```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
data = pd.read_csv('Admission_Predict.csv')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

```
data.shape
```

```
(400, 9)
```

```
data.isnull().any()
```

```
Serial No.           False
GRE Score            False
TOEFL Score          False
University Rating    False
SOP                  False
LOR                  False
CGPA                 False
Research             False
Chance of Admit      False
dtype: bool
```

```
data=data.rename(columns = {'Chance of Admit ':'Chance of Admit'})
```

```
data.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CG
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000(
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598(
std	115.614301	11.473646	6.069514	1.143728	1.006869	0.898478	0.596(
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.800(
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.170(
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.610(
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.062(
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920(

```
sns.distplot(data['GRE Score'])
```

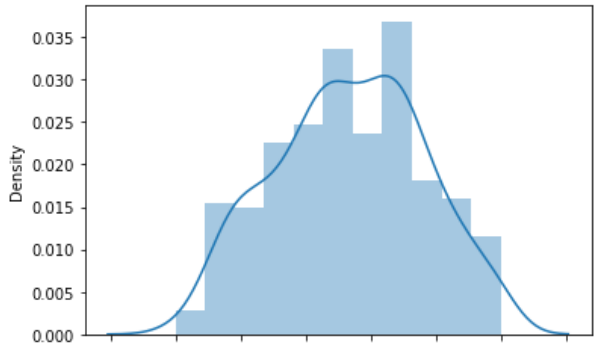
```
<ipython-input-114-64e93544a305>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['GRE Score'])
<Axes: xlabel='GRE Score', ylabel='Density'>
```

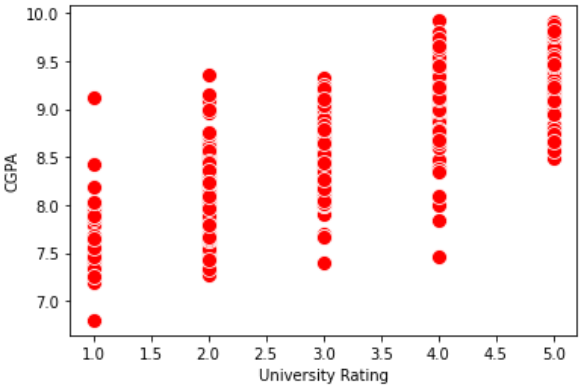



```
sns.pairplot(data=data,hue='Research',markers=["^","v"],palette='inferno')
```

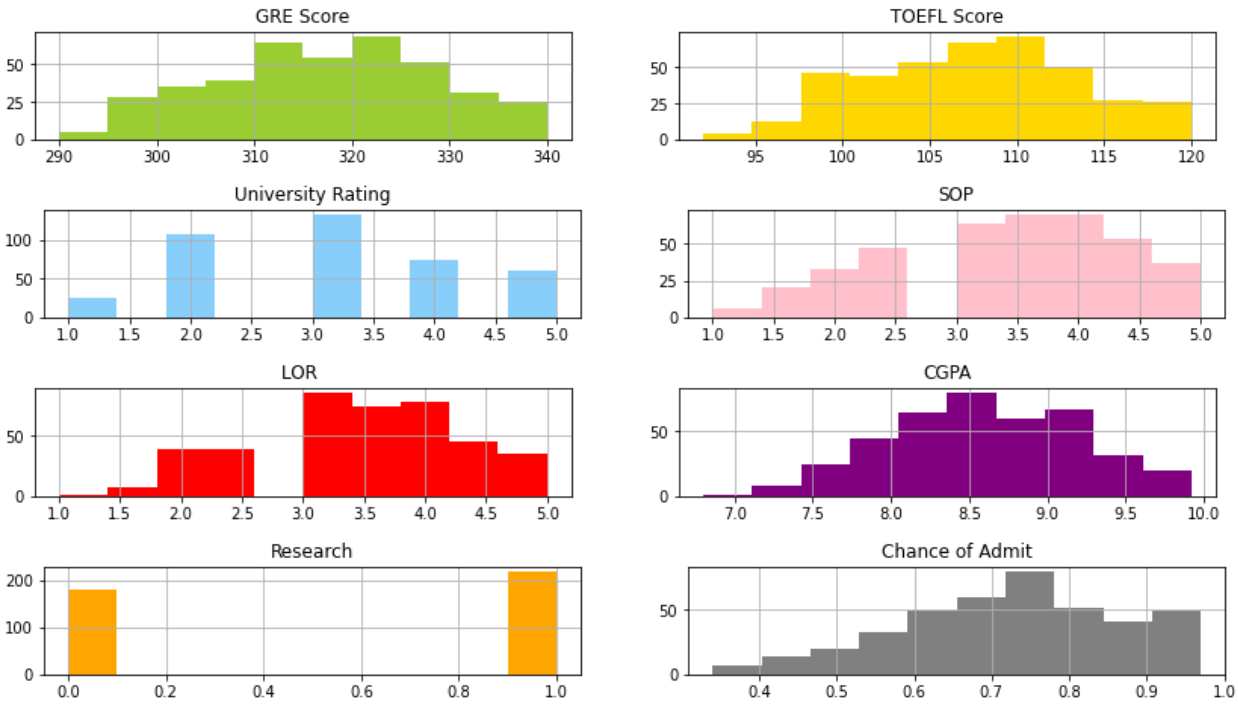
```
<seaborn.axisgrid.PairGrid at 0x7f306fe55dc0>
```

```
sns.scatterplot(x='University Rating',y='CGPA',data=data,color='Red',s=100)
```

```
<Axes: xlabel='University Rating', ylabel='CGPA'>
```



```
category = ['GRE Score','TOEFL Score','University Rating','SOP','LOR ','CGPA','Research','Chance of Admit']
color = ['Yellowgreen','gold','lightskyblue','pink','red','purple','orange','gray']
start = True
for i in np.arange(4):
  fig = plt.figure(figsize=(14,8))
  plt.subplot2grid((4,2),(i,0))
  data[category[2*i]].hist(color=color[2*i],bins=10)
  plt.title(category[2*i])
  plt.subplot2grid((4,2),(i,1))
  data[category[2*i+1]].hist(color=color[2*i+1],bins=10)
  plt.title(category[2*i+1])
  plt.subplots_adjust(hspace = 0.7,wspace = 0.2)
  plt.show()
```




```
X=data.drop(['Serial No.','Chance of Admit'],axis=1) #input data_set
X.shape
```

```
(400, 7)
```

```
y=data['Chance of Admit'] #output labels
y.shape
```

```
(400,)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.20)
```

```
X_train.shape
```

```
(320, 7)
```

```
Y_train.shape
```

```
(320,)
```

```
X_test.shape
```

```
(80, 7)
```

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
X_train[X_train.columns] = scaler.fit_transform(X_train[X_train.columns])
X_test[X_test.columns] = scaler.transform(X_test[X_test.columns])
X_train.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
55	0.60	0.392857	0.50	0.50	0.500	0.288462	0.0
92	0.16	0.214286	0.25	0.75	0.500	0.394231	0.0
98	0.84	0.964286	0.75	1.00	0.875	0.782051	1.0
61	0.34	0.321429	0.50	0.75	0.500	0.448718	0.0
267	0.48	0.535714	0.50	0.50	0.625	0.439103	1.0

```
from sklearn.ensemble import RandomForestRegressor
rgr=RandomForestRegressor()
rgr.fit(X_train,y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
rgr.score(X_test,y_test)
```

```
0.8645939004471195
```

```
import xgboost as xgb
xg = xgb.XGBRegressor()
xg.fit(X_train,y_train)
```

```
▾                                XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

```
xg.score(X_test,y_test)
```

```
0.8512548384314844
```

```
y_predict=rgr.predict(X_test)
y_predict
```

```
array([[1.    , 0.7321],
       [0.    , 0.6412],
       [1.    , 0.8807],
       [0.    , 0.5496],
       [0.    , 0.6561],
       [1.    , 0.8755],
       [1.    , 0.6466],
       [1.    , 0.7712],
       [0.    , 0.6509],
       [1.    , 0.73  ],
       [0.    , 0.7268],
       [1.    , 0.8065],
       [1.    , 0.7059],
       [0.    , 0.6641],
       [1.    , 0.7094],
       [1.    , 0.9599],
       [1.    , 0.6336],
       [0.    , 0.4431],
       [0.    , 0.6567],
       [0.    , 0.587 ],
       [0.    , 0.6544],
       [1.    , 0.79  ],
       [0.    , 0.7028],
       [1.    , 0.6088],
       [0.    , 0.7017],
       [0.    , 0.7309],
       [0.    , 0.7008],
       [1.    , 0.919 ],
       [1.    , 0.9193],
       [0.    , 0.687 ],
       [0.    , 0.607 ],
       [1.    , 0.5837],
       [1.    , 0.7388],
       [0.    , 0.5429],
       [0.    , 0.7129],
       [1.    , 0.8724],
       [1.    , 0.7848],
       [1.    , 0.8996],
       [0.    , 0.6801],
       [1.    , 0.5964],
       [0.    , 0.5093],
       [1.    , 0.7769],
       [1.    , 0.7505],
       [1.    , 0.9323],
       [1.    , 0.6575],
       [1.    , 0.8772],
       [0.    , 0.68  ],
       [1.    , 0.5415],
       [0.    , 0.5718],
       [0.    , 0.7226],
       [1.    , 0.8137],
       [1.    , 0.7315],
       [1.    , 0.9682],
       [0.    , 0.6715],
       [0.    , 0.6651],
       [0.    , 0.6851],
       [0.    , 0.6884],
       [0.    , 0.6544],
```

```
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error
import numpy as np

print('Mean Absolute Error:', mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', mean_squared_error(y_test, y_predict))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_predict)))
```

```
Mean Absolute Error: 0.026305000000000005
Mean Squared Error: 0.0023678379999999997
Root Mean Squared Error: 0.048660435674169625
```

```
x=data.iloc[:,0:7].values
x
```

```
array([[  1.  , 337.  , 118.  , ...,   4.5 ,   4.5 ,   9.65],
       [  2.  , 324.  , 107.  , ...,   4.  ,   4.5 ,   8.87],
       [  3.  , 316.  , 104.  , ...,   3.  ,   3.5 ,   8.  ],
       ...,
       [398.  , 330.  , 116.  , ...,   5.  ,   4.5 ,   9.45],
       [399.  , 312.  , 103.  , ...,   3.5 ,   4.  ,   8.78],
       [400.  , 333.  , 117.  , ...,   5.  ,   4.  ,   9.66]])
```

```
y=data.iloc[:,7].values
y
```

```
array([1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1])
```

```
y_train=(y_train>0.5)
y_train
```

```
array([[ True,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [ True, False],
       [ True,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [False, False],
       [ True, False],
       [False,  True],
       [ True,  True],
       [False,  True],
       [False, False],
       [False,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [ True,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [False,  True],
       [ True,  True],
       [False,  True],
       [False,  True],
       [False,  True],
       [False,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [ True,  True],
```

```
       [False,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [ True,  True],
       [ True,  True],
       [False,  True],
       [ True,  True],
       [False,  True],
       [ True,  True],
       [False, False],
       [False,  True],
```

```
from sklearn.ensemble import RandomForestRegressor
rgr=RandomForestRegressor()
rgr.fit(X_train,y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-51-be4629b61eca> in <module>
      1 from sklearn.ensemble import RandomForestRegressor
      2 rgr=RandomForestRegressor()
----> 3 rgr.fit(X_train,y_train)

                                 3 frames
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py in check_consistent_length(*arrays)
    395     uniques = np.unique(lengths)
    396     if len(uniques) > 1:
--> 397         raise ValueError(
    398             "Found input variables with inconsistent numbers of samples: %r"
    399             % [int(l) for l in lengths]

ValueError: Found input variables with inconsistent numbers of samples: [320, 280]

SEARCH STACK OVERFLOW
```

```
y_test=(y_test>0.5)
```

```
from sklearn import linear_model
```

```
logr=linear_model.LogisticRegression()
logr.fit(x_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-72-b48356931302> in <module>
      1 logr=linear_model.LogisticRegression()
----> 2 logr.fit(x_train, y_train)

                                 4 frames
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py in column_or_1d(y, dtype, warn)
   1200         return _asarray_with_order(xp.reshape(y, -1), order="C", xp=xp)
   1201
-> 1202     raise ValueError(
   1203         "y should be a 1d array, got an array of shape {} instead.".format(shape)
   1204     )

ValueError: y should be a 1d array, got an array of shape (320, 2) instead.

SEARCH STACK OVERFLOW
```

```
from sklearn.linear_model.LogisticRegression import LogisticRegression
cls =LogisticRegression(random_state =0)
lr=cls.fit(x_train, y_train)
```

```
y_pred =lr.predict(x_test)
y_pred
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-70-90ef05a43428> in <module>
----> 1 from sklearn.linear_model.LogisticRegression import LogisticRegression
      2 cls =LogisticRegression(random_state =0)
      3 lr=cls.fit(x_train, y_train)
      4 y_pred =lr.predict(x_test)
      5 y_pred

ModuleNotFoundError: No module named 'sklearn.linear_model.LogisticRegression'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

SEARCH STACK OVERFLOW

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Activation, Dropout
from tensorflow.keras.optimizers import Adam
```

```
pip install scikit-learn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.9/dist-packages (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-packages (from scikit-learn) (1.1.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-packages (from scikit-learn) (1.22.4
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn)
```

```
model=Sequential()
model.add(Dense(7,activation ='relu',input_dim=7))
model.add(Dense(7,activation='relu'))
model.add(Dense(1,activation='linear'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 56

 dense_1 (Dense)             (None, 7)                 56

 dense_2 (Dense)             (None, 1)                 8

=================================================================
Total params: 120
Trainable params: 120
Non-trainable params: 0
_________________________________________________________________
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 56

 dense_1 (Dense)             (None, 7)                 56

 dense_2 (Dense)             (None, 1)                 8
```

```
=================================================================
Total params: 120
Trainable params: 120
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
model.fit(X_train, Y_train ,batch_size = 20, epochs = 100)
```

```
Epoch 1/100
16/16 [==============================] - 1s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 2/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 3/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 4/100
16/16 [==============================] - 0s 3ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 5/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 6/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 7/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 8/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 9/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 10/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 11/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 12/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 13/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 14/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 15/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 16/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 17/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 18/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 19/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 20/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 21/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 22/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 23/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 24/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 25/100
16/16 [==============================] - 0s 3ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 26/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 27/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 28/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
Epoch 29/100
16/16 [==============================] - 0s 2ms/step - loss: 6.6239 - accuracy: 0.5656
```

```
from sklearn.metrics import accuracy_score
train_predictions = model.predict(X_train)
print(train_predictions)
```

```
10/10 [==============================] - 0s 2ms/step
[[37.273647]
 [36.787815]
 [37.923885]
```

```
       [39.464966]
       [34.76741 ]
       [38.41909 ]
       [38.924435]
       [37.176636]
       [37.57719 ]
       [38.131252]
       [38.552387]
       [37.67822 ]
       [35.68864 ]
       [37.674908]
       [36.89513 ]
       [37.560062]
       [36.837364]
       [37.993275]
       [38.068462]
       [37.24006 ]
       [36.171654]
       [36.40561 ]
       [38.156116]
       [39.559914]
       [37.189247]
       [38.012554]
       [37.50345 ]
       [36.321228]
       [36.29469 ]
       [34.577885]
       [34.881382]
       [37.522907]
       [35.842518]
       [36.42893 ]
       [35.246155]
       [36.296577]
       [37.585148]
       [37.630455]
       [38.42495 ]
       [38.17503 ]
       [35.53037 ]
       [36.680527]
       [37.576492]
       [39.5046  ]
       [37.241177]
       [38.201893]
       [37.591892]
       [37.636566]
       [38.801025]
       [38.041904]
       [34.811424]
       [36.75469 ]
       [39.767624]
       [38.805954]
       [38.955135]
       [38.21064 ]
       [37.54479 ]
```

```
train_acc = model.evaluate(X_train, Y_train, verbose=0)[1]
print(train_acc)
```

```
0.565625011920929
```

```
test_acc = model.evaluate(X_test, Y_test, verbose=0)[1]
print(test_acc)
```

```
0.4749999940395355
```

```
Y_pred = model.predict(X_test)
Y_pred = (Y_pred > 0.5)
```

```
3/3 [==============================] - 0s 5ms/step
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

```
[[ 0 42]
 [ 0 38]]
```

```
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-34-8d4c999ca11b> in <module>
----> 1 model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

NameError: name 'model' is not defined
```

SEARCH STACK OVERFLOW

```
model.fit(x_train,y_train,batch_size = 20, epochs = 100)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-40-14986039b438> in <module>
----> 1 model.fit(x_train,y_train,batch_size = 20, epochs = 100)

AttributeError: module 'tensorflow.keras' has no attribute 'fit'
```

SEARCH STACK OVERFLOW