Aaron Rock's Individual Report

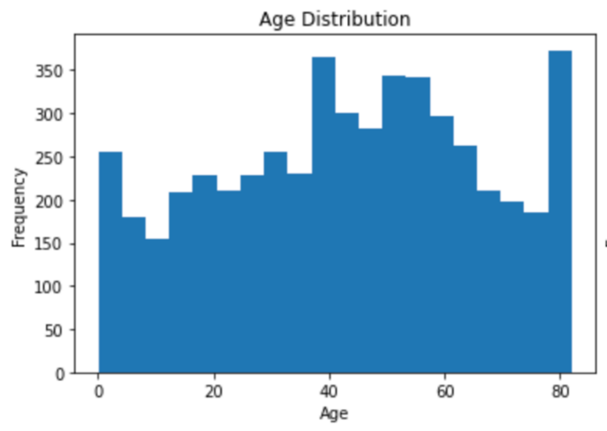
Introduction:

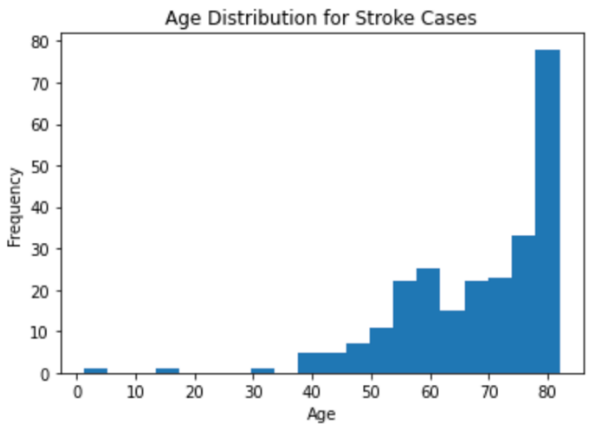
As a group we decided on using public Stroke data from Kaggle to build machine learning models to best predict whether a patient is at risk. As a group we met throughout the second half of the semester to lay out a plan and to split certain parts up such as who wants what models and what part of the EDA will people do. Our project has five main parts: EDA, Data Cleaning, Preprocessing, Feature Reduction, and Model Building. The stroke data we sourced from Kaggle consisted of *5,101 data points* with *12 variables*. The variables are *ID, Gender, Hypertension, Heart Disease, Marital Status, Work Type, Residence Type, Smoking Status, and Stroke*. The target variable in this project is stroke which can be classified by yes or no represented by 1's or 0's. We split the tasks up by dividing the variables into thirds for the exploratory data analysis so each could plot useful insights. Data cleaning was contributed by another group member. For preprocessing and more specifically encoding and scaling each of us included this as we all performed different techniques to see what had the best results. For the machine building portion my group members focused on an abundance of different model, and I worked on two, the Multi-Layer Perceptron model and Keras.

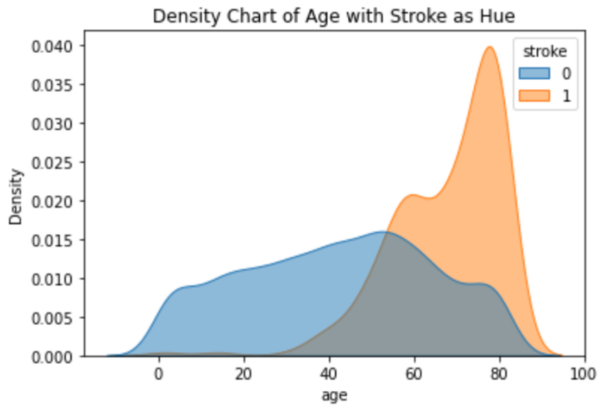
Description of Individual Work:

For the portion contributed by myself I will start with the exploratory data analysis. Like I stated above we split the variables in thirds, and I was tasked with the variables *age, heart disease, and marital status*. Here are the plots that I created from these variables.

Age:

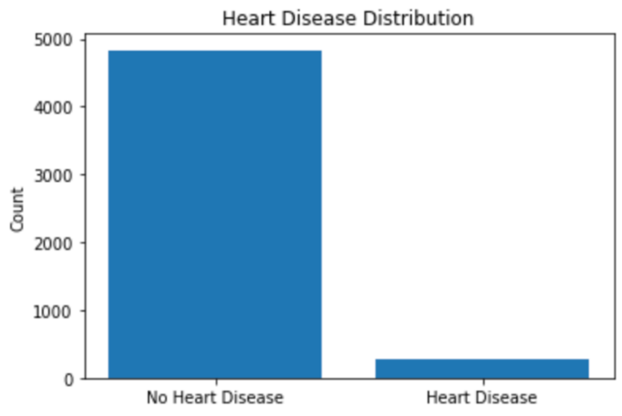


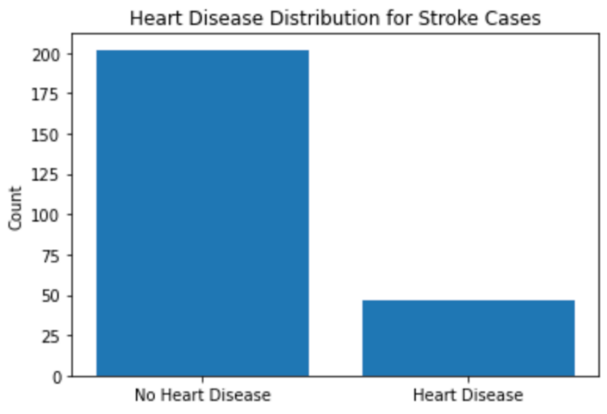




From these plots it is easier to see that the older ages have more frequent strokes.

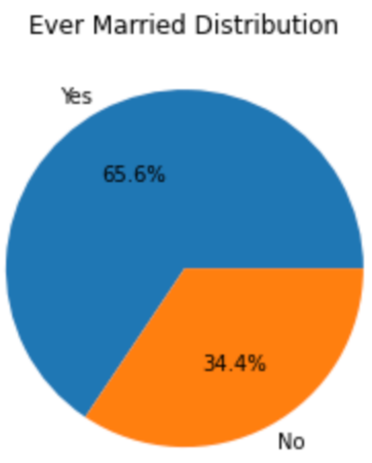
Heart Disease:

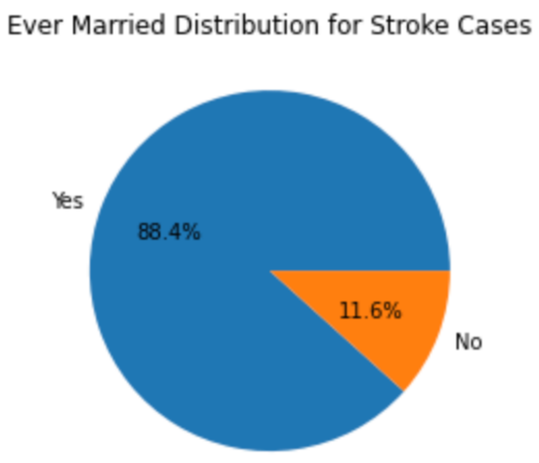


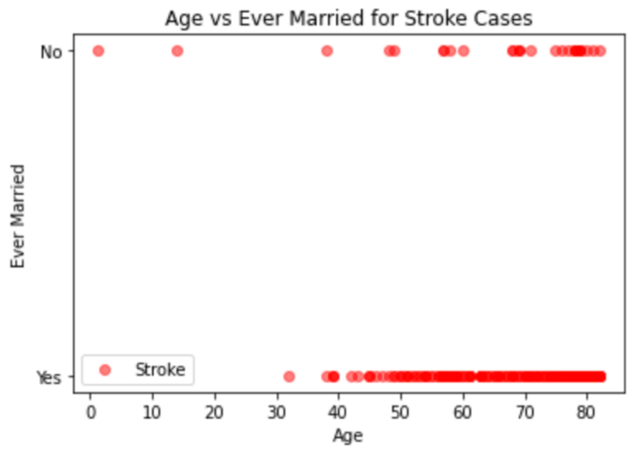


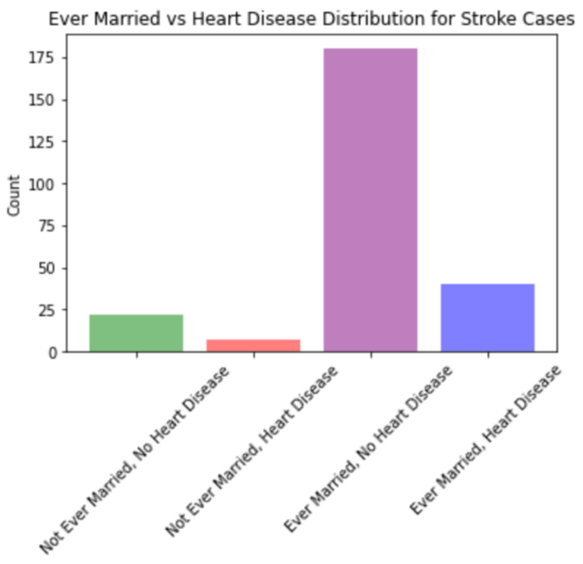
The plots show an imbalance in the cases of heart disease with the data having far less confirmed cases of heart disease. When looking at the plot that takes in to consideration if they have also had a stroke there is still a small imbalance.

Marital Status:



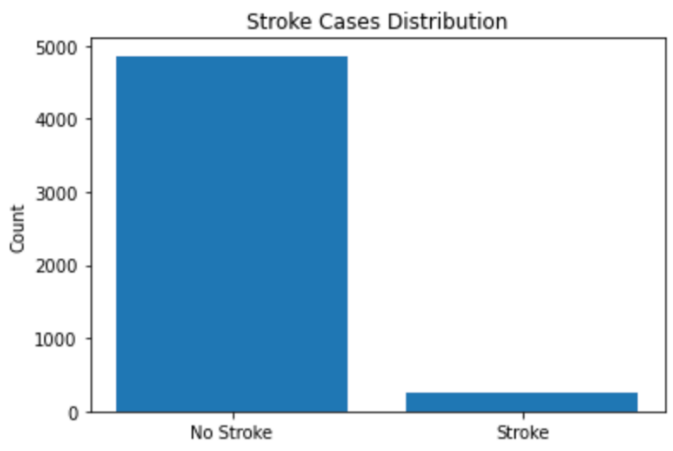


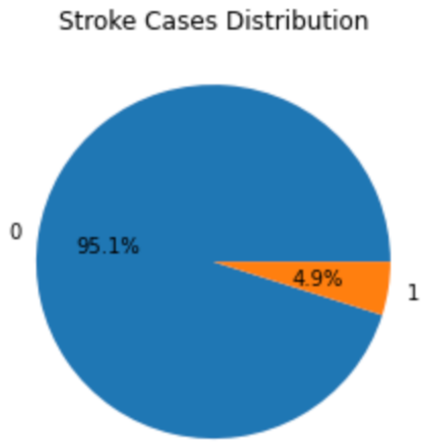




Marital status was more balanced in the dataset than heart disease, but we can see that when we filter if they have had a stroke 88% are married.

Stroke (each group member plotted the stroke variable):





For the variable stroke we can see just how imbalanced this is represented in the dataset. This is addressed later so we have better built models.

Both Keras and the MLP Classifier are neural networks and are Multi-Layer Perceptron models that have two major steps feedforward and then backpropagation to properly update the weights of the neurons and calculate the error.

Forward Propagation

$$\mathbf{a}^0 = \mathbf{p},$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \text{ for } m = 0, 1, \ldots, M-1,$$

$$\mathbf{a} = \mathbf{a}^M.$$

Backward Propagation

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}),$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \text{ for } m = M-1, \ldots, 2, 1,$$

where

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \ldots & 0 \\ 0 & \dot{f}^m(n_2^m) & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & \dot{f}^m(n_{S^m}^m) \end{bmatrix},$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}.$$

Weight Update (Approximate Steepest Descent)

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T,$$

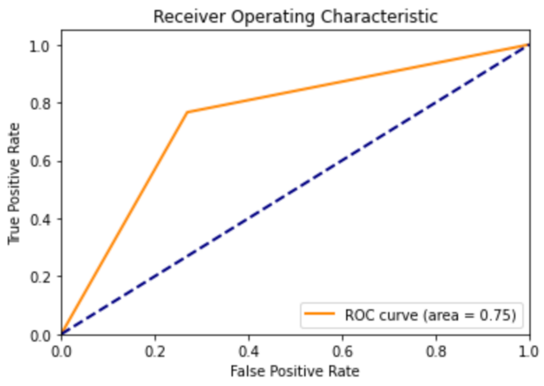
$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m.$$

Just as we discussed in class you forward propagate through the neural network to obtain the output and then take the difference from the expected. Then perform weight and bias updates and forward propagate again this cycle continues until you meet the criteria you set. The formulas above is taken from the book to show the formulation of this algorithm. Sklearn's MLP Classifier is the one I was most comfortable using as handles most of the complexity for you as long as you feed it the correct parameters. For the Keras implementation I built a low complexity model and slowly introduced more and more complexity as the model improved.

Results:

The Multi-Layer Perceptron Classifier from the sklearn is where most of my time was spent. After, correcting the use of SMOTE on the training data to help balance the training data I built 3 different MLP Classifier using a few different techniques including GridSearchCV, PCA, and RFE for my own individual model. The first MLP Classifier is a simple implementation that is fed a parameter space uses GridSearchCV to find the optimal parameters and then uses those params to predict stroke. For GridSearchCV, I am setting the parameter scoring to evaluate based off the recall score due to the imbalance issues and the fact we want to have better metric for over predicting with false positives rather than missing true negative.

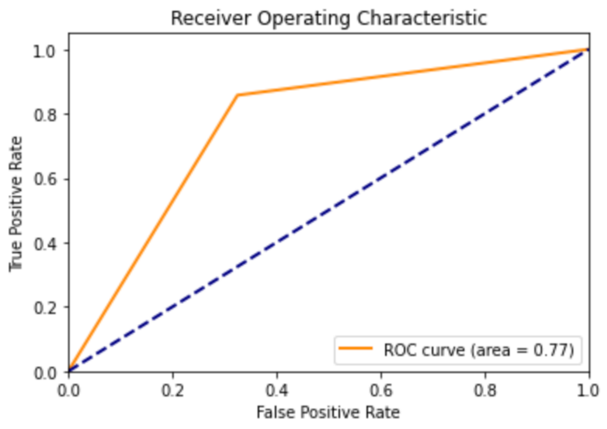
```
              precision    recall  f1-score   support

           0       0.98      0.73      0.84      1456
           1       0.13      0.77      0.22        77

    accuracy                           0.73      1533
   macro avg       0.56      0.75      0.53      1533
weighted avg       0.94      0.73      0.81      1533

Accuracy: 0.733
Precision: 0.131
Recall: 0.766
F1-score: 0.223
```



We can see from this first model that accuracy and recall are not terrible. Again, precision and the F1-score will be lower as we are scoring these models based off of the recall mostly with respect to the accuracy. The ROC is at a good AUC score of 0.75.

For the next MLP Classifier I wanted to relook at RFE and let it reduce the features that work best for the models performance for recall and accuracy. I am using sklearn ensemble to utilize Random Forest Classifier as an estimator to the RFECV feature selection. After getting the selected x train and x test, I am once again passing these variables to MLP Classifier to let GridSearchCV find optimal hyper parameters for the MLP model. It is worth noting that I started using a negative 1 for n jobs to utilize all processors on my computer as it started becoming computationally heavy and why in code some parameter spaces are commented out once the best parameters are found.

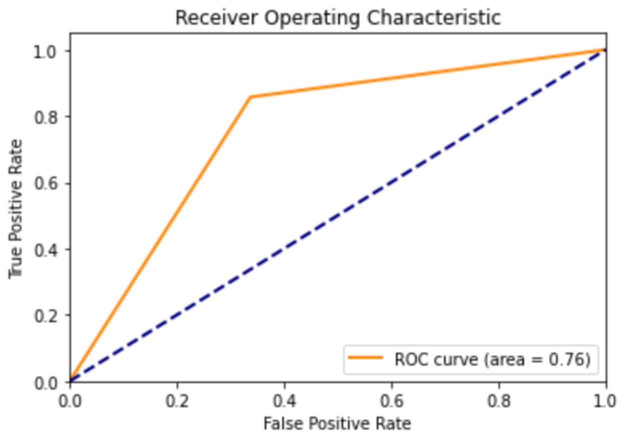
```
              precision    recall  f1-score   support

           0       0.99      0.68      0.80      1456
           1       0.12      0.86      0.21        77

    accuracy                           0.68      1533
   macro avg       0.56      0.77      0.51      1533
weighted avg       0.95      0.68      0.77      1533

Accuracy: 0.684
Precision: 0.122
Recall: 0.857
F1-score: 0.214
```



Allowing RFE to reduce to the most useful features increaded the recall 9%, but lowered the accuracy to 68%. This raised the ROC Curve to an AUC to 0.77.

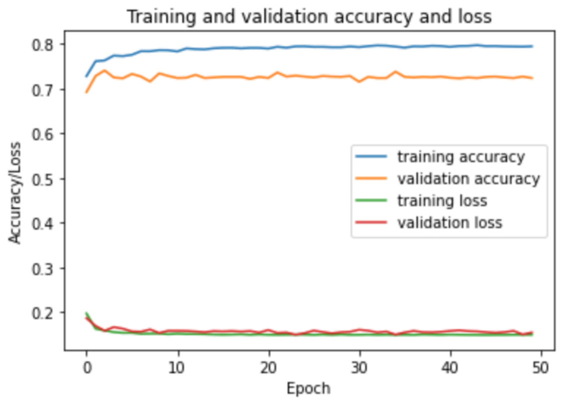
The last MLP Classifier I want to explore was backed by implementing Principal Component Analysis. The idea behind using PCA, even for a dataset with not a lot of variables was to reduce noise and feed it the variables we have to see if there are any combinations that will amplify results. To build this specific computation I used the sklearn pipeline to build a two-step process of implementing PCA with MLP. This again used GridSearchCV to find all optimal parameters that were in the parameter space.

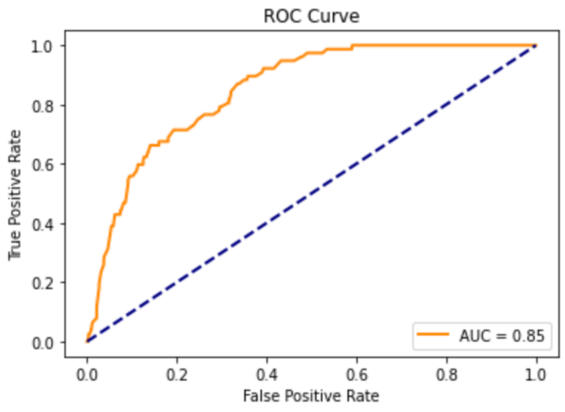
```
              precision    recall  f1-score   support

           0       0.99      0.66      0.79      1456
           1       0.12      0.86      0.21        77

    accuracy                           0.67      1533
   macro avg       0.55      0.76      0.50      1533
weighted avg       0.95      0.67      0.76      1533

Accuracy: 0.673
Precision: 0.118
Recall: 0.857
F1-score: 0.208
```



The last sklearn model perfromed pretty similar to the RFE model, which makes me believe that for the future the feature engineering needs to be optimized. The recall was again at 86% the accuracy was a percent lower at 67%. Thus bringing down the ROC curve to 0.76.

Along with MLP Classifier I built and optimized the Keras model. I experimented by starting simple and introduce more complexity as recall and accuracy improve. Both my implementations of Keras were 2 layer neural networks with a Droput layer with rate of 0.2 between the layers. Both models used the relu activation function in the first layer and sigmoid in the output layer. The first Keras model had 30 neurons in the first layer and 1 neuron in the outer layer. For this model I used MSE as the loss function and adam as my optimizer.

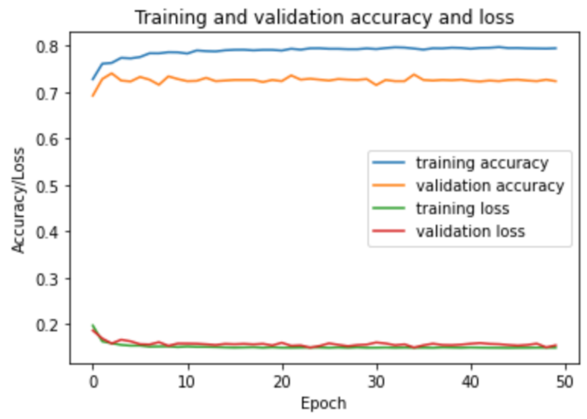


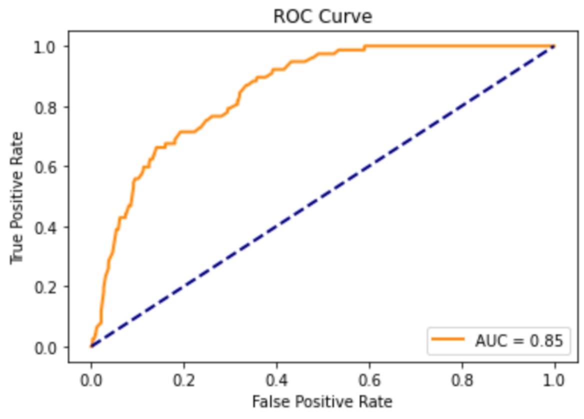


```
              precision    recall  f1-score   support

           0       0.98      0.72      0.83      1456
           1       0.13      0.77      0.22        77

    accuracy                           0.72      1533
   macro avg       0.56      0.74      0.53      1533
weighted avg       0.94      0.72      0.80      1533

Accuracy: 0.724
Precision: 0.127
Recall: 0.766
F1-score: 0.218
```

The Keras model performed relatively well. We can see from the training and test accuracy are decently close. I would like to see it closer together like the loss lines. The model had 77% recall and 72% accuracy. The ROC curve had the highest AUC score at 0.85.

The second implementation of the Keras model had 256 neurons in the first layer and 1 neuron in the outer layer. For this attempt I used adam optimizer again but lowered the rate from 0.001 to 0.0001 and used binary entropy as the loss function.





```
              precision    recall  f1-score   support

           0       0.98      0.73      0.84      1456
           1       0.13      0.77      0.23        77

    accuracy                           0.74      1533
   macro avg       0.56      0.75      0.53      1533
weighted avg       0.94      0.74      0.81      1533

Accuracy: 0.736
Precision: 0.132
Recall: 0.766
F1-score: 0.226
```

The last Keras model was slightly better than the first version by having an improved accuracy at 74% with recall being the same at 77%. The ROC is the same at 0.85.

Summary and Conclusions:

The project was much more difficult than I had anticipated. We had a massive mistake in the implementation of SMOTE at first that led to extreme overfitting. After resolving the issue, it started showing cracks in our models. My models especially but starting off simple and introducing complexity was a great way improve these neural network models. In the future this project needed better feature engineering, more data points, and more variables. We are using recall as the primary scoring metric, but this also allows for many false positives which is better than true negatives in the case of stroke. However, in the real world setting we also do not want to predict stroke on truly healthy people meaning I would like to increase the precision score. My best model for recall was using RFE and the MLP Classifier. I also believe that the Keras model could be wildly enhanced and could be the best neural network model over the sklearn MLP Classifier due to the modularity of building it from scratch.

Code from the Internet: 38%

Resources:

https://chat.openai.com/

https://towardsdatascience.com/5-smote-techniques-for-oversampling-your-imbalance-data-b8155bdbe2b5

https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a https://nicolo-albanese.medium.com/tips-on-principal-component-analysis-7116265971ad

https://scikit-learn.org/stable/tutorial/statistical_inference/putting_together.html

https://towardsdatascience.com/powerful-feature-selection-with-recursive-feature-elimination-rfe-of-sklearn-23efb2cdb54e