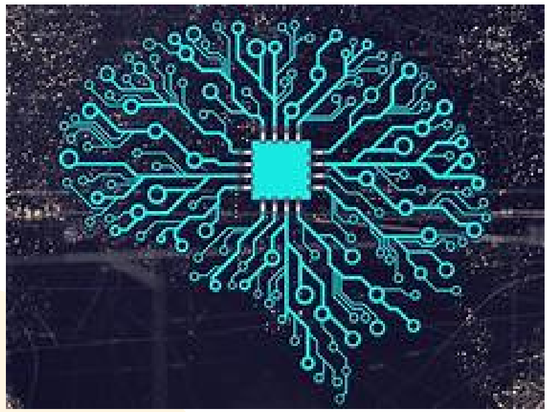
STROKE PREDICTION SYSTEM

ML PROJECT



SUBMITTED TO:
MR. DEBENDRA K DHIR

SUBMITTED BY:
HEMLATA
0230119202022

LIST OF CONTENTS

This project focuses on developing a stroke prediction system using machine learning techniques for buisness problems.

INTRODUCTION

This system aims to leverage ML algorithms to predict the likelihood of an individual experiencing a stroke based on various risk factors and medical history. By analyzing a comprehensive dataset encompassing demographic information, lifestyle factors, medical history, and clinical indicators, ML models can identify patterns and correlations that may indicate an increased risk of stroke.

PROBLEM DEFINITION

The development of a stroke prediction system utilizing ML algorithms holds immense potential for improving stroke prevention and management. By leveraging the power of data-driven insights, this system can assist healthcare professionals in identifying at-risk individuals, ultimately reducing the burden of stroke-related morbidity and mortality.

1) Data Collection: Gathering relevant data points including demographics (age, gender).
2) Data Preprocessing: Cleaning and preprocessing the dataset to handle missing values, encode categorical variables.
3) Model Selection: Employing various ML algorithms such as logistic regression, decision trees, random forests, support vector machines (SVM).
3) Model Training and Evaluation: Training the selected models on a portion of the dataset and evaluating their performance using metrics like accuracy, precision, recall, and area under the ROC curve (AUC).

PROJECT OBJECTIVE

- **Early Identification:** ML-based stroke prediction enables early identification of individuals at high risk, allowing for timely interventions and preventive measures.
- **Personalized Risk Assessment**: By considering multiple risk factors and individual characteristics, the system provides personalized risk assessments tailored to each individual's profile.
- **Improved Clinical Decision-Making:** Clinicians can use the system as a decision support tool to prioritize resources, recommend lifestyle modifications, and initiate preventive therapies for high-risk patients.

DATA CLEANING

1) First step was to import all libraries used in the project. Then Load the DATASET (healthcare-dataset-stroke-data.csv)
2) I will use a DECISION TREE to predict the missing BMI.

```
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

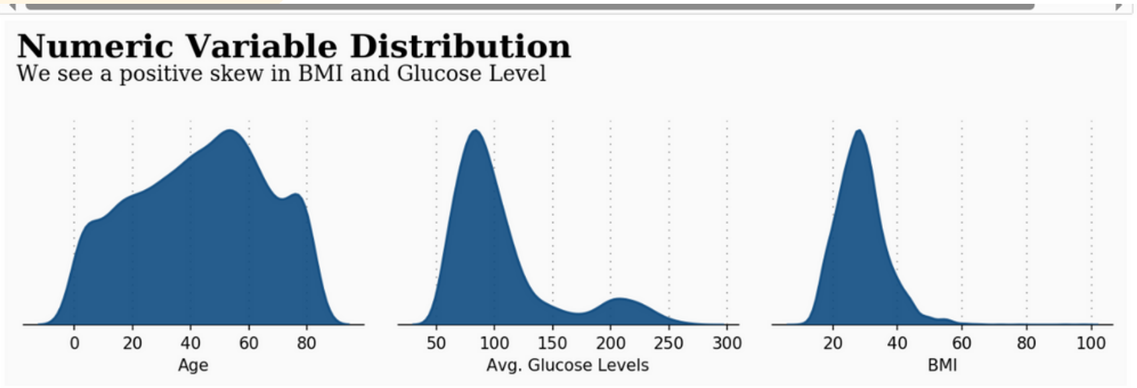
```
In [8]: # Missing Data
        df.isnull().sum()
```

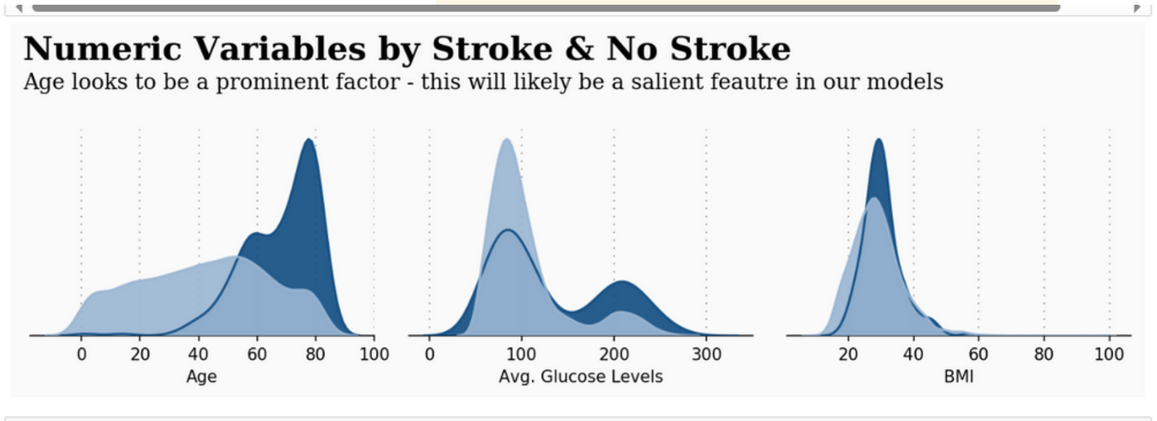
```
In [10]: DT_bmi_pipe = Pipeline( steps=[
                                        ('scale',StandardScaler()),
                                        ('lr',DecisionTreeRegressor(random_state=42))
                                       ])
         X = df[['age','gender','bmi']].copy()
         X.gender = 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
```

DATA VISUALIZATION

I'll explore the data, Does age makes one more likely to suffer a stroke? What about gender? Or BMI?
These are all questions that can be explored and answered with some data visualization.

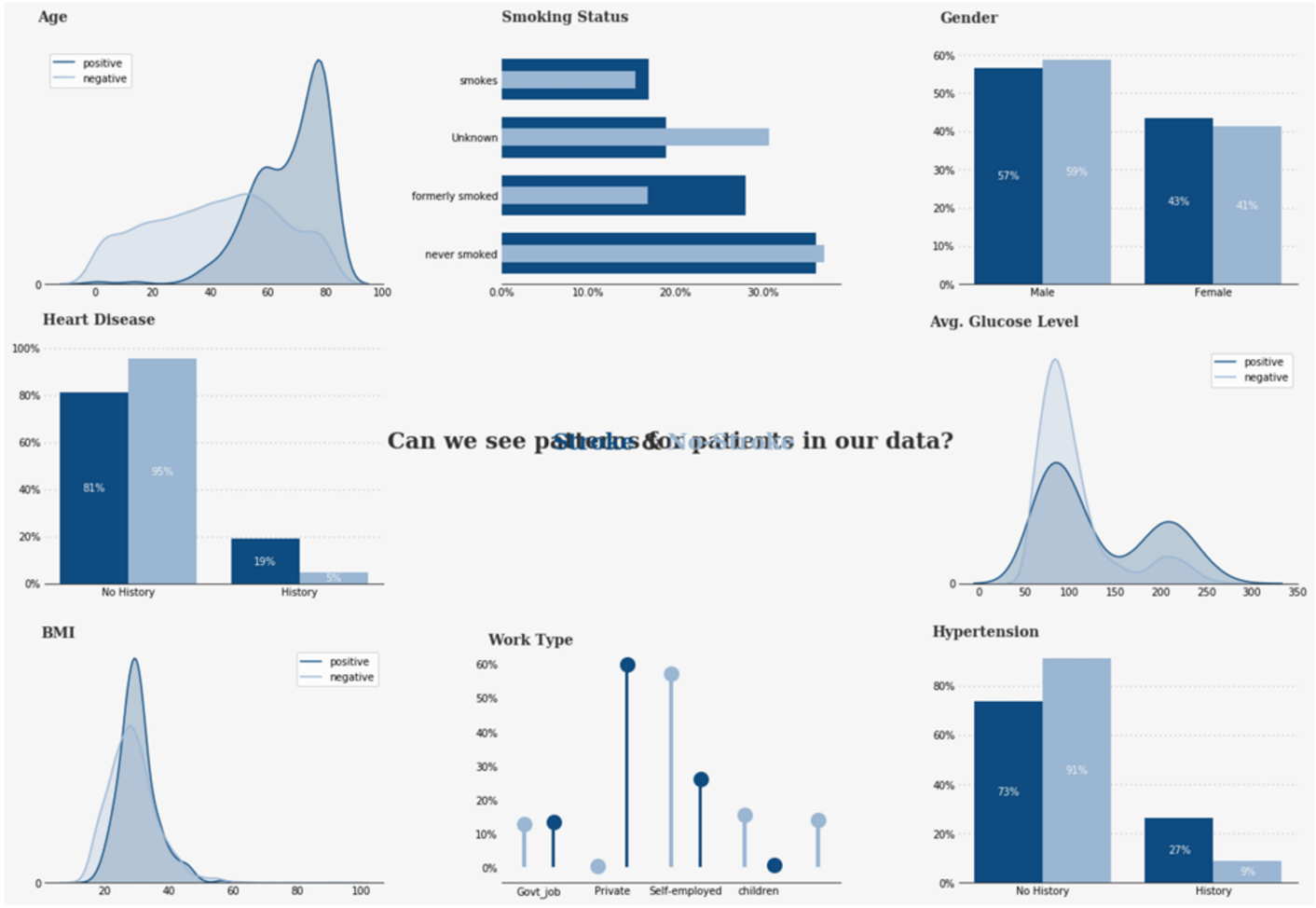




This confirms what our intuitions told us. The older you get, the more at risk you get.

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.

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



Insights, The plots above are quite enlightening.
As discussed earlier, we again note the importance of Age, amongst other things.

DATA BALANCING :

1) I will use the SMOTE (Synthetic Minority Over-sampling Technique) to balance our dataset.
2) 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 [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 [43]:
```
# 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
```

1) I will model Random Forest, SVM, and Logisitc Regression for this classificatioin task

1) 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
```

1) Random forest is having Good accuracy but poor recall

Accuracy Score: 0.877
F1 Score: 0.144

2) USING LOGISTIC REGRESSION

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=42))])

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.
```

```
              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
```

ACCURACY SCORE: 0.756
F1 SCORE: 0.188

It's recall score is much better than Random Forest's - even if the overall acuracy is down.

3) SUPPORT VECTOR MACHINE (SVM)

Using SUPPORT VECTOR MACHINE (SVM) :-

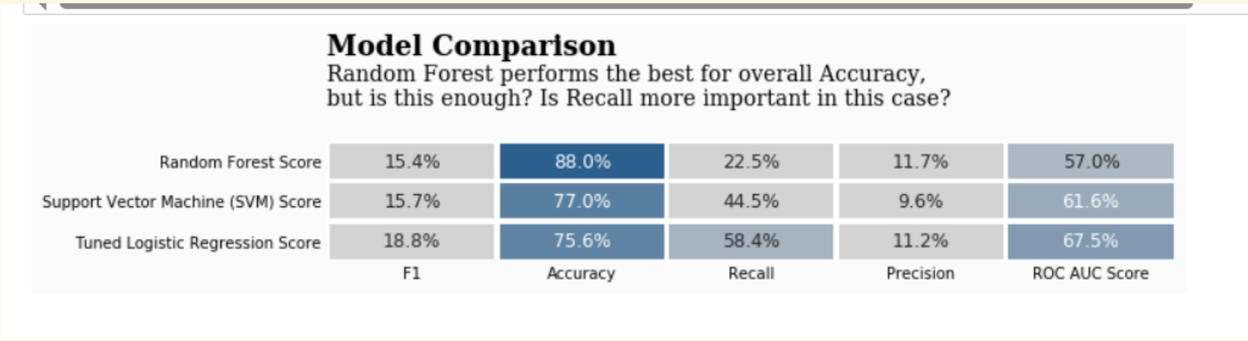
In [60]:
```
svm_pipeline = Pipeline(steps = [('scale',StandardScaler()),('SVM',SVC(C=1000,gamma=0.01,kernel='rbf',random_state=42))])

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
```

ACCURACY SCORE: 0.764
F1 SCORE: 0.146

MODEL COMPARISON



1) The tuned Random Forest gave us a much higher accuracy score of around 94%, but with a recall for Stroke patients of 2%.

2) The original model had an accuracy of 88%, but a recall for stroke patients of 24%.

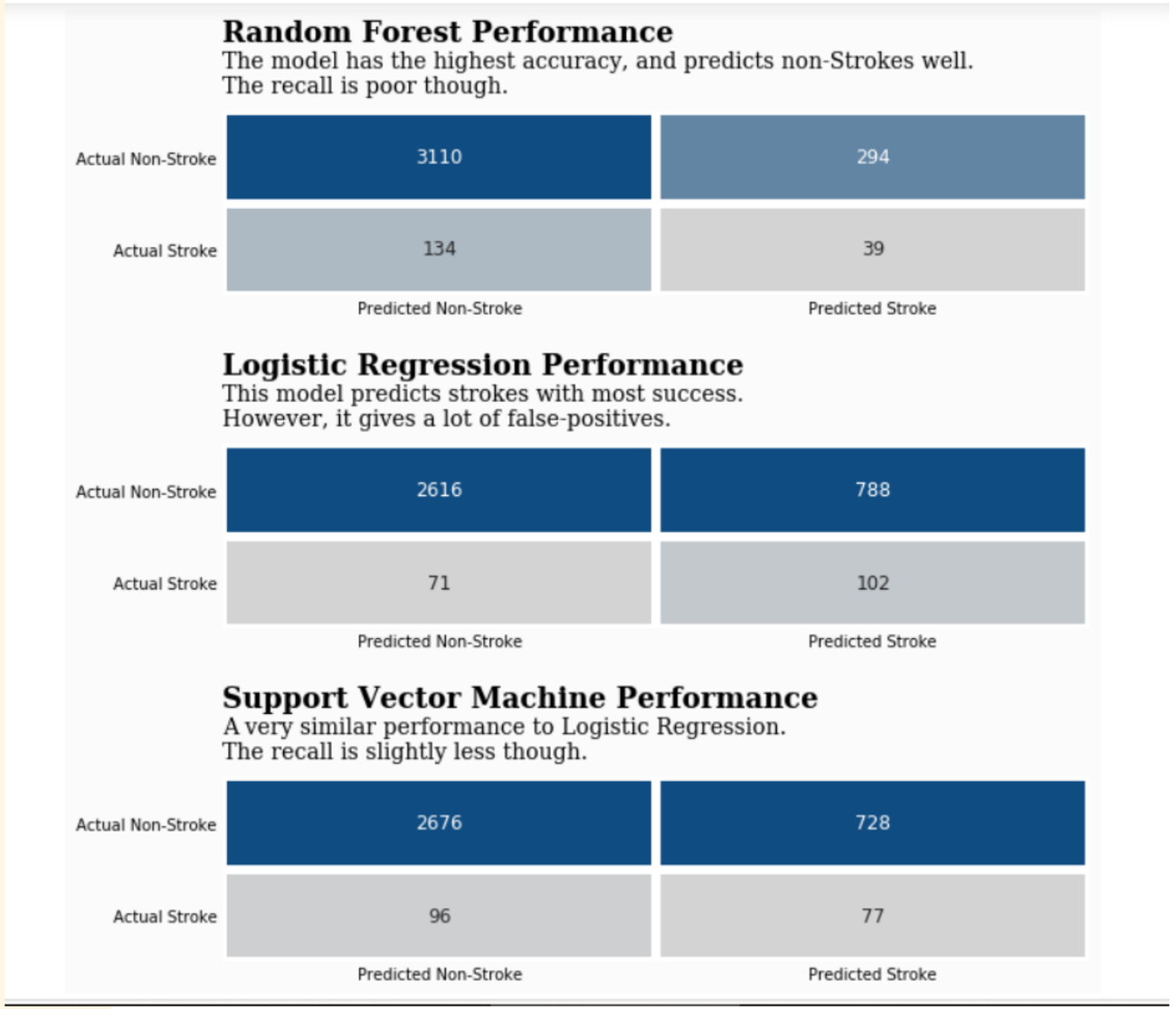
MODELS BY MODEL CONFUSION MATRIX

1) All of our models have quite a high accuracy, the highest being 95% (Tuned Random Forest)
.
2) But the recall of Strokes is quite poor across the board.

3) Seeing as Random Forest did have the highest accuracy, I will delve deeper in to the model and how it works - woth feature importance & LIME.

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

I would opt for Logistic Regression.
It Has a decent accuracy, and the best recall



Feature Selection Techniques

1) LOGISTIC REGRESSION WITH LIME

a) Lime stands for Local Interpretable Model-agnostic Explanations.

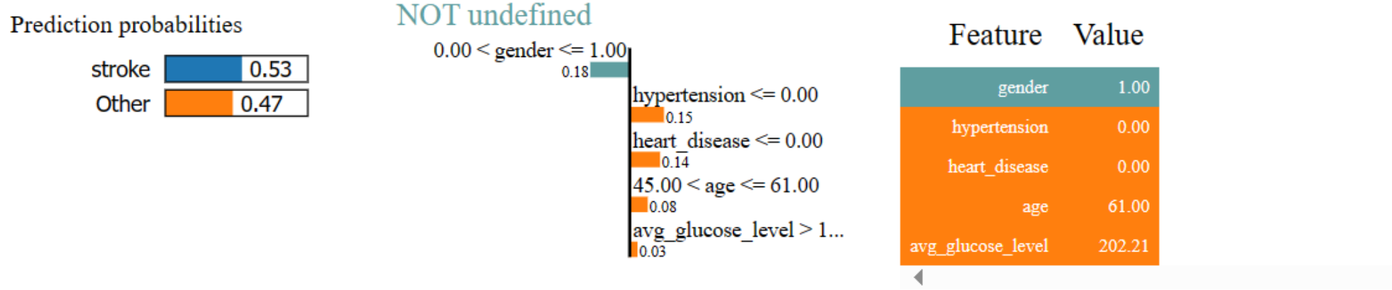
b) When it comes to model interpretation, sometimes it is useful to unpack and focus on one example at a time. The LIME package enables just that

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)
```



2) ELI5 FOR FEATURE EXPLANATION:-

a) 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 values in.

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

CONCLUSION

1) We started by exploring our data and noticed that certain features, such as Age, looked to be good indicators for predicting a stroke.

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

3) I then tried hyperparameter tuning on all models to see if I could improve their results

4) While Random Forest negative had the highest accuracy, the Tuned Logistic Regression model provided the best recall and f1 score.

5) I therefore selected the Tuned Logistic Regression as my model.

6) Finally, I used LIME & ELI5 on our chosen Logistic Regression model to show how the features interact with one another to produce a final prediction in the model.

7) This is a very powerful tool which can be used to help explain and demonstrate how machine learning models work to business stakeholders.

REFERENCE

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


thankyou