefore this could be our predictor.
3. The blood sugar level from the box plot is between 70 to 130 which is normal level for before and after meal. This probabbly is not a good predictor for stroke.

In [ ]:

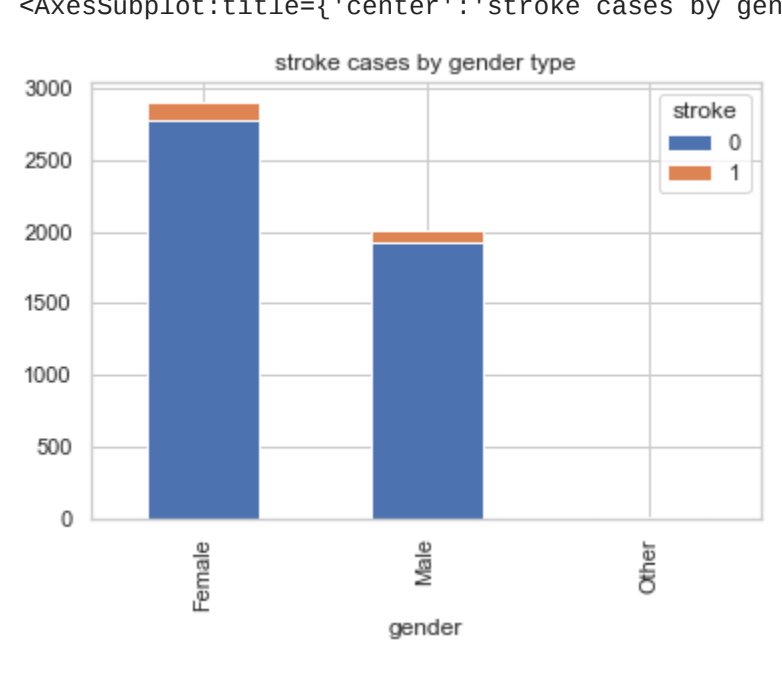
Used group by function to group variables. Used count function to count number of stroke cases in each subgroups.

1. Created bar charts and clustered bar charts for gender, work_type, Residence_type, smoking_status, heart_disease, hypertension columns to show the stroke cases.

In [14]:
```python
group_gender = Stroke_df.groupby(['gender','stroke'])['stroke'].count().reset_index(name="count")
print(group_gender)
group_gender.pivot(index = 'gender', columns = 'stroke', values = 'count').plot(kind = 'bar', stacked=True, title = 'stroke cases by gender type')
```

```
   gender  stroke  count
0  Female       0   2777
1  Female       1    120
2    Male       0   1922
3    Male       1     89
4   Other       0      1
```

Out[14]:
```
<AxesSubplot:title={'center':'stroke cases by gender type'}, xlabel='gender'>
```

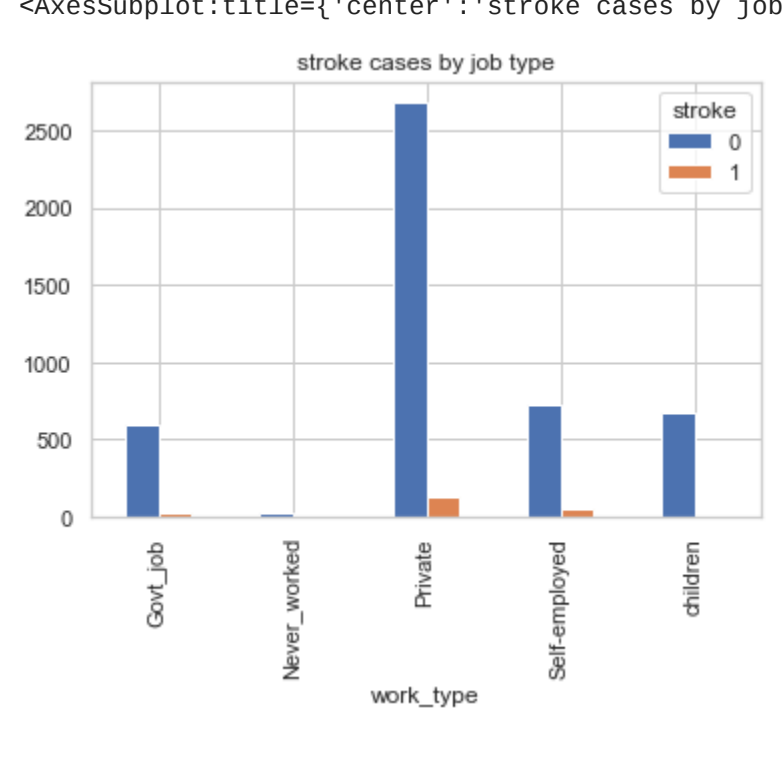


Female has more stroke cases than male. The clustered bar chart shows that the geneder could be good predictor for stroke.

In [15]:
```python
group_work = Stroke_df.groupby(["work_type", "stroke"])["stroke"].count().reset_index(name="count")
print(group_work)
group_work.pivot(index = 'work_type', columns = 'stroke', values = 'count').plot(kind = 'bar', title = 'stroke cases by job type')
```

```
       work_type  stroke  count
0       Govt_job       0    602
1       Govt_job       1     28
2   Never_worked       0     22
3        Private       0   2684
4        Private       1    127
5  Self-employed       0    722
6  Self-employed       1     53
7       children       0    670
8       children       1      1
```

Out[15]:
```
<AxesSubplot:title={'center':'stroke cases by job type'}, xlabel='work_type'>
```

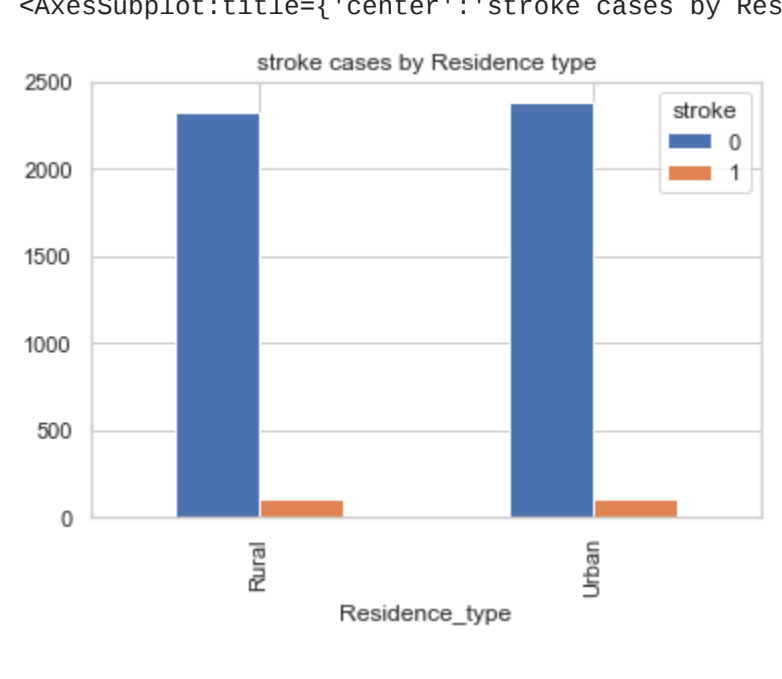


There is huge variation of stroke cases in work_type. Therefore, work_type could be our predictor.

In [16]:
```python
group_Residence_type = Stroke_df.groupby(["Residence_type", "stroke"])["stroke"].count().reset_index(name="count")
print(group_Residence_type)
group_Residence_type.pivot(index = 'Residence_type', columns = 'stroke', values = 'count').plot(kind = 'bar', title = 'stroke cases by Residence type')
```

```
  Residence_type  stroke  count
0          Rural       0   2319
1          Rural       1    100
2          Urban       0   2381
3          Urban       1    109
```

Out[16]:
```
<AxesSubplot:title={'center':'stroke cases by Residence type'}, xlabel='Residence_type'>
```

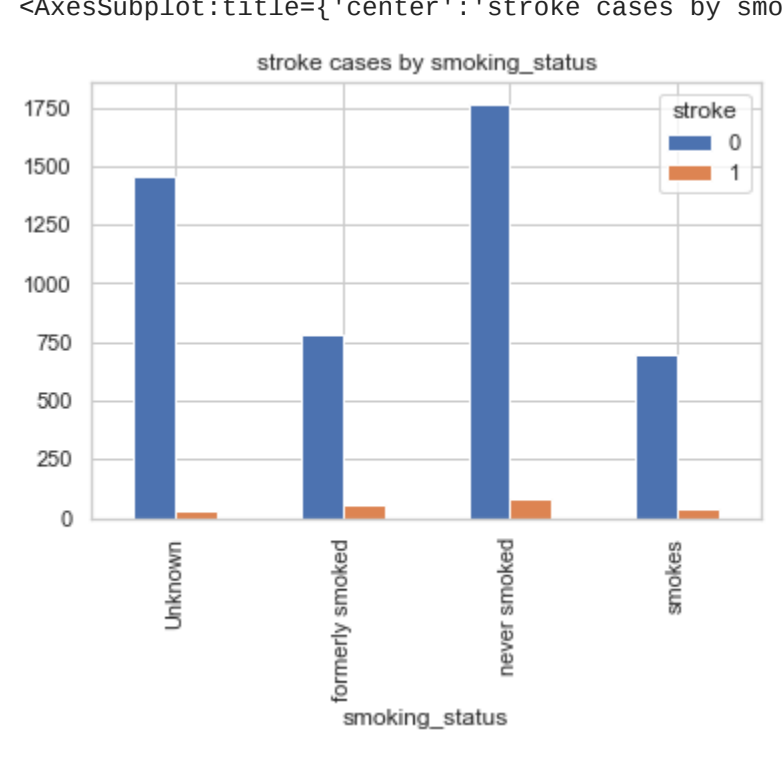


The Residence_type do not show much difference between rural and urban living areas. This probabbly is not our predicting variable.

In [17]:
```python
group_smoking_status = Stroke_df.groupby(["smoking_status", "stroke"])["stroke"].count().reset_index(name="count")
print(group_smoking_status)
group_smoking_status.pivot(index = 'smoking_status', columns = 'stroke', values = 'count').plot(kind = 'bar', title = 'stroke cases by smoking_status')
```

```
    smoking_status  stroke  count
0          Unknown       0   1454
1          Unknown       1     29
2  formerly smoked       0    780
3  formerly smoked       1     57
4     never smoked       0   1768
5     never smoked       1     84
6           smokes       0    698
7           smokes       1     39
```

Out[17]:
```
<AxesSubplot:title={'center':'stroke cases by smoking_status'}, xlabel='smoking_status'>
```

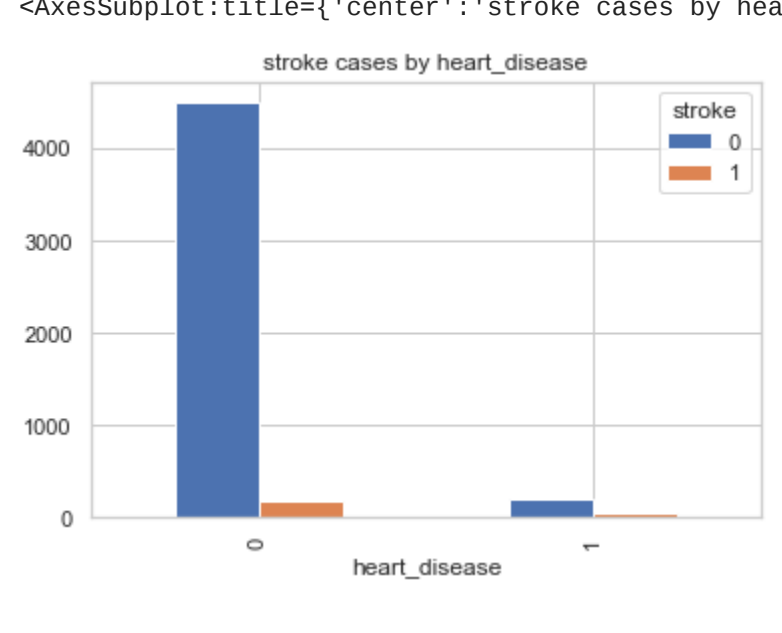


The stroke case is different in the people with different smoking habits. The smoking_status could be a predictor for stroke.

In [18]:
```python
group_heart_disease = Stroke_df.groupby(['heart_disease','stroke'])['stroke'].count().reset_index(name="count")
print(group_heart_disease)
group_heart_disease.pivot(index = 'heart_disease', columns = 'stroke', values = 'count').plot(kind = 'bar', title = 'stroke cases by heart_disease')
```

```
   heart_disease  stroke  count
0              0       0   4497
1              0       1    169
2              1       0    203
3              1       1     40
```

Out[18]:
```
<AxesSubplot:title={'center':'stroke cases by heart_disease'}, xlabel='heart_disease'>
```

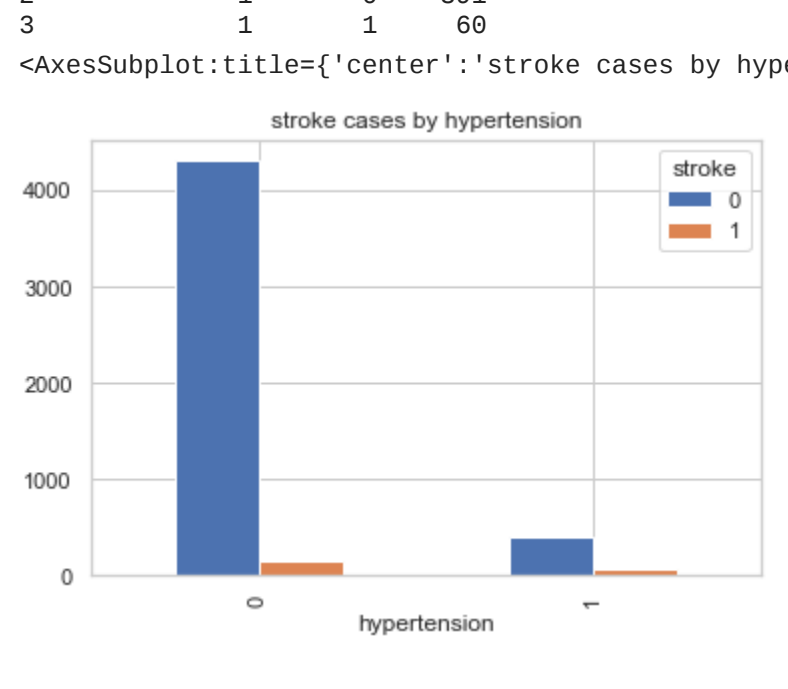


From the above bar chart we can see that people without the heart disease tend to have more stroke cases than the peope with the heart disease.

In [19]:
```python
group_hypertension = Stroke_df.groupby(['hypertension','stroke'])['stroke'].count().reset_index(name="count")
print(group_hypertension)
group_hypertension.pivot(index = 'hypertension', columns = 'stroke', values = 'count').plot(kind = 'bar', title = 'stroke cases by hypertension')
```

```
   hypertension  stroke  count
0             0       0   4309
1             0       1    149
2             1       0    391
3             1       1     60
```

Out[19]:
```
<AxesSubplot:title={'center':'stroke cases by hypertension'}, xlabel='hypertension'>
```



Above bar chart shows that people who do not have hypertension tend to have higher cases of stroke.

Created dummy variables

In this section, I have created dummy variables for columns like smoking status to convert categorical values into numerical values.

In [20]:
```python
categorical_vars=['gender','work_type','Residence_type','smoking_status', 'hypertension', 'heart_disease']
for var in categorical_vars:
    categorical_list='var'+'_'+var
    categorical_list = pd.get_dummies(Stroke_df[var], prefix=var)
    data1=Stroke_df.join(categorical_list)
    Stroke_df=data1
categorical_vars=['gender','work_type','Residence_type','smoking_status', 'hypertension', 'heart_disease']
data_vars=Stroke_df.columns.values.tolist()
to_keep=[i for i in data_vars if i not in categorical_vars]
```

Final Columns to keep

In [21]:
```python
data_final=Stroke_df[to_keep]
data_final.columns.values
```

Out[21]:
```
array(['id', 'age', 'avg_glucose_level', 'bmi', 'stroke', 'gender_Female',
       'gender_Male', 'gender_Other', 'work_type_Govt_job',
       'work_type_Never_worked', 'work_type_Private',
       'work_type_Self-employed', 'work_type_children',
       'Residence_type_Rural', 'Residence_type_Urban',
       'smoking_status_Unknown', 'smoking_status_formerly smoked',
       'smoking_status_never smoked', 'smoking_status_smokes',
       'hypertension_0', 'hypertension_1', 'heart_disease_0',
       'heart_disease_1'], dtype=object)
```

Dropped Columns

Removed columns that are unnecessary for the analysis.

In [22]:
```python
data_final.drop(['id'], axis = 1, inplace = True)
```

```
/Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages/pandas/core/frame.py:4308: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

In [23]:
```python
data_final.columns
data_final.head()

data_final = data_final.drop(['gender_Female', 'gender_Other', 'hypertension_1', 'heart_disease_1',
                              'Residence_type_Rural', 'smoking_status_Unknown'],axis=1)
data_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4909 entries, 0 to 5109
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   age                             4909 non-null   int64
 1   avg_glucose_level               4909 non-null   float64
 2   bmi                             4909 non-null   float64
 3   stroke                          4909 non-null   int64
 4   gender_Male                     4909 non-null   uint8
 5   work_type_Govt_job              4909 non-null   uint8
 6   work_type_Never_worked          4909 non-null   uint8
 7   work_type_Private               4909 non-null   uint8
 8   work_type_Self-employed         4909 non-null   uint8
 9   work_type_children              4909 non-null   uint8
 10  Residence_type_Urban            4909 non-null   uint8
 11  smoking_status_formerly smoked  4909 non-null   uint8
 12  smoking_status_never smoked     4909 non-null   uint8
 13  smoking_status_smokes           4909 non-null   uint8
 14  hypertension_0                  4909 non-null   uint8
 15  heart_disease_0                 4909 non-null   uint8
dtypes: float64(2), int64(2), uint8(12)
memory usage: 378.3 KB
```

Creating Training Data and Test Data

Divided the final data into x values (Independent variables) and y values (Depedent variables) for the test.

In [34]:
```python
X = data_final.loc[:, data_final.columns != 'stroke']
y = data_final.loc[:, data_final.columns == 'stroke']
from imblearn.over_sampling import SMOTE
os = SMOTE(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
columns = X_train.columns
os_data_X,os_data_y=os.fit_resample(X_train, y_train)
os_data_X = pd.DataFrame(data=os_data_X,columns=columns )
os_data_y= pd.DataFrame(data=os_data_y,columns=['stroke'])
# we can Check the numbers of our data
print("length of oversampled data is ",len(os_data_X))
print("Number of no stroke in oversampled data",len(os_data_y[os_data_y['stroke']==0]))
print("Number of stroke",len(os_data_y[os_data_y['stroke']==1]))
print("Proportion of no stroke data in oversampled data is ",len(os_data_y[os_data_y['stroke']==0])/len(os_data_X))
print("Proportion of stroke data in oversampled data is ",len(os_data_y[os_data_y['stroke']==1])/len(os_data_X))
```

```
length of oversampled data is  6564
Number of no stroke in oversampled data 3282
Number of stroke 3282
Proportion of no stroke data in oversampled data is  0.5
Proportion of stroke data in oversampled data is  0.5
```

In [35]:
```python
#y_data = data_final['stroke']

#x_data = data_final.drop('stroke', axis = 1)
```

In [36]:
```python
#from sklearn.model_selection import train_test_split
```

In [37]:
```python
#x_training_data, x_test_data, y_training_data, y_test_data = train_test_split(x_data, y_data, test_size = 0.3)
```

In [38]:
```python
from sklearn.linear_model import LogisticRegression
```

Training the Logistic Regression Model

Created a model by instantiating an instance of the LogisticRegression.

Called fit method and passed in x values and y values.

Created a variable called predictions and used model logistic regression model to make predictions.

Used the model to calculate the performance metrics for our logistic regression.

In [39]:
```python
#model = LogisticRegression()
```

In [40]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
X_train, X_test, y_train, y_test = train_test_split(os_data_X, os_data_y, test_size=0.3, random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
/Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/divyabastola/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[40]:
```
▾ LogisticRegression
LogisticRegression()
```

In [41]:
```python
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))
```

```
Accuracy of logistic regression classifier on test set: 0.90
```

In [42]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[889  94]
 [110 877]]
```

In [43]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.90       983
           1       0.90      0.89      0.90       987

    accuracy                           0.90      1970
   macro avg       0.90      0.90      0.90      1970
weighted avg       0.90      0.90      0.90      1970
```

Our model has calculated the accuracy as 90%.

Interpretation: Of the entire test set, 90% of the stroke cases are predicted by the dependent variables.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: