In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import sklearn
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [2]:
```
train = pd.read_csv('data/red_wine_train.csv',index_col=0)

#dropping columns based on heatmap correlations in EDA
train = train.drop(['fixed acidity', 'chlorides', 'density', 'total sulfur dioxide'
train.head()
```

Out[2]:

	volatile acidity	citric acid	sulphates	alcohol	quality
1348	0.655	0.03	0.39	9.5	5
117	0.560	0.12	0.50	9.4	6
1150	0.330	0.32	0.76	12.8	7
235	0.630	0.00	0.58	9.0	6
91	0.490	0.28	1.95	9.9	6

In [3]:
```
test = pd.read_csv('data/red_wine_test.csv', index_col=0)
test = test.drop(['fixed acidity', 'chlorides', 'density', 'total sulfur dioxide',
```

In [4]:
```
###begin raw
```

In [5]:
```
x_train = train.drop('quality',axis=1) #training df without class column

x_test = test.drop('quality',axis=1) #testing df without class column

y_train = train['quality'] #training df only class column

y_test = test['quality'] #testing df only class column

y_train = y_train.map({3:0, 4:1, 5:2, 6:3, 7:4, 8:5})
y_test = y_test.map({3:0, 4:1, 5:2, 6:3, 7:4, 8:5})
```

In [6]:
```
n_inputs = [x_train.shape[1]] #n cols => n inputs in model
n_units = 12
n_batch = 100
n_epochs = 10
```

In [7]:
```
tf.keras.backend.clear_session() #resets parameters, necessary before each new mode
keras.utils.set_random_seed(0) #setting random seed for the entire program

modelRaw = tf.keras.Sequential([tf.keras.layers.Dense(units=n_units, activation='re
                                tf.keras.layers.Dense(units=6, activation='softmax'
```

```
modelRaw.summary()

modelRaw.compile(optimizer='adam',loss='mae', metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                60

 dense_1 (Dense)             (None, 6)                 78

=================================================================
Total params: 138
Trainable params: 138
Non-trainable params: 0
_________________________________________________________________
```

In [8]:
```
modelRaw.fit(x_train, y_train, batch_size=n_batch, epochs=n_epochs)
```

```
Epoch 1/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 2/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 3/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 4/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 5/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 6/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 7/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 8/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 9/10
13/13 [==============================] - 0s 1ms/step - loss: 2.4726 - accuracy: 0.39
87
Epoch 10/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.39
87
```

Out[8]: <keras.callbacks.History at 0x26ca4736f40>

In [9]:
```
###end raw

###begin scaled
```

In [10]:
```
#recombining train & test to get overall max and min values so test and train are s
whole_set = pd.concat([train,test])
whole_set.describe() #summary to show all columns have varying scales
```

Out[10]:

	volatile acidity	citric acid	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	0.527821	0.270976	0.658149	10.422983	5.636023
std	0.179060	0.194801	0.169507	1.065668	0.807569
min	0.120000	0.000000	0.330000	8.400000	3.000000
25%	0.390000	0.090000	0.550000	9.500000	5.000000
50%	0.520000	0.260000	0.620000	10.200000	6.000000
75%	0.640000	0.420000	0.730000	11.100000	6.000000
max	1.580000	1.000000	2.000000	14.900000	8.000000

In [11]:
```
#drop quality class label column before scaling
whole_set.drop('quality',axis=1, inplace=True)

#build scaler
scaler = MinMaxScaler() #build scaler
scaler.fit(whole_set) #fit scaler to entire df w/o quality col
```

Out[11]:
```
MinMaxScaler()
```

In [12]:
```
x_trainScaled=scaler.transform(x_train)
x_testScaled=scaler.transform(x_test)

#make transformed data in a dataframe (.transform returns arrays, we want df) using
x_trainScaled = pd.DataFrame(x_trainScaled, columns=x_train.columns)
x_testScaled = pd.DataFrame(x_testScaled, columns=x_test.columns)
```

In [13]:
```
x_trainScaled.describe() #now each attr column has min 0 and max 1
```

Out[13]:

	volatile acidity	citric acid	sulphates	alcohol
count	1279.000000	1279.000000	1279.000000	1279.000000
mean	0.281668	0.266059	0.195751	0.310976
std	0.120700	0.193606	0.099773	0.166797
min	0.000000	0.000000	0.000000	0.000000
25%	0.191781	0.090000	0.131737	0.169231
50%	0.273973	0.250000	0.173653	0.276923
75%	0.356164	0.420000	0.239521	0.415385
max	1.000000	1.000000	1.000000	1.000000

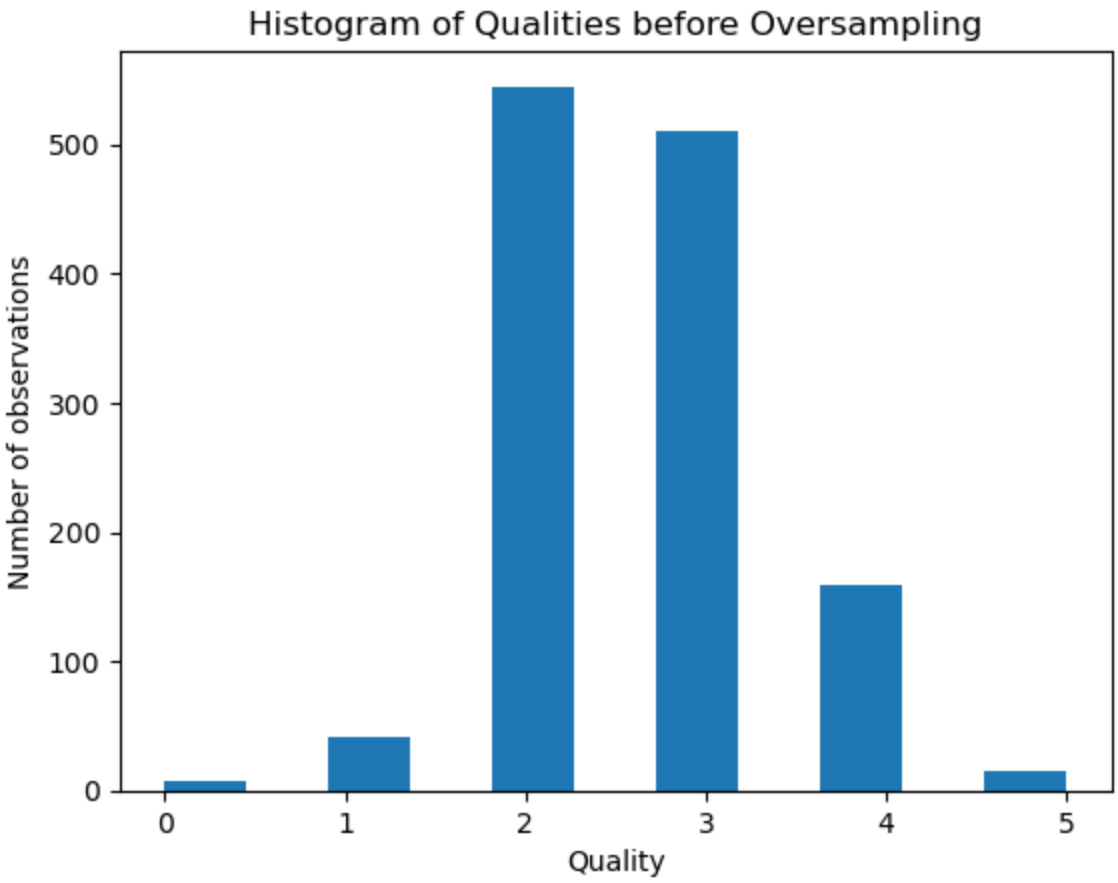
In [14]:
```
tf.keras.backend.clear_session() #resets parameters, necessary before each new mode

modelScaled = tf.keras.Sequential([tf.keras.layers.Dense(units=n_units, activation=
                                   tf.keras.layers.Dense(units=6, activation='soft

modelScaled.summary()

modelScaled.compile(optimizer='adam', loss='mae', metrics=['accuracy'])

modelScaled.fit(x_trainScaled, y_train, batch_size=n_batch, epochs=n_epochs)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                60

 dense_1 (Dense)             (None, 6)                 78

=================================================================
Total params: 138
Trainable params: 138
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 2/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 3/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 4/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 5/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 6/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 7/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 8/10
13/13 [==============================] - 0s 1ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 9/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
Epoch 10/10
13/13 [==============================] - 0s 2ms/step - loss: 2.4726 - accuracy: 0.10
16
```

Out[14]: <keras.callbacks.History at 0x26ca5aa44f0>

In [15]:
```
###end scaled

###begin oversampled
```

In [16]:
```
#plot to show that oversampling balanced the data
plt.hist(y_train, bins=11)
plt.title("Histogram of Qualities before Oversampling")
plt.ylabel('Number of observations')
plt.xlabel('Quality')
plt.show()
```



In [17]:
```
#dividing train into separate dfs for each class value
qual3 = (train[train["quality"]==3])
qual4 = (train[train["quality"]==4])
qual5 = (train[train["quality"]==5])
qual6 = (train[train["quality"]==6])
qual7 = (train[train["quality"]==7])
qual8 = (train[train["quality"]==8])

#number of samples per class label to determine inbalance
n_qual3 = len(qual3) #8
n_qual4 = len(qual4) #42
n_qual5 = len(qual5) #545
n_qual6 = len(qual6) #510
n_qual7 = len(qual7) #159
n_qual8 = len(qual8) #15

n_max = max(n_qual3, n_qual4, n_qual5, n_qual6, n_qual7, n_qual8) #545
```

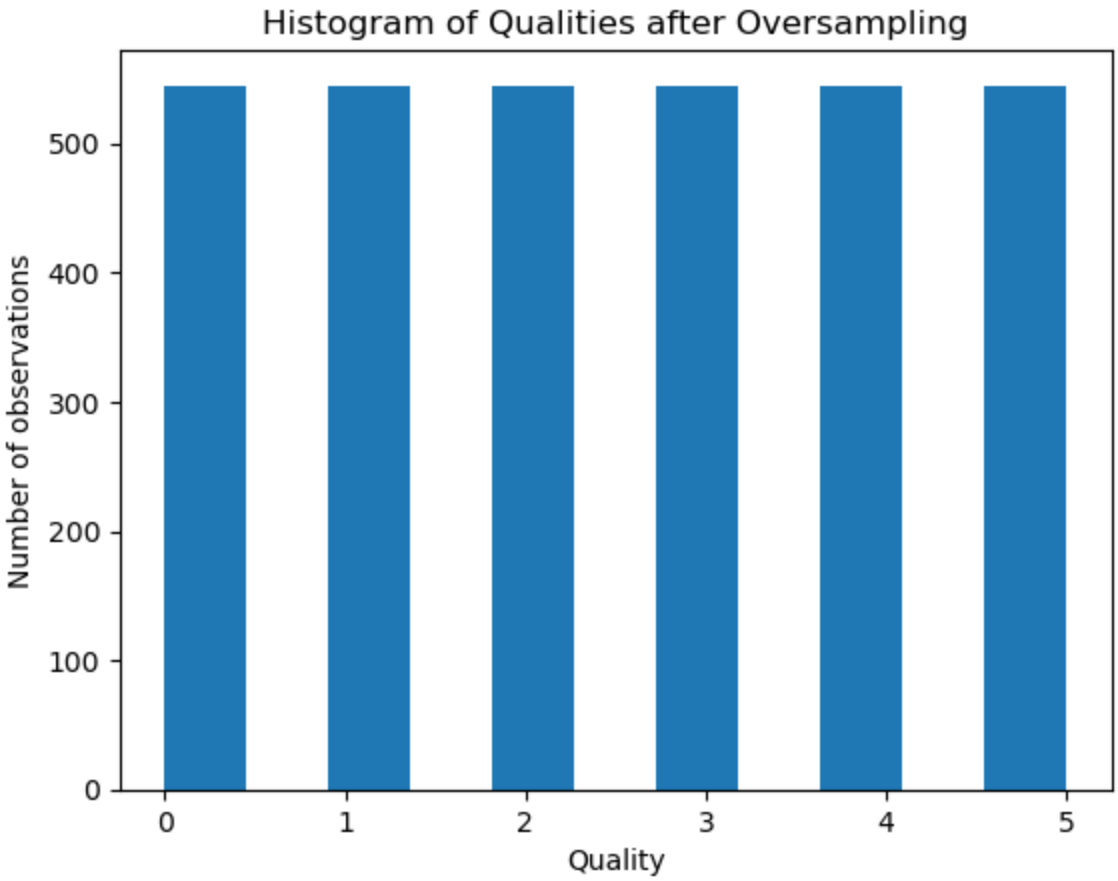
In [18]:
```
#oversample so that each class has the same number of observations,
#equal to the number of observations of quality 5
qual3Oversampled = qual3.sample(n_max, replace=True)
qual4Oversampled = qual4.sample(n_max, replace=True)
qual6Oversampled = qual6.sample(n_max, replace=True)
qual7Oversampled = qual7.sample(n_max, replace=True)
qual8Oversampled = qual8.sample(n_max, replace=True)
```

In [19]:
```
#concat back into one df. this is unscaled
```

```
trainOversampled = pd.concat([qual3Oversampled, qual4Oversampled, qual5, qual6Overs
```

In [20]:
```
#scaling oversampled
x_trainOversampled = trainOversampled.drop('quality',axis=1) #training df without c

y_trainOversampled = trainOversampled['quality'] #training df only class column

#map values of train from [3,8] to [0,5]
y_trainOversampled = y_trainOversampled.map({3:0, 4:1, 5:2, 6:3, 7:4, 8:5})
```

In [21]:
```
#plot to show that oversampling balanced the data
plt.hist(y_trainOversampled, bins=11)
plt.title("Histogram of Qualities after Oversampling")
plt.ylabel('Number of observations')
plt.xlabel('Quality')
plt.show()
```



In [22]:
```
#scale trainOversampled w scaler from before
#the same scaler will work bcus our oversampled data will have same range as raw da
x_trainOversampledScaled=scaler.transform(x_trainOversampled)

#make transformed data in a dataframe (.transform returns arrays, we want df) using
x_trainScaledOversampled = pd.DataFrame(x_trainOversampledScaled, columns=x_train.c
```

In [23]:
```
tf.keras.backend.clear_session() #resets parameters, necessary before each new mode

modelOversampledScaled = tf.keras.Sequential([tf.keras.layers.Dense(units=n_units,
                                              tf.keras.layers.Dense(units=6, acti
```

```
modelOversampledScaled.summary()

modelOversampledScaled.compile(optimizer='adam', loss='mae', metrics=['accuracy'])

modelOversampledScaled.fit(x_trainOversampledScaled, y_trainOversampled, batch_size
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                60

 dense_1 (Dense)             (None, 6)                 78

=================================================================
Total params: 138
Trainable params: 138
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
33/33 [==============================] - 1s 2ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 2/10
33/33 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 3/10
33/33 [==============================] - 0s 1ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 4/10
33/33 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 5/10
33/33 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 6/10
33/33 [==============================] - 0s 1ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 7/10
33/33 [==============================] - 0s 1ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 8/10
33/33 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 9/10
33/33 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.13
06
Epoch 10/10
33/33 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.13
06
```

Out[23]: <keras.callbacks.History at 0x26ca5d13640>

In [24]:
```
###end oversampled
```

```
###begin weighted
#returning to scaled data, then calculating weights and scaling
```

In [25]:
```
#make each column a numpy array for class weights computation
qual3_numpy = qual3['quality'].to_numpy()
qual4_numpy = qual4['quality'].to_numpy()
qual5_numpy = qual5['quality'].to_numpy()
qual6_numpy = qual6['quality'].to_numpy()
qual7_numpy = qual7['quality'].to_numpy()
qual8_numpy = qual8['quality'].to_numpy()

#combine numpy arrays into whole numpy and set variable for class values
whole_numpy = np.concatenate((qual3_numpy, qual4_numpy, qual5_numpy, qual6_numpy, q
unique_classes = np.unique(whole_numpy)

#compute weights with sklearn method
weights = sklearn.utils.class_weight.compute_class_weight(class_weight='balanced',
weightsDict = {i:w for i,w in enumerate(weights)}
```

In [26]:
```
tf.keras.backend.clear_session() #resets parameters, necessary before each new mode

modelWeighted = tf.keras.Sequential([tf.keras.layers.Dense(units=n_units, activatio
                                     tf.keras.layers.Dense(units=6, activation='sof

modelWeighted.summary()

modelWeighted.compile(optimizer='adam', loss='mae', metrics=['accuracy'])

modelWeighted.fit(x_trainScaled, y_train, batch_size=n_batch, epochs=n_epochs, clas
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                60

 dense_1 (Dense)             (None, 6)                 78

=================================================================
Total params: 138
Trainable params: 138
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
13/13 [==============================] - 1s 3ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 2/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 3/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 4/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 5/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 6/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 7/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 8/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 9/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
Epoch 10/10
13/13 [==============================] - 0s 2ms/step - loss: 2.3889 - accuracy: 0.42
69
```

Out[26]: <keras.callbacks.History at 0x26ca7f117f0>

In [27]:
```
#function to output accuracy, precision, recall, and F1 for each model
#input model name, model object, and appropriate test set x_test (scaled or unscale
#output header and metric scores as percents

def outputMetrics(modelname, model, x_test):
    y_pred = model.predict(x_test)

    y_pred_df = (pd.DataFrame(y_pred))

    y_pred_label = y_pred.argmax(axis=1)
```

```
    f1 = f1_score(y_test,y_pred_label,average="weighted",zero_division=1)
    recall = recall_score(y_test,y_pred_label,average="weighted",zero_division=1)
    precision = precision_score(y_test, y_pred_label,average="weighted",zero_divisi
    accuracy = accuracy_score(y_test,y_pred_label)
    print(modelname, "Metrics:")
    print("Accuracy: {:.2f}%".format(accuracy*100))
    print("Precision: {:.2f}%".format(precision*100))
    print("Recall: {:.2f}%".format(recall*100))
    print("F1: {:.2f}%".format(f1*100))

outputMetrics("Raw Model", modelRaw, x_test)
outputMetrics("Scaled Model", modelScaled, x_testScaled)
outputMetrics("Scaled & Oversampled Model", modelOversampledScaled, x_testScaled)
outputMetrics("Weighted Model", modelWeighted, x_testScaled)
```

```
10/10 [==============================] - 0s 2ms/step
Raw Model Metrics:
Accuracy: 40.00%
Precision: 76.00%
Recall: 40.00%
F1: 22.86%
10/10 [==============================] - 0s 1ms/step
Scaled Model Metrics:
Accuracy: 5.62%
Precision: 50.71%
Recall: 5.62%
F1: 5.63%
10/10 [==============================] - 0s 1ms/step
Scaled & Oversampled Model Metrics:
Accuracy: 3.12%
Precision: 56.69%
Recall: 3.12%
F1: 1.37%
10/10 [==============================] - 0s 2ms/step
Weighted Model Metrics:
Accuracy: 42.19%
Precision: 74.86%
Recall: 42.19%
F1: 25.22%
```