QBS Competition 1 - Report

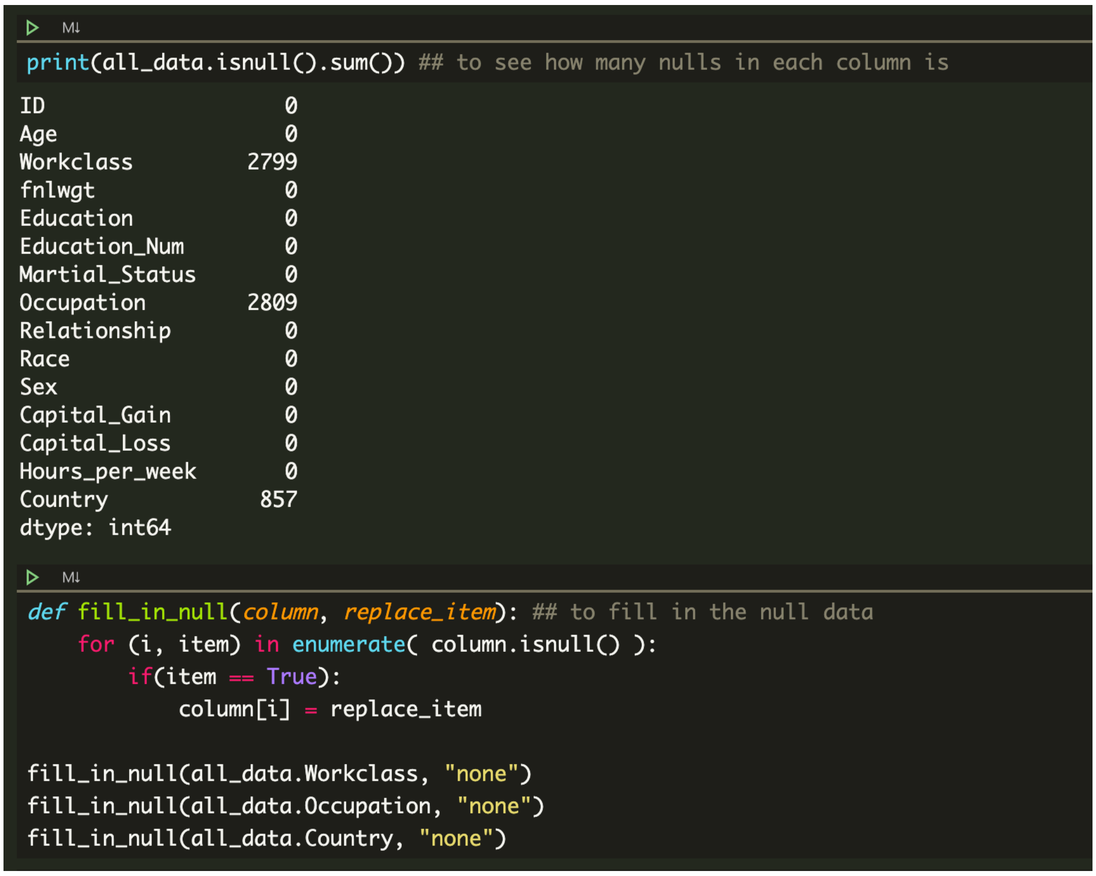
(A) what I do for data pre-processing (remove NA or feature engineering……)

(B) explorative data analysis (EDA)

For (A) and (B), I first process the null data, then process the numerical data and the categorical data. As for explorative data analysis, I only draw a few pictures. Instead, I check whether there are null by function, or see the correlation of columns and Target by table, and I also check the distribution of 0 and 1 of the training set, which would be useful for the following model training.

I. process the null data:

I check the overall data, and see how many and what columns have NaN data. Then, I write a function to fill in all the null data



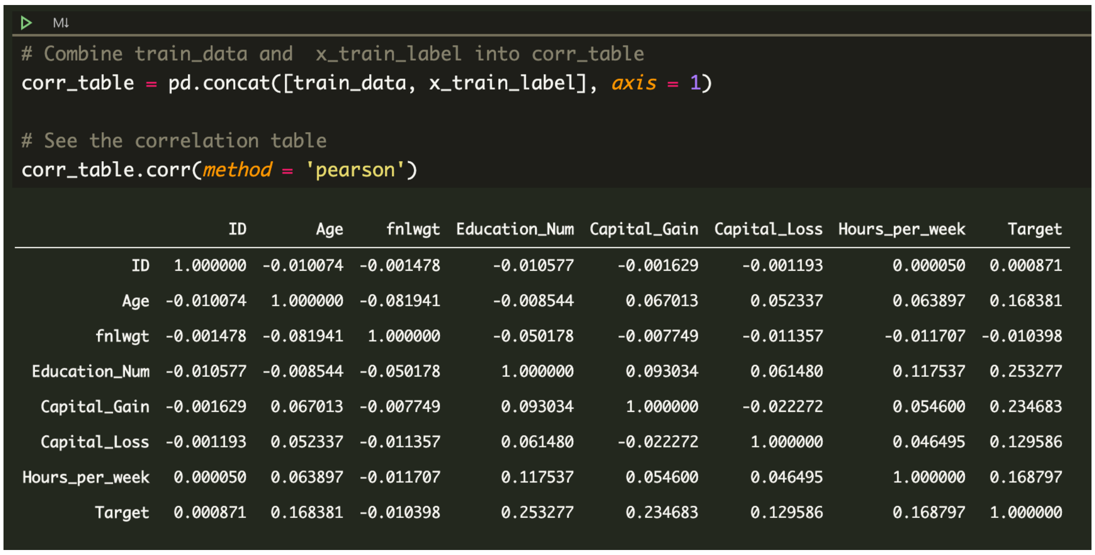
```
print(all_data.isnull().sum()) ## to see how many nulls in each column is
```

```
ID                   0
Age                  0
Workclass         2799
fnlwgt               0
Education            0
Education_Num        0
Martial_Status       0
Occupation        2809
Relationship         0
Race                 0
Sex                  0
Capital_Gain         0
Capital_Loss         0
Hours_per_week       0
Country            857
dtype: int64
```

```
def fill_in_null(column, replace_item): ## to fill in the null data
    for (i, item) in enumerate( column.isnull() ):
        if(item == True):
            column[i] = replace_item

fill_in_null(all_data.Workclass, "none")
fill_in_null(all_data.Occupation, "none")
fill_in_null(all_data.Country, "none")
```

II. process the numerical data

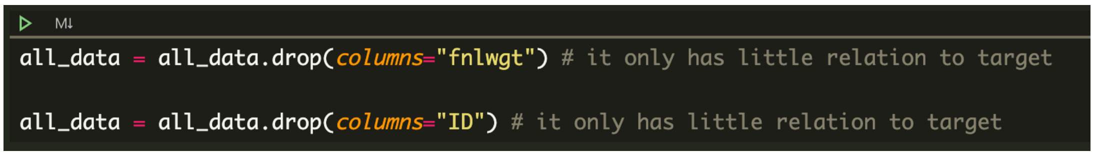
I first make a table to see the correlation between other columns and Target.



```python
# Combine train_data and  x_train_label into corr_table
corr_table = pd.concat([train_data, x_train_label], axis = 1)

# See the correlation table
corr_table.corr(method = 'pearson')
```

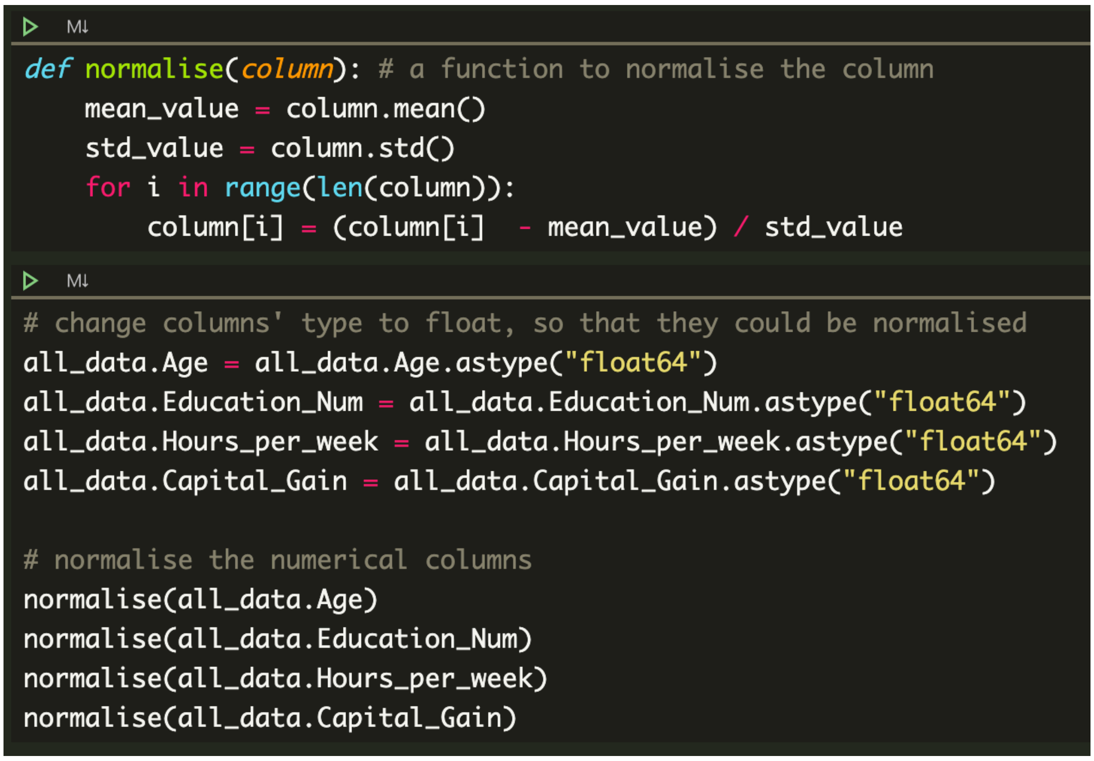
```
                     ID       Age    fnlwgt  Education_Num  Capital_Gain  Capital_Loss  Hours_per_week    Target
ID             1.000000 -0.010074 -0.001478      -0.010577     -0.001629     -0.001193        0.000050  0.000871
Age           -0.010074  1.000000 -0.081941      -0.008544      0.067013      0.052337        0.063897  0.168381
fnlwgt        -0.001478 -0.081941  1.000000      -0.050178     -0.007749     -0.011357       -0.011707 -0.010398
Education_Num -0.010577 -0.008544 -0.050178       1.000000      0.093034      0.061480        0.117537  0.253277
Capital_Gain  -0.001629  0.067013 -0.007749       0.093034      1.000000     -0.022272        0.054600  0.234683
Capital_Loss  -0.001193  0.052337 -0.011357       0.061480     -0.022272      1.000000        0.046495  0.129586
Hours_per_week 0.000050  0.063897 -0.011707       0.117537      0.054600      0.046495        1.000000  0.168797
Target         0.000871  0.168381 -0.010398       0.253277      0.234683      0.129586        0.168797  1.000000
```

Then, I remove the “fnlwgt”, and “ID” column, because they have little correlation with Target.



```python
all_data = all_data.drop(columns="fnlwgt") # it only has little relation to target

all_data = all_data.drop(columns="ID") # it only has little relation to target
```

After that, I normalize the numerical data, so that they would be easier to fit the model.

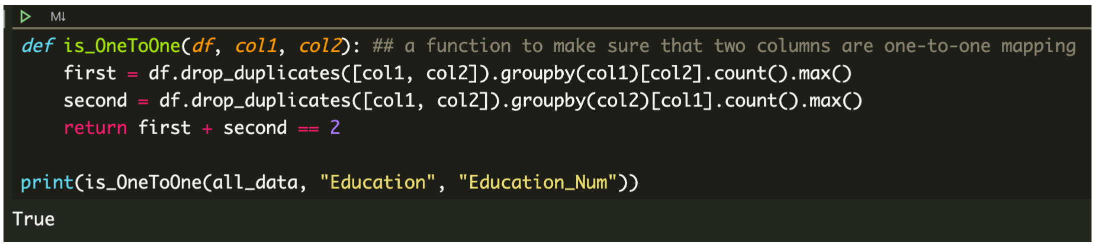


```python
def normalise(column): # a function to normalise the column
    mean_value = column.mean()
    std_value = column.std()
    for i in range(len(column)):
        column[i] = (column[i]  - mean_value) / std_value
```

```python
# change columns' type to float, so that they could be normalised
all_data.Age = all_data.Age.astype("float64")
all_data.Education_Num = all_data.Education_Num.astype("float64")
all_data.Hours_per_week = all_data.Hours_per_week.astype("float64")
all_data.Capital_Gain = all_data.Capital_Gain.astype("float64")

# normalise the numerical columns
normalise(all_data.Age)
normalise(all_data.Education_Num)
normalise(all_data.Hours_per_week)
normalise(all_data.Capital_Gain)
```

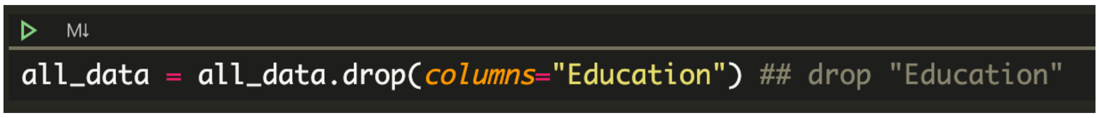
III. process the categorical data

I suspect that "Education" and "Education_Num" are one-to-one mapping, so I check it by a function. Then, I find that they are indeed one-to-one mapping. So, I remove the "Education" column.



```
def is_OneToOne(df, col1, col2): ## a function to make sure that two columns are one-to-one mapping
    first = df.drop_duplicates([col1, col2]).groupby(col1)[col2].count().max()
    second = df.drop_duplicates([col1, col2]).groupby(col2)[col1].count().max()
    return first + second == 2

print(is_OneToOne(all_data, "Education", "Education_Num"))
```

```
True
```

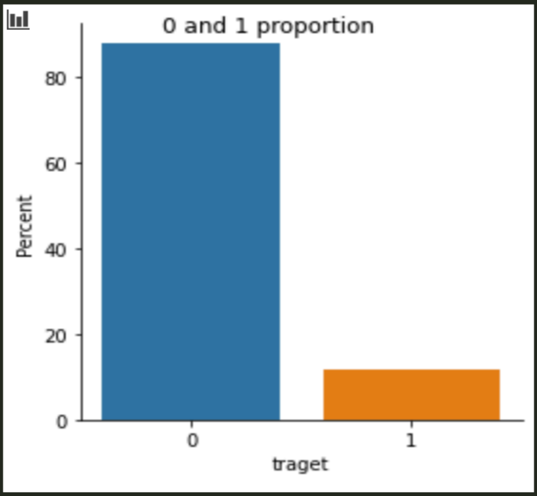


```
all_data = all_data.drop(columns="Education") ## drop "Education"
```

IV. check the distribution of 0 and 1 of training data set

I find that there is imbalanced in training data label, so I will make some adjustment when I build the model to avoid the model tends to predict target as 0.

```
0: 88.12 %
1: 11.88 %
```



(C) model development process:

I. DL model draft, parameter initialization, parameter tuning

I only follow the basic model which is show in the textbook

II. DL revised model draft, parameter initialization, parameter tuning

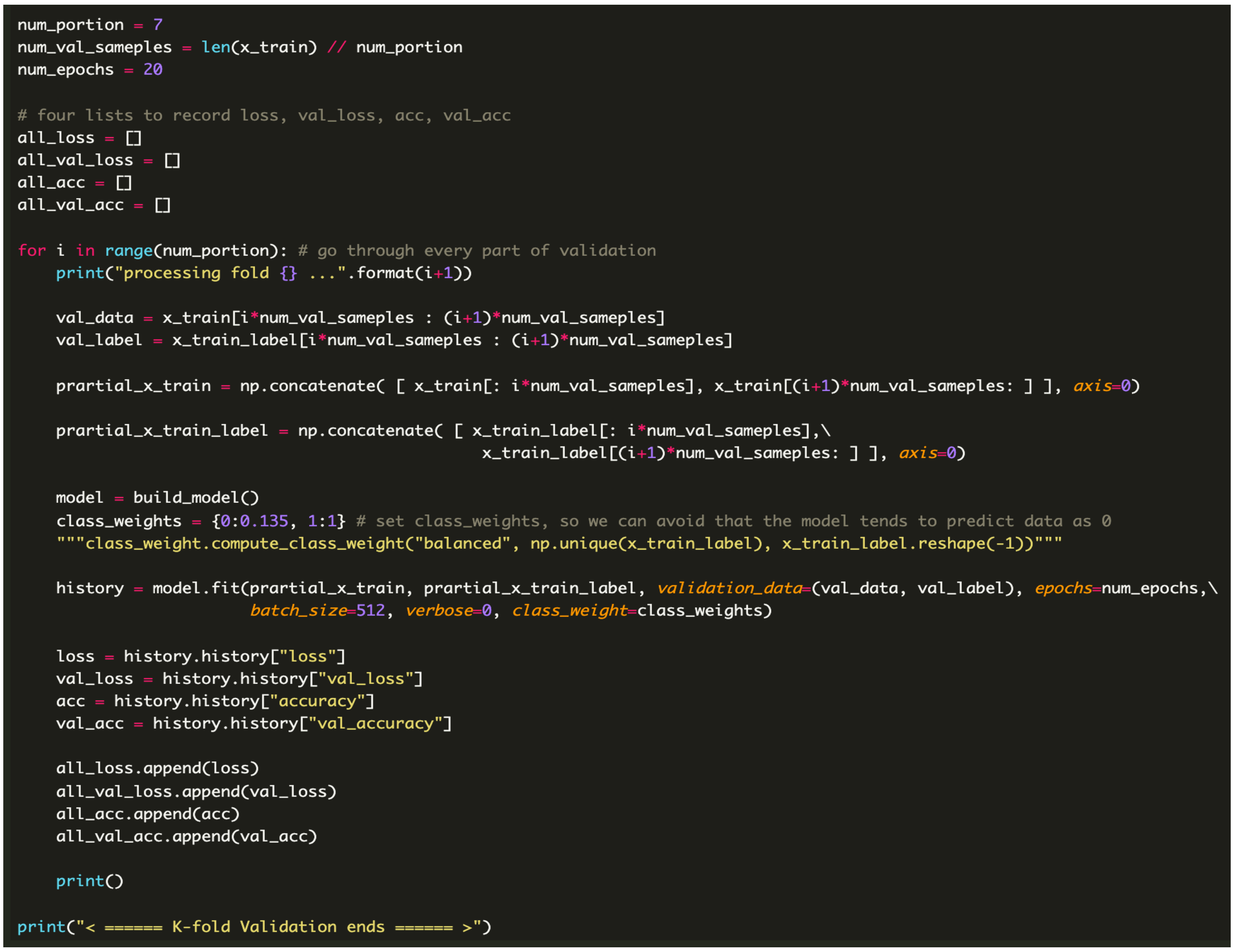
I find that there may be imbalanced in the label of training data, so I try to resample the data by oversampling and under sampling, yet it doesn't perform well.
And I use K-fold to find the best number of epochs.

III. DL finalized model, parameter initialization, parameter tuning

I find that add the hyper parameters would help, such as "dropout" and "kernel_regularizer".
Also, I find that setting the class weight would perform better that resampling, because resampling may cause overfitting in this situation, yet setting the class weight could avoid overfitting in this situation.
Hence, I end up using K-fold, hyper parameters, and setting up the class weight to build and fit my finalized model.

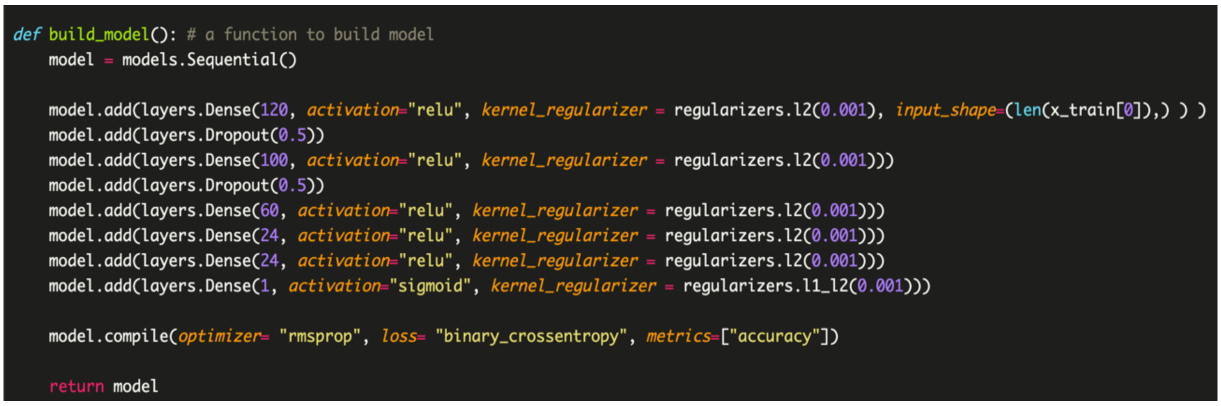
My finalized model:

1. Part of K-fold



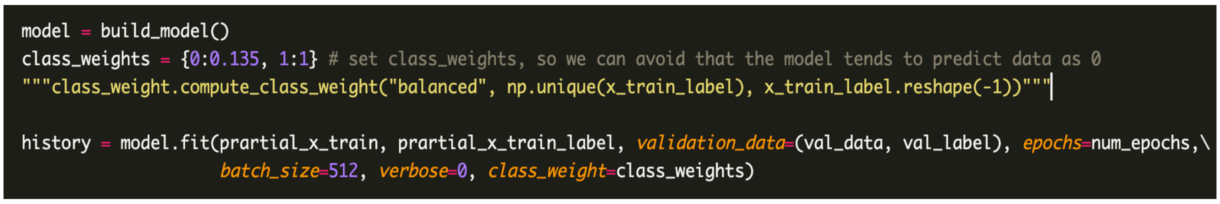
```python
num_portion = 7
num_val_sameples = len(x_train) // num_portion
num_epochs = 20

# four lists to record loss, val_loss, acc, val_acc
all_loss = []
all_val_loss = []
all_acc = []
all_val_acc = []

for i in range(num_portion): # go through every part of validation
    print("processing fold {} ...".format(i+1))

    val_data = x_train[i*num_val_sameples : (i+1)*num_val_sameples]
    val_label = x_train_label[i*num_val_sameples : (i+1)*num_val_sameples]

    prartial_x_train = np.concatenate( [ x_train[: i*num_val_sameples], x_train[(i+1)*num_val_sameples: ] ], axis=0)

    prartial_x_train_label = np.concatenate( [ x_train_label[: i*num_val_sameples],\
                                            x_train_label[(i+1)*num_val_sameples: ] ], axis=0)

    model = build_model()
    class_weights = {0:0.135, 1:1} # set class_weights, so we can avoid that the model tends to predict data as 0
    """class_weight.compute_class_weight("balanced", np.unique(x_train_label), x_train_label.reshape(-1))"""

    history = model.fit(prartial_x_train, prartial_x_train_label, validation_data=(val_data, val_label), epochs=num_epochs,\
                        batch_size=512, verbose=0, class_weight=class_weights)

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    acc = history.history["accuracy"]
    val_acc = history.history["val_accuracy"]

    all_loss.append(loss)
    all_val_loss.append(val_loss)
    all_acc.append(acc)
    all_val_acc.append(val_acc)

    print()

print("< ====== K-fold Validation ends ====== >")
```

2. Part of hyperparameters



```python
def build_model(): # a function to build model
    model = models.Sequential()

    model.add(layers.Dense(120, activation="relu", kernel_regularizer = regularizers.l2(0.001), input_shape=(len(x_train[0]),) ) )
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(100, activation="relu", kernel_regularizer = regularizers.l2(0.001)))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(60, activation="relu", kernel_regularizer = regularizers.l2(0.001)))
    model.add(layers.Dense(24, activation="relu", kernel_regularizer = regularizers.l2(0.001)))
    model.add(layers.Dense(24, activation="relu", kernel_regularizer = regularizers.l2(0.001)))
    model.add(layers.Dense(1, activation="sigmoid", kernel_regularizer = regularizers.l1_l2(0.001)))

    model.compile(optimizer= "rmsprop", loss= "binary_crossentropy", metrics=["accuracy"])

    return model
```

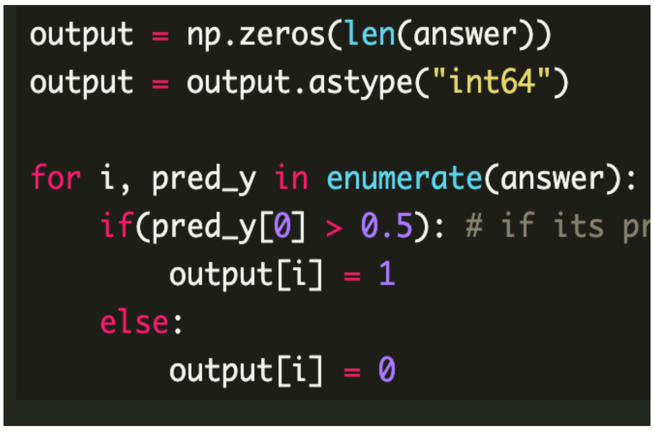
3. Part of setting the class weight



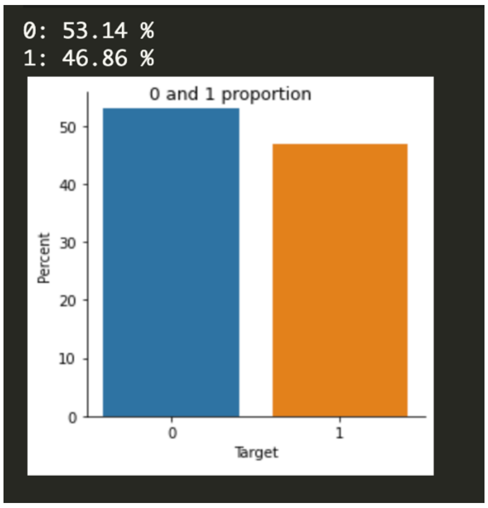
```python
model = build_model()
class_weights = {0:0.135, 1:1} # set class_weights, so we can avoid that the model tends to predict data as 0
"""class_weight.compute_class_weight("balanced", np.unique(x_train_label), x_train_label.reshape(-1))"""

history = model.fit(prartial_x_train, prartial_x_train_label, validation_data=(val_data, val_label), epochs=num_epochs,\
                    batch_size=512, verbose=0, class_weight=class_weights)
```

(d) The prediction result

The prediction result would be probability. I transform those whose probability < 0.5 into 0, and others into 1. The prediction result ["0" : "1"] would be about [53 : 46].



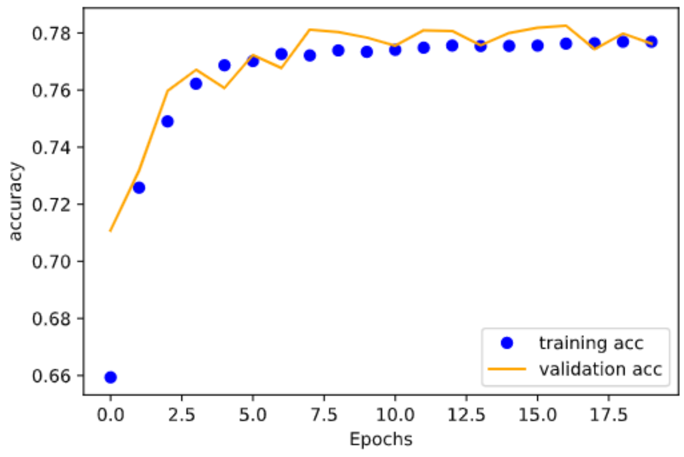
```
output = np.zeros(len(answer))
output = output.astype("int64")

for i, pred_y in enumerate(answer):
    if(pred_y[0] > 0.5): # if its pr
        output[i] = 1
    else:
        output[i] = 0
```



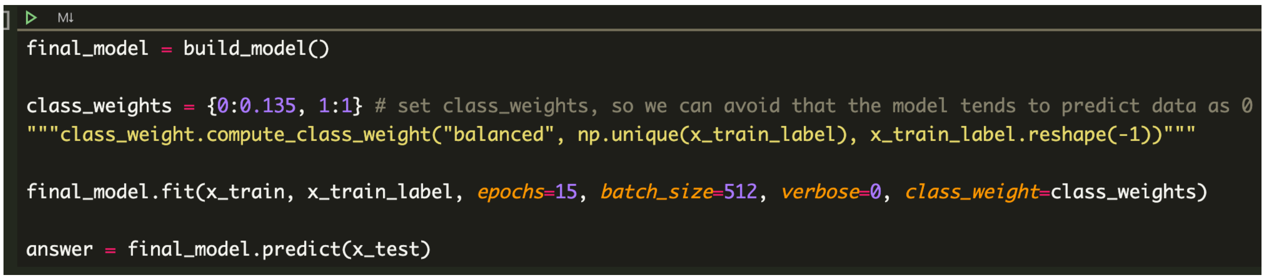
```
0: 53.14 %
1: 46.86 %
```

(e) explaining the model prediction to make people trust and understand whether your predictive machine is reasonable, understandable, and trustable.

This is the accuracy of K-fold Validation, we could see that it is convergent, and doesn't happen overfitting. Thus, I can choose a best number of epochs to rebuild the model and fit it with test data. And also, we know that k-fold would run many times, so we would be more confident about the validation accuracy curve.



Also, rebuild the model with the parameters and hyper-parameters which are confirmed working well, could make sure that the model we fit test data to is relative great.



```
final_model = build_model()

class_weights = {0:0.135, 1:1} # set class_weights, so we can avoid that the model tends to predict data as 0
"""class_weight.compute_class_weight("balanced", np.unique(x_train_label), x_train_label.reshape(-1))"""

final_model.fit(x_train, x_train_label, epochs=15, batch_size=512, verbose=0, class_weight=class_weights)

answer = final_model.predict(x_test)
```

Thus, beside the abovementioned technique, we do many things to remove the null data and noisy data, and we also remove the column which has only little correlation with the Target, and many others data processing.

To sum up, we process the data appropriately and use a sequence of technique to get better model, so we can be confident about our prediction.

(f) Your modeling and data analysis based on the lectures, tutorials, and assigned readings.

The model I build:

The basic and K-fold part is based on the textbook (FC) and tutorials.

The hyper-parameters part is based on the textbook (FC)

The class weight setting is based on the information given by TA and google and a classmate.

The data analysis I do is based on the self-study:

Data visualization and Feature Engineering.

(e) learning progress, reflection, and feedback for the teaching team's reference

In my learning process, I consider that finding someone to discuss with is important. Because I have never learned things about machine learning before, I may have some blind spot, in that case, even if I google very hard, sometimes I can't get the point to improve the performance of my model. For example, the idea of setting the class weight is from the other student in this class, and it really helps me a lot.

For feedback, I hope that TA and Teacher could taught us how to deal with some particular situation, because when it's my time to put developing the prediction in practice, I find much more problem that I've never seen in the course or textbook, even self-reading. Maybe it's because of lacking of experience, but, if possible, I still want to know more technique in class.