Group No 297

Group Member Names:

1. Chakradhar Kowsik 2020FC04737
2. Piyush Pawar 2021FC04728
3. Santhosh Mathai 2021FA04008

1. Problem Statement

Students are expected to identify a classification / regression problem of your choice. You have to detail the problem under this heading which basically addresses the following questions.

1. What is the problem that you are trying to solve?
2. What kind of prediction (classification / regression) task are you performing?

ENSURE THAT YOU ARE USING NUMERICAL / CATEGORICAL DATA only.

DO NOT use images or textual data.

Score: 1 Mark in total (0.5 mark each)

----------Type the answers below this line----------

1. Problem Statement

Developing a Deep learning model for a Bank Marketing Dataset, With the given data to find out whether the customer will subscribe to a Term Deposite by analysing a Bank Marketing Dataset.

2. Problem Kind

We are trying to classify customer into two groups either subscriber or not so we are trying to solve **classification** problem. To be more precise it is binary classification problem.

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import os.path
import time
from datetime import datetime,date,time
import requests
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder,MinMa
from sklearn.decomposition import PCA
```

```
from sklearn.compose import make_column_selector as selector
from sklearn.metrics  import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from zipfile import ZipFile
import itertools

from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D
from keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

from keras import optimizers

import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
import random as rn
```

In [2]:

```
class Util:
    def __init__(self):
        pass
    def downloadfile(self,data_zip_url,zip_filename):
        start_time = datetime.now()
        if not os.path.exists(zip_filename):
            print('Downloading the zip file from source {}.'.format(data_zip_url))
            zip_file = None
            try:
                resp=requests.get(data_zip_url,verify=False)
                if resp.status_code ==200:
                    print('Downloaded file successfully.')
                    print('Saving Zipfile {} to disk'.format(zip_filename))
                    zip_file = open(zip_filename,'wb')
                    zip_file.write(resp.content)
                else:
                    print('Downloading file failed!')
            except ex:
                print('Downloading file failed!')
            finally:
                if zip_file != None:
                    zip_file.close()
        else:
            print('File {} already downloaded'.format(zip_filename))
        end_time = datetime.now()
        print('Time taken to download and save is {}'.format(end_time- start_time))

    def createfile(self,csv_filename):
        start_time = datetime.now()
        if not os.path.exists(csv_filename):
            csv_file = None
            zip_csv_file = None
            try:
                zip_csv_file = zip.open('bank-additional/'+csv_filename,'r')
                print('Creating file {}.'.format(csv_filename))
                csv_file = open(csv_filename,'wb')
                csv_file.write(zip_csv_file.read())
            except ex:
                print('An error occurred while creating file {}'.format(csv_filenar
            finally:
                if csv_file != None:
```

```
                    csv_file.close()
                if zip_csv_file != None:
                    zip_csv_file.close()
        else:
            print('File {} already exists in current directory'.format(csv_filename
        end_time = datetime.now()
        print('Time taken to download and save is {}'.format(end_time- start_time))
util_obj= Util()
```

2. Data Acquisition

For the problem identified by you, students have to find the data source themselves from any data source.

2.1 Download the data directly

In [3]:
```
##---------Type the code below this line-----------------##
```

In [4]:
```
data_zip_url='https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-
zip_filename = 'bank-additional.zip'
util_obj.downloadfile(data_zip_url,zip_filename)
```

```
File bank-additional.zip already downloaded
Time taken to download and save is 0:00:00.000332
```

In [5]:
```
zip= ZipFile(zip_filename)
print('List of file in zip {}'.format(zip_filename))
zip.printdir()
```

```
List of file in zip bank-additional.zip
File Name                                             Modified             Size
bank-additional/                               2014-03-26 11:28:00            0
bank-additional/.DS_Store                      2014-03-25 10:52:16         6148
__MACOSX/                                      2014-03-26 11:28:12            0
__MACOSX/bank-additional/                      2014-03-26 11:28:12            0
__MACOSX/bank-additional/._.DS_Store           2014-03-25 10:52:16           82
bank-additional/.Rhistory                      2014-03-25 16:27:14         3943
bank-additional/bank-additional-full.csv       2014-03-26 11:22:30      5834924
bank-additional/bank-additional-names.txt      2014-03-26 11:27:36         5458
bank-additional/bank-additional.csv            2014-03-26 11:23:34       583898
__MACOSX/._bank-additional                     2014-03-26 11:28:00          205
```

In [6]:
```
csv_filename = 'bank-additional-full.csv'
util_obj.createfile(csv_filename)
```

```
Creating file bank-additional-full.csv.
Time taken to download and save is 0:00:00.023591
```

2.2 Code for converting the above downloaded data into a form suitable for DL

Since in the data set it is indicated that None is replaced in data set with unknown, so while reading making it as None.

In [7]:
```
##---------Type the code below this line-----------------##
```

In [8]:
```
df = pd.read_csv(csv_filename,delimiter=";",index_col=None,na_values=['unknown'])
df.head()
```

Out[8]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon
1	57	services	married	high.school	NaN	no	no	telephone	may	mon
2	37	services	married	high.school	no	yes	no	telephone	may	mon
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon
4	56	services	married	high.school	no	no	yes	telephone	may	mon

5 rows × 21 columns

2.3 Write your observations from the above.

1. Size of the dataset
2. What type of data attributes are there?

Score: 2 Mark

----------Type the answers below this line----------

In [9]:
```
print(df.shape)
print(df.info())
print(df.describe())
print(df.size)
```

```
(41188, 21)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             40858 non-null  object 
 2   marital         41108 non-null  object 
 3   education       39457 non-null  object 
 4   default         32591 non-null  object 
 5   housing         40198 non-null  object 
 6   loan            40198 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
None
                age      duration      campaign         pdays      previous  \
count  41188.00000  41188.000000  41188.000000  41188.000000  41188.000000   
mean      40.02406    258.285010      2.567593    962.475454      0.172963   
std       10.42125    259.279249      2.770014    186.910907      0.494901   
min       17.00000      0.000000      1.000000      0.000000      0.000000   
25%       32.00000    102.000000      1.000000    999.000000      0.000000   
50%       38.00000    180.000000      2.000000    999.000000      0.000000   
75%       47.00000    319.000000      3.000000    999.000000      0.000000   
max       98.00000   4918.000000     56.000000    999.000000      7.000000   

       emp.var.rate  cons.price.idx  cons.conf.idx     euribor3m   nr.employed  
count  41188.000000    41188.000000   41188.000000  41188.000000  41188.000000  
mean       0.081886       93.575664     -40.502600      3.621291   5167.035911  
std        1.570960        0.578840       4.628198      1.734447     72.251528  
min       -3.400000       92.201000     -50.800000      0.634000   4963.600000  
25%       -1.800000       93.075000     -42.700000      1.344000   5099.100000  
50%        1.100000       93.749000     -41.800000      4.857000   5191.000000  
75%        1.400000       93.994000     -36.400000      4.961000   5228.100000  
max        1.400000       94.767000     -26.900000      5.045000   5228.100000  
864948
```

Bank client data:

1. age (numeric)
2. job : type of job (categorical: "admin.","blue-collar","entrepreneur","housemaid","management","retired","self-employed","services","student","technician","unemployed","unknown")

3. marital : marital status (categorical: "divorced","married","single","unknown"; note: "divorced" means divorced or widowed)
4. education (categorical: "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degr
5. default: has credit in default? (categorical: "no","yes","unknown")
6. housing: has housing loan? (categorical: "no","yes","unknown")
7. loan: has personal loan? (categorical: "no","yes","unknown")
8. contact: contact communication type (categorical: "cellular","telephone")
9. month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
10. day_of_week: last contact day of the week (categorical: "mon","tue","wed","thu","fri")
11. duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y="no"). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
12. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14. previous: number of contacts performed before this campaign and for this client (numeric)
15. poutcome: outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")
16. emp.var.rate: employment variation rate. quarterly indicator (numeric)
17. cons.price.idx: consumer price index. monthly indicator (numeric)
18. cons.conf.idx: consumer confidence index. monthly indicator (numeric)
19. euribor3m: euribor 3 month rate. daily indicator (numeric)
20. nr.employed: number of employees. quarterly indicator (numeric)

Output variable (desired target):

21. y. has the client subscribed a term deposit? (binary: "yes","no")

3. Data Preparation

Perform the data prepracessing that is required for the data that you have downloaded.

3.1 Apply techiniques

to remove duplicate data
to impute or remove missing data
to remove data inconsistencies

IF ANY

In [10]:
```
null_cols=[]

for col in df.columns:
    #print(df[col].unique())
    null_count = df[col].isna().sum()
    if null_count>0:
        print(col, null_count)
        null_cols.append(col)
```

```
job 330
marital 80
education 1731
default 8597
housing 990
loan 990
```

In [11]:
```
##---------Type the code below this line-----------------##
```

In [12]:
```
df.apply(pd.value_counts)
```

Out[12]:

	age	job	marital	education	default	housing	loan	contact	month	day
-50.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
-50.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
-49.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
-47.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
-46.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	
tue	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
unemployed	NaN	1014.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
university.degree	NaN	NaN	NaN	12168.0	NaN	NaN	NaN	NaN	NaN	
wed	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
yes	NaN	NaN	NaN	NaN	3.0	21576.0	6248.0	NaN	NaN	

1973 rows × 21 columns

In [13]:
```
mdf= df.copy()
# We need to drop this duration column as this is identical to output parameter
#i.e. duration>0 then subscribed but this will be noted
#only when user calls so if he is saying greater than 0 we obviously know he has su
mdf = mdf.drop(['duration'],axis=1)
mdf = mdf.dropna()
print('The {} % of original data set contains all information'.format(round(mdf.sha
print('The columns considered for next step {}'.format(mdf.columns))
len(mdf.columns)
```

```
The 74.02 % of original data set contains all information
The columns considered for next step Index(['age', 'job', 'marital', 'education',
'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

Out[13]: 20

In [14]:
```
display(mdf)
```

	age	job	marital	education	default	housing	loan	contact	month	da
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	
6	59	admin.	married	professional.course	no	no	no	telephone	may	
...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	

30488 rows × 20 columns

In [15]:
```
##---------Type the code below this line-----------------##
```

3.2 Encode categorical data

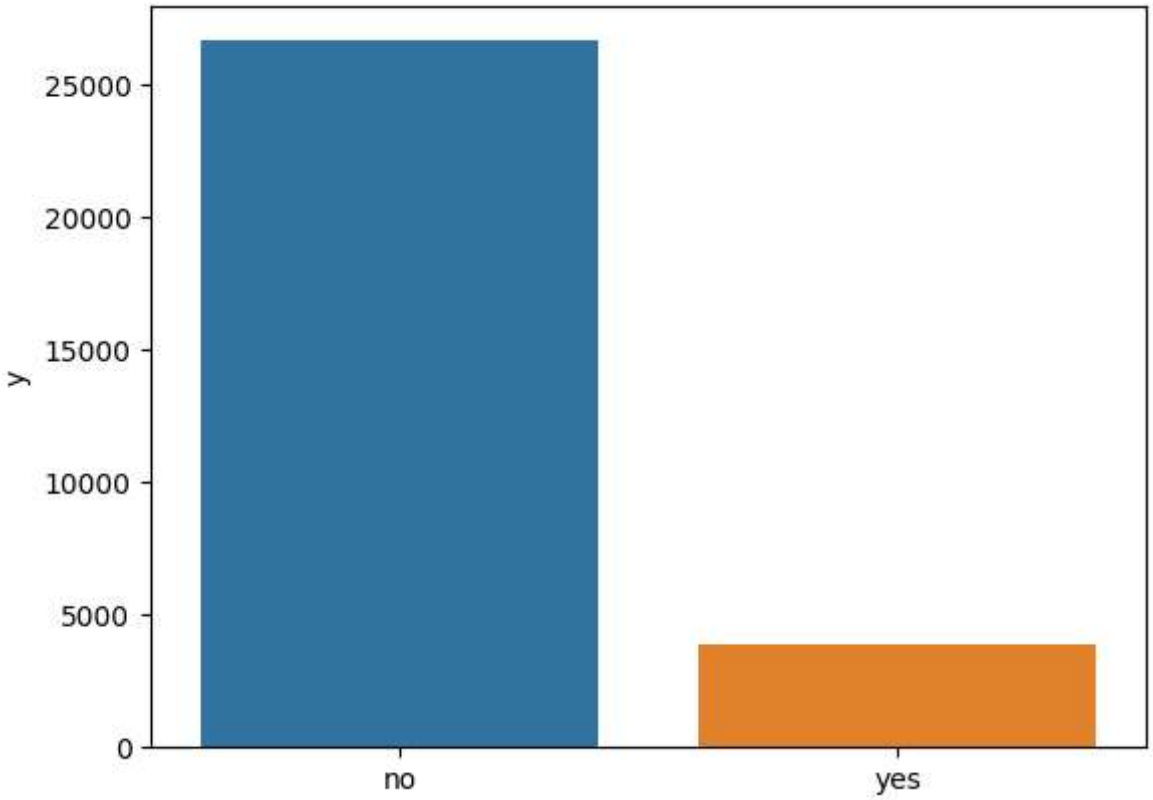
In [ ]:
```
```

In [16]:
```
#identify categorical columns
categorical_columns_selector = selector(dtype_include=object)
categorical_columns = categorical_columns_selector(mdf)
categorical_columns
```

Out[16]:
```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'poutcome',
 'y']
```

In [17]:
```
sns.barplot(x=mdf['y'].value_counts().index,y=mdf['y'].value_counts())
```

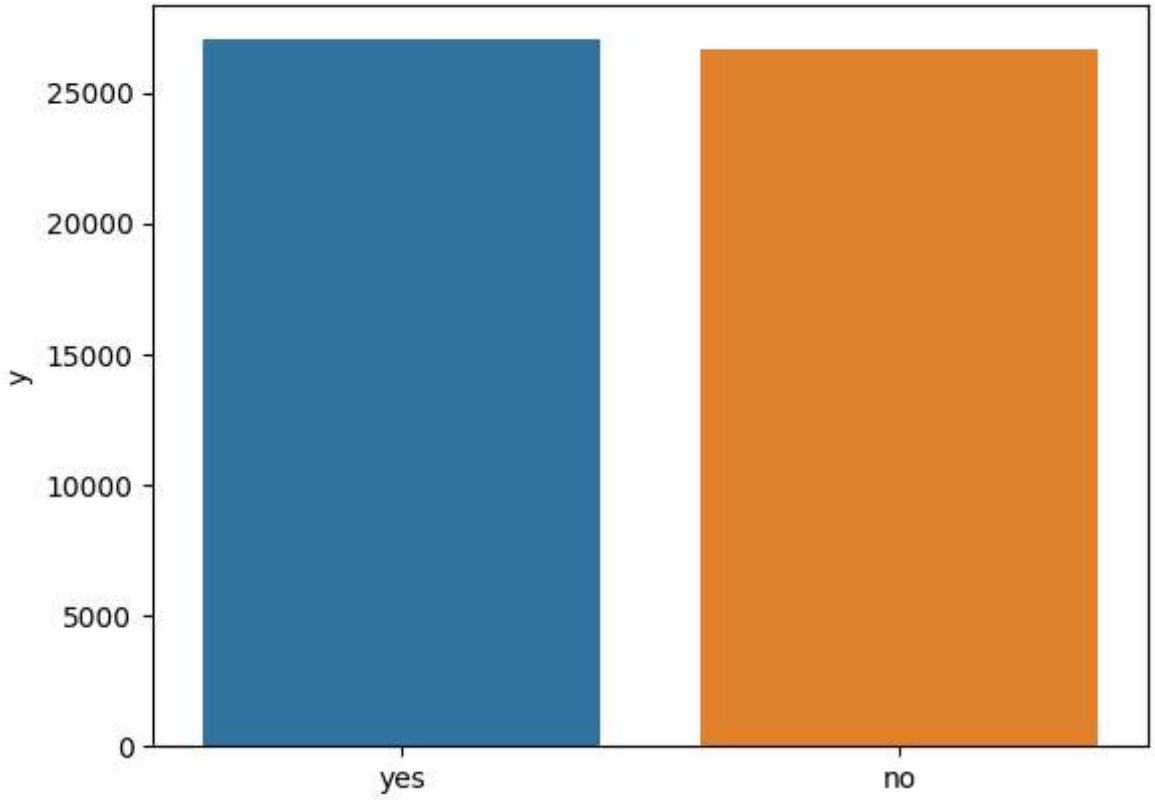
Out[17]:
```
<AxesSubplot: ylabel='y'>
```



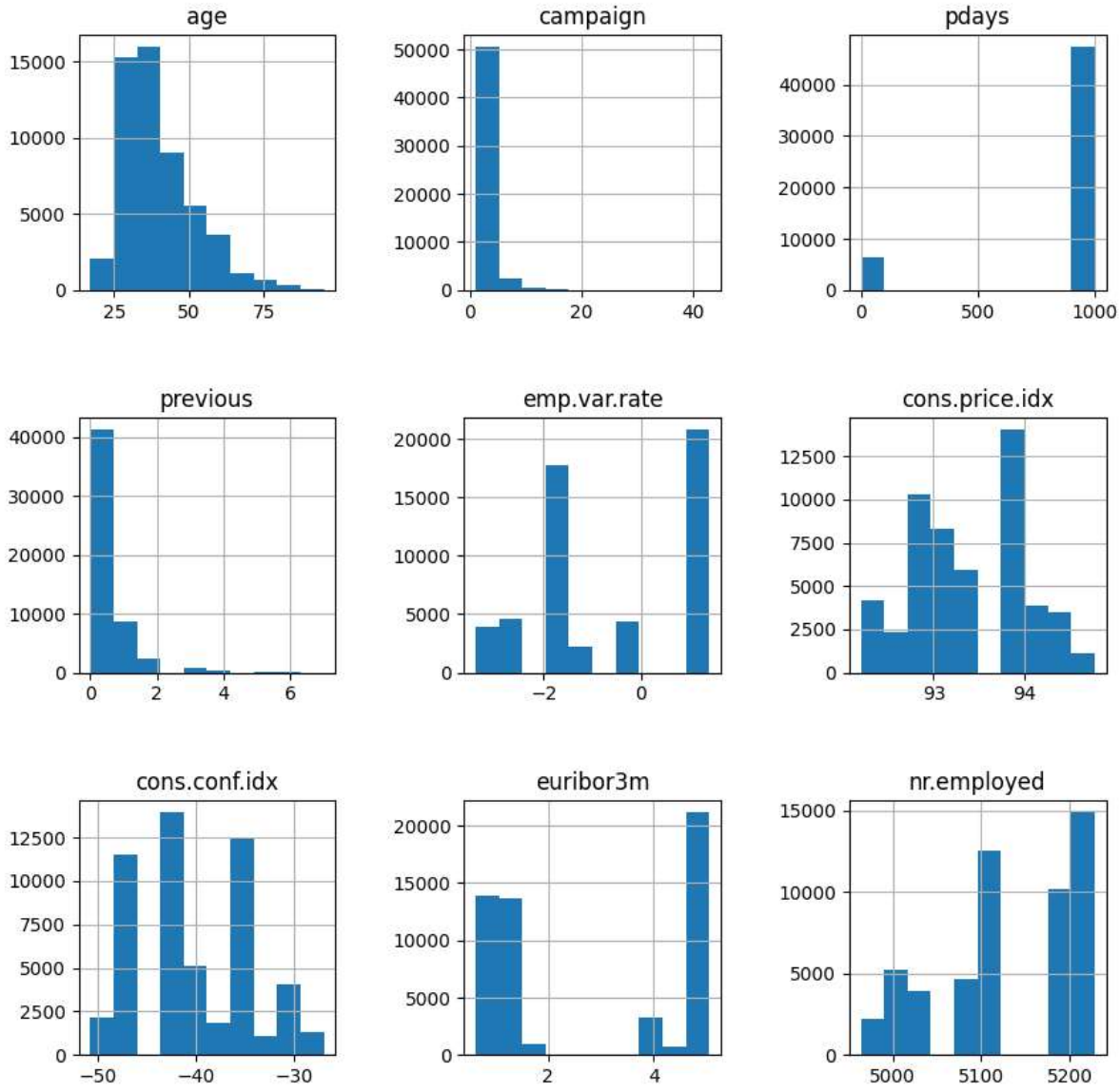
There is odd distribution in yes and no values of target variable so we need to correct the data by inducing rows into data set

In [18]:
```
y_with_yes_values = mdf[mdf['y']=='yes']
while len(mdf[mdf['y']=='yes'])/len(mdf[mdf['y']=='no'])< 0.90:
    mdf=pd.concat([mdf, y_with_yes_values])
sns.barplot(x=mdf['y'].value_counts().index,y=mdf['y'].value_counts())
mdf.shape
```

Out[18]:
```
(53642, 20)
```



In [19]:
```
import matplotlib.pyplot as plt
cols = ['age','campaign','pdays','previous','emp.var.rate','cons.price.idx','cons.c
mdf.hist(column=cols,figsize=(10,10))
plt.subplots_adjust(wspace = 0.5, hspace = 0.5)
plt.show()
```



We can clearly see the pdays is having only two values with huge difference so

In [20]:
```
def function (row):
    if(row['pdays']==999):
        return 0;
    return 1;
mdf['pdays2']=mdf.apply(lambda row: function(row),axis=1)
#changing the value 999 in pdays column to  value 30
def function1 (row):
    if(row['pdays']==999):
        return 30;
    return row['pdays'];
mdf['pdays']=mdf.apply(lambda row: function1(row),axis=1)

#changing the type of pdays to int
mdf['pdays']=mdf['pdays'].astype(int)
mdf.head()
```

Out[20]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	
6	59	admin.	married	professional.course	no	no	no	telephone	may	

5 rows × 21 columns

In [21]:
```
nominal = ['job','marital','education','contact','month','day_of_week']
mdf = pd.get_dummies(mdf,columns=nominal)
mdf['y']=mdf['y'].map({'yes': 1,'no': 0})
mdf.head()
```

Out[21]:

	age	default	housing	loan	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.i
0	56	no	no	no	1	30	0	nonexistent	1.1	93.9
2	37	no	yes	no	1	30	0	nonexistent	1.1	93.9
3	40	no	no	no	1	30	0	nonexistent	1.1	93.9
4	56	no	no	yes	1	30	0	nonexistent	1.1	93.9
6	59	no	no	no	1	30	0	nonexistent	1.1	93.9

5 rows × 53 columns

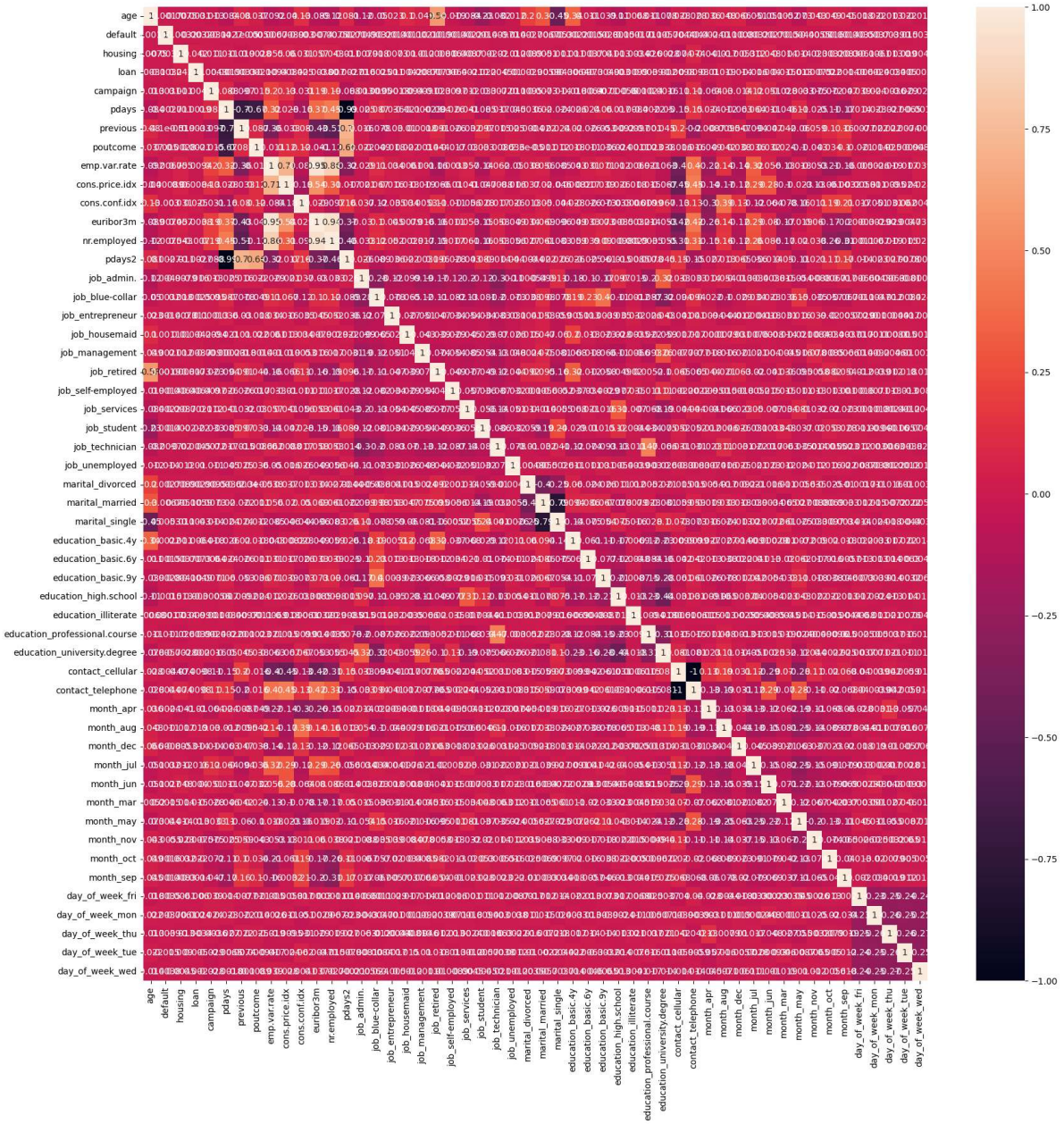
In [22]:
```
mdf.columns
```

Out[22]:
```
Index(['age', 'default', 'housing', 'loan', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed', 'y', 'pdays2', 'job_admin.',
       'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'marital_divorced',
       'marital_married', 'marital_single', 'education_basic.4y',
       'education_basic.6y', 'education_basic.9y', 'education_high.school',
       'education_illiterate', 'education_professional.course',
       'education_university.degree', 'contact_cellular', 'contact_telephone',
       'month_apr', 'month_aug', 'month_dec', 'month_jul', 'month_jun',
       'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',
       'day_of_week_fri', 'day_of_week_mon', 'day_of_week_thu',
       'day_of_week_tue', 'day_of_week_wed'],
      dtype='object')
```

In [24]:
```
mdf['poutcome'] = mdf['poutcome'].map({'failure': -1,'nonexistent': 0,'success': 1}
mdf['default'] = mdf['default'].map({'yes': -1,'no': 1})
mdf['housing'] = mdf['housing'].map({'yes': -1,'no': 1})
mdf['loan'] = mdf['loan'].map({'yes': -1,'no': 1})
```

In [25]:
```
df_corr = mdf.drop(['y'],axis=1).corr()
```

In [26]:
```
plt.figure(figsize=(20, 20))
sns.heatmap(df_corr, annot = True)
plt.show()
```



3.3 Normalize the data

In [27]:
```
cols = ['age','campaign', 'pdays', 'previous','emp.var.rate', 'cons.price.idx', 'co
labels = mdf.y
scaler = MinMaxScaler()
for col in cols:
    mdf[cols] = scaler.fit_transform(mdf[cols])
```

In [28]:
```
mdf.head()
```

Out[28]:

	age	default	housing	loan	campaign	pdays	previous	poutcome	emp.var.rate	cons.pr
0	0.500000	1	1	1	0.0	1.0	0.0	0	0.9375	0.(
2	0.256410	1	-1	1	0.0	1.0	0.0	0	0.9375	0.(
3	0.294872	1	1	1	0.0	1.0	0.0	0	0.9375	0.(
4	0.500000	1	1	-1	0.0	1.0	0.0	0	0.9375	0.(
6	0.538462	1	1	1	0.0	1.0	0.0	0	0.9375	0.(

5 rows × 53 columns

3.4 Feature Engineering

if any

In [29]:
```
#---
```

3.5 Identify the target variables.

Separate the data front the target such that the dataset is in the form of (X,y) or (Features, Label)

Discretize / Encode the target variable or perform one-hot encoding on the target or any other as and if required.

In [30]:
```
y=mdf.y
x_scaled = mdf.drop(['y'],axis=1)
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y,test_size=0.3)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(37549, 52)
(16093, 52)
(37549,)
(16093,)
```

3.6 Split the data into training set and testing set

In [31]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
feature_train, feature_val, label_train, label_val = train_test_split(x_train, y_tr
```

3.7 Report

Mention the method adopted and justify why the method was used

to remove duplicate data, if present

to impute or remove missing data, if present
to remove data inconsistencies, if present
to encode categorical data
the normalization technique used

If the any of the above are not present, then also add in the report below.

Report the size of the training dataset and testing dataset

Score: 3 Marks

---------Type the answer below this line----------##

There are Ordinal Variables ('poutcome', 'default', 'housing' and 'loan' are ordinal ordinal variables)

There are Nominal Variables(One Hot Encoding) 'job', 'maritial', 'education', 'contact', 'month', 'day_of_week' are Nominal Variables

Since there is 74% data with all values so removed the remainging 26% which contains one or more null values

Normalize the data 'age','campaign', 'pdays', 'previous','emp.var.rate', 'cons.price.idx', 'cons.conf.idx','euribor3m', 'nr.employed'

4. Deep Neural Network Architecture

4.1 Design the architecture that you will be using to solve the prediction problem identified.

Add dense layers, specifying the number of units in each layer and the activation function used in the layer.

In [33]:
```
##---------Type the code below this line-----------------##
```

In [34]:
```
model = Sequential()

model.add(Dense(52, input_dim=52, activation='relu', name= 'input'))
#model.add(Dense(32, activation='relu', name= 'hidden_1'))
model.add(Dense(64,  activation='relu', name= 'hidden_2'))
model.add(Dense(128, activation='relu', name= 'hidden_3'))
model.add(Dense(256,  activation='relu', name= 'hidden_4'))
model.add(Dense(512, activation='relu', name= 'hidden_5'))
model.add(Dense(1, input_dim=52, activation='sigmoid', name= 'output'))
```

4.2 Report

Report the following and provide justification for the same.

Number of layers
Number of units in each layer
Activation function used in each hidden layer
Activation function used in the output layer
Total number of trainable parameters

Score: 4 Marks

---------Type the answer below this line----------##

5. Training the model

5.1 Configure the training

Configure the model for training, by using appropriate optimizers and regularizations

In [35]:
```
##---------Type the code below this line-----------------##
```

In [36]:
```
opt = Adam(learning_rate= 0.001)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics= ['accuracy'])
```

In [37]:
```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (Dense)               (None, 52)                2756

 hidden_2 (Dense)            (None, 64)                3392

 hidden_3 (Dense)            (None, 128)               8320

 hidden_4 (Dense)            (None, 256)               33024

 hidden_5 (Dense)            (None, 512)               131584

 output (Dense)              (None, 1)                 513

=================================================================
Total params: 179,589
Trainable params: 179,589
Non-trainable params: 0
_________________________________________________________________
None
```

5.2 Train the model

In [38]:
```
##---------Type the code below this line-----------------##
```

In [39]:
```
training_results = model.fit(feature_train, label_train, verbose=1, batch_size=175,
```

```
Epoch 1/200
194/194 [==============================] - 1s 4ms/step - loss: 0.5461 - accuracy:
0.7360 - val_loss: 0.5524 - val_accuracy: 0.7302
Epoch 2/200
194/194 [==============================] - 1s 3ms/step - loss: 0.5198 - accuracy:
0.7575 - val_loss: 0.5334 - val_accuracy: 0.7457
Epoch 3/200
194/194 [==============================] - 1s 3ms/step - loss: 0.5085 - accuracy:
0.7634 - val_loss: 0.5264 - val_accuracy: 0.7454
Epoch 4/200
194/194 [==============================] - 1s 3ms/step - loss: 0.4935 - accuracy:
0.7735 - val_loss: 0.5381 - val_accuracy: 0.7385
Epoch 5/200
194/194 [==============================] - 1s 3ms/step - loss: 0.4769 - accuracy:
0.7816 - val_loss: 0.5137 - val_accuracy: 0.7507
Epoch 6/200
194/194 [==============================] - 1s 3ms/step - loss: 0.4551 - accuracy:
0.7898 - val_loss: 0.5071 - val_accuracy: 0.7643
Epoch 7/200
194/194 [==============================] - 1s 3ms/step - loss: 0.4343 - accuracy:
0.8002 - val_loss: 0.4875 - val_accuracy: 0.7670
Epoch 8/200
194/194 [==============================] - 1s 3ms/step - loss: 0.4142 - accuracy:
0.8106 - val_loss: 0.4798 - val_accuracy: 0.7720
Epoch 9/200
194/194 [==============================] - 1s 4ms/step - loss: 0.3902 - accuracy:
0.8208 - val_loss: 0.4606 - val_accuracy: 0.7859
Epoch 10/200
194/194 [==============================] - 1s 3ms/step - loss: 0.3676 - accuracy:
0.8300 - val_loss: 0.4670 - val_accuracy: 0.7864
Epoch 11/200
194/194 [==============================] - 1s 3ms/step - loss: 0.3558 - accuracy:
0.8352 - val_loss: 0.4388 - val_accuracy: 0.8005
Epoch 12/200
194/194 [==============================] - 1s 3ms/step - loss: 0.3334 - accuracy:
0.8446 - val_loss: 0.4288 - val_accuracy: 0.8037
Epoch 13/200
194/194 [==============================] - 1s 3ms/step - loss: 0.3171 - accuracy:
0.8510 - val_loss: 0.4247 - val_accuracy: 0.8013
Epoch 14/200
194/194 [==============================] - 1s 4ms/step - loss: 0.3042 - accuracy:
0.8582 - val_loss: 0.4294 - val_accuracy: 0.8075
Epoch 15/200
194/194 [==============================] - 1s 3ms/step - loss: 0.2948 - accuracy:
0.8605 - val_loss: 0.4263 - val_accuracy: 0.8130
Epoch 16/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2840 - accuracy:
0.8663 - val_loss: 0.4403 - val_accuracy: 0.8101
Epoch 17/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2699 - accuracy:
0.8743 - val_loss: 0.4295 - val_accuracy: 0.8136
Epoch 18/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2605 - accuracy:
0.8760 - val_loss: 0.4291 - val_accuracy: 0.8133
Epoch 19/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2532 - accuracy:
0.8804 - val_loss: 0.4322 - val_accuracy: 0.8234
Epoch 20/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2506 - accuracy:
0.8835 - val_loss: 0.4054 - val_accuracy: 0.8218
Epoch 21/200
```

```
194/194 [==============================] - 1s 4ms/step - loss: 0.2422 - accuracy:
0.8858 - val_loss: 0.4250 - val_accuracy: 0.8322
Epoch 22/200
194/194 [==============================] - 1s 3ms/step - loss: 0.2351 - accuracy:
0.8910 - val_loss: 0.3966 - val_accuracy: 0.8360
Epoch 23/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2304 - accuracy:
0.8928 - val_loss: 0.4297 - val_accuracy: 0.8202
Epoch 24/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2224 - accuracy:
0.8955 - val_loss: 0.4032 - val_accuracy: 0.8296
Epoch 25/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2223 - accuracy:
0.8955 - val_loss: 0.4035 - val_accuracy: 0.8445
Epoch 26/200
194/194 [==============================] - 1s 5ms/step - loss: 0.2156 - accuracy:
0.8997 - val_loss: 0.4278 - val_accuracy: 0.8399
Epoch 27/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2092 - accuracy:
0.9026 - val_loss: 0.4025 - val_accuracy: 0.8381
Epoch 28/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2037 - accuracy:
0.9041 - val_loss: 0.4218 - val_accuracy: 0.8344
Epoch 29/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2004 - accuracy:
0.9057 - val_loss: 0.4025 - val_accuracy: 0.8349
Epoch 30/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1958 - accuracy:
0.9086 - val_loss: 0.4077 - val_accuracy: 0.8423
Epoch 31/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1961 - accuracy:
0.9090 - val_loss: 0.4162 - val_accuracy: 0.8442
Epoch 32/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1903 - accuracy:
0.9110 - val_loss: 0.4401 - val_accuracy: 0.8501
Epoch 33/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2053 - accuracy:
0.9056 - val_loss: 0.4064 - val_accuracy: 0.8458
Epoch 34/200
194/194 [==============================] - 1s 4ms/step - loss: 0.2006 - accuracy:
0.9070 - val_loss: 0.4352 - val_accuracy: 0.8477
Epoch 35/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1832 - accuracy:
0.9145 - val_loss: 0.4457 - val_accuracy: 0.8493
Epoch 36/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1783 - accuracy:
0.9157 - val_loss: 0.4337 - val_accuracy: 0.8511
Epoch 37/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1772 - accuracy:
0.9174 - val_loss: 0.4411 - val_accuracy: 0.8581
Epoch 38/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1747 - accuracy:
0.9196 - val_loss: 0.4152 - val_accuracy: 0.8557
Epoch 39/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1715 - accuracy:
0.9216 - val_loss: 0.4406 - val_accuracy: 0.8458
Epoch 40/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1682 - accuracy:
0.9238 - val_loss: 0.4300 - val_accuracy: 0.8583
Epoch 41/200
194/194 [==============================] - 1s 5ms/step - loss: 0.1663 - accuracy:
```

```
0.9223 - val_loss: 0.4511 - val_accuracy: 0.8570
Epoch 42/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1677 - accuracy:
0.9224 - val_loss: 0.4735 - val_accuracy: 0.8437
Epoch 43/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1733 - accuracy:
0.9210 - val_loss: 0.4349 - val_accuracy: 0.8554
Epoch 44/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1607 - accuracy:
0.9269 - val_loss: 0.4313 - val_accuracy: 0.8607
Epoch 45/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1588 - accuracy:
0.9277 - val_loss: 0.4118 - val_accuracy: 0.8663
Epoch 46/200
194/194 [==============================] - 1s 5ms/step - loss: 0.1578 - accuracy:
0.9272 - val_loss: 0.4113 - val_accuracy: 0.8668
Epoch 47/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1576 - accuracy:
0.9281 - val_loss: 0.4276 - val_accuracy: 0.8634
Epoch 48/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1597 - accuracy:
0.9264 - val_loss: 0.4396 - val_accuracy: 0.8586
Epoch 49/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1815 - accuracy:
0.9184 - val_loss: 0.4573 - val_accuracy: 0.8618
Epoch 50/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1446 - accuracy:
0.9339 - val_loss: 0.4973 - val_accuracy: 0.8631
Epoch 51/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1427 - accuracy:
0.9344 - val_loss: 0.4733 - val_accuracy: 0.8716
Epoch 52/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1369 - accuracy:
0.9396 - val_loss: 0.4871 - val_accuracy: 0.8698
Epoch 53/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1343 - accuracy:
0.9393 - val_loss: 0.4926 - val_accuracy: 0.8692
Epoch 54/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1384 - accuracy:
0.9385 - val_loss: 0.4957 - val_accuracy: 0.8621
Epoch 55/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1424 - accuracy:
0.9352 - val_loss: 0.4997 - val_accuracy: 0.8722
Epoch 56/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1547 - accuracy:
0.9319 - val_loss: 0.4468 - val_accuracy: 0.8684
Epoch 57/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1340 - accuracy:
0.9413 - val_loss: 0.4945 - val_accuracy: 0.8695
Epoch 58/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1348 - accuracy:
0.9401 - val_loss: 0.5045 - val_accuracy: 0.8666
Epoch 59/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1358 - accuracy:
0.9385 - val_loss: 0.4935 - val_accuracy: 0.8658
Epoch 60/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1295 - accuracy:
0.9429 - val_loss: 0.4987 - val_accuracy: 0.8663
Epoch 61/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1421 - accuracy:
0.9382 - val_loss: 0.4589 - val_accuracy: 0.8692
```

```
Epoch 62/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1321 - accuracy:
0.9414 - val_loss: 0.4947 - val_accuracy: 0.8732
Epoch 63/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1334 - accuracy:
0.9406 - val_loss: 0.4730 - val_accuracy: 0.8780
Epoch 64/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1275 - accuracy:
0.9446 - val_loss: 0.4639 - val_accuracy: 0.8855
Epoch 65/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1215 - accuracy:
0.9469 - val_loss: 0.4789 - val_accuracy: 0.8759
Epoch 66/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1224 - accuracy:
0.9457 - val_loss: 0.4769 - val_accuracy: 0.8754
Epoch 67/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1159 - accuracy:
0.9487 - val_loss: 0.4940 - val_accuracy: 0.8852
Epoch 68/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1144 - accuracy:
0.9497 - val_loss: 0.5357 - val_accuracy: 0.8788
Epoch 69/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1168 - accuracy:
0.9485 - val_loss: 0.5188 - val_accuracy: 0.8860
Epoch 70/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1173 - accuracy:
0.9484 - val_loss: 0.4571 - val_accuracy: 0.8852
Epoch 71/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1313 - accuracy:
0.9443 - val_loss: 0.4899 - val_accuracy: 0.8788
Epoch 72/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1099 - accuracy:
0.9522 - val_loss: 0.5341 - val_accuracy: 0.8804
Epoch 73/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1122 - accuracy:
0.9493 - val_loss: 0.5641 - val_accuracy: 0.8815
Epoch 74/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1271 - accuracy:
0.9451 - val_loss: 0.5467 - val_accuracy: 0.8735
Epoch 75/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1139 - accuracy:
0.9501 - val_loss: 0.4942 - val_accuracy: 0.8892
Epoch 76/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1029 - accuracy:
0.9540 - val_loss: 0.5287 - val_accuracy: 0.8927
Epoch 77/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1004 - accuracy:
0.9566 - val_loss: 0.5256 - val_accuracy: 0.8908
Epoch 78/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1274 - accuracy:
0.9464 - val_loss: 0.4312 - val_accuracy: 0.8908
Epoch 79/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1077 - accuracy:
0.9527 - val_loss: 0.5521 - val_accuracy: 0.8874
Epoch 80/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1051 - accuracy:
0.9551 - val_loss: 0.4948 - val_accuracy: 0.8905
Epoch 81/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1079 - accuracy:
0.9536 - val_loss: 0.5321 - val_accuracy: 0.8858
Epoch 82/200
```

```
194/194 [==============================] - 1s 3ms/step - loss: 0.1047 - accuracy:
0.9551 - val_loss: 0.4799 - val_accuracy: 0.8903
Epoch 83/200
194/194 [==============================] - 1s 3ms/step - loss: 0.0991 - accuracy:
0.9578 - val_loss: 0.5423 - val_accuracy: 0.8850
Epoch 84/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1125 - accuracy:
0.9519 - val_loss: 0.5318 - val_accuracy: 0.8812
Epoch 85/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1226 - accuracy:
0.9479 - val_loss: 0.4999 - val_accuracy: 0.8823
Epoch 86/200
194/194 [==============================] - 1s 3ms/step - loss: 0.1190 - accuracy:
0.9501 - val_loss: 0.4954 - val_accuracy: 0.8913
Epoch 87/200
194/194 [==============================] - 1s 3ms/step - loss: 0.0922 - accuracy:
0.9605 - val_loss: 0.4968 - val_accuracy: 0.8948
Epoch 88/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0970 - accuracy:
0.9589 - val_loss: 0.5166 - val_accuracy: 0.8916
Epoch 89/200
194/194 [==============================] - 1s 3ms/step - loss: 0.0964 - accuracy:
0.9593 - val_loss: 0.4754 - val_accuracy: 0.8929
Epoch 90/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0884 - accuracy:
0.9622 - val_loss: 0.5242 - val_accuracy: 0.8972
Epoch 91/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0902 - accuracy:
0.9606 - val_loss: 0.5024 - val_accuracy: 0.8972
Epoch 92/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0952 - accuracy:
0.9597 - val_loss: 0.5181 - val_accuracy: 0.8889
Epoch 93/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1048 - accuracy:
0.9559 - val_loss: 0.5131 - val_accuracy: 0.8858
Epoch 94/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1014 - accuracy:
0.9555 - val_loss: 0.5065 - val_accuracy: 0.8924
Epoch 95/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0875 - accuracy:
0.9631 - val_loss: 0.4663 - val_accuracy: 0.9007
Epoch 96/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0861 - accuracy:
0.9632 - val_loss: 0.5461 - val_accuracy: 0.8945
Epoch 97/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0937 - accuracy:
0.9607 - val_loss: 0.4851 - val_accuracy: 0.8977
Epoch 98/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0889 - accuracy:
0.9627 - val_loss: 0.5349 - val_accuracy: 0.8948
Epoch 99/200
194/194 [==============================] - 1s 4ms/step - loss: 0.1030 - accuracy:
0.9559 - val_loss: 0.5082 - val_accuracy: 0.8889
Epoch 100/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0883 - accuracy:
0.9623 - val_loss: 0.5040 - val_accuracy: 0.8983
Epoch 101/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0855 - accuracy:
0.9630 - val_loss: 0.5707 - val_accuracy: 0.8975
Epoch 102/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0860 - accuracy:
```

```
0.9645 - val_loss: 0.5618 - val_accuracy: 0.9001
Epoch 103/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0860 - accuracy:
0.9645 - val_loss: 0.5772 - val_accuracy: 0.8919
Epoch 104/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0943 - accuracy:
0.9612 - val_loss: 0.5210 - val_accuracy: 0.8967
Epoch 105/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0791 - accuracy:
0.9672 - val_loss: 0.5960 - val_accuracy: 0.8945
Epoch 106/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0934 - accuracy:
0.9617 - val_loss: 0.5309 - val_accuracy: 0.8820
Epoch 107/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0895 - accuracy:
0.9632 - val_loss: 0.5549 - val_accuracy: 0.8951
Epoch 108/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0871 - accuracy:
0.9644 - val_loss: 0.5313 - val_accuracy: 0.8985
Epoch 109/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0822 - accuracy:
0.9653 - val_loss: 0.5722 - val_accuracy: 0.8919
Epoch 110/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0798 - accuracy:
0.9673 - val_loss: 0.5443 - val_accuracy: 0.8991
Epoch 111/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0771 - accuracy:
0.9682 - val_loss: 0.5387 - val_accuracy: 0.8991
Epoch 112/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0768 - accuracy:
0.9679 - val_loss: 0.5913 - val_accuracy: 0.8831
Epoch 113/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0791 - accuracy:
0.9675 - val_loss: 0.5196 - val_accuracy: 0.8959
Epoch 114/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0813 - accuracy:
0.9666 - val_loss: 0.5737 - val_accuracy: 0.8924
Epoch 115/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0862 - accuracy:
0.9648 - val_loss: 0.6151 - val_accuracy: 0.8874
Epoch 116/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0842 - accuracy:
0.9653 - val_loss: 0.5633 - val_accuracy: 0.8937
Epoch 117/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0708 - accuracy:
0.9701 - val_loss: 0.5323 - val_accuracy: 0.9057
Epoch 118/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0823 - accuracy:
0.9656 - val_loss: 0.5889 - val_accuracy: 0.8956
Epoch 119/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0788 - accuracy:
0.9676 - val_loss: 0.4766 - val_accuracy: 0.9039
Epoch 120/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0851 - accuracy:
0.9651 - val_loss: 0.5263 - val_accuracy: 0.9028
Epoch 121/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0770 - accuracy:
0.9685 - val_loss: 0.5560 - val_accuracy: 0.8953
Epoch 122/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0816 - accuracy:
0.9670 - val_loss: 0.5978 - val_accuracy: 0.8956
```

```
Epoch 123/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0744 - accuracy:
0.9696 - val_loss: 0.5569 - val_accuracy: 0.8924
Epoch 124/200
194/194 [==============================] - 1s 6ms/step - loss: 0.0751 - accuracy:
0.9689 - val_loss: 0.6793 - val_accuracy: 0.8855
Epoch 125/200
194/194 [==============================] - 1s 6ms/step - loss: 0.1148 - accuracy:
0.9534 - val_loss: 0.4934 - val_accuracy: 0.8956
Epoch 126/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0718 - accuracy:
0.9709 - val_loss: 0.5374 - val_accuracy: 0.9004
Epoch 127/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0713 - accuracy:
0.9711 - val_loss: 0.6079 - val_accuracy: 0.8924
Epoch 128/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0688 - accuracy:
0.9717 - val_loss: 0.6133 - val_accuracy: 0.8945
Epoch 129/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0645 - accuracy:
0.9738 - val_loss: 0.6184 - val_accuracy: 0.8972
Epoch 130/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0678 - accuracy:
0.9719 - val_loss: 0.6261 - val_accuracy: 0.8943
Epoch 131/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0693 - accuracy:
0.9717 - val_loss: 0.6300 - val_accuracy: 0.9025
Epoch 132/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0751 - accuracy:
0.9699 - val_loss: 0.6448 - val_accuracy: 0.8999
Epoch 133/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0669 - accuracy:
0.9731 - val_loss: 0.5555 - val_accuracy: 0.9084
Epoch 134/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0773 - accuracy:
0.9692 - val_loss: 0.5621 - val_accuracy: 0.8985
Epoch 135/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0778 - accuracy:
0.9683 - val_loss: 0.6062 - val_accuracy: 0.8876
Epoch 136/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0711 - accuracy:
0.9705 - val_loss: 0.5531 - val_accuracy: 0.8969
Epoch 137/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0727 - accuracy:
0.9704 - val_loss: 0.5184 - val_accuracy: 0.9079
Epoch 138/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0686 - accuracy:
0.9717 - val_loss: 0.6301 - val_accuracy: 0.8975
Epoch 139/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0637 - accuracy:
0.9746 - val_loss: 0.5470 - val_accuracy: 0.9057
Epoch 140/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0594 - accuracy:
0.9758 - val_loss: 0.5902 - val_accuracy: 0.9031
Epoch 141/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0624 - accuracy:
0.9741 - val_loss: 0.6218 - val_accuracy: 0.8969
Epoch 142/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0787 - accuracy:
0.9689 - val_loss: 0.6141 - val_accuracy: 0.8905
Epoch 143/200
```

```
194/194 [==============================] - 1s 4ms/step - loss: 0.0723 - accuracy:
0.9708 - val_loss: 0.5553 - val_accuracy: 0.9087
Epoch 144/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0669 - accuracy:
0.9733 - val_loss: 0.6067 - val_accuracy: 0.9047
Epoch 145/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0619 - accuracy:
0.9755 - val_loss: 0.6165 - val_accuracy: 0.9020
Epoch 146/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0619 - accuracy:
0.9750 - val_loss: 0.6002 - val_accuracy: 0.9049
Epoch 147/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0655 - accuracy:
0.9731 - val_loss: 0.6260 - val_accuracy: 0.9028
Epoch 148/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0766 - accuracy:
0.9688 - val_loss: 0.5922 - val_accuracy: 0.8948
Epoch 149/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0692 - accuracy:
0.9730 - val_loss: 0.6112 - val_accuracy: 0.9009
Epoch 150/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0655 - accuracy:
0.9735 - val_loss: 0.6620 - val_accuracy: 0.8999
Epoch 151/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0705 - accuracy:
0.9725 - val_loss: 0.6152 - val_accuracy: 0.9009
Epoch 152/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0643 - accuracy:
0.9738 - val_loss: 0.6378 - val_accuracy: 0.9060
Epoch 153/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0642 - accuracy:
0.9747 - val_loss: 0.6558 - val_accuracy: 0.8991
Epoch 154/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0724 - accuracy:
0.9713 - val_loss: 0.5414 - val_accuracy: 0.9055
Epoch 155/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0590 - accuracy:
0.9752 - val_loss: 0.6111 - val_accuracy: 0.9113
Epoch 156/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0606 - accuracy:
0.9760 - val_loss: 0.5872 - val_accuracy: 0.9108
Epoch 157/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0595 - accuracy:
0.9761 - val_loss: 0.5943 - val_accuracy: 0.9095
Epoch 158/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0670 - accuracy:
0.9738 - val_loss: 0.6042 - val_accuracy: 0.8975
Epoch 159/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0911 - accuracy:
0.9651 - val_loss: 0.6284 - val_accuracy: 0.8961
Epoch 160/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0634 - accuracy:
0.9747 - val_loss: 0.5702 - val_accuracy: 0.9044
Epoch 161/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0561 - accuracy:
0.9775 - val_loss: 0.5527 - val_accuracy: 0.9172
Epoch 162/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0564 - accuracy:
0.9776 - val_loss: 0.6268 - val_accuracy: 0.8967
Epoch 163/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0607 - accuracy:
```

```
0.9757 - val_loss: 0.6039 - val_accuracy: 0.9081
Epoch 164/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0545 - accuracy:
0.9780 - val_loss: 0.6707 - val_accuracy: 0.9076
Epoch 165/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0587 - accuracy:
0.9761 - val_loss: 0.6823 - val_accuracy: 0.9012
Epoch 166/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0643 - accuracy:
0.9738 - val_loss: 0.5921 - val_accuracy: 0.9023
Epoch 167/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0586 - accuracy:
0.9756 - val_loss: 0.6413 - val_accuracy: 0.8975
Epoch 168/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0681 - accuracy:
0.9729 - val_loss: 0.7022 - val_accuracy: 0.9015
Epoch 169/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0716 - accuracy:
0.9727 - val_loss: 0.6757 - val_accuracy: 0.9039
Epoch 170/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0613 - accuracy:
0.9758 - val_loss: 0.6289 - val_accuracy: 0.9092
Epoch 171/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0540 - accuracy:
0.9781 - val_loss: 0.6837 - val_accuracy: 0.9087
Epoch 172/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0509 - accuracy:
0.9785 - val_loss: 0.6617 - val_accuracy: 0.9084
Epoch 173/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0719 - accuracy:
0.9720 - val_loss: 0.5818 - val_accuracy: 0.8967
Epoch 174/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0706 - accuracy:
0.9719 - val_loss: 0.5938 - val_accuracy: 0.8929
Epoch 175/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0651 - accuracy:
0.9750 - val_loss: 0.5226 - val_accuracy: 0.9158
Epoch 176/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0489 - accuracy:
0.9795 - val_loss: 0.5998 - val_accuracy: 0.9119
Epoch 177/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0547 - accuracy:
0.9775 - val_loss: 0.6044 - val_accuracy: 0.9116
Epoch 178/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0543 - accuracy:
0.9783 - val_loss: 0.6518 - val_accuracy: 0.9081
Epoch 179/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0570 - accuracy:
0.9770 - val_loss: 0.6578 - val_accuracy: 0.9025
Epoch 180/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0646 - accuracy:
0.9745 - val_loss: 0.6318 - val_accuracy: 0.9047
Epoch 181/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0647 - accuracy:
0.9750 - val_loss: 0.6626 - val_accuracy: 0.9052
Epoch 182/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0534 - accuracy:
0.9788 - val_loss: 0.6282 - val_accuracy: 0.9055
Epoch 183/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0573 - accuracy:
0.9772 - val_loss: 0.5811 - val_accuracy: 0.9121
```

```
Epoch 184/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0599 - accuracy:
0.9757 - val_loss: 0.6444 - val_accuracy: 0.9052
Epoch 185/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0515 - accuracy:
0.9788 - val_loss: 0.5911 - val_accuracy: 0.9150
Epoch 186/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0745 - accuracy:
0.9709 - val_loss: 0.5578 - val_accuracy: 0.9076
Epoch 187/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0541 - accuracy:
0.9785 - val_loss: 0.5733 - val_accuracy: 0.9161
Epoch 188/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0533 - accuracy:
0.9787 - val_loss: 0.6012 - val_accuracy: 0.9158
Epoch 189/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0493 - accuracy:
0.9795 - val_loss: 0.5388 - val_accuracy: 0.9174
Epoch 190/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0548 - accuracy:
0.9791 - val_loss: 0.6391 - val_accuracy: 0.9116
Epoch 191/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0528 - accuracy:
0.9784 - val_loss: 0.6826 - val_accuracy: 0.9121
Epoch 192/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0648 - accuracy:
0.9745 - val_loss: 0.6318 - val_accuracy: 0.9009
Epoch 193/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0556 - accuracy:
0.9779 - val_loss: 0.6104 - val_accuracy: 0.9097
Epoch 194/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0607 - accuracy:
0.9758 - val_loss: 0.5654 - val_accuracy: 0.9089
Epoch 195/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0591 - accuracy:
0.9767 - val_loss: 0.6952 - val_accuracy: 0.9111
Epoch 196/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0622 - accuracy:
0.9757 - val_loss: 0.6416 - val_accuracy: 0.9057
Epoch 197/200
194/194 [==============================] - 1s 4ms/step - loss: 0.0529 - accuracy:
0.9794 - val_loss: 0.5858 - val_accuracy: 0.9089
Epoch 198/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0503 - accuracy:
0.9807 - val_loss: 0.5741 - val_accuracy: 0.9089
Epoch 199/200
194/194 [==============================] - 1s 6ms/step - loss: 0.0517 - accuracy:
0.9791 - val_loss: 0.7515 - val_accuracy: 0.9031
Epoch 200/200
194/194 [==============================] - 1s 5ms/step - loss: 0.0522 - accuracy:
0.9792 - val_loss: 0.6687 - val_accuracy: 0.9071
```

Justify your choice of optimizers and regulizations used and the hyperparameters tuned

Score: 4 Marks

---------Type the answers below this line----------##

As the previously summerized we are doing classification, and using Adam optimizer and choosed binary cross entropy as we have binary classificaion

and we have a output of sigmoid since the probability might occur in between 0 to 1 so if we round off the output values we will get the correct accuracy

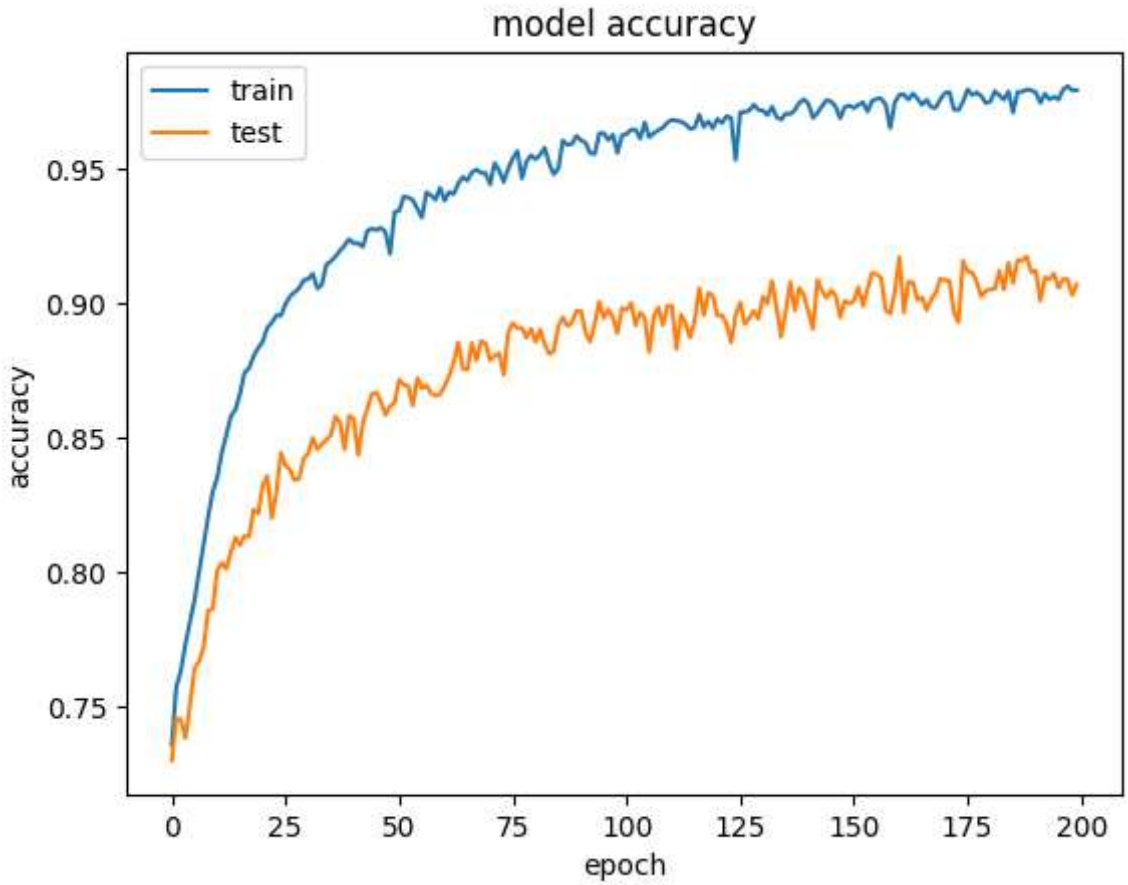
6. Test the model

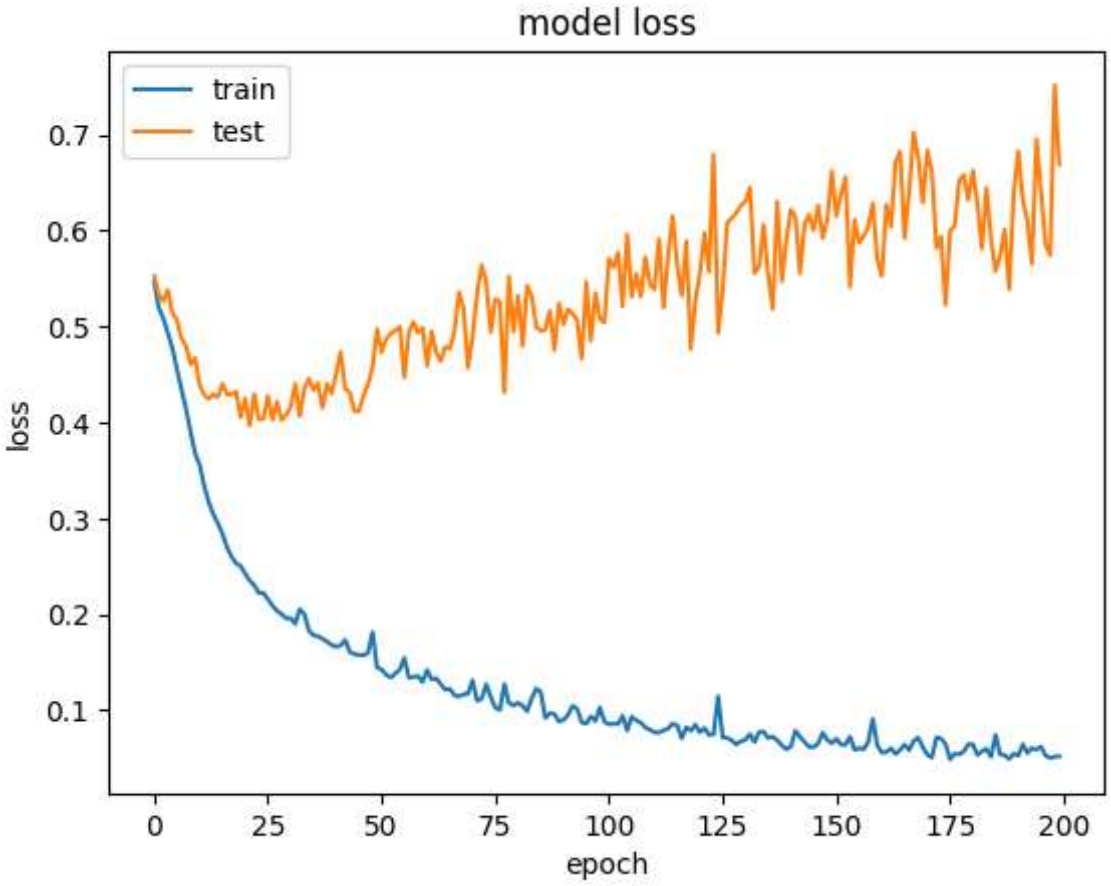
Score: 2 Marks

In [40]:
```
##---------Type the code below this line-----------------##
```

In [41]:
```
# list all data in history
print(training_results.history.keys())
#summarize history for accuracy
#plt.figure(figsize=(20,20))
plt.plot(training_results.history['accuracy'])
plt.plot(training_results.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(training_results.history['loss'])
plt.plot(training_results.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





In [42]:
```
results = model.evaluate(x_test, y_test)
results
```

```
503/503 [==============================] - 1s 1ms/step - loss: 0.5990 - accuracy:
0.9178
```

Out[42]:
```
[0.5990447402000427, 0.9177903532981873]
```

In [43]:
```
y_pred= model.predict(x_test)
```

```
503/503 [==============================] - 1s 1ms/step
```

In [44]:
```
y_pred.round().astype(int).shape
```

Out[44]:
```
(16093, 1)
```

In [45]:
```
print(y_pred.round().astype(int))
y_test
```

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

Out[45]:
```
25005    0
24342    0
28766    1
14297    0
37720    1
        ..
33760    1
22192    1
30207    1
40319    0
8348     0
Name: y, Length: 16093, dtype: int64
```

7. Conclusion

Plot the training and validation loss Report the testing accuracy and loss.

Report values for preformance study metrics like accuracy, precision, recall, F1 Score.

A proper comparision based on different metrics should be done and not just accuracy alone, only then the comparision becomes authentic. You may use Confusion matrix, classification report, MAE etc per the requirement of your application/problem.

Score 2 Marks

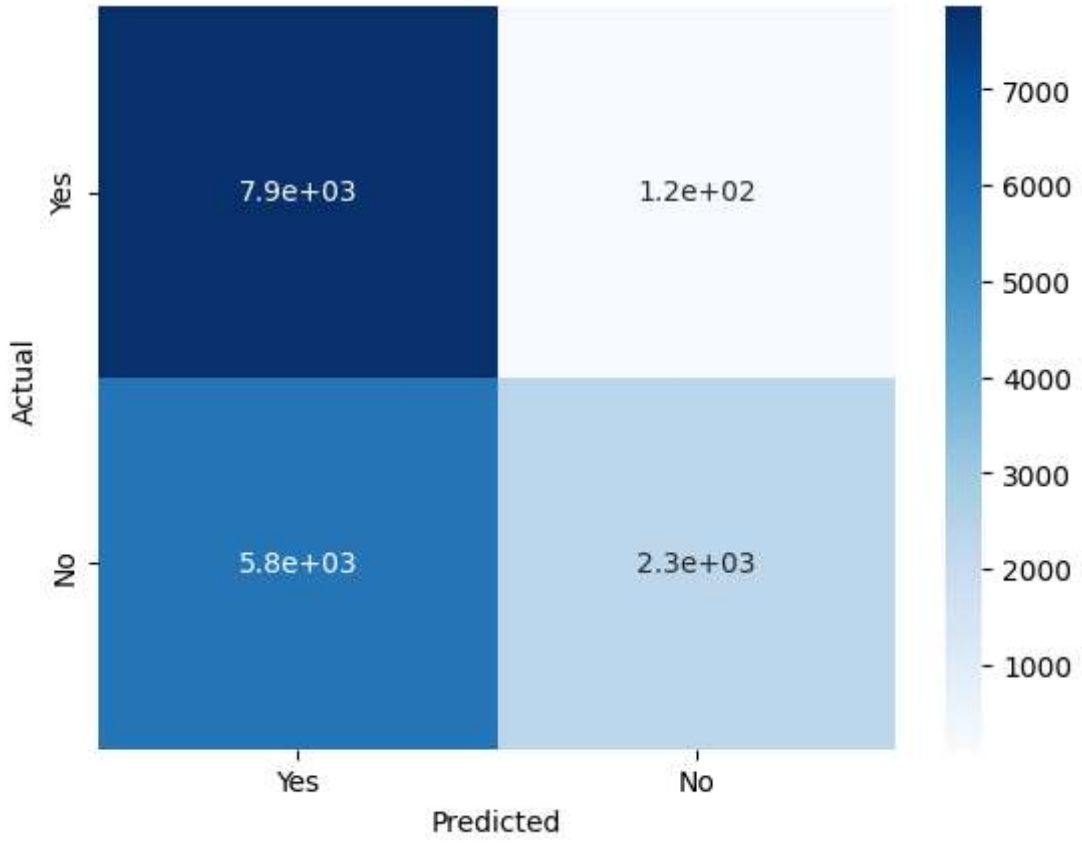
In [46]:
```
##---------Type the code below this line-----------------##
```

In [47]:
```
conf_matrix = tf.math.confusion_matrix(y_test, y_pred)
heatmap_labels = ['Yes','No']
df_cm = pd.DataFrame(conf_matrix.numpy(), columns=heatmap_labels, index = heatmap_
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
#print(dir(conf_matrix))
print(conf_matrix.shape)
print(conf_matrix.numpy())

sns.heatmap(df_cm,cmap='Blues',annot=True)
```

```
(2, 2)
[[7863  122]
 [5794 2314]]
```

Out[47]: <AxesSubplot: xlabel='Predicted', ylabel='Actual'>



In [48]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
print("F1 Score: ", f1_score(y_test, y_pred.round(), average="macro"))
print("Precision Score: ", precision_score(y_test, y_pred.round(), average="macro"))
print("Recall Score: ", recall_score(y_test, y_pred.round(), average="macro"))
```

```
F1 Score:  0.9173815577977342
Precision Score:  0.9248006594068143
Recall Score:  0.9173063069352718
```

8. Solution

What is the solution that is proposed to solve the business problem discussed in Section 1. Also share your learnings while working through solving the problem in terms of challenges, observations, decisions made etc.

Score 2 Marks

##---------Type the answers below this line----------##

The problem is solved using deep learning models, if we carefully observe there is a increase in loss values as well and as well as increase in

accuracy. Which tells that model is not over fitted to particular to confined result set but predictions are very near. and the false positive rates are high, so we can dig deeper to the data set and see which parameter is causing issue and we can reduce the loss and false positve. Model is having a decent accuracy scores and othe merices like F1 scor, Precision, Recall.

coming to the challenges & learning part there are many challenges faced during building the model which made the things to learn :)

1. The data is skewed so correcting the data set is challenging.
2. There are few columns which lead to overfitting such as duration so careful study of the data set is necessary.
3. There are missing values imputation will lead to any undesired circumstances like skewness changing distribution of original data should be carefully observed.
4. The data set is mix of ordinal, nominal values so different techniques are needed to standardize and normalize data
5. Finally choosing the right kind of mix and match of the layers and corresponding optimizers and activation functions

NOTE

All Late Submissions will incur a penalty of -2 marks. So submit your assignments on time.

Good Luck