Chronic Kidney Disease

April 28, 2020

1 Import Libraries

[1]:
```
#Import Libraries
import glob
from keras.models import Sequential, load_model
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import keras as k
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.
```

2 Loading the Dataset:

[2]:
```
train = pd.read_csv('../input/kidney_disease.csv')
train.head()
```

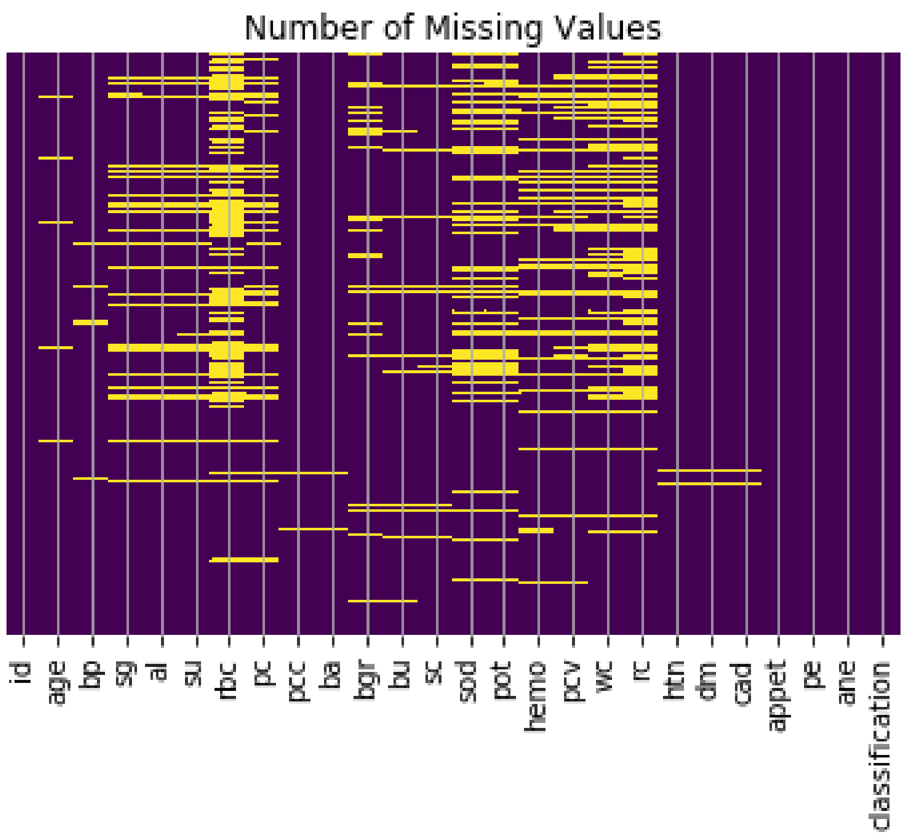
[2]:
```
   id   age    bp     sg   al   su     rbc        pc         pcc          ba  \
0   0  48.0  80.0  1.020  1.0  0.0     NaN    normal  notpresent  notpresent
1   1   7.0  50.0  1.020  4.0  0.0     NaN    normal  notpresent  notpresent
2   2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent  notpresent
3   3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present  notpresent
4   4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent  notpresent

       ...         pcv    wc   rc  htn   dm  cad appet   pe  ane classification
0      ...          44  7800  5.2  yes  yes   no  good   no   no            ckd
1      ...          38  6000  NaN   no   no   no  good   no   no            ckd
2      ...          31  7500  NaN   no  yes   no  poor   no  yes            ckd
3      ...          32  6700  3.9  yes   no   no  poor  yes  yes            ckd
4      ...          35  7300  4.6   no   no   no  good   no   no            ckd
```

```
[5 rows x 26 columns]
```

2.1 Data Preprocessing

2.1.1 Graph Showing Missing Values in Patients Data

[3]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
plt.grid()
plt.title("Number of Missing Values")
plt.savefig('missing.png')
```



2.1.2 Getting rid of ALL ROWS with Nans

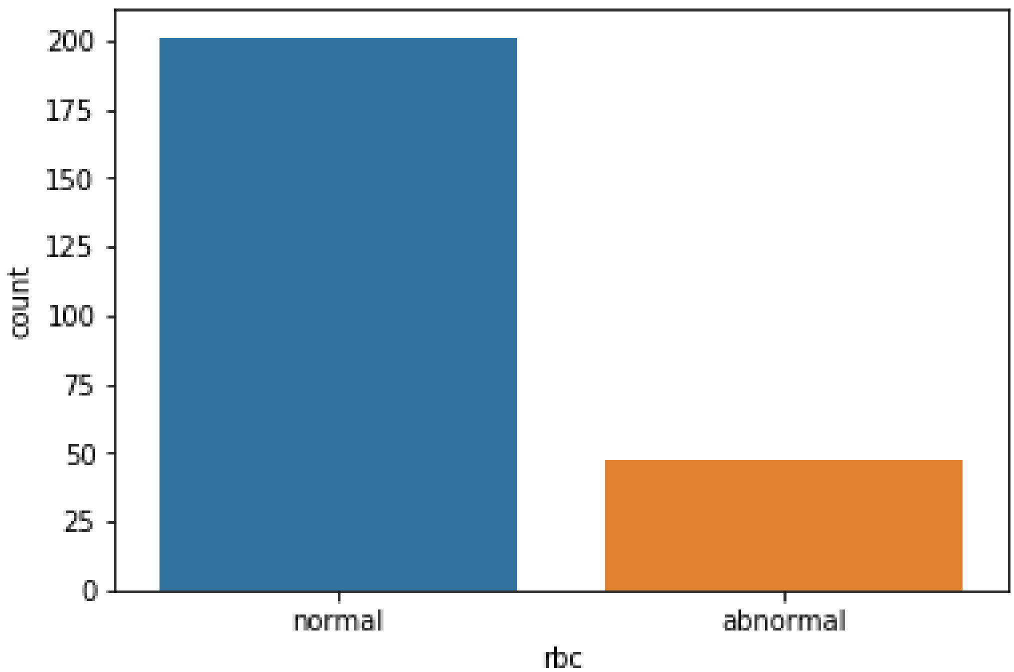
[4]:
```
for i in ['rc','wc','pcv']:
    train[i] = train[i].str.extract('(\d+)').astype(float)
for i in␣
 ↪['age','bp','sg','al','su','bgr','bu','sc','sod','pot','hemo','rc','wc','pcv']:
 ↪
    train[i].fillna(train[i].mean(),inplace=True)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:2: FutureWarning:
currently extract(expand=None) means expand=False (return
Index/Series/DataFrame) but in a future version of pandas this will be changed
to expand=True (return DataFrame)
```

2.2 Data Visualization

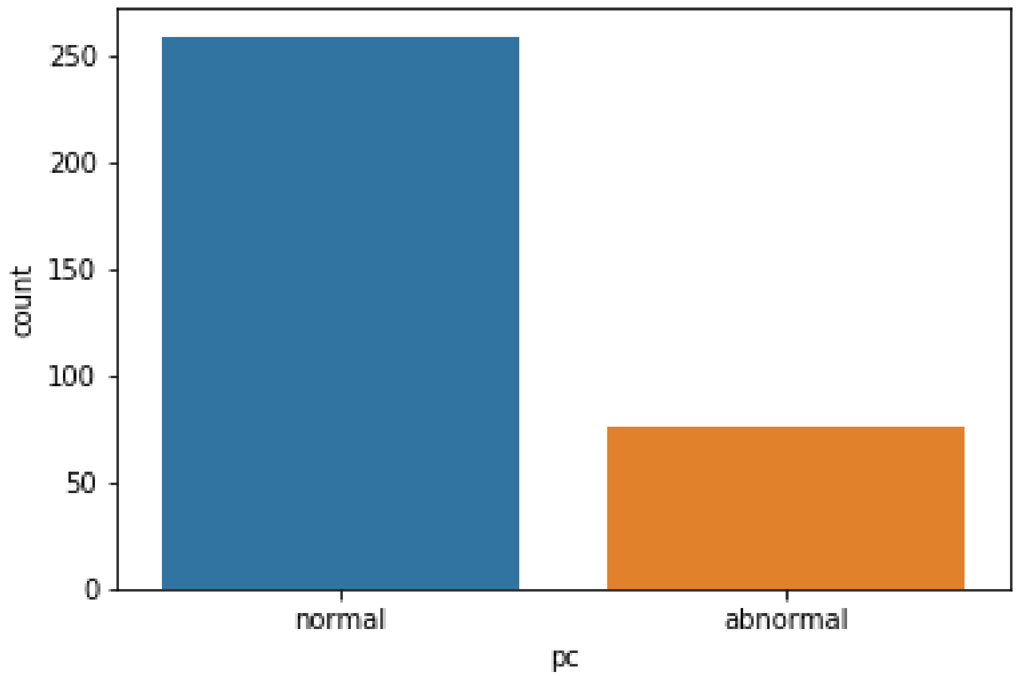
2.2.1 Plot showing RBC count (normal/ abnormal)

[5]:
```
sns.countplot(data=train,x='rbc')
train['rbc'].fillna('normal',inplace=True)
```



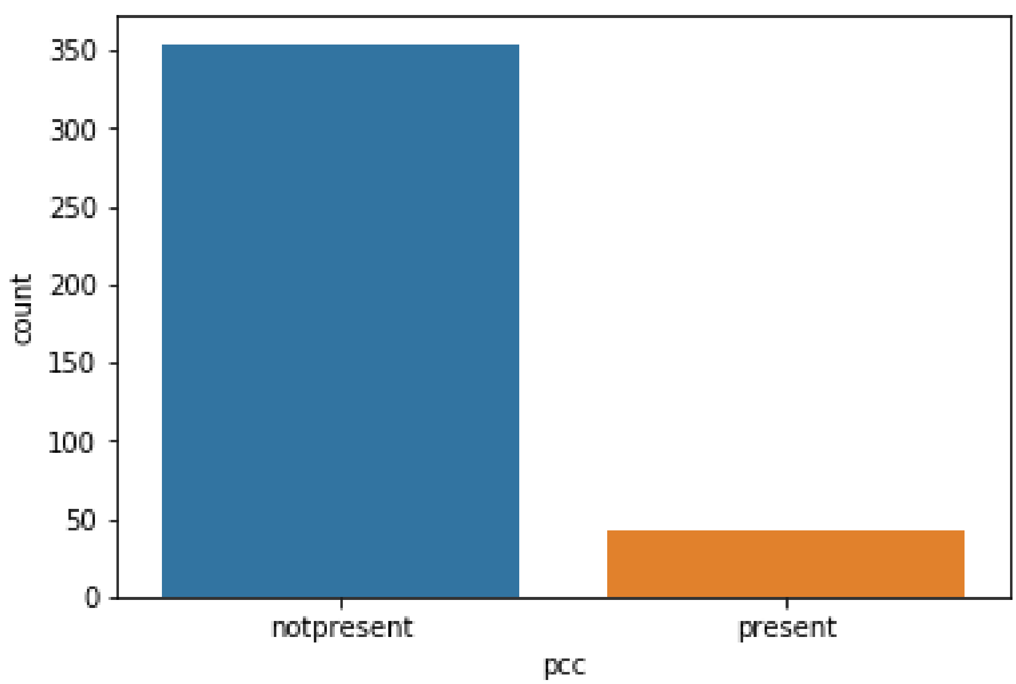
2.2.2 Plot showing Protein C (normal/ abnormal)

[6]:
```
sns.countplot(data=train,x='pc')
train['pc'].fillna('normal',inplace=True)
```



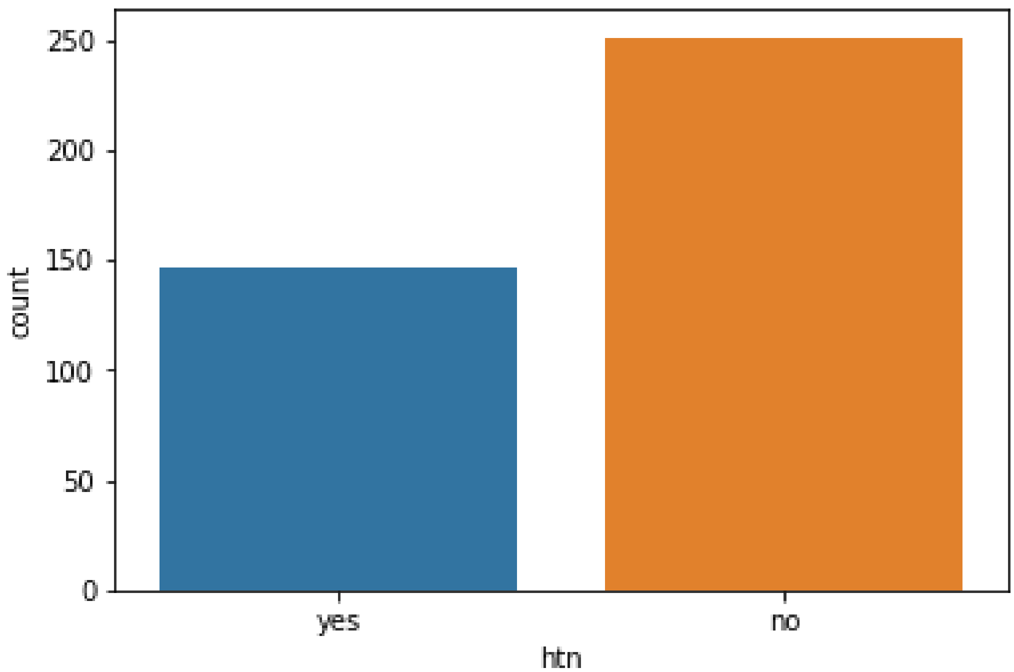
2.2.3 Plot showing Prothrombin complex concentrate (present/ not present)

[7]:
```
sns.countplot(data=train,x='pcc')
train['pcc'].fillna('notpresent',inplace=True)
```



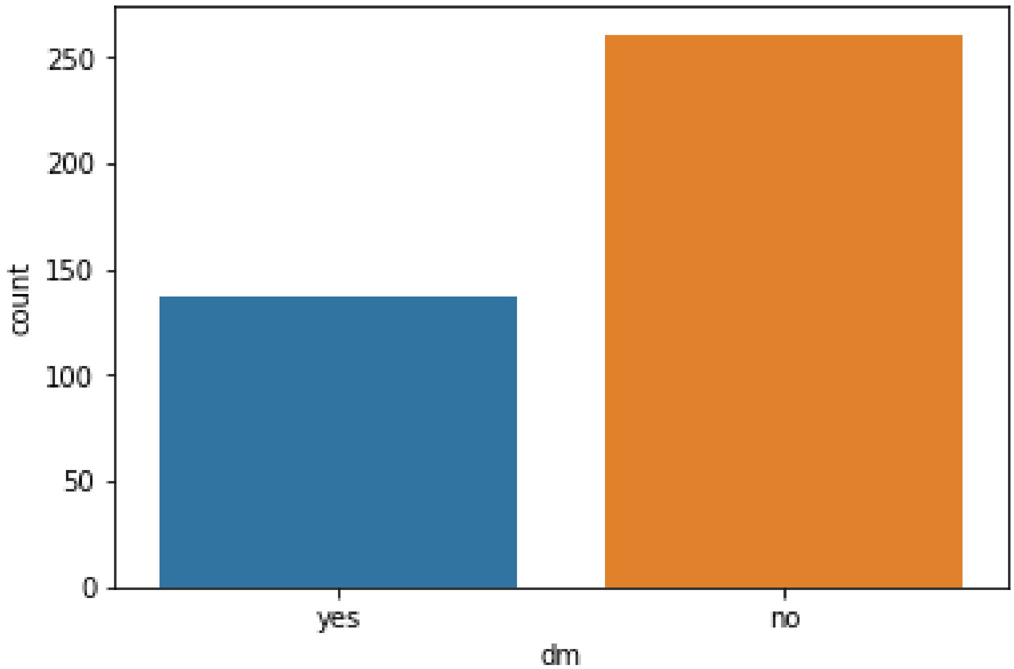
2.2.4 Plot showing count of patients having Hypertension

[8]:
```
sns.countplot(data=train,x='htn')
train['htn'].fillna('no',inplace=True)
```



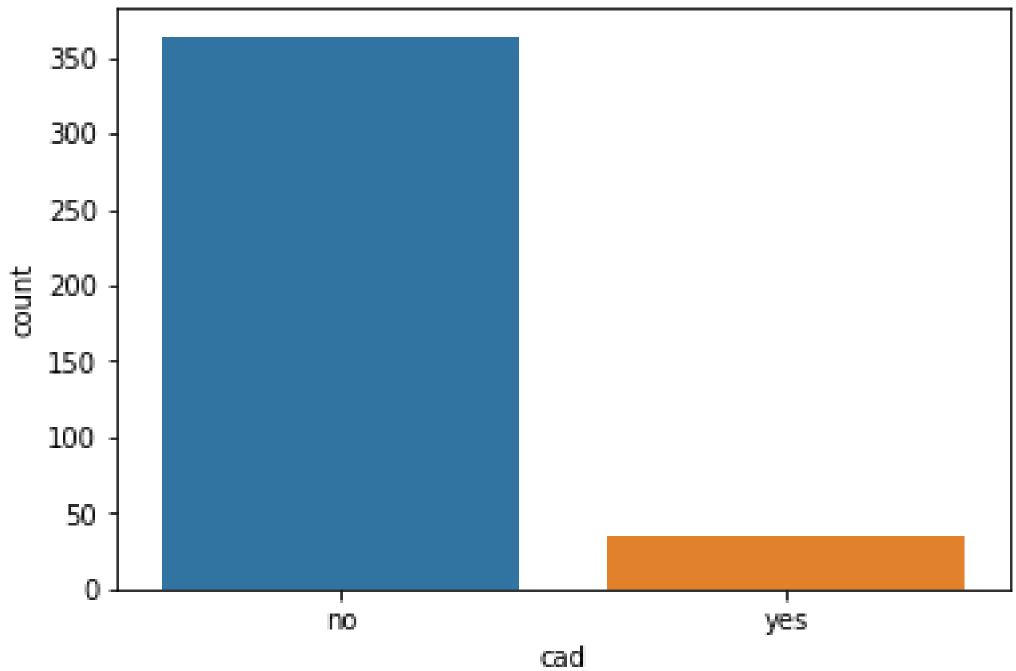
2.2.5 Plot showing count of patients with Diabetes Mellitus

[9]:
```
train['dm'] = train['dm'].replace(to_replace={'\tno':'no','\tyes':'yes',' yes':
 ↪'yes'})
sns.countplot(data=train,x='dm')
train['dm'].fillna('no',inplace=True)
```



2.2.6 Plot Showing count of Coronary Artery Disease

[10]:
```
train['cad'] = train['cad'].replace(to_replace='\tno',value='no')
sns.countplot(data=train,x='cad')
train['cad'].fillna('no',inplace=True)
```



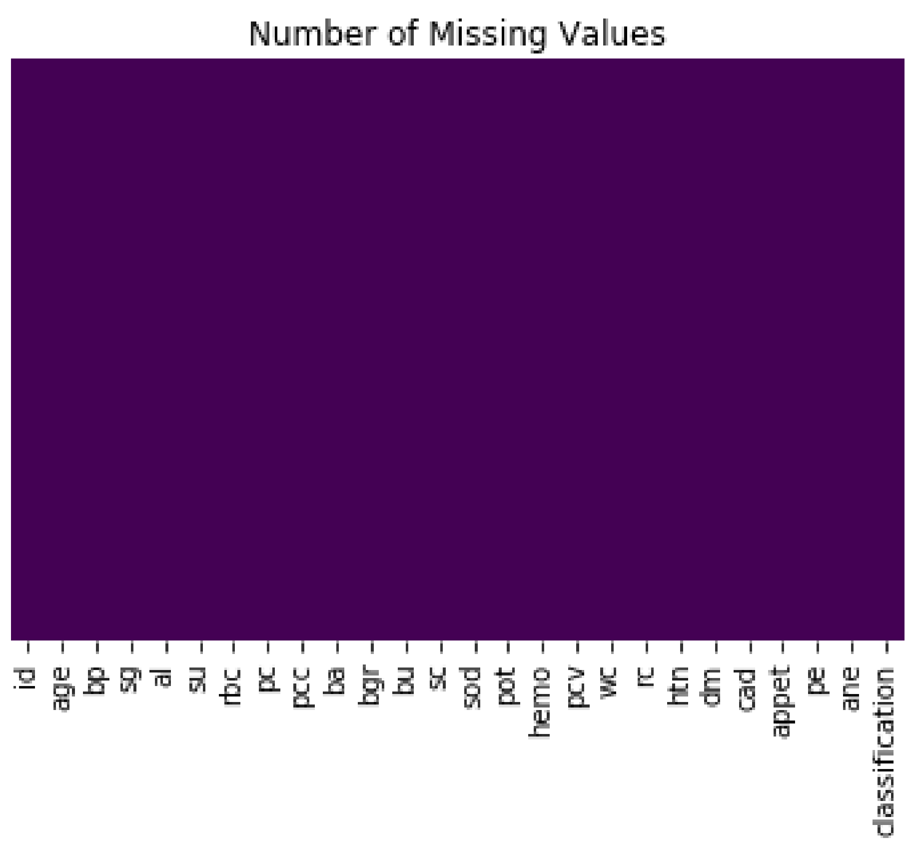
[11]:
```
train['appet'].fillna('good',inplace=True)
train['pe'].fillna('no',inplace=True)
train['ane'].fillna('no',inplace=True)
train['ba'].fillna('notpresent',inplace=True)

train['cad'] = train['cad'].replace(to_replace='ckd\t',value='ckd')
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
id                  400 non-null int64
age                 400 non-null float64
bp                  400 non-null float64
sg                  400 non-null float64
al                  400 non-null float64
su                  400 non-null float64
rbc                 400 non-null object
pc                  400 non-null object
pcc                 400 non-null object
ba                  400 non-null object
bgr                 400 non-null float64
bu                  400 non-null float64
sc                  400 non-null float64
sod                 400 non-null float64
```

```
pot                400 non-null float64
hemo               400 non-null float64
pcv                400 non-null float64
wc                 400 non-null float64
rc                 400 non-null float64
htn                400 non-null object
dm                 400 non-null object
cad                400 non-null object
appet              400 non-null object
pe                 400 non-null object
ane                400 non-null object
classification     400 non-null object
dtypes: float64(14), int64(1), object(11)
memory usage: 81.3+ KB
```

2.2.7 We can see there are no missing values now as we have filled every missing data.

[12]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
plt.title("Number of Missing Values")
plt.savefig('missing_updated.png')
```



[13]:
```
from sklearn.preprocessing import LabelEncoder

for i in␣
↪['rbc','pc','pcc','ba','htn','dm','cad','appet','pe','ane','classification']:
    train[i] = LabelEncoder().fit_transform(train[i])
```

[14]:
```
from sklearn.preprocessing import MinMaxScaler

for i in train.columns:
    train[i] = MinMaxScaler().fit_transform(train[i].astype(float).values.
↪reshape(-1, 1))
```

[15]:
```
X = train.drop(['id','classification'],axis=1)
Y = train['classification']
```

2.2.8 Model

[16]:
```
model = Sequential()

model.add(Dense(100,input_dim=X.shape[1],activation='relu'))
model.add(Dense(50,activation='relu'))
model.add(Dense(25,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[17]:
```
history = model.fit(X,Y,epochs=50,batch_size=40,validation_split=.2,verbose=2)
```

```
Train on 320 samples, validate on 80 samples
Epoch 1/50
 - 0s - loss: 0.5788 - acc: 0.7656 - val_loss: 1.0298 - val_acc: 0.0000e+00
Epoch 2/50
 - 0s - loss: 0.4663 - acc: 0.7750 - val_loss: 1.1574 - val_acc: 0.0000e+00
Epoch 3/50
 - 0s - loss: 0.3951 - acc: 0.7750 - val_loss: 1.0556 - val_acc: 0.0000e+00
Epoch 4/50
 - 0s - loss: 0.3312 - acc: 0.7750 - val_loss: 0.8539 - val_acc: 0.0000e+00
Epoch 5/50
 - 0s - loss: 0.2770 - acc: 0.7750 - val_loss: 0.7313 - val_acc: 0.1625
Epoch 6/50
 - 0s - loss: 0.2395 - acc: 0.8969 - val_loss: 0.6257 - val_acc: 0.9250
Epoch 7/50
 - 0s - loss: 0.2104 - acc: 0.9781 - val_loss: 0.5270 - val_acc: 0.9875
Epoch 8/50
 - 0s - loss: 0.1849 - acc: 0.9844 - val_loss: 0.4554 - val_acc: 0.9875
Epoch 9/50
 - 0s - loss: 0.1625 - acc: 0.9781 - val_loss: 0.3427 - val_acc: 1.0000
Epoch 10/50
```

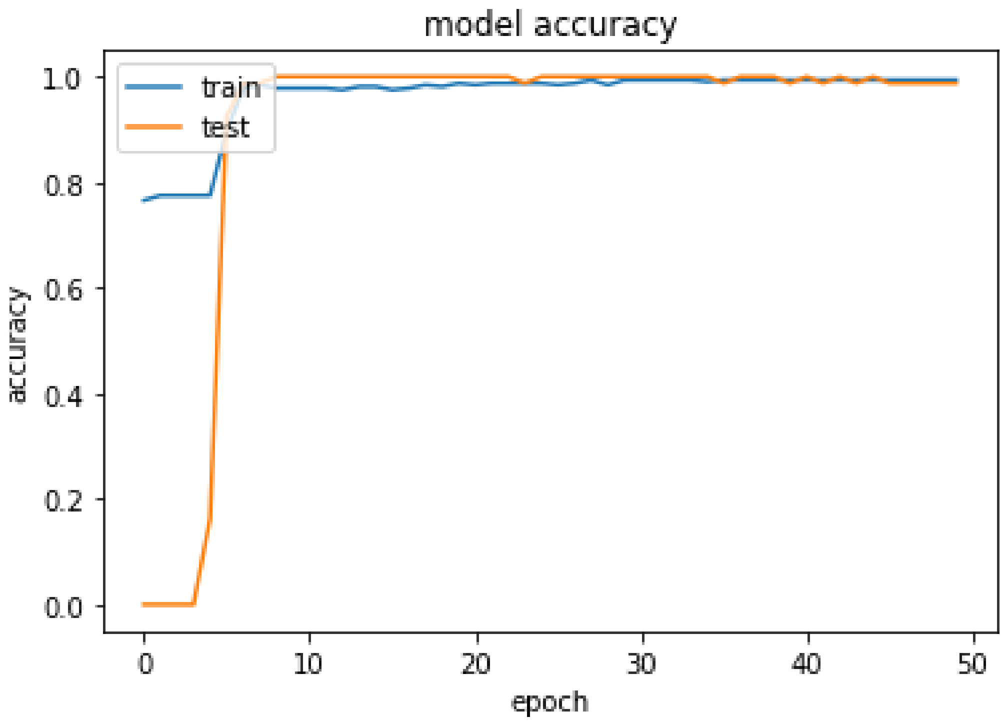
```
 - 0s - loss: 0.1422 - acc: 0.9781 - val_loss: 0.2546 - val_acc: 1.0000
Epoch 11/50
 - 0s - loss: 0.1275 - acc: 0.9781 - val_loss: 0.1824 - val_acc: 1.0000
Epoch 12/50
 - 0s - loss: 0.1140 - acc: 0.9781 - val_loss: 0.1782 - val_acc: 1.0000
Epoch 13/50
 - 0s - loss: 0.1045 - acc: 0.9750 - val_loss: 0.1256 - val_acc: 1.0000
Epoch 14/50
 - 0s - loss: 0.0952 - acc: 0.9813 - val_loss: 0.1452 - val_acc: 1.0000
Epoch 15/50
 - 0s - loss: 0.0916 - acc: 0.9813 - val_loss: 0.0746 - val_acc: 1.0000
Epoch 16/50
 - 0s - loss: 0.0840 - acc: 0.9750 - val_loss: 0.1221 - val_acc: 1.0000
Epoch 17/50
 - 0s - loss: 0.0796 - acc: 0.9781 - val_loss: 0.0705 - val_acc: 1.0000
Epoch 18/50
 - 0s - loss: 0.0703 - acc: 0.9844 - val_loss: 0.1066 - val_acc: 1.0000
Epoch 19/50
 - 0s - loss: 0.0703 - acc: 0.9813 - val_loss: 0.0540 - val_acc: 1.0000
Epoch 20/50
 - 0s - loss: 0.0671 - acc: 0.9875 - val_loss: 0.0698 - val_acc: 1.0000
Epoch 21/50
 - 0s - loss: 0.0583 - acc: 0.9844 - val_loss: 0.0455 - val_acc: 1.0000
Epoch 22/50
 - 0s - loss: 0.0584 - acc: 0.9875 - val_loss: 0.0630 - val_acc: 1.0000
Epoch 23/50
 - 0s - loss: 0.0538 - acc: 0.9875 - val_loss: 0.0390 - val_acc: 1.0000
Epoch 24/50
 - 0s - loss: 0.0483 - acc: 0.9875 - val_loss: 0.0702 - val_acc: 0.9875
Epoch 25/50
 - 0s - loss: 0.0471 - acc: 0.9875 - val_loss: 0.0378 - val_acc: 1.0000
Epoch 26/50
 - 0s - loss: 0.0441 - acc: 0.9844 - val_loss: 0.0377 - val_acc: 1.0000
Epoch 27/50
 - 0s - loss: 0.0414 - acc: 0.9875 - val_loss: 0.0461 - val_acc: 1.0000
Epoch 28/50
 - 0s - loss: 0.0382 - acc: 0.9938 - val_loss: 0.0338 - val_acc: 1.0000
Epoch 29/50
 - 0s - loss: 0.0376 - acc: 0.9844 - val_loss: 0.0391 - val_acc: 1.0000
Epoch 30/50
 - 0s - loss: 0.0337 - acc: 0.9938 - val_loss: 0.0360 - val_acc: 1.0000
Epoch 31/50
 - 0s - loss: 0.0324 - acc: 0.9938 - val_loss: 0.0293 - val_acc: 1.0000
Epoch 32/50
 - 0s - loss: 0.0303 - acc: 0.9938 - val_loss: 0.0249 - val_acc: 1.0000
Epoch 33/50
 - 0s - loss: 0.0286 - acc: 0.9938 - val_loss: 0.0330 - val_acc: 1.0000
Epoch 34/50
```

```
 - 0s - loss: 0.0270 - acc: 0.9938 - val_loss: 0.0287 - val_acc: 1.0000
Epoch 35/50
 - 0s - loss: 0.0260 - acc: 0.9906 - val_loss: 0.0267 - val_acc: 1.0000
Epoch 36/50
 - 0s - loss: 0.0235 - acc: 0.9938 - val_loss: 0.0351 - val_acc: 0.9875
Epoch 37/50
 - 0s - loss: 0.0221 - acc: 0.9937 - val_loss: 0.0256 - val_acc: 1.0000
Epoch 38/50
 - 0s - loss: 0.0218 - acc: 0.9938 - val_loss: 0.0243 - val_acc: 1.0000
Epoch 39/50
 - 0s - loss: 0.0204 - acc: 0.9938 - val_loss: 0.0225 - val_acc: 1.0000
Epoch 40/50
 - 0s - loss: 0.0188 - acc: 0.9938 - val_loss: 0.0271 - val_acc: 0.9875
Epoch 41/50
 - 0s - loss: 0.0160 - acc: 0.9938 - val_loss: 0.0187 - val_acc: 1.0000
Epoch 42/50
 - 0s - loss: 0.0153 - acc: 0.9938 - val_loss: 0.0284 - val_acc: 0.9875
Epoch 43/50
 - 0s - loss: 0.0144 - acc: 0.9938 - val_loss: 0.0173 - val_acc: 1.0000
Epoch 44/50
 - 0s - loss: 0.0130 - acc: 0.9938 - val_loss: 0.0230 - val_acc: 0.9875
Epoch 45/50
 - 0s - loss: 0.0119 - acc: 0.9938 - val_loss: 0.0185 - val_acc: 1.0000
Epoch 46/50
 - 0s - loss: 0.0124 - acc: 0.9938 - val_loss: 0.0234 - val_acc: 0.9875
Epoch 47/50
 - 0s - loss: 0.0113 - acc: 0.9938 - val_loss: 0.0207 - val_acc: 0.9875
Epoch 48/50
 - 0s - loss: 0.0104 - acc: 0.9938 - val_loss: 0.0194 - val_acc: 0.9875
Epoch 49/50
 - 0s - loss: 0.0102 - acc: 0.9938 - val_loss: 0.0253 - val_acc: 0.9875
Epoch 50/50
 - 0s - loss: 0.0097 - acc: 0.9938 - val_loss: 0.0183 - val_acc: 0.9875
```

2.2.9 Plot showing Model Accuracy

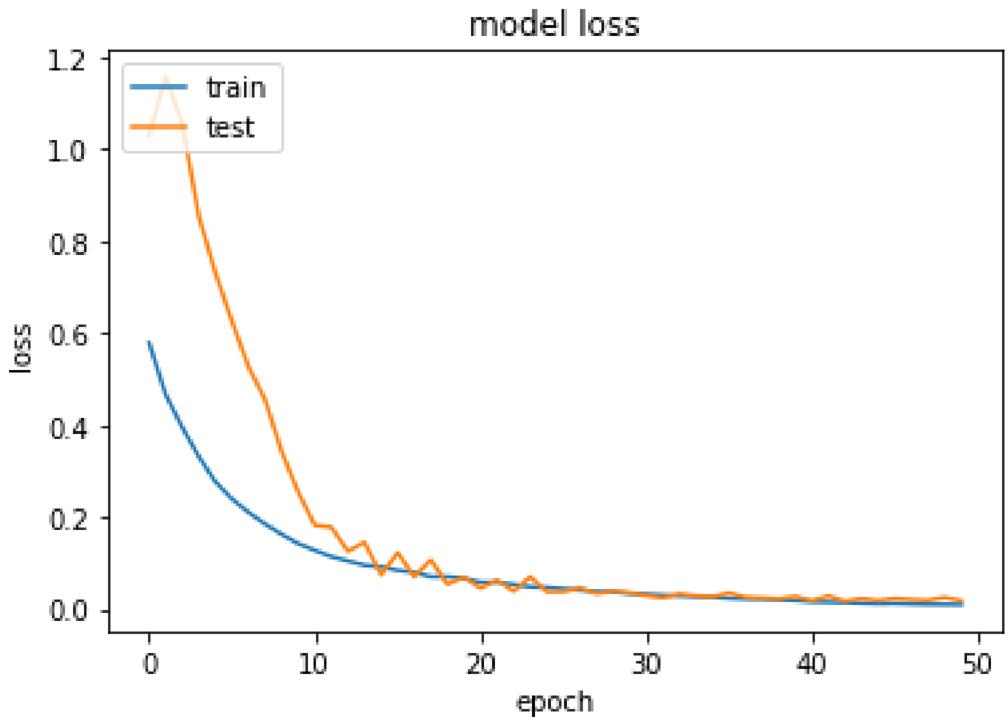
[18]:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



2.2.10 Plot showing Model Loss

[19]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



2.2.11 Getting Accuracy of 99.25% from our model.

[20]:
```
scores = model.evaluate(X,Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
400/400 [==============================] - 0s 34us/step

acc: 99.25%
```