STROKE PREDICTION THROUGH CLASSIFICATION

University of Denver - Ritchie School of Engineering

DS Tools 2 – Winter 2021

ABSTRACT

An exploration of machine learning classification algorithms to predict stroke incidence. Data was collected from Kaggle and subsequently cleaned and analyzed using existing Python data science libraries and packages.

Contributors

Sameer Patel

Research Question

For this project, the team decided to determine whether, through the application of techniques and methodologies central to the fields of data science and machine learning, a patient is likely to suffer a stroke based on a combination of input variables.

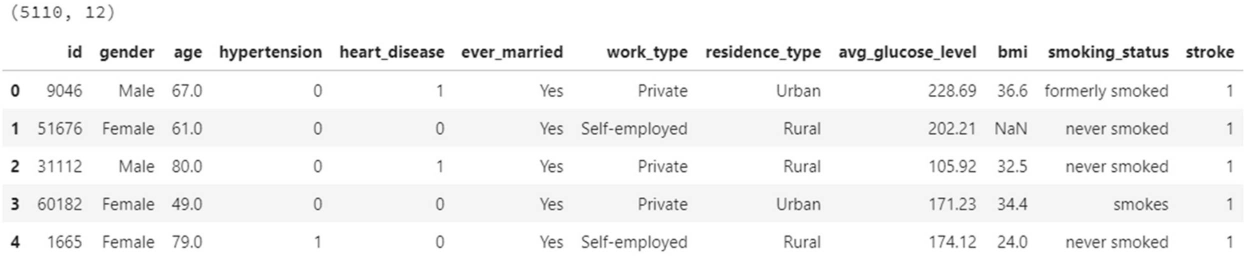
In order to do so, the raw data would need to be analyzed through techniques central to the classification aspects of machine learning.

Data Set Source and Description

The dataset was sourced from Kaggle (https://www.kaggle.com/fedesoriano/stroke-prediction-dataset) and consists of 5110 examples of patient data, each with 11 features + 1 binary indicator variable for whether the patient had suffered a stroke or not. Of the variables, one is trivial, seven are categorical (all nominal), and three are numerical. The target variable, “stroke”, is a binary indicator as aforementioned.

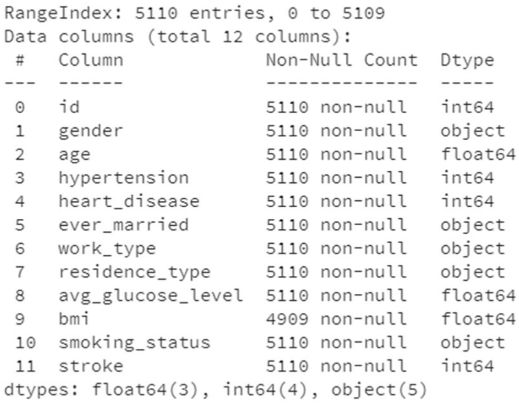
The data was first loaded into a pandas dataframe after downloading the .csv file from the link provided above.

A snippet of the dataset along with datatypes and unique values for the categorical variables is given below:

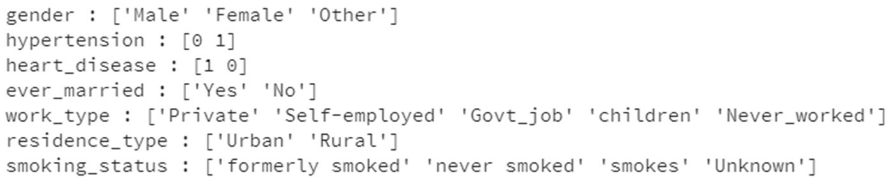


```
(5110, 12)
```

	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
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1



```
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)
```



```
gender : ['Male' 'Female' 'Other']
hypertension : [0 1]
heart_disease : [1 0]
ever_married : ['Yes' 'No']
work_type : ['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
residence_type : ['Urban' 'Rural']
smoking_status : ['formerly smoked' 'never smoked' 'smokes' 'Unknown']
```

Description of Variables and Data Preprocessing

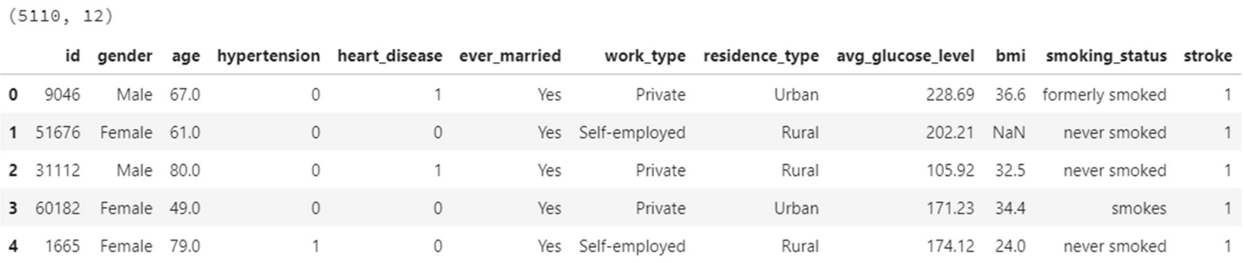
As aforementioned, the data consists of 11 features and 1 binary target. Of the feature variables, one is trivial, seven are categorical (all nominal), and three are numerical.

From the unique values shown in the preceding section, we can see that no ordinality exists for any of the categorical variables. As such, we will one-hot encode these variables before building our model so that they may be used as inputs to our machine learning classifier.

The data preprocessing steps undertaken for this analysis are as follows:

1. Drop inconsequential columns
2. Remove NaNs
    a. "bmi" is the only column with missing values
    b. Because we are building a classifier for something medically-sensitive, and because the number of missing values is less than 5% of the total number of values, we will exclude examples with missing values instead of imputing so as to preserve the integrity of the data.
    c. One-hot encoding of categorical features

A snippet of the resulting dataframe after the pre-processing steps described above is presented below:



```
(5110, 12)
```

	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
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

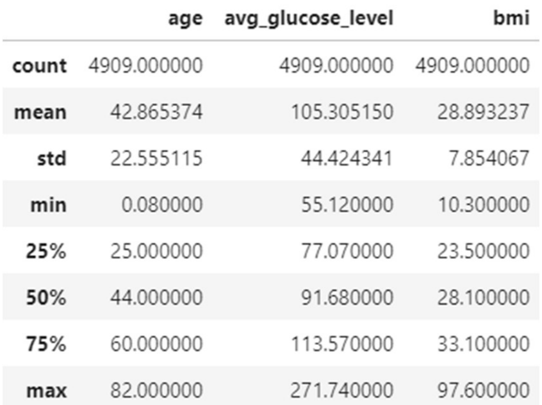
Exploratory Data Analysis (EDA)

It's important to scrutinize feature variables prior to model fitting in order to discern whether the dataset is a good candidate for the classification algorithms the team intends to fit.

Common facets of EDA that are explored prior to model fitting include, but are not limited to:

- Independence of feature variables
    - Multicollinearity
- Distribution of feature variables
- Balance of classes in target variable
- Presence of outliers
- Correlation of independent variables with target

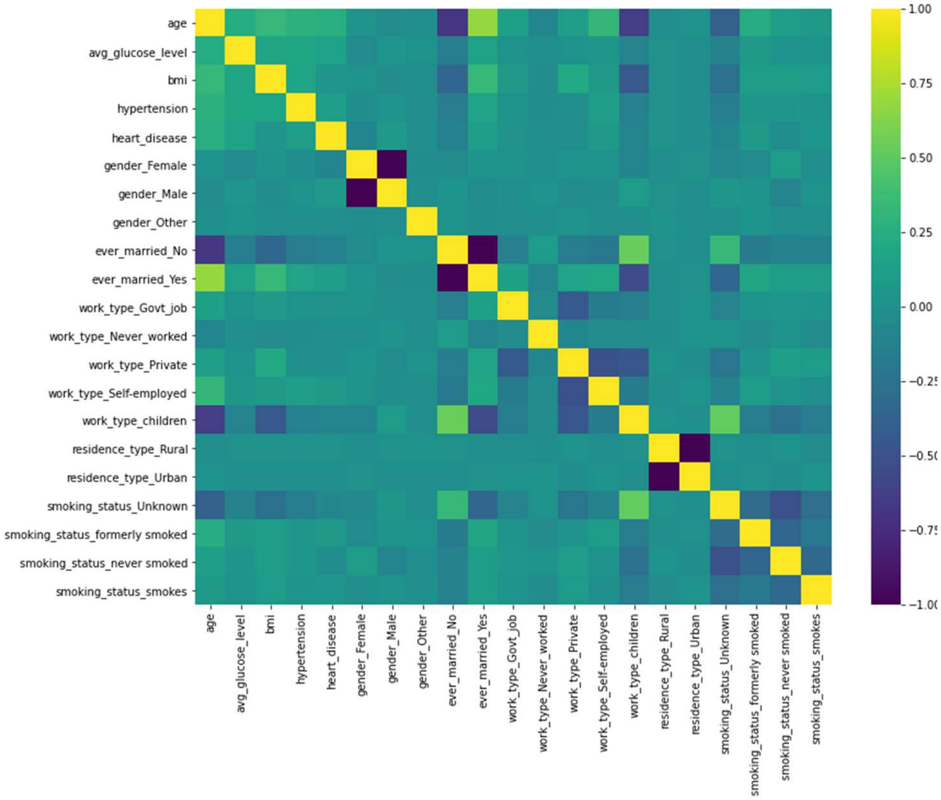
We first start with displaying descriptive statistics of the continuous columns of the dataset using pandas.describe().



	age	avg_glucose_level	bmi
count	4909.000000	4909.000000	4909.000000
mean	42.865374	105.305150	28.893237
std	22.555115	44.424341	7.854067
min	0.080000	55.120000	10.300000
25%	25.000000	77.070000	23.500000
50%	44.000000	91.680000	28.100000
75%	60.000000	113.570000	33.100000
max	82.000000	271.740000	97.600000

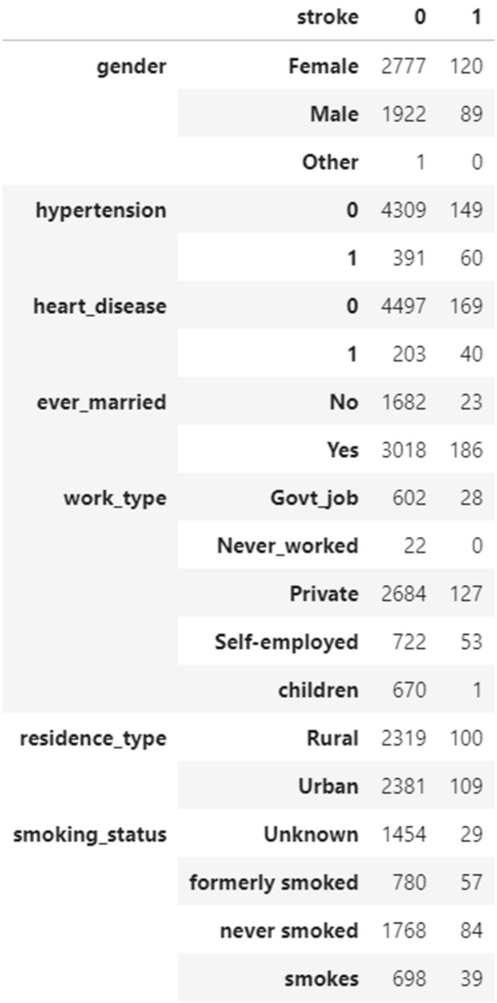
Immediately we can see that outliers do indeed exist within the dataset, specifically in the avg_glucose_level and bmi columns. Outliers would typically be corrected through a variety of correction methodologies (ie. replacing outliers with 1.5 times the IQR, removing them, etc.). In our case, the data will be preserved as we are dealing with actual patient data and any observable trends need to be based on an undoctored dataset. To complement this, feature scaling prior to model fitting will be done through MinMaxScaler() in sklearn, which is robust to outliers within feature columns.

Next, we will evaluate any correlation between the features using a correlation plot and associated heatmap. Note: it is only possible to visualize Pearson's R correlation due to the conversion of categorical features to numerical as conducted above (one-hot encoding).



The correlation heatmap does not indicate extreme unexpected correlation between the features outside of predictable correlations such as age and marriage/kids, age and bmi, or the one-hot encoded binary variables. Thus, the assumption of low multicollinearity necessary for logistic regression is met. Note that Random Forests are built on an ensemble of Decision Trees, which require that no assumptions about the underlying feature data are met prior to model fitting, as they are based on binary splits of the inputs.

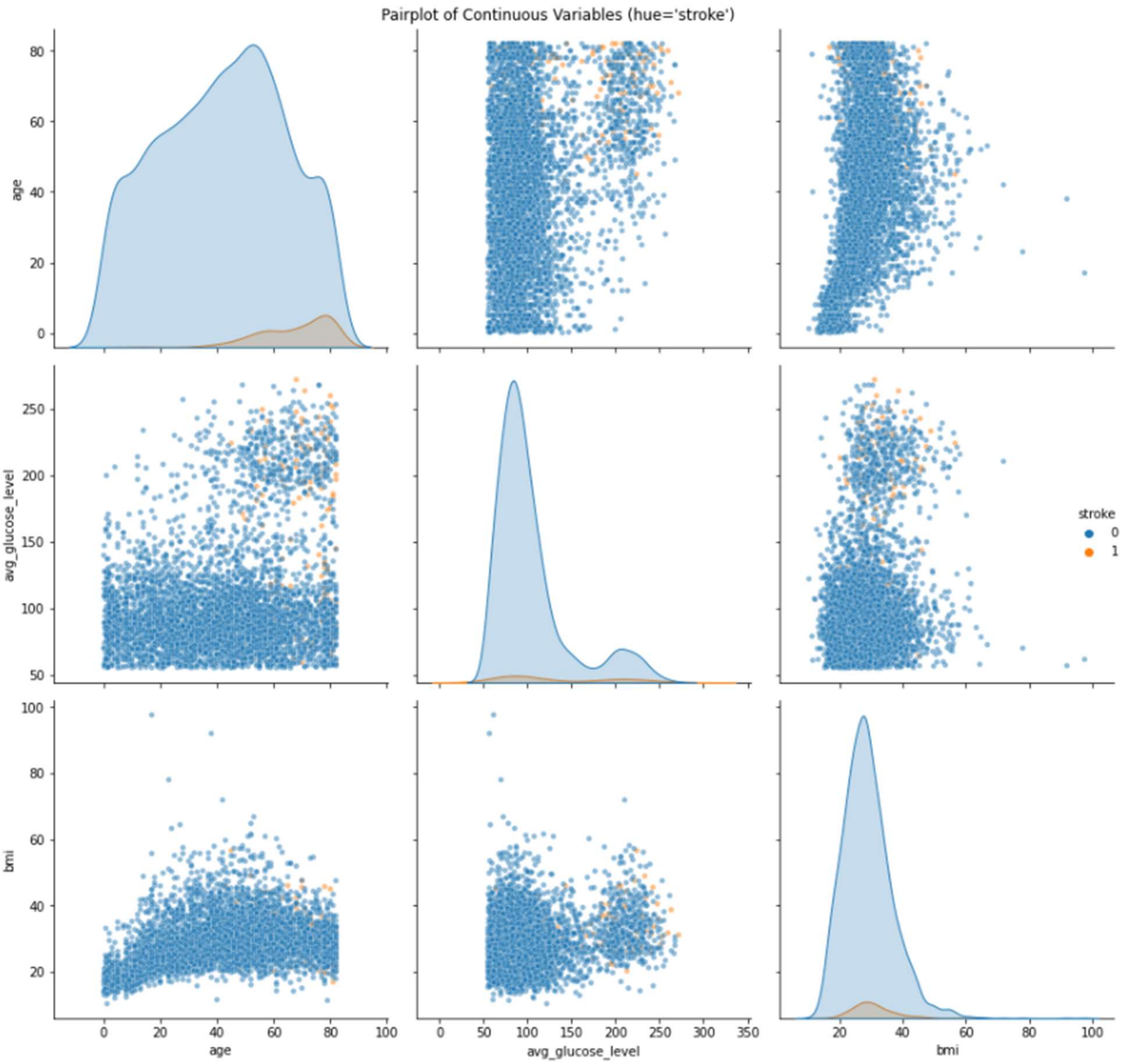
Now we can explore the relationship between each of the feature variables and the target. For categorical variables, this will be accomplished with contingency tables. For continuous variables, this will be accomplished with a pairwise scatterplot.



	stroke	0	1
gender	Female	2777	120
	Male	1922	89
	Other	1	0
hypertension	0	4309	149
	1	391	60
heart_disease	0	4497	169
	1	203	40
ever_married	No	1682	23
	Yes	3018	186
work_type	Govt_job	602	28
	Never_worked	22	0
	Private	2684	127
	Self-employed	722	53
	children	670	1
residence_type	Rural	2319	100
	Urban	2381	109
smoking_status	Unknown	1454	29
	formerly smoked	780	57
	never smoked	1768	84
	smokes	698	39

From the contingency tables, we can see that the dataset is imbalanced with respect to the target, with stroke=1 examples being vastly under-represented. To correct this, we will oversample the training set before fitting to our models in order to allow the underlying algorithms to better pick up on feature patterns and allow for easier differentiation between the classes. We can also notice that the residence_type attribute exhibits almost no relationship with the target, as it has a more-or-less uniform distribution based on stroke class.

A pairwise scatterplot, with density histograms along the main diagonal, is presented below:



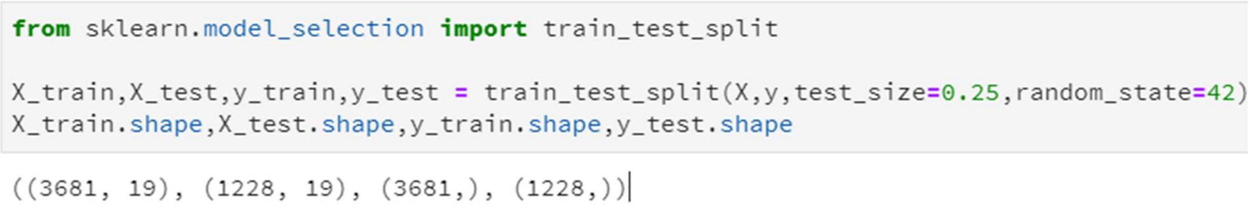
The scatterplots and histograms are separated by class, with orange representing stroke=1 cases. From the density histograms, we notice that both age and avg_glucose_level do appear to have some relationship with the target. However, the densities for bmi for both classes appear to be centered around the same point, leading us to believe that bmi might not be as strong of an indicator as the other features. Bear in mind, this conclusion was reached entirely through visual scrutinization of the plots above, and might not actually be the case.

Based on the EDA conducted, residence_type was dropped from the feature set prior to model fitting.

Data Partitioning

Splitting the data into training and test sets is essential when performing most machine learning tasks, as it allows the model to be evaluated on an “unseen” set of data and can help the scientist determine if the model is overfitting the training data.

We will split the data along a 70/30 train/test ratio in order to build and evaluate our classification algorithms.

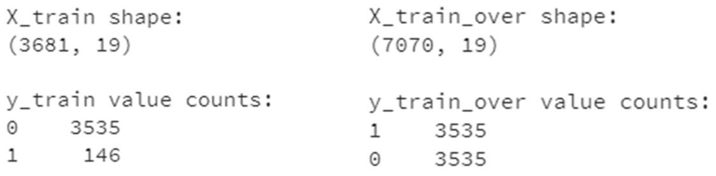


```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((3681, 19), (1228, 19), (3681,), (1228,))
```

Next, we will oversample the training set prior to fitting our models to correct for the imbalanced class distribution and make it easier for the models to pick up on patterns that distinguish the classes. The choice to oversample the minority class instead of undersampling the majority class was taken because even though oversampling could in some cases lead to overfitting, undersampling leads to loss of information. In our case, the problem from undersampling outweighs the problem from overfitting, as we are dealing with sensitive medical records. For oversampling, we will use the imbalanced-learn library (https://imbalanced-learn.org/stable/over_sampling.html).

The shape of the data prior to oversampling, and after oversampling, are show below:



```
X_train shape:                 X_train_over shape:
(3681, 19)                     (7070, 19)

y_train value counts:          y_train_over value counts:
0    3535                      1    3535
1     146                      0    3535
```

Model Building

We are now ready to build our models on the processed and partitioned stroke data. For this analysis, we will explore Random Forest and Logistic Regression classification algorithms.

- Random Forest:
  - Non-parametric (no assumptions about underlying distribution of the variables)
  - Ensemble learning method (base = Decision Trees)
- Logistic Regression:
  - Parametric
  - Learning method uses a logistic function (logit) to model a binary dependent variable based on probabilities

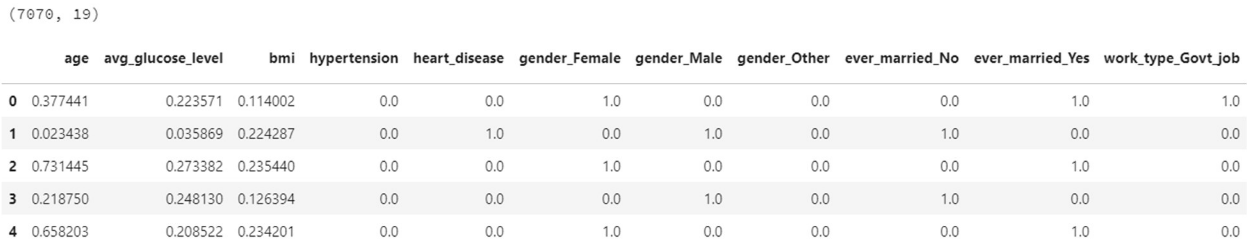
Before fitting the data to the LogisticRegression() classifier in sklearn, we must first scale continuous variables using MinMaxScaler() from sklearn. MinMaxScaler is robust to outliers (which do exist within our continuous features) and transforms the range of the data to within 0 and 1, which equalizes the variance with the one-hot encoded categorical variables.

Though logistic regression differs from linear regression in that the continuous features do not need to have a normal distribution, scaling is still important in order to equalize the feature importances to the output.

We will fit the scaler to the X_train_over data, then transform both X_train_over and X_test before fitting our model.

Note: Scaling is not required for RandomForestClassifier(), but doing so will not significantly affect the model performance one way or another. It only has an effect on the interpretation of the model trees, as features will now be split based on scaled values rather than values that make sense within each feature variable.

A snippet of the dataframe after scaling is presented below:



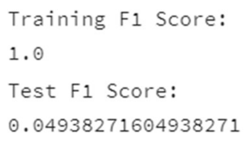
```
(7070, 19)
```

	age	avg_glucose_level	bmi	hypertension	heart_disease	gender_Female	gender_Male	gender_Other	ever_married_No	ever_married_Yes	work_type_Govt_job
0	0.377441	0.223571	0.114002	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0
1	0.023438	0.035869	0.224287	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0
2	0.731445	0.273382	0.235440	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
3	0.218750	0.248130	0.126394	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0
4	0.658203	0.208522	0.234201	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0

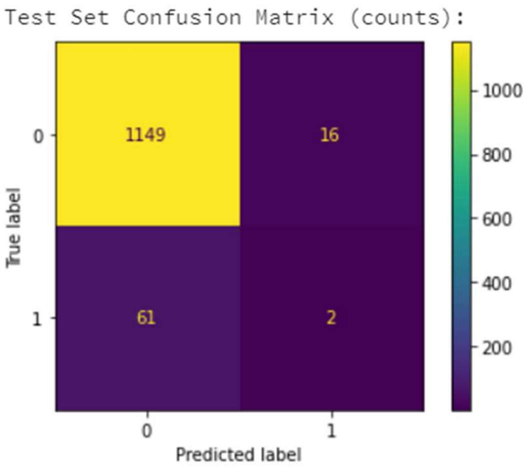
Random Forest

There are a number of hyperparameters that can be tuned when using a Random Forest classifier.

Because this stage of the analysis is simply to fit the data to each classifier for the purposes of comparison, the default parameters were chosen. Below are the training and test set F1 scores as well as the test set confusion matrix resultant of this model.



```
Training F1 Score:
1.0
Test F1 Score:
0.04938271604938271
```



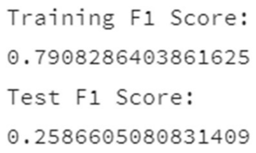
Test Set Confusion Matrix (counts):

Based on the scores for both the training and test sets, we can immediately see that overfitting is occurring.

Logistic Regression

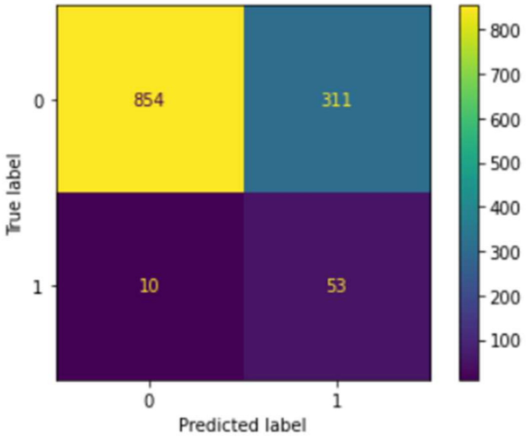
Similarly to the Random Forest model, default hyperparameters were chosen for the LogisticRegression() method in sklearn.

Below are the training and test set F1 scores as well as the test set confusion matrix resultant of this model.



```
Training F1 Score:
0.7908286403861625
Test F1 Score:
0.2586605080831409
```

```
Test Set Confusion Matrix (counts):
```



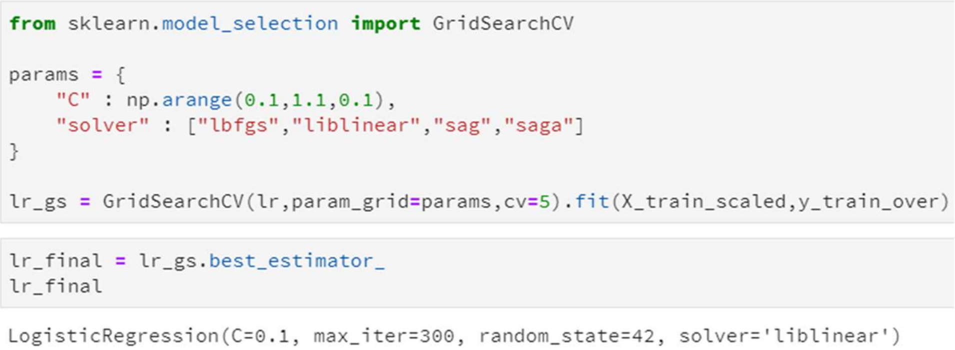
As was the case with Random Forest, the F1 scores for Logistic Regression indicate that overfitting is occurring with this dataset. However, the F1 score and confusion matrix for logistic regression give much more favorable results on the test set. This indicates that Logistic Regression might be a more suitable model to use for classification of our specific dataset.

We will continue our analysis using this model.

Model Evaluation

We can tune the hyperparameters of the Logistic Regression model to see if there is any improvement when it is applied to the test data. We will do this by utilizing GridSearchCV() within sklearn.

There isn't much to tune as far as hyperparameters go for Logistic Regression; for this analysis we will explore different values of C as well as different algorithmic solvers. The "C" parameter is the inverse of the regularization term, and controls the shrinkage of feature coefficients for the specified penalty ("L1" in this case).

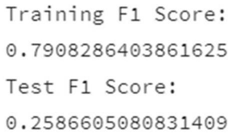


```
from sklearn.model_selection import GridSearchCV

params = {
    "C" : np.arange(0.1,1.1,0.1),
    "solver" : ["lbfgs","liblinear","sag","saga"]
}

lr_gs = GridSearchCV(lr,param_grid=params,cv=5).fit(X_train_scaled,y_train_over)
```

```
lr_final = lr_gs.best_estimator_
lr_final
```

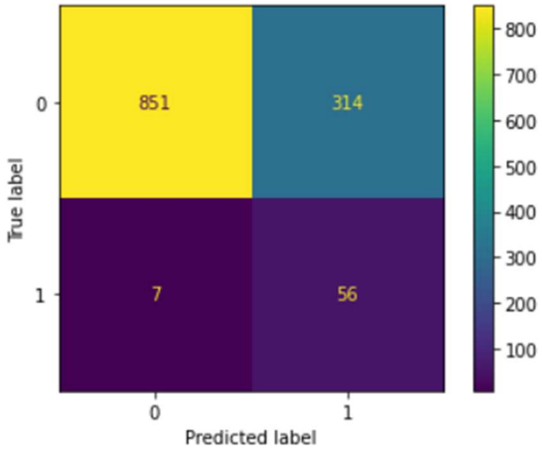
```
LogisticRegression(C=0.1, max_iter=300, random_state=42, solver='liblinear')
```

The above code indicates that the “liblinear” solver with C=0.1 gives the best overall results for all combinations of the hyperparameters that were tuned. The resulting training and test set F1 scores as well as the test set confusion matrix using this best estimator is shown below:



```
Training F1 Score:
0.7908286403861625
Test F1 Score:
0.2586605080831409
```

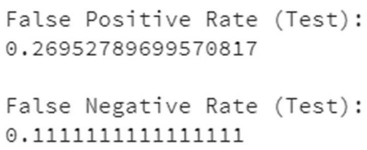
```
Test Set Confusion Matrix (counts):
```



Conclusion

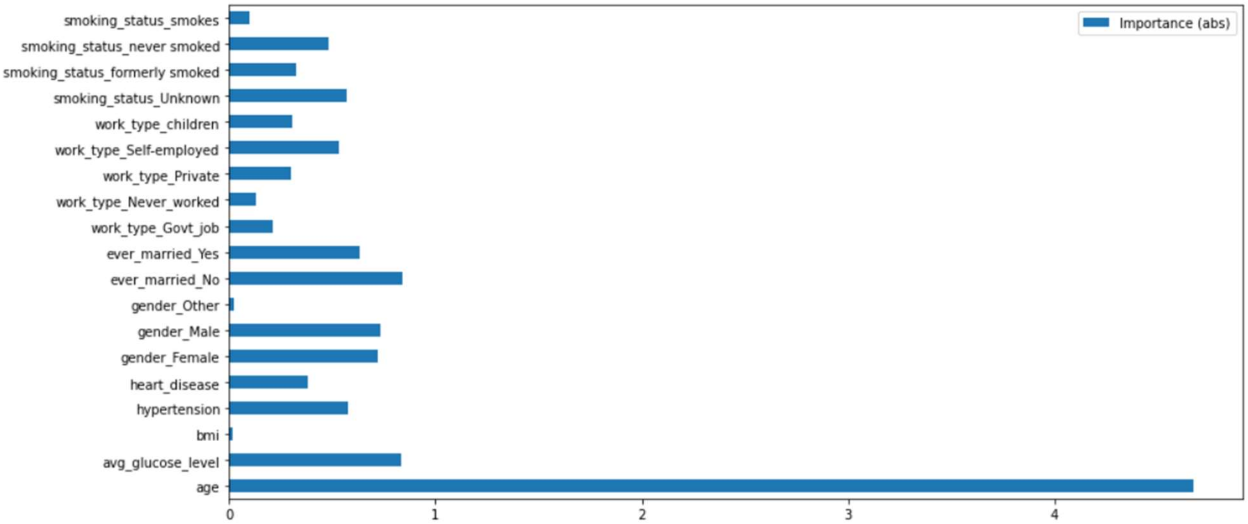
Hyperparameter tuning did increase the F1 score for both the training and test sets, somewhat, but the discrepancy between the two remains significant, and overfitting still seems to be an issue. This can be explained by the imbalanced class distribution within the test set.

Though the F1 score for the test set is indeed low, we can see from the confusion matrix that the final model correctly predicted 56 of 64 positive stroke patients based on the features, and correctly predicted 851 of 1165 negative stroke patients. Typically, when dealing with medical data, false negatives are much costlier than false positives; our model has a much lower false negative rate than a false positive rate, so the model did an acceptable job classifying stroke patients despite the imbalanced dataset.



```
False Positive Rate (Test):
0.26952789699570817

False Negative Rate (Test):
0.1111111111111111
```

We can now extract the feature importances of our model and display them in a bar chart.



From the plot, a number of observations can be concluded.

- Interestingly, bmi had very little effect on predicting strokes (according to the logistic regression model we fit)
- gender_other also, as expected, had little effect on the model, being that it was only represented in a single case in the original data.
- We can see from the coefficients that avg_glucose_level, gender, hypertension, and marriage status were most influential for our classification algorithm.

Further Analysis

Further analysis would most certainly include exploration of other classification algorithms, such as K-Nearest Neighbors (KNN), Support Vector Classifiers (SVC), and Naive Bayes implementations.

In addition, models could be rerun after a primary analysis, with unimportant features and attribute values (identified above) dropped from the dataset. This would ideally reduce overfitting whilst simultaneously increasing the F1 score on the test dataset.