Assignment 3:

#

Double-click to write down your name and surname.

Name: Josh Bryden

Honour Pledge

Declaration:

1. I declare that this assessment item is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere or previously, or produced independently of this course (e.g. for a third party such as your place of employment) and acknowledge that the assessor of this item may, for the purpose of assessing this item:

```
a. Reproduce this assessment item and provide a copy to another
member of the University; and/or

b. Communicate a copy of this assessment item to a plagiarism
checking service (which may then retain a copy of the assessment
item on its database for the purpose of future plagiarism
checking).
```

#

#

#

2: I have a backup copy of the assessment.

#

#

1. Introduction

Predictive Algorithm to Forecast the Onset of Diabetes Mellitus in the Pima Native American population.

Task: The task is to build a predictive algorithm using the techniques we learned in this course.

Objective: To assess the role of machine learning-based automated software for forecasting diabetes mellitus (DM) by using 8 variables that have been found to be significant risk factors for DM amongst Pimas and other populations.

Question: Is it possible to build a predictive algorithm to forecast whether a patient will develop DM based on the data available?

Deadline: 22-Nov 2020 23.59 h

Study Population as described in the paper "Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus" by Jack W. Smith et al.:
"The data used in this study were from the Pima Indian population near Phoenix, Arizona. That population has been under continuous study since 1965 by the National Institute of Diabetes and Digestive and Kidney Diseases because of its high incidence rate of diabetes.

Each community resident over 5 years of age was asked to undergo a standardized examination every two years, which included an oral glucose tolerance test.

Diabetes was diagnosed according to World Health Organization Criteria; that is, if the 2-hour post load plasma glucose was at least 200 mg/dl (11.1 mmol/l) at any survey examination or if the Indian Health Service Hospital serving the community found a glucose concentration of at least 200 mg/dl during the course of routine medical care.

Eight variables were chosen to form the basis for forecasting the onset of diabetes within five years in Pima Indian women. Those variables were chosen because they have been found to be significant risk factors for diabetes among Pimas or other populations."

Context: Doctors have developed an intervention that could prevent or delay DM if patients are treated prior to disease onset but it is quite expensive, and they can't treat everyone. They would like to try to predict who will develop the disease within the next 5 years so that they can target their intervention to those most at risk.

Case Selection as described in the paper "Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus" by Jack W. Smith et al.

"Diabetes was defined as a plasma glucose concentration greater than 200 mg/dl two hours following the ingestion of 75 g of a carbohydrate solution. Cases were drawn from the pool of examinations which met the following criteria:

1. The subject was female.
2. The subject was >= 21 years of age at the time of the index examination. An index examination refers to the study that was chosen for use in this model. It does not necessarily correspond to the chronologically first examination for this subject.
3. Only one examination was selected per subject. That examination was one that revealed a nondiabetic GTT and met one of the following two criteria:

   a. Diabetes was diagnosed within five years of the examination, OR

   b. A GTT performed five or more years later failed to reveal diabetes mellitus.
4. If diabetes occurred within one year of an examination, that examination was excluded from the study to remove from the forecasting model those cases that were potentially easier to forecast. In 75% of the excluded examinations, DM was diagnosed within six months."

Data: Using these criteria, 1,000 examinations were selected. This dataset is not the original one. **This dataset has been created specifically for this assignment.**

Our hypothesis is that Machine Learning algorithms could learn to forecast whether a given individual would develop DM within five years based on the value of the eight input variables. Binary variable "Outcome": 1/Yes, 0/No

2. Instructions

These are the specifications (step 0 of our machine learning work-flow- Question):

1. We are going to deliver one predictive model to predict diabetes
2. You will design several machine learning algorithms, choose one, and give a rationale explaining why you choose that algorithm. **Take into account that the time to train the classifiers can be long. You might need to find workarounds to solve this problem if the time is too long.**
3. Use the evaluation metrics that you consider useful to evaluate your models.
4. Write an introduction for each section and a small paragraph before each block of Python code with the description of what the blocks do; explanations must be clear; no “sanity checks” (although these checks are encouraged during the construction of the algorithms);

   Only the final version of the document should be submitted.

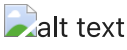
   Print clear labels in the printed results and explain in short and concise sentences the steps that you followed. Justify your decisions and why they were made.

   Explain the rationale behind each decision. **Document the code**

   Please write all the steps that you will follow during your analysis. I would like to read the rationale behind all of your decisions.

   In addition, it is important to adopt the practice of commenting and documenting your code, as you will most likely later work in teams developing such algorithms.

1. Format: Jupyter Notebook and **PDF** of the Jupyter Notebook.
2. Programming Language: Python.
3. You can be as creative as you wish.
4. You might have to use an iterative approach in which you must come back to a previous point depending on the analysis.
5. The assignment will be mark using the rubric provided below.



Question 1: Data dictionary: create a complete data dictionary with clear explanations of the

meaning and units of each variable. (5 marks)

#

Data Dictionary:

Glucose - The plasma glucose concentraion at 2 hours post an oral glucose tolerance test (GTT), measured in mg/dl.

Outcome - A binary variable representing whether or not the patient was diagnosed with diatbetes mellitus (DM) within 5 years. 1 representing the patient was diagnosed with DM and 0 representing no diagnosis.

Diabetes Pedigree Function - A function used to provide a basic history of DM in relatives of the patient. Uses information from parents, grandparents, siblings, aunts, uncles and cousins. It aims to give the genetic influence of both relatives with DM and those without on the risk of the patient in question delveloping DM.

Age - The age of the patient in years.

Pregnancies - The number of times the patient has been pregnant.

Skin Thickness - The thickness of the skin on the triceps muscle (posterior of forearm) measured in mm.

Blood Pressue - The diastolic blood pressure of the patient, measured in mmHg.

BMI - The body mass index (BMI) of the patient, measured as the weight of the paitent in kg, divided by the height of the patient, squared (Weight/Height^2).

Insulin - Serum insulin levels 2 hours post GTT. Measured in µU/mL.

#

Question 2: Compute basic general statistics of the data. Clean the data if needed. (5 marks)

#

The below chunk of code contains the required libraries for this notebook. We predominatly use pandas for reading our data (csv file), matplotlib and seaborn for visualisations, and sklearn and tensorflow.keras for building our machine learning models. This chunk also contains the seeds for this notebook to ensure reproducability.

#

In [122…

```
# Install required libraries for reading in data and visualisation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import train test split function:
from sklearn.model_selection import train_test_split
# Import the standard scaler function:
```

```
from sklearn.preprocessing import StandardScaler
# Import the logistic regression model:
from sklearn.linear_model import LogisticRegression
# Import the pipeline function:
from sklearn.pipeline import Pipeline
# Import the grid search CV class:
from sklearn.model_selection import GridSearchCV
# Import the RandomizesearchCV function:
from sklearn.model_selection import RandomizedSearchCV
# Import the gradient boosting classifier
from sklearn.ensemble import GradientBoostingClassifier

# Import tf.keras
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.layers import Flatten
from tensorflow.keras import backend as K
from tensorflow.keras import optimizers
# Import pacakges to link sklearn with tf.keras
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier


# Import the accuracy_score function:
from sklearn.metrics import accuracy_score
# Import the classification report function:
from sklearn.metrics import classification_report
# Import the confusion matrix function:
from sklearn.metrics import confusion_matrix


# Set global seed to make sure results are reproducible:
np.random.seed(1111)
# Set tensorflow seed
from tensorflow import set_random_seed
set_random_seed(1111)
```

The below code loads in our dataset from the csv file 'diabetes_final.csv' and ouputs the first 5 rows of the data for examination

In [123…

```
# load in data
data = pd.read_csv('data/diabetes_final.csv')
#head of data
data.head()
```

Out[123…

	Glucose	Outcome	DiabetesPedigreeFunction	Age	Pregnancies	SkinThickness	BloodPres
0	105	0	0.260000	24.0	0.0	41.0	
1	123	0	0.498000	30.0	2.0	28.0	
2	116	0	0.089000	25.0	3.0	26.0	
3	106	0	0.100000	31.0	1.0	35.0	
4	133	1	0.341471	42.0	8.0	11.0	

The code chunk below checks for the common missing values types (Na and null) within our dataset by columns. We confirm that there is no NA or null values present.

In [124…

```
# check for missing data
print(data.isna().any())
print()
print(data.isnull().any())
```

```
Glucose                     False
Outcome                     False
DiabetesPedigreeFunction    False
Age                         False
Pregnancies                 False
SkinThickness               False
BloodPressure               False
BMI                         False
Insulin                     False
dtype: bool

Glucose                     False
Outcome                     False
DiabetesPedigreeFunction    False
Age                         False
Pregnancies                 False
SkinThickness               False
BloodPressure               False
BMI                         False
Insulin                     False
dtype: bool
```

To quickly examine the datasets basic statistical features, we call the describe() function to obtain these statisitical measure by column.

In [125…
```
# use describe to generate basic stats for each variable
data.describe()
```

Out[125…

	Glucose	Outcome	DiabetesPedigreeFunction	Age	Pregnancies	SkinT
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	100
mean	121.375000	0.331000	0.471447	33.317000	4.144000	2
std	31.540342	0.470809	0.329022	11.916313	3.432644	1
min	0.000000	0.000000	0.084000	21.000000	0.000000	
25%	99.000000	0.000000	0.254000	24.000000	1.000000	1
50%	117.000000	0.000000	0.370000	29.000000	3.000000	2
75%	142.250000	1.000000	0.605001	41.000000	6.000000	3
max	199.000000	1.000000	2.420000	69.000000	19.000000	7

The above shows us that the minimum values for some of the columns (Glucose, Skin thickness, Blood pressure, BMI and Insulin) is zero. Hence there appears to be missing data for these rows (and not in the form of na or nulls). Lets investigate...

In [126…
```
for column in data.columns:
    print("{} has {} rows with zero values".format(column, sum(data[column]==
```

```
Glucose has 2 rows with zero values
Outcome has 669 rows with zero values
DiabetesPedigreeFunction has 0 rows with zero values
Age has 0 rows with zero values
Pregnancies has 102 rows with zero values
SkinThickness has 204 rows with zero values
BloodPressure has 31 rows with zero values
BMI has 17 rows with zero values
Insulin has 348 rows with zero values
```

As we can see above there is 2 rows missing in Glucose (which may or may not be correct), 204 rows in Skin Thickness, 31 rows in BP, 17 in BMI and 348 in Insulin. Due to the large number of missing values, we would lose around 1/3 of the data if we deleted these rows so

at the cost of reduced variance we shall make these values the mean of their column, excluding the zeros.

The below chunk of code replaces the zeros found in the column list 'replace_list' with the numpy value 'NaN' before replacing the 'NaN' value with the mean of the column. Due to uncertantly around the two zero Glucose values, these have been left alone.

In [127…
```
# create list of column names to replace - note we will leave the two 0 gluco
replace_list = ['SkinThickness', 'BloodPressure', 'BMI', 'Insulin']
# loop over list and change 0 values to na, then change na to the mean of tha
for i in replace_list:
    data[i] = data[i].replace(0,np.nan)
    data[i] = data[i].replace(np.nan, data[i].mean())
```

To check the balance/ imbalance of the dataset we perform a value_counts function on the Outcome column to see the number of 0 and 1's are present.

In [128…
```
# Determine the number of patients with and without DM in the dataset (also h
data['Outcome'].value_counts()
```

Out[128…
```
0    669
1    331
Name: Outcome, dtype: int64
```

We see that there is around a 7:3 ratio of imbalance in the dataset (no DM: to DM)

Question 3: Visualise all the features. Clean the data if needed. (10 marks)

#

To visualise each column of the dataset as a histogram, we define the function create_plot that takes the column of the dataset and the units of said column to output a histogram of each column of data to examine its distribution of variables.

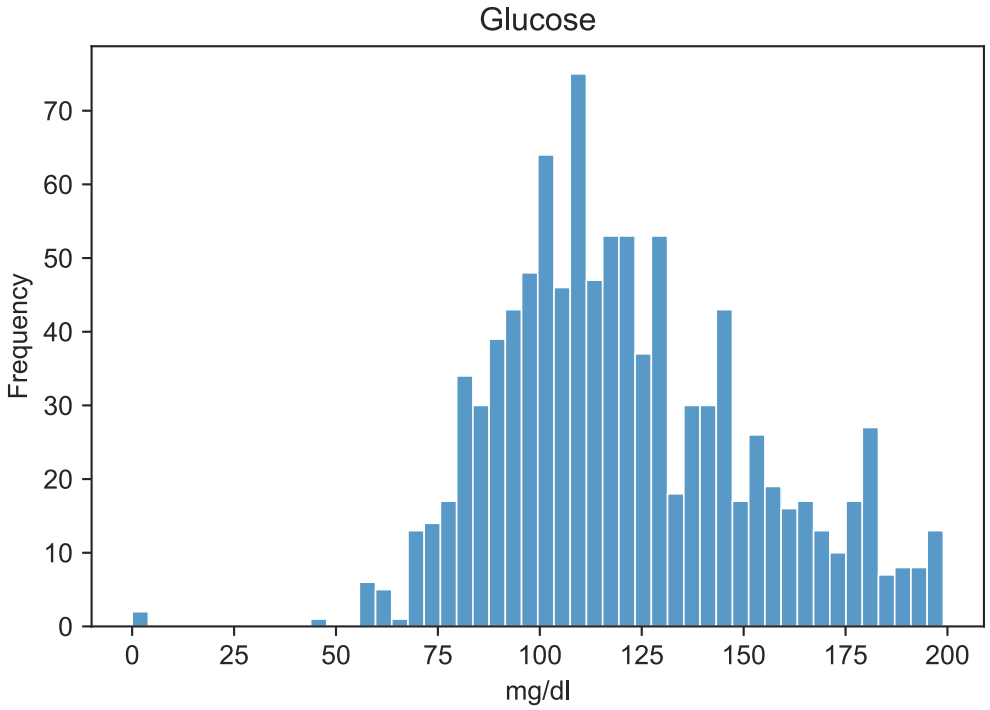
#

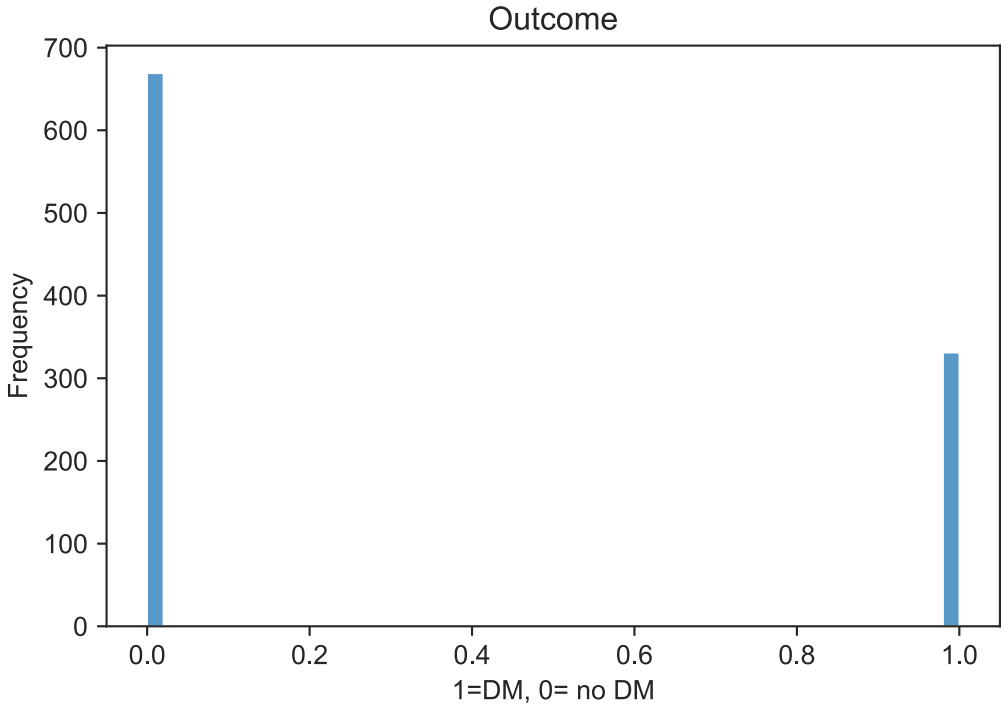
In [129…
```
# define create_plot function to create a plot with matplotlib.pyplot
def create_plot(column, units, ylabel="Frequency"):
    """
    Creates a histogram plot of the column of interest with xlablel as units
    and ylabel as the frequency. Returns the plot
    """
    ax = sns.histplot(data[column], bins=50)
    ax.set_title(column)
    ax.set_ylabel(ylabel)
    ax.set_xlabel(units)
    plt.show()
```

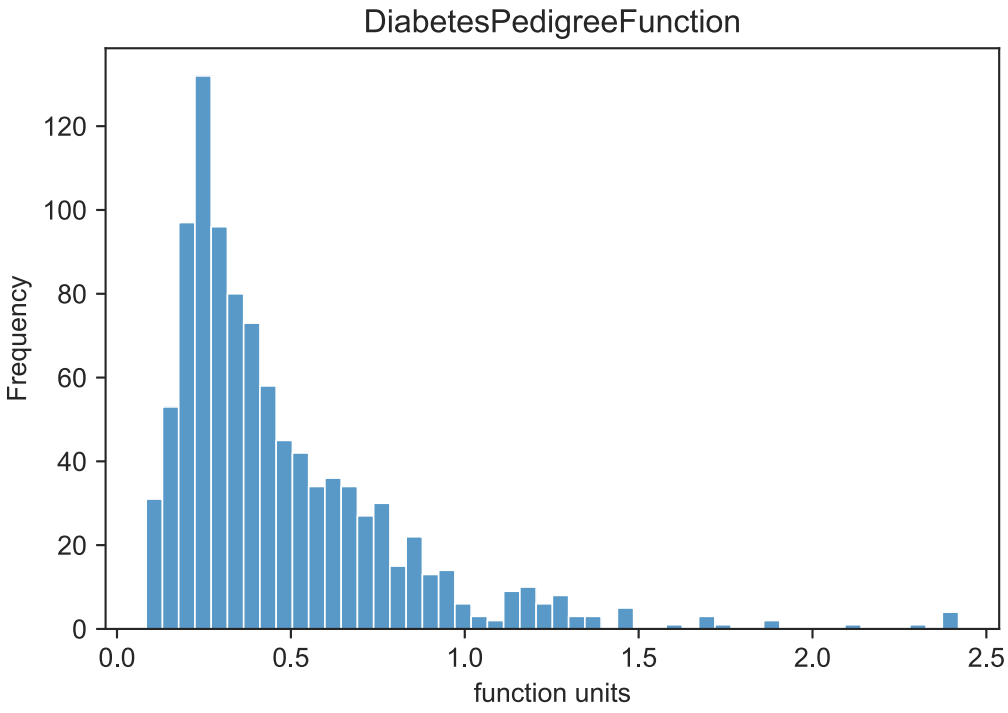
We then call upon our create_plot function with each column and the units of said column to examine its distribution.

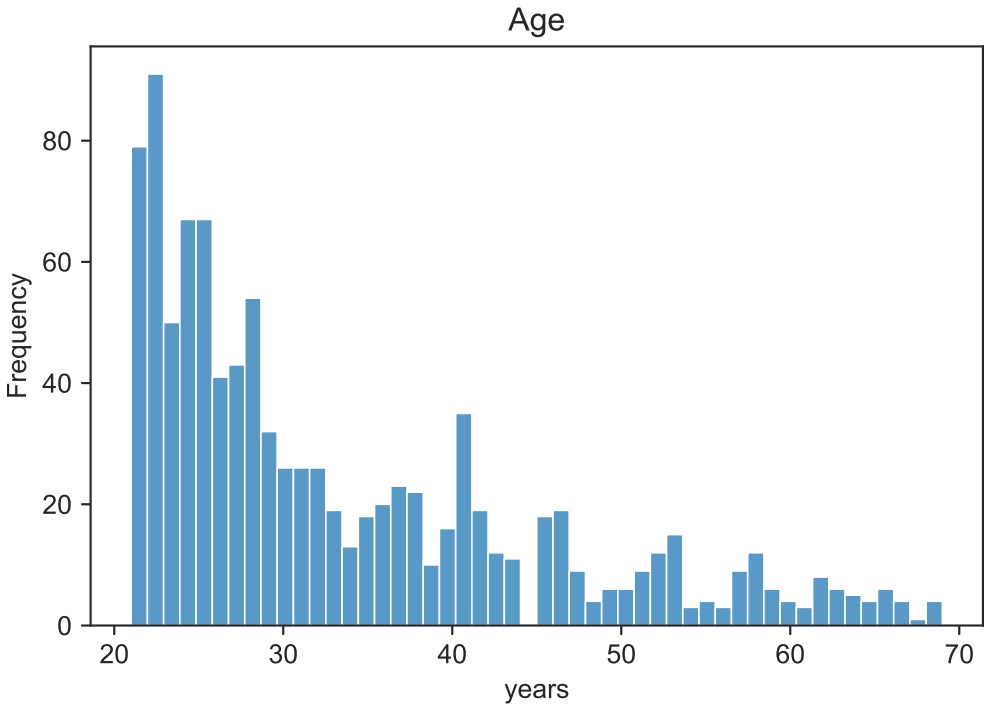
In [130…
```
#create plots for each variable with function defined above
create_plot(column='Glucose',units='mg/dl')
create_plot(column='Outcome',units='1=DM, 0= no DM')
create_plot(column='DiabetesPedigreeFunction',units='function units')
create_plot(column='Age',units='years')
create_plot(column='Pregnancies',units='number of pregnancies')
create_plot(column='SkinThickness',units='mm')
create_plot(column='BloodPressure',units='mmHg')
```

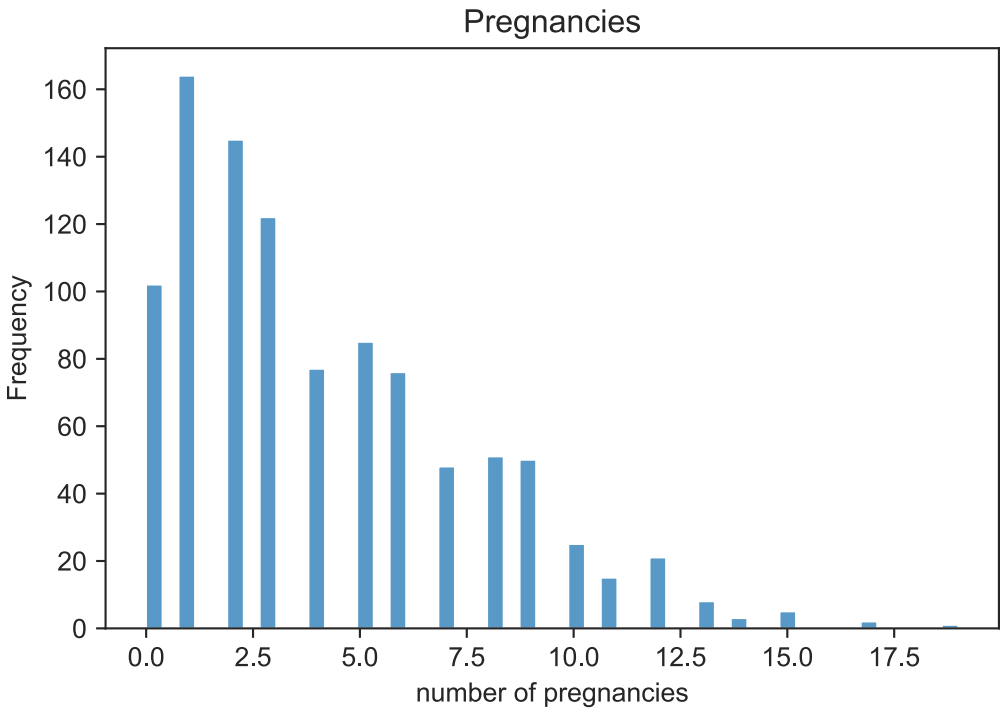
```
create_plot(column='BMI',units='mmHg')
create_plot(column='Insulin',units='µU/mL')
```

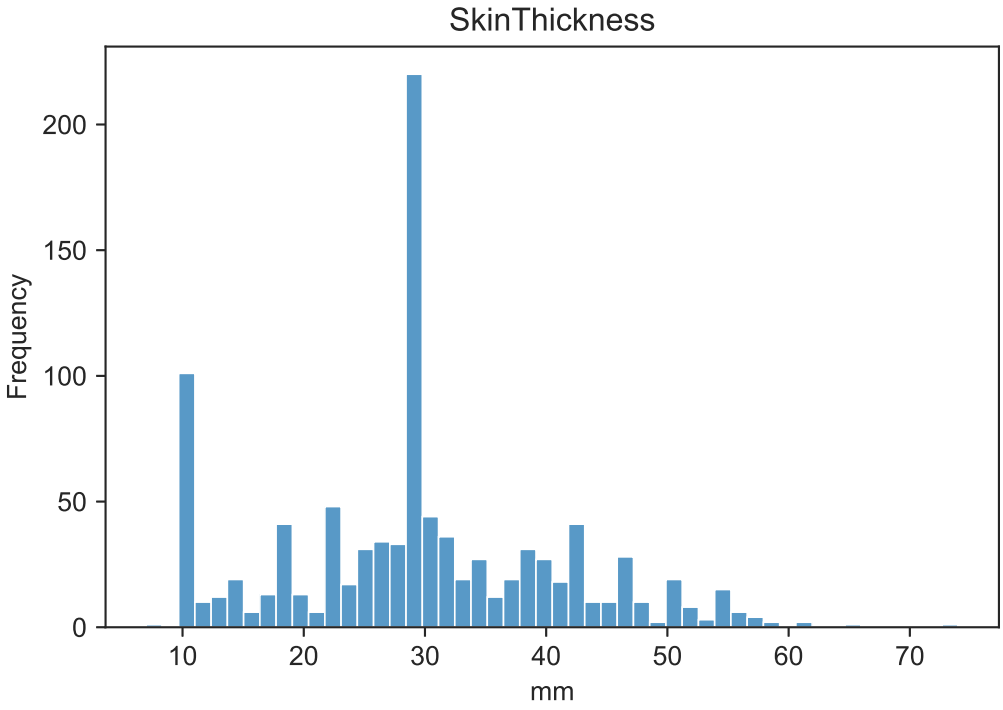


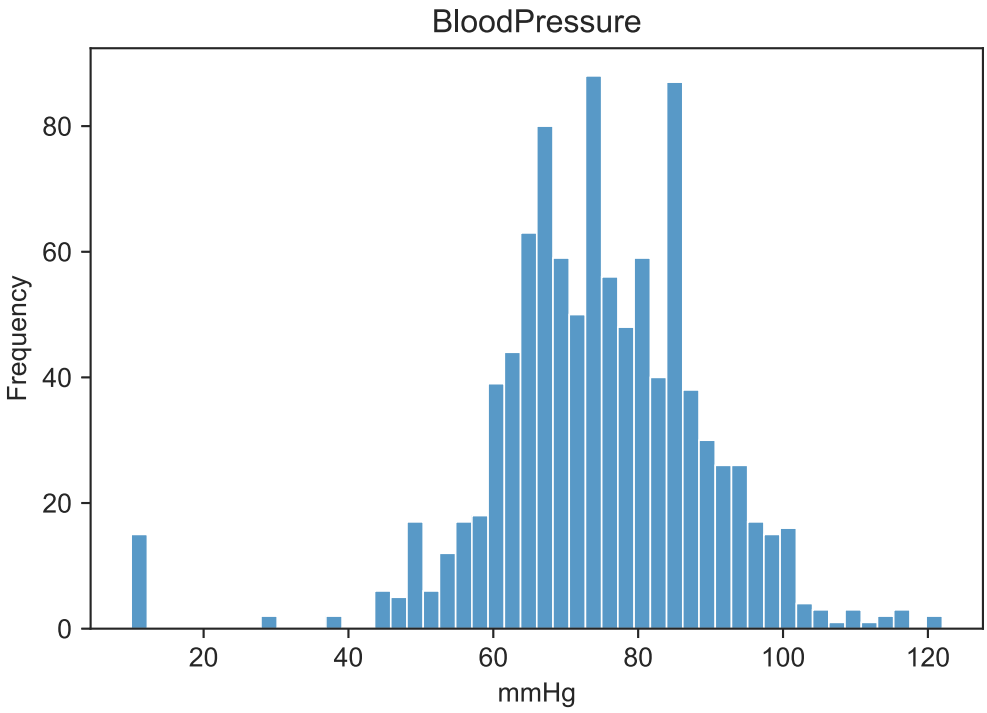


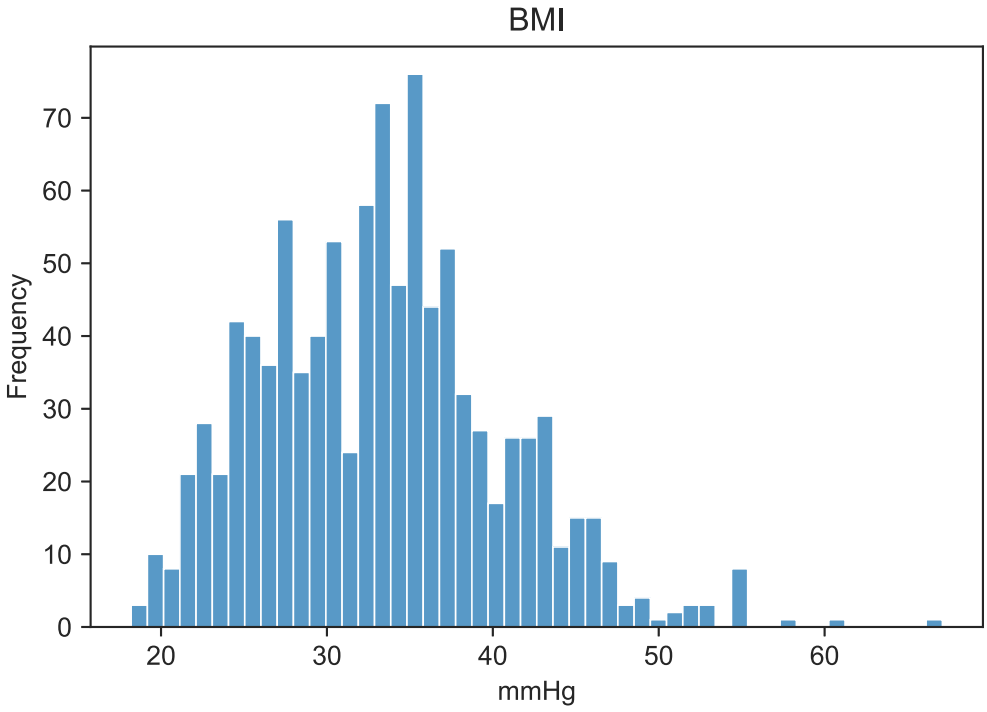

DiabetesPedigreeFunction
Frequency
0
20
40
60
80
100
120
0.0
0.5
1.0
1.5
2.0
2.5
function units

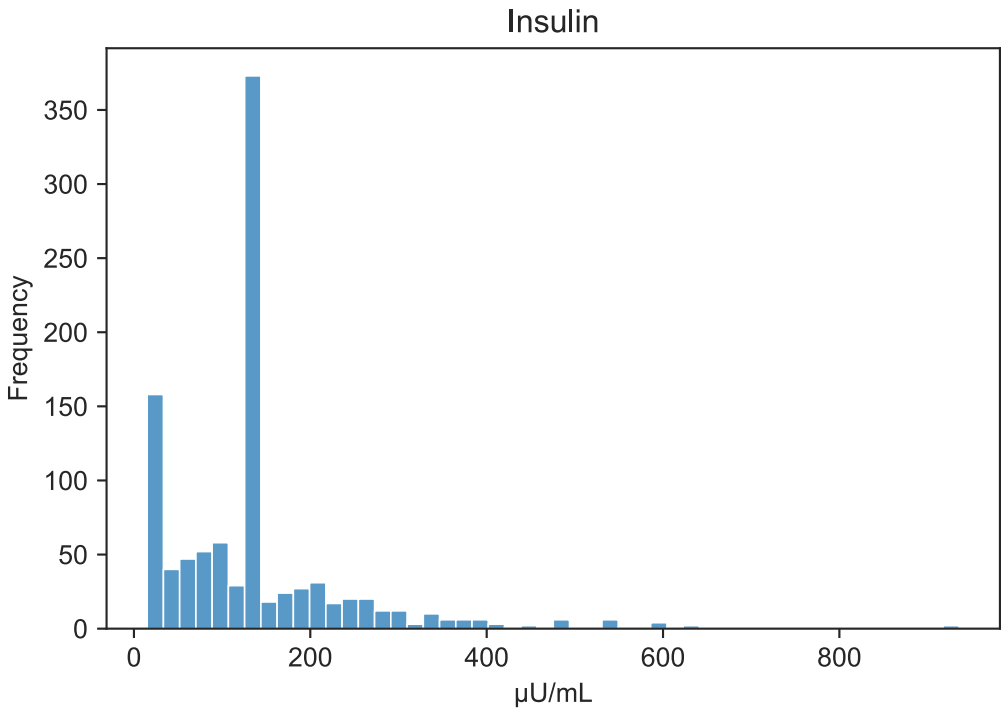

Age
Frequency
0
20
40
60
80
20
30
40
50
60
70
years


Pregnancies
Frequency
160
140
120
100
80
60
40
20
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
number of pregnancies


SkinThickness
Frequency
200
150
100
50
0
10
20
30
40
50
60
70
mm


BloodPressure
Frequency
0
20
40
60
80
20
40
60
80
100
120
mmHg


BMI
Frequency
0
10
20
30
40
50
60
70
20
30
40
50
60
mmHg

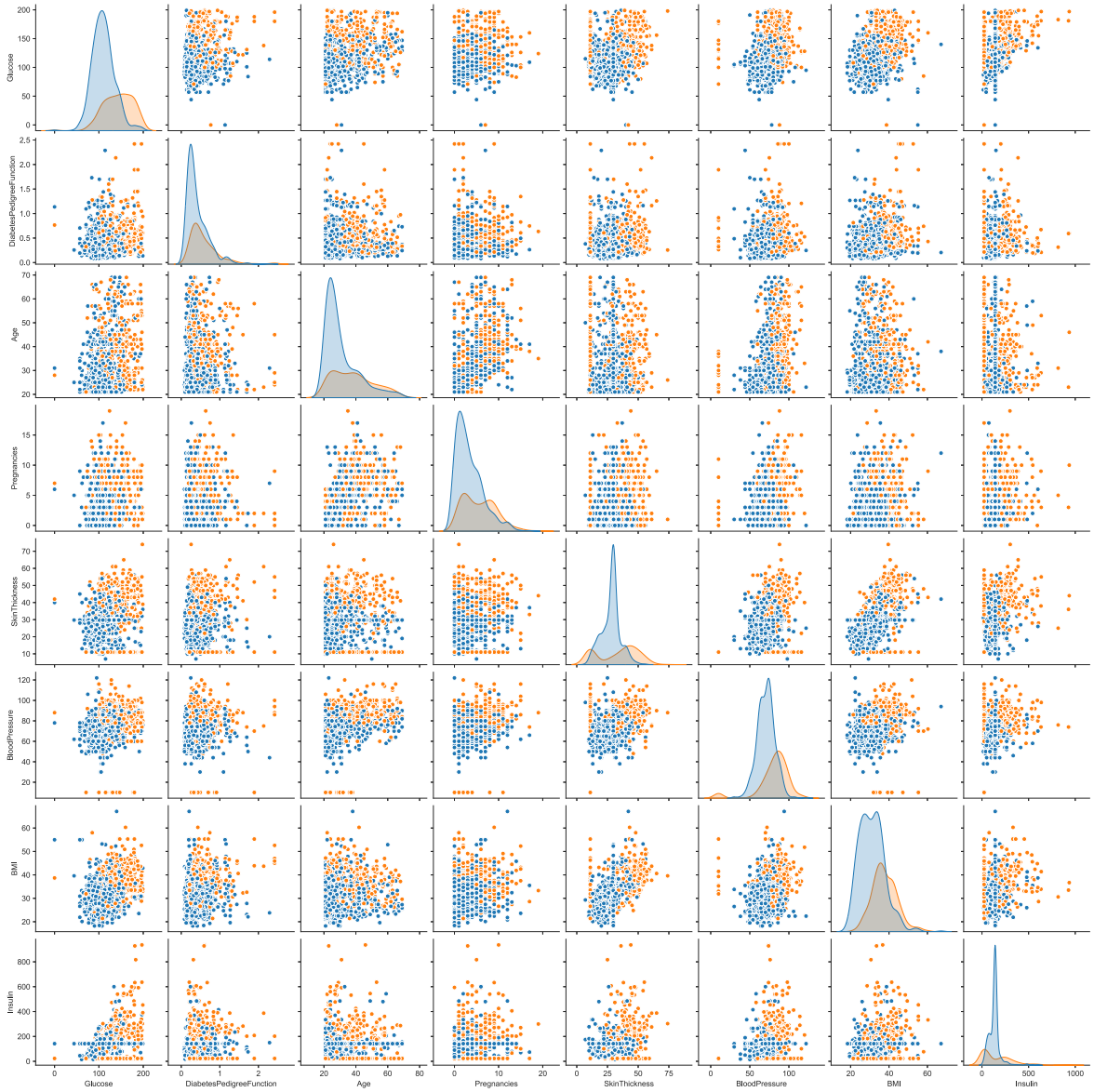


From the above plots we can see that our cleaning of the data (changing zeros to the mean) is ever present in our columns in which this occured. Noteably seen in the Insulin and Skin Thickness columns where one of the bins has a frequency in the hundreds. Interestingly, we see that the dataset contains higher levels of younger people and that the Glucose levels are positively skewed.

To investigate some of these variables closer, we call upon a seaborn pairplot (matrix) to see how these variables interact with each other by Outcome (with and without DM).

In [131…

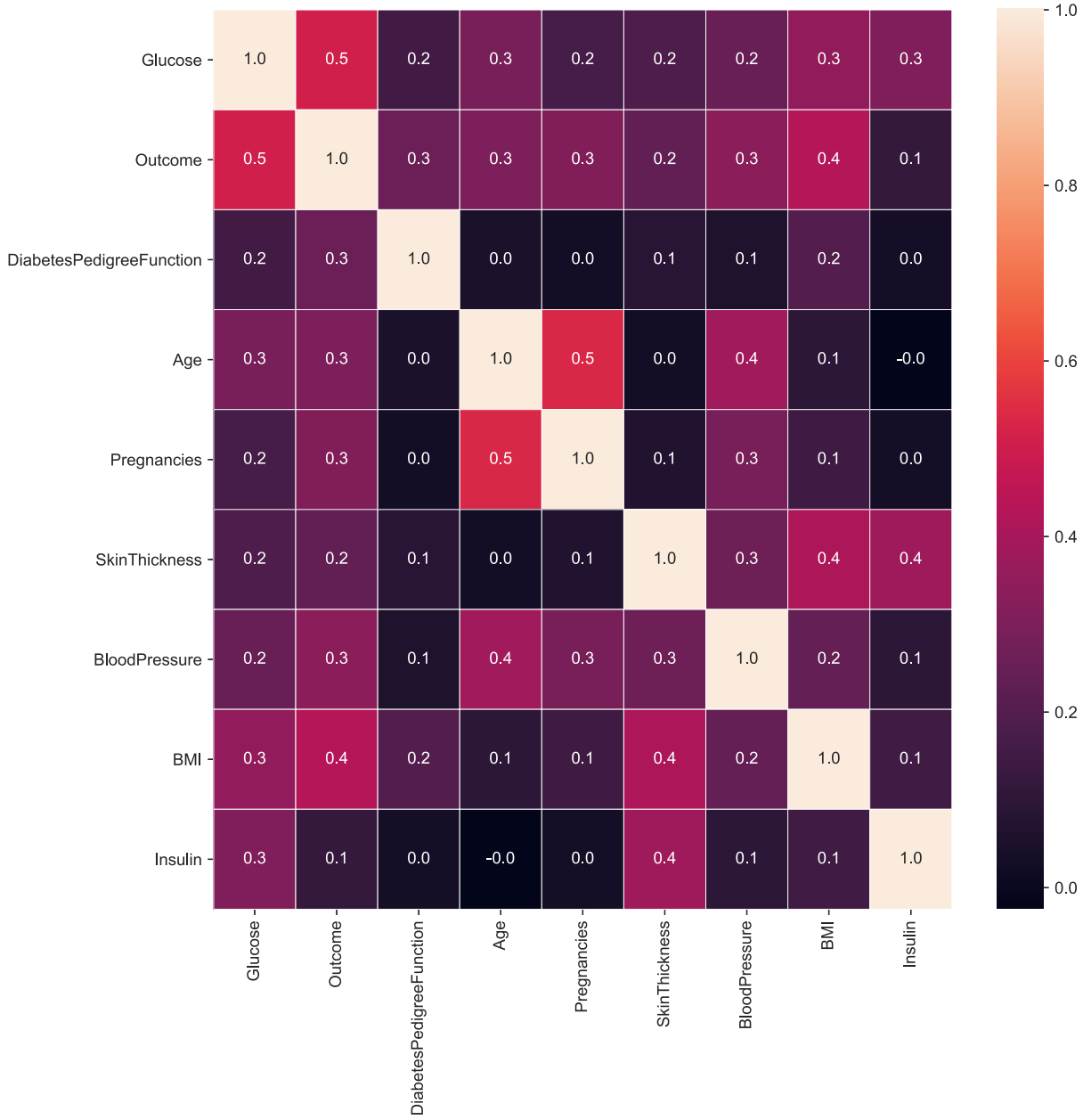
```
# create a pairplot
sns.set_style("ticks")
sns.pairplot(data, hue='Outcome')
plt.show()
```



From the pairplot above we see little obvious correlation between the variables. In order to investigate further we call upon a confusion matrix below to see a numerical output.

In [132…
```
# Create a correlation matrix
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data.corr(), annot=True, linewidths=0.5, fmt='0.1f', ax=ax)
```

Out[132… <AxesSubplot:>



From the confusion matrix above we can see a correlation of 0.5 between the variables of Outcome and Glucose and Age and Pregnancies. The first significant correlation appears to show the biological nature of the glucose test, in that a higher glucose level, should in theory lead to a higher Outcome (higher being the value of 1, meaning the patient has DM). The second correlation whilst not of relvence to the research question, is understandable due to the fact that a younger woman may have had less time to have a child, hence the correlation with age and number of Pregnancies.

Question 4: Choose hyper-parameters and train the next 3 ML models using GridSearchCV

1. Logistic Regression,

2. Gradient Boosted Tree and

3. Dense Neural Network.

(30 marks)

Before we can begin creating our machine learning models, we must split our data up into training and test data. The below chunk of code calls train_test_split fuction to split our data up into 80% training data and 20% test data. The function then splits off the outcome column into y_test and y_train to seperate the Outcome from the predictors.

Initial data prep

In [133…
```
# Divide dataset into X (input variables) and y (output variables)
X = data.drop(axis=1, columns=['Outcome'])
y = data['Outcome']

# Split X and y into 80% training data and 20% test data
# Set random state for reproducability and stratify by y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand
```

#

The below code defines our logisitc regression model and a scaler (as logistic regression is distance based) before creating a pipeline to scale and define our model. THe code then defines our hyperparameter grid over which we call upon GridSearchCV to perofrm an exhaustive search over our parameter grid to find the optimal hyperparametes for our model.

#

Logistic Regression

In [134…
```
# Code here for Logistic Regression
# Set global seed to make sure results are reproducible:
np.random.seed(1111)
# Define the logistic regression model
log_reg = LogisticRegression()

# Scale features for logistic regression
standard_scalar = StandardScaler()

# Create a pipeline to use the standard scalar and our log_reg model
pipe_logistic = Pipeline([('Transform', standard_scalar),('Estimator', log_re

# Define our hyperparameter grid for our log_reg model (parameters can be fou
parameter_grid_logistic = {'Estimator__C':[0.001, 0.01, 0.1, 1, 10, 100],
                           'Estimator__penalty':['l1','l2'],
                           'Estimator__class_weight':['None', 'balanced',{0:0
                           'Estimator__solver':['liblinear','lbfgs','newton-c
```

The code below fits out parmeter grid to GridSearch and outputs the best F1 score and parameters found. We use the F1 score here as it is the harmonic mean between precision and recall for our model, hence it contains more information than the accuracy alone.

In [135…
```
# Assign GridSearchCV to our pipline and parameter grid
grid_search_logistic = GridSearchCV(pipe_logistic, parameter_grid_logistic, c

# Fit the grid search and pipeline
grid_search_logistic.fit(X_train, y_train)
```

```
# Print the best parameters found and best score found
print("Best hyper-parameters for Logistic Regression: {}".format(grid_search_
print("Best cross-validation average f1-score for Logistic Regression: {:.3f}
```

```
Best hyper-parameters for Logistic Regression: {'Estimator__C': 10, 'Estimator
__class_weight': 'None', 'Estimator__penalty': 'l1', 'Estimator__solver': 'sag
a'}
Best cross-validation average f1-score for Logistic Regression: 0.832
```

Gradient Boosted Tree

The code chunk below defines our gradient boosted tree model and our parameter grid. We have chosen to use RandomSearchCV initally here to cut down on the compute time, by narrowing the range of parameters for which to exhaustively GridSearch over. The code will output the F1 score and optimal parameters found.

Random Search CV

In [136…

```
# Code here for Gradient Boosted Tree
# Set global seed to make sure results are reproducible:
np.random.seed(1111)
# Define out Gradient Boosted Tree model with random state set to zero for re
gbt = GradientBoostingClassifier(random_state=0)

# Define our hyperparameter grid for our gbt model - we will use RandomizedSe
random_parameter_grid_gbt = {'n_estimators': [10, 50, 100, 200, 500, 1000, 12
                    'learning_rate': [0.1,0.2,0.3,0.4,0.5],
                    'min_samples_split': [int(x) for x in np.linspace(start=2
                    'min_samples_leaf': [int(x) for x in np.linspace(start=2,
                    'max_depth': [int(x) for x in np.linspace(start=2, stop=3
                    'max_features': ['auto','sqrt','log2']}



# Define our Random Search CV and use F1 score
random_search_gbt = RandomizedSearchCV(gbt, random_parameter_grid_gbt, cv=5, 
# Fit our Random search CV
random_search_gbt.fit(X_train, y_train)

# Print the best parameters found and best score found
print("Best hyper-parameters for Gradient Boosted Tree: {}".format(random_sea
print("Best cross-validation average f1-score for Gradient Boosted Tree: {:.3
```

```
Best hyper-parameters for Gradient Boosted Tree: {'n_estimators': 100, 'min_sa
mples_split': 6, 'min_samples_leaf': 10, 'max_features': 'sqrt', 'max_depth':
4, 'learning_rate': 0.1}
Best cross-validation average f1-score for Gradient Boosted Tree: 0.942
```

After narrowing the parameter range, we perform a GridSearch over the narrowed parameters to find the optimal hyperparamters for the model. We then print the F1 score and best paramters found.

GridSearchCV

In [137…

```
# Define our hyperparameter grid for our gbt model
parameter_grid_gbt = {'n_estimators': [int(x) for x in np.linspace(start=10, 
                    'learning_rate': [0.05,0.1,0.15,0.2],
                    'min_samples_split': [int(x) for x in np.linspace(start=2
                    'min_samples_leaf': [int(x) for x in np.linspace(start=10
                    'max_depth': [int(x) for x in np.linspace(start=2, stop=1
                    'max_features': ['sqrt']}
```

```
# Define our gridsearch using for our gbt model (parameters found here: https
grid_search_gbt = GridSearchCV(gbt, parameter_grid_gbt, cv=5, scoring='f1_wei
grid_search_gbt.fit(X_train, y_train)

# Print the best parameters found and best score found
print("Best hyper-parameters for Gradient Boosted Tree: {}".format(grid_searc
print("Best cross-validation average f1-score for Gradient Boosted Tree: {:.3
```

```
Best hyper-parameters for Gradient Boosted Tree: {'learning_rate': 0.1, 'max_d
epth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 13, 'min_samples_split':
2, 'n_estimators': 200}
Best cross-validation average f1-score for Gradient Boosted Tree: 0.945
```

The code chunk below defines our nerual network model with keras and defines a function to build our Dense nerual network. As we are using keras and sklearn, we require a wrapper between the two, which takes a function of our neural network as a argument.

Dense Neural Network

Define a function to build our neural net, add in layers and a dropout term. This is required for the sklearn and keras wrapper.

In [138…

```
# Code here for Dense Neural Network - note our input is (1000,8) and output
# Set global seed to make sure results are reproducible:
np.random.seed(1111)
# Create a function to create neural net and add a dropout term
def dense_neural_net(DropoutL1):
    # Create our model
    our_model = tensorflow.keras.Sequential()
    # add 1st layer - note that our input shape is 8 due to the 8 variables
    our_model.add(Dense(8, activation='relu', input_shape=(8,),kernel_initial
    # add dropout term
    our_model.add(Dropout(DropoutL1))
    # add 2nd layer
    our_model.add(Dense(8, activation='relu',kernel_initializer='uniform'))
    # add output layer
    our_model.add(Dense(1, activation='sigmoid',kernel_initializer='uniform')
    # compile - note that tf..keras cannot use tf F1 score module, hence use
    our_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[
    our_model.summary()
    return our_model
```

Begin to build the model using the Keras Classifier that will allow us to perform a grid search on it, and pass our function dense_neural_net to it that will create our DNN

In [139…

```
dnn = KerasClassifier(build_fn=dense_neural_net)
```

GridSearchCV

Before performing our gridsearch, we must scale our data. We can use a pipeline from sklearn to scale and perfrom our gridsearch in one move as we cross validate within our training set k times to tune our parameters.

In [140…

```
# Define our scaler again
standard_scalar = StandardScaler()

# Define the pipine using the scalar and our DNN defined above with KerasClas
pipe_dnn = Pipeline([('Transform', standard_scalar),('Estimator', dnn)])

# Define our parameter gridba
parameter_grid_dnn = {'Estimator__epochs':[200, 250, 300, 350, 400],
              'Estimator__batch_size': [25, 50, 100, 200],
```

```
                'Estimator__DropoutL1':[0.1, 0.2, 0.3, 0.4]}

# Define our GridSearchCV
grid_search_dnn = GridSearchCV(estimator=pipe_dnn, param_grid=parameter_grid_

# Fit the model
grid_search_dnn.fit(X_train, y_train)

# Print the best parameters found and best score found
print("Best hyper-parameters for the Dense Neural Network: {}".format(grid_se
print("Best cross-validation average accuracy for the Dense Neural Network: {
```

```
5/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2999 - ac
c: 0.8863
Epoch 156/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2912 - ac
c: 0.8863
Epoch 157/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2857 - ac
c: 0.8950
Epoch 158/350
800/800 [==============================] - 0s 33us/sample - loss: 0.2893 - ac
c: 0.8850
Epoch 159/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2798 - ac
c: 0.8863
Epoch 160/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2868 - ac
c: 0.8850
Epoch 161/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2843 - ac
c: 0.8888
Epoch 162/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2836 - ac
c: 0.8900
Epoch 163/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2841 - ac
c: 0.8975
Epoch 164/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2830 - ac
c: 0.8913
Epoch 165/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2879 - ac
c: 0.8925
Epoch 166/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2709 - ac
c: 0.8988
Epoch 167/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2849 - ac
c: 0.8863
Epoch 168/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2889 - ac
c: 0.9013
Epoch 169/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2700 - ac
c: 0.9038
Epoch 170/350
800/800 [==============================] - 0s 32us/sample - loss: 0.2754 - ac
c: 0.9013
Epoch 171/350
800/800 [==============================] - 0s 61us/sample - loss: 0.2733 - ac
c: 0.8963
Epoch 172/350
800/800 [==============================] - 0s 56us/sample - loss: 0.2866 - ac
c: 0.9025
Epoch 173/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2779 - ac
c: 0.8988
```

```
Epoch 174/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2791 - ac
c: 0.8975
Epoch 175/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2869 - ac
c: 0.8888
Epoch 176/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2905 - ac
c: 0.8800
Epoch 177/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2633 - ac
c: 0.9062
Epoch 178/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2787 - ac
c: 0.8988
Epoch 179/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2801 - ac
c: 0.8975
Epoch 180/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2875 - ac
c: 0.8950
Epoch 181/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2753 - ac
c: 0.9025
Epoch 182/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2787 - ac
c: 0.8900
Epoch 183/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2749 - ac
c: 0.8888
Epoch 184/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2812 - ac
c: 0.9050
Epoch 185/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2763 - ac
c: 0.8963
Epoch 186/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2662 - ac
c: 0.8950
Epoch 187/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2671 - ac
c: 0.8975
Epoch 188/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2766 - ac
c: 0.9013
Epoch 189/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2776 - ac
c: 0.8988
Epoch 190/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2666 - ac
c: 0.9013
Epoch 191/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2783 - ac
c: 0.8900
Epoch 192/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2984 - ac
c: 0.8775
Epoch 193/350
800/800 [==============================] - 0s 50us/sample - loss: 0.2753 - ac
c: 0.8988
Epoch 194/350
800/800 [==============================] - 0s 57us/sample - loss: 0.2829 - ac
c: 0.8913
Epoch 195/350
800/800 [==============================] - 0s 53us/sample - loss: 0.2684 - ac
c: 0.9013
Epoch 196/350
800/800 [==============================] - 0s 58us/sample - loss: 0.2722 - ac
c: 0.9000
```

```
Epoch 197/350
800/800 [==============================] - 0s 33us/sample - loss: 0.2606 - ac
c: 0.9038
Epoch 198/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2585 - ac
c: 0.8963
Epoch 199/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2666 - ac
c: 0.9000
Epoch 200/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2539 - ac
c: 0.9050
Epoch 201/350
800/800 [==============================] - 0s 77us/sample - loss: 0.2729 - ac
c: 0.8913
Epoch 202/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2721 - ac
c: 0.9013
Epoch 203/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2654 - ac
c: 0.9038
Epoch 204/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2641 - ac
c: 0.9025
Epoch 205/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2655 - ac
c: 0.9000
Epoch 206/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2516 - ac
c: 0.9062
Epoch 207/350
800/800 [==============================] - 0s 33us/sample - loss: 0.2754 - ac
c: 0.9000
Epoch 208/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2713 - ac
c: 0.9050
Epoch 209/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2680 - ac
c: 0.9038
Epoch 210/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2827 - ac
c: 0.8875
Epoch 211/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2806 - ac
c: 0.9038
Epoch 212/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2787 - ac
c: 0.8950
Epoch 213/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2874 - ac
c: 0.9000
Epoch 214/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2640 - ac
c: 0.9013
Epoch 215/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2825 - ac
c: 0.8963
Epoch 216/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2551 - ac
c: 0.9100
Epoch 217/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2762 - ac
c: 0.8938
Epoch 218/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2761 - ac
c: 0.8950
Epoch 219/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2636 - ac
c: 0.9100
```

```
Epoch 220/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2667 - ac
c: 0.9025
Epoch 221/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2829 - ac
c: 0.8963
Epoch 222/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2444 - ac
c: 0.9112
Epoch 223/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2622 - ac
c: 0.8938
Epoch 224/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2656 - ac
c: 0.9038
Epoch 225/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2805 - ac
c: 0.8963
Epoch 226/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2692 - ac
c: 0.9013
Epoch 227/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2825 - ac
c: 0.8988
Epoch 228/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2635 - ac
c: 0.9050
Epoch 229/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2783 - ac
c: 0.8913
Epoch 230/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2592 - ac
c: 0.9025
Epoch 231/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2601 - ac
c: 0.8975
Epoch 232/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2563 - ac
c: 0.9038
Epoch 233/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2621 - ac
c: 0.9038
Epoch 234/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2651 - ac
c: 0.9025
Epoch 235/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2550 - ac
c: 0.9000
Epoch 236/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2675 - ac
c: 0.8975
Epoch 237/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2771 - ac
c: 0.8900
Epoch 238/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2554 - ac
c: 0.9100
Epoch 239/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2721 - ac
c: 0.9087
Epoch 240/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2648 - ac
c: 0.9025
Epoch 241/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2572 - ac
c: 0.9087
Epoch 242/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2748 - ac
c: 0.8925
```

```
Epoch 243/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2505 - ac
c: 0.9000
Epoch 244/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2514 - ac
c: 0.9087
Epoch 245/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2961 - ac
c: 0.8925
Epoch 246/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2700 - ac
c: 0.8975
Epoch 247/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2656 - ac
c: 0.9075
Epoch 248/350
800/800 [==============================] - 0s 42us/sample - loss: 0.2502 - ac
c: 0.9087
Epoch 249/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2408 - ac
c: 0.9137
Epoch 250/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2668 - ac
c: 0.9038
Epoch 251/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2513 - ac
c: 0.9050
Epoch 252/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2679 - ac
c: 0.9050
Epoch 253/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2581 - ac
c: 0.9013
Epoch 254/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2675 - ac
c: 0.8988
Epoch 255/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2779 - ac
c: 0.9050
Epoch 256/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2701 - ac
c: 0.8913
Epoch 257/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2598 - ac
c: 0.9013
Epoch 258/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2724 - ac
c: 0.8950
Epoch 259/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2768 - ac
c: 0.8913
Epoch 260/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2674 - ac
c: 0.8963
Epoch 261/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2809 - ac
c: 0.8975
Epoch 262/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2570 - ac
c: 0.9125
Epoch 263/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2495 - ac
c: 0.9000
Epoch 264/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2770 - ac
c: 0.8850
Epoch 265/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2734 - ac
c: 0.9000
```

```
Epoch 266/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2550 - ac
c: 0.8988
Epoch 267/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2648 - ac
c: 0.8963
Epoch 268/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2730 - ac
c: 0.8963
Epoch 269/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2689 - ac
c: 0.8975
Epoch 270/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2676 - ac
c: 0.8925
Epoch 271/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2693 - ac
c: 0.8938
Epoch 272/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2659 - ac
c: 0.9075
Epoch 273/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2828 - ac
c: 0.8913
Epoch 274/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2662 - ac
c: 0.9013
Epoch 275/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2473 - ac
c: 0.9125
Epoch 276/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2594 - ac
c: 0.9100
Epoch 277/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2593 - ac
c: 0.9013
Epoch 278/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2696 - ac
c: 0.8925
Epoch 279/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2762 - ac
c: 0.9038
Epoch 280/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2646 - ac
c: 0.8950
Epoch 281/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2485 - ac
c: 0.9087
Epoch 282/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2561 - ac
c: 0.9038
Epoch 283/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2417 - ac
c: 0.9187
Epoch 284/350
800/800 [==============================] - 0s 34us/sample - loss: 0.2495 - ac
c: 0.9000
Epoch 285/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2700 - ac
c: 0.8875
Epoch 286/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2563 - ac
c: 0.9150
Epoch 287/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2649 - ac
c: 0.9125
Epoch 288/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2590 - ac
c: 0.9025
```

```
Epoch 289/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2794 - ac
c: 0.9000
Epoch 290/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2593 - ac
c: 0.9013
Epoch 291/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2743 - ac
c: 0.8938
Epoch 292/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2742 - ac
c: 0.9000
Epoch 293/350
800/800 [==============================] - 0s 68us/sample - loss: 0.2552 - ac
c: 0.9100
Epoch 294/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2539 - ac
c: 0.9025
Epoch 295/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2562 - ac
c: 0.9062
Epoch 296/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2525 - ac
c: 0.9075
Epoch 297/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2601 - ac
c: 0.9038
Epoch 298/350
800/800 [==============================] - 0s 42us/sample - loss: 0.2661 - ac
c: 0.9000
Epoch 299/350
800/800 [==============================] - 0s 36us/sample - loss: 0.2529 - ac
c: 0.8963
Epoch 300/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2472 - ac
c: 0.9075
Epoch 301/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2638 - ac
c: 0.9000
Epoch 302/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2739 - ac
c: 0.8988
Epoch 303/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2469 - ac
c: 0.9112
Epoch 304/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2646 - ac
c: 0.9038
Epoch 305/350
800/800 [==============================] - 0s 42us/sample - loss: 0.2484 - ac
c: 0.9087
Epoch 306/350
800/800 [==============================] - 0s 42us/sample - loss: 0.2491 - ac
c: 0.9112
Epoch 307/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2747 - ac
c: 0.8925
Epoch 308/350
800/800 [==============================] - 0s 57us/sample - loss: 0.2670 - ac
c: 0.8975
Epoch 309/350
800/800 [==============================] - 0s 64us/sample - loss: 0.2409 - ac
c: 0.9100
Epoch 310/350
800/800 [==============================] - 0s 55us/sample - loss: 0.2576 - ac
c: 0.8988
Epoch 311/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2638 - ac
c: 0.9025
```

```
Epoch 312/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2551 - ac
c: 0.9025
Epoch 313/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2440 - ac
c: 0.9075
Epoch 314/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2664 - ac
c: 0.9000
Epoch 315/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2570 - ac
c: 0.9087
Epoch 316/350
800/800 [==============================] - 0s 37us/sample - loss: 0.2694 - ac
c: 0.9013
Epoch 317/350
800/800 [==============================] - 0s 35us/sample - loss: 0.2587 - ac
c: 0.8988
Epoch 318/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2389 - ac
c: 0.9112
Epoch 319/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2758 - ac
c: 0.8988
Epoch 320/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2541 - ac
c: 0.9062
Epoch 321/350
800/800 [==============================] - 0s 42us/sample - loss: 0.2457 - ac
c: 0.9112
Epoch 322/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2610 - ac
c: 0.8963
Epoch 323/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2613 - ac
c: 0.9150
Epoch 324/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2492 - ac
c: 0.9112
Epoch 325/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2407 - ac
c: 0.9013
Epoch 326/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2557 - ac
c: 0.9125
Epoch 327/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2492 - ac
c: 0.9087
Epoch 328/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2845 - ac
c: 0.8875
Epoch 329/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2549 - ac
c: 0.8963
Epoch 330/350
800/800 [==============================] - 0s 43us/sample - loss: 0.2706 - ac
c: 0.8963
Epoch 331/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2541 - ac
c: 0.9000
Epoch 332/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2884 - ac
c: 0.8888
Epoch 333/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2548 - ac
c: 0.9125
Epoch 334/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2658 - ac
c: 0.9038
```

```
Epoch 335/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2495 - ac
c: 0.9112
Epoch 336/350
800/800 [==============================] - 0s 44us/sample - loss: 0.2644 - ac
c: 0.8988
Epoch 337/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2772 - ac
c: 0.8888
Epoch 338/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2547 - ac
c: 0.9038
Epoch 339/350
800/800 [==============================] - 0s 42us/sample - loss: 0.2660 - ac
c: 0.8975
Epoch 340/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2641 - ac
c: 0.8963
Epoch 341/350
800/800 [==============================] - 0s 42us/sample - loss: 0.2685 - ac
c: 0.9025
Epoch 342/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2574 - ac
c: 0.9050
Epoch 343/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2596 - ac
c: 0.9025
Epoch 344/350
800/800 [==============================] - 0s 44us/sample - loss: 0.2566 - ac
c: 0.9025
Epoch 345/350
800/800 [==============================] - 0s 39us/sample - loss: 0.2651 - ac
c: 0.8963
Epoch 346/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2563 - ac
c: 0.9013
Epoch 347/350
800/800 [==============================] - 0s 40us/sample - loss: 0.2487 - ac
c: 0.9075
Epoch 348/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2666 - ac
c: 0.9050
Epoch 349/350
800/800 [==============================] - 0s 38us/sample - loss: 0.2580 - ac
c: 0.9075
Epoch 350/350
800/800 [==============================] - 0s 41us/sample - loss: 0.2815 - ac
c: 0.8938
Best hyper-parameters for the Dense Neural Network: {'Estimator__DropoutL1':
0.2, 'Estimator__batch_size': 25, 'Estimator__epochs': 350}
Best cross-validation average accuracy for the Dense Neural Network: 0.920
```

Question 5: Evaluate the 3 models. (15 marks)

#

The code below evaluates each of the three models. Each code block makes a prediction on the X_test data based off the best parameters found and creates a confusion matrix to determine the true postive, true negative, false postive and false negative rates. The code also produces a classification report to output various parameters, such as accuracy, F1 score and more.

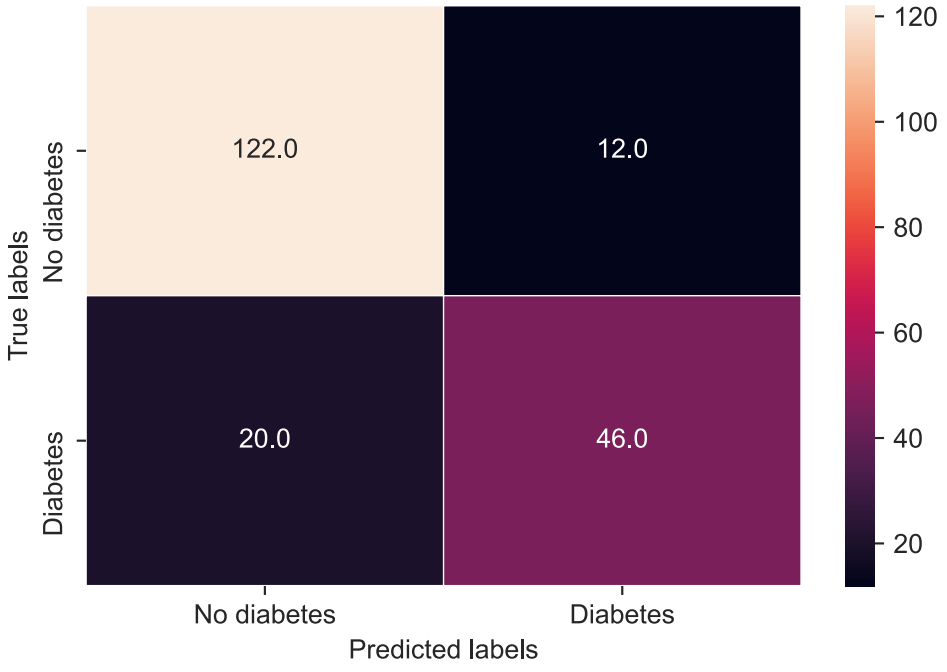
#

In [141…

```
# Evaluation for Logistic Regression
```

```
# Predict based off the grid search model parameters on test set
y_pred_logistic = grid_search_logistic.predict(X_test)
# PRredict based off the grid search on training set
y_pred_logistic_training = grid_search_logistic.predict(X_train)

# Create a heatmap
# add labels
label = {'No diabetes', 'Diabetes'}
#create matrix
matrix = confusion_matrix(y_test, y_pred_logistic)
ax = plt.subplot()
# using seaborn create the heatmap and add labels
sns.heatmap(matrix, annot=True,linewidths=0.5, fmt='0.1f', ax=ax, xticklabels=
# add extra labels for pred and true
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels')

# Determine the accuracy of the model and print classification report
print('Accuracy score: {:.2f}'.format(accuracy_score(y_test, y_pred_logistic)
print()
print('Classification report: \n{}'.format(classification_report(y_test, y_pr
print()

# Calculate accuracy of prediction & see if model is overfitting/underfitting
print("Accuracy on training set: {:.2f}".format(accuracy_score(y_train, y_pre
print("Accuracy on test set: {:.2f}".format(accuracy_score(y_test, y_pred_log
```

```
Accuracy score: 0.84

Classification report:
              precision    recall  f1-score   support

           0       0.86      0.91      0.88       134
           1       0.79      0.70      0.74        66

    accuracy                           0.84       200
   macro avg       0.83      0.80      0.81       200
weighted avg       0.84      0.84      0.84       200


Accuracy on training set: 0.84
Accuracy on test set: 0.84
```

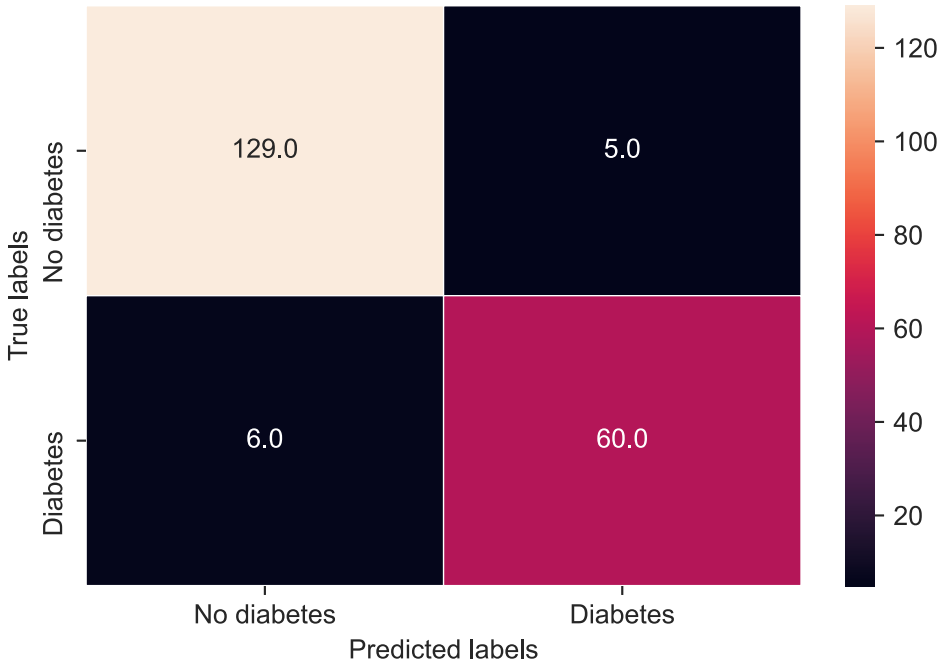


In [142…

```
# Evaluation for Gradient Boosted Tree
# Predict based off the grid search model parameters on test set
```

```
y_pred_gbt = grid_search_gbt.predict(X_test)
# PRredict based off the grid search on training set
y_pred_gbt_training = grid_search_gbt.predict(X_train)

# Create a heatmap
# add labels
label = {'No diabetes', 'Diabetes'}
#create matrix
matrix = confusion_matrix(y_test, y_pred_gbt)
ax = plt.subplot()
# using seaborn create the heatmap and add labels
sns.heatmap(matrix, annot=True,linewidths=0.5, fmt='0.1f', ax=ax, xticklabels=
# add extra labels for pred and true
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels')

# Determine the accuracy of the model and print classification report
print('Accuracy score: {:.2f}'.format(accuracy_score(y_test, y_pred_gbt)))
print()
print('Classification report: \n{}'.format(classification_report(y_test, y_pr
print()

# Calculate accuracy of prediction & see if model is overfitting/underfitting
print("Accuracy on training set: {:.2f}".format(accuracy_score(y_train, y_pre
print("Accuracy on test set: {:.2f}".format(accuracy_score(y_test, y_pred_gbt
```

```
Accuracy score: 0.94

Classification report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       134
           1       0.92      0.91      0.92        66

    accuracy                           0.94       200
   macro avg       0.94      0.94      0.94       200
weighted avg       0.94      0.94      0.94       200


Accuracy on training set: 0.98
Accuracy on test set: 0.94
```

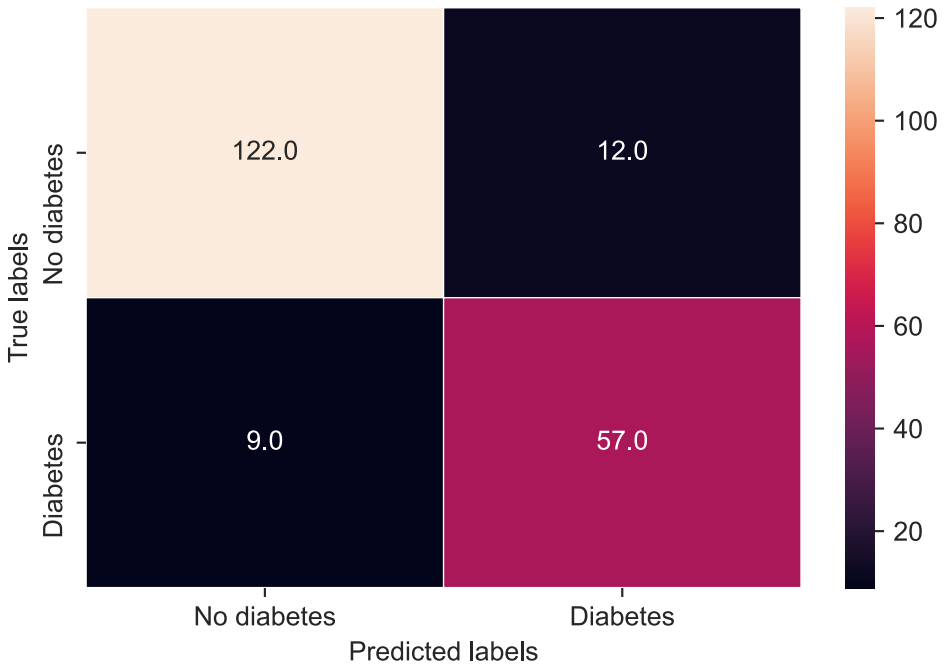


In [143…

```
# Evaluation for Dense Neural Network

# Predict based off the grid search model parameters on test set
```

```
y_pred_dnn = grid_search_dnn.predict(X_test)
# PRredict based off the grid search on training set
y_pred_dnn_training = grid_search_dnn.predict(X_train)

# Create a heatmap
# add labels
label = {'No diabetes', 'Diabetes'}
#create matrix
matrix = confusion_matrix(y_test, y_pred_dnn)
ax = plt.subplot()
# using seaborn create the heatmap and add labels
sns.heatmap(matrix, annot=True,linewidths=0.5, fmt='0.1f', ax=ax, xticklabels=
# add extra labels for pred and true
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels')

# Determine the accuracy of the model and print classification report
print('Accuracy score: {:.2f}'.format(accuracy_score(y_test, y_pred_dnn)))
print()
print('Classification report: \n{}'.format(classification_report(y_test, y_pr
print()

# Calculate accuracy of prediction & see if model is overfitting/underfitting
print("Accuracy on training set: {:.2f}".format(accuracy_score(y_train, y_pre
print("Accuracy on test set: {:.2f}".format(accuracy_score(y_test, y_pred_dnn
```

```
Accuracy score: 0.90

Classification report:
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       134
           1       0.83      0.86      0.84        66

    accuracy                           0.90       200
   macro avg       0.88      0.89      0.88       200
weighted avg       0.90      0.90      0.90       200


Accuracy on training set: 0.93
Accuracy on test set: 0.90
```



Question 6: "Ensemble averaging is the

process of creating multiple models and combining them to produce a desired output". [https://en.wikipedia.org/wiki/Ensemble_averagi

In this question, calculate the average of the probabilities predicted by each of the previous trained models.

Use this average probability to predict the final class.

(15 marks)

#

To create an ensemble average, the code below calls predict_proba which outputs the probabilities of each row of data (patient) belonging to both the negative (Outcome =0) and postive (Outcome=1) classes. The code takes the average of each of the models probas, then takes the probabilities of the postive class and makes a prediction of zero or 1.

#

In [144…

```
# Model Averaging
# Predict proba (probabilitity) for each model - will print predictions for e
# use predict proba on each model
log_reg_proba = grid_search_logistic.predict_proba(X_test)
gbt_proba = grid_search_gbt.predict_proba(X_test)
dnn_proba = grid_search_dnn.predict_proba(X_test)
# calculate the average of each model
ensemble = (log_reg_proba + gbt_proba + dnn_proba)/3
# take the second column of these probabilities (the positive column - ie tha
ensemble = ensemble[:,1]
# convert probabilities to a 1 (diabetes present) or 0 (no DM) based upon whe
ensemble = np.where(ensemble> 0.5,1,0)
print(ensemble)
```

```
[0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 0
 0 1 1 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0
 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 1 0 0 0 0 1 0 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 1
 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0]
```

Question 7: Evaluate the new ensemble "model" created in question 6. (5 marks)

#

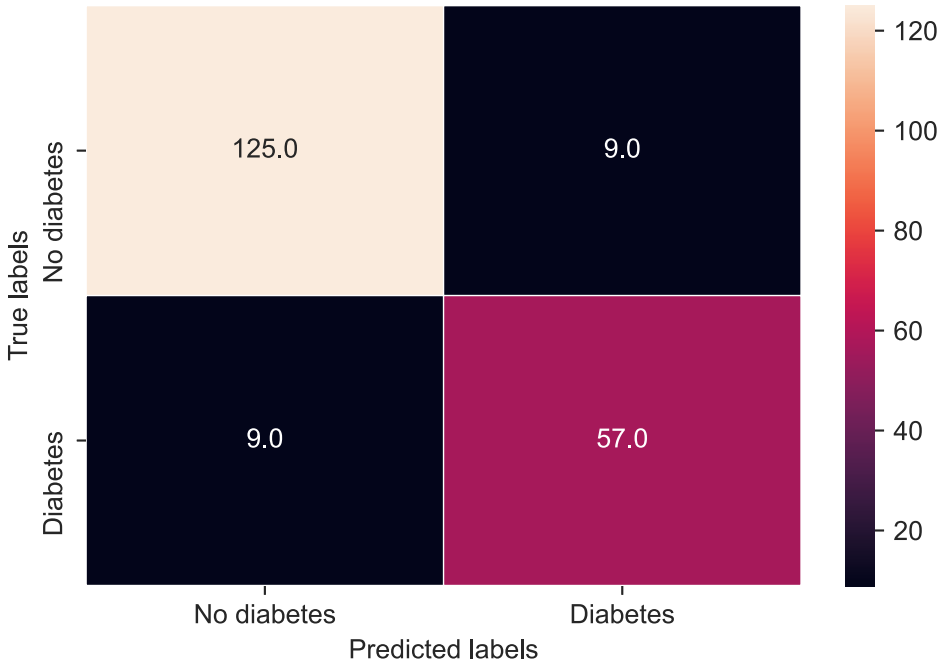
The code chunk below takes the probabilies from the ensemble model and creates a confusion matrix of the true/false and positve/negative rates. We also print the classification report to determine the F1 score, and accuracy.

#

In [145…

```
# Evaluation for Ensemble Model
# Create a heatmap
# add labels
label = {'No diabetes', 'Diabetes'}
#create matrix
matrix = confusion_matrix(y_test, ensemble)
ax = plt.subplot()
# using seaborn create the heatmap and add labels
sns.heatmap(matrix, annot=True,linewidths=0.5, fmt='0.1f', ax=ax, xticklabels=
# add extra labels for pred and true
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels')

# Determine the accuracy of the model and print classification report
print('Accuracy score: {:.2f}'.format(accuracy_score(y_test, ensemble)))
print()
print('Classification report: \n{}'.format(classification_report(y_test, ense
```

```
Accuracy score: 0.91

Classification report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       134
           1       0.86      0.86      0.86        66

    accuracy                           0.91       200
   macro avg       0.90      0.90      0.90       200
weighted avg       0.91      0.91      0.91       200
```



Question 8:

Print all the evaluation metrics together in order to compare the 4 models:

a. Choose the best model or none of them according to the evaluation metrics that you selected and this specific context/application of the predictive model.

b. Provide a discussion and conclusion statement about your work and the justification for your choice of model.

(15 marks)

#

The goal of the three models above was to address the research question: "Machine Learning algorithms could learn to forecast whether a given individual would develop DM within five years based on the value of the eight input variables. Binary variable "Outcome": 1/Yes, 0/No". The choice we have here between the three models depends highly upon the context in which the model will be used. To paraphrase from the outline, doctors have found a treatment for DM if the treatment is used prior to the onset of DM. Hence they want to know who will develop DM within 5 years, with the caveat that the treatment is quite expensive, and will not be able to treat everyone. This leads to the conclusion that the model with the highest true postive rate (TPR) would be the best fitting model as it will capture the most number of patients at risk of developing DM. However the model with the highest TPR may also have a higher false positive rate (FPR) resulting in prescribing treatments that are not required. From a cost standpoint this may not be the most effective, however it ensures that the highest number of at risk patients are recieving the treatment that they require.

Using the logic outlined above alongside the following the fact that our Gradient Boosted Tree model, performed the best with unseen data (an accuracy of 94%) compared to the logistic regression (accuracy of 84%), the neural network (accuracy of 92%) or the ensemble model (accuracy of 93%). Thus we can conclude that the Gradient Boosted Tree moel is expected to predict onset of diabetes with a higher degree of accuracy than the other models presented. Furthermore the gradient boosted tree model achieved the highest weighted F1 score of the models, of 0.94 compared to 0.93 for the ensemble model, 0.91 for the dense neural network and 0.84 for the logistic regression model.. The F1 score in this scenario is a more robust method of measuring and comparing the models as it its the mean between precision and recall of the model, thus with an imbalanced dataset (in our case a 7:3 ratio of imbalance) means the model could score 70% by guessing diabetes isn't present in all cases, hence the use of teh F1 score. Finally the gradient boosted tree model obtained the lowest false positive rate (FPR) amongst the models presented, with a FPR of 2.5%. In context, this means our gradient boosted model will only falsely predict a patient will have onset diabetes within 5 years, 2.5% of the time; thus leading to lower costs for the doctors and treatment, ultimately leading to more funds available to treat patients who are at risk of diabetes.

In conclusion, we chose the Gradient Boosted Tree model for this scenario due to the above mentioned points; ideally a balanced dataset would enhance the model, however this is not practical in reality due to the population at hand. Limitations of the model stems from tree based models in general in that they cannot predict outcomes where patients have values outside the range of the training data and that the running time of the model will increase dramatically with much larger datasets. Overall our gradient boosted tree model appears to be the best fitting model for the Indian diabetes dataset.

#

#

#