



Heart-Disease-Identiffication-Using-Various-Machine-Learning-Models / Heart Disease classification Machine Learning.ipynb

Gitpod

 **dantecomedia** Rename Heart Disease classification Machine Learning (2).ipynb to Hea...
536e436 4 days ago

1 contributor


<>





Raw

Blame

History







2 lines (1 sloc) 622 KB

Introduction

In this kernel, i have tried to introduce various machine learning models and their implementation. The sole purpose of this kernel is to correctly identify the presence of any heart disease based on the given input. We proceed by giving a general introduction towards the disease followed by dataset's attributes and what they resemble. As we descend down the kernel , i have used various Machine Learning and Deep Learning approach.

Note:- This Kernel is subject to get updated as soon as i find something which can be revelant to the context.
Please Upvote if you like the Kernel

About Heart Disease

Heart disease is a general term that means that the heart is not working normally. Babies can be born with heart disease. This is called congenital heart disease. If people get heart disease later, it is called acquired heart disease. Most heart disease is acquired. The three most common types of acquired heart disease are:

- **Coronary Artery Disease (acronym CAD)**
- **Congestive Heart Failure (CHF)**
- **Bad Heart Rhythms**

•

DEATHS

Heart disease is the biggest killer of both men and women in the United States, England, Wales, and Canada. For example, heart disease causes 4 out of every 10 deaths in the United States. This is more than all kinds of cancer put together. Also, one person dies of heart disease about every minute in the United States alone.

Loading common dependencies

```
In [1]: # This Python 3 environment comes with many helpful analytics libraries instal
Led
# It is defined by the kaggle/python docker image: https://github.com/kaggle/d
ocker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will lis
t the files in the input directory

import os
print(os.listdir("../input"))
import warnings
warnings.filterwarnings("ignore")
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA, TruncatedSVD
import matplotlib.patches as mpatches
import time

# Classifier Libraries
```

```

# Scikit-Learn Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import collections

# Deep Learning Libraries
from keras.models import Sequential
from keras.layers import Dense,Dropout

# Misc. Libraries
from imblearn.datasets import fetch_datasets
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from imblearn.metrics import classification_report_imbalanced
from collections import Counter
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix,f1_score,precision_score,recall_score,accuracy_score
import warnings
warnings.filterwarnings("ignore")

import seaborn as sns
import matplotlib.pyplot as plt

['heart.csv']

Using TensorFlow backend.

```

- **Going through the dataset, first impression**

Loading the dataset

```
In [2]: data=pd.read_csv("../input/heart.csv")
```

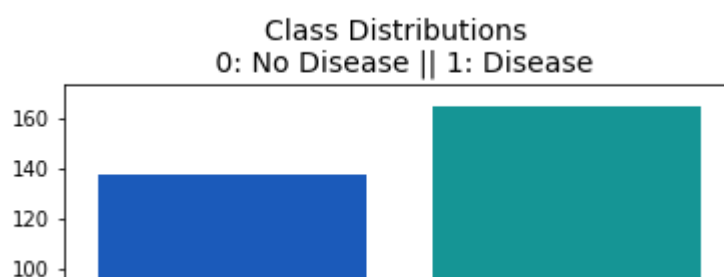
```
In [3]: print('Heart Disease', round(data['target'].value_counts()[1]/len(data) * 100,
2), '% of the target ')
print('No heart Disease', round(data['target'].value_counts()[0]/len(data) * 100,2), '% of the target')
```

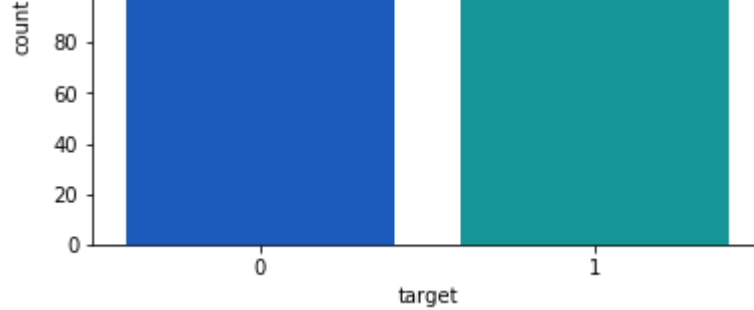
```
Heart Disease 54.46 % of the target
No heart Disease 45.54 % of the target
```

We see that the Heart disease occurred 54.3 % of the times in the dataset, whilst the 45.7% were the no heart disease. So, we need to balance the dataset otherwise it might get overfit. This will help the model to find pattern in the dataset which contributes to the heart disease and which doesn't.

```
In [4]: sns.countplot('target', data=data, palette="winter")
plt.title('Class Distributions \n 0: No Disease || 1: Disease', fontsize=14)
```

```
Out[4]: Text(0.5, 1.0, 'Class Distributions \n 0: No Disease || 1: Disease')
```





In [5]: `data.head()`

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targ
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Attributes and what does they resemble

- **age**: Age of patient
- **sex**: Sex, 1 for male
- **cp**: chest pain
- **trestbps**: resting blood pressure, more than 120 over 80 and less than 140 over 90 (120/80-140/90): You have a normal blood pressure reading but it is a little higher than it should be, and you should try to lower it. Make healthy changes to your lifestyle.
- **chol**: serum cholesterol, shows the amount of triglycerides present. Triglycerides are another lipid that can be measured in the blood.
- **fbs**: fasting blood sugar larger 120mg/dl (1 true), less than 100 mg/dL (5.6 mmol/L) is normal, 100 to 125 mg/dL (5.6 to 6.9 mmol/L) is considered prediabetes
- **rest**: ecg resting electrode.
- **thalach**: maximum heart rate achieved, maximum heart rate is 220 minus your age.
- **exang**: exercise induced angina (1 yes), Angina is a type of chest pain caused by reduced blood flow to the heart. Angina is a symptom of coronary artery disease.
- **oldpeak**: ST depression induced by exercise relative to rest
- **slope**: slope of peak exercise ST
- **ca**: number of major vessel
- **thal**: no explanation provided, but probably thalassemia (3 normal; 6 fixed defect; 7 reversible defect)
- **result**: (1 anomaly) num diagnosis of heart disease (angiographic disease status)

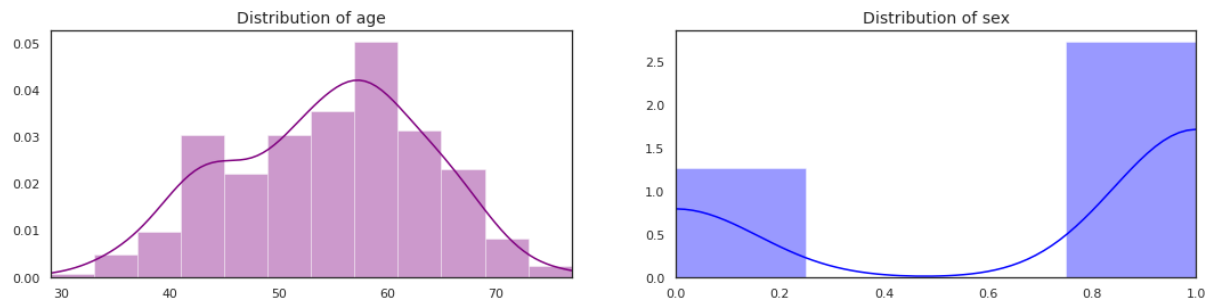
Distribution: Checking how the attribute values are distributed and determining their skewness

In [6]: `sns.set(style="white", palette="PuBuGn_d", color_codes=True)`

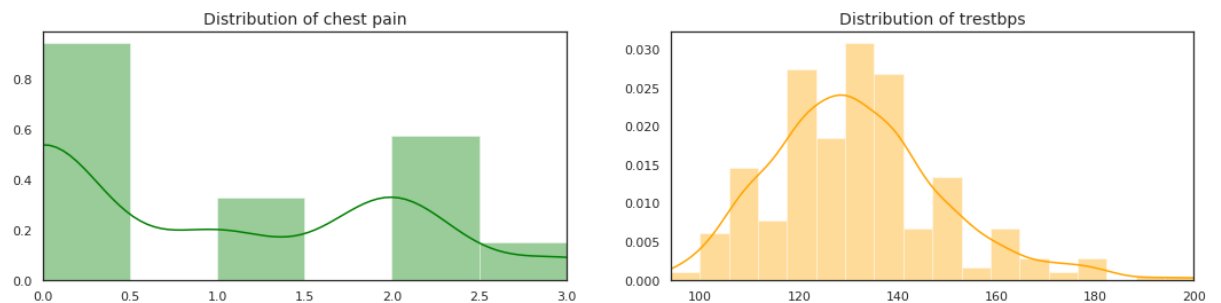
In [7]:

```
fig, ax = plt.subplots(1, 2, figsize=(18,4))
age = data['age'].values
sex = data['sex'].values
sns.distplot(age, ax=ax[0], color='purple')
ax[0].set_title('Distribution of age', fontsize=14)
ax[0].set_xlim([min(age), max(age)])
sns.distplot(sex, ax=ax[1], color='b')
ax[1].set_title('Distribution of sex', fontsize=14)
```

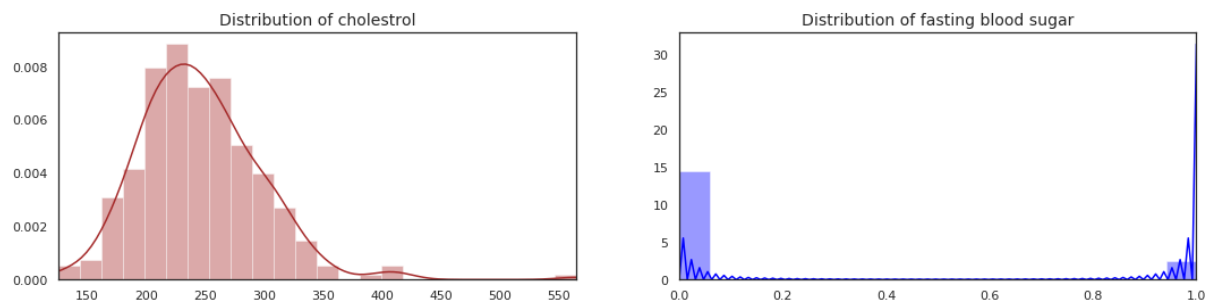
```
ax[1].set_xlim([min(sex), max(sex)])
plt.show()
```



```
In [8]: fig, ax = plt.subplots(1, 2, figsize=(18,4))
cp = data['cp'].values
trestbps = data['trestbps'].values
sns.distplot(cp, ax=ax[0], color='green')
ax[0].set_title('Distribution of chest pain', fontsize=14)
ax[0].set_xlim([min(cp), max(cp)])
sns.distplot(trestbps, ax=ax[1], color='orange')
ax[1].set_title('Distribution of trestbps', fontsize=14)
ax[1].set_xlim([min(trestbps), max(trestbps)])
plt.show()
```

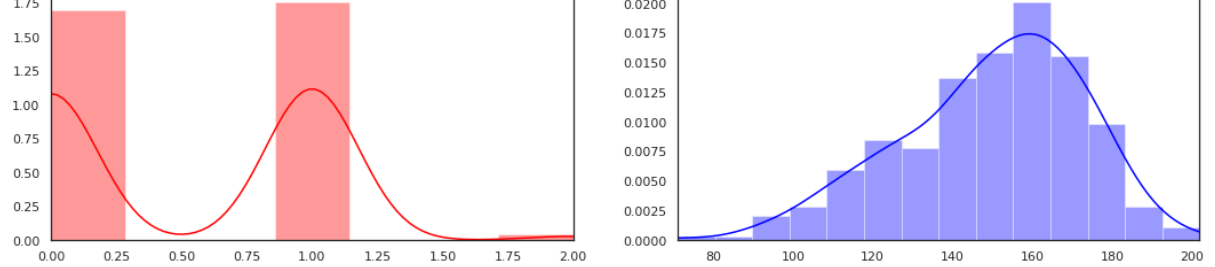


```
In [9]: fig, ax = plt.subplots(1, 2, figsize=(18,4))
chol = data['chol'].values
fbs = data['fbs'].values
sns.distplot(chol, ax=ax[0], color='brown')
ax[0].set_title('Distribution of cholesterol', fontsize=14)
ax[0].set_xlim([min(chol), max(chol)])
sns.distplot(fbs, ax=ax[1], color='blue')
ax[1].set_title('Distribution of fasting blood sugar', fontsize=14)
ax[1].set_xlim([min(fbs), max(fbs)])
plt.show()
```

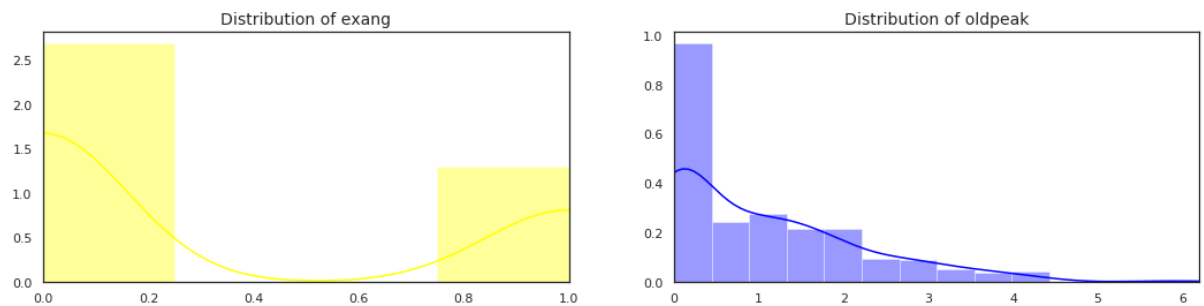


```
In [10]: fig, ax = plt.subplots(1, 2, figsize=(18,4))
restecg = data['restecg'].values
thalach = data['thalach'].values
sns.distplot(restecg, ax=ax[0], color='r')
ax[0].set_title('Distribution of ecg resting electrode', fontsize=14)
ax[0].set_xlim([min(restecg), max(restecg)])
sns.distplot(thalach, ax=ax[1], color='b')
ax[1].set_title('Distribution of thalach', fontsize=14)
ax[1].set_xlim([min(thalach), max(thalach)])
plt.show()
```

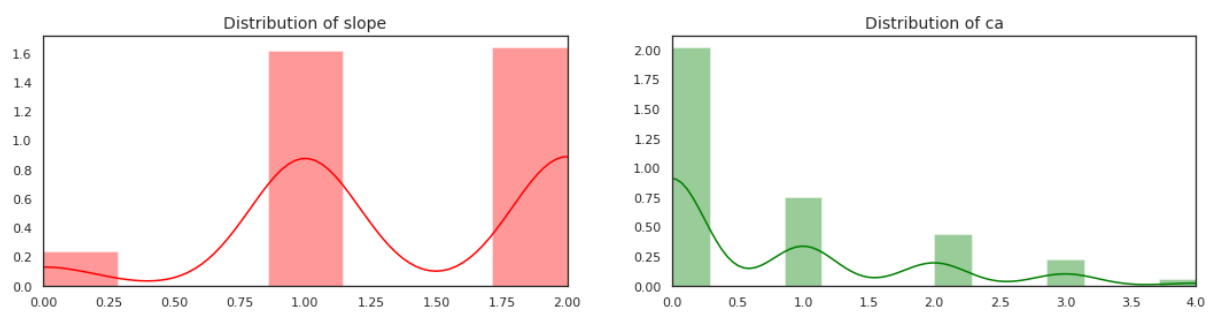




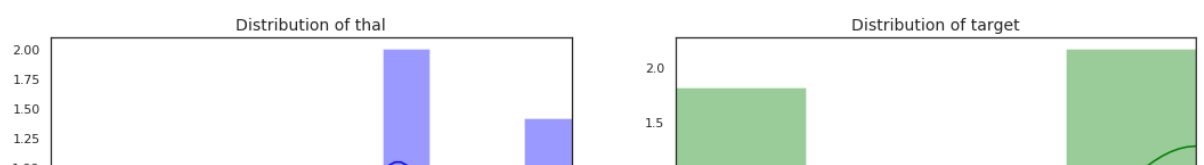
```
In [11]: fig, ax = plt.subplots(1, 2, figsize=(18,4))
exang = data['exang'].values
oldpeak = data['oldpeak'].values
sns.distplot(exang, ax=ax[0], color='yellow')
ax[0].set_title('Distribution of exang', fontsize=14)
ax[0].set_xlim([min(exang), max(exang)])
sns.distplot(oldpeak, ax=ax[1], color='b')
ax[1].set_title('Distribution of oldpeak', fontsize=14)
ax[1].set_xlim([min(oldpeak), max(oldpeak)])
plt.show()
```

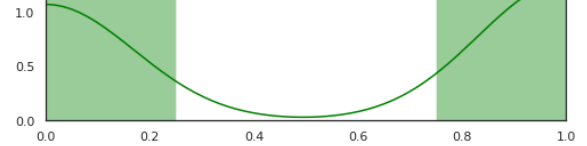
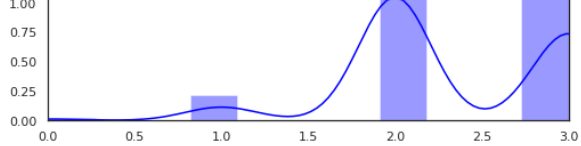


```
In [12]: fig, ax = plt.subplots(1, 2, figsize=(18,4))
slope = data['slope'].values
ca = data['ca'].values
sns.distplot(slope, ax=ax[0], color='red')
ax[0].set_title('Distribution of slope', fontsize=14)
ax[0].set_xlim([min(slope), max(slope)])
sns.distplot(ca, ax=ax[1], color='green')
ax[1].set_title('Distribution of ca', fontsize=14)
ax[1].set_xlim([min(ca), max(ca)])
plt.show()
```



```
In [13]: fig, ax = plt.subplots(1, 2, figsize=(18,4))
thal = data['thal'].values
target = data['target'].values
sns.distplot(thal, ax=ax[0], color='blue')
ax[0].set_title('Distribution of thal', fontsize=14)
ax[0].set_xlim([min(thal), max(thal)])
sns.distplot(target, ax=ax[1], color='green')
ax[1].set_title('Distribution of target', fontsize=14)
ax[1].set_xlim([min(target), max(target)])
plt.show()
```





Getting Insight

In [14]: `data.describe()`

Out[14]:

	age	sex	cp	trestbps	chol	fbs	restecg
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000

Checking for null values

In [15]: `data.isnull().any()`

Out[15]:

```

age          False
sex          False
cp           False
trestbps     False
chol         False
fbs          False
restecg      False
thalach      False
exang        False
oldpeak      False
slope        False
ca           False
thal         False
target       False
dtype: bool

```

No null values

Counting the total target values in each class

In [16]: `data.target.value_counts()`

Out[16]:

```

1      165
0      138
Name: target, dtype: int64

```

Checking for the existence of any duplicate values. The duplicates should be tackled down safely, otherwise would affect in generalization of the model. There might be a chance if duplicates are not dealt properly, they

might show up in the test dataset which are also in the training dataset.

```
In [17]: data[data.duplicated()==True]
```

```
Out[17]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
164	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1

```
In [18]: data.drop_duplicates(inplace=True)
data[data.duplicated()==True]
```

```
Out[18]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
--	-----	-----	----	----------	------	-----	---------	---------	-------	---------	-------	----	------	--------

```
In [19]: cmap = sns.diverging_palette(250, 15, s=75, l=40,n=9, center="dark")
```

Researchers found that throughout life, men were about twice as likely as women to have a heart attack. That higher risk persisted even after they accounted for traditional risk factors for heart disease, including high cholesterol, high blood pressure, diabetes, body mass index, and physical activity. <https://www.health.harvard.edu/heart-health/throughout-life-heart-attacks-are-twice-as-common-in-men-than-women> (<https://www.health.harvard.edu/heart-health/throughout-life-heart-attacks-are-twice-as-common-in-men-than-women>)

Creating a Normal Distributed Dataset

This new dataset comprises of equal number of heart disease and non heart disease . This, thus, helps the model to identify the pattern which causes the disease. We won't be testing the result on this new dataset but on original dataset.

```
In [20]: data = data.sample(frac=1)

# total heart disease data classes 164 rows.
non_hd_data = data.loc[data['target'] == 0]
hd_data = data.loc[data['target'] == 1][:138]

b_data = pd.concat([non_hd_data,hd_data])

# Shuffle dataframe rows
b_data = b_data.sample(frac=1, random_state=7)

b_data.head()
```

```
Out[20]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
288	57	1	0	110	335	0	1	143	1	3.0	1	1	3	0
33	54	1	2	125	273	0	0	152	0	0.5	0	1	2	1
56	48	1	0	122	222	0	0	186	0	0.0	2	0	2	1
205	52	1	0	128	255	0	1	161	1	0.0	2	1	3	0
31	65	1	0	120	177	0	1	140	0	0.4	2	0	3	1

There are three broad reasons for computing a correlation matrix.

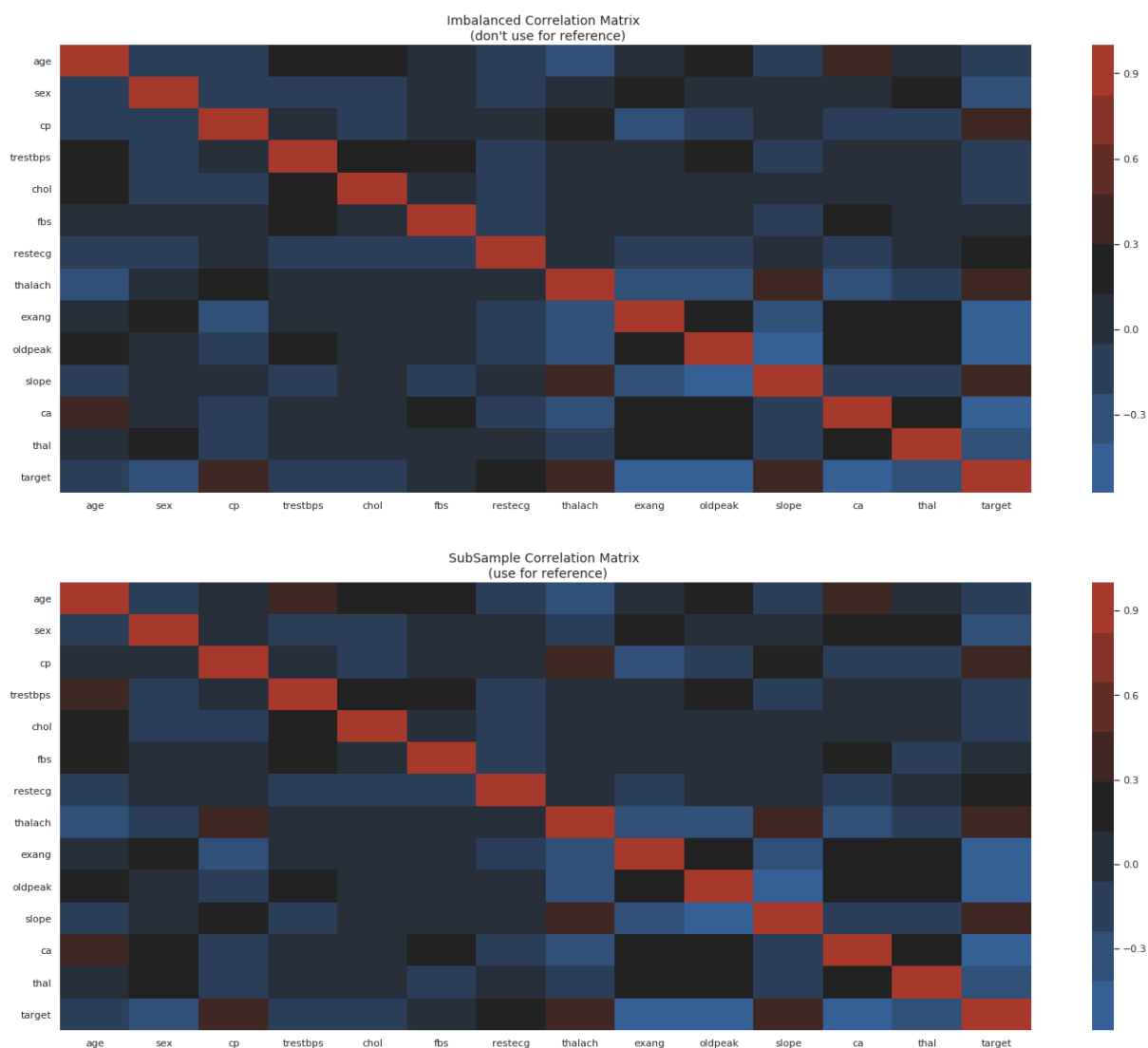
- To summarize a large amount of data where the goal is to see patterns. In our example above, the observable pattern is that all the variables highly correlate with each other.
- To input into other analyses. For example, people commonly use correlation matrixes as inputs for exploratory factor analysis, confirmatory factor analysis, structural equation models, and linear regression when excluding missing values pairwise.
- As a diagnostic when checking other analyses. For example, with linear regression a high amount of correlations suggests that the linear regression's estimates will be unreliable.

Here, we check how each values contribute to the target values

```
In [21]: f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))

# Entire DataFrame
corr = data.corr()
sns.heatmap(corr, cmap=cmap, annot_kws={'size':20}, ax=ax1)
ax1.set_title("Imbalanced Correlation Matrix \n (don't use for reference)", fo
ntsize=14)

sub_sample_corr = b_data.corr()
sns.heatmap(sub_sample_corr, cmap=cmap, annot_kws={'size':20}, ax=ax2)
ax2.set_title('SubSample Correlation Matrix \n (use for reference)', fontsize=
14)
plt.show()
```



From the correlation matrix, we analyze that , Chest Pain,fbs,restecg, thalach, slope contribute more to the heart disease and hence, are **positive correleations** .

Age, Sex, Exang, oldpeak and ca show **negative correlation** .

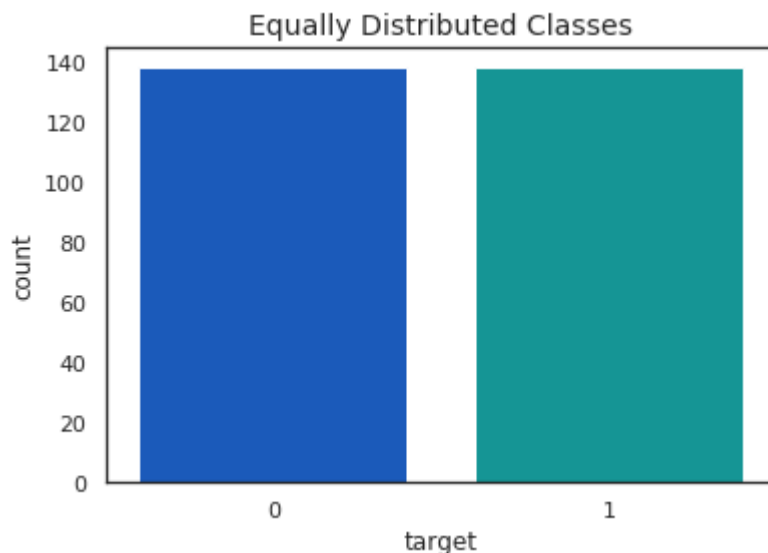
```
In [22]: print('Distribution of the Classes in the new balanced dataset')
print(b_data['target'].value_counts()/len(b_data))
sns.countplot('target', data=b_data, palette='winter')
plt.title('Equally Distributed Classes', fontsize=14)
plt.show()
```

Distribution of the Classes in the new balanced dataset

1 0.5

0 0.5

Name: target, dtype: float64



```
In [23]: fig, axes = plt.subplots(ncols=6,figsize=(20,4))
sns.boxplot(x='target',y='age',data=b_data, palette='Oranges', ax=axes[0])
axes[0].set_title('Age vs Target distribution')

sns.boxplot(x='target',y='sex' ,data=b_data, palette='Oranges', ax=axes[1])
axes[1].set_title("Sex vs Target distribution")

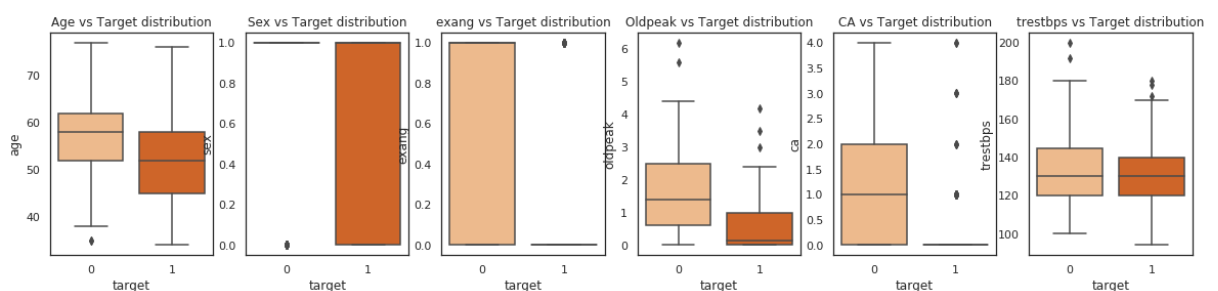
sns.boxplot(x='target',y='exang' ,data=b_data, palette='Oranges', ax=axes[2])
axes[2].set_title("exang vs Target distribution")

sns.boxplot(x='target',y='oldpeak' ,data=b_data, palette='Oranges', ax=axes[3
])
axes[3].set_title("Oldpeak vs Target distribution")

sns.boxplot(x='target',y='ca' ,data=b_data, palette='Oranges', ax=axes[4])
axes[4].set_title("CA vs Target distribution")

sns.boxplot(x='target',y='trestbps',data=b_data, palette='Oranges',ax=axes[5])
axes[5].set_title("trestbps vs Target distribution")
```

Out[23]: Text(0.5, 1.0, 'trestbps vs Target distribution')



```
In [24]: fig, axes = plt.subplots(ncols=7,figsize=(20,4))
sns.boxplot(x='target',y='cp',data=b_data, palette='winter', ax=axes[0])
axes[0].set_title('Chest Pain vs Target distribution')
```

```

sns.boxplot(x='target',y='fbs' ,data=b_data, palette='winter', ax=axes[1])
axes[1].set_title("fbs vs Target ")

sns.boxplot(x='target',y='restecg' ,data=b_data, palette='winter', ax=axes[2])
axes[2].set_title("restecg vs Target distribution")

sns.boxplot(x='target',y='thalach' ,data=b_data, palette='winter', ax=axes[3])
axes[3].set_title("thalach vs Target ")

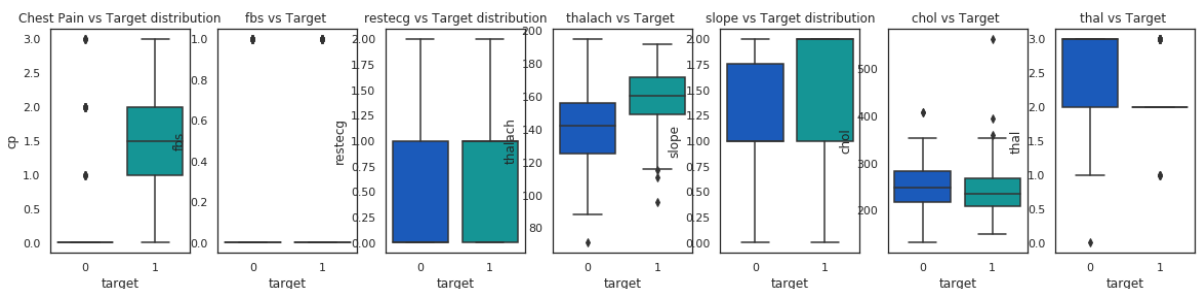
sns.boxplot(x='target',y='slope' ,data=b_data, palette='winter', ax=axes[4])
axes[4].set_title("slope vs Target distribution")

sns.boxplot(x='target',y='chol' ,data=b_data, palette='winter', ax=axes[5])
axes[5].set_title("chol vs Target ")

sns.boxplot(x='target',y='thal' ,data=b_data, palette='winter', ax=axes[6])
axes[6].set_title("thal vs Target ")

```

Out[24]: Text(0.5, 1.0, 'thal vs Target ')



From the boxplot, we analyze the outliers which have been impacting the correleation matrices, this way we can eliminate the outliers and thus we can improve the accuracy.

In [25]: `from scipy.stats import norm`

```

f,(ax1,ax2,ax3,ax4)=plt.subplots(1,4,figsize=(20,4))
age_d=b_data['age'].loc[b_data['target']==0].values
sns.distplot(age_d,ax=ax1,fit=norm,color='g')
ax1.set_title("Age Distirbution \n (Non Heart Disease)", fontsize='14')

sex_d=b_data['sex'].loc[b_data['target']==0].values
sns.distplot(sex_d,ax=ax2,fit=norm,color='red')
ax2.set_title("Sex Distribution \n (Non Heart Disease)", fontsize='14')

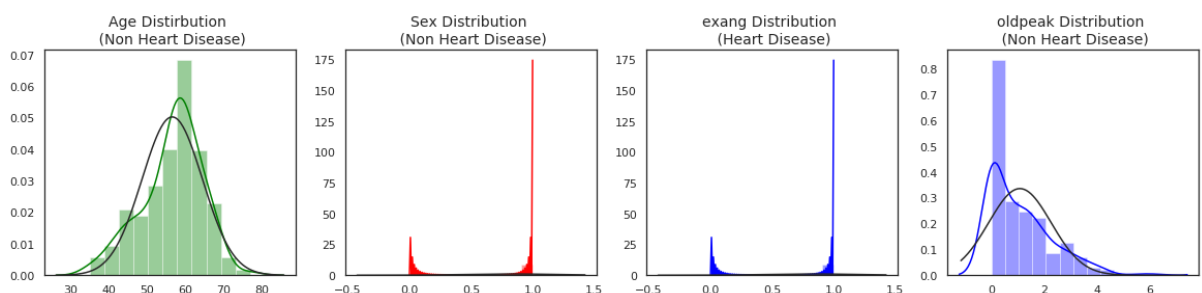
exang_d=b_data['exang'].loc[b_data['target']==1].values
sns.distplot(exang_d,ax=ax3,fit=norm,color='blue')
ax3.set_title("exang Distribution \n (Heart Disease)", fontsize='14')

'''oldpeak_d=b_data['oldpeak'].loc[b_data['target']==1].values
sns.distplot(oldpeak_d,ax=ax4,fit=norm,color='blue')
ax4.set_title("oldpeak Distribution \n (Heart Disease)", fontsize='14')'''

oldpeak_d=b_data['oldpeak'].values
sns.distplot(oldpeak_d,ax=ax4,fit=norm,color='blue')
ax4.set_title("oldpeak Distribution \n (Non Heart Disease)", fontsize='14')

```

Out[25]: Text(0.5, 1.0, 'oldpeak Distribution \n (Non Heart Disease)')



```
In [26]: f,(ax1,ax2,ax3,ax4,ax5)=plt.subplots(1,5,figsize=(20,4))
ca_d=b_data['ca'].loc[b_data['target']==1].values
sns.distplot(ca_d,ax=ax1,fit=norm,color='y')
ax1.set_title("CA Distirbution \n (Heart Disease)", fontsize='14')

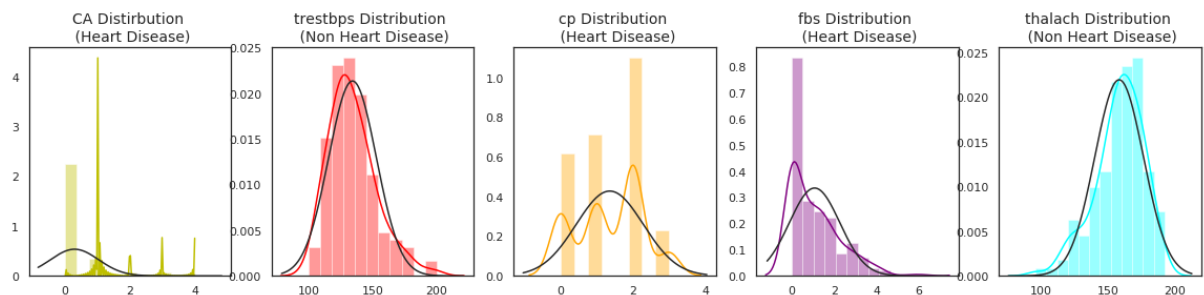
trestbps_d=b_data['trestbps'].loc[b_data['target']==0].values
sns.distplot(trestbps_d,ax=ax2,fit=norm,color='red')
ax2.set_title("trestbps Distribution \n (Non Heart Disease)", fontsize='14')

cp_d=b_data['cp'].loc[b_data['target']==1].values
sns.distplot(cp_d,ax=ax3,fit=norm,color='orange')
ax3.set_title("cp Distribution \n (Heart Disease)", fontsize='14')

fbs_d=b_data['oldpeak'].values
sns.distplot(fbs_d,ax=ax4,fit=norm,color='purple')
ax4.set_title("fbs Distribution \n (Heart Disease)", fontsize='14')

thalach_d=b_data['thalach'].loc[b_data['target']==1].values
sns.distplot(thalach_d,ax=ax5,fit=norm,color='cyan')
ax5.set_title("thalach Distribution \n (Non Heart Disease)", fontsize='14')
```

Out[26]: Text(0.5, 1.0, 'thalach Distribution \n (Non Heart Disease)')



```
In [27]: X=b_data.iloc[:, :-1]
y=b_data.iloc[:, -1]
```

```
In [28]: from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.2,random_state=20)
```

The Model

Using Random Forest and then using GridSearch CV to find the best parameter

```
In [29]: from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100,max_depth=5)
model.fit(X_train, Y_train)
```

Out[29]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=5, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)

```
In [30]: Y_pred=model.predict(X_test)
accuracy=accuracy_score(Y_test,Y_pred)
accuracy
```

Out[30]: 0.8214285714285714

```
In [31]: from sklearn.model_selection import GridSearchCV
param_grid = {'n_estimators': [1, 500], 'max_depth': [1, 15]}
```

```
parameters = {'n_estimators':[1,500], 'max_depth':[1, 15]}
clf = GridSearchCV(model, parameters, cv=5)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'max_depth': 15, 'n_estimators': 500}
```

Using the parameter achieved from the GridSearch to

```
In [32]: model = RandomForestClassifier(n_estimators=500,max_depth=1)
model.fit(X_train, Y_train)
Y_pred=model.predict(X_test)
cm=confusion_matrix(Y_test.tolist(),Y_pred.tolist())
print("Accuracy:",accuracy_score(Y_test,Y_pred))
print("Precision Score :",precision_score(Y_test,Y_pred))
print("f1 Score :",f1_score(Y_test,Y_pred))
print("Confusion Matrix: \n",cm)
```

```
Accuracy: 0.8035714285714286
Precision Score : 0.78125
f1 Score : 0.8196721311475409
Confusion Matrix:
[[20  7]
 [ 4 25]]
```

```
In [33]: total=sum(sum(cm))
sensitivity = cm[0,0]/(cm[0,0]+cm[1,0])
print('Sensitivity : ', sensitivity )
specificity = cm[1,1]/(cm[1,1]+cm[0,1])
print('Specificity : ', specificity)
```

```
Sensitivity : 0.8333333333333334
Specificity : 0.78125
```

```
In [34]: X_train.shape
```

```
Out[34]: (220, 13)
```

```
In [35]: model=Sequential()
model.add(Dense(128, init="uniform", input_dim=13, activation='relu'))
model.add(Dense(64, init="uniform", activation="relu"))
model.add(Dense(1, init="uniform", activation="sigmoid"))
model.compile(loss="binary_crossentropy", metrics=['accuracy'], optimizer='adam')
model.summary()
history=model.fit(X_train,Y_train, epochs=100, batch_size=100)
```

WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	1792
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 1)	65

=====
Total params: 10,113
Trainable params: 10,113
Non-trainable params: 0

WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python

on/ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

Epoch 1/100

220/220 [=====] - 0s 1ms/step - loss: 0.7151 - acc: 0.4955

Epoch 2/100

220/220 [=====] - 0s 26us/step - loss: 0.7009 - acc: 0.5091

Epoch 3/100

220/220 [=====] - 0s 28us/step - loss: 0.6760 - acc: 0.6091

Epoch 4/100

220/220 [=====] - 0s 24us/step - loss: 0.6659 - acc: 0.6045

Epoch 5/100

220/220 [=====] - 0s 27us/step - loss: 0.6498 - acc: 0.6636

Epoch 6/100

220/220 [=====] - 0s 23us/step - loss: 0.6381 - acc: 0.6591

Epoch 7/100

220/220 [=====] - 0s 30us/step - loss: 0.6260 - acc: 0.6500

Epoch 8/100

220/220 [=====] - 0s 23us/step - loss: 0.6232 - acc: 0.6636

Epoch 9/100

220/220 [=====] - 0s 28us/step - loss: 0.6244 - acc: 0.6182

Epoch 10/100

220/220 [=====] - 0s 25us/step - loss: 0.6049 - acc: 0.7000

Epoch 11/100

220/220 [=====] - 0s 24us/step - loss: 0.6079 - acc: 0.6636

Epoch 12/100

220/220 [=====] - 0s 26us/step - loss: 0.5916 - acc: 0.6955

Epoch 13/100

220/220 [=====] - 0s 26us/step - loss: 0.5849 - acc: 0.6864

Epoch 14/100

220/220 [=====] - 0s 27us/step - loss: 0.5923 - acc: 0.6682

Epoch 15/100

220/220 [=====] - 0s 29us/step - loss: 0.5828 - acc: 0.7091

Epoch 16/100

220/220 [=====] - 0s 28us/step - loss: 0.5733 - acc: 0.7045

Epoch 17/100

220/220 [=====] - 0s 26us/step - loss: 0.5851 - acc: 0.6773

Epoch 18/100

220/220 [=====] - 0s 24us/step - loss: 0.5704 - acc: 0.6773

Epoch 19/100

220/220 [=====] - 0s 29us/step - loss: 0.5953 - acc: 0.6818

Epoch 20/100

220/220 [=====] - 0s 30us/step - loss: 0.5821 - acc: 0.7000

Epoch 21/100

220/220 [=====] - 0s 22us/step - loss: 0.6124 - acc:

0.6545
Epoch 22/100
220/220 [=====] - 0s 24us/step - loss: 0.5592 - acc:
0.6955
Epoch 23/100
220/220 [=====] - 0s 24us/step - loss: 0.5883 - acc:
0.6864
Epoch 24/100
220/220 [=====] - 0s 24us/step - loss: 0.5590 - acc:
0.7000
Epoch 25/100
220/220 [=====] - 0s 23us/step - loss: 0.6094 - acc:
0.6545
Epoch 26/100
220/220 [=====] - 0s 24us/step - loss: 0.5523 - acc:
0.7045
Epoch 27/100
220/220 [=====] - 0s 24us/step - loss: 0.5534 - acc:
0.7182
Epoch 28/100
220/220 [=====] - 0s 22us/step - loss: 0.5393 - acc:
0.7273
Epoch 29/100
220/220 [=====] - 0s 23us/step - loss: 0.5432 - acc:
0.6955
Epoch 30/100
220/220 [=====] - 0s 27us/step - loss: 0.5504 - acc:
0.7227
Epoch 31/100
220/220 [=====] - 0s 24us/step - loss: 0.5338 - acc:
0.7227
Epoch 32/100
220/220 [=====] - 0s 26us/step - loss: 0.5509 - acc:
0.7136
Epoch 33/100
220/220 [=====] - 0s 25us/step - loss: 0.5261 - acc:
0.7182
Epoch 34/100
220/220 [=====] - 0s 27us/step - loss: 0.5373 - acc:
0.7045
Epoch 35/100
220/220 [=====] - 0s 23us/step - loss: 0.5403 - acc:
0.7091
Epoch 36/100
220/220 [=====] - 0s 24us/step - loss: 0.5189 - acc:
0.7136
Epoch 37/100
220/220 [=====] - 0s 26us/step - loss: 0.5367 - acc:
0.7091
Epoch 38/100
220/220 [=====] - 0s 32us/step - loss: 0.5171 - acc:
0.7227
Epoch 39/100
220/220 [=====] - 0s 26us/step - loss: 0.5297 - acc:
0.7227
Epoch 40/100
220/220 [=====] - 0s 26us/step - loss: 0.5191 - acc:
0.7364
Epoch 41/100
220/220 [=====] - 0s 25us/step - loss: 0.5152 - acc:
0.7364
Epoch 42/100
220/220 [=====] - 0s 31us/step - loss: 0.5031 - acc:
0.7318
Epoch 43/100
220/220 [=====] - 0s 25us/step - loss: 0.5004 - acc:

0.7364
Epoch 44/100
220/220 [=====] - 0s 25us/step - loss: 0.4981 - acc:
0.7455
Epoch 45/100
220/220 [=====] - 0s 24us/step - loss: 0.4959 - acc:
0.7409
Epoch 46/100
220/220 [=====] - 0s 25us/step - loss: 0.4936 - acc:
0.7455
Epoch 47/100
220/220 [=====] - 0s 24us/step - loss: 0.4912 - acc:
0.7455
Epoch 48/100
220/220 [=====] - 0s 24us/step - loss: 0.4840 - acc:
0.7364
Epoch 49/100
220/220 [=====] - 0s 35us/step - loss: 0.4846 - acc:
0.7500
Epoch 50/100
220/220 [=====] - 0s 37us/step - loss: 0.4787 - acc:
0.7591
Epoch 51/100
220/220 [=====] - 0s 27us/step - loss: 0.4776 - acc:
0.7545
Epoch 52/100
220/220 [=====] - 0s 24us/step - loss: 0.4838 - acc:
0.7545
Epoch 53/100
220/220 [=====] - 0s 26us/step - loss: 0.4715 - acc:
0.7682
Epoch 54/100
220/220 [=====] - 0s 27us/step - loss: 0.4784 - acc:
0.7500
Epoch 55/100
220/220 [=====] - 0s 24us/step - loss: 0.4625 - acc:
0.7682
Epoch 56/100
220/220 [=====] - 0s 24us/step - loss: 0.4727 - acc:
0.7727
Epoch 57/100
220/220 [=====] - 0s 23us/step - loss: 0.4800 - acc:
0.7500
Epoch 58/100
220/220 [=====] - 0s 24us/step - loss: 0.4801 - acc:
0.7455
Epoch 59/100
220/220 [=====] - 0s 24us/step - loss: 0.4727 - acc:
0.7591
Epoch 60/100
220/220 [=====] - 0s 23us/step - loss: 0.4605 - acc:
0.7727
Epoch 61/100
220/220 [=====] - 0s 26us/step - loss: 0.4766 - acc:
0.7545
Epoch 62/100
220/220 [=====] - 0s 25us/step - loss: 0.4622 - acc:
0.7500
Epoch 63/100
220/220 [=====] - 0s 26us/step - loss: 0.4823 - acc:
0.7591
Epoch 64/100
220/220 [=====] - 0s 26us/step - loss: 0.4336 - acc:
0.7909
Epoch 65/100
220/220 [=====] - 0s 24us/step - loss: 0.4331 - acc:

0.7818
Epoch 66/100
220/220 [=====] - 0s 27us/step - loss: 0.4324 - acc:
0.7727
Epoch 67/100
220/220 [=====] - 0s 28us/step - loss: 0.4235 - acc:
0.7818
Epoch 68/100
220/220 [=====] - 0s 29us/step - loss: 0.4218 - acc:
0.7818
Epoch 69/100
220/220 [=====] - 0s 23us/step - loss: 0.4184 - acc:
0.7909
Epoch 70/100
220/220 [=====] - 0s 24us/step - loss: 0.4146 - acc:
0.7864
Epoch 71/100
220/220 [=====] - 0s 26us/step - loss: 0.4097 - acc:
0.7864
Epoch 72/100
220/220 [=====] - 0s 26us/step - loss: 0.4047 - acc:
0.8045
Epoch 73/100
220/220 [=====] - 0s 25us/step - loss: 0.3967 - acc:
0.8182
Epoch 74/100
220/220 [=====] - 0s 26us/step - loss: 0.4059 - acc:
0.8045
Epoch 75/100
220/220 [=====] - 0s 25us/step - loss: 0.4137 - acc:
0.7955
Epoch 76/100
220/220 [=====] - 0s 25us/step - loss: 0.3916 - acc:
0.8182
Epoch 77/100
220/220 [=====] - 0s 33us/step - loss: 0.3919 - acc:
0.8227
Epoch 78/100
220/220 [=====] - 0s 24us/step - loss: 0.4100 - acc:
0.7864
Epoch 79/100
220/220 [=====] - 0s 24us/step - loss: 0.3977 - acc:
0.7909
Epoch 80/100
220/220 [=====] - 0s 23us/step - loss: 0.3980 - acc:
0.7955
Epoch 81/100
220/220 [=====] - 0s 24us/step - loss: 0.3731 - acc:
0.8364
Epoch 82/100
220/220 [=====] - 0s 23us/step - loss: 0.4161 - acc:
0.8045
Epoch 83/100
220/220 [=====] - 0s 25us/step - loss: 0.4642 - acc: