Connect colab to drive to access dataset

```
from google.colab import drive
drive.mount("/Drive")
```

```
Drive already mounted at /Drive; to attempt to forcibly remount, call drive.mount("/
```

```
import pandas as pd
```

verify data in dataset

```
df = pd.read_csv('/Drive/My Drive/Colab Notebooks/lab experiment/exam/ilp.csv')
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Amin
0	65	Female	0.7	0.1	187	
1	62	Male	10.9	5.5	699	
2	62	Male	7.3	4.1	490	
3	58	Male	1.0	0.4	182	
4	72	Male	3.9	2.0	195	

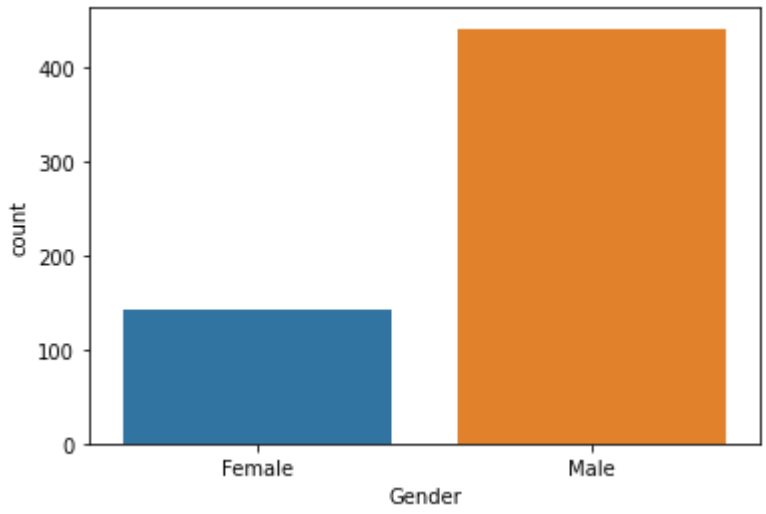
```
df.info()
```

```
df.shape
```

```
(583, 11)
```

```
import seaborn as sns

n_records = len(df.index)
n_records_liv_pos = len(df[df['Dataset'] == 1])
n_records_liv_neg = len(df[df['Dataset'] == 2])
percent_liver_disease_pos = (n_records_liv_pos/n_records)*100

print("Number of records: {}".format(n_records))
print("Number of patients likely to have liver disease {}".format(n_records_liv_pos))
print("Number of patients unlikely to have liver disease {}".format(n_records_liv_neg))
print("Percentage of patients likely to have liver disease {}%".format(percent_liver_disea

sns.countplot(data=df, x = 'Gender', label='Count')
```

```
Number of records: 583
Number of patients likely to have liver disease 416
Number of patients unlikely to have liver disease 167
Percentage of patients likely to have liver disease 71.35506003430532%
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarni
  import pandas.util.testing as tm
<matplotlib.axes._subplots.AxesSubplot at 0x7f3fe369a748>
```



changing gender values to 0, 1 ; 0 indicates female and 1 indicate male

```
df['Gender'] = df['Gender'].astype('category')
df['Gender'] = df['Gender'].cat.codes
```

```
df[:10]
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Amin
0	65	0	0.7	0.1	187	
1	62	1	10.9	5.5	699	
2	62	1	7.3	4.1	490	
3	58	1	1.0	0.4	182	
4	72	1	3.9	2.0	195	
5	46	1	1.8	0.7	208	
6	26	0	0.9	0.2	154	
7	29	0	0.9	0.3	202	
8	17	1	0.9	0.3	202	
9	55	1	0.7	0.2	290	

```
# Drop Dataset field
#df = df.drop(['Dataset'], axis=1)
#df.head()
```

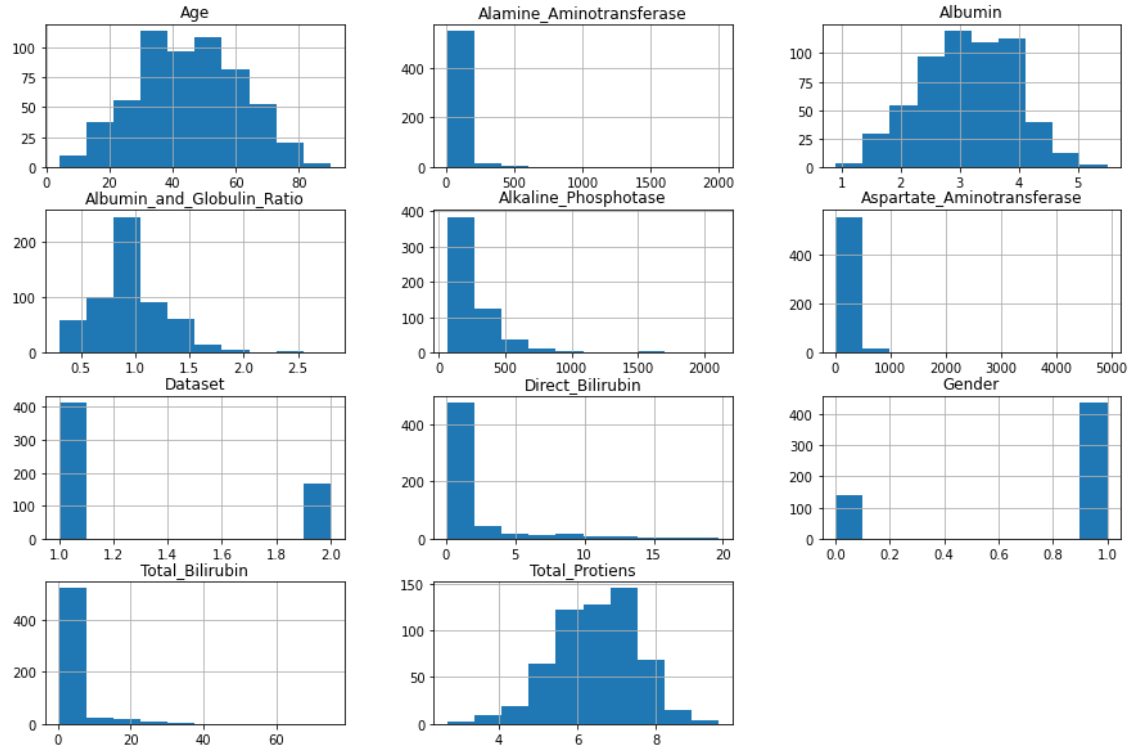
Remove missing values

```
df = df.dropna()
df[:10]
```

```
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  Alamine_Amin
```

```
# plot histogram
df.hist(figsize=(15,10))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe3157b38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe3104dd8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe30c5080>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe30f82e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe30ab550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe305f7b8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe3012a20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe2fc5c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe2fc5cc0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe2fbb198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe2f70400>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3fe2f22668>]],
      dtype=object)
```

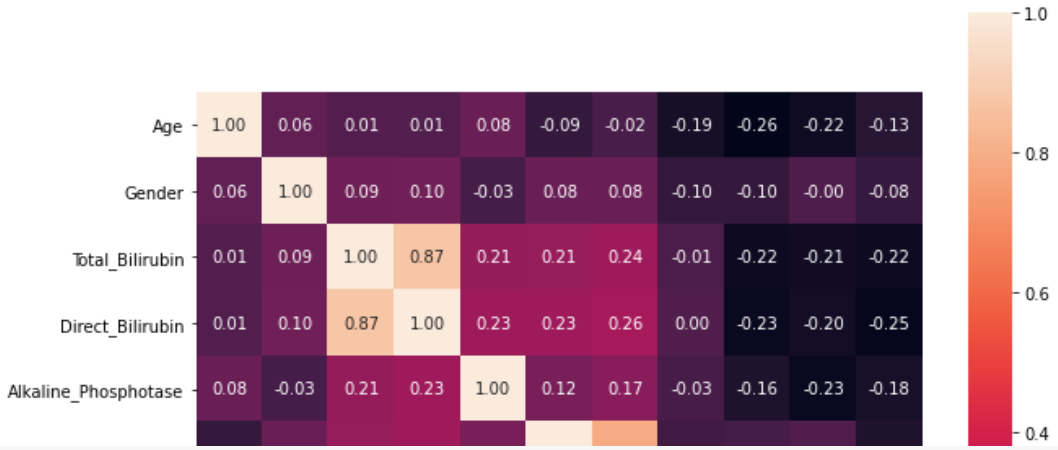



```
df.describe()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase
count	579.000000	579.000000	579.000000	579.000000	579.000000
mean	44.782383	0.758204	3.315371	1.494128	291.366149
std	16.221786	0.428542	6.227716	2.816499	243.561863
min	4.000000	0.000000	0.400000	0.100000	63.000000
25%	33.000000	1.000000	0.800000	0.200000	175.500000
50%	45.000000	1.000000	1.000000	0.300000	208.000000
75%	58.000000	1.000000	2.600000	1.300000	298.000000
max	90.000000	1.000000	75.000000	19.700000	2110.000000

```
from matplotlib import pyplot as plt
%matplotlib inline
```

```
# calculate correlation coefficients for the dataset
correlations = df.corr()

# and visualize
plt.figure(figsize=(10, 10))
g = sns.heatmap(correlations, cbar = True, square = True, annot=True, fmt= '.2f', annot_kw
```




```
from sklearn.preprocessing import MinMaxScaler
from IPython.display import display

scaler = MinMaxScaler()
numerical = ['Age', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase', 'Alamir
             'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin', 'Albumin_and_Globu

liver_minmax_transform = pd.DataFrame(data = df)
liver_minmax_transform[numerical] = scaler.fit_transform(df[numerical])

display(liver_minmax_transform.head(n = 5))
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_
0	0.709302	0	0.004021	0.000000	0.060576	
1	0.674419	1	0.140751	0.275510	0.310699	
2	0.674419	1	0.092493	0.204082	0.208598	
3	0.627907	1	0.008043	0.015306	0.058134	
4	0.790698	1	0.046917	0.096939	0.064485	

```
import numpy as np
```

```
X = np.array(df)[:, 0:10]
X
```

```
array([[0.70930233, 0.        , 0.00402145, ..., 0.5942029 , 0.52173913,
        0.24      ],
       [0.6744186 , 1.        , 0.14075067, ..., 0.69565217, 0.5       ,
        0.176     ]
```

```
y = np.array(df)[:, 10].reshape(X.shape[0],1) - 1
y[:10]
```

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

Optimization algorithm for binary classification of liver dieseae (1=+/ 0=-)

```
def sigmoid(X, c):
    return 1/(1+np.exp(-X.dot(c)))
```

```
def logistic_obj(X,c,y):

    # c dx1 vector of weights
    # X nxd matrix of data values
    # y nx1 vector of labels

    obj_value = (-1/X.shape[0]) * np.sum(y*np.log(sigmoid(X, c)) + (1 - y)*np.log(1 - sigm

    return obj_value
```

```
def logistic_grad(X,c,y):

    return (1/X.shape[0]) * np.dot(X.T, sigmoid(X, c) - y)
```

```
def logistic_sgd(X,c,y):

    (n,d) = X.shape

    idx = np.random.randint(n)

    x = X[idx, :].reshape(1, d)

    z = np.dot(x.T, sigmoid(x, c) - y)
```

```
    z = np.dot(X.T, sigmoid(X, c) - y)

    return z.T
```

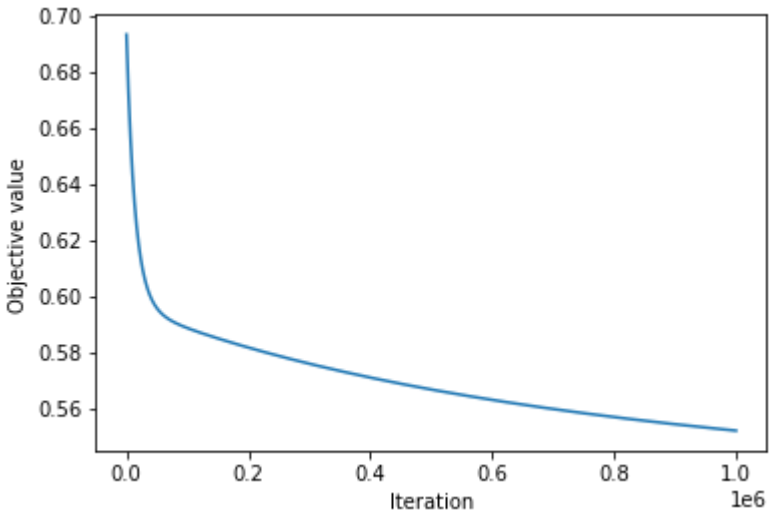
```
x0 = np.array([50., 1., 0.5, 5.,400.,25.,8.,7.2,3.2,0.8]).reshape(10,1)
x1 = np.array([54., 1., 0.3, 5.5,200.,23.,7.,7.4,3.1,0.5]).reshape(10,1)

[x0, x1]
```

```
[array([[ 50. ],
        [  1. ],
        [  0.5],
        [  5. ],
        [400. ],
        [ 25. ],
        [  8. ],
        [  7.2],
        [  3.2],
        [  0.8]]), array([[ 54. ],
        [  1. ],
        [  0.3],
        [  5.5],
        [200. ],
        [ 23. ],
        [  7. ],
        [  7.4],
        [  3.1],
        [  0.5]])]
```

```
eta = 0.0001

iterations = 1000000

c = np.zeros((10,1))

# We will use the following vector to keep track of objective values
objective_values = np.zeros((iterations,1))

for i in range(iterations):

    c = c - eta * logistic_grad(X,c,y)

    # store current objective value
    objective_values[i] = logistic_obj(X,c,y)



# Plot the objective as a function of the iteration
plt.plot(objective_values)
plt.xlabel('Iteration')
plt.ylabel('Objective value')
```

```
print(c)
```

```
[[-1.04532219]
 [-0.41555215]
 [-0.62292214]
 [-1.20909339]
 [-0.86616795]
 [-0.58406425]
 [-0.33115585]
 [-0.40171405]
 [ 0.4226848 ]
 [ 0.43187364]]
```



Confusion matrix is used to evaluate the accuracy of a classification

```
from sklearn.metrics import confusion_matrix

confusion_matrix(sigmoid(X, c) > 0.5, y == 1)
```

```
array([[414, 163],
       [  0,   2]])
```

Compare Naive Prediction with confusion_matrix prediction

```
liver_data_labels = df['Dataset']
true_pos = n_records_liv_pos
false_pos = liver_data_labels.count() - true_pos
true_neg = 0
false_neg = 0
print('true positives: {} | true negatives: {}'.format(true_pos,false_pos))
```

```
# Calculate accuracy, precision and recall
accuracy = true_pos/liver_data_labels.count()
recall = true_pos/(true_pos + false_neg)
precision = true_pos/(true_pos + false_pos)
print('accuracy: {} | precision: {} | recall: {}'.format(accuracy, precision, recall))

# Calculate F-score using the formula above
beta = 2
#F score, also called the F1 score or F measure, is a measure of a test’s accuracy.
fscore = (1 + beta * beta) * (precision * recall) / (beta * beta * precision + recall)

# Result
print("Naive Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, fscore
```

```
true positives: 416 | true negatives: 163
accuracy: 0.7184801381692574 | precision: 0.7184801381692574 | recall: 1.0
Naive Predictor: [Accuracy score: 0.7185, F-score: 0.9273]
```