Technical Report:	**Stroke Predictor**
Authors:	**David Cottrell (18152465)**
Module Code:	**CMP6202**
Coordinator:	**Hossein Ghomeshi**
Date:	**06/12/21**
Word Count:	**3026**
Page Count:	**20**

Contents

Abstract

This project explored a dataset containing the health information of individuals who did and did not experience a Stroke – using data visualisation techniques, to find correlations between features, with the final goal of training a machine learning model to predict future cases of Strokes.

Introduction

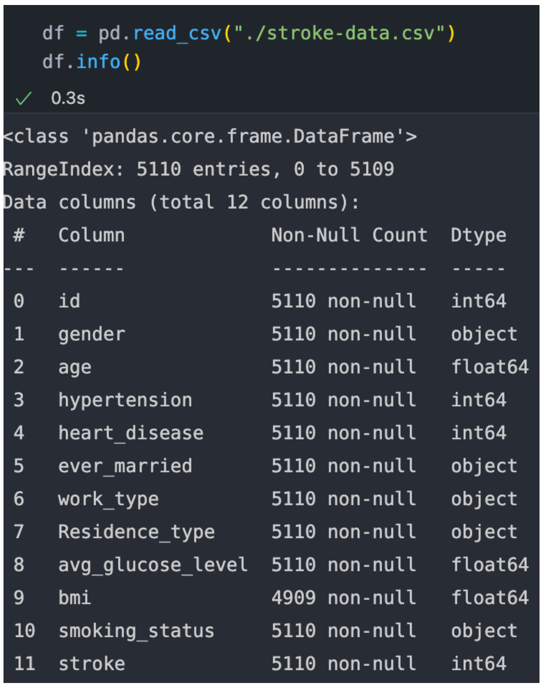
Stroke is a life-threatening condition caused when the brain either stops receiving the required amount of blood or if bleeding occurs within the brain itself. Machine learning is becoming an increasingly popular tool within healthcare, as it can "uncover new possibilities for researchers, physicians, and patients, allowing them to make more informed decisions and achieve better outcomes" (Cutillo, C.M., Sharma, K.R., Foschini, L. et al. 2020). In this case, a model will be developed that could help predict whether a patient is likely to experience stroke in the future – allowing them to take the necessary steps to help avoid it. It could also be used to help identify the cause of death if that patient had already died of stroke.

Dataset Overview

The dataset chosen for this project is the Stroke Prediction Dataset published by Federico Palacios (2021) on Kaggle. This dataset contains health information for 5110 individuals, 249 of which suffered from a stroke and 4861 that didn't. There are 10 features for each individual relating to a possible factor that could increase the likelihood of a Stroke, including gender, age, hypertension and BMI, along with the individual's "id" and target feature "stroke" (figure 1).

The source of this dataset is confidential as it contains sensitive, personal information collected from real people and was collected for the purpose of building a stroke prediction model.

The features selected for the dataset are supported by medical resources, such as the NHS who state some likely causes of stroke are "being overweight… smoking… high blood pressure (hypertension)… diabetes… age" (2019).



```
df = pd.read_csv("./stroke-data.csv")
df.info()
✓ 0.3s
<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
```

Figure 1: Dataset Overview

Although the features “ever_married”, “work_type” and “residence_type” seem less obviously linked to stroke, they do have evidence to be linked to the likelihood of stroke, as if the individual lives in an area with high air pollution for example, “air pollution is thought to increase the risk of stroke by hardening arteries in the brain” (Sample, I. 2016). “ever_married” and “work_type” may be also be linked to the amount of stress an individual experiences, which also increases the chance of experiencing a stroke.

> “Stress can cause the heart to work harder, increase blood pressure, and increase sugar and fat levels in the blood. These things, in turn, can increase the risk of clots forming and travelling to the heart or brain, causing a heart attack or stroke” (Heart&Stroke, n.d.).

Target Problem

Using the dataset identified in the previous section, this project will aim to develop a machine learning model, to predict whether or not an individual will experience a stroke, based upon their health factors as shown in the dataset.

Existing methods for predicating the health of an individual, based on some variables linked to the patient, include the Cox Proportional-hazards Model (Cox, 1972). This is a regression algorithm used to find the “probability a certain event (e.g. death) happens at a particular time” (Glen, S. 2018), however more modern machine learning techniques and algorithms have “demonstrated superior predictive value” (Chun *et al.*, 2021) than that of more traditional methods such as the Cox Model. Therefore, this project will aim to build a stroke classification model that potentially exceeds the accuracy of traditional prediction methods.

Model Development

Approach

Throughout this project I followed a generic data science project workflow (figure 2), which starts with exploring and manipulating the data to find insights and correlations between features in the Data Preparation phase, as well as ensuring the data is split into training and testing sets and is ready to be processed within the Model Training phase. This next stage will compare the effectiveness of different machine learning algorithms applied to the prepared data with the goal of choosing the most appropriate one with the highest performing accuracy across testing data. Finally, the model will be evaluated and optimised using the methods appropriate to the chosen model.

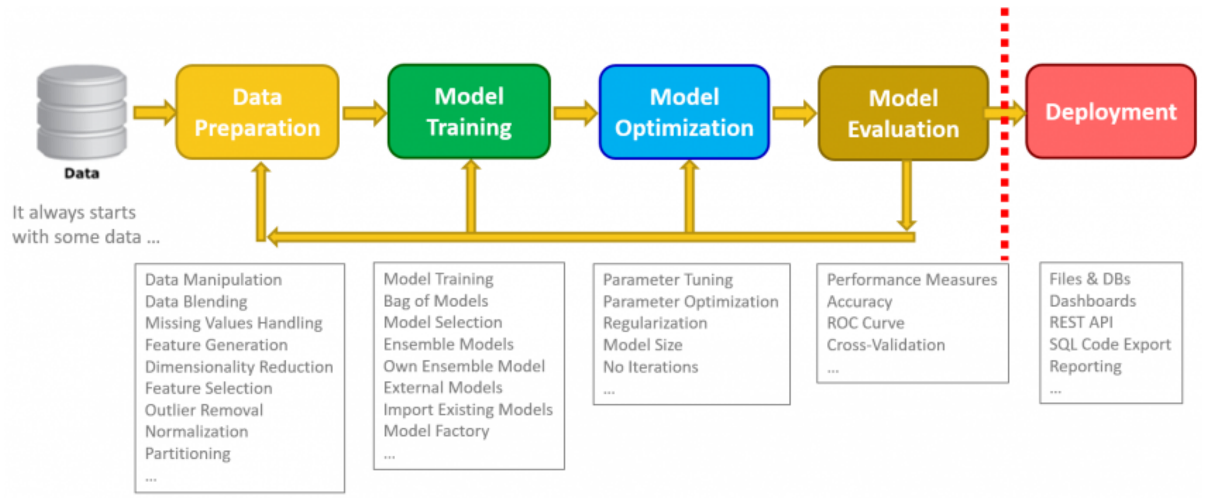
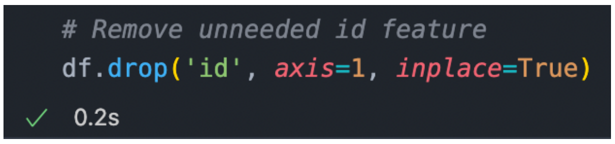


Figure 2: Data science project cycle (Silipq, R. 2017)

Data Preparation

As shown in Figure 1 previously, some obvious first steps can be taken to begin preparing the data. The "id" feature can be removed as it has no significance in this project (figure 3) and there are 201 cases where "bmi" is null which will need to be addressed.

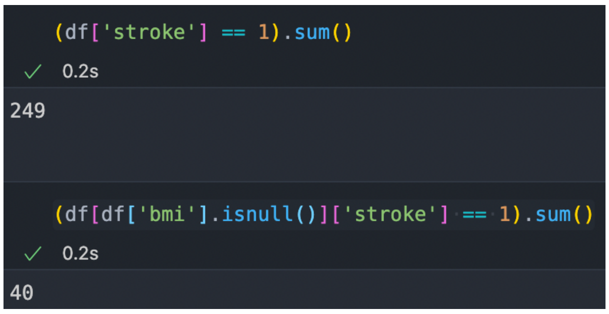
When working with missing data, the entries that include a piece of missing data can either be removed, or an estimation of the most likely value can be added manually. This is a process known as Imputation.



```
# Remove unneeded id feature
df.drop('id', axis=1, inplace=True)
✓ 0.2s
```

Figure 3: Removing feature "id"

As stated by MastersInDataScience (n.d.), imputation is "most useful when the percentage of missing data is low". Therefore, imputation will be used as the missing BMI entries in this dataset make up under 4% of the total entries in the dataset, and the entries where the BMI is missing account for 40 out of the 249 cases of stroke (figure 4), meaning simply removing these entries would make the dataset even more unbalanced.



```
(df['stroke'] == 1).sum()
✓ 0.2s
249

(df[df['bmi'].isnull()]['stroke'] == 1).sum()
✓ 0.2s
40
```

Figure 4: Cases of missing bmi with stroke

This dataset contains a large number of outliers for the "avg_glucose_level" and "bmi" features (figure 5). As BMI and glucose levels can vary greatly in the real world, these are not invalid or impossible values so will not be removed.

The decision to keep the outliers is also supported by Dave Parth in his exploration of the same problem in which he states, "people having stroke have an average glucose level of more than 100. There are some obvious outliers in patients who have no stroke but there are some chances of this being genuine records" (2021).

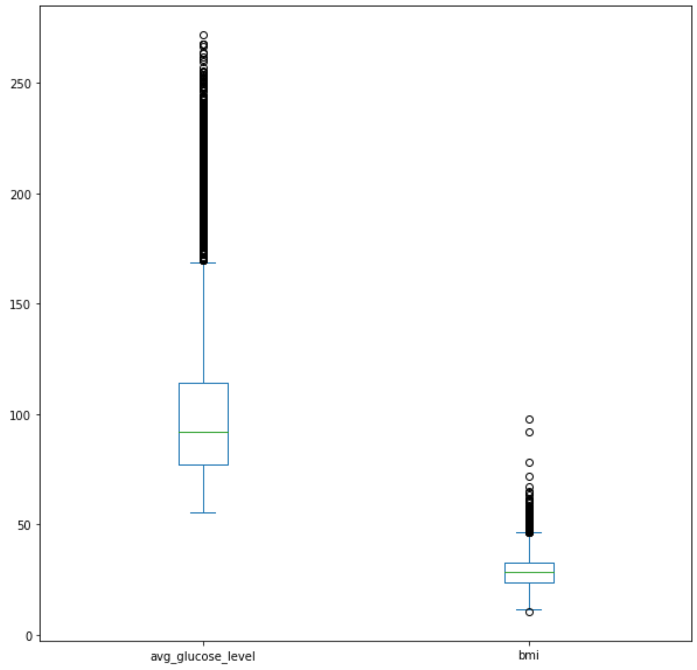


Figure 5: Outliers

To determine the influence each of the three continuous features has on the likelihood of a stroke, they were plotted against the target feature using a Kernel Density Estimation plot (figures 6,7 and 8). The areas coloured in orange show the range in which the highest number of stroke occurred against the density of the other two variables. This provides a clear view of the correlation between stroke and the other variables.

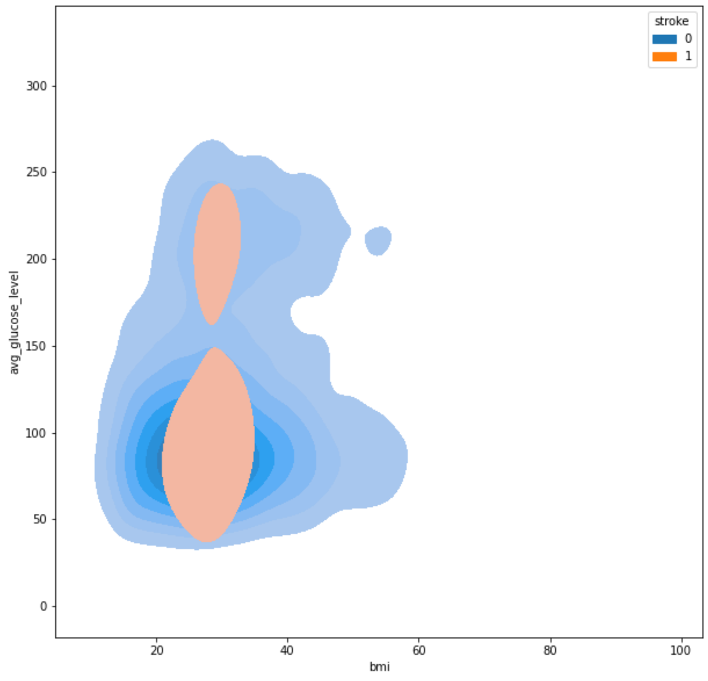


Figure 6: Glucose Level and BMI

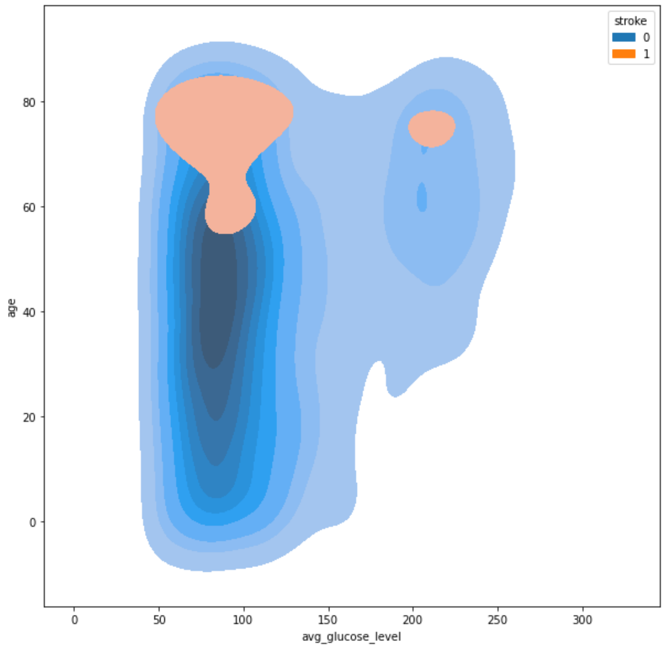


Figure 7: Age and Glucose Level

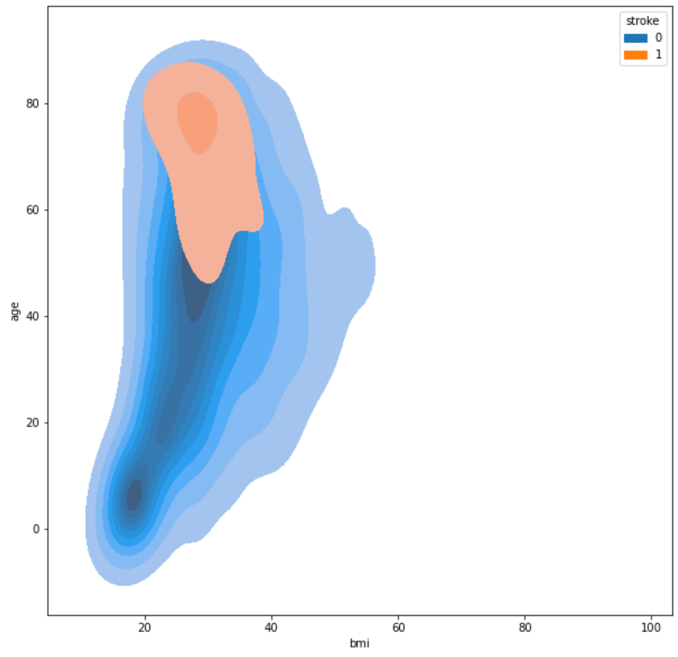


Figure 8: Age and BMI

These show that age is by far the most influential factor towards stroke out of the three analysed variables. As demonstrated in figure 6, the BMI appears to have a lesser role in the likelihood of a stroke than the average glucose level.

Figure 9 shows there is not a strong correlation between gender and stroke. There are slightly more cases of stroke within females, however this is possibly due to the dataset being imbalanced – having more female entries than male.

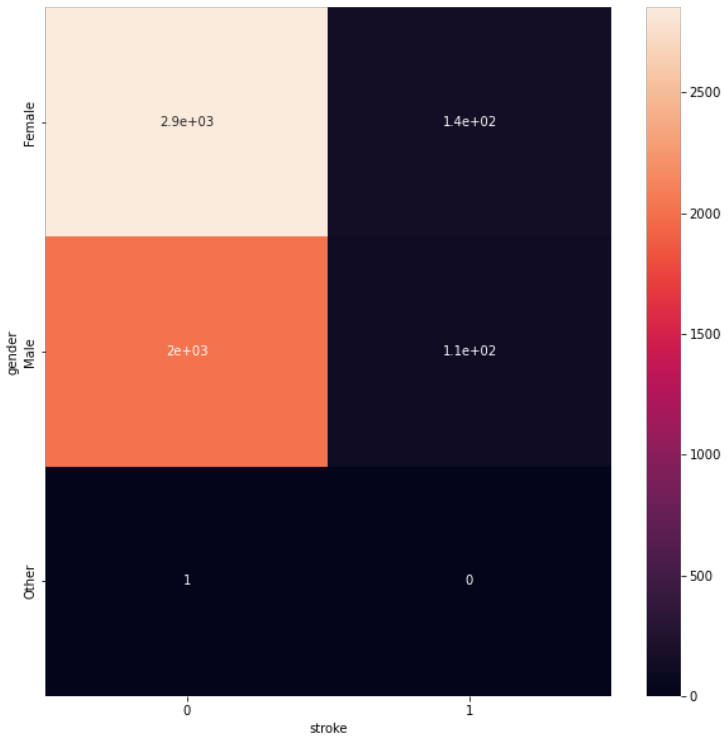


Figure 9: Gender and Stroke

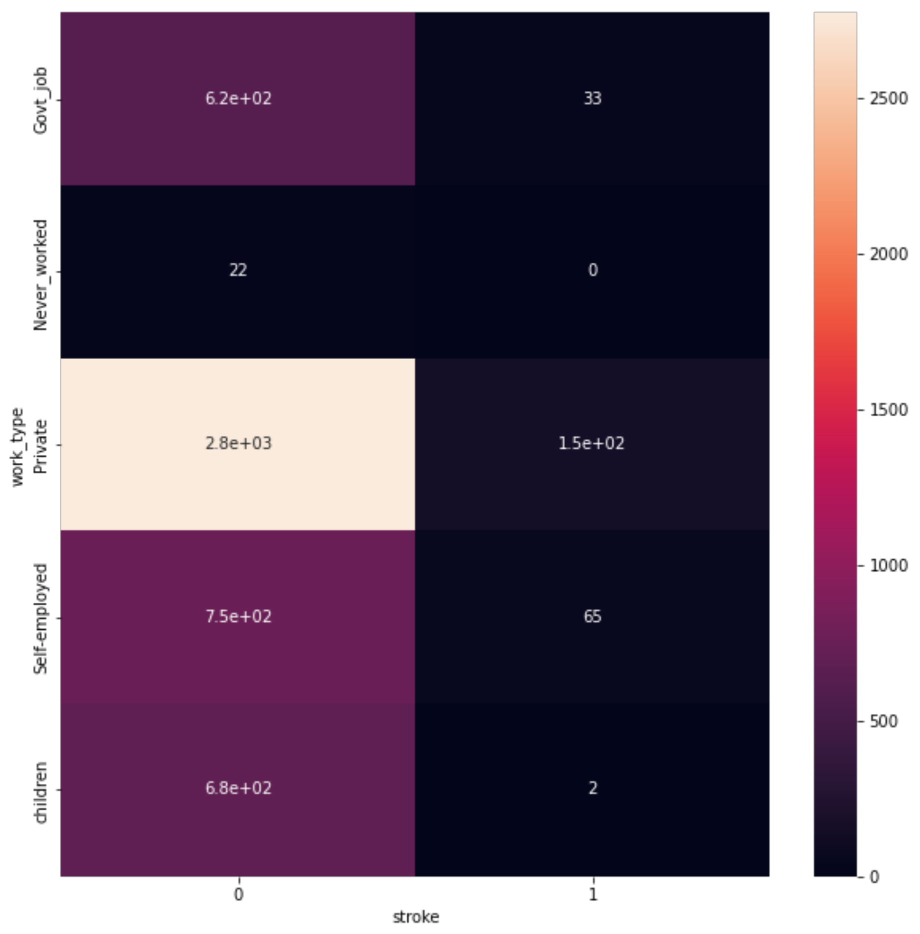
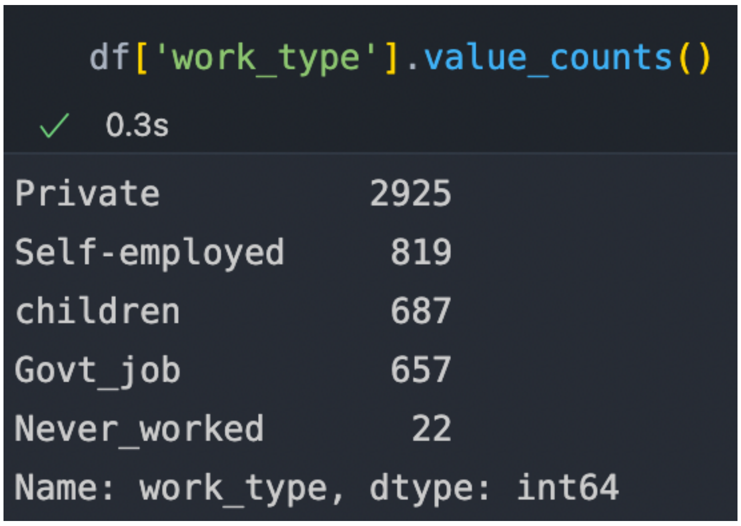


Figure 10: Stroke and Work Type

Figure 10 shows the “Private” work type has the most cases of stroke within the dataset. Similarly to gender however, “Private” is also by far the most common work type recorded (figure 11), meaning the difference in the amount of stroke within work types is possibly due to over representation in the data set.



```
df['work_type'].value_counts()
✓ 0.3s
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
```

Figure 11: Work Type Catagory Values

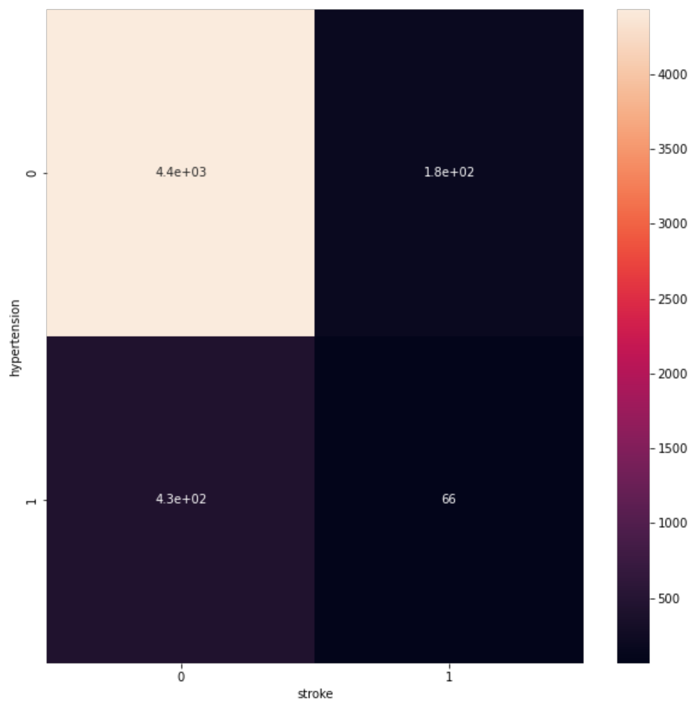


Figure 12: Hypertension and Stroke

Contrary to existing medical research, figure 12 shows of the cases where people had stroke in this dataset, very few also had hypertension. This could be due to the fact that only 498, cases of people in this dataset had hypertension in total.

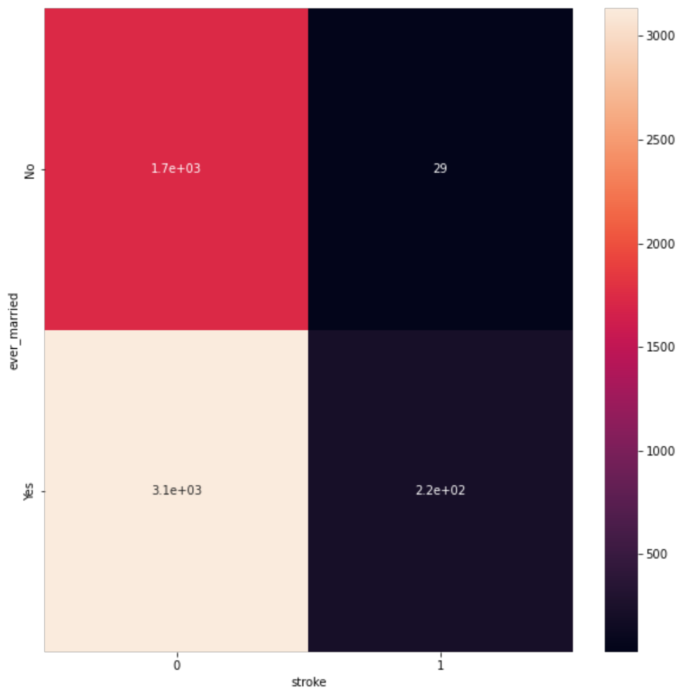


Figure 13: Ever Married and Stroke

Although there are more cases of people who have been married than never in the dataset, figure 13 shows a significant enough correlation between being married and stroke that is likely an important feature to include.

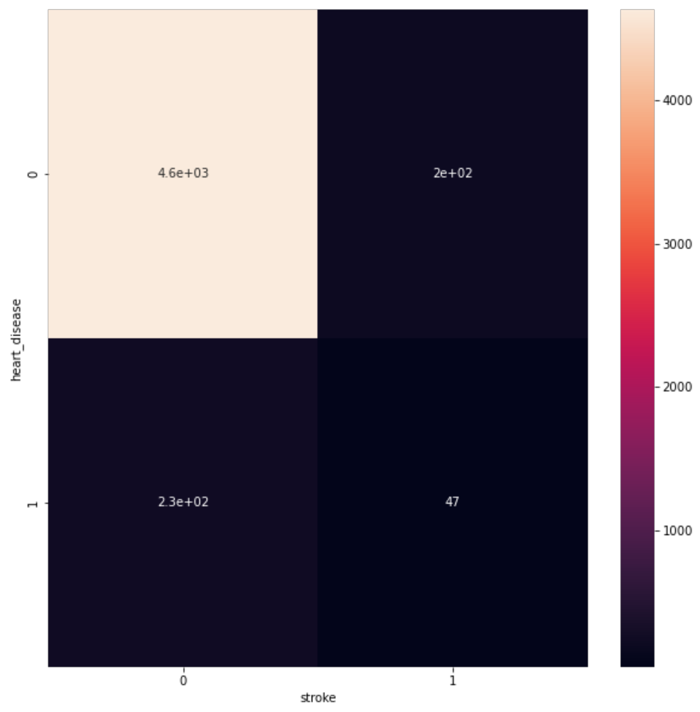


Figure 14: Heart Disease and Stroke

Heart disease does not seem to have a great influence over brain stroke, although this feature contains vastly more entries for people without heart disease than with, so no concrete conclusion can be made.

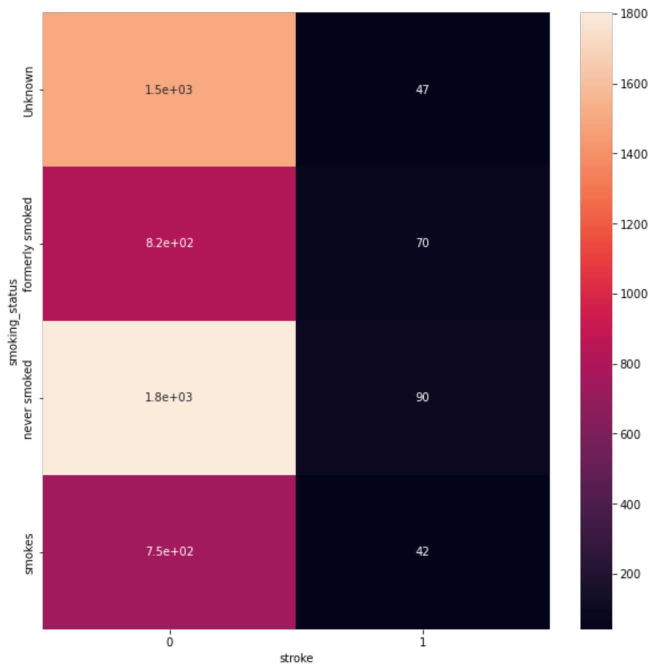
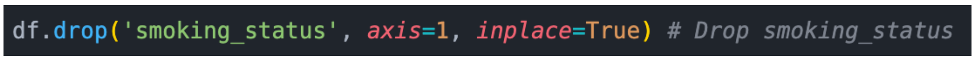


Figure 15: smoking status and stroke

Figure 15 shows the most people who experienced stroke in this dataset never smoked, despite smoking being proven to lead to stroke. This feature will be removed when training as the model will believe never smoking will increase the chance of stroke.

To remove the “smoking_status” feature, the code snippet shown in figure 16 below will be used.



```
df.drop('smoking_status', axis=1, inplace=True) # Drop smoking_status
```

Figure 16: Drop smoking_status

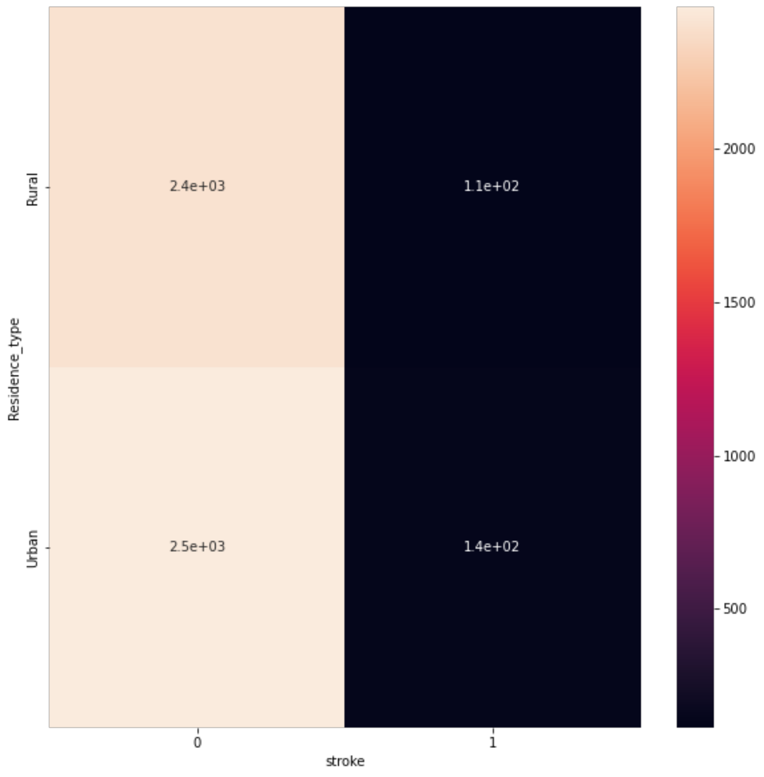


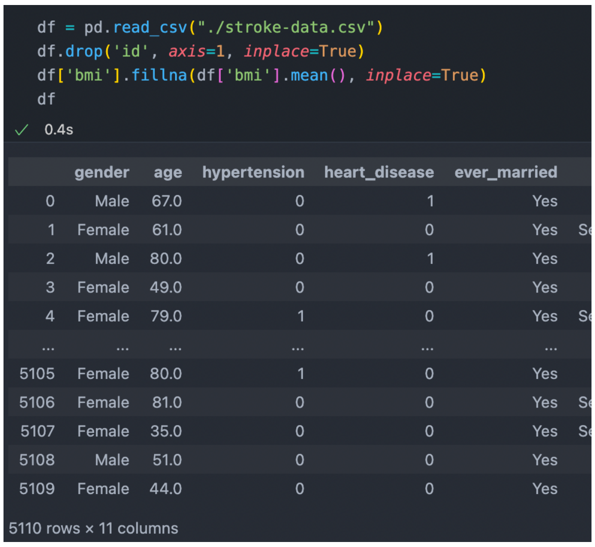
Figure 17: Residence Type and Stroke

Figure 17 shows a small increase in the cases of stroke in urban areas, supporting the previously discussed research.

Machine Learning algorithms can only process numeric values, and as this dataset contains both categorical and continuous features, a technique known as Category Encoding will be used to ensure every cell within the dataset contains a numeric value. Currently, this dataset contains strings for the features “ever_married”, “gender”, “work_type” and “Residence_type”. Two common types of Category Encoding techniques are One Hot Encoding and Label Encoding, in this example, One Hot Encoding will be used. This is because the data to be encoded is nominal and will prevent the chosen machine learning algorithm from perceiving some order or relationship between categories which can result from using Label Encoding – resulting in a less accurate model.

> “Depending upon the data values and type of data, label encoding induces a new problem since it uses number sequencing. The problem using the number is that they introduce relation/comparison between them” (Yadav, 2019).

Figures 18 and 19 show the dataset before and after One Hot Encoding.



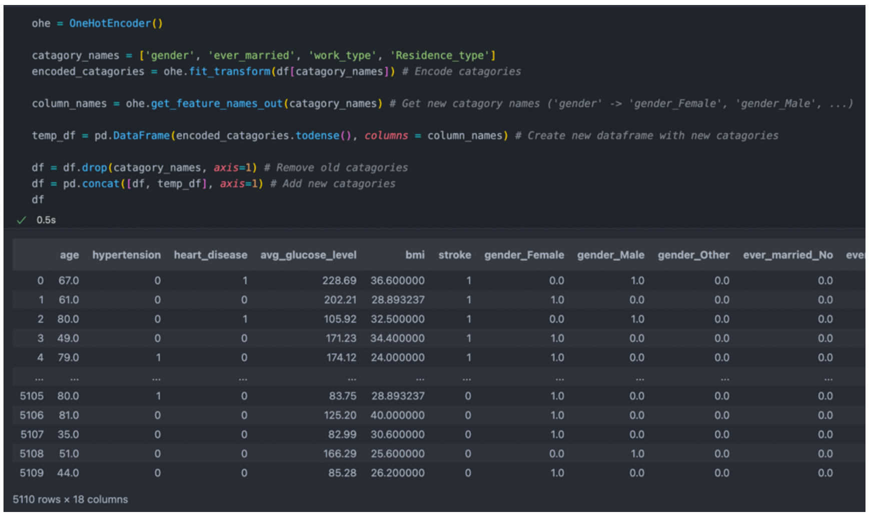
```python
df = pd.read_csv("./stroke-data.csv")
df.drop('id', axis=1, inplace=True)
df['bmi'].fillna(df['bmi'].mean(), inplace=True)
df
```

✓ 0.4s

	gender	age	hypertension	heart_disease	ever_married	
0	Male	67.0	0	1	Yes	
1	Female	61.0	0	0	Yes	Se
2	Male	80.0	0	1	Yes	
3	Female	49.0	0	0	Yes	
4	Female	79.0	1	0	Yes	Se
...	...	...	...	...	...	
5105	Female	80.0	1	0	Yes	
5106	Female	81.0	0	0	Yes	Se
5107	Female	35.0	0	0	Yes	Se
5108	Male	51.0	0	0	Yes	
5109	Female	44.0	0	0	Yes	

5110 rows × 11 columns

Figure 19: Before OHE



```python
ohe = OneHotEncoder()

catagory_names = ['gender', 'ever_married', 'work_type', 'Residence_type']
encoded_catagories = ohe.fit_transform(df[catagory_names]) # Encode catagories

column_names = ohe.get_feature_names_out(catagory_names) # Get new catagory names ('gender' -> 'gender_Female', 'gender_Male', ...)

temp_df = pd.DataFrame(encoded_catagories.todense(), columns = column_names) # Create new dataframe with new catagories

df = df.drop(catagory_names, axis=1) # Remove old catagories
df = pd.concat([df, temp_df], axis=1) # Add new catagories
df
```

✓ 0.5s

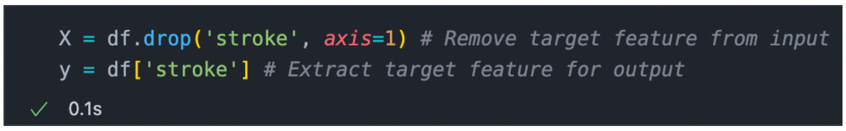
	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Female	gender_Male	gender_Other	ever_married_No	eve
0	67.0	0	1	228.69	36.600000	1	0.0	1.0	0.0	0.0	
1	61.0	0	0	202.21	28.893237	1	1.0	0.0	0.0	0.0	
2	80.0	0	1	105.92	32.500000	1	0.0	1.0	0.0	0.0	
3	49.0	0	0	171.23	34.400000	1	1.0	0.0	0.0	0.0	
4	79.0	1	0	174.12	24.000000	1	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	
5105	80.0	1	0	83.75	28.893237	0	1.0	0.0	0.0	0.0	
5106	81.0	0	0	125.20	40.000000	0	1.0	0.0	0.0	0.0	
5107	35.0	0	0	82.99	30.600000	0	1.0	0.0	0.0	0.0	
5108	51.0	0	0	166.29	25.600000	0	0.0	1.0	0.0	0.0	
5109	44.0	0	0	85.28	26.200000	0	1.0	0.0	0.0	0.0	

5110 rows × 18 columns

Figure 18: After OHE

As shown in the figures above, One Hot Encoding introduces the problem of increased dimensionality within the dataset, as each possible value a category could take, is given its own column. In larger datasets this can cause issues, where Principal Component Analysis (PCA) should be used to reduce the dimensionality post Category Encoding. PCA is an unsupervised machine learning algorithm that finds patterns within a dataset called Principal Components, "Principal components are orthogonal projections (perpendicular) of data onto lower-dimensional space" (Biswal, A. 2021). In this dataset however, it has only added 7 columns and will be unlikely to cause issues or performance gains when training the models.

As this is a supervised learning task, data should be split into input and output sets. The input being every feature other than the target feature "stroke" and will be used to "train" the model in which are the positive cases of stroke, and which are not.

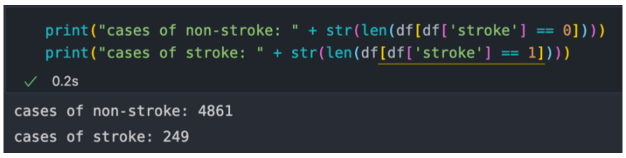


```python
X = df.drop('stroke', axis=1) # Remove target feature from input
y = df['stroke'] # Extract target feature for output
```

✓ 0.1s

Figure 20: Input/Output sets

As the dataset is so imbalanced – having only 249 cases of stroke compared to 4861 of non-stroke (figure 21), the model will likely fail to classify positive cases of stroke. To remedy this, Synthetic Minority Oversampling Technique (SMOTE) can be used to create new artificial instances of under-represented cases within the dataset. This aims to improve a model's ability to successfully classify all possible outcomes. In this case, SMOTE will be used to increase the amount of stroke records within the dataset. As this dataset contain a mixture of categorical and continuous data, SMOTENC will be used as this will generate both categorical and continuous data (figure 22).

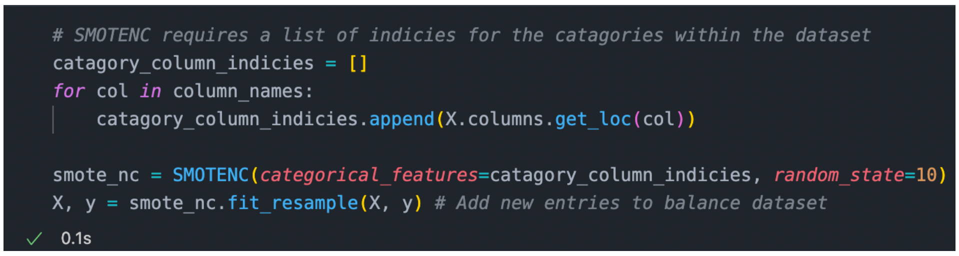


```python
print("cases of non-stroke: " + str(len(df[df['stroke'] == 0])))
print("cases of stroke: " + str(len(df[df['stroke'] == 1])))
```

✓ 0.2s

```
cases of non-stroke: 4861
cases of stroke: 249
```

Figure 21: Stroke imbalance



```
# SMOTENC requires a list of indicies for the catagories within the dataset
catagory_column_indicies = []
for col in column_names:
    catagory_column_indicies.append(X.columns.get_loc(col))

smote_nc = SMOTENC(categorical_features=catagory_column_indicies, random_state=10)
X, y = smote_nc.fit_resample(X, y) # Add new entries to balance dataset
```

✓ 0.1s

Figure 22: Adding minority cases with SMOTENC

The data should be split into training and test sets, where the test set is used to evaluate the model's performance on new, unrecognised data, the same way it would when deployed. The ratio in which the data is split depends on the amount of data available. In this case, a split of 80% training and 20% testing will be used as this is the most recommended ratio and will provide enough data to ensure the accuracy of the model is valid.

Gradient descent and distance-based machine learning algorithms perform optimally when the values for each feature are within similar ranges, "Gradient descent as an optimization technique require data to be scaled … Distance algorithms like KNN, K-means, and SVM are most affected by the range of features" (Bhandari, A. 2020), this can be achieved using a technique called Feature Scaling. Depending on the algorithm being used, either Normalisation or Standardisation should be used.

Normalisation will cause each value in the dataset to be between 0 and 1 and should be used with distance-based algorithms like KNN and "when you know that the distribution of your data does not follow a Gaussian distribution". Standardisation on the other hand uses the following formula to centre each value "around the mean with a unit standard deviation" (Bhandari, A. 2020) by subtracting each value from the mean and diving by the standard deviation:

$$z = \frac{x_i - \mu}{\sigma}$$

Feature scaling has little to no effect on tree-based models such as Decision Trees and Random Forest, as these algorithms learn by finding patterns within data – patterns that would remain unchanged through scaling.

Model Training

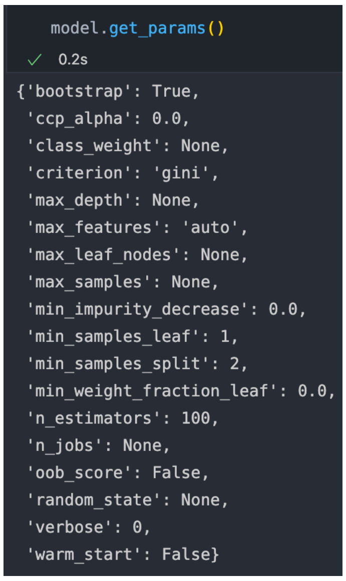
5 different models were trained and tested using a combination of standardisation and normalisation depending on the algorithm being used to find the highest performing model. As shown in figure 23, Random Forest is by far the most accurate model, although being the 2nd slowest to train.

Algorithm	Training Score	Test Score	Cross Validation Accuracy	Training Time
Random Forest (Without feature scalling)	0.9997	0.9357	Mean: 0.940 Std: 0.010	0.4450
SVM (With Standardisation)	0.8617	0.8467	Mean: 0.856 Std: 0.015	0.5348
KNN (With Normalisation)	0.9213	0.9059	Mean: 0.903 Std: 0.008	0.0014
Logistic Regression (With Standardisation)	0.8168	0.8097	Mean: 0.815 Std: 0.012	0.0314
Decision Tree (Without feature scalling)	1.0	0.9110	Mean: 0.912 Std: 0.012	0.0358

Figure 23: Model Performance Comparison

The Random Forest algorithm combines many decision trees to form a more accurate result than using a single tree. Decision trees continually make split decisions at each "node" of the tree based on some pre-defined condition until it can make a confident prediction at the final node. Random Forest takes "the average or mean of the output from various trees" (Mbaabu, O. 2020), thus increasing the accuracy of the final prediction.

Hyperparameters allow elements of the learning process for a machine learning algorithm to be customised, to best suit the specific training data with the aim of improving the model's accuracy. The hyperparameters for the Random Forest Classifier control the amount of trees built with the "n_estimators" parameter, the characteristics of each tree with parameters including "max_depth" and "min_sample_split" which control the maximum number of branches from the root node and the minimum number of required observations to split a node and "max_features" and "bootstrap" which control how the training data is used to build the trees.

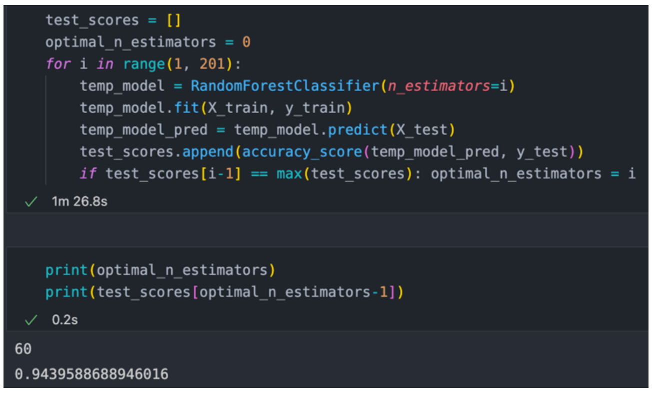


```
model.get_params()
✓ 0.2s
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

Figure 24: Default Hyperparameters

Currently, the metrics recorded in figure 23 were with the default parameters for that model (figure 24) and could potentially be improved if the ideal combination of hyperparameters could be identified.

To determine how the model reacts to different numbers of trees, models were trained using 1 to 200 n_estimators (figure 25) and their scores recorded and plotted (figure 26).



```
test_scores = []
optimal_n_estimators = 0
for i in range(1, 201):
    temp_model = RandomForestClassifier(n_estimators=i)
    temp_model.fit(X_train, y_train)
    temp_model_pred = temp_model.predict(X_test)
    test_scores.append(accuracy_score(temp_model_pred, y_test))
    if test_scores[i-1] == max(test_scores): optimal_n_estimators = i
```
```
✓ 1m 26.8s
```
```
print(optimal_n_estimators)
print(test_scores[optimal_n_estimators-1])
```
```
✓ 0.2s
60
0.9439588688946016
```

Figure 25: Finding Optimal Number of Trees

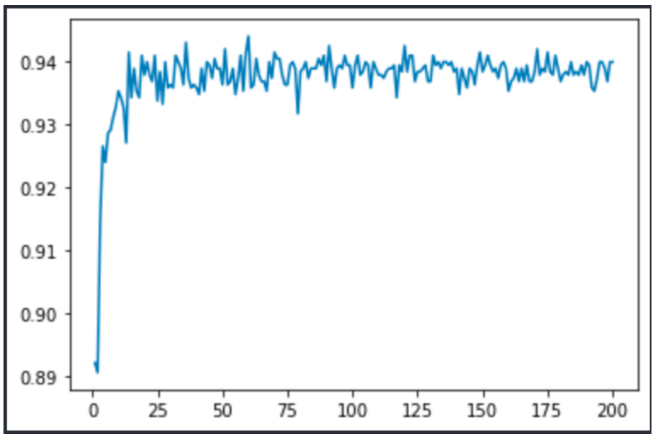
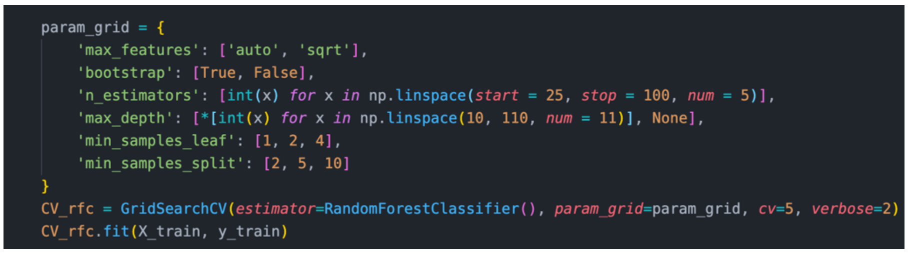


Figure 26: n_estimators scores

There is a clear stabilisation after approximately 25 trees. Intuitively, increasing the number of trees the algorithm uses should increase accuracy with the drawback of increased training and prediction times "higher number of trees give you better performance but makes your code slower" (Srivastava, T. 2015). As shown in figure 25, the number of trees used that produced the highest score was 60.

The optimal set of hyperparameters can be found using hyperparameter tuning algorithms, such as Random and Grid Search. Random Search trains a model for every combination of hyperparameters to find the highest performing combination, whereas Grid Search evaluates a trained model for each possible combination of a pre-defined list of hyperparameters and will be used in this example (figure 27).

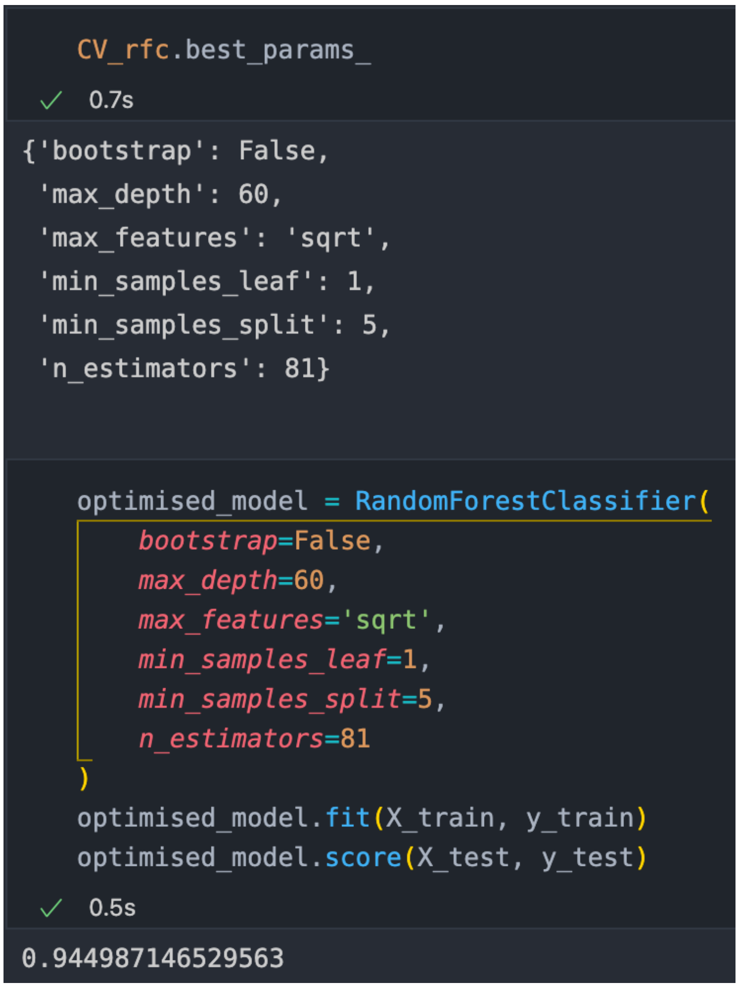


```
param_grid = {
    'max_features': ['auto', 'sqrt'],
    'bootstrap': [True, False],
    'n_estimators': [int(x) for x in np.linspace(start = 25, stop = 100, num = 5)],
    'max_depth': [*[int(x) for x in np.linspace(10, 110, num = 11)], None],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10]
}
CV_rfc = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_grid, cv=5, verbose=2)
CV_rfc.fit(X_train, y_train)
```

Figure 27: GridSearchCV

As figure 26 shows, the optimal number of n_estimators is likely between 25 and 100, the parameter grid was given a list of numbers within this range. Other parameters within the grid are possible values the hyperparameters could have. Figure 28 shows the result of the Grid Search algorithm based on the provided parameters in figure 27.

This shows an improvement of roughly 0.00103 over the previous best result shown in figure 25, meaning the random forest algorithm is unlikely to improve much beyond 94% accuracy.



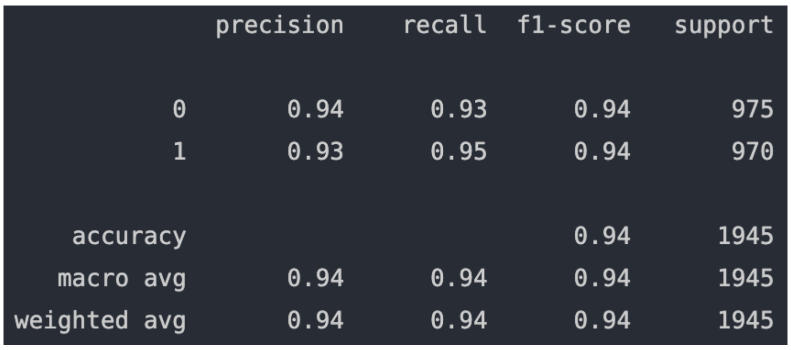
```
CV_rfc.best_params_
```
✓ 0.7s
```
{'bootstrap': False,
 'max_depth': 60,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 81}
```

```
optimised_model = RandomForestClassifier(
    bootstrap=False,
    max_depth=60,
    max_features='sqrt',
    min_samples_leaf=1,
    min_samples_split=5,
    n_estimators=81
)
optimised_model.fit(X_train, y_train)
optimised_model.score(X_test, y_test)
```
✓ 0.5s
```
0.944987146529563
```

Figure 28: GridSearchCV Result

Model Evaluation

Figure 29 shows the classification report for the final optimised Random Forest model.



```
              precision    recall  f1-score   support

           0       0.94      0.93      0.94       975
           1       0.93      0.95      0.94       970

    accuracy                           0.94      1945
   macro avg       0.94      0.94      0.94      1945
weighted avg       0.94      0.94      0.94      1945
```

Figure 29: Random Forest Classification Report

This shows that 94% of non-stroke cases and 93% of positive cases were classified correctly and the below confusion matrix shows a more detailed view of the model's predictions.

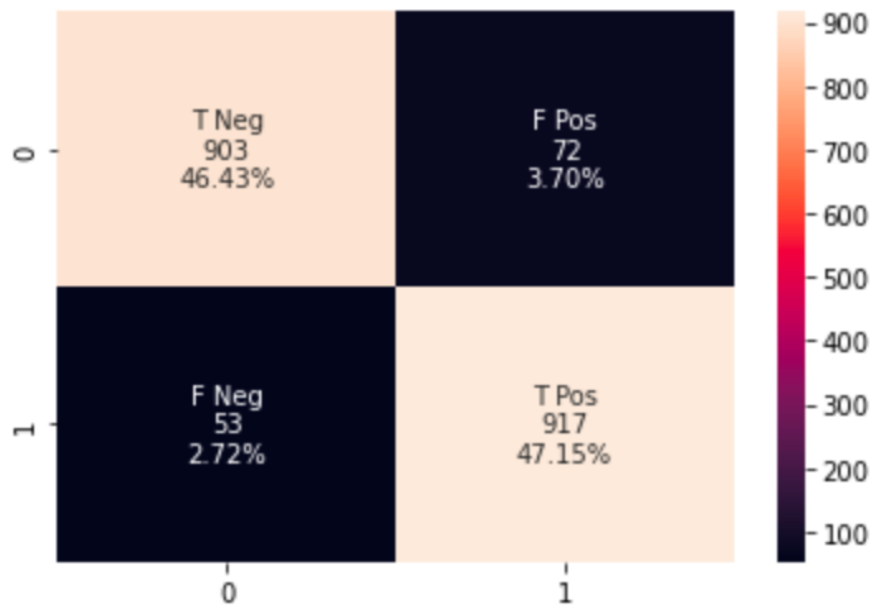
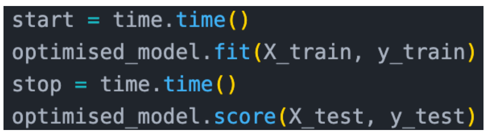


Figure 30: Random Forest Confusion Matrix

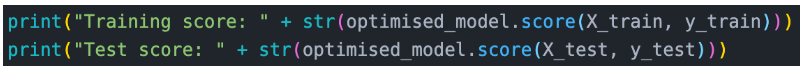
The total time taken to train the model was calculated using the code shown in figure 31 and has increased from 0.4450s to 0.5134s with the optimisations in accuracy.



```
start = time.time()
optimised_model.fit(X_train, y_train)
stop = time.time()
optimised_model.score(X_test, y_test)
```

Figure 31: Measuring training time

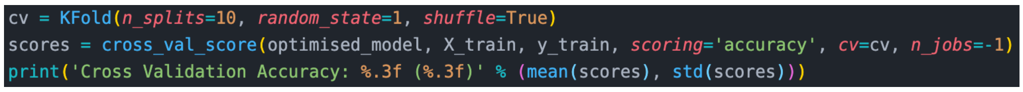
The test and train scores were calculated using the code snippet shown in figure 32.



```
print("Training score: " + str(optimised_model.score(X_train, y_train)))
print("Test score: " + str(optimised_model.score(X_test, y_test)))
```

Figure 32: Scoring Models

K-Fold Cross-Validation was also used to measure each model's performance. The training data was randomly organised and split into 10 sets (in this example), the model is then trained on the data and compared to each subsequent random set, as shown in figure 33.



```
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(optimised_model, X_train, y_train, scoring='accuracy', cv=cv, n_jobs=-1)
print('Cross Validation Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Figure 33: Model Cross-Validation

Conclusion

Given that the repercussions of this model failing to predict a patient's potential stroke are so serious, this model's 94% accuracy is not likely to be of the standard that would be necessary for use in production.

If the dataset contained more instances of actual stroke, more confidence could be put into the validity of the accuracy of the model, as currently 4,612 of the 4,861 cases of stroke, the model was trained on were generated artificially through SMOTE.

Some features within the dataset have low correlation with stroke, either because of an imbalance or the feature having no real-world linkage to stroke. To improve the accuracy of the model, health factors that contribute more towards stroke could be added.

Tuning the model's hyperparameters showed little effect on the model's accuracy, which indicates further work should go into exploring the correlations between features within the dataset and/or investigating other learning algorithms.

References

Bhandari, A (2020) *Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization*. Available at: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/ (Accessed: 19 November 2021).

Biswal, A (2021) *PCA In Machine Learning - Your Complete Guide To Principal Component Analysis*. Available at: https://www.simplilearn.com/tutorials/machine-learning-tutorial/principal-component-analysis (Accessed: 17 November 2021).

Chun *et al* (2021) *Stroke risk prediction using machine learning: a prospective cohort study of 0.5 million Chinese adults*. Available at: https://academic.oup.com/jamia/article/28/8/1719/6272889 (Accessed: 10 November 2021).

Cutillo, C.M., Sharma, K.R., Foschini, L. *et al.* (2020) *Machine intelligence in healthcare—perspectives on trustworthiness, explainability, usability, and transparency.* Available at: https://doi.org/10.1038/s41746-020-0254-2 (Accessed: 17 November 2021).

Dave, P (2021) *How to create a Stroke Prediction Model?*. Available at: https://www.analyticsvidhya.com/blog/2021/05/how-to-create-a-stroke-prediction-model/ (Accessed: 11 November 2021).

Glen, S (2018) *Cox Regression / Cox Model: Simple Definition*. Available at: https://www.statisticshowto.com/cox-regression-model/ (Accessed: 9 November 2021).

Heart&Stroke (n.d.) *Stress basics*. Available at: https://www.heartandstroke.ca/healthy-living/reduce-stress/stress-basics (Accessed: 10 November 2021).

Lee, K. Miller, M. Shah, A (2018) *Air Pollution and Stroke*. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5836577/ (Accessed: 8 November 2021).

MastersInDataScience (n.d.) *How to Deal with Missing Data*. Available at: https://www.mastersindatascience.org/learning/how-to-deal-with-missing-data/ (Accessed: 12 November 2021).

Mbaabu, O (2020) *Introduction to Random Forest in Machine Learning*. Available at: https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/ (Accessed: 02 December 2021).

Sample, I (2016) *Air pollution now major contributor to stroke, global study finds*. Available at: https://www.theguardian.com/science/2016/jun/09/air-pollution-now-major-contributor-to-stroke (Accessed: 17 November 2021).

Silipq, R (2017) *Analytics and Beyond!*. Available at: https://www.knime.com/blog/analytics-and-beyond (Accessed: 10 November 2021).

Srivastava, T (2015) *Why to tune Machine Learning Algorithms?*. Available at: https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/ (Accessed: 03 December 2021).

Yadav, D (2019) *Categorical encoding using Label-Encoding and One-Hot-Encoder*. Available at: https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd (Accessed: 17 November 2021).