Machine Learning

CAB 420 Assignment 1A

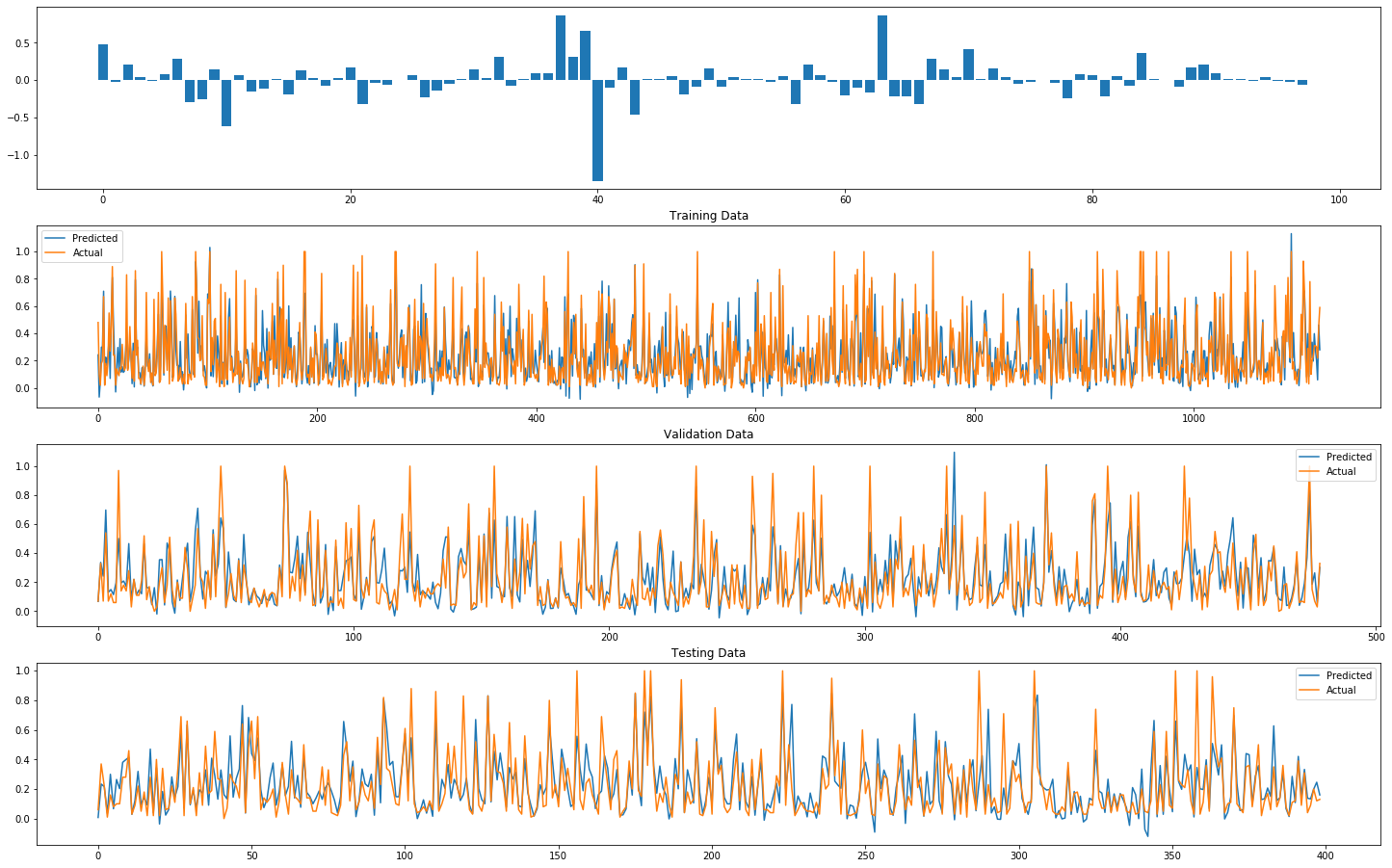
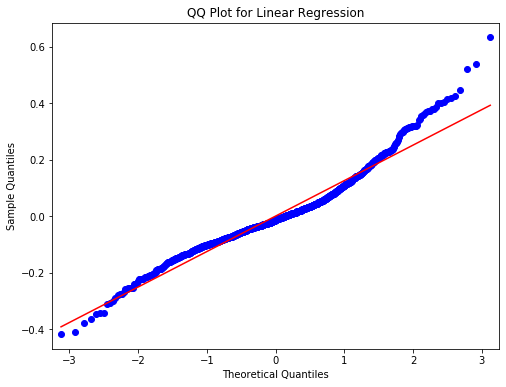
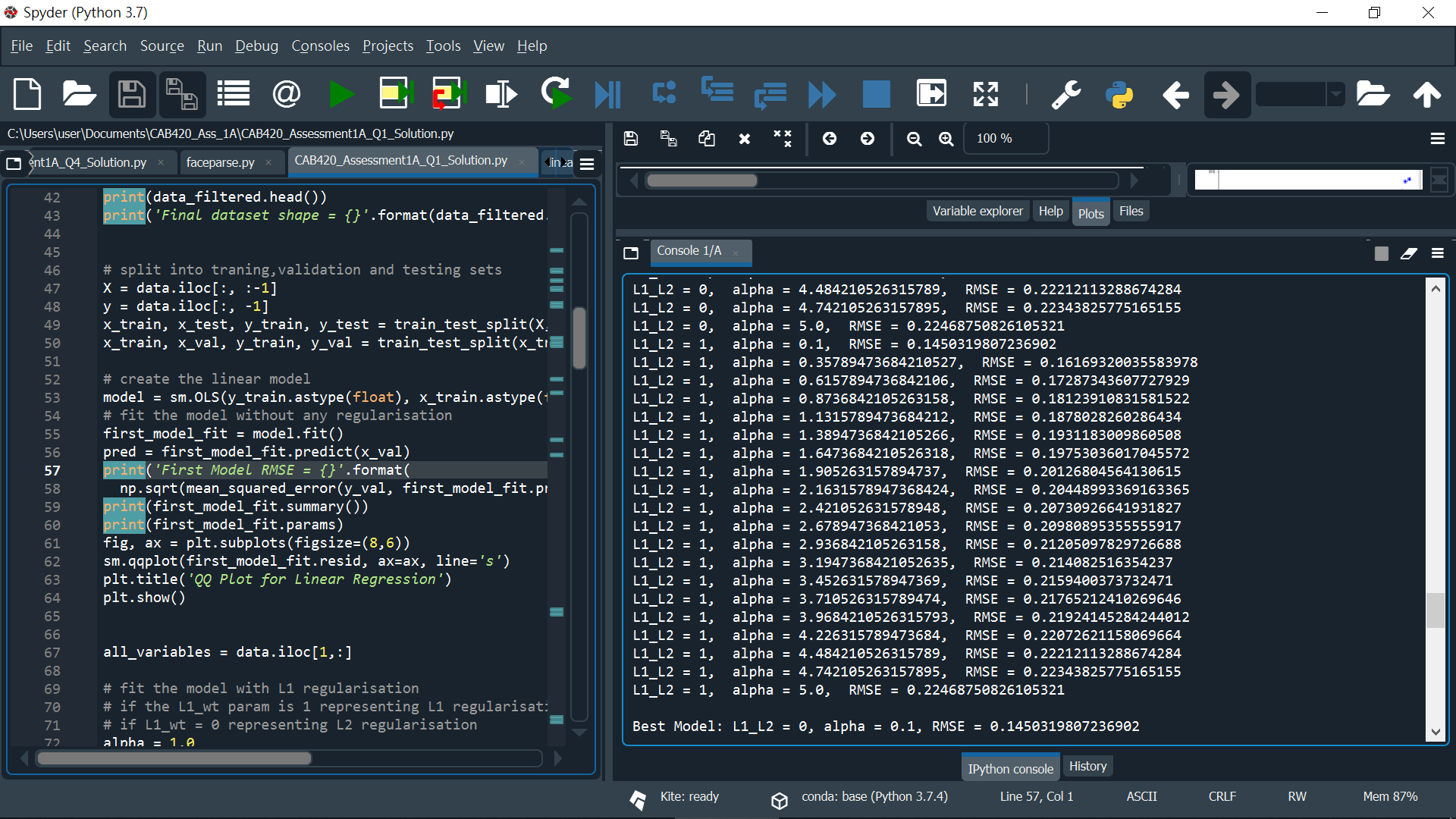
(Ho Fong Law n10107321, Kiki Mutiara n10031014)

Question 1)

First, import the CSV that we require. exclude the non-analysing columns such as country community etc. After removed selected columns. According to the data from CSV. There are some ‘?’ symbols that can’t analysis also. Therefore, we should discard any of this for improvement of the analyses. On the other hand, some information included ‘nan’ possibility. Thus, we rescan data set and remove them also for finally data shape

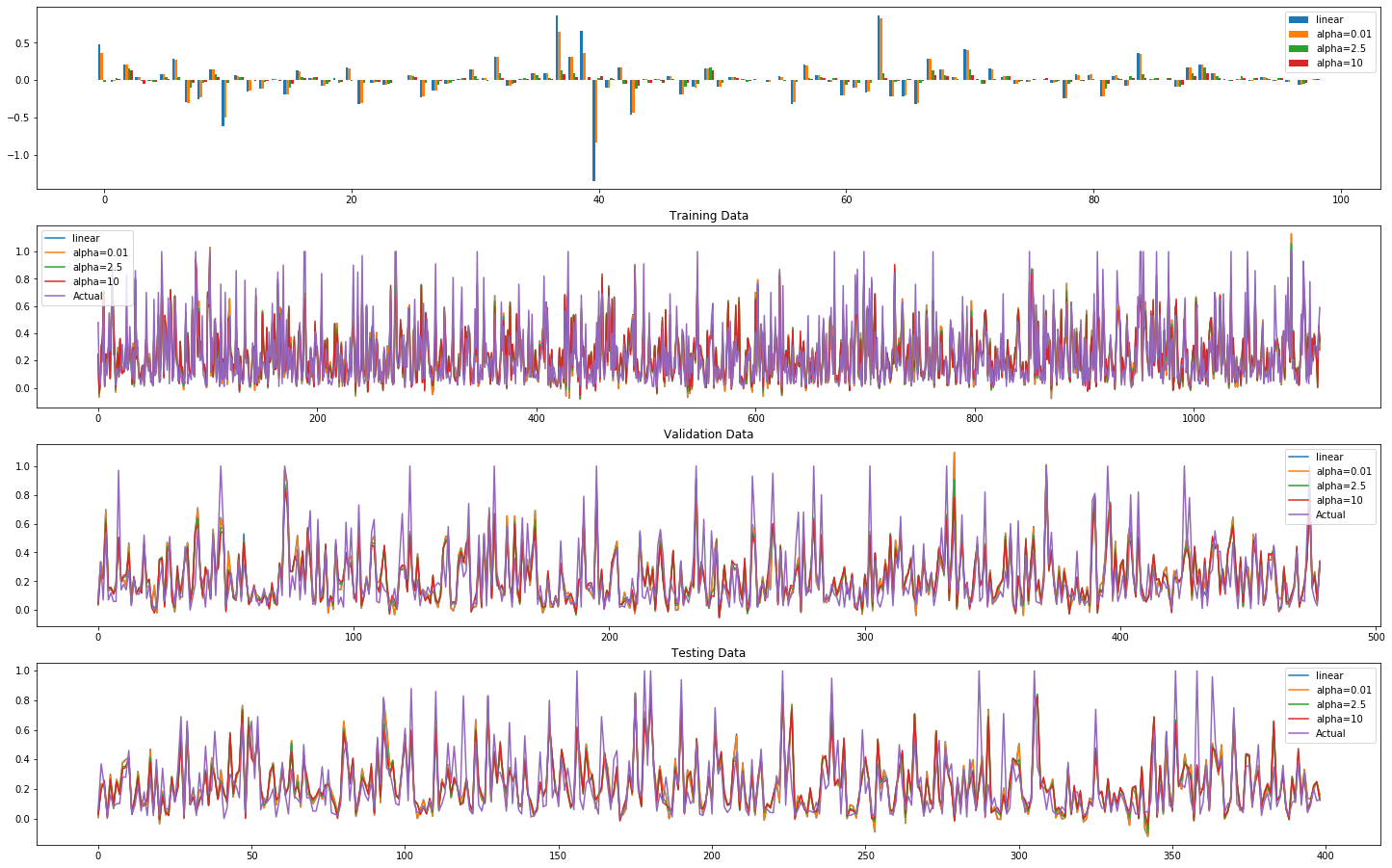
Base on the assignment requirement and steps we did before. We removed 23 items plus 4 non-analysis items that. however, we still left 1994 rows for further prediction.

After finalizing data sets, we separate the model into training, validation and testing set.80% for the training set and 20% will belong validation set. Inside the training set, we would further separate the 30% to the testing set for experimenting effectiveness of prediction

The first model we handle is linear, based on the QQ plot analysis, we know data is not ideal for diversity. We tested out the data separated with L1 and L2 models to find out the best alpha and lowest root mean square. After the process, we got the best result when we apply alpha to 0.1. And graph showing ideal prediction on criminal per capital with 0.145 RMSE. 

Lasso regression come after, we separately applying the 0.01, 0.1 and 0.5 alpha for model training. Turn out 0.5 not providing any foresight. The reason because the data set values all using tiny value therefore If affect prediction on a larger number. According to the graph we could observe, the alpha 0.01 given out the best result with only 0.15 root mean square of the testing set. 

At the final, Ridge regression, we used 2.5,5 and 10 for an alpha, as we recognise the ridge regression with alpha 2.5 return the best result with only 0.141 RMSE.



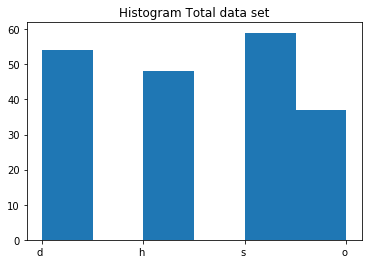
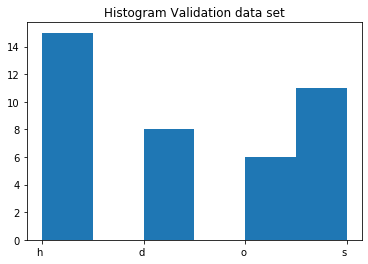
In conclusion with comparing three methods, we found ridge regression giving the most suitable prediction as it provided the lowest RMSE value.

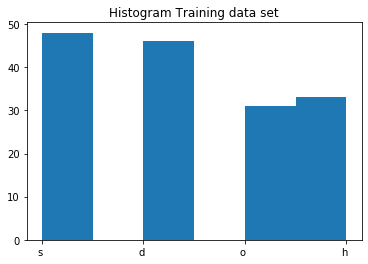


Question 2)

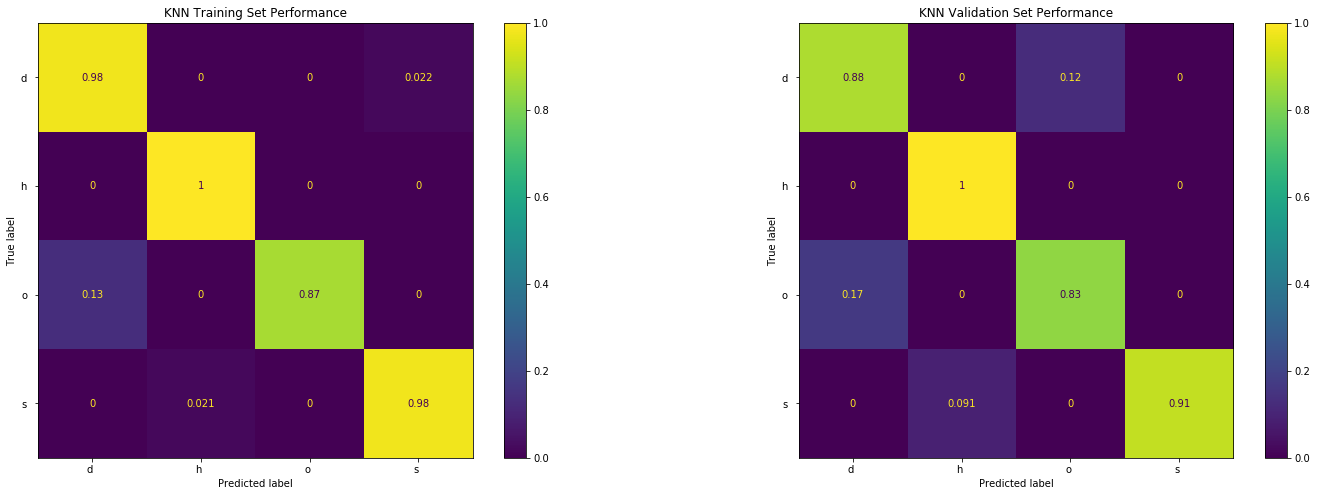
In the model that we see, we going to import forest data for getting data inside, first, we do the same action as the last question. We divided into training data, testing data and validation data. which we store testing in x\_test,y\_test and training in X, Y. We then separate 20 % into validation data.

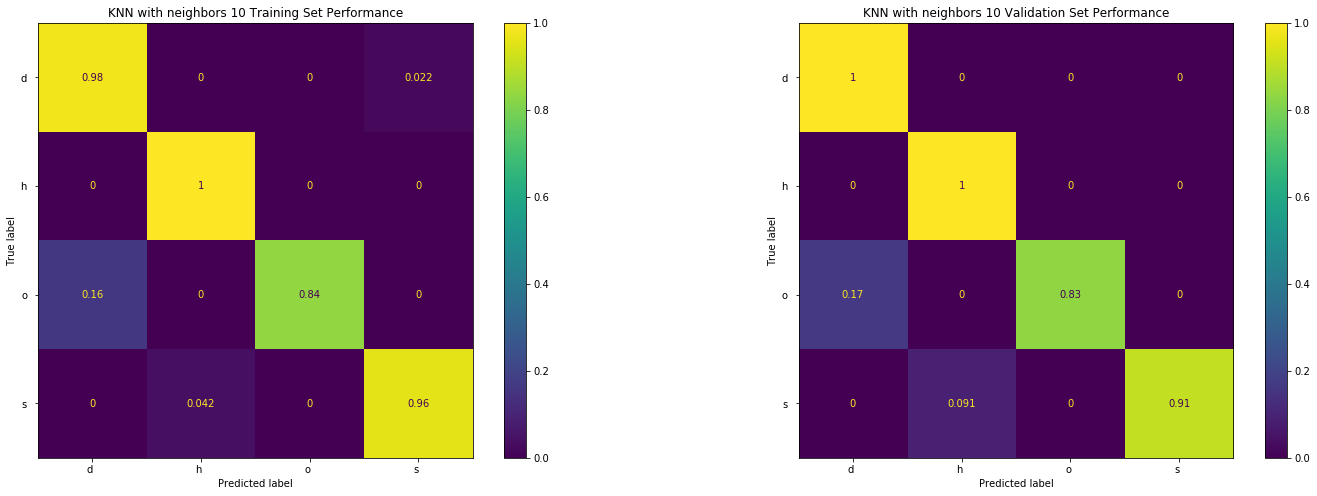
For the first two functions, we separate plot function into two which provided for validation model and test model. Those function return accuracy of prediction in the different prediction model



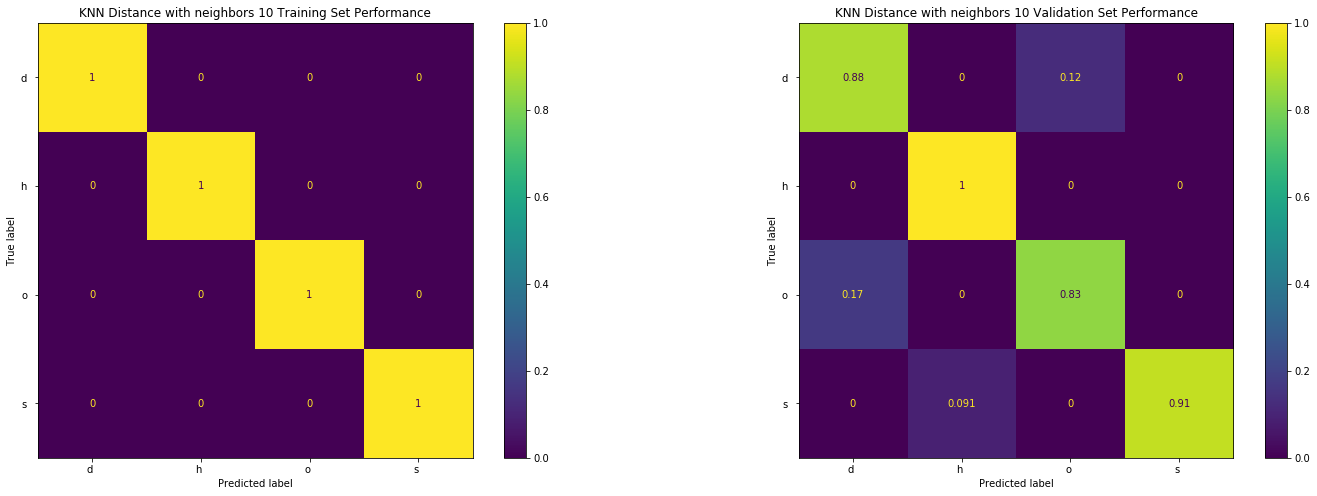


In KNNN model, we got to adjust model with a neighbour, which I give it to 5,10 and 15 all three predict a great accuracy in validation set but 10 neighbours are the best. Therefore, I put the 10 neighbours adding with distance for the optimizer, it turns out to have overfitted problems. Thus, will the keep the selection without distance optimizer

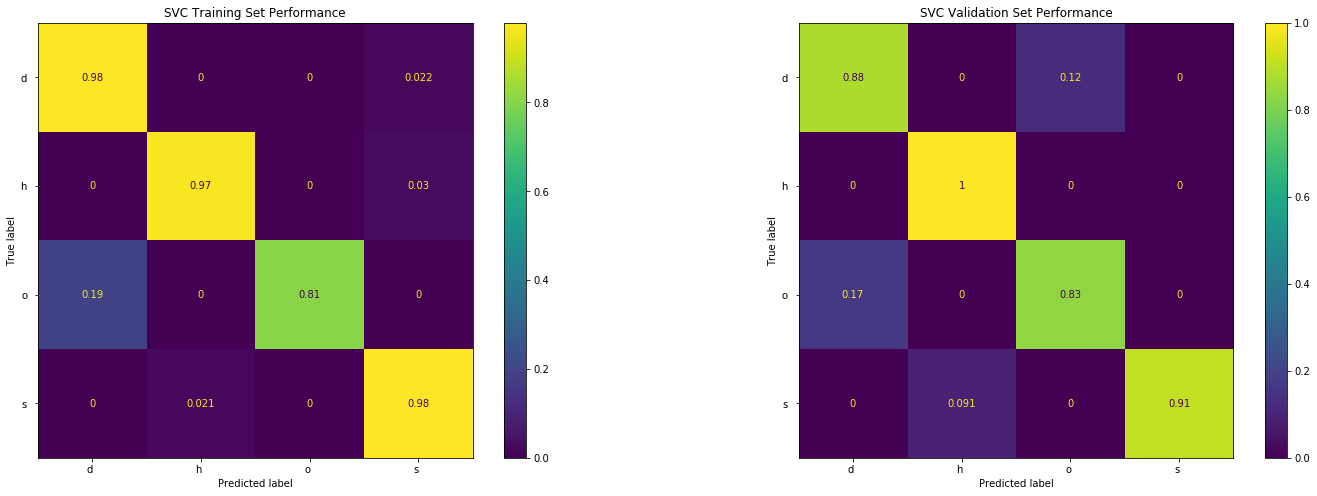


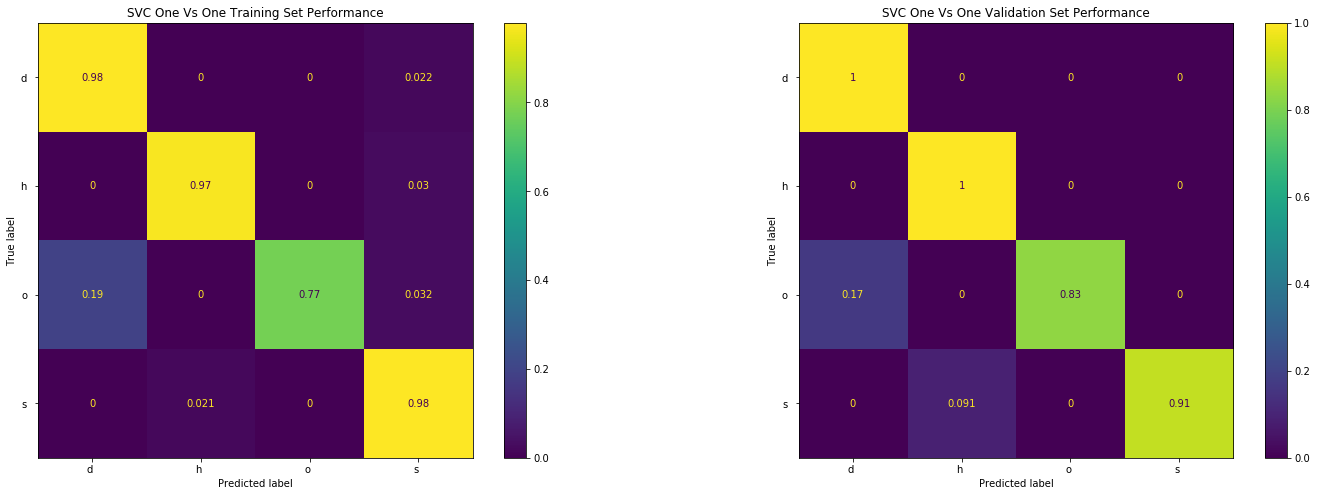


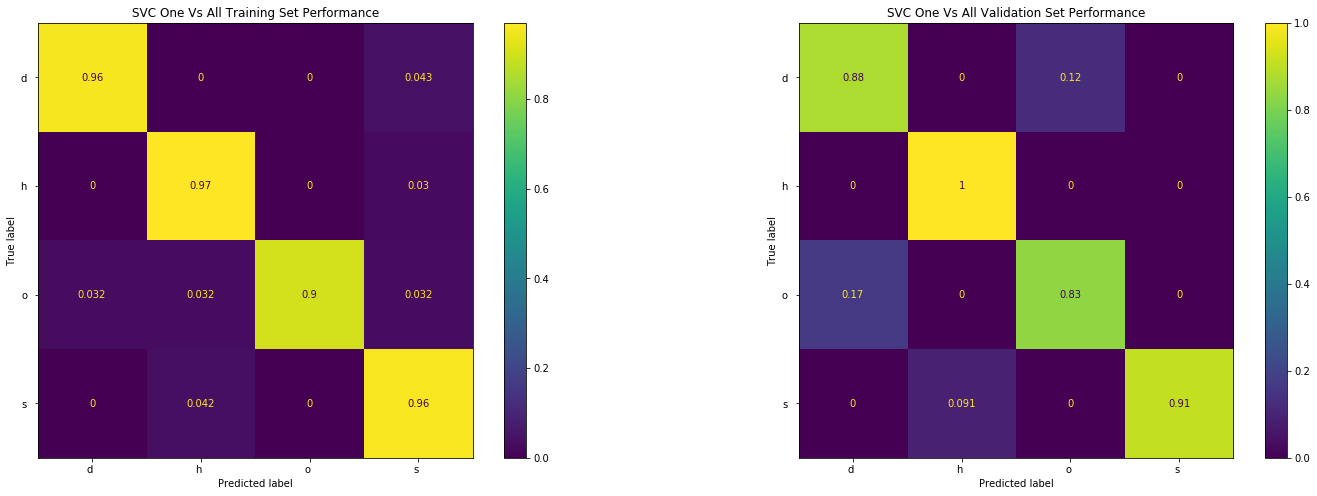




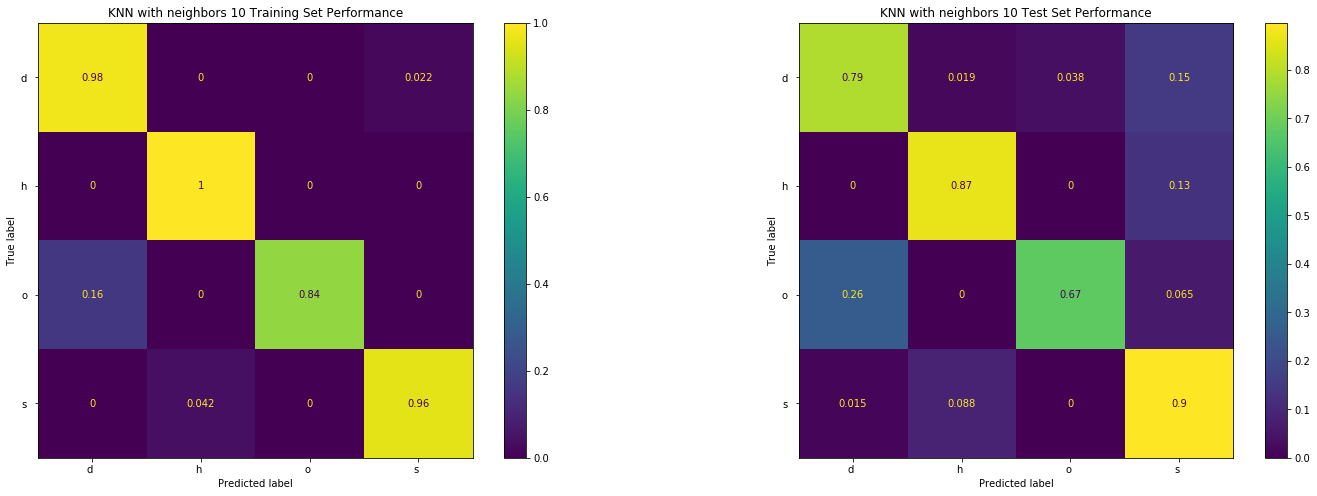
In Svc model, we use SVC only, SVC in one vs one and SVC in one vs all model to predict validation set. Turn out SVC one vs one has the best answer with 0.95 accuracies.

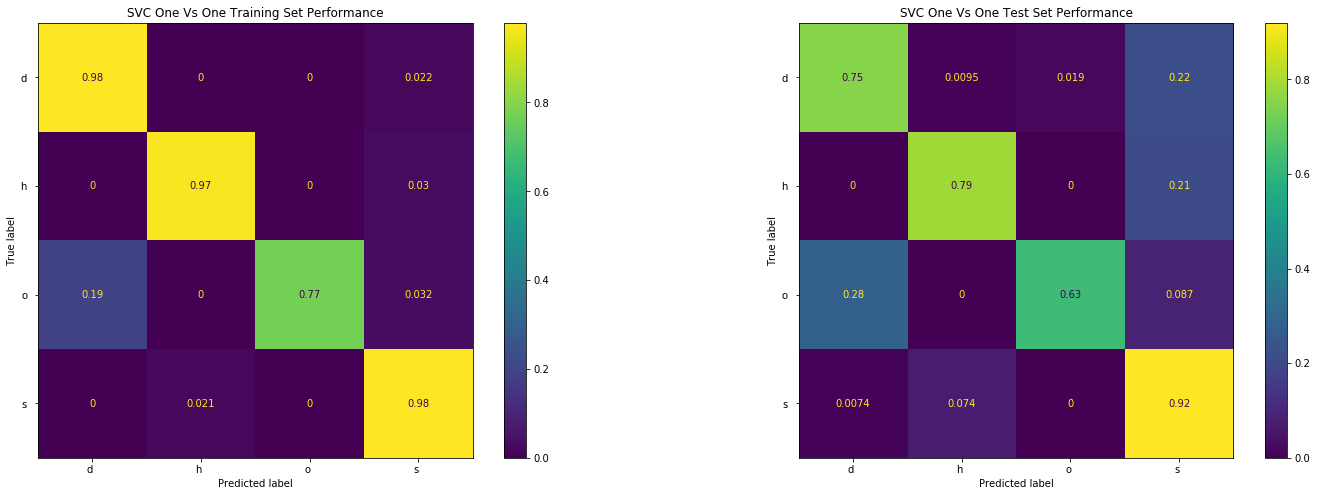


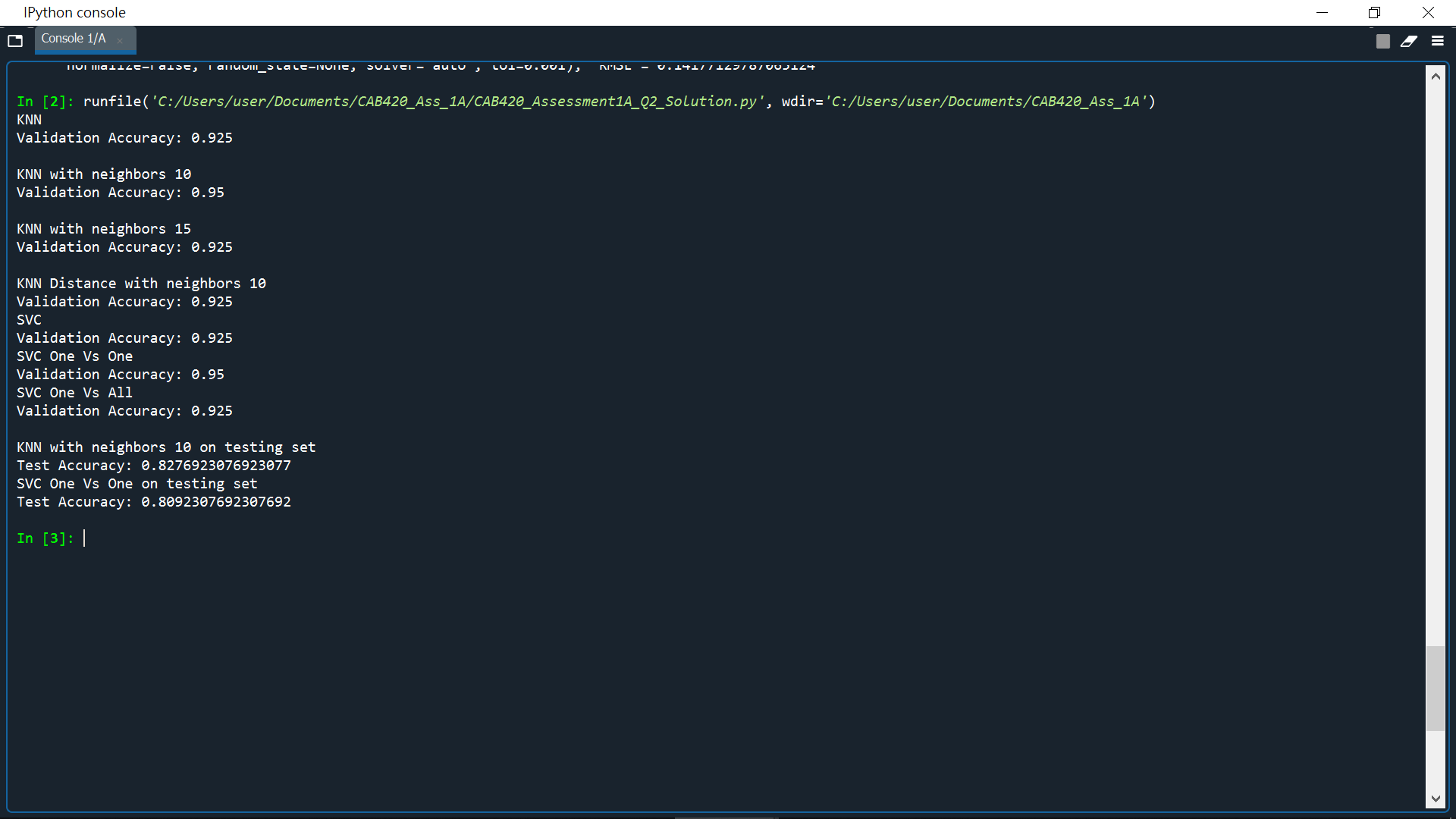




At last, Base on what we choose, we must compare the testing result on KNN with 10 neighbour and SVC one vs one model. The result shows the KNN model is slightly better on predict testing set with 2% improvement.





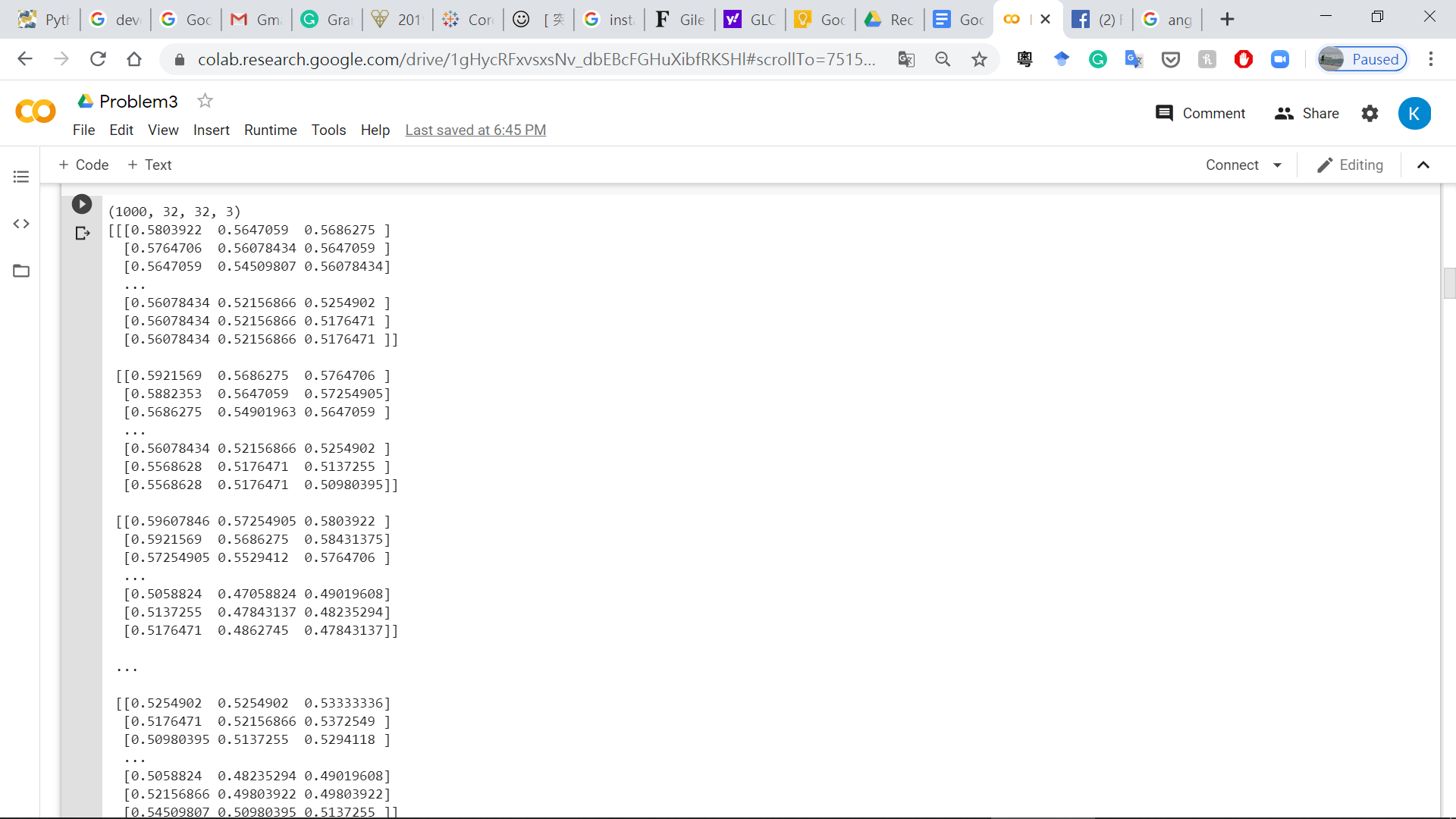


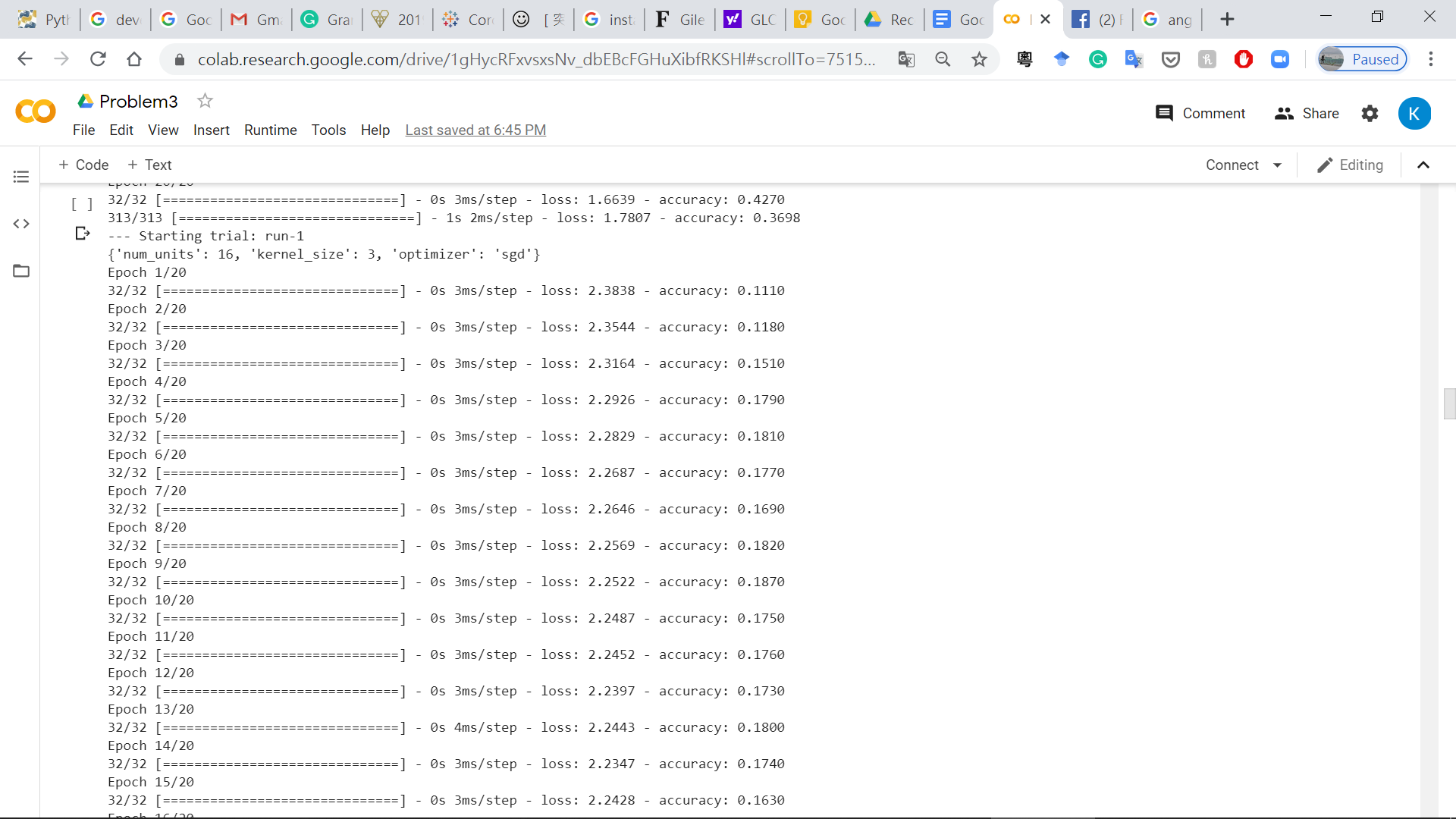
Question3)

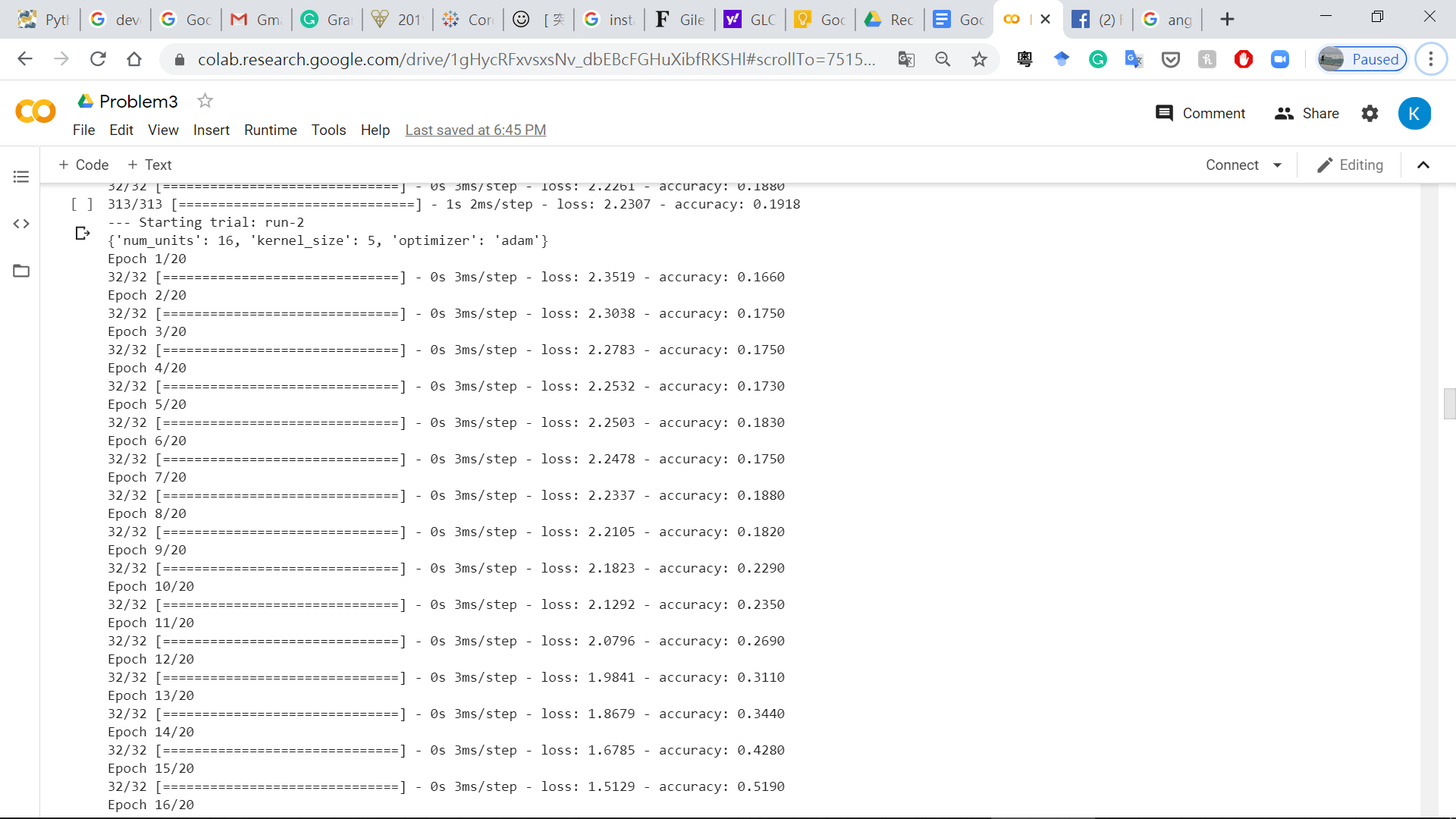
For this task is how we use limited data to train and predict numbers from street house numbers. The training model was given 100 examples and others with 1000 examples which each number were separate in average.

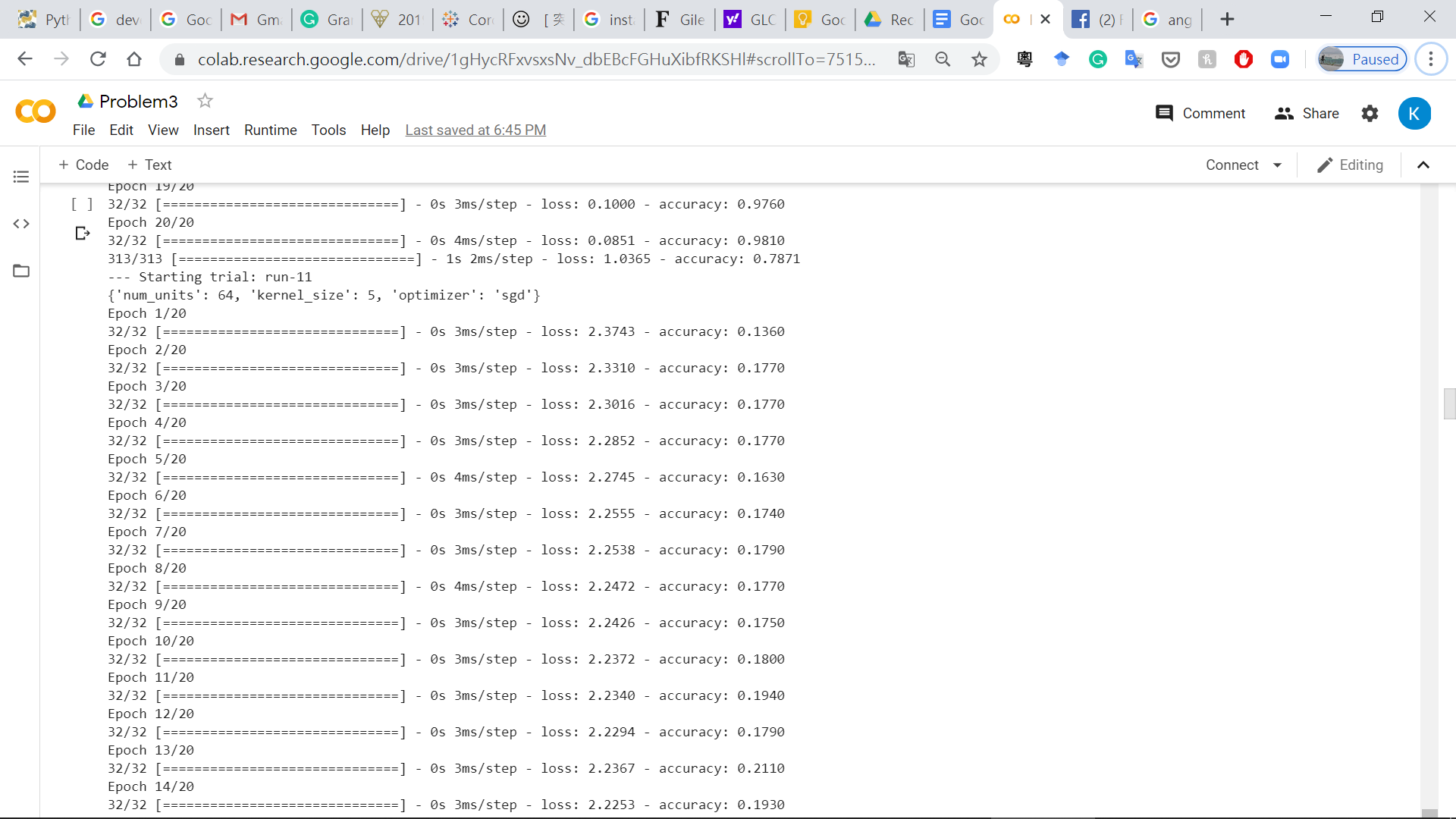


First requirement, we need training model and start build a model with non-data augmentation. According to the questions, first we need import data from mat files and transform it displayable pictures. What we have to do is reorder data set secuquence.1000,32,32,3 is the data set shape after we transform. For model training we got various size include 16 32 64 numbers for parameters and Adam optimizer would also be tested. Kernel size we would limit in 3 or 5 difference as final argument. For selection we listed above can provided the best prediction parameter and import them for training. Beside of that, our deep learning model individually using convert 2d twice for connect surround pixel. Then Max Pooling help with resize to the shape we want. As our observation, we notice photo have not much unnecessary data and in low pixel. Therefore, drop is unnecessary for us to add. Training accuracy reached over 90%. while testing accuracy also reach 80%. Provided that prediction are effective by model set 64 kernel size 5 with ADAM optimize.



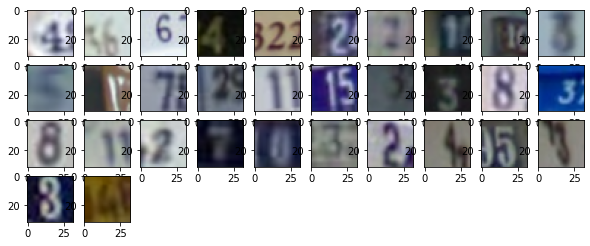


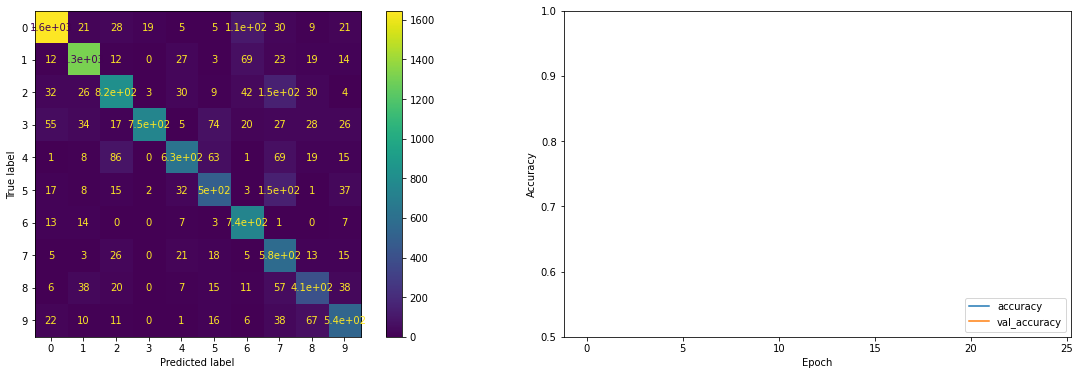




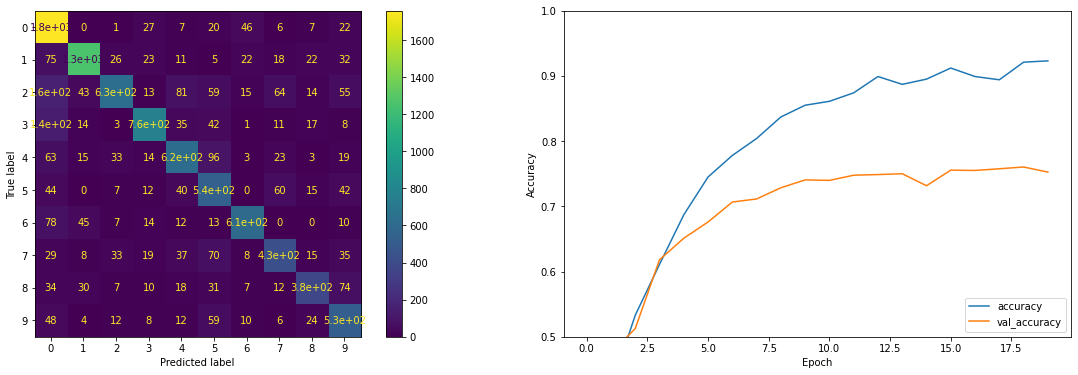


Next part, we going to use augmentation for training. By generate graph with different angle, size and zoom ratio by little between range in 5%. Below is the accuracy after augmentation. As we see graph cannot show info properly, but confusion graph provided a summarize that each number class prediction is in high percentage.





Last part with requirements in fine tuning. For training this model, we would reference lesson resource which is VGG CIFAP small. As data set provided the size of each image is 32x32, Kmnist and mnist need size with 28x28.For simple design guideline. We prefer to use the same input size CIFAP for fine tuning. Small version is enough for training as we only have limited data for resource. Overall model has surprising 90 % test accuracy and 70% validation accuracy.



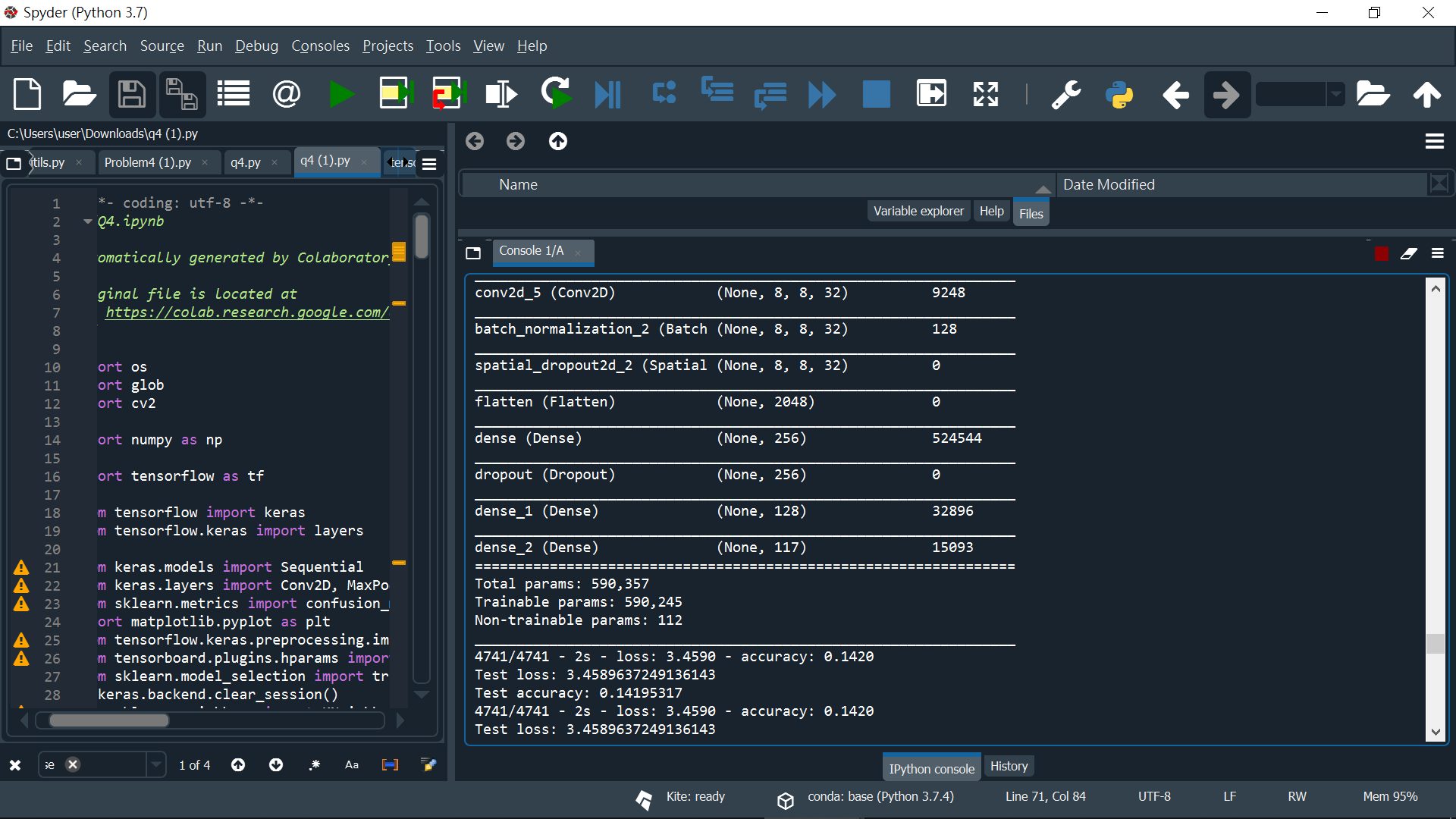
Question 4)

we analysis a huge amount of human image and training it with two methods, the first one using training, validation and testing set for evaluating. Import the UTKfile and import each domain with age, gender, race and image into data. Review each picture and check where they could correct display and reshape from 200X200 to 32x32.This purpose speed up image processing. At the end, we display graph shape how long with the age data set length.



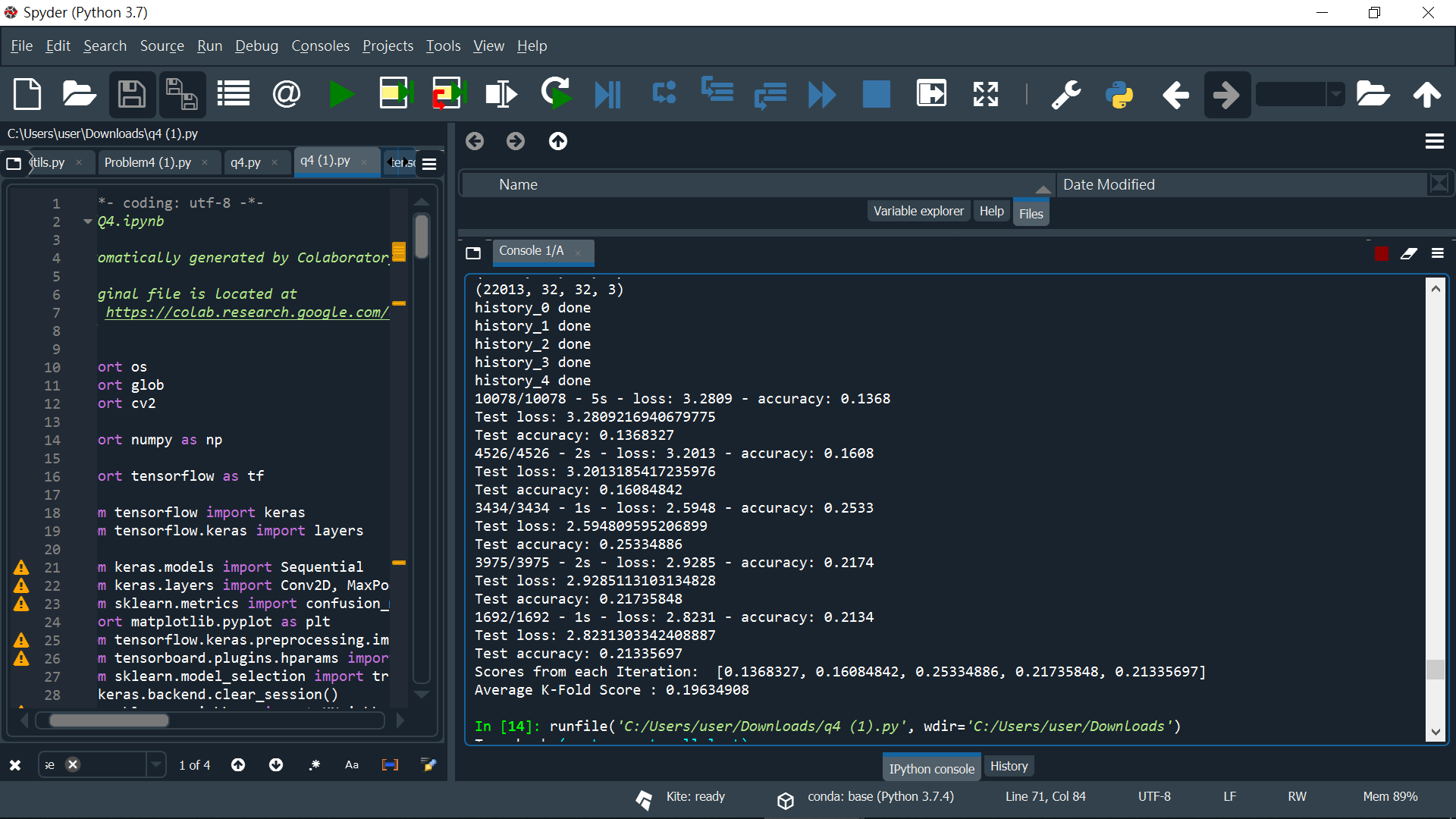
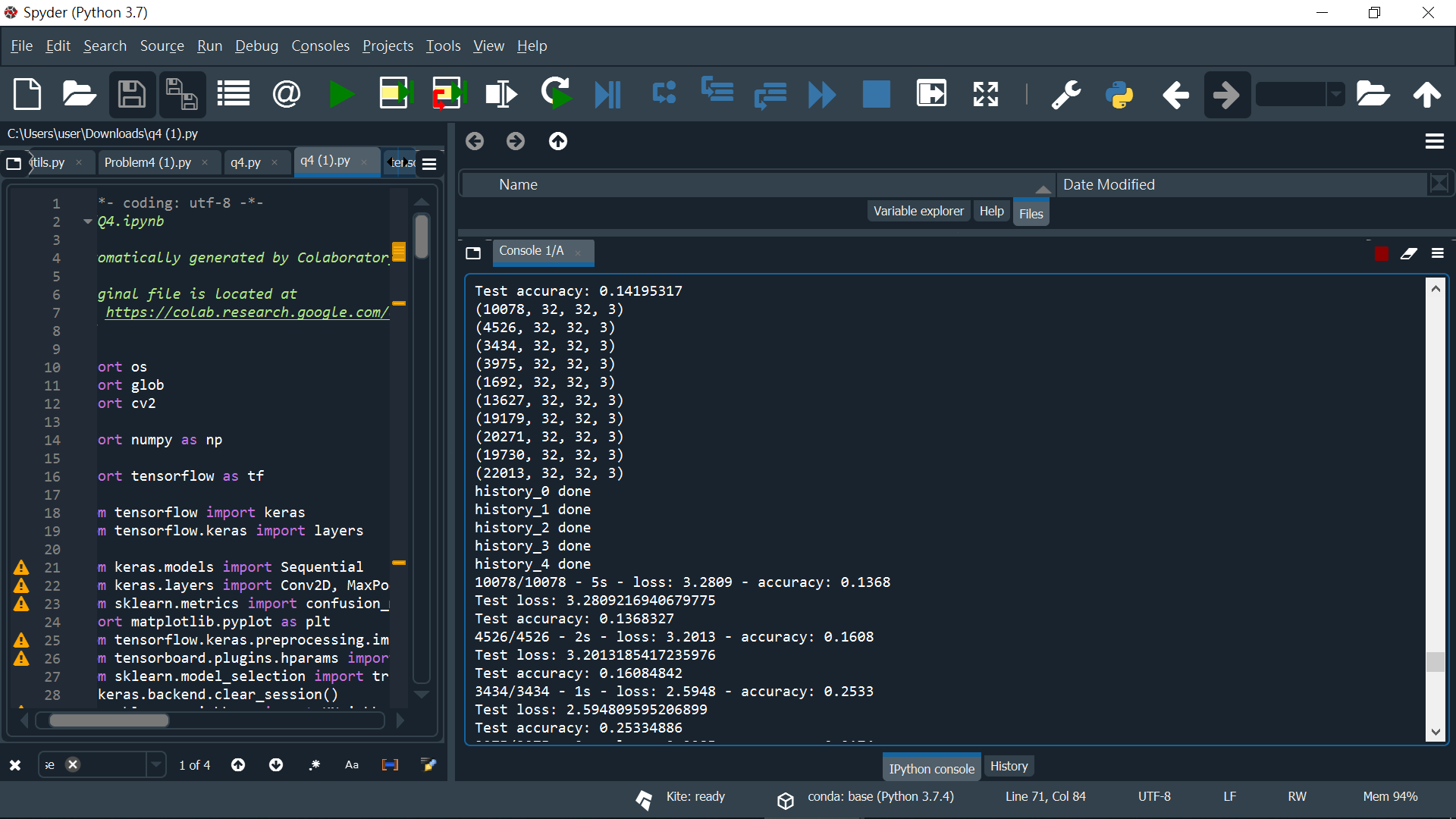


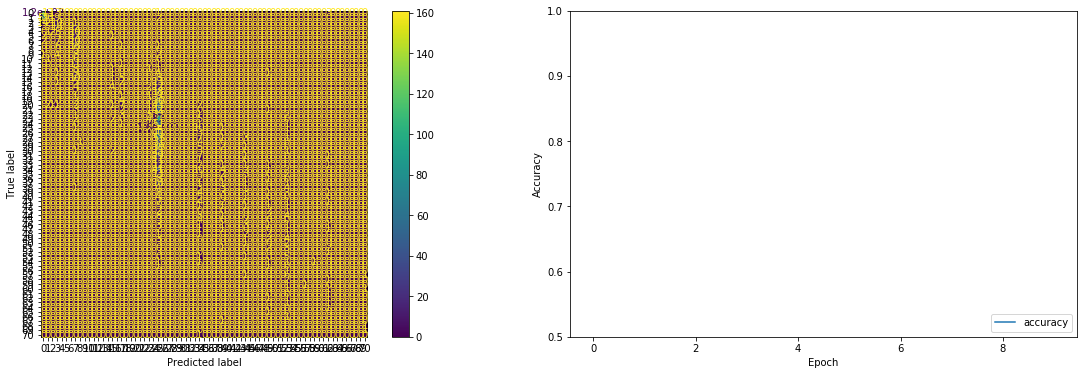
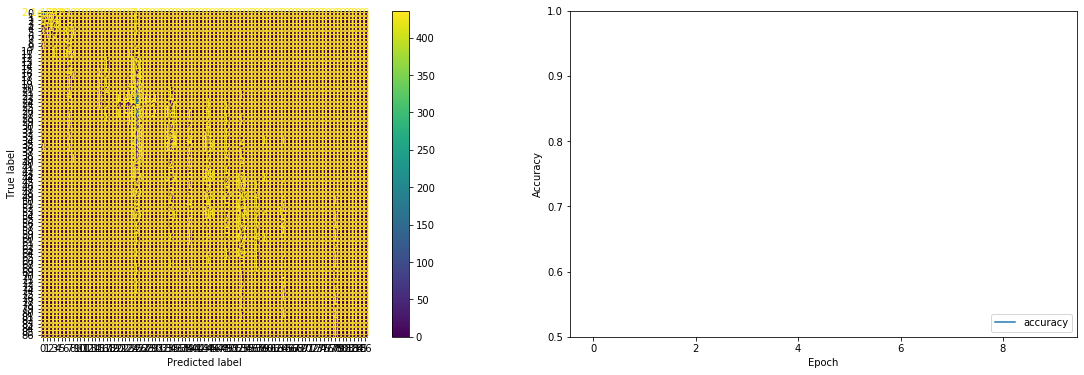
Now we could divide data into 3 parts and start to build a model. The model this time will be used with 3 convert 2d to expand the data information and randomly drop out unnecessary one to prevent overfitting, then we flatten the image and dense them into 256 and output the with dense 117.After building the model, we put our data to fit in and evaluate model accuracy.



Result show, using the first method have low with only 14% prediction rate. Which means this method is difficult to analyse how old the human in the pictures is.

Next, Cross-validation data are the next method. Similar to the last question it is necessary to import data first, however, based on the race to classify data is the additional step for pre-training. We resize the image and turn it to float 32. For the last step, we put each cross-validation data set which if race equal to 0 then cross-validation will be given those left for training into the model. Calculate each testing set accuracy. Overall, 60 % testing set have a correct prediction. Far better than the first method





Appendix Question 1:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels import api as sm

from sklearn.metrics import mean\_squared\_error

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression, Lasso, Ridge

def load\_question\_1(root\_dir=r'C:\Users\user\Downloads\CAB420\_Assessment1A\_Data\Data'):

communitiesData = pd.read\_csv(os.path.join(root\_dir, 'Q1/communities.csv'))

return communitiesData

data = load\_question\_1()

# remove index column as they are not predictive

data = data.drop([' state ',' county ', ' community ', ' communityname string', ' fold '], axis=1)

print(data.head())

# find data that involve undefine value,for example : '?' value

columns\_to\_remove = []

for column in data.columns.values:

if np.sum(data[column] == '?' ) > 0:

# add this column to the list that should be removed

columns\_to\_remove.append(column)

print(columns\_to\_remove)

print(len(columns\_to\_remove))

# remove those column

data = data.drop(columns\_to\_remove, axis=1)

print(data.shape)

# now drop any rows that contain a Nan, and deleted??

print(np.sum(data.isna(), axis=1))

print(np.sum(np.sum(data.isna(), axis=1) > 0))

nans = data.isna()

nans.to\_csv('nans.csv')

data\_filtered = data.dropna(axis=0)

# final dataset checking

print(data\_filtered.head())

print('Final dataset shape = {}'.format(data\_filtered.shape))

# split into traning,validation and testing sets

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

x\_train, x\_val, y\_train, y\_val = train\_test\_split(x\_train, y\_train, test\_size=0.3, random\_state=1)

# create the linear model

model = sm.OLS(y\_train.astype(float), x\_train.astype(float))

# fit the model without any regularisation

first\_model\_fit = model.fit()

pred = first\_model\_fit.predict(x\_val)

print('First Model RMSE = {}'.format(

np.sqrt(mean\_squared\_error(y\_val, first\_model\_fit.predict(x\_val)))))

print(first\_model\_fit.summary())

print(first\_model\_fit.params)

fig, ax = plt.subplots(figsize=(8,6))

sm.qqplot(first\_model\_fit.resid, ax=ax, line='s')

plt.title('QQ Plot for Linear Regression')

plt.show()

all\_variables = data.iloc[1,:]

# fit the model with L1 regularisation

# if the L1\_wt param is 1 representing L1 regularisation

# if L1\_wt = 0 representing L2 regularisation

alpha = 1.0

l1\_model\_fit = model.fit\_regularized(alpha=alpha, L1\_wt=1)

pred = l1\_model\_fit.predict(x\_val)

print('L1: alpha = {}, RMSE = {}'.format(

alpha, np.sqrt(mean\_squared\_error(y\_val, l1\_model\_fit.predict(x\_val)))))

# fit the model with L2 regularisation

l2\_model\_fit = model.fit\_regularized(alpha=alpha, L1\_wt=0)

pred = l2\_model\_fit.predict(x\_val)

print('L2: alpha = {}, RMSE = {}'.format(

alpha, np.sqrt(mean\_squared\_error(y\_val, l2\_model\_fit.predict(x\_val)))))

# experimenting L1 and L2 parameters for finding best RMSE

# By making a huge number on best\_rmse for being overwritten

best\_rmse = 10e12

best\_alpha = []

best\_L1\_L2 = []

# set up different ranges of alpha for L1 and L2

alpha\_list = np.linspace(0.1, 5.0, 20)

# value that diterment whether we should used L1 or L2

L1\_L2\_list = [0, 1]

for L1\_L2 in L1\_L2\_list:

for alpha in alpha\_list:

model\_cross\_fit = model.fit\_regularized(alpha=alpha, L1\_wt=0)

pred = model\_cross\_fit.predict(x\_val)

rmse = np.sqrt(mean\_squared\_error(y\_val, model\_cross\_fit.predict(x\_val)))

print('L1\_L2 = {}, alpha = {}, RMSE = {}'.format(L1\_L2, alpha, rmse))

# save model with lowest RMSE

if rmse < best\_rmse:

best\_rmse = rmse

best\_alpha = alpha

best\_L1\_L2 = L1\_L2

print('\nBest Model: L1\_L2 = {}, alpha = {}, RMSE = {}'.format(

best\_L1\_L2, best\_alpha, best\_rmse))

# create validation data

linear = LinearRegression(fit\_intercept = False).fit(X = x\_train.to\_numpy(), y = y\_train.to\_numpy())

fig = plt.figure(figsize=[25, 16])

ax = fig.add\_subplot(4, 1, 1)

ax.bar(range(len(linear.coef\_)), linear.coef\_)

ax = fig.add\_subplot(4, 1, 2)

ax.plot(linear.predict(x\_train), label='Predicted')

ax.plot(y\_train.to\_numpy(), label='Actual')

ax.set\_title('Training Data')

ax.legend()

ax = fig.add\_subplot(4, 1, 3)

ax.plot(linear.predict(x\_val), label='Predicted')

ax.plot(y\_val.to\_numpy(), label='Actual')

ax.set\_title('Validation Data')

ax.legend()

ax = fig.add\_subplot(4, 1, 4)

ax.plot(linear.predict(x\_test), label='Predicted')

ax.plot(y\_test.to\_numpy(), label='Actual')

ax.set\_title('Testing Data')

ax.legend();

#train model with Lasso Regression

lasso\_1 = Lasso(fit\_intercept=False, alpha=0.01).fit(X = x\_train.to\_numpy(), y = y\_train.to\_numpy())

lasso\_2 = Lasso(fit\_intercept=False, alpha=0.1).fit(X = x\_train.to\_numpy(), y = y\_train.to\_numpy())

lasso\_3 = Lasso(fit\_intercept=False, alpha=0.5).fit(X = x\_train.to\_numpy(), y = y\_train.to\_numpy())

#plot the graph

fig = plt.figure(figsize=[25, 16])

ax = fig.add\_subplot(4, 1, 1)

w = 0.2

pos = np.arange(0, len(linear.coef\_), 1)

ax.bar(pos - w\*2, linear.coef\_, width=w, label='linear')

ax.bar(pos - w, lasso\_1.coef\_, width=w, label='alpha=0.01')

ax.bar(pos, lasso\_2.coef\_, width=w, label='alpha=0.1')

ax.bar(pos + w, lasso\_3.coef\_, width=w, label='alpha=0.5')

ax.legend()

ax = fig.add\_subplot(4, 1, 2)

ax.plot(linear.predict(x\_train), label='linear')

ax.plot(lasso\_1.predict(x\_train), label='alpha=0.01')

ax.plot(lasso\_2.predict(x\_train), label='alpha=0.1')

ax.plot(lasso\_3.predict(x\_train), label='alpha=0.5')

ax.plot(y\_train.to\_numpy(), label='Actual')

ax.set\_title('Training Data')

ax.legend()

ax = fig.add\_subplot(4, 1, 3)

ax.plot(linear.predict(x\_val), label='linear')

ax.plot(lasso\_1.predict(x\_val), label='alpha=0.01')

ax.plot(lasso\_2.predict(x\_val), label='alpha=0.1')

ax.plot(lasso\_3.predict(x\_val), label='alpha=0.5')

ax.plot(y\_val.to\_numpy(), label='Actual')

ax.set\_title('Validation Data')

ax.legend()

ax = fig.add\_subplot(4, 1, 4)

ax.plot(linear.predict(x\_test), label='linear')

ax.plot(lasso\_1.predict(x\_test), label='alpha=0.01')

ax.plot(lasso\_2.predict(x\_test), label='alpha=0.1')

ax.plot(lasso\_3.predict(x\_test), label='alpha=0.5')

ax.plot(y\_test.to\_numpy(), label='Actual')

ax.set\_title('Testing Data')

ax.legend();

for lasso in [lasso\_1,lasso\_2]:

rmse = np.sqrt(mean\_squared\_error(y\_val, lasso.predict(x\_val)))

print('\nValudation set :{}, RMSE = {}'.format(str(lasso), rmse))

rmse = np.sqrt(mean\_squared\_error(y\_test, lasso.predict(x\_test)))

print('\nTesting set :{}, RMSE = {}'.format(str(lasso), rmse))

#train modle with Ridge Regression

ridge\_1 = Ridge(fit\_intercept=False, alpha=0.01).fit(X = x\_train.to\_numpy(), y = y\_train.to\_numpy())

ridge\_2 = Ridge(fit\_intercept=False, alpha=2.5).fit(X = x\_train.to\_numpy(), y = y\_train.to\_numpy())

ridge\_3 = Ridge(fit\_intercept=False, alpha=10).fit(X = x\_train.to\_numpy(), y = y\_train.to\_numpy())

#plot the graph

fig = plt.figure(figsize=[25, 16])

ax = fig.add\_subplot(4, 1, 1)

w = 0.2

pos = np.arange(0, len(linear.coef\_), 1)

ax.bar(pos - w\*2, linear.coef\_, width=w, label='linear')

ax.bar(pos - w, ridge\_1.coef\_, width=w, label='alpha=0.01')

ax.bar(pos, ridge\_2.coef\_, width=w, label='alpha=2.5')

ax.bar(pos + w, ridge\_3.coef\_, width=w, label='alpha=10')

ax.legend()

ax = fig.add\_subplot(4, 1, 2)

ax.plot(linear.predict(x\_train), label='linear')

ax.plot(ridge\_1.predict(x\_train), label='alpha=0.01')

ax.plot(ridge\_2.predict(x\_train), label='alpha=2.5')

ax.plot(ridge\_3.predict(x\_train), label='alpha=10')

ax.plot(y\_train.to\_numpy(), label='Actual')

ax.set\_title('Training Data')

ax.legend()

ax = fig.add\_subplot(4, 1, 3)

ax.plot(linear.predict(x\_val), label='linear')

ax.plot(ridge\_1.predict(x\_val), label='alpha=0.01')

ax.plot(ridge\_2.predict(x\_val), label='alpha=2.5')

ax.plot(ridge\_3.predict(x\_val), label='alpha=10')

ax.plot(y\_val.to\_numpy(), label='Actual')

ax.set\_title('Validation Data')

ax.legend()

ax = fig.add\_subplot(4, 1, 4)

ax.plot(linear.predict(x\_test), label='linear')

ax.plot(ridge\_1.predict(x\_test), label='alpha=0.01')

ax.plot(ridge\_2.predict(x\_test), label='alpha=2.5')

ax.plot(ridge\_3.predict(x\_test), label='alpha=10')

ax.plot(y\_test.to\_numpy(), label='Actual')

ax.set\_title('Testing Data')

ax.legend();

for ridge in [ridge\_1,ridge\_2,ridge\_3]:

rmse = np.sqrt(mean\_squared\_error(y\_val, ridge.predict(x\_val)))

print('\nValudation set :{}, RMSE = {}'.format(str(ridge), rmse))

rmse = np.sqrt(mean\_squared\_error(y\_test, ridge.predict(x\_test)))

print('\nTesting set :{}, RMSE = {}'.format(str(ridge), rmse))

Question2

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import plot\_confusion\_matrix

from sklearn.svm import SVC

from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier

# reading in the binary data set

forest\_data = pd.read\_csv(r'C:\Users\user\Downloads\CAB420\_Assessment1A\_Data\Data\Q2\training.csv')

test = pd.read\_csv(r'C:\Users\user\Downloads\CAB420\_Assessment1A\_Data\Data\Q2\testing.csv')

X\_test = test.drop('class', axis=1)

Y\_test = test['class']

# seperating into our covariates/feratures and our response variable

# can get the response variable by just dropping the `quality` column (which is our response variable)

X = forest\_data.drop('class', axis=1)

# now get the response variable by just getting the `quality` column

Y = forest\_data['class']

# lets separate it into train and test splits as well

# will use 80% for train, 20% for validation

X\_train, X\_val, Y\_train, Y\_val = train\_test\_split(X, Y, test\_size=0.2, random\_state=7)

def eval\_model\_val(model\_name, model, X\_train, Y\_train, X\_val, Y\_val):

fig = plt.figure(figsize=[25, 8])

ax = fig.add\_subplot(1, 2, 1)

conf = plot\_confusion\_matrix(model, X\_train, Y\_train, normalize='true', ax=ax)

conf.ax\_.set\_title(model\_name + ' Training Set Performance');

ax = fig.add\_subplot(1, 2, 2)

conf = plot\_confusion\_matrix(model, X\_val, Y\_val, normalize='true', ax=ax)

conf.ax\_.set\_title(model\_name + ' Validation Set Performance');

pred = model.predict(X\_val)

print('Validation Accuracy: ' + str(sum(pred == Y\_val)/len(Y\_val)))

def eval\_model(model\_name, model, X\_train, Y\_train, X\_test, Y\_test):

fig = plt.figure(figsize=[25, 8])

ax = fig.add\_subplot(1, 2, 1)

conf = plot\_confusion\_matrix(model, X\_train, Y\_train, normalize='true', ax=ax)

conf.ax\_.set\_title(model\_name + ' Training Set Performance');

ax = fig.add\_subplot(1, 2, 2)

conf = plot\_confusion\_matrix(model, X\_test, Y\_test, normalize='true', ax=ax)

conf.ax\_.set\_title(model\_name + ' Test Set Performance');

pred = model.predict(X\_test)

print('Test Accuracy: ' + str(sum(pred == Y\_test)/len(Y\_test)))

print('KNN')

cknn = KNeighborsClassifier(n\_neighbors=5)

cknn.fit(X\_train, Y\_train)

eval\_model\_val('KNN' , cknn, X\_train, Y\_train, X\_val, Y\_val)

print('\nKNN with neighbors 10')

cknn = KNeighborsClassifier(n\_neighbors=10)

cknn.fit(X\_train, Y\_train)

eval\_model\_val('KNN with neighbors 10', cknn, X\_train, Y\_train, X\_val, Y\_val)

# eval\_model(cknn, X\_train, Y\_train, X\_test, Y\_test)

print('\nKNN with neighbors 15')

cknn = KNeighborsClassifier(n\_neighbors=15)

cknn.fit(X\_train, Y\_train)

eval\_model\_val('KNN with neighbors 15', cknn, X\_train, Y\_train, X\_val, Y\_val)

print('\nKNN Distance with neighbors 10')

cknn = KNeighborsClassifier(n\_neighbors=10, weights='distance')

cknn.fit(X\_train, Y\_train)

eval\_model\_val('KNN Distance with neighbors 10', cknn, X\_train, Y\_train, X\_val, Y\_val)

# eval\_model(cknn, X\_train, Y\_train, X\_test, Y\_test)

plt.figure()

plt.hist(Y, 6)

plt.title('Histogram Total data set')

plt.figure()

plt.hist(Y\_train, 6)

plt.title('Histogram Training data set')

plt.figure()

plt.hist(Y\_val, 6)

plt.title('Histogram Validation data set')

print('SVC')

svm = SVC()

svm.fit(X\_train, Y\_train)

eval\_model\_val('SVC', svm, X\_train, Y\_train, X\_val, Y\_val)

print('SVC One Vs One')

onevsone\_svm = OneVsOneClassifier(SVC())

onevsone\_svm.fit(X\_train, Y\_train)

eval\_model\_val('SVC One Vs One', onevsone\_svm, X\_train, Y\_train, X\_val, Y\_val)

print('SVC One Vs All')

onevsall\_svm = OneVsRestClassifier(SVC())

onevsall\_svm.fit(X\_train, Y\_train)

eval\_model\_val('SVC One Vs All', onevsall\_svm, X\_train, Y\_train, X\_val, Y\_val)

#comparison of final two model which is KNN with neighbors 10 and SVC One Vs One

print('\nKNN with neighbors 10 on testing set')

cknn = KNeighborsClassifier(n\_neighbors=10)

cknn.fit(X\_train, Y\_train)

eval\_model('KNN with neighbors 10', cknn, X\_train, Y\_train, X\_test, Y\_test)

print('SVC One Vs One on testing set')

onevsone\_svm = OneVsOneClassifier(SVC())

onevsone\_svm.fit(X\_train, Y\_train)

eval\_model('SVC One Vs One', onevsone\_svm, X\_train, Y\_train, X\_test, Y\_test)

Question 3

# -\*- coding: utf-8 -\*-

"""Problem3

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1gHycRFxvsxsNv\_dbEBcFGHuXibfRKSHl

"""

from google.colab import drive

drive.mount('/content/drive')

import os

import datetime

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorboard import notebook

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

from keras.optimizers import SGD

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorboard.plugins.hparams import api as hp

tf.keras.backend.clear\_session()

from scipy.io import loadmat

#Load training and test data

test = loadmat(r"C:\Users\user\Downloads\CAB420\_Assessment1A\_Data\Data\Q3\q3\_test")

test\_X = test["test\_X"]

test\_Y = test["test\_Y"]

train = loadmat(r"C:\Users\user\Downloads\CAB420\_Assessment1A\_Data\Data\Q3\q3\_train")

train\_X = train["train\_X"]

train\_Y = train["train\_Y"]

train\_X = np.swapaxes(train\_X, 3, 0)

test\_X = np.swapaxes(test\_X, 3, 0)

train\_X = np.swapaxes(train\_X, 3, 1)

test\_X = np.swapaxes(test\_X, 3, 1)

train\_X = np.swapaxes(train\_X, 3, 2)

test\_X = np.swapaxes(test\_X, 3, 2)

train\_X = train\_X.astype('float32') / 255

test\_X = test\_X.astype('float32') / 255

fig = plt.figure(figsize=[10, 10])

for i in range(100):

ax = fig.add\_subplot(10, 10, i + 1)

ax.imshow(train\_X[i,:,:,:])

print(np.shape(train\_X))

print(train\_X[1,:,:,:])

print(np.shape(train\_Y))

HP\_NUM\_UNITS = hp.HParam('num\_units', hp.Discrete([16, 32, 64]))

HP\_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd']))

HP\_KERNEL\_SIZE = hp.HParam('kernel\_size', hp.Discrete([3, 5]))

METRIC\_ACCURACY = 'accuracy'

with tf.summary.create\_file\_writer('logs/hparam\_tuning').as\_default():

hp.hparams\_config(

hparams=[HP\_NUM\_UNITS, HP\_OPTIMIZER, HP\_KERNEL\_SIZE],

metrics=[hp.Metric(METRIC\_ACCURACY, display\_name='Accuracy')],

)

def train\_test\_model(hparams):

inputs = keras.Input(shape=(32, 32, 3, ), name='train\_X')

x = layers.Conv2D(hparams[HP\_NUM\_UNITS], (5,5), activation='relu')(inputs)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.Conv2D(hparams[HP\_NUM\_UNITS]\*2, (5,5), activation='relu')(x)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.Flatten()(x)

x = layers.Dense(hparams[HP\_NUM\_UNITS]\*2, activation='relu')(x)

x = layers.Dense(hparams[HP\_NUM\_UNITS], activation='relu')(x)

# the output

outputs = layers.Dense(11)(x)

# build the model, and print a summary

model\_cnn = keras.Model(inputs=inputs, outputs=outputs, name='cnn\_model')

model\_cnn.compile(

optimizer=keras.optimizers.Adam(),

loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'],

)

model\_cnn.fit(train\_X, train\_Y, epochs=20, ) # Run with 1 epoch to speed things up for demo purposes

\_, accuracy = model\_cnn.evaluate(test\_X, test\_Y)

return accuracy

def run(run\_dir, hparams):

with tf.summary.create\_file\_writer(run\_dir).as\_default():

hp.hparams(hparams) # record the values used in this trial

accuracy = train\_test\_model(hparams)

tf.summary.scalar(METRIC\_ACCURACY, accuracy, step=1)

session\_num = 0

for num\_units in HP\_NUM\_UNITS.domain.values:

for kernel\_size in (HP\_KERNEL\_SIZE.domain.values):

for optimizer in HP\_OPTIMIZER.domain.values:

hparams = {

HP\_NUM\_UNITS: num\_units,

HP\_KERNEL\_SIZE: kernel\_size,

HP\_OPTIMIZER: optimizer,

}

run\_name = "run-%d" % session\_num

print('--- Starting trial: %s' % run\_name)

print({h.name: hparams[h] for h in hparams})

run('logs/hparam\_tuning/' + run\_name, hparams)

session\_num += 1

"""# Part 1

Train a model from scratch, using no data augmentation, on the provided abridged

SVHN training set.

"""

def build\_model():

# our model, input in an image shape

inputs = keras.Input(shape=(32, 32, 3, ), name='train\_X')

x = layers.Conv2D(64, (5,5), activation='relu')(inputs)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.Conv2D(128, (5,5), activation='relu')(x)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.Flatten()(x)

x = layers.Dense(128, activation='relu')(x)

x = layers.Dense(64, activation='relu')(x)

# the output

outputs = layers.Dense(11)(x)

# build the model, and print a summary

model\_cnn = keras.Model(inputs=inputs, outputs=outputs, name='cnn\_model')

return model\_cnn

model\_cnn = build\_model()

model\_cnn.summary()

model\_cnn.compile(loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

optimizer=keras.optimizers.Adam(),

metrics=['accuracy'])

history = model\_cnn.fit(train\_X, train\_Y,

epochs=20,

validation\_data=(test\_X, test\_Y), verbose=False)

def eval\_model(model, history, x\_test, y\_test):

test\_scores = model.evaluate(x\_test, y\_test, verbose=2)

print('Test loss:', test\_scores[0])

print('Test accuracy:', test\_scores[1])

pred = model.predict(x\_test);

indexes = tf.argmax(pred, axis=1)

gt\_idx = tf.argmax(y\_test, axis=1)

cm = confusion\_matrix(y\_test, indexes)

fig = plt.figure(figsize=[20, 6])

ax = fig.add\_subplot(1, 2, 1)

c = ConfusionMatrixDisplay(cm, display\_labels=range(10))

c.plot(ax = ax)

ax = fig.add\_subplot(1,2,2)

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label='val\_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0.5,1])

plt.legend(loc='lower right')

eval\_model(model\_cnn, history, test\_X, test\_Y)

"""# Part 2

Train a model from scratch, using the data augmentation of your choice, on the provided abridged SVHN training set.

"""

datagen = ImageDataGenerator(# rotate between -5, +5 degrees

rotation\_range=5,

# horiziontal shift by +/- 5% of the image width

width\_shift\_range=0.05,

# vertical shift by +/- 5% of the image width

height\_shift\_range=0.05,

# range for zooming

zoom\_range=0.1)

batch = datagen.flow(train\_X, train\_Y)

fig = plt.figure(figsize=[10, 10])

for i,img in enumerate(batch[0][0]):

ax = fig.add\_subplot(10, 10, i + 1)

ax.imshow(img[:,:,:])

model\_cnn = build\_model()

model\_cnn.summary()

model\_cnn.compile(loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

optimizer=keras.optimizers.Adam(),

metrics=['accuracy'])

history = model\_cnn.fit(datagen.flow(train\_X, train\_Y),

steps\_per\_epoch=32,

epochs=25,

validation\_data=(test\_X, test\_Y), verbose=False)

eval\_model(model\_cnn, history, test\_X, test\_Y)

"""# Part 3

Fine tune an existing model, trained on another dataset used in CAB420 (such as

MNIST, KMINST or CIFAR), on the provided abridged SVHN training set. Data

augmentation may also be used if you so choose.

"""

model = keras.models.load\_model('/content/drive/My Drive/Colab Notebooks/vgg\_3stage\_CIFAR\_small.h5')

model.summary()

outputs = layers.Dense(64, activation='relu')(model.layers[-10].output)

outputs = layers.Dense(11, activation="relu")(outputs)

new\_model = keras.Model(inputs=model.input, outputs=outputs)

new\_model.summary()

for layer in new\_model.layers[:-7]:

layer.trainable = False

for layer in new\_model.layers:

print(layer, layer.trainable)

new\_model.compile(loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

optimizer=keras.optimizers.Adam(),

metrics=['accuracy'])

history = new\_model.fit(train\_X, train\_Y,

epochs=20,

validation\_data=(test\_X, test\_Y), verbose=False)

eval\_model(new\_model, history, test\_X, test\_Y)

Question 4

import os

import glob

import cv2

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay, mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorboard.plugins.hparams import api as hp

from sklearn.model\_selection import train\_test\_split

tf.keras.backend.clear\_session()

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import plot\_confusion\_matrix

path = r'C:\Users\user\Downloads\CAB420\_Assessment1A\_Data\Data\Q4\UTKFace\\*'

files = glob.glob(path)

data = []

for f in files:

d = {}

head, tail = os.path.split(f)

parts = tail.split('\_')

if (len(parts) == 4):

d['age'] = int(parts[0])

d['gender'] = int(parts[1])

d['race'] = int(parts[2])

d['image'] = cv2.imread(f)

data.append(d)

else:

print('Could not load: ' + f + '! Incorrectly formatted filename')

temp\_X = np.array([d['image'] for d in data[:]])

print('The shape of temp\_X is :',np.shape(temp\_X))

X = np.zeros((23705,32,32,3))

for i in range(23705):

try:

X[i] = cv2.resize(temp\_X[i], (32, 32))

except IndexError as e:

print('Invalid frame!')

continue

Y = np.array([d['age'] for d in data[:23705]])

X = X.astype('float32') / 255

fig = plt.figure(figsize=[10, 10])

for i in range(100):

ax = fig.add\_subplot(10, 10, i + 1)

ax.imshow(X[i,:,:,:])

print('The shape of pictures are :',np.shape(X))

print('The age set shape is :',np.shape(Y))

train\_X, test\_X, train\_Y, test\_Y = train\_test\_split(X, Y, test\_size=0.15, random\_state=7)

train\_X, val\_X, train\_Y, val\_Y = train\_test\_split(train\_X, train\_Y, test\_size=0.15, random\_state=1)

"""# Part 1

Train a model from scratch, using no data augmentation, on the provided abridged

SVHN training set.

"""

def build\_model(num\_classes):

# inputs = keras.Input(shape=(200,200,3, ), name='train\_X')

# x = layers.Conv2D(64, (5,5), activation='relu')(inputs)

# x = layers.MaxPooling2D((2, 2))(x)

# x = layers.Conv2D(128, (5,5), activation='relu')(x)

# x = layers.MaxPooling2D((2, 2))(x)

# x = layers.Flatten()(x)

# x = layers.Dense(128, activation='relu')(x)

# x = layers.Dense(64, activation='relu')(x)

# # the output

# outputs = layers.Dense(117)(x)

inputs = keras.Input(shape=(32, 32, 3, ), name='img')

x = layers.Conv2D(filters=8, kernel\_size=(3,3), padding='same', activation='relu')(inputs)

x = layers.Conv2D(filters=8, kernel\_size=(3,3), padding='same', activation='relu')(inputs)

# batch normalisation, before the non-linearity

x = layers.BatchNormalization()(x)

# max pooling, 2x2, which will downsample the image

x = layers.MaxPool2D(pool\_size=(2, 2))(x)

# rinse and repeat with 2D convs, batch norm,and max pool

x = layers.Conv2D(filters=16, kernel\_size=(3,3), padding='same', activation='relu')(x)

x = layers.Conv2D(filters=16, kernel\_size=(3,3), padding='same', activation='relu')(x)

x = layers.BatchNormalization()(x)

x = layers.MaxPool2D(pool\_size=(2, 2))(x)

# final conv2d, batch norm

x = layers.Conv2D(filters=32, kernel\_size=(3,3), padding='same', activation='relu')(x)

x = layers.Conv2D(filters=32, kernel\_size=(3,3), padding='same', activation='relu')(x)

x = layers.BatchNormalization()(x)

# flatten layer

x = layers.Flatten()(x)

# we'll use a couple of dense layers here, mainly so that we can show what another dropout layer looks like

# in the middle

x = layers.Dense(256, activation='relu')(x)

x = layers.Dense(128, activation='relu')(x)

# the output

outputs = layers.Dense(117, activation=None)(x)

model\_cnn = keras.Model(inputs=inputs, outputs=outputs, name='kmnist\_cnn\_model')

return model\_cnn

model\_cnn = build\_model(117)

model\_cnn.summary()

model\_cnn.compile(loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

optimizer=keras.optimizers.RMSprop(),

metrics=['accuracy'])

history = model\_cnn.fit(train\_X, train\_Y,

epochs=10,

validation\_data=(val\_X, val\_Y),verbose=False)

def eval\_model(model, history, x\_test, y\_test):

test\_scores = model.evaluate(x\_test, y\_test, verbose=2)

print('Test loss:', test\_scores[0])

print('Test accuracy:', test\_scores[1])

pred = model.predict(x\_test);

indexes = tf.argmax(pred, axis=1)

cm = confusion\_matrix(y\_test, indexes)

fig = plt.figure(figsize=[20, 6])

ax = fig.add\_subplot(1, 2, 1)

c = ConfusionMatrixDisplay(cm, display\_labels=range(117))

c.plot(ax = ax)

ax = fig.add\_subplot(1,2,2)

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label='val\_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0.5,1])

plt.legend(loc='lower right')

eval\_model(model\_cnn, history, test\_X, test\_Y)

def eval\_model\_no\_val(model, history, x\_test, y\_test):

test\_scores = model.evaluate(x\_test, y\_test, verbose=2)

print('Test loss:', test\_scores[0])

print('Test accuracy:', test\_scores[1])

pred = model.predict(x\_test);

indexes = tf.argmax(pred, axis=1)

cm = confusion\_matrix(y\_test, indexes)

fig = plt.figure(figsize=[20, 6])

ax = fig.add\_subplot(1, 2, 1)

c = ConfusionMatrixDisplay(cm, display\_labels=range(117))

c.plot(ax = ax)

ax = fig.add\_subplot(1,2,2)

plt.plot(history.history['accuracy'], label='accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0.5,1])

plt.legend(loc='lower right')

return test\_scores[1]

eval\_model(model\_cnn, history, test\_X, test\_Y)

"""# Part 2

Train a model from cross-fold evaluation protocol based on the race annotation.

"""

temp\_X\_0 = []

temp\_X\_1 = []

temp\_X\_2 = []

temp\_X\_3 = []

temp\_X\_4 = []

temp\_Y\_0 = []

temp\_Y\_1 = []

temp\_Y\_2 = []

temp\_Y\_3 = []

temp\_Y\_4 = []

for d in data[:23705]:

try:

if d['race'] == 0:

temp\_X\_0.append(cv2.resize(d['image'], (32, 32)))

temp\_Y\_0.append(d['age'])

if d['race'] == 1:

temp\_X\_1.append(cv2.resize(d['image'], (32, 32)))

temp\_Y\_1.append(d['age'])

if d['race'] == 2:

temp\_X\_2.append(cv2.resize(d['image'], (32, 32)))

temp\_Y\_2.append(d['age'])

if d['race'] == 3:

temp\_X\_3.append(cv2.resize(d['image'], (32, 32)))

temp\_Y\_3.append(d['age'])

if d['race'] == 4:

temp\_X\_4.append(cv2.resize(d['image'], (32, 32)))

temp\_Y\_4.append(d['age'])

except cv2.error as e:

print('Invalid frame!')

cv2.waitKey()

print(np.array(temp\_X\_0).shape)

print(np.array(temp\_X\_1).shape)

print(np.array(temp\_X\_2).shape)

print(np.array(temp\_X\_3).shape)

print(np.array(temp\_X\_4).shape)

temp\_Y\_0 = np.array(temp\_Y\_0)

temp\_Y\_1 = np.array(temp\_Y\_1)

temp\_Y\_2 = np.array(temp\_Y\_2)

temp\_Y\_3 = np.array(temp\_Y\_3)

temp\_Y\_4 = np.array(temp\_Y\_4)

temp\_X\_0 = np.array(temp\_X\_0).astype('float32') / 255

temp\_X\_1 = np.array(temp\_X\_1).astype('float32') / 255

temp\_X\_2 = np.array(temp\_X\_2).astype('float32') / 255

temp\_X\_3 = np.array(temp\_X\_3).astype('float32') / 255

temp\_X\_4 = np.array(temp\_X\_4).astype('float32') / 255

cross\_set\_0\_train\_x = np.concatenate((temp\_X\_1,temp\_X\_2,temp\_X\_3,temp\_X\_4),axis=0)

cross\_set\_0\_train\_y = np.concatenate((temp\_Y\_1,temp\_Y\_2,temp\_Y\_3,temp\_Y\_4),axis=0)

cross\_set\_1\_train\_x = np.concatenate((temp\_X\_0,temp\_X\_2,temp\_X\_3,temp\_X\_4),axis=0)

cross\_set\_1\_train\_y = np.concatenate((temp\_Y\_0,temp\_Y\_2,temp\_Y\_3,temp\_Y\_4),axis=0)

cross\_set\_2\_train\_x = np.concatenate((temp\_X\_1,temp\_X\_0,temp\_X\_3,temp\_X\_4),axis=0)

cross\_set\_2\_train\_y = np.concatenate((temp\_Y\_1,temp\_Y\_0,temp\_Y\_3,temp\_Y\_4),axis=0)

cross\_set\_3\_train\_x = np.concatenate((temp\_X\_1,temp\_X\_2,temp\_X\_0,temp\_X\_4),axis=0)

cross\_set\_3\_train\_y = np.concatenate((temp\_Y\_1,temp\_Y\_2,temp\_Y\_0,temp\_Y\_4),axis=0)

cross\_set\_4\_train\_x = np.concatenate((temp\_X\_1,temp\_X\_2,temp\_X\_3,temp\_X\_0),axis=0)

cross\_set\_4\_train\_y = np.concatenate((temp\_Y\_1,temp\_Y\_2,temp\_Y\_3,temp\_Y\_0),axis=0)

print(cross\_set\_0\_train\_x.shape)

print(cross\_set\_1\_train\_x.shape)

print(cross\_set\_2\_train\_x.shape)

print(cross\_set\_3\_train\_x.shape)

print(cross\_set\_4\_train\_x.shape)

history\_0 = model\_cnn.fit(cross\_set\_0\_train\_x, cross\_set\_0\_train\_y,

epochs=10,

verbose=False)

print('history\_0 done')

history\_1 = model\_cnn.fit(cross\_set\_1\_train\_x, cross\_set\_1\_train\_y,

epochs=10,

verbose=False)

print('history\_1 done')

history\_2 = model\_cnn.fit(cross\_set\_2\_train\_x, cross\_set\_2\_train\_y,

epochs=10,

verbose=False)

print('history\_2 done')

history\_3 = model\_cnn.fit(cross\_set\_3\_train\_x, cross\_set\_3\_train\_y,

epochs=10,

verbose=False)

print('history\_3 done')

history\_4 = model\_cnn.fit(cross\_set\_4\_train\_x, cross\_set\_4\_train\_y,

epochs=10,

verbose=False)

print('history\_4 done')

scores = []

scores.append(eval\_model\_no\_val(model\_cnn, history\_0, temp\_X\_0, temp\_Y\_0))

scores.append(eval\_model\_no\_val(model\_cnn, history\_1, temp\_X\_1, temp\_Y\_1))

scores.append(eval\_model\_no\_val(model\_cnn, history\_2, temp\_X\_2, temp\_Y\_2))

scores.append(eval\_model\_no\_val(model\_cnn, history\_3, temp\_X\_3, temp\_Y\_3))

scores.append(eval\_model\_no\_val(model\_cnn, history\_4, temp\_X\_4, temp\_Y\_4))

print('Scores from each Iteration: ', scores)

print('Average K-Fold Score :' , np.mean(scores))