Dataset_Stoke

May 30, 2021

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import matplotlib.ticker as mtick
import seaborn as sns
sns.set_style('white')
import plotly.express as px
import plotly.graph_objs as pgo
import plotly.offline as pyo
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
pyo.init_notebook_mode()
from imblearn.over_sampling import SMOTE
import scikitplot as skplt

import joblib

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,cross_val_score


from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
import os

from sklearn.metrics import␣
↪classification_report,confusion_matrix,accuracy_score, recall_score,␣
↪precision_score,f1_score
import warnings
warnings.filterwarnings('ignore')

plt.rc('figure',figsize=(17,13))
```

```
sns.set_context('paper',font_scale=2)

def set_seed(seed=20210524):
    np.random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'
set_seed()
```

1 Step one: Handle null data

1.1 have a glance at the data

[2]:
```
df_stroke = pd.read_csv("./data/healthcare-dataset-stroke-data.csv")
df_stroke.head()
```

[2]:
```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

       work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0        Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2        Private          Rural             105.92  32.5     never smoked
3        Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1
```

[3]:
```
df_stroke.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
```

1.2 Find null datas

[4]:
```
df_stroke.isnull().sum()
```

[4]:
```
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

1.3 To deal with NaN data, I want to evaluate these bmi data with age and gender, using decision tree

[5]:
```
bmi_pipe = Pipeline( steps = [
    ('scaling',StandardScaler()),
    ('lr',DecisionTreeRegressor(random_state = 30))
] )

cp = df_stroke[['age','gender','bmi']].copy()
cp.gender = cp.gender.replace({'Male':0,'Female':1,'Other':-1}).astype(np.uint8)

miss_bmi = cp[cp.bmi.isnull()]
cp = cp[~cp.bmi.isnull()]
bmi = cp.pop('bmi')
bmi_pipe.fit(cp,bmi)
predict_bmi = pd.Series(bmi_pipe.predict(miss_bmi[['age','gender']]),index =
 ↪miss_bmi.index)
df_stroke.loc[miss_bmi.index,'bmi'] = predict_bmi
```

having a look at the predicted bmi data

[6]:
```
predict_bmi
```

[6]:
```
1       29.879487
8       30.556098
13      27.247222
19      30.841860
27      33.146667
          …
5039    32.716000
5048    28.313636
5093    31.459322
5099    28.313636
5105    28.476923
Length: 201, dtype: float64
```

1.4 so now we have all the datas

[7]:
```
df_stroke.isnull().sum()
```

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

2 Step two: Now, let's find the relation ship among every predictors

2.1 change these string value to districted integers

[8]:
```
df_stroke_int = df_stroke.copy()
df_stroke_int.gender = df_stroke_int.gender.replace({'Male':0,'Female':
↪1,'Other':-1}).astype(np.uint8)
df_stroke_int.smoking_status = df_stroke_int.smoking_status.replace({'formerly␣
↪smoked':5,'never smoked':0,'smokes':3,'Unknown':-1}).astype(np.uint8)
df_stroke_int.work_type = df_stroke_int.work_type.replace({'children':
↪4,'Govt_job':3,'Never_worked':2,'Self-employed':1,'Private':0}).astype(np.
↪uint8)
```

```
df_stroke_int.ever_married = df_stroke_int.ever_married.replace({'Yes':1,'No':
 ↪0}).astype(np.uint8)
df_stroke_int.Residence_type = df_stroke_int.Residence_type.replace({'Rural':
 ↪1,'Urban':0}).astype(np.uint8)
df_stroke_int
```

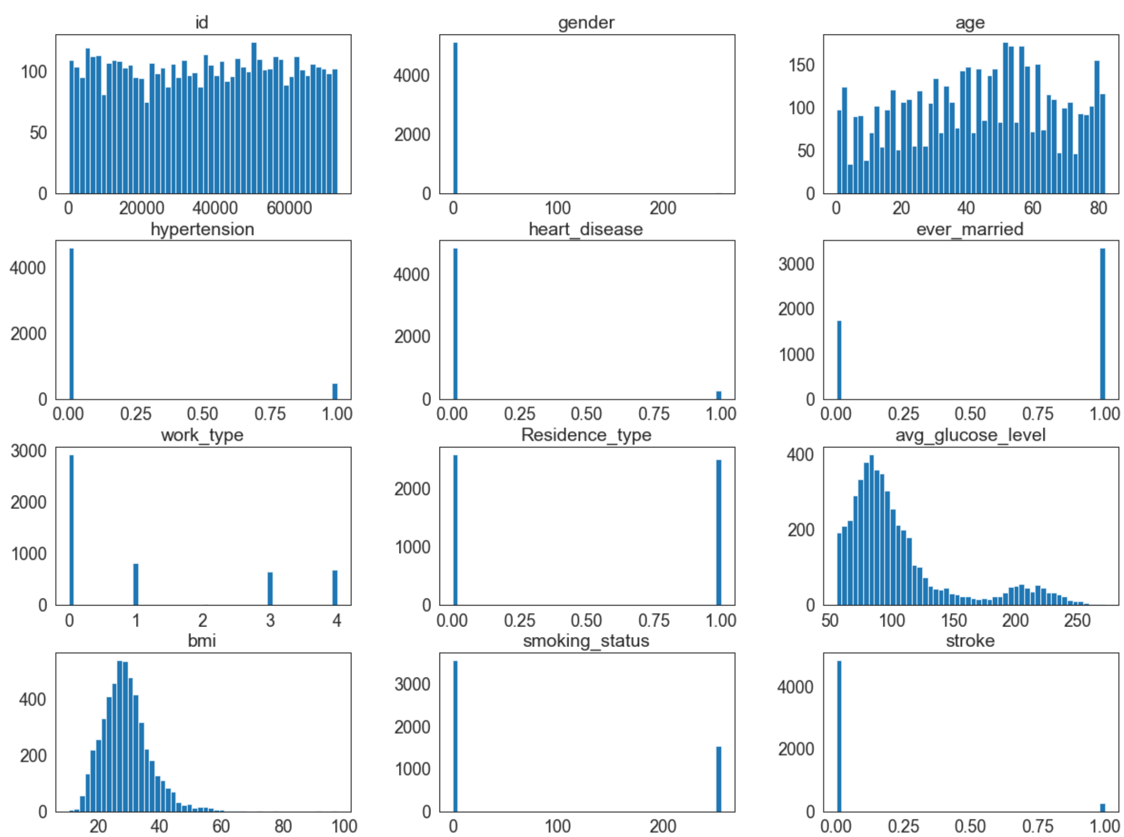
[8]:

```
         id  gender   age  hypertension  heart_disease  ever_married  \
0      9046       0  67.0             0              1             1
1     51676       1  61.0             0              0             1
2     31112       0  80.0             0              1             1
3     60182       1  49.0             0              0             1
4      1665       1  79.0             1              0             1
…       …     …   …           …            …            …
5105  18234       1  80.0             1              0             1
5106  44873       1  81.0             0              0             1
5107  19723       1  35.0             0              0             1
5108  37544       0  51.0             0              0             1
5109  44679       1  44.0             0              0             1

      work_type  Residence_type  avg_glucose_level        bmi  smoking_status  \
0             0               0             228.69  36.600000               5
1             1               1             202.21  29.879487               0
2             0               1             105.92  32.500000               0
3             0               0             171.23  34.400000               3
4             1               1             174.12  24.000000               0
…          …              …                …        …             …
5105          0               0              83.75  28.476923               0
5106          1               0             125.20  40.000000               0
5107          1               1              82.99  30.600000               0
5108          0               1             166.29  25.600000               5
5109          3               0              85.28  26.200000             255

      stroke
0          1
1          1
2          1
3          1
4          1
…        …
5105       0
5106       0
5107       0
5108       0
5109       0

[5110 rows x 12 columns]
```

2.2 plot the distribution of every predictors with histogtram

[9]:
```
df_stroke_int.hist(bins = 50,grid = False, figsize = (20,15))
```

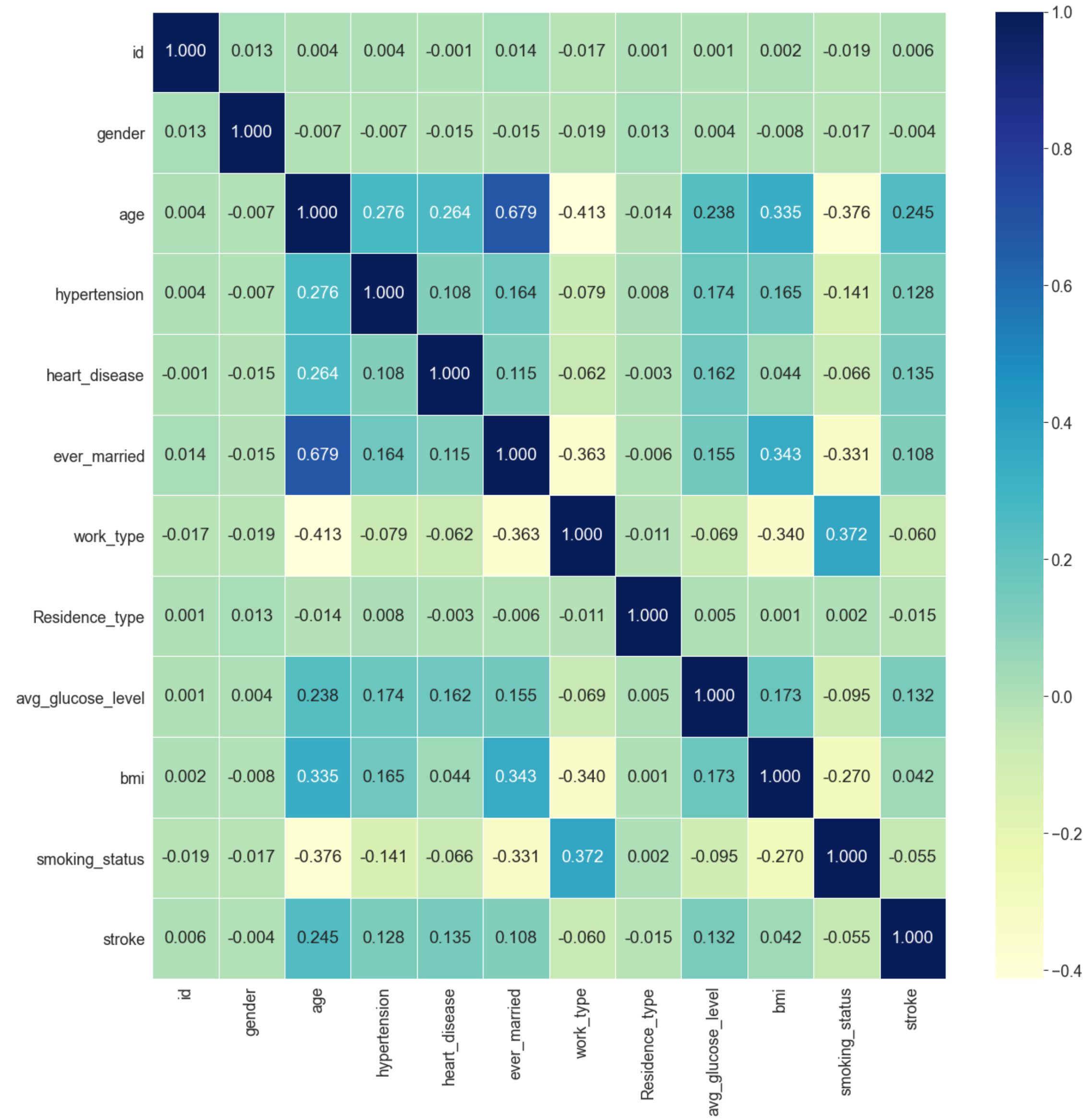
[9]:
```
array([[<AxesSubplot:title={'center':'id'}>,
        <AxesSubplot:title={'center':'gender'}>,
        <AxesSubplot:title={'center':'age'}>],
       [<AxesSubplot:title={'center':'hypertension'}>,
        <AxesSubplot:title={'center':'heart_disease'}>,
        <AxesSubplot:title={'center':'ever_married'}>],
       [<AxesSubplot:title={'center':'work_type'}>,
        <AxesSubplot:title={'center':'Residence_type'}>,
        <AxesSubplot:title={'center':'avg_glucose_level'}>],
       [<AxesSubplot:title={'center':'bmi'}>,
        <AxesSubplot:title={'center':'smoking_status'}>,
        <AxesSubplot:title={'center':'stroke'}>]], dtype=object)
```



2.3 plot the heat map

[10]:
```
f, ax = plt.subplots(figsize = (20,20))
sns.heatmap(df_stroke_int.corr(), annot=True, fmt=".3f", linewidths=0.8,
 ↪ax=ax,cmap="YlGnBu")
```

[10]: <AxesSubplot:>



According to the upper picture, we found that the propability of stroke has little related to work_type, Residence_type and smoking_status.

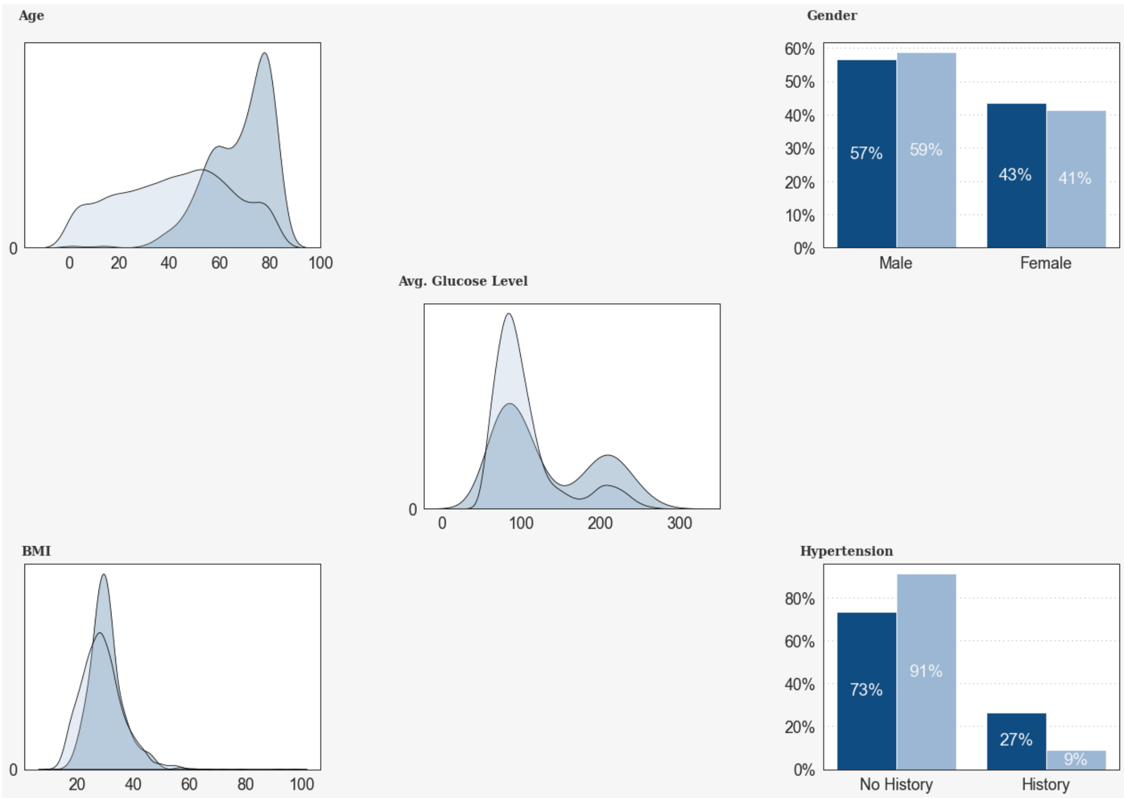
Let’s go further.

[11]:
```
df_isstroke = df_stroke[df_stroke['stroke'] == 1]
df_notstroke = df_stroke[df_stroke['stroke'] != 1]
```

[12]:
```
fig = plt.figure(figsize=(22,15))
gs = fig.add_gridspec(3, 3)
gs.update(wspace=0.35, hspace=0.27)
ax0 = fig.add_subplot(gs[0, 0])
ax1 = fig.add_subplot(gs[0, 2])
ax2 = fig.add_subplot(gs[1, 1])
ax3 = fig.add_subplot(gs[2, 0])
ax4 = fig.add_subplot(gs[2, 2])
background_color = "#f6f6f6"
fig.patch.set_facecolor(background_color) # figure background color

#age
ax0.grid(color='gray', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
positive = pd.DataFrame(df_isstroke["age"])
negative = pd.DataFrame(df_notstroke["age"])
sns.kdeplot(positive["age"], ax=ax0,color="#0f4c81", shade=True,␣
 ↪ec='black',label="positive")
sns.kdeplot(negative["age"], ax=ax0, color="#9bb7d4", shade=True,␣
 ↪ec='black',label="negative")
ax0.yaxis.set_major_locator(mtick.MultipleLocator(2))
ax0.set_ylabel('')
ax0.set_xlabel('')
ax0.text(-20, 0.0465, 'Age', fontsize=14, fontweight='bold',␣
 ↪fontfamily='serif', color="#323232")
#gender
positive = pd.DataFrame(df_isstroke["gender"].value_counts())
positive["Percentage"] = positive["gender"].apply(lambda x: x/
 ↪sum(positive["gender"])*100)
negative = pd.DataFrame(df_notstroke["gender"].value_counts())
negative["Percentage"] = negative["gender"].apply(lambda x: x/
 ↪sum(negative["gender"])*100)

x = np.arange(len(positive))
ax1.text(-0.4, 68.5, 'Gender', fontsize=14, fontweight='bold',␣
 ↪fontfamily='serif', color="#323232")
ax1.grid(color='gray', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
ax1.bar(x, height=positive["Percentage"], zorder=3, color="#0f4c81", width=0.4)

####problem

ax1.bar(x+0.4, height=negative.loc[['Female','Male'],]["Percentage"], zorder=3,␣
 ↪color="#9bb7d4", width=0.4)
ax1.set_xticks(x + 0.4 / 2)
ax1.set_xticklabels(['Male','Female'])
```

```python
ax1.yaxis.set_major_formatter(mtick.PercentFormatter())
ax1.yaxis.set_major_locator(mtick.MultipleLocator(10))
for i,j in zip([0, 1], positive["Percentage"]):
    ax1.annotate(f'{j:0.0f}%',xy=(i, j/2), color='#f6f6f6', horizontalalignment='center', verticalalignment='center')
for i,j in zip([0, 1], negative["Percentage"]):
    ax1.annotate(f'{j:0.0f}%',xy=(i+0.4, j/2), color='#f6f6f6', horizontalalignment='center', verticalalignment='center')


#Avg. Glucose level
ax2.grid(color='gray', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
positive = pd.DataFrame(df_isstroke["avg_glucose_level"])
negative = pd.DataFrame(df_notstroke["avg_glucose_level"])
sns.kdeplot(positive["avg_glucose_level"], ax=ax2,color="#0f4c81",ec='black', shade=True, label="positive")
sns.kdeplot(negative["avg_glucose_level"], ax=ax2, color="#9bb7d4", ec='black',shade=True, label="negative")
ax2.text(-55, 0.01855, 'Avg. Glucose Level',
         fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")
ax2.yaxis.set_major_locator(mtick.MultipleLocator(2))
ax2.set_ylabel('')
ax2.set_xlabel('')

#BMI
ax3.grid(color='gray', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
positive = pd.DataFrame(df_isstroke["bmi"])
negative = pd.DataFrame(df_notstroke["bmi"])
sns.kdeplot(positive["bmi"], ax=ax3,color="#0f4c81", ec='black',shade=True, label="positive")
sns.kdeplot(negative["bmi"], ax=ax3, color="#9bb7d4",ec='black', shade=True, label="negative")
ax3.text(-0.06, 0.09, 'BMI',
         fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")
ax3.yaxis.set_major_locator(mtick.MultipleLocator(2))
ax3.set_ylabel('')
ax3.set_xlabel('')

#Hypertension
positive = pd.DataFrame(df_isstroke["hypertension"].value_counts())
positive["Percentage"] = positive["hypertension"].apply(lambda x: x/sum(positive["hypertension"])*100)
negative = pd.DataFrame(df_notstroke["hypertension"].value_counts())
negative["Percentage"] = negative["hypertension"].apply(lambda x: x/sum(negative["hypertension"])*100)
```

```
x = np.arange(len(positive))
ax4.text(-0.45, 100, 'Hypertension', fontsize=14, fontweight='bold',␣
↪fontfamily='serif', color="#323232")
ax4.grid(color='gray', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
ax4.bar(x, height=positive["Percentage"], zorder=3, color="#0f4c81", width=0.4)
ax4.bar(x+0.4, height=negative["Percentage"], zorder=3, color="#9bb7d4",␣
↪width=0.4)
ax4.set_xticks(x + 0.4 / 2)
ax4.set_xticklabels(['No History','History'])
ax4.yaxis.set_major_formatter(mtick.PercentFormatter())
ax4.yaxis.set_major_locator(mtick.MultipleLocator(20))
for i,j in zip([0, 1], positive["Percentage"]):
    ax4.annotate(f'{j:0.0f}%',xy=(i, j/2), color='#f6f6f6',␣
↪horizontalalignment='center', verticalalignment='center')
for i,j in zip([0, 1], negative["Percentage"]):
    ax4.annotate(f'{j:0.0f}%',xy=(i+0.4, j/2), color='#f6f6f6',␣
↪horizontalalignment='center', verticalalignment='center')
```



So we build model with upper predictors

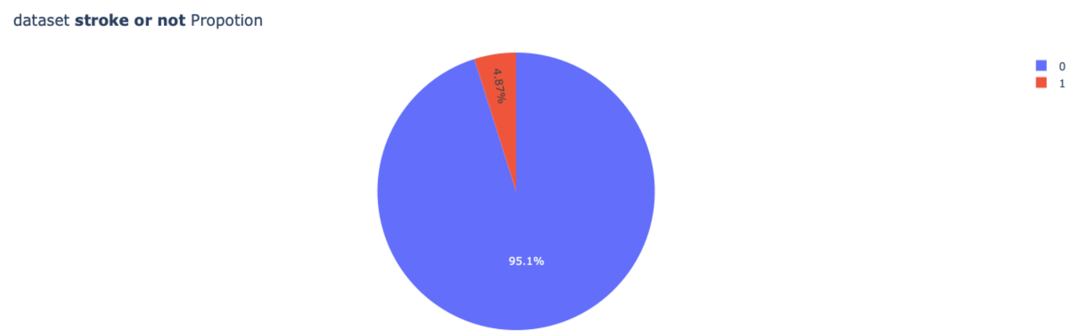
3 Step three: Before building models, we affirm the training and testing datas first.

3.1 We first look at the number of the data who has stroke and who has't

[13]:
```
nbisstroke = len(df_isstroke)
nbnotstroke = len(df_notstroke)
print('number of people who has stroked',nbisstroke)
print("number of people who hasn't stroked",nbnotstroke)
```

```
number of people who has stroked 249
number of people who hasn't stroked 4861
```

[14]:
```
fig = px.pie(df_stroke,names='stroke')
fig.update_layout(title='dataset <b>stroke or not</b> Propotion')
fig.show()
```



We find that there are too many data of who hasn't attacked by stroke, so we should balance these numbers of datas.

We now solve over-sambling problem.

3.2 We use SMOTE(Synthetic Minority Over-sampling Technique) to generate more datas.

The traditional over-sampling methode, just duplicating these minority sample and making no new infomation to the dataset, which will easily causes over-fitting problem. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Aliking the equation: $x\_new = x + rand(0,1) * (\hat{x} - x)$

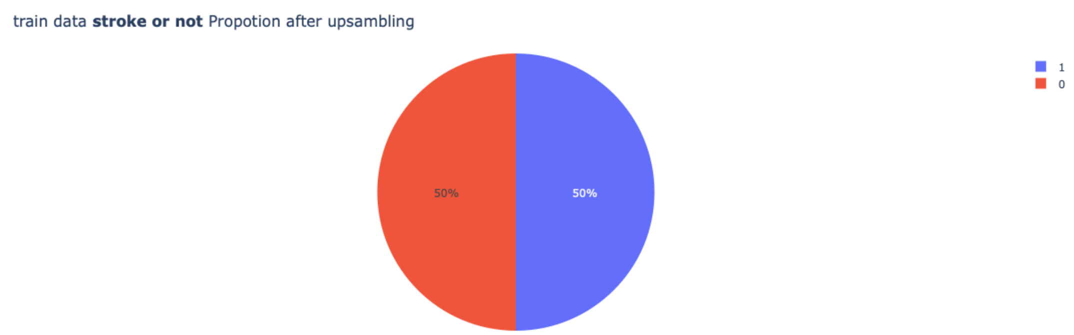
[15]:
```
oversample = SMOTE()
df_precise =
 ↪df_stroke[['gender','age','hypertension','avg_glucose_level','bmi','stroke']]
```

```
df_precise.gender = df_precise.gender.replace({'Male':0,'Female':1,'Other':-1}).
 ↪astype(np.uint8)
df_test = df_precise.sample(int(len(df_stroke)*0.2),random_state = 30)
df_train = df_precise.drop(index = df_test.index)

x_test, y_test =␣
 ↪df_test[['gender','age','hypertension','avg_glucose_level','bmi']],␣
 ↪df_test['stroke']
x_train, y_train =␣
 ↪df_train[['gender','age','hypertension','avg_glucose_level','bmi']],␣
 ↪df_train['stroke']

x_test, y_test = oversample.fit_resample(x_test,y_test)
adjust_test = x_test.assign(stroke = y_test)
x_train, y_train = oversample.fit_resample(x_train,y_train)
adjust_train = x_train.assign(stroke = y_train)
```

[16]:
```
print(len(adjust_test[adjust_test.stroke == 1]),len(adjust_test[adjust_test.
 ↪stroke == 0]))
print(len(adjust_train[adjust_train.stroke == 1]),len(adjust_train[adjust_train.
 ↪stroke == 0]))
```

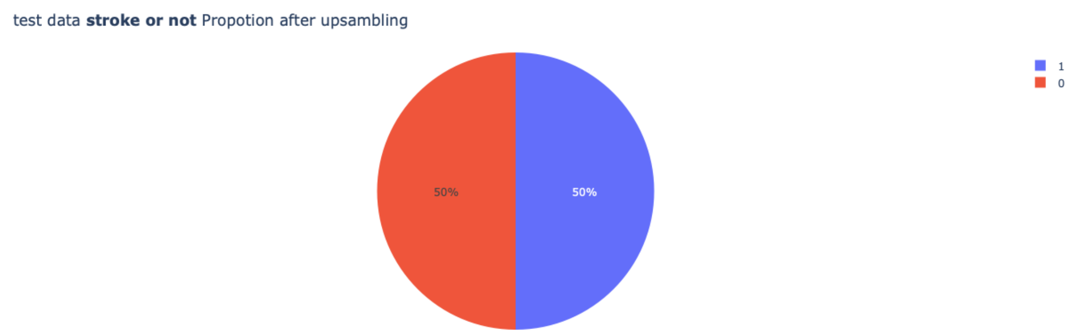
```
986 986
3875 3875
```

[17]:
```
fig = px.pie(adjust_train,names='stroke')
fig.update_layout(title='train data <b>stroke or not</b> Propotion after␣
 ↪upsambling')
fig.show()
```



[18]:
```
fig = px.pie(adjust_test,names='stroke')
fig.update_layout(title='test data <b>stroke or not</b> Propotion after␣
 ↪upsambling')
```

```
fig.show()
```



4 Step four: Build models and analyse the result

I will try following models:

Random Forest

Logistic Regression

Decision Tree

MLP Neural Network with L-BFGS

MLP Neural Network with SGD

Support Vector Machine

[19]:
```
# model
RF_pipeline = Pipeline(steps = [
    ('scale',StandardScaler()),
    ('RF',RandomForestClassifier(random_state = 30))
])
LogR_pipeline = Pipeline(steps = [
    ('scale',StandardScaler()),
    ('LogR',LogisticRegression(random_state = 30))
])
DT_pipeline = Pipeline(steps = [
    ('scale',StandardScaler()),
    ('DT',DecisionTreeClassifier(random_state = 30))
])
MLP_lbfgs_pipeline = Pipeline(steps = [
    ('scale',StandardScaler()),
    ('MLP',MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5,
↪2),random_state = 30))
```

```
])
MLP_SGD_pipeline = Pipeline(steps = [
    ('scale',StandardScaler()),
    ('MLP',MLPClassifier(hidden_layer_sizes=(5,2), random_state=30,␣
 ↪max_iter=40, warm_start=True))
])
SVM_pipeline = Pipeline(steps = [
    ('scale',StandardScaler()),
    ('SVM',SVC(random_state = 30))
])
```

[20]:
```
# cross validation
RF_CV = cross_val_score(RF_pipeline, x_train, y_train, cv = 10, scoring = 'f1')
LogR_CV = cross_val_score(LogR_pipeline, x_train, y_train, cv = 10, scoring =␣
 ↪'f1')
DT_CV = cross_val_score(DT_pipeline, x_train, y_train, cv = 10, scoring = 'f1')
MLP_lbfgs_CV = cross_val_score(MLP_lbfgs_pipeline, x_train, y_train, cv = 10,␣
 ↪scoring = 'f1')
MLP_SGD_CV = cross_val_score(MLP_SGD_pipeline, x_train, y_train, cv = 10,␣
 ↪scoring = 'f1')
SVM_CV = cross_val_score(SVM_pipeline, x_train, y_train, cv = 10, scoring =␣
 ↪'f1')
```

[21]:
```
print('The correction rate of cross validation of Random Forest: ', RF_CV.
 ↪mean())
print('The correction rate of cross validation of Logistic Regression: ',␣
 ↪LogR_CV.mean())
print('The correction rate of cross validation of Decision Tree: ', DT_CV.
 ↪mean())
print('The correction rate of cross validation of MuliLayer Perception with␣
 ↪L-BFGS algorithm: ', MLP_lbfgs_CV.mean())
print('The correction rate of cross validation of MuliLayer Perception with␣
 ↪Stochastic Gradient Descent algorithm: ', MLP_SGD_CV.mean())
print('The correction rate of cross validation of Support Vector Machine : ',␣
 ↪SVM_CV.mean())
```

```
The correction rate of cross validation of Random Forest:  0.9254277042396863
The correction rate of cross validation of Logistic Regression:
0.8125404112385366
The correction rate of cross validation of Decision Tree:  0.9000791816875742
The correction rate of cross validation of MuliLayer Perception with L-BFGS
algorithm:  0.8193992873591979
The correction rate of cross validation of MuliLayer Perception with Stochastic
Gradient Descent algorithm:  0.81339693090295
The correction rate of cross validation of Support Vector Machine :
0.8224945320864562
```

From what we have seen above, we can conclusion easly that the model Random Forest preform best.

But one who is illed wouldn't like to get a wrong answer, to make it more reasonable, we next calculate the Recall Rate

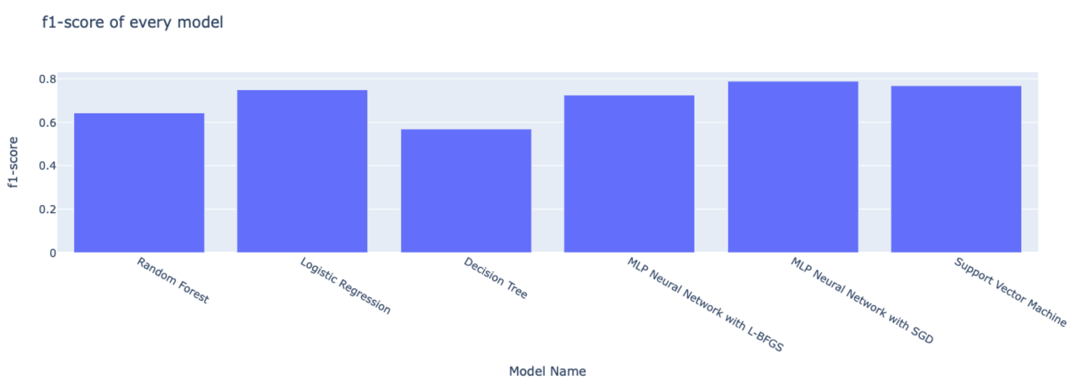
[22]:
```
RF_pipeline.fit(x_train,y_train)
LogR_pipeline.fit(x_train,y_train)
DT_pipeline.fit(x_train,y_train)
MLP_lbfgs_pipeline.fit(x_train,y_train)
MLP_SGD_pipeline.fit(x_train,y_train)
SVM_pipeline.fit(x_train,y_train)
```

[22]:
```
Pipeline(steps=[('scale', StandardScaler()), ('SVM', SVC(random_state=30))])
```

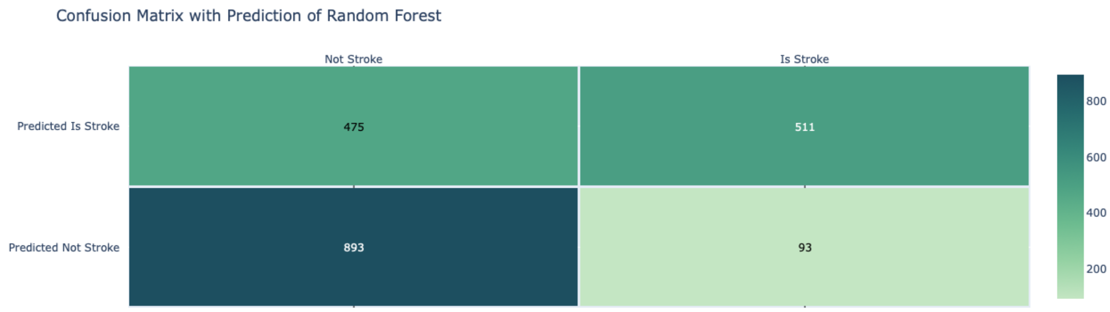
[23]:
```
# predict and calculate confusion-matrix and f1-score
RF_pred = RF_pipeline.predict(x_test)
LogR_pred = LogR_pipeline.predict(x_test)
DT_pred = DT_pipeline.predict(x_test)
MLP_lbfgs_pred = MLP_lbfgs_pipeline.predict(x_test)
MLP_SGD_pred = MLP_SGD_pipeline.predict(x_test)
SVM_pred = SVM_pipeline.predict(x_test)

RF_conf = confusion_matrix(y_test, RF_pred)
LogR_conf = confusion_matrix(y_test, LogR_pred)
DT_conf = confusion_matrix(y_test, DT_pred)
MLP_lbfgs_conf = confusion_matrix(y_test, MLP_lbfgs_pred)
MLP_SGD_conf = confusion_matrix(y_test, MLP_SGD_pred)
SVM_conf = confusion_matrix(y_test, SVM_pred)

RF_f1 = f1_score(y_test, RF_pred)
LogR_f1 = f1_score(y_test, LogR_pred)
DT_f1 = f1_score(y_test, DT_pred)
MLP_lbfgs_f1 = f1_score(y_test, MLP_lbfgs_pred)
MLP_SGD_f1 = f1_score(y_test, MLP_SGD_pred)
SVM_f1 = f1_score(y_test, SVM_pred)

RF_acc = accuracy_score(y_test, RF_pred)
LogR_acc = accuracy_score(y_test, LogR_pred)
DT_acc = accuracy_score(y_test, DT_pred)
MLP_lbfgs_acc = accuracy_score(y_test, MLP_lbfgs_pred)
MLP_SGD_acc = accuracy_score(y_test, MLP_SGD_pred)
SVM_acc = accuracy_score(y_test, SVM_pred)

RF_rec = recall_score(y_test, RF_pred)
LogR_rec = recall_score(y_test, LogR_pred)
DT_rec = recall_score(y_test, DT_pred)
MLP_lbfgs_rec = recall_score(y_test, MLP_lbfgs_pred)
MLP_SGD_rec = recall_score(y_test, MLP_SGD_pred)
```

```
SVM_rec = recall_score(y_test, SVM_pred)

RF_pre = precision_score(y_test, RF_pred)
LogR_pre = precision_score(y_test, LogR_pred)
DT_pre = precision_score(y_test, DT_pred)
MLP_lbfgs_pre = precision_score(y_test, MLP_lbfgs_pred)
MLP_SGD_pre = precision_score(y_test, MLP_SGD_pred)
SVM_pre = precision_score(y_test, SVM_pred)
```

[24]:
```
# f1-score
fig = pgo.Figure()
fig.add_trace(pgo.Bar(x = ['Random Forest','Logistic Regression','Decision␣
↪Tree','MLP Neural Network with L-BFGS','MLP Neural Network with␣
↪SGD','Support Vector Machine'],y =␣
↪[RF_f1,LogR_f1,DT_f1,MLP_lbfgs_f1,MLP_SGD_f1,SVM_f1]))
fig.update_layout(title = 'f1-score of every model', xaxis_title = 'Model␣
↪Name', yaxis_title = 'f1-score')
```



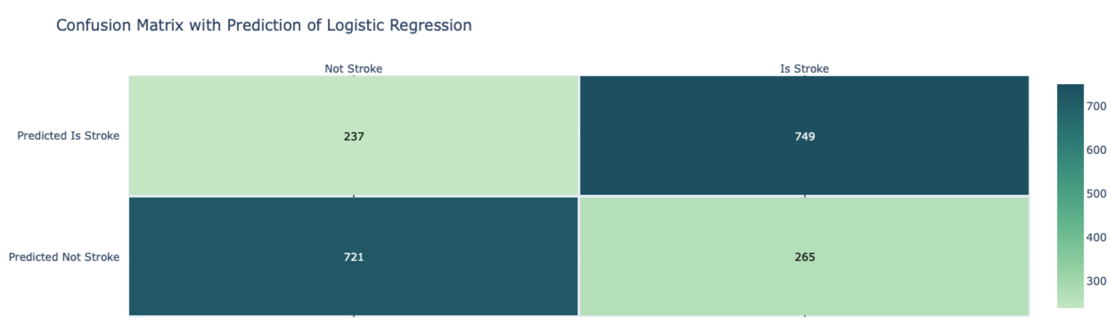
[25]:
```
# Random Forest
fig = ff.create_annotated_heatmap(RF_conf, x=['Not Stroke','Is Stroke'],␣
↪y=['Predicted Not Stroke','Predicted Is Stroke'],␣
↪colorscale='blugrn',xgap=3,ygap=3)
fig['data'][0]['showscale'] = True
fig.update_layout(title='Confusion Matrix with Prediction of Random Forest')
fig.show()
print(classification_report(y_test,RF_pred))
print('Accuracy Score: ', RF_acc)
```




```
              precision    recall  f1-score   support

           0       0.65      0.91      0.76       986
           1       0.85      0.52      0.64       986

    accuracy                           0.71      1972
   macro avg       0.75      0.71      0.70      1972
weighted avg       0.75      0.71      0.70      1972

Accuracy Score:  0.7119675456389453
```

[26]:

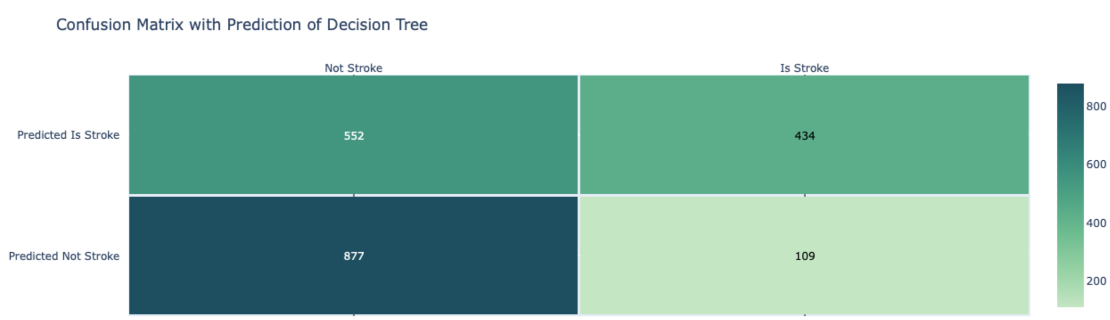
```
# Logistic Regression
fig = ff.create_annotated_heatmap(LogR_conf, x=['Not Stroke','Is Stroke'],␣
 ↪y=['Predicted Not Stroke','Predicted Is Stroke'],␣
 ↪colorscale='blugrn',xgap=3,ygap=3)
fig['data'][0]['showscale'] = True
fig.update_layout(title='Confusion Matrix with Prediction of Logistic␣
 ↪Regression')
fig.show()
print(classification_report(y_test,LogR_pred))
print('Accuracy Score: ', LogR_acc)
```



```
              precision    recall  f1-score   support

           0       0.75      0.73      0.74       986
           1       0.74      0.76      0.75       986

    accuracy                           0.75      1972
   macro avg       0.75      0.75      0.75      1972
weighted avg       0.75      0.75      0.75      1972

Accuracy Score:  0.7454361054766734
```

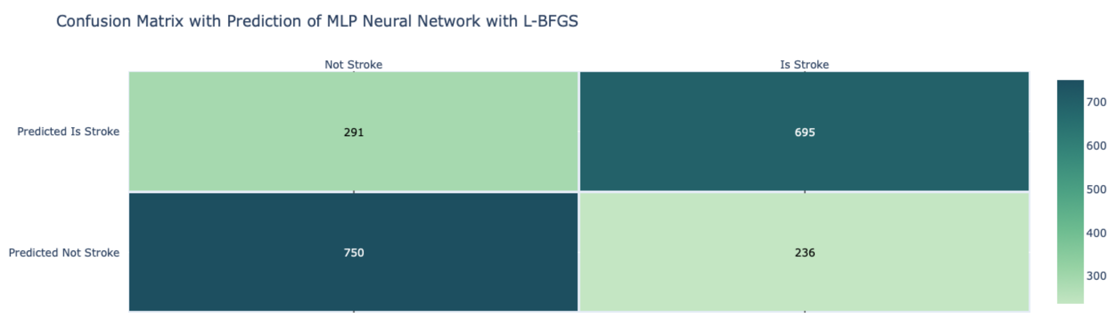
[27]:

```python
# Decision Tree
fig = ff.create_annotated_heatmap(DT_conf, x=['Not Stroke','Is Stroke'],␣
 ↪y=['Predicted Not Stroke','Predicted Is Stroke'],␣
 ↪colorscale='blugrn',xgap=3,ygap=3)
fig['data'][0]['showscale'] = True
fig.update_layout(title='Confusion Matrix with Prediction of Decision Tree')
fig.show()
print(classification_report(y_test,DT_pred))
print('Accuracy Score: ', DT_acc)
```



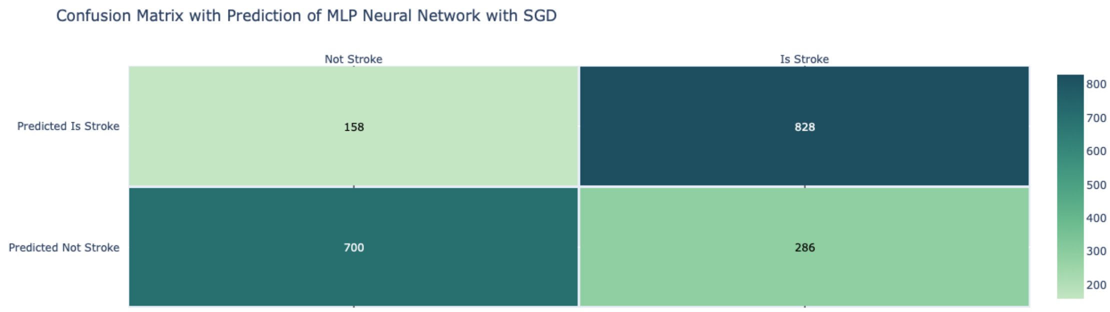

```
              precision    recall  f1-score   support

           0       0.61      0.89      0.73       986
           1       0.80      0.44      0.57       986

    accuracy                           0.66      1972
   macro avg       0.71      0.66      0.65      1972
weighted avg       0.71      0.66      0.65      1972

Accuracy Score:  0.6648073022312373
```

[28]:
```python
# MLP Neural Network with L-BFGS
fig = ff.create_annotated_heatmap(MLP_lbfgs_conf, x=['Not Stroke','Is Stroke'],␣
 ↪y=['Predicted Not Stroke','Predicted Is Stroke'],␣
 ↪colorscale='blugrn',xgap=3,ygap=3)
fig['data'][0]['showscale'] = True
fig.update_layout(title='Confusion Matrix with Prediction of MLP Neural Network␣
 ↪with L-BFGS')
fig.show()
print(classification_report(y_test,MLP_lbfgs_pred))
print('Accuracy Score: ', MLP_lbfgs_acc)
```




```
              precision    recall  f1-score   support

           0       0.72      0.76      0.74       986
           1       0.75      0.70      0.73       986

    accuracy                           0.73      1972
   macro avg       0.73      0.73      0.73      1972
weighted avg       0.73      0.73      0.73      1972

Accuracy Score:  0.7327586206896551
```

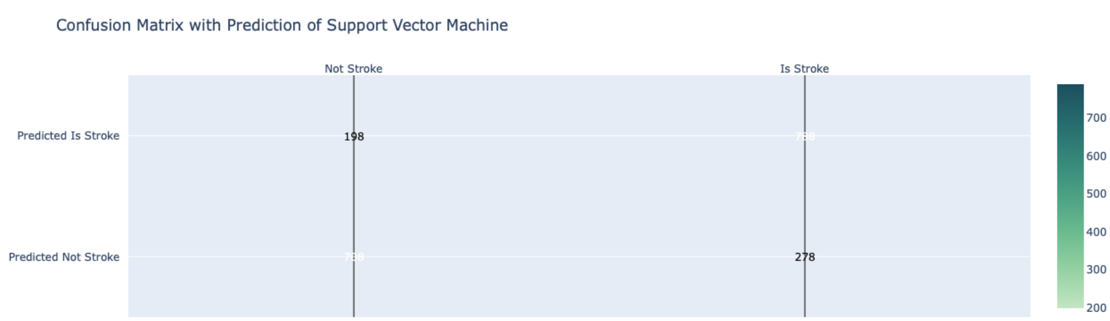
[29]:
```python
# MLP Neural Network with SGD
fig = ff.create_annotated_heatmap(MLP_SGD_conf, x=['Not Stroke','Is Stroke'],␣
 ↪y=['Predicted Not Stroke','Predicted Is Stroke'],␣
 ↪colorscale='blugrn',xgap=3,ygap=3)
fig['data'][0]['showscale'] = True
fig.update_layout(title='Confusion Matrix with Prediction of MLP Neural Network␣
 ↪with SGD')
fig.show()
print(classification_report(y_test,MLP_SGD_pred))
print('Accuracy Score: ', MLP_SGD_acc)
```




```
              precision    recall  f1-score   support

           0       0.82      0.71      0.76       986
           1       0.74      0.84      0.79       986

    accuracy                           0.77      1972
   macro avg       0.78      0.77      0.77      1972
weighted avg       0.78      0.77      0.77      1972

Accuracy Score:  0.7748478701825557
```

[30]:

```python
# Support Vector Machine
fig = ff.create_annotated_heatmap(SVM_conf, x=['Not Stroke','Is Stroke'],␣
 ↪y=['Predicted Not Stroke','Predicted Is Stroke'],␣
 ↪colorscale='blugrn',xgap=3,ygap=3)
fig['data'][0]['showscale'] = True
fig.update_layout(title='Confusion Matrix with Prediction of Support Vector␣
 ↪Machine')
fig.show()
print(classification_report(y_test,SVM_pred))
print('Accuracy Score: ', SVM_acc)
```



```
              precision    recall  f1-score   support

           0       0.78      0.72      0.75       986
           1       0.74      0.80      0.77       986

    accuracy                           0.76      1972
   macro avg       0.76      0.76      0.76      1972
weighted avg       0.76      0.76      0.76      1972

Accuracy Score:  0.7586206896551724
```

5 Step five: Model Selection

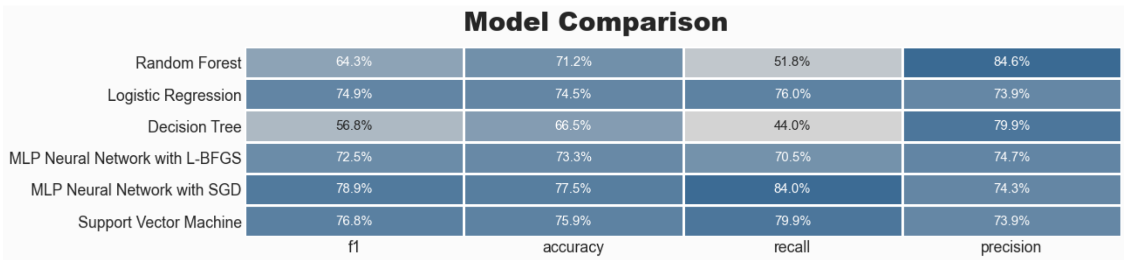
[31]:

```
# make dataframes to plot
RF_df = pd.DataFrame(data = [RF_f1,RF_acc,RF_rec,RF_pre], columns = ['Random␣
↪Forest'], index = ['f1','accuracy','recall','precision'])
LogR_df = pd.DataFrame(data = [LogR_f1,LogR_acc,LogR_rec,LogR_pre], columns =␣
↪['Logistic Regression'], index = ['f1','accuracy','recall','precision'])
DT_df = pd.DataFrame(data = [DT_f1,DT_acc,DT_rec,DT_pre], columns = ['Decision␣
↪Tree'], index = ['f1','accuracy','recall','precision'])
MLP_lbfgs_df = pd.DataFrame(data =␣
↪[MLP_lbfgs_f1,MLP_lbfgs_acc,MLP_lbfgs_rec,MLP_lbfgs_pre], columns = ['MLP␣
↪Neural Network with L-BFGS'], index = ['f1','accuracy','recall','precision'])
MLP_SGD_df = pd.DataFrame(data =␣
↪[MLP_SGD_f1,MLP_SGD_acc,MLP_SGD_rec,MLP_SGD_pre], columns = ['MLP Neural␣
↪Network with SGD'], index = ['f1','accuracy','recall','precision'])
SVM_df = pd.DataFrame(data = [SVM_f1,SVM_acc,SVM_rec,SVM_pre], columns =␣
↪['Support Vector Machine'], index = ['f1','accuracy','recall','precision'])
```

[32]:

```
df_analysis = round(pd.
↪concat([RF_df,LogR_df,DT_df,MLP_lbfgs_df,MLP_SGD_df,SVM_df], axis=1) , 6)
colors = ["lightgray","lightgray","#0f4c81"]
colormap = matplotlib.colors.LinearSegmentedColormap.from_list("", colors)
background_color = "#fbfbfb"
fig = plt.figure(figsize=(18,16))
gs = fig.add_gridspec(3, 5)
gs.update(wspace=0.1, hspace=0.5)
ax0 = fig.add_subplot(gs[0, :])
sns.heatmap(df_analysis.T, cmap=colormap,annot=True,fmt=".1%",vmin=0,vmax=0.95,␣
↪linewidths=2.5,cbar=False,ax=ax0,annot_kws={"fontsize":15})
fig.patch.set_facecolor(background_color) # figure background color
ax0.set_facecolor(background_color)
ax0.text(1,-0.5,'Model␣
↪Comparison',fontsize=30,fontweight='bold',fontfamily='Arial Black')
```

```
ax0.tick_params(axis=u'both', which=u'both',length=0)

plt.show()
```



5.0.1 So obviously, we find that the model of MLP Neural Network with SGD algorithm has the highest recall rate of class_1 (is_stroke) which means that it will misclassify least cases of patient who has stoke.

6 Step six: Find the best parameters

6.1 Find the iterater times of largest recall rate

[34]:
```
def MLP_SGD_opt():
    MLP_SGD_rec_max = -1
    MLP_SGD_rec_max_index = -1
    for times in np.arange(1,50):
        MLP_SGD_pipeline = Pipeline(steps = [
            ('scale',StandardScaler()),
            ('MLP',MLPClassifier(hidden_layer_sizes=(5,2), random_state=30,␣
 ↪max_iter=times, warm_start=True))
        ])
        MLP_SGD_pipeline.fit(x_train,y_train)
        MLP_SGD_pred = MLP_SGD_pipeline.predict(x_test)
        MLP_SGD_rec = recall_score(y_test, MLP_SGD_pred)
#         print(MLP_SGD_rec,' , ',times)
        if(MLP_SGD_rec_max <= MLP_SGD_rec):
            MLP_SGD_rec_max = MLP_SGD_rec
            MLP_SGD_rec_max_index = times

        else:
            break
    return [MLP_SGD_rec_max,MLP_SGD_rec_max_index]
[m,i] = MLP_SGD_opt()
print("recall rate: ",m,"index: ",i)
```

```
recall rate:  0.8539553752535497 index:  24
```

6.2 Replot the accuracy rate of this parameter

[35]:
```
MLP_SGD_pipeline = Pipeline(steps = [
    ('scale',StandardScaler()),
    ('MLP',MLPClassifier(hidden_layer_sizes=(5,2), random_state=30,␣
 ↪max_iter=23, warm_start=True)) #0.981
])

MLP_SGD_pipeline.fit(x_train,y_train)
MLP_SGD_pred = MLP_SGD_pipeline.predict(x_test)
MLP_SGD_conf = confusion_matrix(y_test, MLP_SGD_pred)
MLP_SGD_f1 = f1_score(y_test, MLP_SGD_pred)
MLP_SGD_acc = accuracy_score(y_test, MLP_SGD_pred)
MLP_SGD_rec = recall_score(y_test, MLP_SGD_pred)
MLP_SGD_pre = precision_score(y_test, MLP_SGD_pred)

print("recall rate:",MLP_SGD_rec)

# MLP Neural Network with SGD
fig = ff.create_annotated_heatmap(MLP_SGD_conf, x=['Not Stroke','Is Stroke'],␣
 ↪y=['Predicted Not Stroke','Predicted Is Stroke'],␣
 ↪colorscale='blugrn',xgap=3,ygap=3)
fig['data'][0]['showscale'] = True
fig.update_layout(title='Confusion Matrix with Prediction of MLP Neural Network␣
 ↪with SGD')
fig.show()
print(classification_report(y_test,MLP_SGD_pred))
print('Accuracy Score: ', MLP_SGD_acc)
```

```
recall rate: 0.8529411764705882
```


```
              precision    recall  f1-score   support

           0       0.83      0.71      0.76       986
```

```
           1       0.74      0.85      0.79       986

    accuracy                           0.78      1972
   macro avg       0.79      0.78      0.78      1972
weighted avg       0.79      0.78      0.78      1972

Accuracy Score:  0.7794117647058824
```

6.3 Conclusion

Due to the randomness of the method stochastic gradient descend and we can do early stopping when we training the data with MultiLayer Network, we can control when to stop by the will of my own. So in this case, I can gain a higher recall rate by sacrificing the accuracy rate. But if we want the best performance of accuracy, (balanced) random forest is still the best choice.

7 Last step: Train the model with all data

[36]:

```python
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from imblearn.over_sampling import SMOTE
from sklearn.neural_network import MLPClassifier
import warnings
warnings.filterwarnings('ignore')
```

[37]:

```python
df_stroke_all = pd.read_csv("./data/healthcare-dataset-stroke-data.csv")
bmi_pipe = Pipeline( steps = [
    ('scaling',StandardScaler()),
    ('lr',DecisionTreeRegressor(random_state = 30))
])
cp = df_stroke_all[['age','gender','bmi']].copy()
cp.gender = cp.gender.replace({'Male':0,'Female':1,'Other':-1}).astype(np.uint8)
miss_bmi = cp[cp.bmi.isnull()]
cp = cp[~cp.bmi.isnull()]
bmi = cp.pop('bmi')
bmi_pipe.fit(cp,bmi)
predict_bmi = pd.Series(bmi_pipe.predict(miss_bmi[['age','gender']]),index =␣
 ↪miss_bmi.index)
df_stroke_all.loc[miss_bmi.index,'bmi'] = predict_bmi

oversample = SMOTE()
df_precise_all =␣
 ↪df_stroke_all[['gender','age','hypertension','avg_glucose_level','bmi','stroke']]
df_precise_all.gender = df_precise_all.gender.replace({'Male':0,'Female':
 ↪1,'Other':-1}).astype(np.uint8)
```

```
x_all, y_all =␣
↪df_precise_all[['gender','age','hypertension','avg_glucose_level','bmi']],␣
↪df_precise_all['stroke']
x_all, y_all = oversample.fit_resample(x_all,y_all)
adjust_all = x_all.assign(stroke = y_all)
MLP_SGD_pipeline = Pipeline(steps = [
    ('scale',StandardScaler()),
    ('MLP',MLPClassifier(hidden_layer_sizes=(5,2), random_state=30,␣
↪max_iter=40, warm_start=True))
])
MLP_SGD_pipeline.fit(x_all,y_all)
```

[37]:
```
Pipeline(steps=[('scale', StandardScaler()),
                ('MLP',
                 MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=40,
                               random_state=30, warm_start=True))])
```

7.1 Test with several data

[38]:
```
MLP_SGD_pipeline.predict(pd.DataFrame([[0,67,0,228,36.6]]))
```

[38]:
```
array([1])
```

[39]:
```
MLP_SGD_pipeline.predict(pd.DataFrame([[1,79,1,83.75,28.4]]))
```

[39]:
```
array([1])
```

7.2 Store the optimal model

[40]:
```
joblib.dump(MLP_SGD_pipeline,'./data/model_MLP_SGD_fitted.pkl')
```

[40]:
```
['./data/model_MLP_SGD_fitted.pkl']
```

[41]:
```
# load the model

# model = joblib.load("./data/model_MLP_SGD_fitted.pkl")

# model.predict(pd.DataFrame([[1,79,1,83.75,28.4]]))
```

[ ]:
```
```