BIRMINGHAM CITY
University

CMP6202
AI and Machine Learning Project
2022–2023

Individual Report

Stroke Predication with the use of Supervised Machine Learning Algorithms

Computer & Data Science
Dharmarlou Bowen
20123076

Table of Contents

1 Report Introduction

The purpose of this report is to develop an empirical understanding and comprehension of different supervised machine learning models through the analysis of a dataset our research team has chosen. Each model's performance will be compared to each other through the use of an evaluation concerning attributes such as precision, accuracy, and error rate.

The models that will be used in this report fall under the umbrella of classification as the problem formulated from our identified dataset attempts to solve if a patient would suffer from a stroke or not.

Once our dataset was identified, exploratory data analysis was performed on it for an initial investigation of the data. It aided in the discovery of our predicative problem as well as what models would be appropriate for solving it. After each model was evaluated and compared a summary of results as well as a reflection was undertaken to bring the report to a close.

1.1 Dataset identification

The dataset our research team discovered and chose to focus on is centred around the premise of Stroke prediction [**1**]. Kaggle was the website the dataset was extracted from; the author states that the original source for the data is confidential and should only be used for educational purposes.

The dataset is comprised of 12 features and 5110 observations. Each one representing a patient concerning factors such as gender, bmi and if they suffered from a stroke or not as the core feature. All the features present in the dataset (with the exception of each patient's unique identifier) will be used to determine what could correlate to the occurrence of a stroke.

1.2 Supervised learning task identification

The predicative problem tackled is a classification problem concerning stroke predication. By analysing the records in the in the dataset we will be attempting to uncover if certain features may serve as more prominent factors in occurrence of strokes. Such features would reveal if a patient may suffer from a stroke or not.

The variable that will be used as the target is the 'Stroke' feature for the establishing of ground truth. This variable is a numerical value that is presented as either a 0 or 1 in the dataset, a 0 means a patient hasn't suffered from a stroke while 1 means the opposite.

1.3 Team Identification

Forename	**Surname**	**Student ID**	**Model(s) developed**
Dharmarlou	Bowen	20123076	SVM
Haban	Islam	20109816	Decision Tree
Gurwinder	Singh	19120141	KNN
Oskar	Ziobro	19140977	Random Forest

2 Exploratory Data Analysis

2.1 Question(s) identification

As the dataset is comprised of 12 features there are specific variables that are probable to present as prime contributors such as BMI, Hypertension and Age to start with. The general assumption can be made but not limited to the fact that a stroke in addition to many other medical issues are closely tied with age.

What is the average age is of a stroke occurrence in the dataset and is it congruent with historical data?

The age of 74 is deemed to be the average age, from sources such as Healthline and directly correlate strokes to many hearts problems [**2**]. As Hypertension and Heart disease concern the heart and are features present in the dataset, it's reasonable to assume that if a patient has heart disease, then high blood pressure would be above a particular threshold given a patient's age.

Features such as gender and marriage in the dataset are assumed to provide less insight into the occurrence of a stroke. This is because the question of if a patient has ever been married before and their experience seems rather subjective when in correlation to stroke.

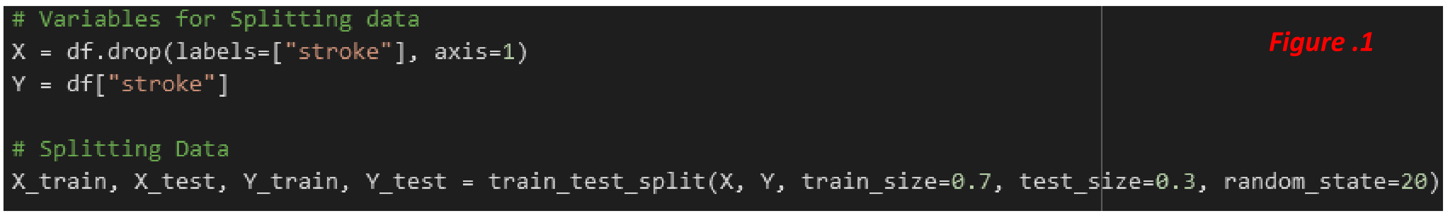
If two patients have previously married, but once has had a stroke and the other has not there is not objective measure that can be implemented solely based on the feature of marriage. It is possible though that there may be a link between gender and marriage for which gender has more cases of strokes an if marriage does present some level of effect.

BMI and Average Glucose are the other two prominent numerical features in the dataset. A similar relationship is expected between a patient's glucose level and BMI as with heart disease and hypertension.

As BMI is concerned with weight and glucose is known to closely mirror insulin levels [**3**], we can assume that a patient's BMI would be higher as the result of a higher average glucose level.

2.2 Splitting the dataset

Our research team have settled on a 70%:30% (Training: Testing) split of the dataset, this was formulated in order to prevent any over/underfitting of the dataset. The sklearn function 'train_test_split()' was utilized to achieve this, with the 'random_state=20' to ensure the continuity of the teams splitting (**Fig. 1**).



```
# Variables for Splitting data
X = df.drop(labels=["stroke"], axis=1)
Y = df["stroke"]

# Splitting Data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7, test_size=0.3, random_state=20)
```

Figure .1

2.3 Exploratory Data Analysis process and results

Uni-variant Exploration

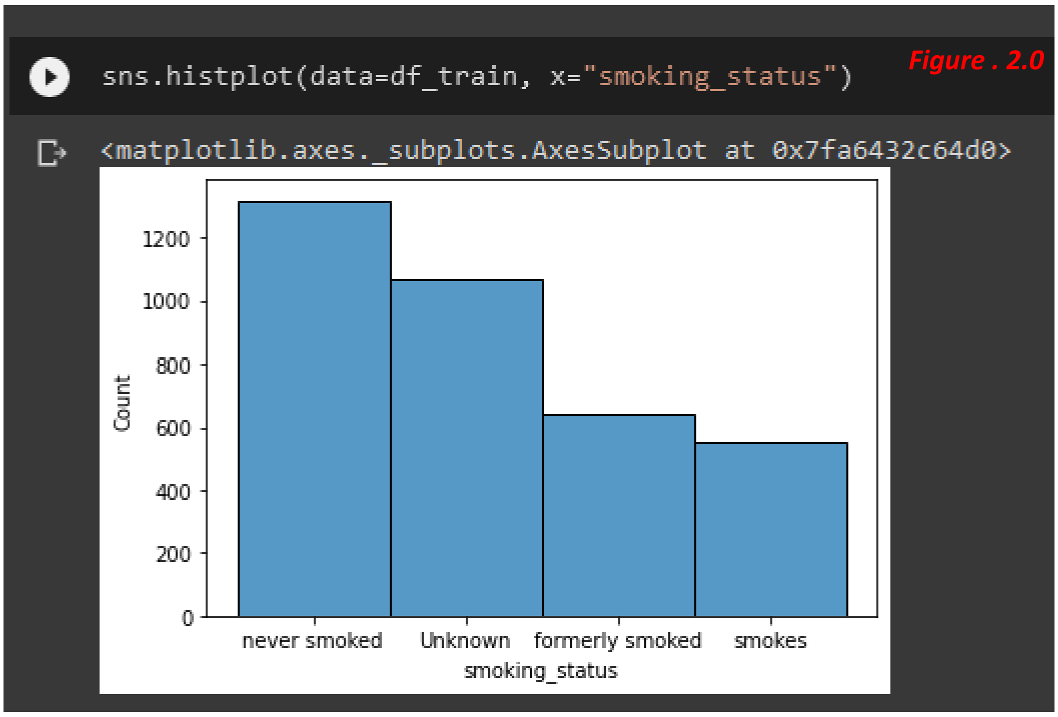


Figure . 2.0

```
sns.histplot(data=df_train, x="smoking_status")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa6432c64d0>
```

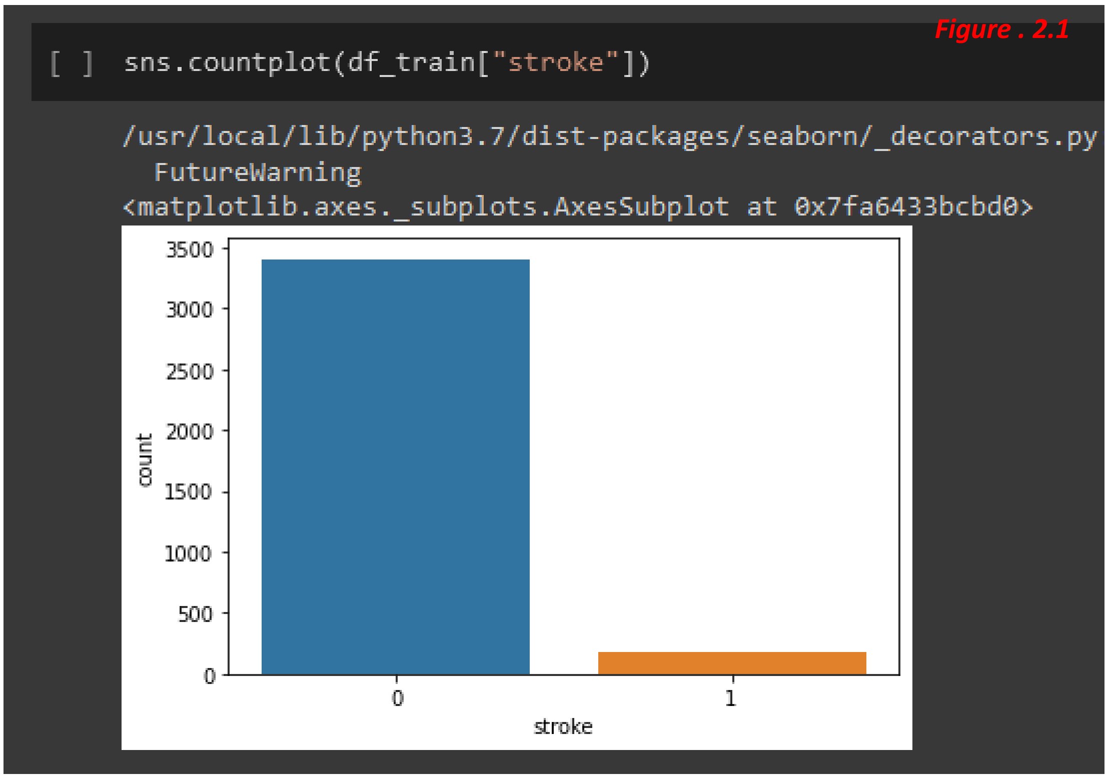


Figure . 2.1

```
[ ] sns.countplot(df_train["stroke"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fa6433bcbd0>
```

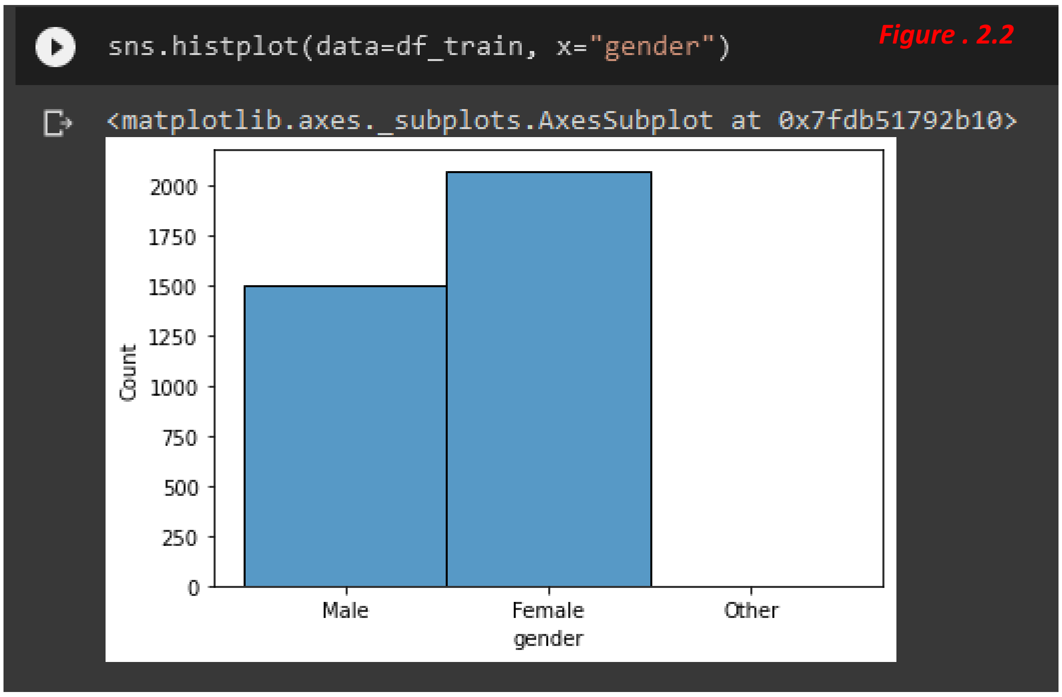


Figure . 2.2

```
sns.histplot(data=df_train, x="gender")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb51792b10>
```

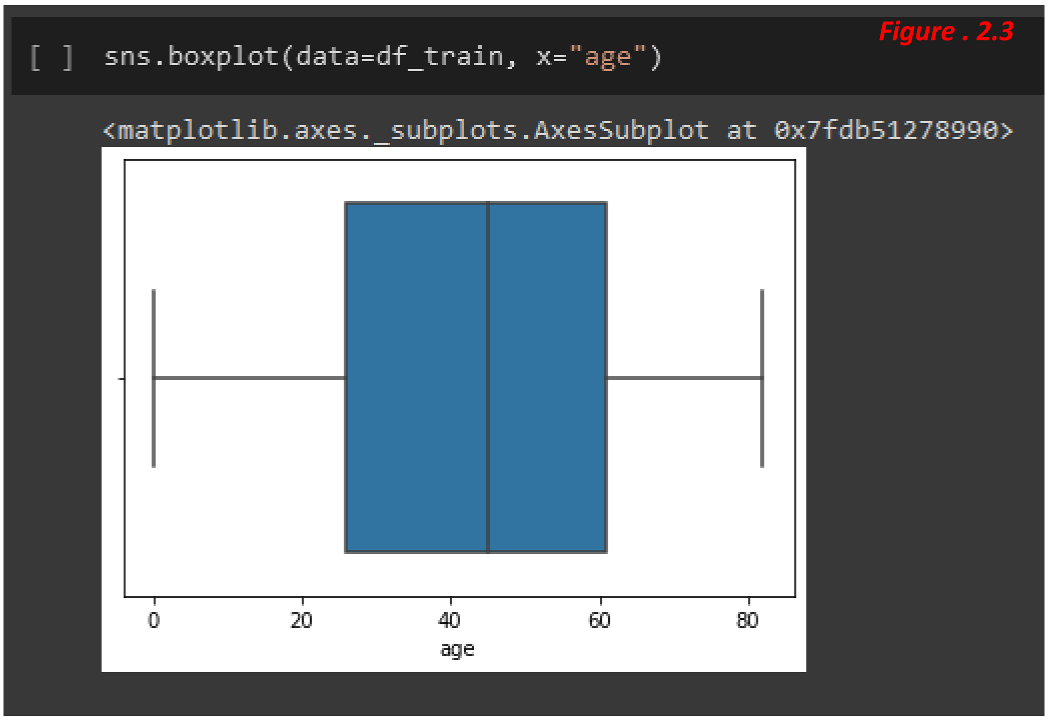


Figure . 2.3

```
[ ] sns.boxplot(data=df_train, x="age")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb51278990>
```

Figure . 2.4

```
sns.histplot(data=df_train, x="ever_married")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb512cf1d0>
```

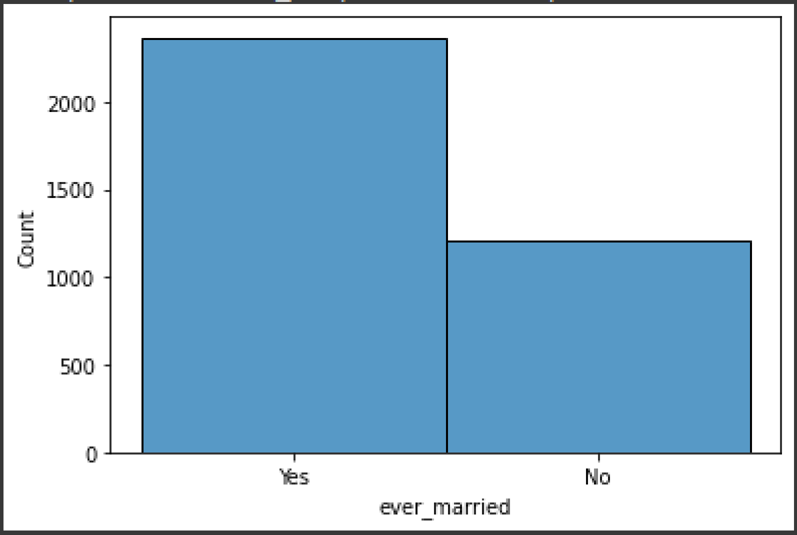
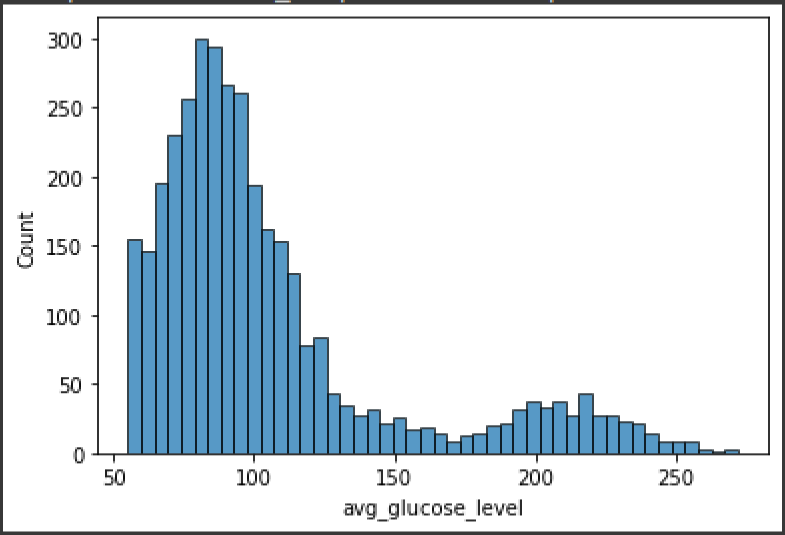


Figure . 2.5

```
[ ] sns.histplot(data=df_train, x="avg_glucose_level")
```

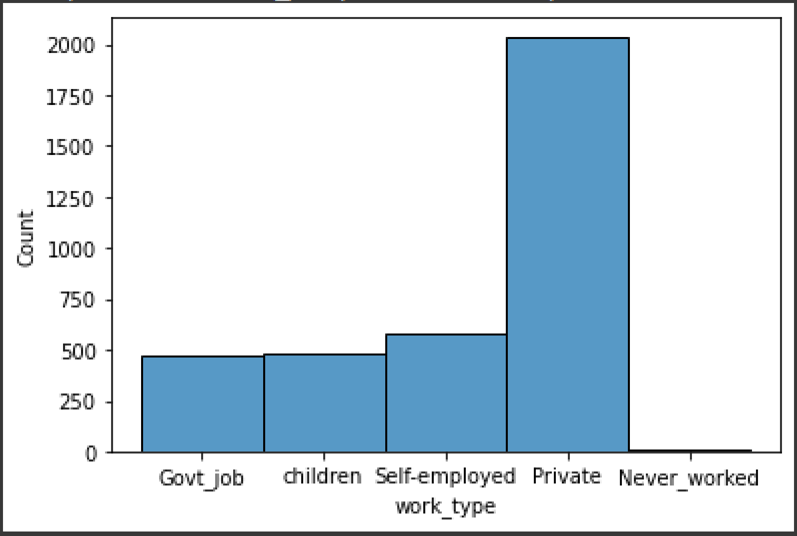
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb5156f890>
```



```
sns.histplot(data=df_train, x="work_type")
```

Figure . 2.6

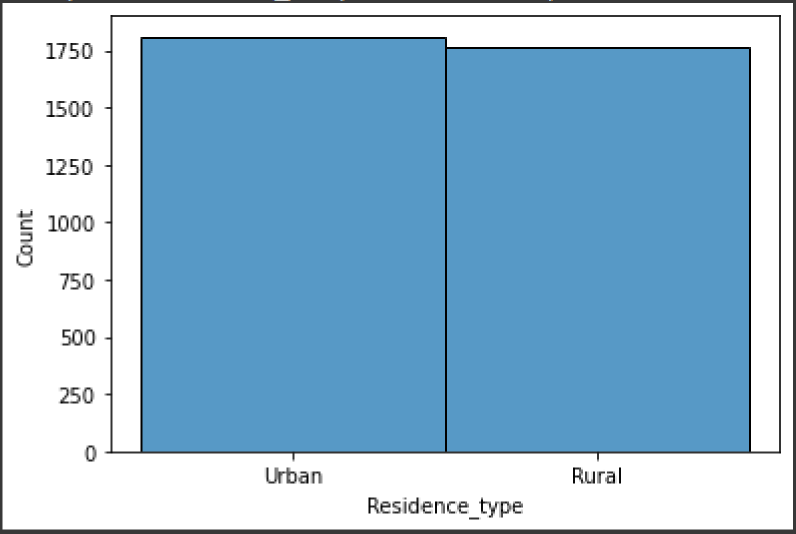
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa6438e7150>
```



```
[ ] sns.histplot(data=df_train, x="Residence_type")
```

Figure . 2.7

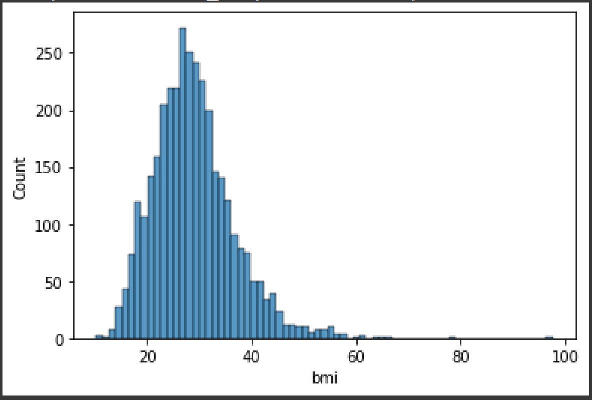
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa643962190>
```



```
sns.histplot(data=df_train, x="bmi")
```

Figure . 2.8

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa64346b090>
```



```
[ ] sns.histplot(data=df_train, x="avg_glucose_level", hue="stroke", multiple="stack")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f257ccae5d0>
```

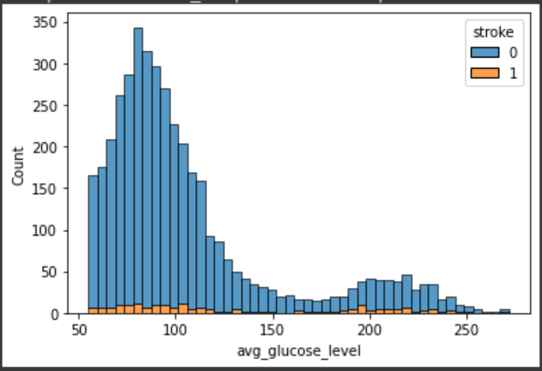
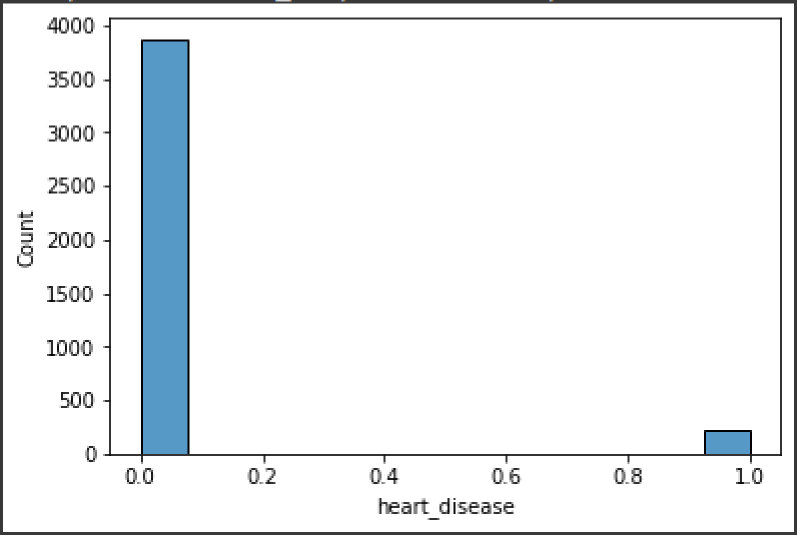


Figure . 2.9

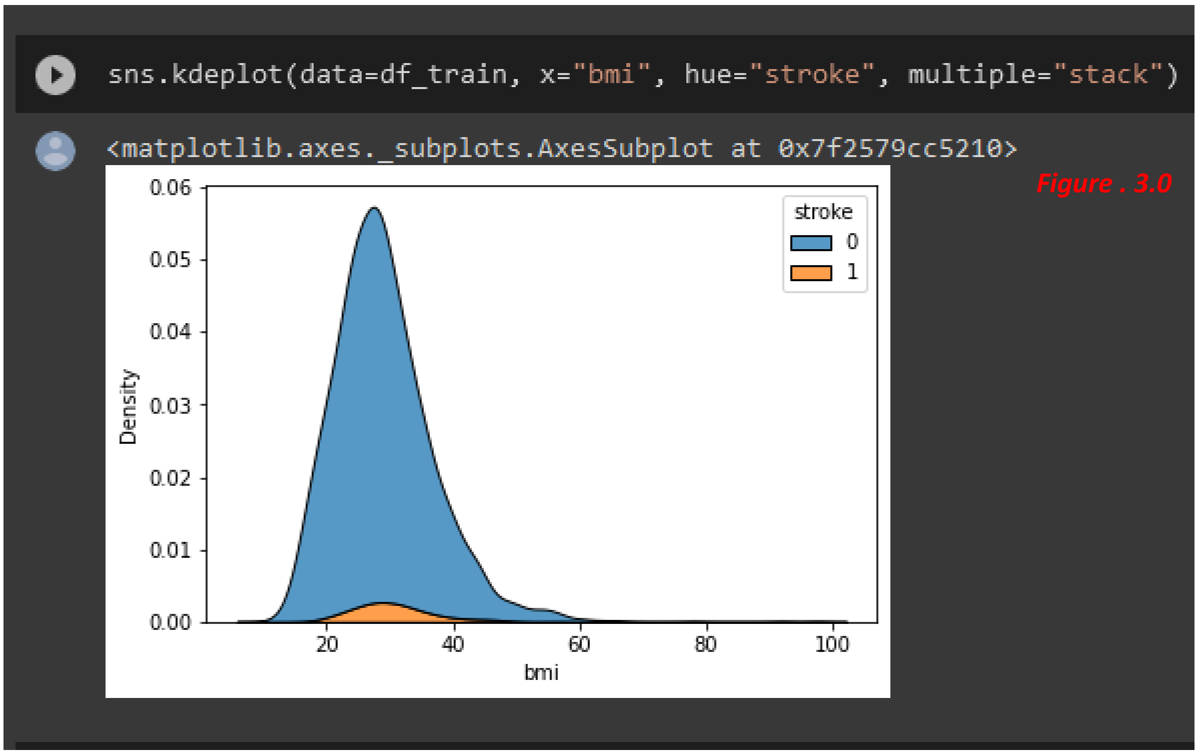
```
[ ]  sns.histplot(data=df_train, x="heart_disease")
```

Figure . 2.10

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f258035bf50>
```



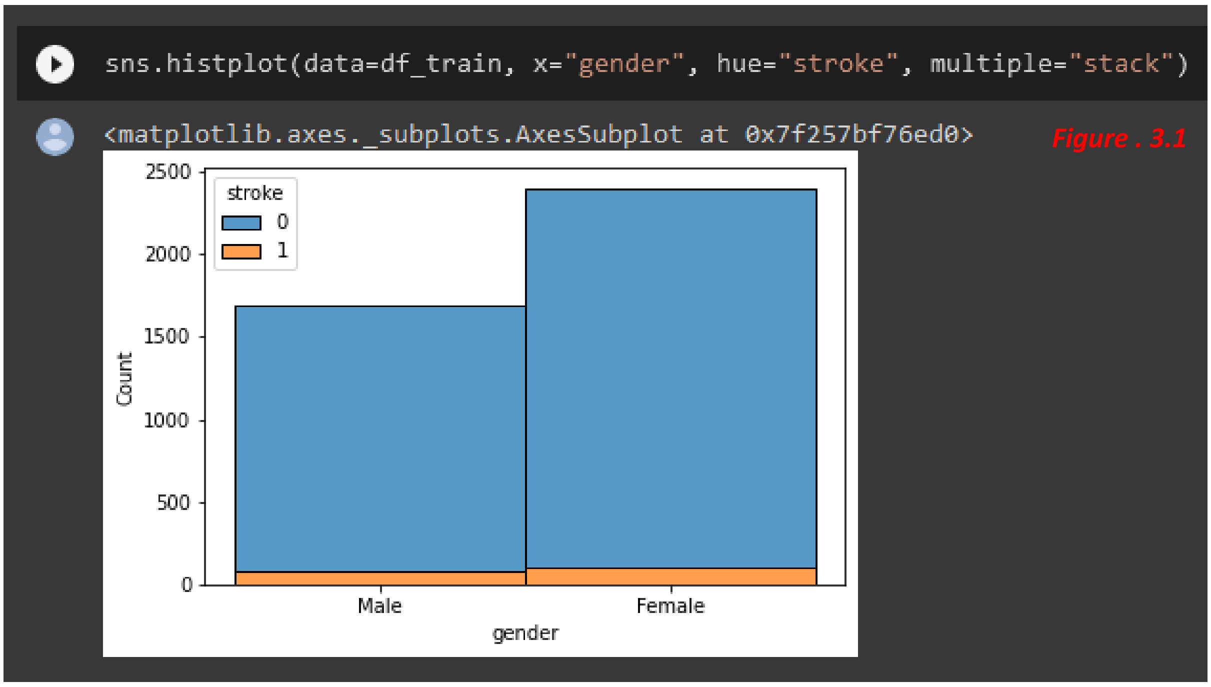
Multi-variant Exploration



```
sns.kdeplot(data=df_train, x="bmi", hue="stroke", multiple="stack")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2579cc5210>
```

Figure . 3.0



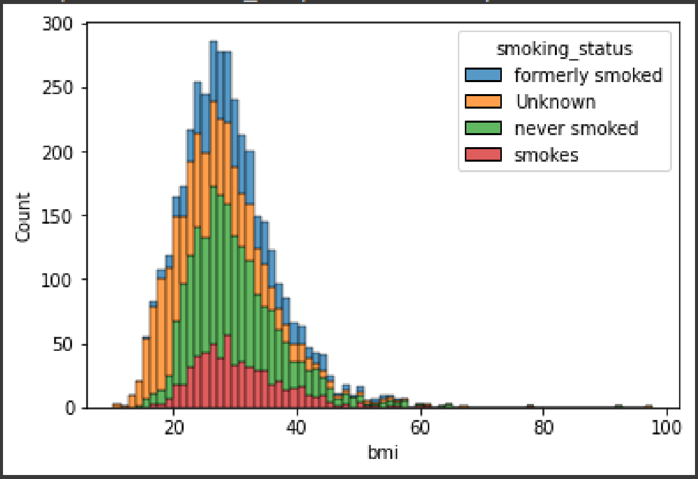
```
sns.histplot(data=df_train, x="gender", hue="stroke", multiple="stack")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f257bf76ed0>
```

Figure . 3.1

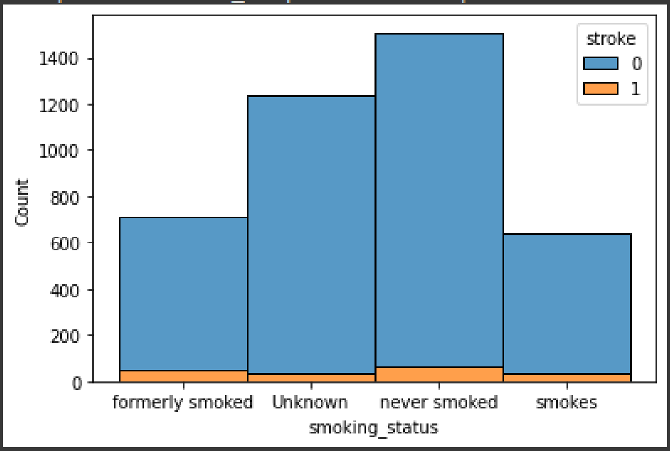
```
[ ]  sns.histplot(data=df_train, x="bmi", hue="smoking_status", multiple="stack")

     <matplotlib.axes._subplots.AxesSubplot at 0x7f2577ec12d0>
```

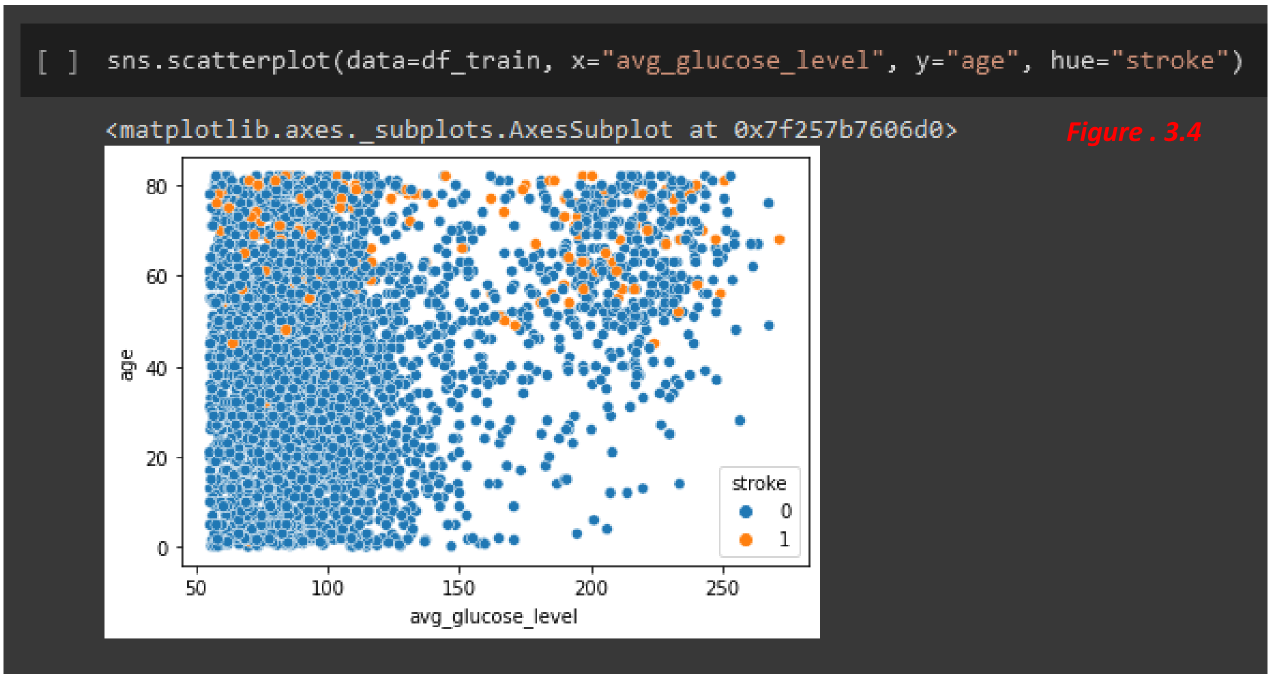
Figure . 3.2



```
[ ]  sns.histplot(data=df_train, x="smoking_status", hue="stroke", multiple="stack")

     <matplotlib.axes._subplots.AxesSubplot at 0x7f25795ede50>
```

Figure . 3.3





```
[ ]  sns.scatterplot(data=df_train, x="avg_glucose_level", y="age", hue="stroke")

     <matplotlib.axes._subplots.AxesSubplot at 0x7f257b7606d0>
```

Figure . 3.4

In the exploratory data analysis the core library used for much of the visualisations was seaborn due to its similarity to matplotlib. A mixture of histplot, scatterplot and boxplots were used throughout the EDA. **Figures 2.0** to **2.10** are of the univariant analysis, while **Figures 3.0** to **3.4** are of the multivariant analysis.

2.4 EDA conclusions

From the EDA performed it was established that 'genders' are quite evenly distributed between 'male' and 'female' but there is only 1 record for the 'other' gender type which may present problems for use in models later on. The 'ever married' feature presents that more people are married than not. From the 'work type' feature those that work in the private sector were the most prominent.

The 'smoking status' is quite evenly distributed, with 'ever smoked' having the greatest number of records. Most of the strokes are appeared to be happening to those between the age gap of 40 and 80, with the age of 80 having the highest number of results. Most of the patients have around 30 BMI score but, most of the strokes happened to the patients with a score of 20 and 40.

Interestingly those that are married present seem to present with more strokes and when the married data is split through the feature of gender, males seem to have a higher occurrence of stroke than female when married. The reverse appears with females that are not married. This may occur for a multitude of reasons but is unfortunately beyond the scope of this report.

3 Experimental Design

3.1 Identification of your chosen supervised learning algorithm(s)

Support vector machines (SVMs) is the machine learning algorithm I selected due to its ability to be used for classification or regression tasks. SVMs are particularly well-suited for classification of complex or small-sized datasets and are known for their ability to achieve high accuracy even when the data is not linearly separable.

With this in mind SVMs are effective in high-dimensional spaces: Stroke prediction often involves the use of many variables or features, such as age, blood pressure, cholesterol levels, and other risk factors. SVMs are able to manage high-dimensional data effectively and can find the hyperplane that maximally separates the different classes in the data.

SVMs are robust to noise and outliers: In medical datasets, it is common to have missing or incorrect data, which can affect the performance of certain machine learning algorithms. SVMs are quite robust to noise and outliers and can still produce satisfactory results even if the data is not completely clean.

3.2 Identification of appropriate evaluation techniques

Our research team settled on the use of evaluation metrics such as precision, accuracy, and error rate due to their ability to provide a simple and comprehensive breakdown down of a models performance. These metrics are used in addition to a confusion matrix to better present the results.

Accuracy measures the proportion of correct predictions made by the model and is calculated as the number of true positive and true negative predictions divided by the total number of predictions. High accuracy indicates that the model is making few mistakes and is a good overall measure of performance.

Precision on the other hand measures the proportion of positive predictions that are actually correct. It is calculated as the number of true positive predictions divided by the total number of positive predictions made by the model. High precision indicates that the model is not making a lot of false positive predictions and is particularly important when the consequences of a false positive are severe (e.g., in medical diagnosis).

The error rate is the opposite of accuracy and measures the proportion of incorrect predictions made by the model. It is calculated as the number of false positive and false negative predictions divided by the total number of predictions. A low error rate indicates that the model is making few mistakes and is a good overall measure of performance.

In general, we want to aim for high accuracy and low error rate, as well as high precision if the consequences of a false positive are severe. By considering these metrics together, we can get a clearer picture of the overall performance of the model and identify any areas that need improvement.

3.3 Data cleaning and Pre-processing transformations

The dataset presented 201 missing values all being localised to the bmi feature, this information was acquired with the use of the 'isna()' function from pandas (**Fig. 4.0**). As some of our dataset's features contained categorical data (**Fig. 4.1**) the first pre-processing step was to perform appropriate value encodings. The key features that required encoding were as follows:

- 'ever_married'
- 'work_type'
- 'Residence_type'
- 'smoking_status'
- 'gender'

The functions used to perform the encoding was the 'get_dummies()' pandas. After further inspection it was encountered that the 'gender' feature contained 1 'Other' value which was discarded (**Fig. 4.3**).

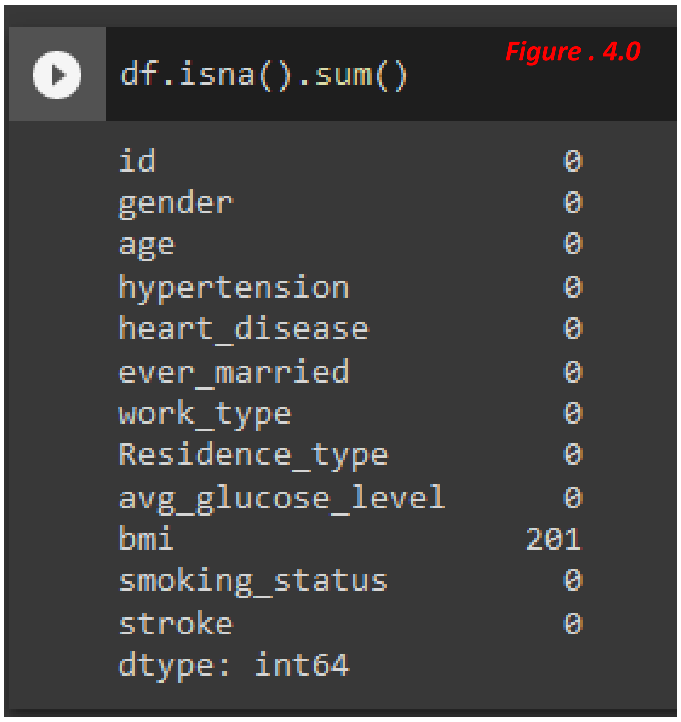


Figure . 4.0

```
df.isna().sum()

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
```

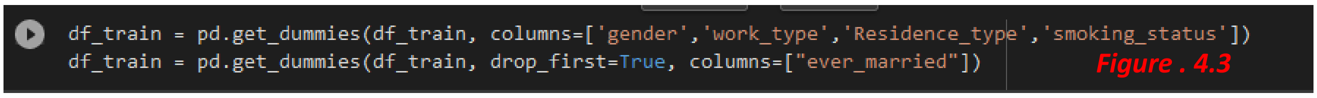


Figure . 4.3

```
df_train = pd.get_dummies(df_train, columns=['gender','work_type','Residence_type','smoking_status'])
df_train = pd.get_dummies(df_train, drop_first=True, columns=["ever_married"])
```

Interpolation was the next action to be performed as there were the missing values of the 'bmi' feature that need to be dealt with. This was achieved with the use of pandas 'interpolation' function.

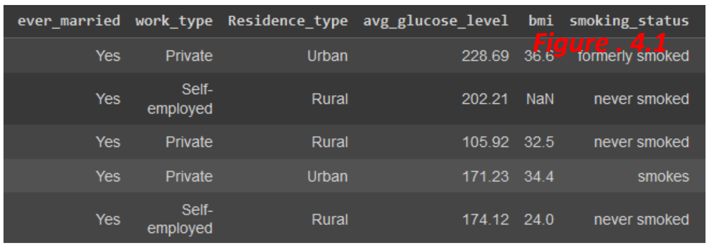


Figure . 4.1

ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
Yes	Private	Urban	228.69	36.6	formerly smoked
Yes	Self-employed	Rural	202.21	NaN	never smoked
Yes	Private	Rural	105.92	32.5	never smoked
Yes	Private	Urban	171.23	34.4	smokes
Yes	Self-employed	Rural	174.12	24.0	never smoked

The final problem to deal with was the accommodation of over/under sampling due to the dataset's class distribution. The stroke class is imbalanced with it presenting a significantly greater number of patients without the occurrence of a stroke than those with. As this 'stroke' class is our target we had to use an appropriate method to deal with oversampling problem. SMOTE's synthetic oversampling functionality was utilised to resolve to issue (**Fig. 4.4**).

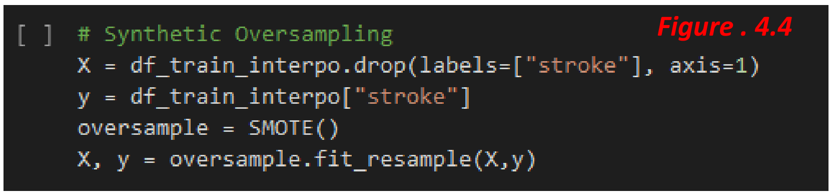


Figure . 4.4

```
[ ]  # Synthetic Oversampling
     X = df_train_interpo.drop(labels=["stroke"], axis=1)
     y = df_train_interpo["stroke"]
     oversample = SMOTE()
     X, y = oversample.fit_resample(X,y)
```

3.4 Limitations and Options

When dealing with the problem of oversampling, there were a few other methods that were considered despite SMOTE's popularity. This is due to how the quality of SMOTE's generated synthetic samples not being of high quality resulting in them either being unrealistic or irrelevant. We did experiment with class weighting as an alternative to dealing with the dataset imbalance but after implementation it actually reduce the performance of the model.

4 Predictive Modelling / Model Development

4.1 The predictive modelling processes

Once the final step of the pre-processed training data was performed with the use of SMOTE it was passed into sklearn's SVC to implement the Support Vector Machine (**Fig. 5.0**). The model was initialised with its default hyperparameters with the exception of the kernel being set to 'linear' as I concluded it would most likely provide greater accuracy over the default 'rbf'. The main default hyperparameters in the model are as follows:

- C=1.0
- Degree=3
- Gamma='scale'
- Class_weight=None

They were all left like this as it only seemed appropriate to modify the kernel ahead of time.

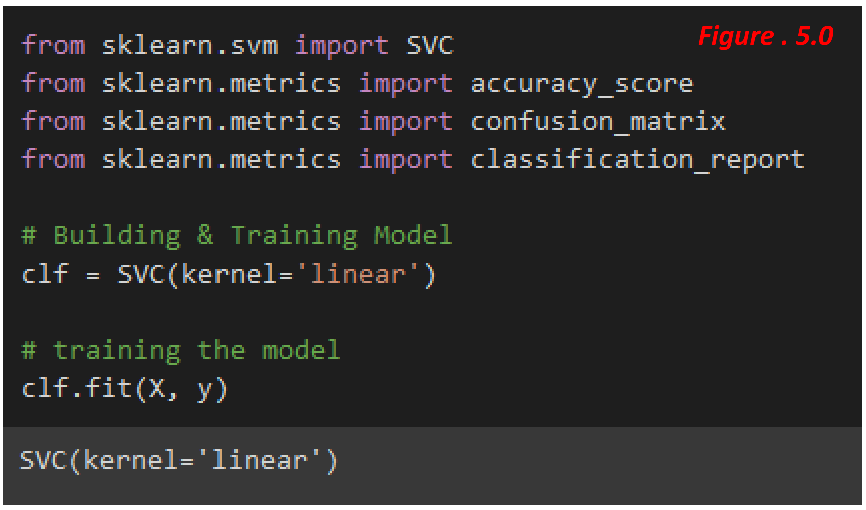


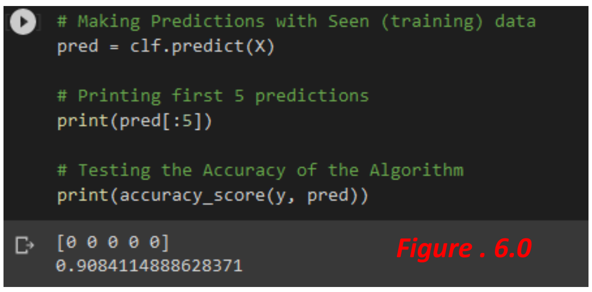
Figure . 5.0

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Building & Training Model
clf = SVC(kernel='linear')

# training the model
clf.fit(X, y)
```

```
SVC(kernel='linear')
```

4.2 Evaluation results on "seen" data

The method of evaluation used on the training set involved the use of the 'accuracy_score()', 'confusion matrix()' and the 'classification report()' in addition to a heatmap. To implement the 'accuracy_score' correctly the 'predict()' function was used to present the predicted values from the trained model, which was used in tandem with the 'y' variable from the pre-processed training data to output a accuracy score (**Fig. 6.0**). An accuracy of 90% was achieved on the seen data.

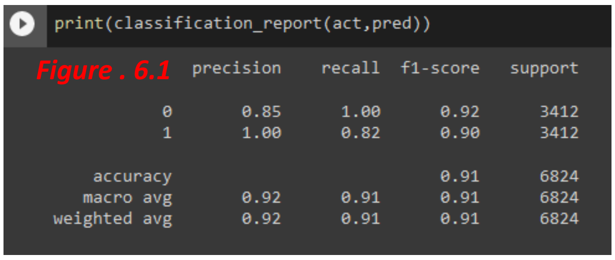


```
# Making Predictions with Seen (training) data
pred = clf.predict(X)

# Printing first 5 predictions
print(pred[:5])

# Testing the Accuracy of the Algorithm
print(accuracy_score(y, pred))
```

```
[0 0 0 0 0]
0.9084114888628371
```

Figure . 6.0

The confusion matrix was the next step of the evaluation, which serves the purpose of providing insight into the performance of the model in accordance with its true values being known. Due to its simplistic representation of the models performance a classification report was also used. As its ability to



```
print(classification_report(act,pred))
```

Figure . 6.1

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      3412
           1       1.00      0.82      0.90      3412

    accuracy                           0.91      6824
   macro avg       0.92      0.91      0.91      6824
weighted avg       0.92      0.91      0.91      6824
```

display a summary with the utilisation of factors such as precision, recall, support and f1-score for each class are far more beneficial. (**Fig. 6.1**).

The performance metrics reveal that in terms of precision, the model identified 85% and 100% of the samples labelled 0 and 1 respectively. This indicates that the model has a low rate of false positives and a high rate of correctly identifying positive class.

Recall, on the hand provides insight into the minimizing of false negatives. The model identified 100% and 82% of the samples labelled 0 and 1 respectively. This indicates that the model has a low rate of false negatives and a high rate of correctly identifying positive instances.

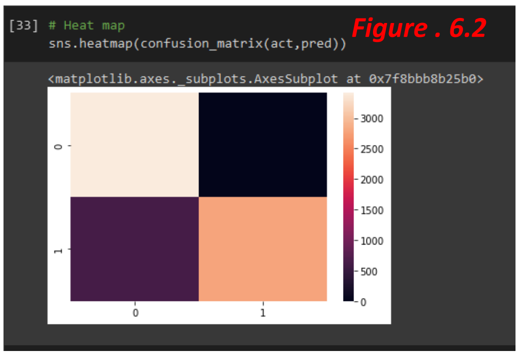


Figure . 6.2

```
[33] # Heat map
     sns.heatmap(confusion_matrix(act,pred))

     <matplotlib.axes._subplots.AxesSubplot at 0x7f8bbb8b25b0>
```

Heatmap was the final tool used to provide a more visual presentation of the data as it may lead to the discovery of any patterns or relationships in the data. To admit with 4 factors being considered the map does appear rather simplistic but its is provided an easy-to-understand visualization of our data shared through the research team (**Fig. 6.2**).

5 Evaluation and further modelling improvements

5.1 Initial evaluation comparison

Figure . 7.0, Gurwinder

```
[[400  83]
 [ 16  12]]
              precision    recall  f1-score   support

         0.0       0.96      0.83      0.89       483
         1.0       0.13      0.43      0.20        28

    accuracy                           0.81       511
   macro avg       0.54      0.63      0.54       511
weighted avg       0.92      0.81      0.85       511
```

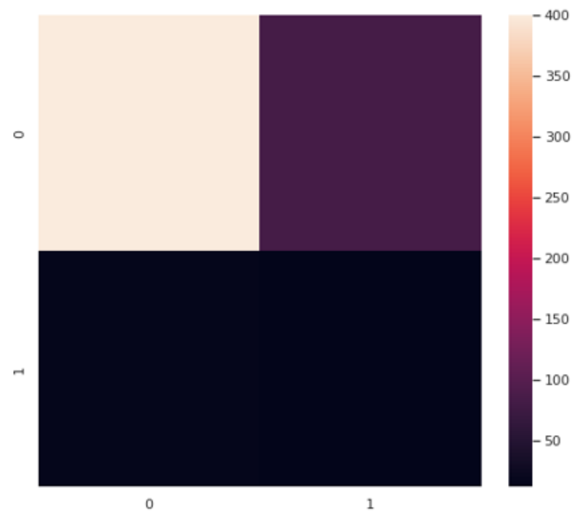
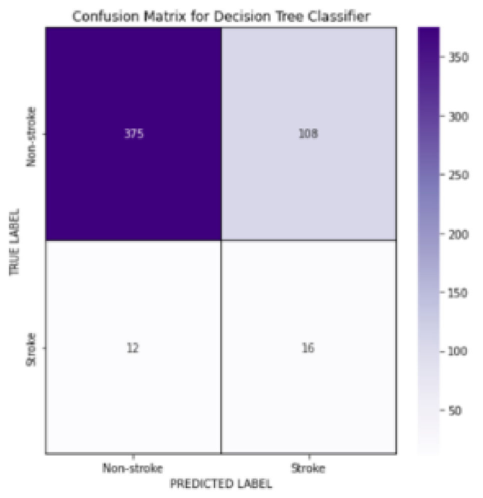
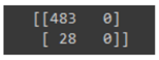


Figure . 7.1, Habban

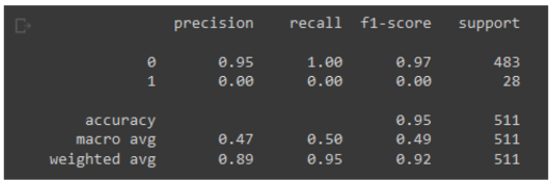
```
              precision    recall  f1-score   support

           0       0.97      0.78      0.86       483
           1       0.13      0.57      0.21        28

    accuracy                           0.77       511
   macro avg       0.55      0.67      0.54       511
weighted avg       0.92      0.77      0.83       511
```



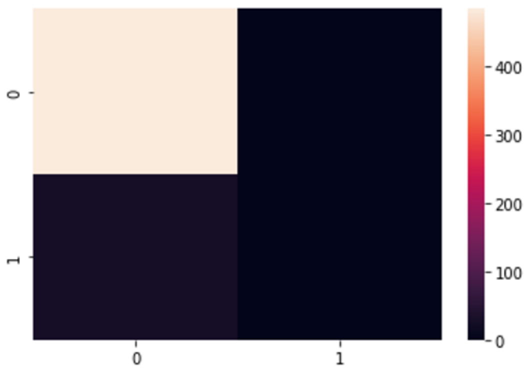


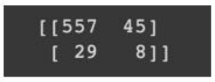
```
[[483   0]
 [ 28   0]]
```

Figure . 7.2, Dharmarlou



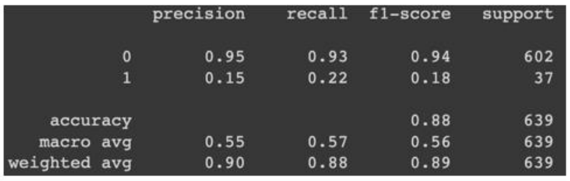
```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       483
           1       0.00      0.00      0.00        28

    accuracy                           0.95       511
   macro avg       0.47      0.50      0.49       511
weighted avg       0.89      0.95      0.92       511
```



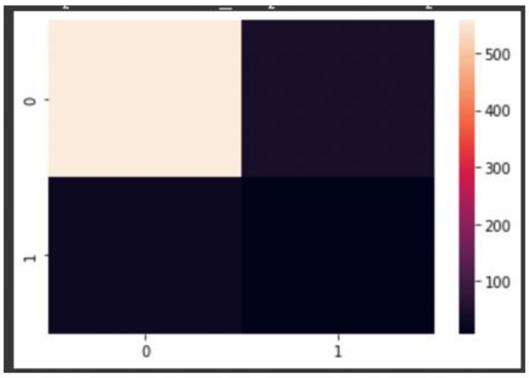


```
[[557  45]
 [ 29   8]]
```

Figure . 7.3, Oskar



```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       602
           1       0.15      0.22      0.18        37

    accuracy                           0.88       639
   macro avg       0.55      0.57      0.56       639
weighted avg       0.90      0.88      0.89       639
```



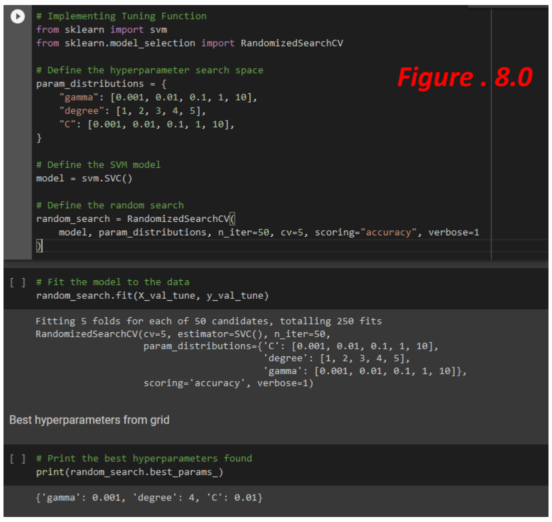
The same evaluation metrics were used on to throughout the team on the unseen data from the validation set. Out of all the models Habban's decision tree (**Fig. 7.1**) presented the best overall performance while mine (**Fig. 7.2**) on the other hand performed the worse as class 1 presented 0.00 for all its metrics. This may be due to the model not being able to identify class 1 or because there is a lack of instances of class 1.

5.2 Further modelling improvements attempted

After the comparison performed in team evaluation, I decided in order to fine tune my model I would use a hyperparameter optimizer as manual fine tuning would have too time consuming and somewhat limited as I my knowledge and experience model is only so developed. The hyperparameter optimizer used was sklearn's 'RandomizedSearchCV' as 'GridSearchCV' would run for a long duration (2+ hours).

The validation model was used in tandem with a hyperparameter optimizer which resulted in a lower score, but this was deem fine as the optimization (**Fig. 8.0**) improved the result of the test set which will be elaborated on in the evaluation of the final results in the next section. The gamma, degree and C were the optimized parameters, with the optimizer outputting the results 'gamma=0.001', 'degree=4', 'C=0.01' and 'kernel=poly'.

The model was retrained with these parameters (**Fig. 8.1**). which presented values in all metrics of a confusion matrix but there was a drop in its accuracy score from 94% shown before fine tuning to 90%.



```
# Implementing Tuning Function
from sklearn import svm
from sklearn.model_selection import RandomizedSearchCV

# Define the hyperparameter search space
param_distributions = {
    "gamma": [0.001, 0.01, 0.1, 1, 10],
    "degree": [1, 2, 3, 4, 5],
    "C": [0.001, 0.01, 0.1, 1, 10],
}

# Define the SVM model
model = svm.SVC()

# Define the random search
random_search = RandomizedSearchCV(
    model, param_distributions, n_iter=50, cv=5, scoring="accuracy", verbose=1
)
```

```
# Fit the model to the data
random_search.fit(X_val_tune, y_val_tune)
```

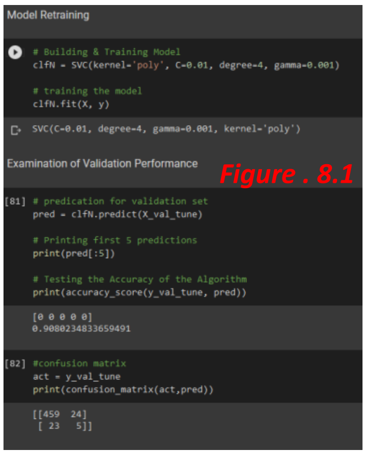
```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
RandomizedSearchCV(cv=5, estimator=SVC(), n_iter=50,
                   param_distributions={'C': [0.001, 0.01, 0.1, 1, 10],
                                        'degree': [1, 2, 3, 4, 5],
                                        'gamma': [0.001, 0.01, 0.1, 1, 10]},
                   scoring='accuracy', verbose=1)
```

Best hyperparameters from grid

```
# Print the best hyperparameters found
print(random_search.best_params_)
```

```
{'gamma': 0.001, 'degree': 4, 'C': 0.01}
```

Figure . 8.0



Model Retraining

```
# Building & Training Model
clfN = SVC(kernel='poly', C=0.01, degree=4, gamma=0.001)

# training the model
clfN.fit(X, y)
```

```
SVC(C=0.01, degree=4, gamma=0.001, kernel='poly')
```

Examination of Validation Performance

```
[81] # predication for validation set
     pred = clfN.predict(X_val_tune)

     # Printing first 5 predictions
     print(pred[:5])

     # Testing the Accuracy of the Algorithm
     print(accuracy_score(y_val_tune, pred))
```

```
[0 0 0 0 0]
0.9080234833659491
```

```
[82] #confusion matrix
     act = y_val_tune
     print(confusion_matrix(act,pred))
```

```
[[459  24]
 [ 23   5]]
```

Figure . 8.1

5.3 Final Evaluation results

The results of the fine-tuned model provided a slight 2% drop in accuracy_score when tested on the unseen data of the testing set from 94% to 92% but provided values in all the parameters of the confusion matrix.

The untuned model presented an similar issue of not being able to identify any instances of class 1, but after being fine-tuned it was able to (**Fig. 9.0**). So in comparison to the untuned validation set performance by tuning the model it was able to better identify class 1 instances.

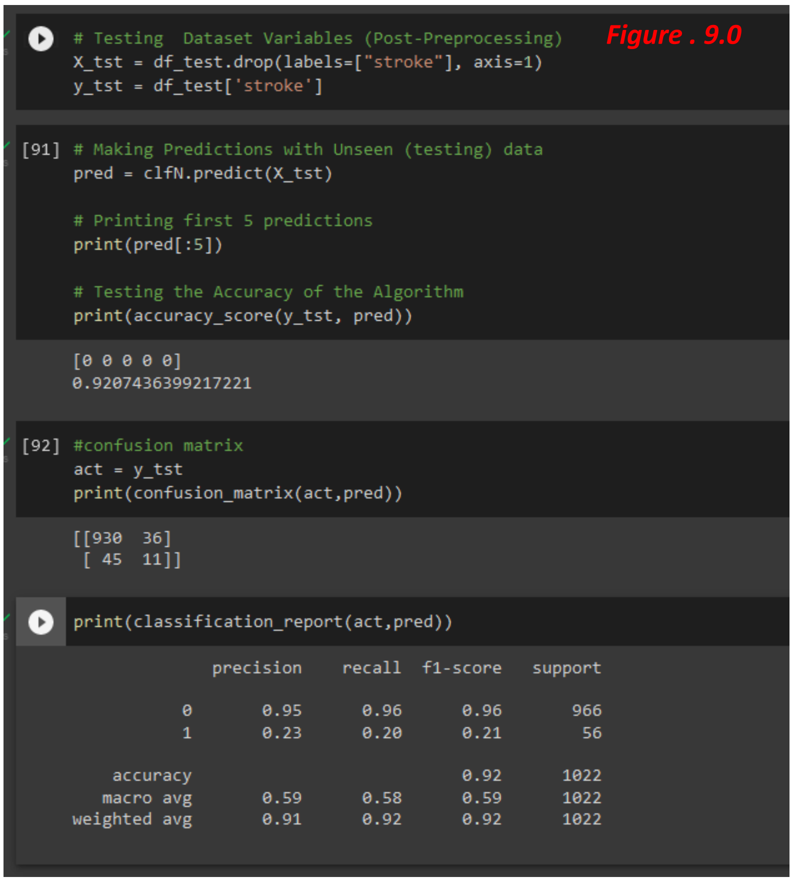


Figure . 9.0

```
# Testing  Dataset Variables (Post-Preprocessing)
X_tst = df_test.drop(labels=["stroke"], axis=1)
y_tst = df_test['stroke']
```

```
[91] # Making Predictions with Unseen (testing) data
     pred = clfN.predict(X_tst)

     # Printing first 5 predictions
     print(pred[:5])

     # Testing the Accuracy of the Algorithm
     print(accuracy_score(y_tst, pred))

     [0 0 0 0 0]
     0.9207436399217221
```

```
[92] #confusion matrix
     act = y_tst
     print(confusion_matrix(act,pred))

     [[930  36]
      [ 45  11]]
```

```
print(classification_report(act,pred))

              precision    recall  f1-score   support

           0       0.95      0.96      0.96       966
           1       0.23      0.20      0.21        56

    accuracy                           0.92      1022
   macro avg       0.59      0.58      0.59      1022
weighted avg       0.91      0.92      0.92      1022
```

5.3 Summary of results

Below is a table displaying a summary of all the results through the development of the model which is split between the training, validation and testing sets.

Data set split	Accuracy score	F1-Score (class 0)	F1-Score (class 1)
Training	0.90	0.92	0.90
Validation	0.94	0.97	0.00
Testing	0.92	0.96	0.21

5.4 Suggested further improvements to the model development process

With all the methods I've learn from deciding which model to chose and the evaluation methods and metrics I would give me consideration to the possible alternatives available to use given the problem I am trying to solve, classification or regression. Now that I have a comprehension of the foundations required to develop a model I would like to implement more advanced techniques in my next project such as manually tuning a model and coming up with my own methods of the tuning process.

5.5 Reflection on Research Team

From working in a team it gave me insight into how people like to solve and approach problems from different perspectives, leading to discoveries of methods I feel I would not have come across on my own with my biases involved.

5.6 Reflection on Individual Learning

I feel more confident in searching for datasets that can be used for my own or academic purposes, my ability to explore data with a greater variety of methods has improved as past projects have proven to not have been as insight as this one centred around stroke predication. Its was interesting seeing how dataset can reflect real world occurrences and how the process of fine tuning and developing a model led to a sense of achievement and enlightenment in terms of data science. It has motived me to find more projects to work on to further improve my skills within the field.

6 References

References use Harvard method.

[1] Kaggle (2021) *Stroke Prediction Dataset*.

Available at: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset (Accessed: 23 October 2022).

[2] Is Your Risk of Having a Stroke Different Based on Your Age Range? (2022) Available at: https://www.healthline.com/health/age-range-for-stroke (Accessed: 25 October 2022).

[3] Understanding blood sugar and weight loss: why tracking glucose with a continuous glucose monitor (CGM) may be more insightful than tracking calories (2022) Available at: https://www.levelshealth.com/blog/glucose-weight-loss (Accessed: 25 October 2022)