STROKE PREDICTION SYSTEM IN BUSINESS

Predicting a Stroke :-

In [ ]:
```
                                    ML (PROJECT)

# Submitted To - DEBENDRA K DHIR                                          Submitted By - HEMLATA
#                                                                         Enrollment NO - 02301192022
```

In [ ]:
```
# WELCOME!!
# Today I will attempt to predict whether or not an individual will suffer a stroke.
# First, I will perform extensive data visualization. This will help me to see if there are any features that look to b
# Next, I will build multiple models and select the best performing one. I will use f1 score as my primary metric as ou
```

In [ ]:
```
# Model Interpretation :-
# I will also delve in to Model Interpretation This is incfedibly important in industry. Often we need to explain very
# algorithms to a non-technical audience, so any tool that can help this process should be mastered.
```

In [ ]:
```
# IMPORTING LIBRARIES :-
```

In [3]:
```
import numpy as np
import pandas as pd
```

In [4]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [5]:
```
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import matplotlib.gridspec as grid_spec
import seaborn as sns
#from imblearn.over_sampling import SMOTE
#import scikitplot as skplt


from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler,LabelEncoder
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.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, recall_score, roc_auc_score, precision_score, f1_score
import warnings
warnings.filterwarnings('ignore')
!pip install pywaffle
```

```
Requirement already satisfied: pywaffle in e:\anaconda\lib\site-packages (1.1.0)
Requirement already satisfied: matplotlib in e:\anaconda\lib\site-packages (from pywaffle) (3.0.3)
Requirement already satisfied: fontawesomefree in e:\anaconda\lib\site-packages (from pywaffle) (6.5.1)
Requirement already satisfied: numpy>=1.10.0 in e:\anaconda\lib\site-packages (from matplotlib->pywaffle) (1.21.6)
Requirement already satisfied: cycler>=0.10 in e:\anaconda\lib\site-packages (from matplotlib->pywaffle) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in e:\anaconda\lib\site-packages (from matplotlib->pywaffle) (1.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in e:\anaconda\lib\site-packages (from matplot
lib->pywaffle) (2.3.1)
Requirement already satisfied: python-dateutil>=2.1 in e:\anaconda\lib\site-packages (from matplotlib->pywaffle) (2.8.
0)
Requirement already satisfied: six in e:\anaconda\lib\site-packages (from cycler>=0.10->matplotlib->pywaffle) (1.12.0)
Requirement already satisfied: setuptools in e:\anaconda\lib\site-packages (from kiwisolver>=1.0.1->matplotlib->pywaff
le) (40.8.0)
```

In [6]:
```
df = pd.read_csv(r'C:\Users\GIRIRAJ KISHOR\Downloads\healthcare-dataset-stroke-data.csv')
```

In [7]:
```
df.head(3)
```

Out[7]:

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

In [8]:
```
# Missing Data
df.isnull().sum()
```

Out[8]:
```
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
```

In [9]:
```
# How can we deal with blanks in our data?
#There are many ways. One can simply drop these records, fill the blanks with the mean, the median, or even simply the
#But there are other, more unusual ways.
#Here I will use a DECISION TREE to predict the missing BMI
```

In [10]:
```
DT_bmi_pipe = Pipeline( steps=[
                               ('scale',StandardScaler()),
                               ('lr',DecisionTreeRegressor(random_state=42))
                              ])
X = df[['age','gender','bmi']].copy()
X.gender = X.gender.replace({'Male':0,'Female':1,'Other':-1}).astype(np.uint8)

Missing = X[X.bmi.isna()]
X = X[~X.bmi.isna()]
Y = X.pop('bmi')
DT_bmi_pipe.fit(X,Y)
predicted_bmi = pd.Series(DT_bmi_pipe.predict(Missing[['age','gender']]),index=Missing.index)
df.loc[Missing.index,'bmi'] = predicted_bmi
```

In [11]:
```
print('Missing values: ',sum(df.isnull().sum()))
```

```
Missing values:  0
```

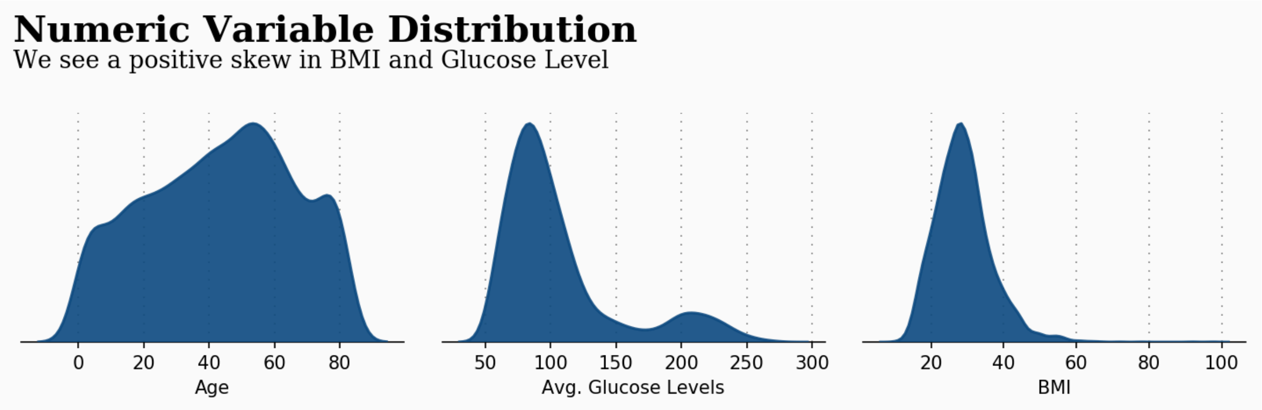
In [12]:
```
# We've replaced all missing values, Now we can move to the next step
```

DATA VISUALIZATION AND PREPARATION

In [13]:
```
# We have now dealt with the missing values in the data.Next, I want to explore the data, Does age makes one more likel
# These are all questions that can be explored and answered with some data visulization.
```

In [14]:
```
variables = [variable for variable in df.columns if variable not in ['id','stroke']]

conts = ['age','avg_glucose_level','bmi']
```

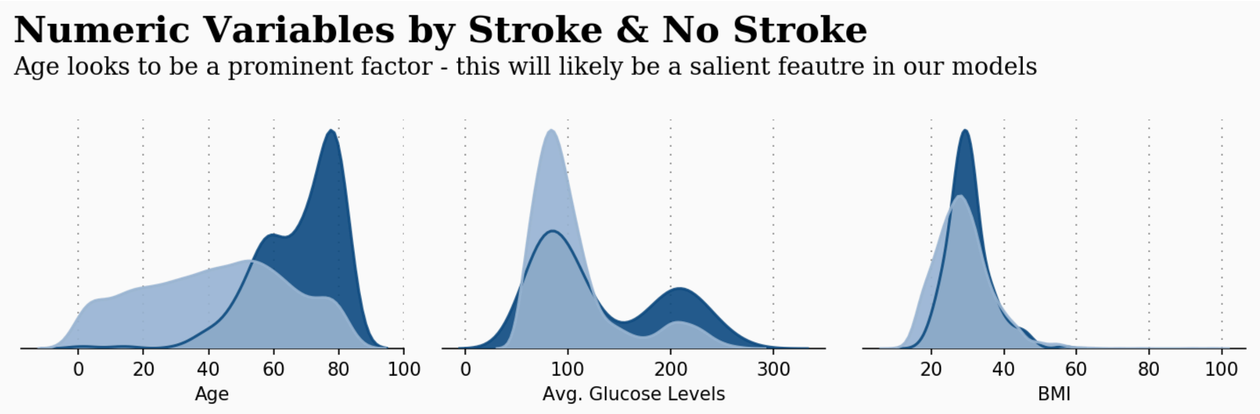
In [15]:
```
fig = plt.figure(figsize=(12, 12), dpi=150, facecolor='#fafafa')
gs = fig.add_gridspec(4, 3)
gs.update(wspace=0.1, hspace=0.4)

background_color = "#fafafa"

plot = 0
for row in range(0, 1):
    for col in range(0, 3):
        locals()["ax"+str(plot)] = fig.add_subplot(gs[row, col])
        locals()["ax"+str(plot)].set_facecolor(background_color)
        locals()["ax"+str(plot)].tick_params(axis='y', left=False)
        locals()["ax"+str(plot)].get_yaxis().set_visible(False)
        for s in ["top","right","left"]:
            locals()["ax"+str(plot)].spines[s].set_visible(False)
        plot += 1

plot = 0
for variable in conts:
    sns.kdeplot(df[variable], ax=locals()["ax"+str(plot)], color='#0f4c81', shade=True, linewidth=1.5, alpha=0.9, zorde
    locals()["ax"+str(plot)].grid(which='major', axis='x', zorder=0, color='gray', linestyle=':', dashes=(1,5))
    plot += 1

ax0.set_xlabel('Age')
ax1.set_xlabel('Avg. Glucose Levels')
ax2.set_xlabel('BMI')

ax0.text(-20, 0.022, 'Numeric Variable Distribution', fontsize=20, fontweight='bold', fontfamily='serif')
ax0.text(-20, 0.02, 'We see a positive skew in BMI and Glucose Level', fontsize=13, fontweight='light', fontfamily='ser

plt.show()
```



In [16]:
```
# So we've gained some understanding on the distributiona of our numeric variables, but we can add more information to
#Let's see how the distribution of our numeric variables is different for those that have strokes, and those that do no
#This could be important for modelling later on
```

In [17]:

```
fig = plt.figure(figsize=(12, 12), dpi=150,facecolor=background_color)
gs = fig.add_gridspec(4, 3)
gs.update(wspace=0.1, hspace=0.4)


plot = 0
for row in range(0, 1):
    for col in range(0, 3):
        locals()["ax"+str(plot)] = fig.add_subplot(gs[row, col])
        locals()["ax"+str(plot)].set_facecolor(background_color)
        locals()["ax"+str(plot)].tick_params(axis='y', left=False)
        locals()["ax"+str(plot)].get_yaxis().set_visible(False)
        for s in ["top","right","left"]:
            locals()["ax"+str(plot)].spines[s].set_visible(False)
        plot += 1

plot = 0

s = df[df['stroke'] == 1]
ns = df[df['stroke'] == 0]

for feature in conts:
    sns.kdeplot(s[feature], ax=locals()["ax"+str(plot)], color='#0f4c81', shade=True, linewidth=1.5, alpha=0.9, zorder=
    sns.kdeplot(ns[feature], ax=locals()["ax"+str(plot)], color='#9bb7d4', shade=True, linewidth=1.5, alpha=0.9, zorder
    locals()["ax"+str(plot)].grid(which='major', axis='x', zorder=0, color='gray', linestyle=':', dashes=(1,5))
    plot += 1

ax0.set_xlabel('Age')
ax1.set_xlabel('Avg. Glucose Levels')
ax2.set_xlabel('BMI')

ax0.text(-20, 0.056, 'Numeric Variables by Stroke & No Stroke', fontsize=20, fontweight='bold', fontfamily='serif')
ax0.text(-20, 0.05, 'Age looks to be a prominent factor - this will likely be a salient feautre in our models',
         fontsize=13, fontweight='light', fontfamily='serif')

plt.show()
```



In [ ]:

```
# This confirms what our intuitions told us. The older you get, the more at risk you get.
#However, you may have notices the low risk values on the y-axis. This is because the dataset is highly imbalanced.
#Only 249 strokes are in our dataset which totals 5000 - around 1 in 20.
```

In [31]:
```
from pywaffle import Waffle

fig = plt.figure(figsize=(7, 2),dpi=150,facecolor=background_color,
    FigureClass=Waffle,
    rows=1,
    values=[1, 19],
    colors=['#0f4c81', "lightgray"],
    characters='🔴',
    font_size=20,vertical=True,
)

fig.text(0.035,0.78,'People Affected by a Stroke in our dataset',fontfamily='serif',fontsize=15,fontweight='bold')
fig.text(0.035,0.65,'This is around 1 in 20 people [249 out of 5000]',fontfamily='serif',fontsize=10)

plt.show()
```

In [32]:
```
# This needs to be considered when modelling of course, but also when formulating risk.
#Strokes are still relatively rare, we are not saying anything is guaranteed, just that risk is increasing
```

In [33]:
```
# Drop single 'Other' gender
no_str_only = no_str_only[(no_str_only['gender'] != 'Other')]
```

In [34]:
```
# General Overview

#We've assessed a few variables so far, and gained some powerful insights.
#I'll now plot several variables in one place, so we can spot interesting trends or features.
#I will split the data in to 'Stroke' and 'No-Stroke' so we can see if these two populations differ in any meaningful w
```

In [35]:
```python
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, 1])
ax2 = fig.add_subplot(gs[0, 2])
ax3 = fig.add_subplot(gs[1, 0])
ax4 = fig.add_subplot(gs[1, 1])
ax5 = fig.add_subplot(gs[1, 2])
ax6 = fig.add_subplot(gs[2, 0])
ax7 = fig.add_subplot(gs[2, 1])
ax8 = fig.add_subplot(gs[2, 2])

background_color = "#f6f6f6"
fig.patch.set_facecolor(background_color) # figure background color

# Plots

## Age

ax0.grid(color='gray', linestyle=':', axis='y', zorder=0, dashes=(1, 5))
positive = pd.DataFrame(str_only["age"])
negative = pd.DataFrame(no_str_only["age"])
sns.kdeplot(positive["age"], ax=ax0, color="#0f4c81", shade=True, label="positive")
sns.kdeplot(negative["age"], ax=ax0, color="#9bb7d4", shade=True, 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")

# Smoking

positive = pd.DataFrame(str_only["smoking_status"].value_counts())
positive["Percentage"] = positive["smoking_status"].apply(lambda x: x/sum(positive["smoking_status"])*100)
negative = pd.DataFrame(no_str_only["smoking_status"].value_counts())
negative["Percentage"] = negative["smoking_status"].apply(lambda x: x/sum(negative["smoking_status"])*100)

ax1.text(0, 4, 'Smoking Status', fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")
ax1.barh(positive.index, positive['Percentage'], color="#0f4c81", zorder=3, height=0.7)
ax1.barh(negative.index, negative['Percentage'], color="#9bb7d4", zorder=3, height=0.3)
ax1.xaxis.set_major_formatter(mtick.PercentFormatter())
ax1.xaxis.set_major_locator(mtick.MultipleLocator(10))

# Gender

positive = pd.DataFrame(str_only["gender"].value_counts())
positive["Percentage"] = positive["gender"].apply(lambda x: x/sum(positive["gender"])*100)
negative = pd.DataFrame(no_str_only["gender"].value_counts())
negative["Percentage"] = negative["gender"].apply(lambda x: x/sum(negative["gender"])*100)

x = np.arange(len(positive))
ax2.text(-0.4, 68.5, 'Gender', fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")
ax2.grid(color='gray', linestyle=':', axis='y', zorder=0, dashes=(1, 5))
ax2.bar(x, height=positive["Percentage"], zorder=3, color="#0f4c81", width=0.4)
ax2.bar(x+0.4, height=negative["Percentage"], zorder=3, color="#9bb7d4", width=0.4)
ax2.set_xticks(x + 0.4 / 2)
ax2.set_xticklabels(['Male', 'Female'])
ax2.yaxis.set_major_formatter(mtick.PercentFormatter())
ax2.yaxis.set_major_locator(mtick.MultipleLocator(10))
for i, j in zip([0, 1], positive["Percentage"]):
    ax2.annotate(f'{j:0.0f}%', xy=(i, j/2), color='#f6f6f6', horizontalalignment='center', verticalalignment='center')
for i, j in zip([0, 1], negative["Percentage"]):
    ax2.annotate(f'{j:0.0f}%', xy=(i+0.4, j/2), color='#f6f6f6', horizontalalignment='center', verticalalignment='cente

# Heart Disease

positive = pd.DataFrame(str_only["heart_disease"].value_counts())
positive["Percentage"] = positive["heart_disease"].apply(lambda x: x/sum(positive["heart_disease"])*100)
negative = pd.DataFrame(no_str_only["heart_disease"].value_counts())
negative["Percentage"] = negative["heart_disease"].apply(lambda x: x/sum(negative["heart_disease"])*100)

x = np.arange(len(positive))
ax3.text(-0.3, 110, 'Heart Disease', fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")
ax3.grid(color='gray', linestyle=':', axis='y', zorder=0, dashes=(1, 5))
ax3.bar(x, height=positive["Percentage"], zorder=3, color="#0f4c81", width=0.4)
ax3.bar(x+0.4, height=negative["Percentage"], zorder=3, color="#9bb7d4", width=0.4)
ax3.set_xticks(x + 0.4 / 2)
ax3.set_xticklabels(['No History', 'History'])
ax3.yaxis.set_major_formatter(mtick.PercentFormatter())
ax3.yaxis.set_major_locator(mtick.MultipleLocator(20))
for i, j in zip([0, 1], positive["Percentage"]):
    ax3.annotate(f'{j:0.0f}%', xy=(i, j/2), color='#f6f6f6', horizontalalignment='center', verticalalignment='center')
for i, j in zip([0, 1], negative["Percentage"]):
    ax3.annotate(f'{j:0.0f}%', xy=(i+0.4, j/2), color='#f6f6f6', horizontalalignment='center', verticalalignment='cente

# Title

ax4.spines["bottom"].set_visible(False)
```

```
ax4.tick_params(left=False, bottom=False)
ax4.set_xticklabels([])
ax4.set_yticklabels([])
ax4.text(0.5, 0.6, 'Can we see patterns for patients in our data?', horizontalalignment='center', verticalalignment='ce
         fontsize=22, fontweight='bold', fontfamily='serif', color="#323232")
ax4.text(0.15, 0.57, "Stroke", fontweight="bold", fontfamily='serif', fontsize=22, color='#0f4c81')
ax4.text(0.41, 0.57, "&", fontweight="bold", fontfamily='serif', fontsize=22, color='#323232')
ax4.text(0.49, 0.57, "No-Stroke", fontweight="bold", fontfamily='serif', fontsize=22, color='#9bb7d4')

# Glucose

ax5.grid(color='gray', linestyle=':', axis='y', zorder=0, dashes=(1, 5))
positive = pd.DataFrame(str_only["avg_glucose_level"])
negative = pd.DataFrame(no_str_only["avg_glucose_level"])
sns.kdeplot(positive["avg_glucose_level"], ax=ax5, color="#0f4c81", shade=True, label="positive")
sns.kdeplot(negative["avg_glucose_level"], ax=ax5, color="#9bb7d4", shade=True, label="negative")
ax5.text(-55, 0.01855, 'Avg. Glucose Level', fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")
ax5.yaxis.set_major_locator(mtick.MultipleLocator(2))
ax5.set_ylabel('')
ax5.set_xlabel('')

# BMI

ax6.grid(color='gray', linestyle=':', axis='y', zorder=0, dashes=(1, 5))
positive = pd.DataFrame(str_only["bmi"])
negative = pd.DataFrame(no_str_only["bmi"])
sns.kdeplot(positive["bmi"], ax=ax6, color="#0f4c81", shade=True, label="positive")
sns.kdeplot(negative["bmi"], ax=ax6, color="#9bb7d4", shade=True, label="negative")
ax6.text(-0.06, 0.09, 'BMI', fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")
ax6.yaxis.set_major_locator(mtick.MultipleLocator(2))
ax6.set_ylabel('')
ax6.set_xlabel('')

# Work Type

positive = pd.DataFrame(str_only["work_type"].value_counts())
positive["Percentage"] = positive["work_type"].apply(lambda x: x/sum(positive["work_type"])*100)
positive = positive.sort_index()

negative = pd.DataFrame(no_str_only["work_type"].value_counts())
negative["Percentage"] = negative["work_type"].apply(lambda x: x/sum(negative["work_type"])*100)
negative = negative.sort_index()

ax7.bar(negative.index, height=negative["Percentage"], zorder=3, color="#9bb7d4", width=0.05)
ax7.scatter(negative.index, negative["Percentage"], zorder=3, s=200, color="#9bb7d4")
ax7.bar(np.arange(len(positive.index))+0.4, height=positive["Percentage"], zorder=3, color="#0f4c81", width=0.05)
ax7.scatter(np.arange(len(positive.index))+0.4, positive["Percentage"], zorder=3, s=200, color="#0f4c81")
ax7.yaxis.set_major_formatter(mtick.PercentFormatter())
ax7.yaxis.set_major_locator(mtick.MultipleLocator(10))
ax7.set_xticks(np.arange(len(positive.index))+0.4 / 2)
ax7.set_xticklabels(list(positive.index), rotation=0)
ax7.text(-0.5, 66, 'Work Type', fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")

# Hypertension

positive = pd.DataFrame(str_only["hypertension"].value_counts())
positive["Percentage"] = positive["hypertension"].apply(lambda x: x/sum(positive["hypertension"])*100)
negative = pd.DataFrame(no_str_only["hypertension"].value_counts())
negative["Percentage"] = negative["hypertension"].apply(lambda x: x/sum(negative["hypertension"])*100)

x = np.arange(len(positive))
ax8.text(-0.45, 100, 'Hypertension', fontsize=14, fontweight='bold', fontfamily='serif', color="#323232")
ax8.grid(color='gray', linestyle=':', axis='y', zorder=0, dashes=(1, 5))
ax8.bar(x, height=positive["Percentage"], zorder=3, color="#0f4c81", width=0.4)
ax8.bar(x+0.4, height=negative["Percentage"], zorder=3, color="#9bb7d4", width=0.4)
ax8.set_xticks(x + 0.4 / 2)
ax8.set_xticklabels(['No History', 'History'])
ax8.yaxis.set_major_formatter(mtick.PercentFormatter())
ax8.yaxis.set_major_locator(mtick.MultipleLocator(20))
for i, j in zip([0, 1], positive["Percentage"]):
    ax8.annotate(f'{j:0.0f}%', xy=(i, j/2), color='#f6f6f6', horizontalalignment='center', verticalalignment='center')
for i, j in zip([0, 1], negative["Percentage"]):
    ax8.annotate(f'{j:0.0f}%', xy=(i+0.4, j/2), color='#f6f6f6', horizontalalignment='center', verticalalignment='cente

# Tidy up

for s in ["top", "right", "left"]:
    for i in range(0, 9):
        locals()["ax"+str(i)].spines[s].set_visible(False)

for i in range(0, 9):
    locals()["ax"+str(i)].set_facecolor(background_color)
    locals()["ax"+str(i)].tick_params(axis=u'both', which=u'both', length=0)
    locals()["ax"+str(i)].set_facecolor(background_color)

plt.show()

# Insights, The plots above are quite enlightening.
```

```
# As discussed earlier, we again note the importance of Age, amongst other things.
```

DATA PREPARATION :-

In [36]:
```python
# Encoding categorical values

df['gender'] = df['gender'].replace({'Male':0,'Female':1,'Other':-1}).astype(np.uint8)
df['Residence_type'] = df['Residence_type'].replace({'Rural':0,'Urban':1}).astype(np.uint8)
df['work_type'] = df['work_type'].replace({'Private':0,'Self-employed':1,'Govt_job':2,'children':-1,'Never_worked':-2})
```

In [37]:
```python
                                          # DATA BALANCING
#Can we predict whether or not an indiviudal will suffer a stroke?
#First, I will use the SMOTE (Synthetic Minority Over-sampling Technique) to balance our dataset.
#Currently, as I mentioned above, there are many more negative examples of a stroke and this could hinder our model.
#This can be addressed using SMOTE.
```

In [38]:
```python
# Inverse of Null Accuracy

print('Inverse of Null Accuracy: ',249/(249+4861))
print('Null Accuracy: ',4861/(4861+249))
```

```
Inverse of Null Accuracy:  0.0487279843444227
Null Accuracy:  0.9512720156555773
```

In [39]:
```python
X  = df[['gender','age','hypertension','heart_disease','work_type','avg_glucose_level','bmi']]
y = df['stroke']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.3, random_state=42)
```

In [40]:
```python
X_test.head(2)
```

Out[40]:

	gender	age	hypertension	heart_disease	work_type	avg_glucose_level	bmi
4688	0	31	0	0	1	64.85	23.0
4478	0	40	0	0	1	65.29	28.3

In [41]:
```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in e:\anaconda\lib\site-packages (0.12.2)
Requirement already satisfied: joblib>=1.1.1 in e:\anaconda\lib\site-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in e:\anaconda\lib\site-packages (from imbalanced-learn) (3.1.0)
Requirement already satisfied: scipy>=1.5.0 in e:\anaconda\lib\site-packages (from imbalanced-learn) (1.7.3)
Requirement already satisfied: numpy>=1.17.3 in e:\anaconda\lib\site-packages (from imbalanced-learn) (1.21.6)
Requirement already satisfied: scikit-learn>=1.0.2 in e:\anaconda\lib\site-packages (from imbalanced-learn) (1.0.2)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
                                          #Baseline
# For such an imbalanced dataset, a useful baseline can be to beat the 'Null Accuracy', and in our case, since we're lo
#For this case, 249/(249+4861) = 0.048
# So a good target to beat would be 5%~ for recall for positive stroke patients.
```

In [42]:
```
# Our data is biased, we can fix this with SMOTE

from imblearn.over_sampling import SMOTE

oversample = SMOTE()
X_train_resh, y_train_resh = oversample.fit_resample(X_train, y_train.ravel())

# Our data is now equal
```

DATA MODELING

In [ ]:
```
# Models :-
# I will model Random Forest, SVM, and Logisitc Regression for this classificatioin task.
# In addition, I will utilise 10 fold cross validation.
```

In [43]:
```
# Models

# Scale our data in pipeline, then split

rf_pipeline = Pipeline(steps = [('scale',StandardScaler()),('RF',RandomForestClassifier(random_state=42))])
svm_pipeline = Pipeline(steps = [('scale',StandardScaler()),('SVM',SVC(random_state=42))])
logreg_pipeline = Pipeline(steps = [('scale',StandardScaler()),('LR',LogisticRegression(random_state=42))])
```

In [44]:
```
rf_cv = cross_val_score(rf_pipeline,X_train_resh,y_train_resh,cv=10,scoring='f1')
svm_cv = cross_val_score(svm_pipeline,X_train_resh,y_train_resh,cv=10,scoring='f1')
logreg_cv = cross_val_score(logreg_pipeline,X_train_resh,y_train_resh,cv=10,scoring='f1')
```

In [45]:
```
print('Mean f1 scores:')
print('Random Forest mean :',cross_val_score(rf_pipeline,X_train_resh,y_train_resh,cv=10,scoring='f1').mean())
print('SVM mean :',cross_val_score(svm_pipeline,X_train_resh,y_train_resh,cv=10,scoring='f1').mean())
print('Logistic Regression mean :',cross_val_score(logreg_pipeline,X_train_resh,y_train_resh,cv=10,scoring='f1').mean()
```

```
Mean f1 scores:
Random Forest mean : 0.9366272051900021
SVM mean : 0.8798886135923784
Logistic Regression mean : 0.8232068683289704
```

Using RANDOM FOREST :-

Random Forest performed the Best

In [ ]:
```
# Now let's try it on the unseen negative data
```

In [46]:
```
rf_pipeline.fit(X_train_resh,y_train_resh)
svm_pipeline.fit(X_train_resh,y_train_resh)
logreg_pipeline.fit(X_train_resh,y_train_resh)

#X = df.loc[:,X.columns]
#Y = df.loc[:,'stroke']

rf_pred   =rf_pipeline.predict(X_test)
svm_pred  = svm_pipeline.predict(X_test)
logreg_pred   = logreg_pipeline.predict(X_test)

rf_cm  = confusion_matrix(y_test,rf_pred )
svm_cm = confusion_matrix(y_test,svm_pred)
logreg_cm  = confusion_matrix(y_test,logreg_pred )

rf_f1  = f1_score(y_test,rf_pred)
svm_f1 = f1_score(y_test,svm_pred)
logreg_f1  = f1_score(y_test,logreg_pred)
```

In [47]:
```
print('Mean f1 scores:')
print('RF mean :',rf_f1)
print('SVM mean :',svm_f1)
print('LR mean :',logreg_f1)
```

```
Mean f1 scores:
RF mean : 0.1541501976284585
SVM mean : 0.15746421267893662
LR mean : 0.19190968955785512
```

In [48]:
```
from sklearn.metrics import plot_confusion_matrix, classification_report

print(classification_report(y_test,rf_pred))

print('Accuracy Score: ',accuracy_score(y_test,rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.94      3404
           1       0.12      0.23      0.15       173

    accuracy                           0.88      3577
   macro avg       0.54      0.57      0.54      3577
weighted avg       0.92      0.88      0.90      3577

Accuracy Score:  0.8803466592116299
```

In [ ]:
```
# Good accuracy, poor recall !!
# I will try using a grid search to find the optimal parameters for our Random Forest
```

In [49]:
```
from sklearn.model_selection import GridSearchCV

n_estimators =[64,100,128,200]
max_features = [2,3,5,7]
bootstrap = [True,False]

param_grid = {'n_estimators':n_estimators,
             'max_features':max_features,
             'bootstrap':bootstrap}
```

In [50]:
```
rfc = RandomForestClassifier()
```

In [51]:
```
rfc = RandomForestClassifier(max_features=2,n_estimators=100,bootstrap=True)

rfc.fit(X_train_resh,y_train_resh)

rfc_tuned_pred = rfc.predict(X_test)
```

In [52]:
```
print(classification_report(y_test,rfc_tuned_pred))

print('Accuracy Score: ',accuracy_score(y_test,rfc_tuned_pred))
print('F1 Score: ',f1_score(y_test,rfc_tuned_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93      3404
           1       0.11      0.21      0.14       173

    accuracy                           0.88      3577
   macro avg       0.53      0.56      0.54      3577
weighted avg       0.92      0.88      0.90      3577

Accuracy Score:  0.8778305842885099
F1 Score:  0.14481409001956946
```

Using LOGISTIC REGRESSION :-

In [ ]:
```
# What about Logistic Regression?
# Logistic Regression had the highest f1 score above, so perhaps we can tune that for better results
```

In [53]:
```
penalty = ['l1','l2']
C = [0.001, 0.01, 0.1, 1, 10, 100]

log_param_grid = {'penalty': penalty,
                  'C': C}
logreg = LogisticRegression()
grid = GridSearchCV(logreg,log_param_grid)
```

In [54]:
```
# Let's use those params now

logreg_pipeline = Pipeline(steps = [('scale',StandardScaler()),('LR',LogisticRegression(C=0.1,penalty='l2',random_state

logreg_pipeline.fit(X_train_resh,y_train_resh)

#logreg.fit(X_train_resh,y_train_resh)

logreg_tuned_pred   = logreg_pipeline.predict(X_test)

print(classification_report(y_test,logreg_tuned_pred))

print('Accuracy Score: ',accuracy_score(y_test,logreg_tuned_pred))
print('F1 Score: ',f1_score(y_test,logreg_tuned_pred))

# So the hyper-parameter tuning has helped the Logisitc Regression model.
# It's recall score is much better than Random Forest's - even if the overall acuracy is down.
```

```
              precision    recall  f1-score   support

           0       0.97      0.77      0.86      3404
           1       0.11      0.58      0.19       173

    accuracy                           0.76      3577
   macro avg       0.54      0.67      0.52      3577
weighted avg       0.93      0.76      0.82      3577

Accuracy Score:  0.7564998602180598
F1 Score:  0.18825722273998136
```

In [ ]:
```
# The art is in finding the balance between 'hits' and 'misses'.
# F1 score is a decent starting point for this as it is the weighted average of several metrics.

# Here's a chart showing what I mean
```

Using SUPPORT VECTOR MACHINE (SVM) :-

In [60]:
```
svm_pipeline = Pipeline(steps = [('scale',StandardScaler()),('SVM',SVC(C=1000,gamma=0.01,kernel='rbf',random_state=42))

svm_pipeline.fit(X_train_resh,y_train_resh)

svm_tuned_pred   = svm_pipeline.predict(X_test)
```

In [61]:
```
print(classification_report(y_test,svm_tuned_pred))

print('Accuracy Score: ',accuracy_score(y_test,svm_tuned_pred))
print('F1 Score: ',f1_score(y_test,svm_tuned_pred))
```

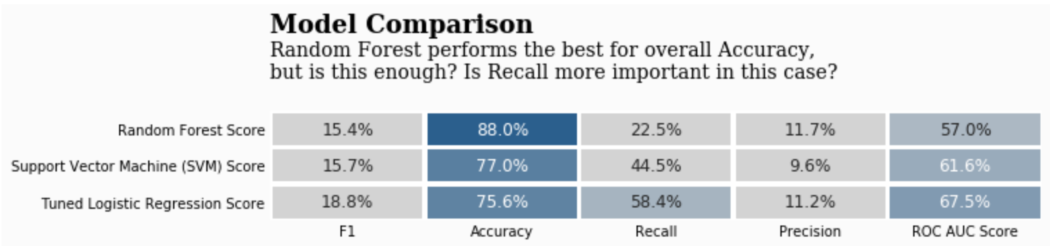
```
              precision    recall  f1-score   support

           0       0.96      0.78      0.86      3404
           1       0.09      0.42      0.15       173

    accuracy                           0.76      3577
   macro avg       0.53      0.60      0.50      3577
weighted avg       0.92      0.76      0.83      3577

Accuracy Score:  0.7648867766284596
F1 Score:  0.1461928934010152
```

Comparison :-

In [ ]:
```
# The tuned Random Forest gave us a much higher accuracy score of around 94%, but with a recall for Stroke patients of 
#The original model had an accuracy of 88%, but a recall for stroke patients of 24%.
#This is often where domain knowledge comes in to play.

# In my opinion, the model is better off predicting those who will suffer a stroke, rather than predicting who will not
```

In [62]:
```
# Make dataframes to plot

rf_df = pd.DataFrame(data=[f1_score(y_test,rf_pred),accuracy_score(y_test, rf_pred), recall_score(y_test, rf_pred),
                   precision_score(y_test, rf_pred), roc_auc_score(y_test, rf_pred)],
             columns=['Random Forest Score'],
             index=["F1","Accuracy", "Recall", "Precision", "ROC AUC Score"])

svm_df = pd.DataFrame(data=[f1_score(y_test,svm_pred),accuracy_score(y_test, svm_pred), recall_score(y_test, svm_pred),
                   precision_score(y_test, svm_pred), roc_auc_score(y_test, svm_pred)],
             columns=['Support Vector Machine (SVM) Score'],
             index=["F1","Accuracy", "Recall", "Precision", "ROC AUC Score"])

lr_df = pd.DataFrame(data=[f1_score(y_test,logreg_tuned_pred),accuracy_score(y_test, logreg_tuned_pred), recall_score(y
                   precision_score(y_test, logreg_tuned_pred), roc_auc_score(y_test, logreg_tuned_pred)],
             columns=['Tuned Logistic Regression Score'],
             index=["F1","Accuracy", "Recall", "Precision", "ROC AUC Score"])
```

In [63]:
```
df_models = round(pd.concat([rf_df,svm_df,lr_df], axis=1),3)
import matplotlib
colors = ["lightgray","lightgray","#0f4c81"]
colormap = matplotlib.colors.LinearSegmentedColormap.from_list("", colors)

background_color = "#fbfbfb"

fig = plt.figure(figsize=(10,8)) # create figure
gs = fig.add_gridspec(4, 2)
gs.update(wspace=0.1, hspace=0.5)
ax0 = fig.add_subplot(gs[0, :])

sns.heatmap(df_models.T, cmap=colormap,annot=True,fmt=".1%",vmin=0,vmax=0.95, linewidths=2.5,cbar=False,ax=ax0,annot_kw
fig.patch.set_facecolor(background_color) # figure background color
ax0.set_facecolor(background_color)

ax0.text(0,-2.15,'Model Comparison',fontsize=18,fontweight='bold',fontfamily='serif')
ax0.text(0,-0.9,'Random Forest performs the best for overall Accuracy,\nbut is this enough? Is Recall more important in
ax0.tick_params(axis=u'both', which=u'both',length=0)


plt.show()
```

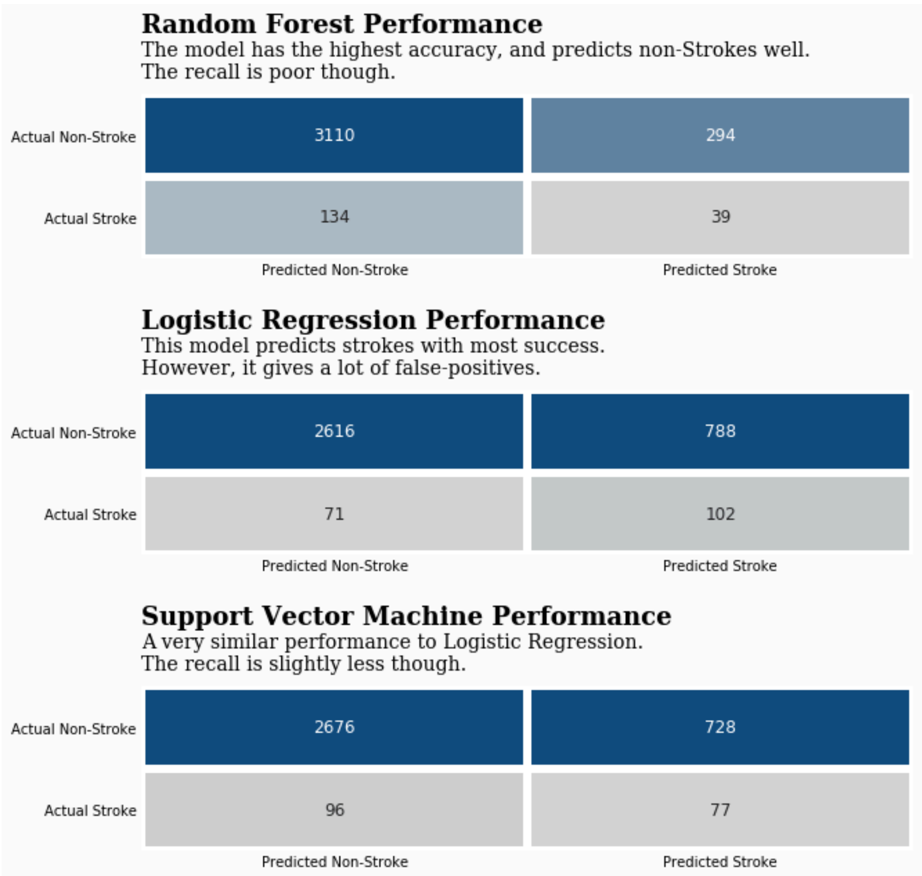


Models By Model Confusion Matrix :-

In [ ]:
```
# Now we have selected our models, we can view how they performed in each prediction.
#A great way to visualise where your data performs well, and where it performs poorly.
```

In [64]:

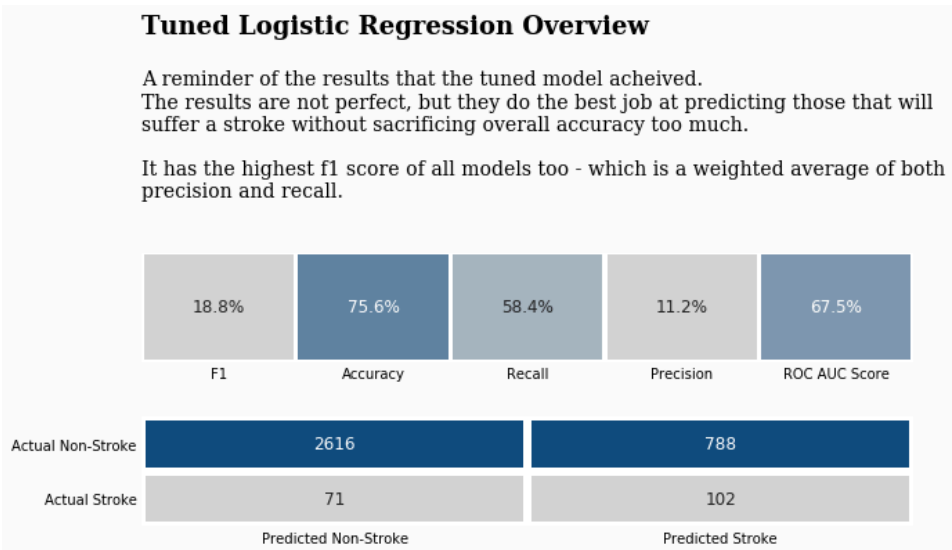
```python
# Plotting our results

colors = ["lightgray","#0f4c81","#0f4c81","#0f4c81","#0f4c81","#0f4c81","#0f4c81","#0f4c81"]
colormap = matplotlib.colors.LinearSegmentedColormap.from_list("", colors)


background_color = "#fbfbfb"

fig = plt.figure(figsize=(10,14)) # create figure
gs = fig.add_gridspec(4, 2)
gs.update(wspace=0.1, hspace=0.8)
ax0 = fig.add_subplot(gs[0, :])
ax1 = fig.add_subplot(gs[1, :])
ax2 = fig.add_subplot(gs[2, :])
ax0.set_facecolor(background_color) # axes background color

# Overall
sns.heatmap(rf_cm, cmap=colormap,annot=True,fmt="d", linewidths=5,cbar=False,ax=ax0,
            yticklabels=['Actual Non-Stroke','Actual Stroke'],xticklabels=['Predicted Non-Stroke','Predicted Stroke'],a

sns.heatmap(logreg_cm, cmap=colormap,annot=True,fmt="d", linewidths=5,cbar=False,ax=ax1,
            yticklabels=['Actual Non-Stroke','Actual Stroke'],xticklabels=['Predicted Non-Stroke','Predicted Stroke'],a

sns.heatmap(svm_cm, cmap=colormap,annot=True,fmt="d", linewidths=5,cbar=False,ax=ax2,
            yticklabels=['Actual Non-Stroke','Actual Stroke'],xticklabels=['Predicted Non-Stroke','Predicted Stroke'],a



ax0.tick_params(axis=u'both', which=u'both',length=0)
background_color = "#fbfbfb"
fig.patch.set_facecolor(background_color) # figure background color
ax0.set_facecolor(background_color)
ax1.tick_params(axis=u'both', which=u'both',length=0)
ax1.set_facecolor(background_color)
ax2.tick_params(axis=u'both', which=u'both',length=0)
ax2.set_facecolor(background_color)

ax0.text(0,-0.75,'Random Forest Performance',fontsize=18,fontweight='bold',fontfamily='serif')
ax0.text(0,-0.2,'The model has the highest accuracy, and predicts non-Strokes well.\nThe recall is poor though.',fontsi

ax1.text(0,-0.75,'Logistic Regression Performance',fontsize=18,fontweight='bold',fontfamily='serif')
ax1.text(0,-0.2,'This model predicts strokes with most success.\nHowever, it gives a lot of false-positives.',fontsize=

ax2.text(0,-0.75,'Support Vector Machine Performance',fontsize=18,fontweight='bold',fontfamily='serif')
ax2.text(0,-0.2,'A very similar performance to Logistic Regression.\nThe recall is slightly less though.',fontsize=14,f


plt.show()
```



Overall Model Success rate :-

In [ ]:
```
# So all of our models have quite a high accuracy, the highest being 95% (Tuned Random Forest).

# But the recall of Strokes is quite poor across the board.

# Seeing as Random Forest did have the highest accuracy, I will delve deeper in to the model and how it works - woth fe

# However, the actual selection of model would be up for debate due to the recall variance.
```

In [65]:
```
# TUNED LOGISTIC REHRESSION

colors = ["lightgray","lightgray","#0f4c81"]
colormap = matplotlib.colors.LinearSegmentedColormap.from_list("", colors)

background_color = "#fbfbfb"

fig = plt.figure(figsize=(10,8)) # create figure
gs = fig.add_gridspec(4, 2)
gs.update(wspace=0.1, hspace=0.5)
ax0 = fig.add_subplot(gs[0, :])
ax1 = fig.add_subplot(gs[1, :])

sns.heatmap(lr_df.T, cmap=colormap,annot=True,fmt=".1%",vmin=0,vmax=0.95,yticklabels='', linewidths=2.5,cbar=False,ax=ax
fig.patch.set_facecolor(background_color) # figure background color
ax0.set_facecolor(background_color)
ax1.set_facecolor(background_color)

ax0.text(0,-2,'Tuned Logistic Regression Overview',fontsize=18,fontweight='bold',fontfamily='serif')
ax0.text(0,-0.3,
'''
A reminder of the results that the tuned model acheived.
The results are not perfect, but they do the best job at predicting those that will
suffer a stroke without sacrificing overall accuracy too much.

It has the highest f1 score of all models too - which is a weighted average of both
precision and recall.
''',fontsize=14,fontfamily='serif')
ax0.tick_params(axis=u'both', which=u'both',length=0)



# Overall

sns.heatmap(logreg_cm, cmap=colormap,annot=True,fmt="d", linewidths=5,cbar=False,ax=ax1,
            yticklabels=['Actual Non-Stroke','Actual Stroke'],vmax=500,vmin=0,xticklabels=['Predicted Non-Stroke','Predi
ax0.tick_params(axis=u'both', which=u'both',length=0)
ax1.tick_params(axis=u'both', which=u'both',length=0)
plt.show()
```



In [ ]:
```
# I would opt for Logistic Regression.
# It Has a decent accuracy, and the best recall. I feel that on balance it provides the best overall results.
```

Feature Selection Techniques

Logistic Regression with LIME :-

In [ ]:
```
# When it comes to model interpretation, sometimes it is useful to unpack and focus on one example at a time.
# The LIME package enables just that.

# Lime stands for Local Interpretable Model-agnostic Explanations - here's an example:
```

In [75]:
```
pip install lime
```

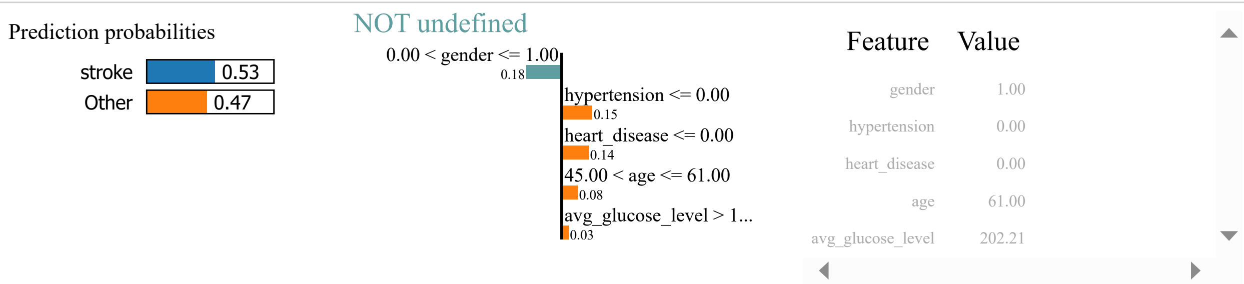
```
Requirement already satisfied: lime in e:\anaconda\lib\site-packages (0.2.0.1)
Requirement already satisfied: tqdm in e:\anaconda\lib\site-packages (from lime) (4.31.1)
Requirement already satisfied: scikit-learn>=0.18 in e:\anaconda\lib\site-packages (from lime) (1.0.2)
Requirement already satisfied: scikit-image>=0.12 in e:\anaconda\lib\site-packages (from lime) (0.14.2)
Requirement already satisfied: scipy in e:\anaconda\lib\site-packages (from lime) (1.7.3)
Requirement already satisfied: matplotlib in e:\anaconda\lib\site-packages (from lime) (3.0.3)
Requirement already satisfied: numpy in e:\anaconda\lib\site-packages (from lime) (1.21.6)
Requirement already satisfied: threadpoolctl>=2.0.0 in e:\anaconda\lib\site-packages (from scikit-learn>=0.18->lime)
(3.1.0)
Requirement already satisfied: joblib>=0.11 in e:\anaconda\lib\site-packages (from scikit-learn>=0.18->lime) (1.3.2)
Requirement already satisfied: networkx>=1.8 in e:\anaconda\lib\site-packages (from scikit-image>=0.12->lime) (2.2)
Requirement already satisfied: six>=1.10.0 in e:\anaconda\lib\site-packages (from scikit-image>=0.12->lime) (1.12.0)
Requirement already satisfied: pillow>=4.3.0 in e:\anaconda\lib\site-packages (from scikit-image>=0.12->lime) (5.4.1)
Requirement already satisfied: PyWavelets>=0.4.0 in e:\anaconda\lib\site-packages (from scikit-image>=0.12->lime) (1.
0.2)
Requirement already satisfied: dask[array]>=1.0.0 in e:\anaconda\lib\site-packages (from scikit-image>=0.12->lime) (1.
1.4)
Requirement already satisfied: cloudpickle>=0.2.1 in e:\anaconda\lib\site-packages (from scikit-image>=0.12->lime) (0.
8.0)
Requirement already satisfied: cycler>=0.10 in e:\anaconda\lib\site-packages (from matplotlib->lime) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in e:\anaconda\lib\site-packages (from matplotlib->lime) (1.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in e:\anaconda\lib\site-packages (from matplot
lib->lime) (2.3.1)
Requirement already satisfied: python-dateutil>=2.1 in e:\anaconda\lib\site-packages (from matplotlib->lime) (2.8.0)
Requirement already satisfied: decorator>=4.3.0 in e:\anaconda\lib\site-packages (from networkx>=1.8->scikit-image>=0.
12->lime) (4.4.0)
Requirement already satisfied: toolz>=0.7.3; extra == "array" in e:\anaconda\lib\site-packages (from dask[array]>=1.0.
0->scikit-image>=0.12->lime) (0.9.0)
Requirement already satisfied: setuptools in e:\anaconda\lib\site-packages (from kiwisolver>=1.0.1->matplotlib->lime)
(40.8.0)
Note: you may need to restart the kernel to use updated packages.
```

In [76]:
```
import lime
import lime.lime_tabular

# LIME has one explainer for all the models
explainer = lime.lime_tabular.LimeTabularExplainer(X.values, feature_names=X.columns.values.tolist(),
                                                   class_names=['stroke'], verbose=True, mode='classification')
```

In [77]:
```
# Choose the jth instance and use it to predict the results for that selection
j = 1
exp = explainer.explain_instance(X.values[j], logreg_pipeline.predict_proba, num_features=5)
```

```
Intercept 0.08251420931085665
Prediction_local [0.30910563]
Right: 0.4681946202784203
```

In [78]:
```
# Show the predictions

exp.show_in_notebook(show_table=True)
```



ELI5 for Feature Explanation :-

In [ ]:
```
# ELI5 stands for Explain it Like I am 5 - a quirky name!

# Here we see the coefficient for each variable - in other words, what our Logistic model puts most value in.
```

In [80]:
```
pip install eli5
```

```
Collecting eli5
Requirement already satisfied: numpy>=1.9.0 in e:\anaconda\lib\site-packages (from eli5) (1.21.6)
Collecting graphviz (from eli5)
  Using cached https://files.pythonhosted.org/packages/de/5e/fcbb22c68208d39edff467809d06c9d81d7d27426460ebc598e55130c
1aa/graphviz-0.20.1-py3-none-any.whl (https://files.pythonhosted.org/packages/de/5e/fcbb22c68208d39edff467809d06c9d81d
7d27426460ebc598e55130c1aa/graphviz-0.20.1-py3-none-any.whl)
Requirement already satisfied: scipy in e:\anaconda\lib\site-packages (from eli5) (1.7.3)
Requirement already satisfied: attrs>17.1.0 in e:\anaconda\lib\site-packages (from eli5) (19.1.0)
Requirement already satisfied: scikit-learn>=0.20 in e:\anaconda\lib\site-packages (from eli5) (1.0.2)
Collecting tabulate>=0.7.7 (from eli5)
  Using cached https://files.pythonhosted.org/packages/40/44/4a5f08c96eb108af5cb50b41f76142f0afa346dfa99d5296fe7202a11
854/tabulate-0.9.0-py3-none-any.whl (https://files.pythonhosted.org/packages/40/44/4a5f08c96eb108af5cb50b41f76142f0afa
346dfa99d5296fe7202a11854/tabulate-0.9.0-py3-none-any.whl)
Collecting jinja2>=3.0.0 (from eli5)
  Using cached https://files.pythonhosted.org/packages/30/6d/6de6be2d02603ab56e72997708809e8a5b0fbfee080735109b40a3564
843/Jinja2-3.1.3-py3-none-any.whl (https://files.pythonhosted.org/packages/30/6d/6de6be2d02603ab56e72997708809e8a5b0fb
fee080735109b40a3564843/Jinja2-3.1.3-py3-none-any.whl)
Requirement already satisfied: six in e:\anaconda\lib\site-packages (from eli5) (1.12.0)
Requirement already satisfied: joblib>=0.11 in e:\anaconda\lib\site-packages (from scikit-learn>=0.20->eli5) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in e:\anaconda\lib\site-packages (from scikit-learn>=0.20->eli5)
(3.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in e:\anaconda\lib\site-packages (from jinja2>=3.0.0->eli5) (2.1.5)
Installing collected packages: graphviz, tabulate, jinja2, eli5
  Found existing installation: Jinja2 2.10
    Uninstalling Jinja2-2.10:
      Successfully uninstalled Jinja2-2.10
Successfully installed eli5-0.13.0 graphviz-0.20.1 jinja2-3.1.3 tabulate-0.9.0
Note: you may need to restart the kernel to use updated packages.
```

In [81]:
```
import eli5

columns_ = ['gender', 'age', 'hypertension', 'heart_disease', 'work_type',
       'avg_glucose_level', 'bmi']

eli5.show_weights(logreg_pipeline.named_steps["LR"], feature_names=columns_)
```

Out[81]: **y=1** top features

Weight?	Feature
+2.052	age
+0.195	bmi
+0.135	avg_glucose_level
-0.281	heart_disease
-0.352	work_type
-0.379	hypertension
-0.401	<BIAS>
-0.728	gender

I Would Select Logistic Regression

In [ ]:
```
# Selection :-

# I would opt for Logistic Regression.
# It Has a decent accuracy, and the best recall. I feel that on balance it provides the best overall results.
```

CONCLUSION

In [ ]:
```
# We started by exploring our data and noticed that certain features, such as Age, looked to be good indicators for pre

# After extensive visualization, we went on to try multiple models.
    #Random Forest, SVM, and Logistic Regression were all tried.

# I then tried hyperparameter tuning on all models to see if I could improve their results.

# While Random Fornegative had the highest accuracy, the Tuned Logistic Regression model provided the best recall and f

# I therefore selected the Tuned Logistic Regression as my model.

# Finally, I used LIME & ELI5 on our chosen Logistic Regression model to show how the features interact with one anothe

# This is a very powerful tool which can be used to help explain and demonstrate how machine learning models work to bu
```

Reference

In [ ]:
```
# DATASET-
https://www.kaggle.com/datasets/m0hammdaliub/stroke-prediction-eda-ml-models

# Understanding Models-

https://www.coursera.org/articles/machine-learning-models
https://www.geeksforgeeks.org/logistic-regression-vs-random-forest-classifier/
https://towardsdatascience.com/understanding-logistic-regression-step-by-step-704a78be7e0a
https://medium.com/filament-ai/painless-explainability-for-nlp-text-models-with-lime-and-eli5-2fa32195a702
```