**Name - Anjali.N

**Project name - Loan Status Prediction

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

df=pd.read_csv("loan.csv")
df.head()
```

Out[1]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

In [2]:
```
df.tail()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

In [3]:
```
df.describe()
```

Out[3]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [4]:
```
df['Property_Area'].value_counts()
```

Out[4]:
```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

In [5]:
```
df['Loan_Status'].value_counts()
```

Out[5]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [6]:
```
df['Self_Employed'].value_counts()
```

Out[6]:
```
No     500
Yes     82
Name: Self_Employed, dtype: int64
```

In [7]:
```
df['Gender'].value_counts()
```

Out[7]:
```
Male      489
Female    112
Name: Gender, dtype: int64
```

In [8]:
```
df['Married'].value_counts()
```

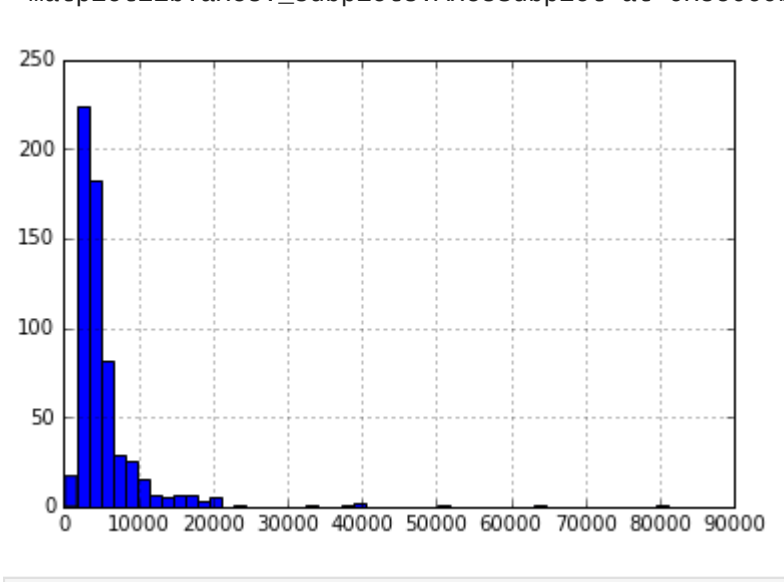
Out[8]:
```
Yes    398
No     213
Name: Married, dtype: int64
```

In [9]:
```
df['Education'].value_counts()
```

Out[9]:
```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

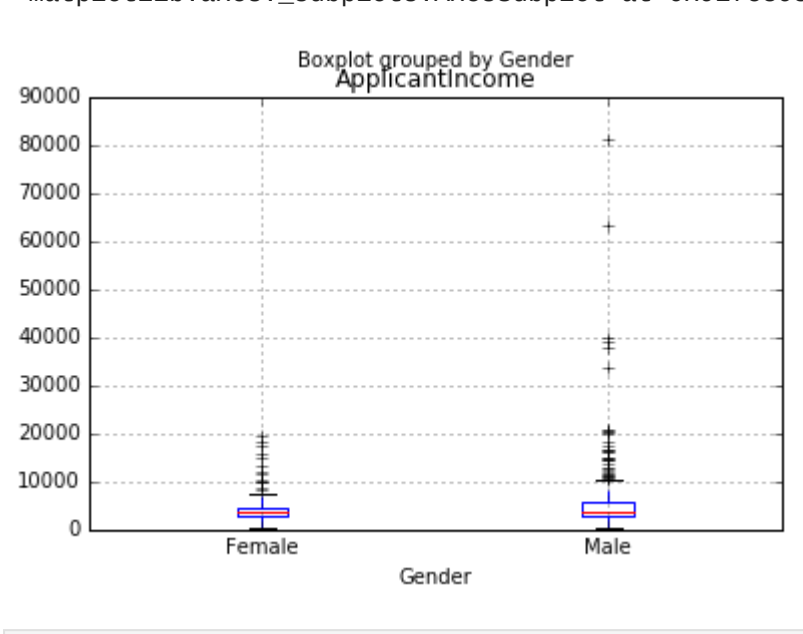
In [10]:
```
df['ApplicantIncome'].hist(bins=50)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x8e660b8>



In [11]:
```
df.boxplot(column='ApplicantIncome', by = 'Gender')
```

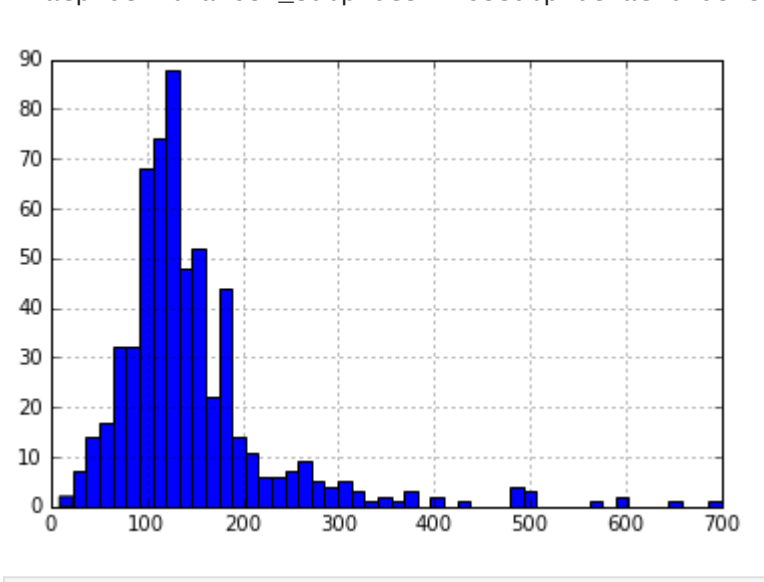
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x917e898>



In [12]:
```
#acc to above output, there are more men with more income
```

In [13]:
```
df['LoanAmount'].hist(bins=50)  #X-axis: the values of loan amount #Y-axis: the frequencies of different values of loan amount
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x9378358>



In [14]:
```
df['Credit_History'].value_counts()
```

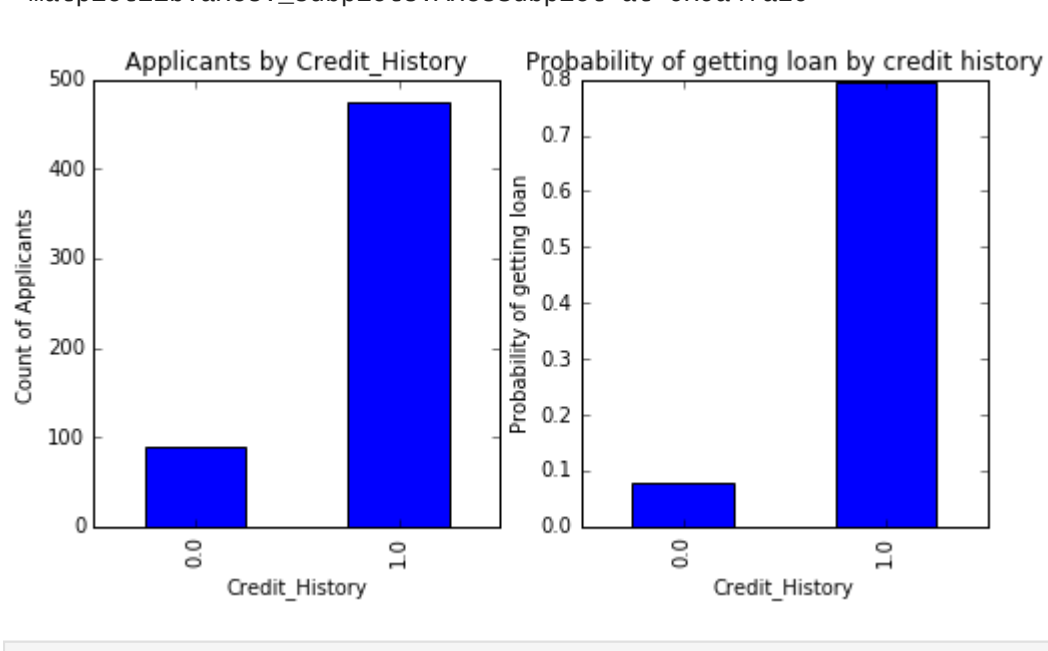
Out[14]:
```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [15]:
```
#creation of pivot tables using python
#the probability of getting loan as per the credit history
pt = df.pivot_table(values='Loan_Status',index=['Credit_History'],aggfunc=lambda x: x.map({'Y':1,'N':0}).mean())
pt
```

Out[15]:
```
Credit_History
0.0    0.078652
1.0    0.795789
Name: Loan_Status, dtype: float64
```

In [17]:
```
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)

ax1.set_xlabel('Credit_History')
ax1.set_ylabel('Count of Applicants')
ax1.set_title("Applicants by Credit_History")
ch.plot(kind='bar')

ax2 = fig.add_subplot(122)
ax2.set_xlabel('Credit_History')
ax2.set_ylabel('Probability of getting loan')
ax2.set_title("Probability of getting loan by credit history")
pt.plot(kind = 'bar')
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x9a4fa20>

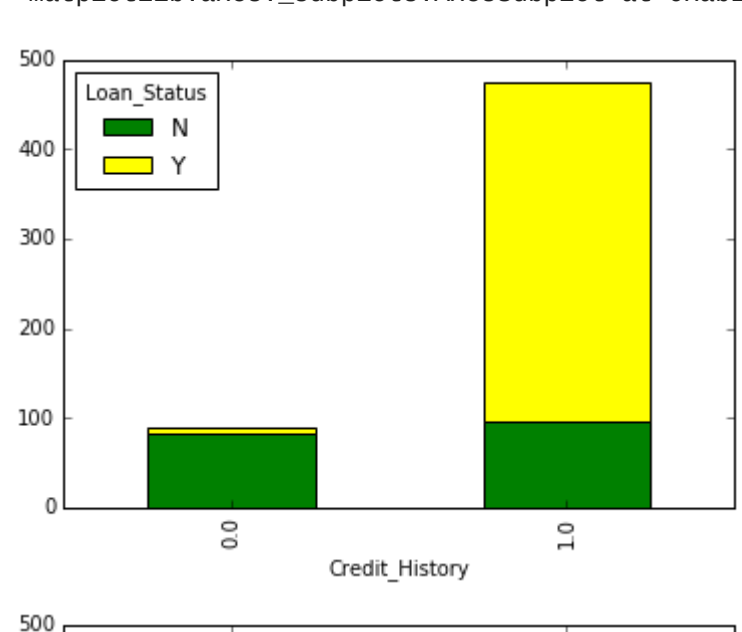


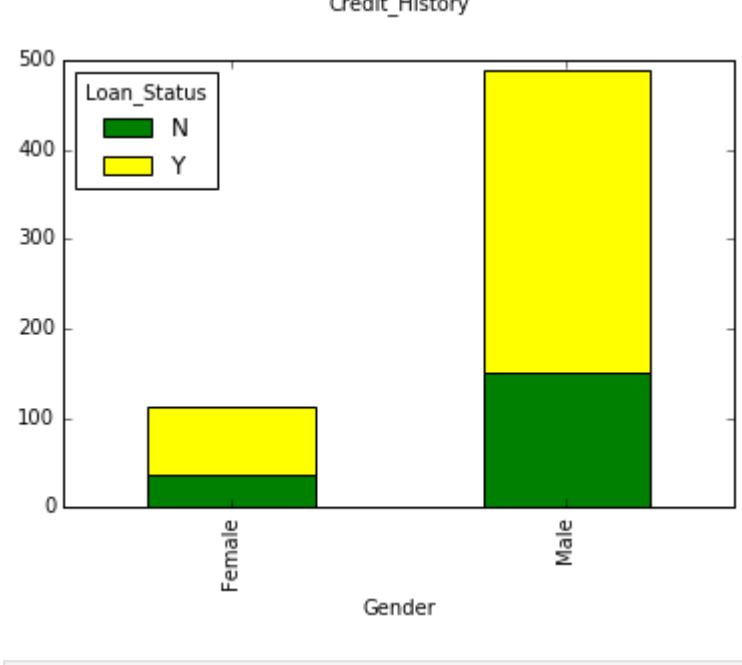
In [18]:
```
pv = df.pivot_table(values='Gender',index=['Credit_History'],aggfunc=lambda x: x.map({'Female':1,'Male':0}).mean())
pv
```

Out[18]:
```
Credit_History
0.0    0.197674
1.0    0.180258
Name: Gender, dtype: float64
```

In [19]:
```
#combining the above 2 plots in a stacked chart
stack_chart = pd.crosstab(df['Credit_History'], df['Loan_Status'])
stack_chart.plot(kind='bar', stacked=True, color=['green','yellow'], grid=False)
stack_chart = pd.crosstab(df['Gender'], df['Loan_Status'])
stack_chart.plot(kind='bar', stacked=True, color=['green','yellow'], grid=False)
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0xab17dd8>





In [20]:
```
df.apply(lambda x: sum(x.isnull()),axis=0) #to tell the missing values in each column
```

Out[20]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [21]:
```
#to fill the missing values by mean
df.LoanAmount = df.LoanAmount.fillna(df.LoanAmount.mean())
df.describe()
```

Out[21]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	84.037468	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.250000	360.00000	1.000000
50%	3812.500000	1188.500000	129.000000	360.00000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [22]:
```
se = df['Self_Employed'].value_counts(ascending=True)
se.head()
```

Out[22]:
```
Yes     82
No     500
Name: Self_Employed, dtype: int64
```

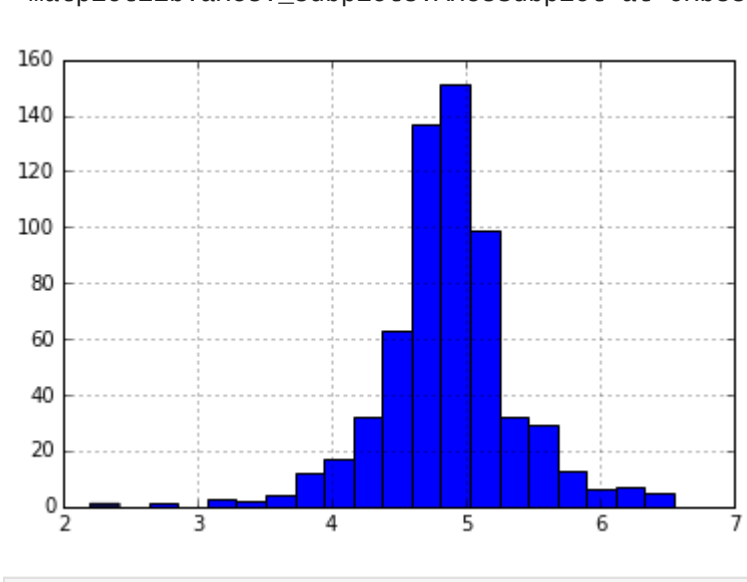
In [23]:
```
#since *no* has ~82% so, we will fill the missing values of this column with "no"
df.Self_Employed = df.Self_Employed.fillna('No')
df.describe()
```

Out[23]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	84.037468	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.250000	360.00000	1.000000
50%	3812.500000	1188.500000	129.000000	360.00000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

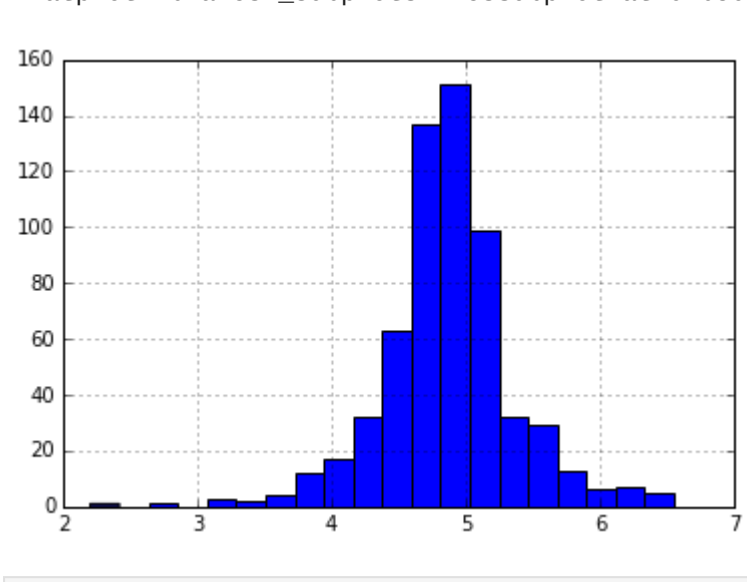
In [28]:
```
df['LoanAmount_log'] = np.log(df['LoanAmount']) #log function to reduce the extreme effect in Loan amount (which we saw in the previous barplot)
df['LoanAmount_log'].hist(bins=20)
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0xb3318d0>



In [29]:
```
#to decrease the extreme values of ApplicantIncome (which can be ssen from the previous barplot), we can add CoApplicantIncome to compensate the value
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df['TotalIncome_log'] = np.log(df['TotalIncome'])
df['LoanAmount_log'].hist(bins=20)
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0xb5049b0>



In [30]:
```
df['Capacity%'] = ((df['LoanAmount'] / df['TotalIncome']) *100).astype(float) #capacity of each applicant of how well he/she is suited to pay back his loan.
df.head(20)
```

Out[30]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
0	LP001002	Male	No	0	Graduate	No	5849	0.0	146.412162	360.0	1.0	Urban	Y	4.9
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.000000	360.0	1.0	Rural	N	4.8
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.000000	360.0	1.0	Urban	Y	4.1
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.000000	360.0	1.0	Urban	Y	4.7
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.000000	360.0	1.0	Urban	Y	4.9
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.000000	360.0	1.0	Urban	Y	5.5
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.000000	360.0	1.0	Urban	Y	4.5
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504.0	158.000000	360.0	0.0	Semiurban	N	5.0
8	LP001018	Male	Yes	2	Graduate	No	4006	1526.0	168.000000	360.0	1.0	Urban	Y	5.1
9	LP001020	Male	Yes	1	Graduate	No	12841	10968.0	349.000000	360.0	1.0	Semiurban	N	5.8
10	LP001024	Male	Yes	2	Graduate	No	3200	700.0	70.000000	360.0	1.0	Urban	Y	4.2
11	LP001027	Male	Yes	2	Graduate	No	2500	1840.0	109.000000	360.0	1.0	Urban	Y	4.6
12	LP001028	Male	Yes	2	Graduate	No	3073	8106.0	200.000000	360.0	1.0	Urban	Y	5.2
13	LP001029	Male	No	0	Graduate	No	1853	2840.0	114.000000	360.0	1.0	Rural	N	4.7
14	LP001030	Male	Yes	2	Graduate	No	1299	1086.0	17.000000	120.0	1.0	Urban	Y	2.8
15	LP001032	Male	No	0	Graduate	No	4950	0.0	125.000000	360.0	1.0	Urban	Y	4.8
16	LP001034	Male	No	1	Not Graduate	No	3596	0.0	100.000000	240.0	NaN	Urban	Y	4.6
17	LP001036	Female	No	0	Graduate	No	3510	0.0	76.000000	360.0	0.0	Urban	N	4.3
18	LP001038	Male	Yes	0	Not Graduate	No	4887	0.0	133.000000	360.0	1.0	Rural	N	4.8
19	LP001041	Male	Yes	0	Graduate	No	2600	3500.0	115.000000	NaN	1.0	Urban	Y	4.7

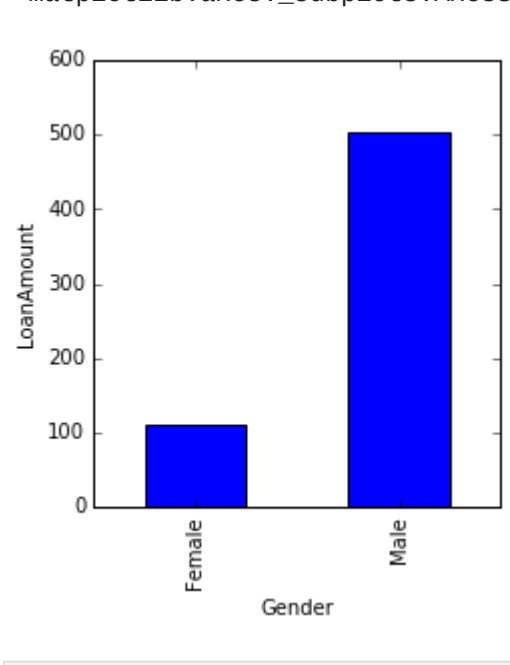
In [31]:
```
#males are around ~81%, so we'll fill the missing values with "male"
df.Gender = df.Gender.fillna('Male')
```

In [32]:
```
gen = df['Gender'].value_counts(ascending=True)
gen.head()
```

Out[32]:
```
Female    112
Male      502
Name: Gender, dtype: int64
```

In [33]:
```
#which gender has more loan amount
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)

ax1.set_xlabel('Gender')
ax1.set_ylabel('LoanAmount')

gen.plot(kind='bar')
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0xb3a1978>



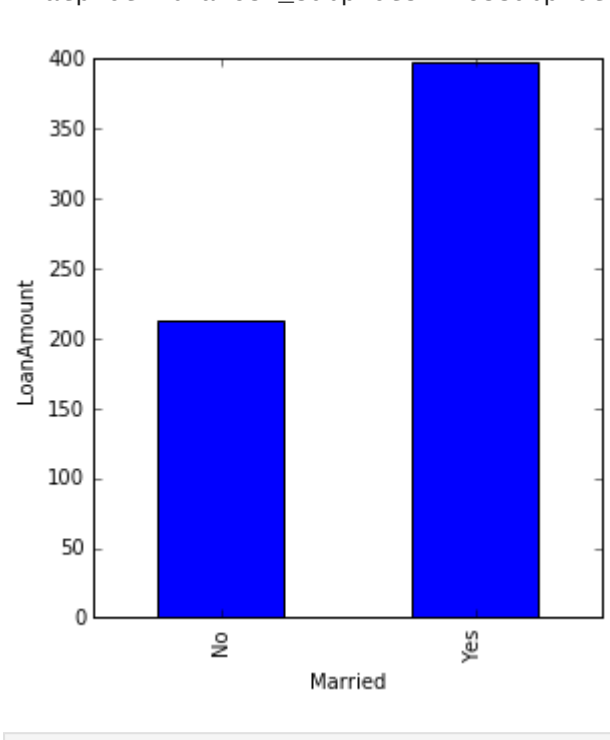
In [34]:
```
md = df['Married'].value_counts(ascending=True)
md.head()
```

Out[34]:
```
No     213
Yes    398
Name: Married, dtype: int64
```

In [35]:
```
#yes has ~65%
#filling the missing values with yes
df.Married = df.Married.fillna('Yes')
```

In [36]:
```
fig = plt.figure(figsize=(10,5))
ax1 = fig.add_subplot(121)

ax1.set_xlabel('Married')
ax1.set_ylabel('LoanAmount')

md.plot(kind='bar')
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0xab68668>



In [41]:
```
loan = df['Loan_Status'].value_counts(ascending=True)
loan.head()
```

Out[41]:
```
0    192
1    422
Name: Loan_Status, dtype: int64
```

In [43]:
```
df.Loan_Status = df.Loan_Status.fillna('1')
df.describe()
```

Out[43]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	600.00000	564.000000	614.000000	614.000000	614.0
mean	0.817590	0.653094	1.719870	0.218241	0.133550	5403.459283	1621.245798	146.412162	342.00000	0.842199	1.037459	0.687296	4.8
std	0.386497	0.476373	1.039056	0.413389	0.340446	6109.041673	2926.248369	84.037468	65.12041	0.364878	0.787482	0.463973	0.4
min	0.000000	0.000000	0.000000	0.000000	0.000000	150.000000	0.000000	9.000000	12.00000	0.000000	0.000000	0.000000	2.1
25%	1.000000	0.000000	1.000000	0.000000	0.000000	2877.500000	0.000000	100.250000	360.00000	1.000000	0.000000	0.000000	4.6
50%	1.000000	1.000000	1.000000	0.000000	0.000000	3812.500000	1188.500000	129.000000	360.00000	1.000000	1.000000	1.000000	4.8
75%	1.000000	1.000000	2.000000	0.000000	0.000000	5795.000000	2297.250000	164.750000	360.00000	1.000000	2.000000	1.000000	5.1
max	1.000000	1.000000	4.000000	1.000000	1.000000	81000.000000	41667.000000	700.000000	480.00000	1.000000	2.000000	1.000000	6.5

In [59]:
```
df.isnull().any()
```

Out[59]:
```
Loan_ID              False
Gender               False
Married              False
Dependents           False
Education            False
Self_Employed        False
ApplicantIncome      False
CoapplicantIncome    False
LoanAmount           False
Loan_Amount_Term      True
Credit_History        True
Property_Area        False
Loan_Status          False
LoanAmount_log       False
TotalIncome          False
TotalIncome_log      False
Capacity%            False
dtype: bool
```

In [60]:
```
ch = df['Credit_History'].value_counts(ascending=True)
ch.head()
```

Out[60]:
```
0.0     89
1.0    475
Name: Credit_History, dtype: int64
```

In [66]:
```
df.Credit_History = df.Credit_History.fillna('1.0')
df.head()
```

Out[66]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmoun
0	LP001002	1	0	1	0	0	5849	0.0	146.412162	360.0	1	2	1	4.98
1	LP001003	1	1	2	0	0	4583	1508.0	128.000000	360.0	1	0	0	4.85
2	LP001005	1	1	1	0	1	3000	0.0	66.000000	360.0	1	2	1	4.18
3	LP001006	1	1	1	1	0	2583	2358.0	120.000000	360.0	1	2	1	4.78
4	LP001008	1	0	1	0	0	6000	0.0	141.000000	360.0	1	2	1	4.94

In [67]:
```
#building predictive model
#we will use scikit-learn (sklearn), for which we need to convert the categorical values into numerical values
from sklearn.preprocessing import LabelEncoder
var_col = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area','Loan_Status']
le = LabelEncoder()  #Encode labels with value between 0 and n_classes-1.
for i in var_col:
    df[i] = le.fit_transform(df[i]) #Fit label encoder and return encoded labels
df.dtypes
```

Out[67]:
```
Loan_ID               object
Gender                 int64
Married                int64
Dependents             int64
Education              int64
Self_Employed          int64
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History        object
Property_Area          int64
Loan_Status            int64
LoanAmount_log       float64
TotalIncome          float64
TotalIncome_log      float64
Capacity%            float64
dtype: object
```

In [68]:
```
#Import models from scikit learn module:
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import KFold   #For K-fold cross validation
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics
```

In [69]:
```
#Generic function for making a classification model and accessing performance:
def classification_model(model, data, predictors, outcome):
    model.fit(data[predictors],data[outcome])
    predictions = model.predict(data[predictors])
    accuracy = metrics.accuracy_score(predictions,data[outcome])
    print "Accuracy : %s" % "{0:.3%}".format(accuracy)

    kf = KFold(data.shape[0], n_folds=5)
    error = []

    for train, test in kf:
        train_predictors = (data[predictors].iloc[train,:])
        train_target = data[outcome].iloc[train]
        model.fit(train_predictors, train_target)
        error.append(model.score(data[predictors].iloc[test,:], data[outcome].iloc[test]))

    print "Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error))
    #Fit the model again so that it can be refered outside the function:
    model.fit(data[predictors],data[outcome])
```

In [70]:
```
#using Logistic reasoning
outcome_var = ['Loan_Status']
model = LogisticRegression()
predictor_var = ['Credit_History']
classification_model(model, df,predictor_var,outcome_var)
```

```
C:\Users\payel\Anaconda2\lib\site-packages\sklearn\utils\validation.py:526: DataConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy : 80.945%
Cross-Validation Score : 80.946%
```

In [72]:
```
#using Logistic reasoning
outcome_var = ['Loan_Status']
model = LogisticRegression()
predictor_var = ['Credit_History','Gender','Married','Education']
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 80.945%
Cross-Validation Score : 80.946%
```

In [75]:
```
#using Decision Tree
model = DecisionTreeClassifier()
predictor_var = ['Credit_History','Gender','Married','Education']
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 80.945%
Cross-Validation Score : 80.946%
```

In [76]:
```
model = DecisionTreeClassifier()
predictor_var = ['Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Credit_History', 'Property_Area',
        'LoanAmount_log','TotalIncome_log']
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
Cross-Validation Score : 69.383%
```

In [74]:
```
#using Random Forest
model = RandomForestClassifier(n_estimators=100)
predictor_var = ['Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Credit_History', 'Property_Area',
        'LoanAmount_log','TotalIncome_log']
classification_model(model, df,predictor_var,outcome_var)
```

```
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  app.launch_new_instance()
Accuracy : 100.000%
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:14: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
Cross-Validation Score : 77.857%
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:19: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
```

In [77]:
```
#using Random Forest
model = RandomForestClassifier(n_estimators=100)
predictor_var = ['Gender', 'Married', 'Dependents', 'Education']
classification_model(model, df,predictor_var,outcome_var)
```

```
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  app.launch_new_instance()
Accuracy : 69.544%
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:14: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
Cross-Validation Score : 66.939%
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:19: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
```

Name-Anjali.N

Project name-Prediction of Heart disease detection

In [ ]:
```
#
```

Import Required Libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import os
import warnings
warnings.filterwarnings('ignore')
```

Import dataset

In [ ]:
```
dataset = pd.read_csv("/content/drive/MyDrive/TCR Internship Project/heart.csv")
```

In [ ]:
```
dataset
```

Out[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

Shape of dataset

In [ ]:
```
dataset.shape
```

Out[ ]:
```
(303, 14)
```

Some Operations on dataset

In [ ]:
```
dataset.head()
```

Out[ ]:

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

In [ ]:
```
dataset.tail()
```

Out[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

In [ ]:
```
type(dataset)
```

Out[ ]:
```
pandas.core.frame.DataFrame
```

In [ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [ ]:
```
dataset.describe()
```

Out[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	30
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	

In [ ]:
```
dataset.columns
```

Out[ ]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

Checking total number of NA values

In [ ]:
```
dataset.isna().sum()
```

Out[ ]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
```

```
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Checking total number of NULL values

In [ ]:
```
dataset.isnull().sum()
```

Out[ ]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [ ]:
```
#
```

Exploratory Data Analysis (EDA)

Analysing the 'target' variable

In [ ]:
```
dataset.target.describe()
```

Out[ ]:
```
count    303.000000
mean       0.544554
std        0.498835
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        1.000000
Name: target, dtype: float64
```

In [ ]:
```
dataset.target.unique()
```

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

In [ ]:
```
#Checking correlation between columns
dataset.corr()["target"].abs().sort_values(ascending=False)
```

Out[ ]:
```
target      1.000000
exang       0.436757
cp          0.433798
oldpeak     0.430696
thalach     0.421741
ca          0.391724
slope       0.345877
thal        0.344029
sex         0.280937
age         0.225439
trestbps    0.144931
restecg     0.137230
chol        0.085239
fbs         0.028046
Name: target, dtype: float64
```

In [ ]:
```
#This shows that most columns are moderately correlated with target, but 'fbs' is very weakly correlated.
```

In [ ]:
```
dataset.target.value_counts()
```

Out[ ]:
```
1    165
0    138
Name: target, dtype: int64
```

Patient without heart problems - labeled as 0

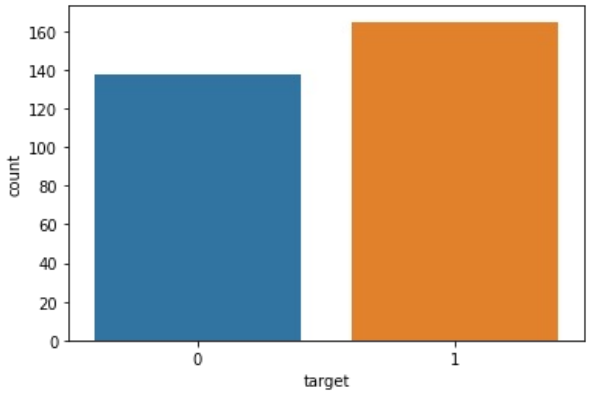
Patient with heart problems - labeled as 1

In [ ]:
```
print("Percentage of patients without heart problems: "+str(round(138*100/303,2)))
print("Percentage of patients with heart problems: "+str(round(165*100/303,2)))
```

```
Percentage of patient without heart problems: 45.54
Percentage of patient with heart problems: 54.46
```

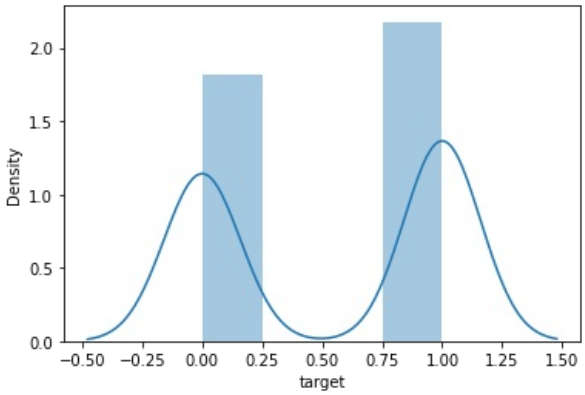
In [ ]:
```
y = dataset["target"]
sns.countplot(y)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a51e5d90>
```



In [ ]:
```
sns.distplot(dataset['target'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991c58910>
```



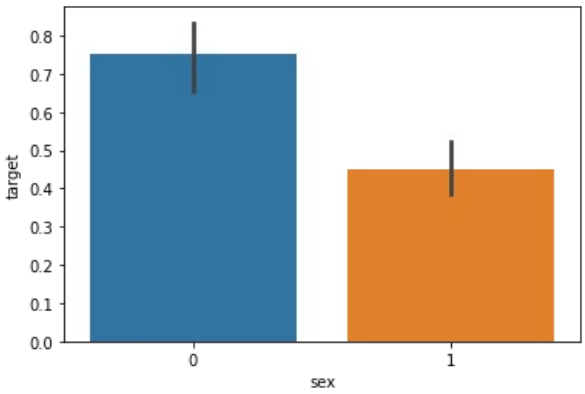
Analysing the 'sex' variable

In [ ]:
```
dataset.sex.value_counts()
```

Out[ ]:
```
1    207
0     96
Name: sex, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["sex"],y)
```

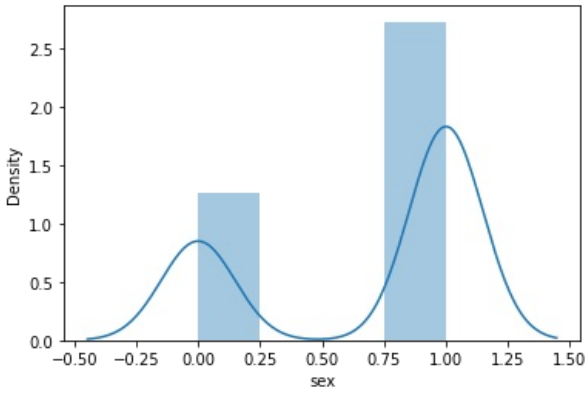
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a5127550>
```



We notice that the 'sex' feature has 2 unique features.

In [ ]:
```
sns.distplot(dataset['sex'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991cd29d0>
```



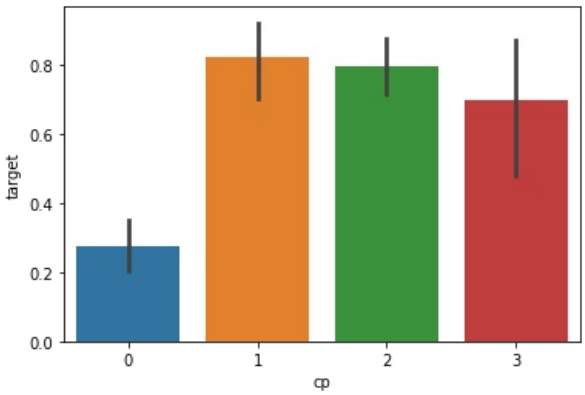
Analysing the 'cp' variable

In [ ]:
```
dataset.cp.value_counts()
```

Out[ ]:
```
0    143
2     87
1     50
3     23
Name: cp, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["cp"],y)
```

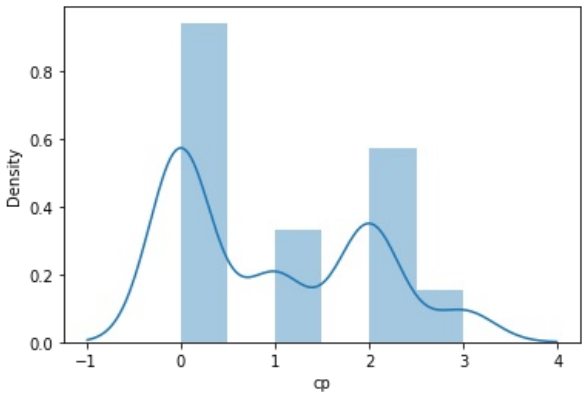
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a4c5a810>
```



The CP feature has values from 0 to 3.We notice, that chest pain of '0', are much less likely to have heart problems

In [ ]:
```
sns.distplot(dataset['cp'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991d89c90>
```



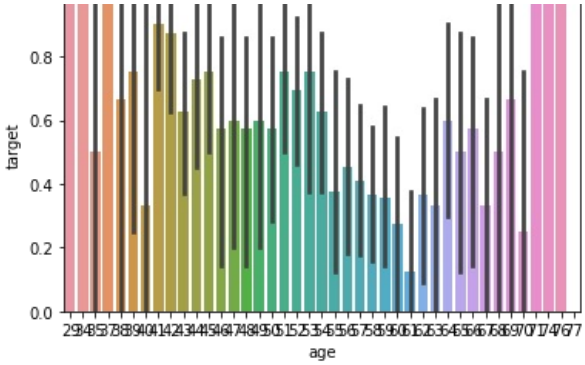
Analysing the 'age' variable

In [ ]:
```
dataset.age.value_counts()
```

Out[ ]:
```
58    19
57    17
54    16
59    14
52    13
51    12
62    11
44    11
60    11
56    11
64    10
41    10
63     9
67     9
55     8
45     8
42     8
53     8
61     8
65     8
43     8
66     7
50     7
48     7
46     7
49     5
47     5
39     4
35     4
68     4
70     4
40     3
71     3
69     3
38     3
34     2
37     2
77     1
76     1
74     1
29     1
Name: age, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["age"],y)
```

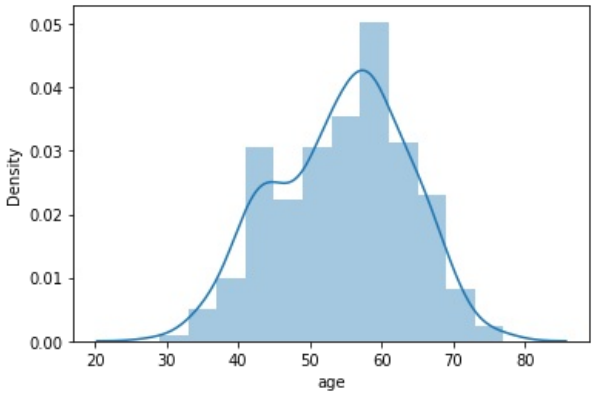
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a4bd5a90>
```



Nothing special here.

In [ ]:
```
sns.distplot(dataset['age'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991d66b90>
```



Analysing the 'trestbps' variable

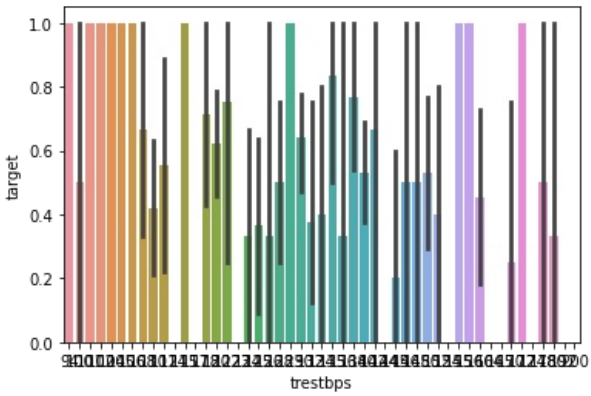
In [ ]:
```
dataset.trestbps.value_counts()
```

Out[ ]:
```
120    37
130    36
140    32
110    19
150    17
138    13
128    12
125    11
160    11
112     9
132     8
118     7
135     6
108     6
124     6
145     5
134     5
152     5
122     4
170     4
100     4
142     3
115     3
136     3
105     3
180     3
126     3
102     2
94      2
144     2
178     2
146     2
148     2
129     1
165     1
101     1
174     1
104     1
```

```
172     1
106     1
156     1
164     1
192     1
114     1
155     1
117     1
154     1
123     1
200     1
Name: trestbps, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["trestbps"],y)
```

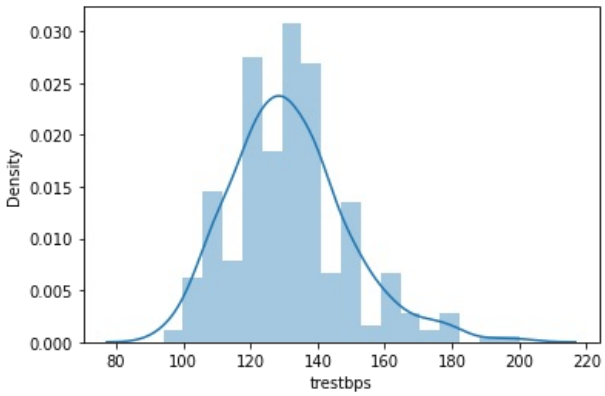
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a4bbb850>
```



Nothing special here.

In [ ]:
```
sns.distplot(dataset['trestbps'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991bd31d0>
```



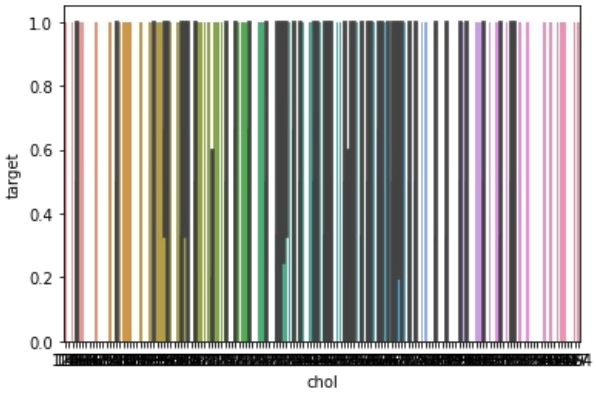
Analysing the 'chol' variable

In [ ]:
```
dataset.chol.value_counts()
```

Out[ ]:
```
234    6
204    6
197    6
269    5
212    5
      ..
278    1
281    1
284    1
290    1
564    1
Name: chol, Length: 152, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["chol"],y)
```

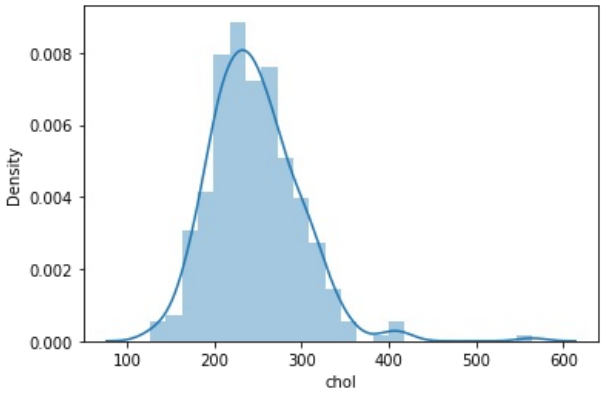
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a4736d10>
```



Nothing special here

In [ ]:
```
sns.distplot(dataset['chol'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991b3ba10>
```



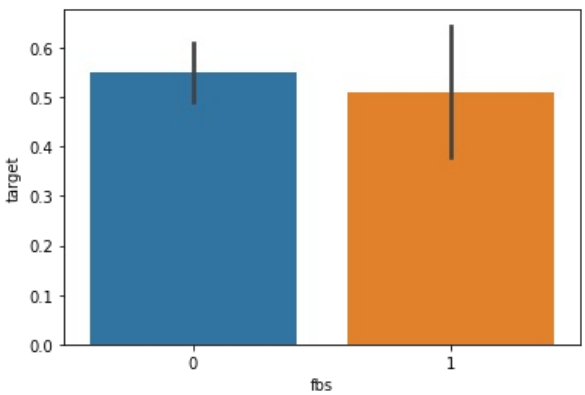
Analysing the 'fbs' variable

In [ ]:
```
dataset.fbs.value_counts()
```

Out[ ]:
```
0    258
1     45
Name: fbs, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["fbs"],y)
```

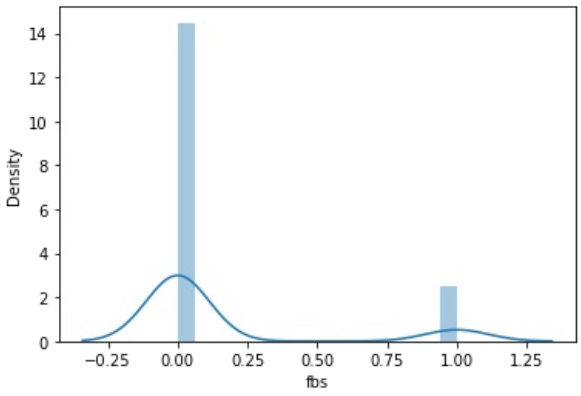
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a4738490>
```



Not much difference here.

In [ ]:
```
sns.distplot(dataset['fbs'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991aa1050>
```



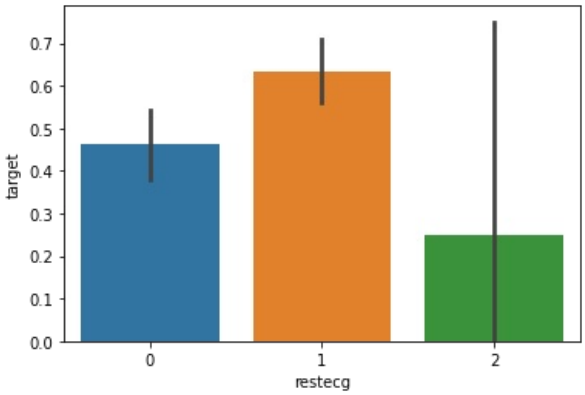
Analysing the 'restecg' variable

In [ ]:
```
dataset.restecg.value_counts()
```

Out[ ]:
```
1    152
0    147
2      4
Name: restecg, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["restecg"],y)
```

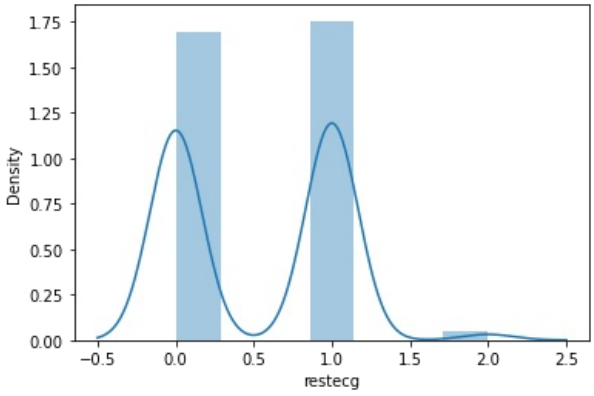
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a41a4f10>
```



We realize that people with restecg '1' and '0' are much more likely to have a heart disease than with restecg '2'

In [ ]:
```
sns.distplot(dataset['restecg'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89919bc990>
```



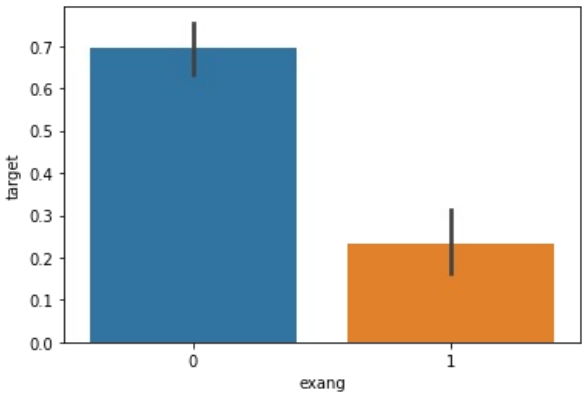
Analysing the 'exang' variable

In [ ]:
```
dataset.exang.value_counts()
```

Out[ ]:
```
0    204
1     99
Name: exang, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["exang"],y)
```

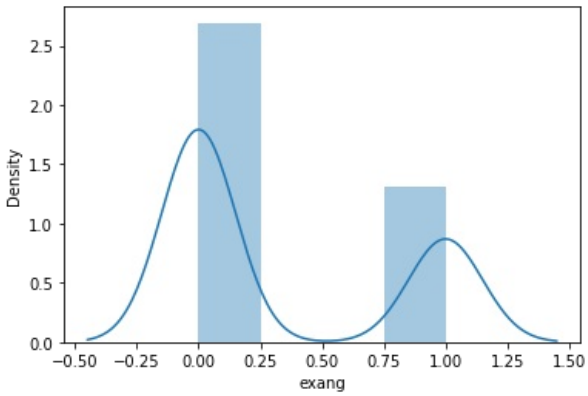
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a4120e90>
```



We notice here that people with exang=1, are much less likely to have heart problems.

In [ ]:
```
sns.distplot(dataset['exang'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991934290>
```



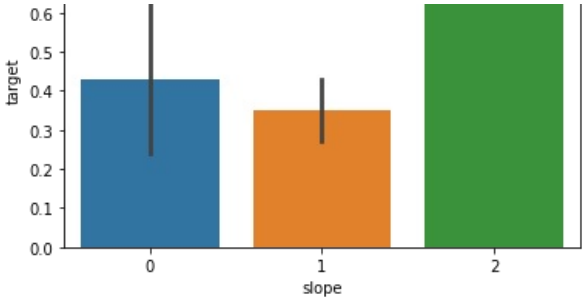
Analysing the 'slope' variable

In [ ]:
```
dataset.slope.value_counts()
```

Out[ ]:
```
2    142
1    140
0     21
Name: slope, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["slope"],y)
```

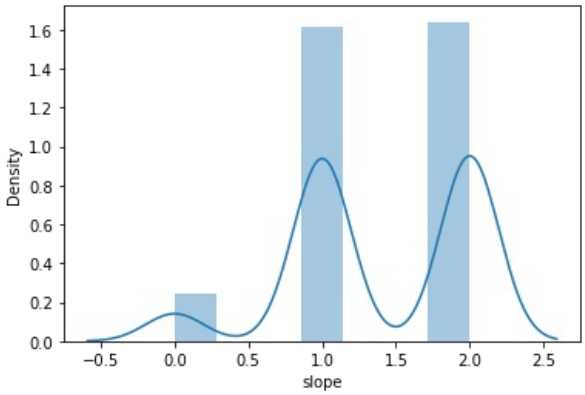
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a40e8b50>
```



We observe, that Slope '2' causes heart pain much more than Slope '0' and '1'

In [ ]:
```
sns.distplot(dataset['slope'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8991915b90>
```



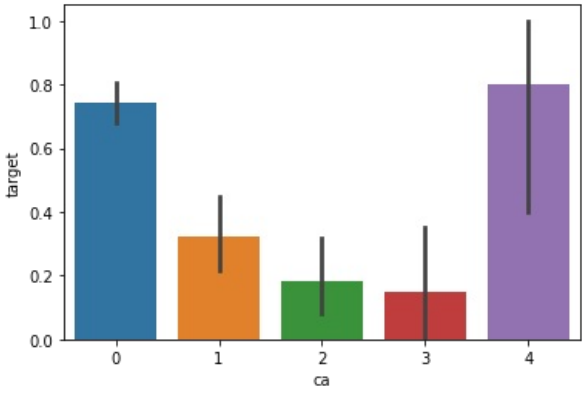
Analysing the 'ca' variable

In [ ]:
```
dataset.ca.value_counts()
```

Out[ ]:
```
0    175
1     65
2     38
3     20
4      5
Name: ca, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["ca"],y)
```

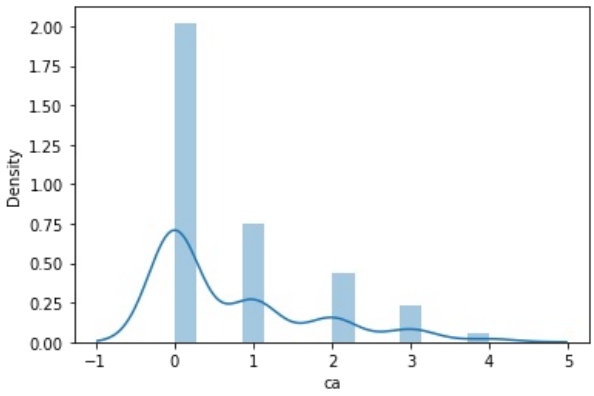
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89a406fa90>
```



We notice that ca=4 has large number of heart patients.

In [ ]:
```
sns.distplot(dataset['ca'])
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f89918781d0>



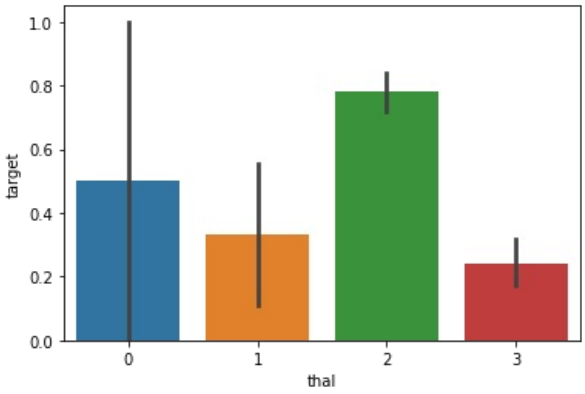
Analysing the 'thal' variable

In [ ]:
```
dataset.thal.value_counts()
```

Out[ ]:
```
2    166
3    117
1     18
0      2
Name: thal, dtype: int64
```

In [ ]:
```
sns.barplot(dataset["thal"],y)
```

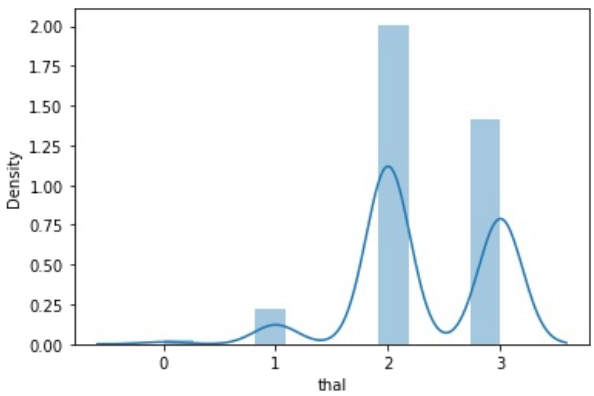
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f89a3fd8450>



thal=2 has large number of heart patients.

In [ ]:
```
sns.distplot(dataset['thal'])
```

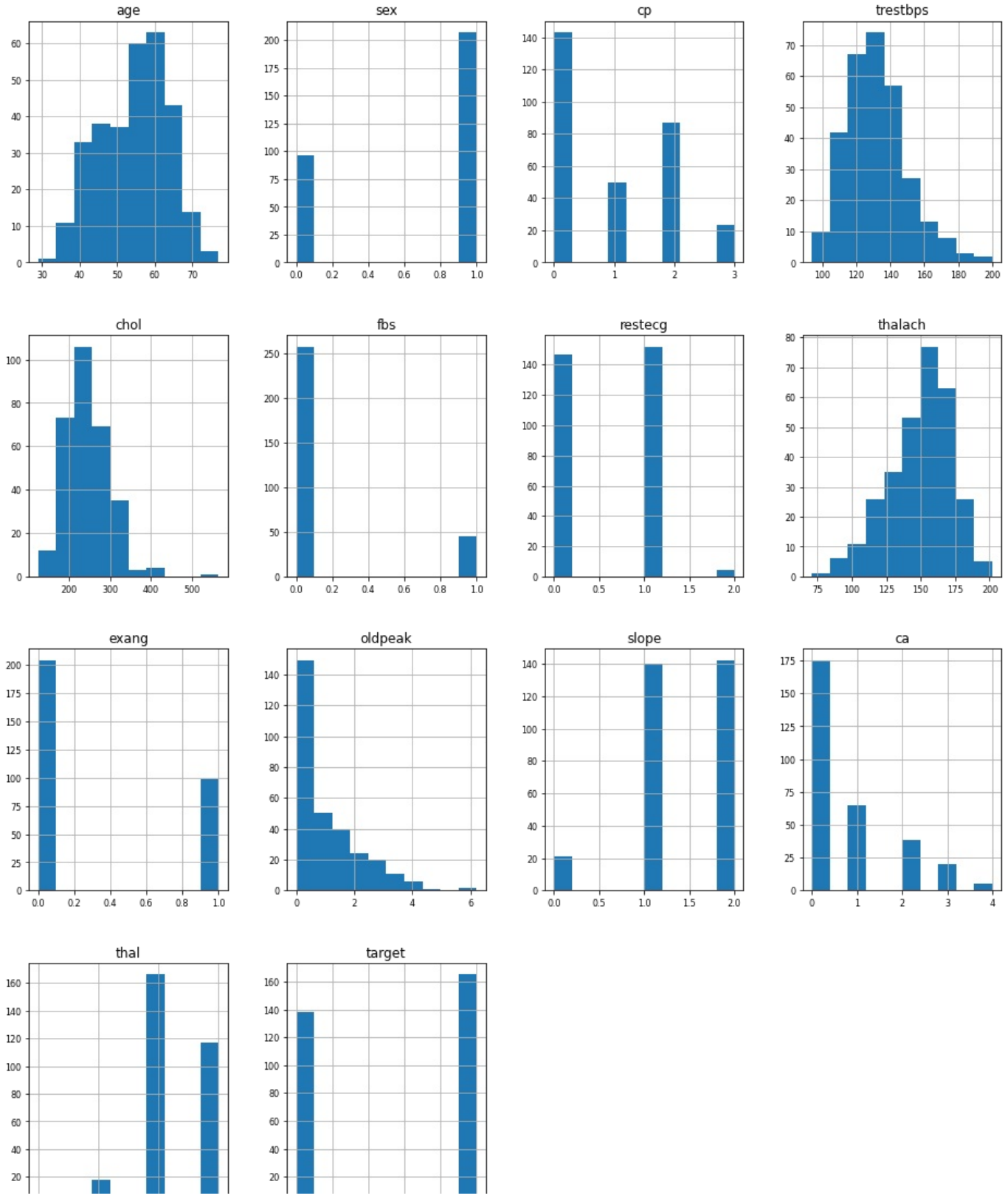
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8991835f50>



Get an overview distribution of each column

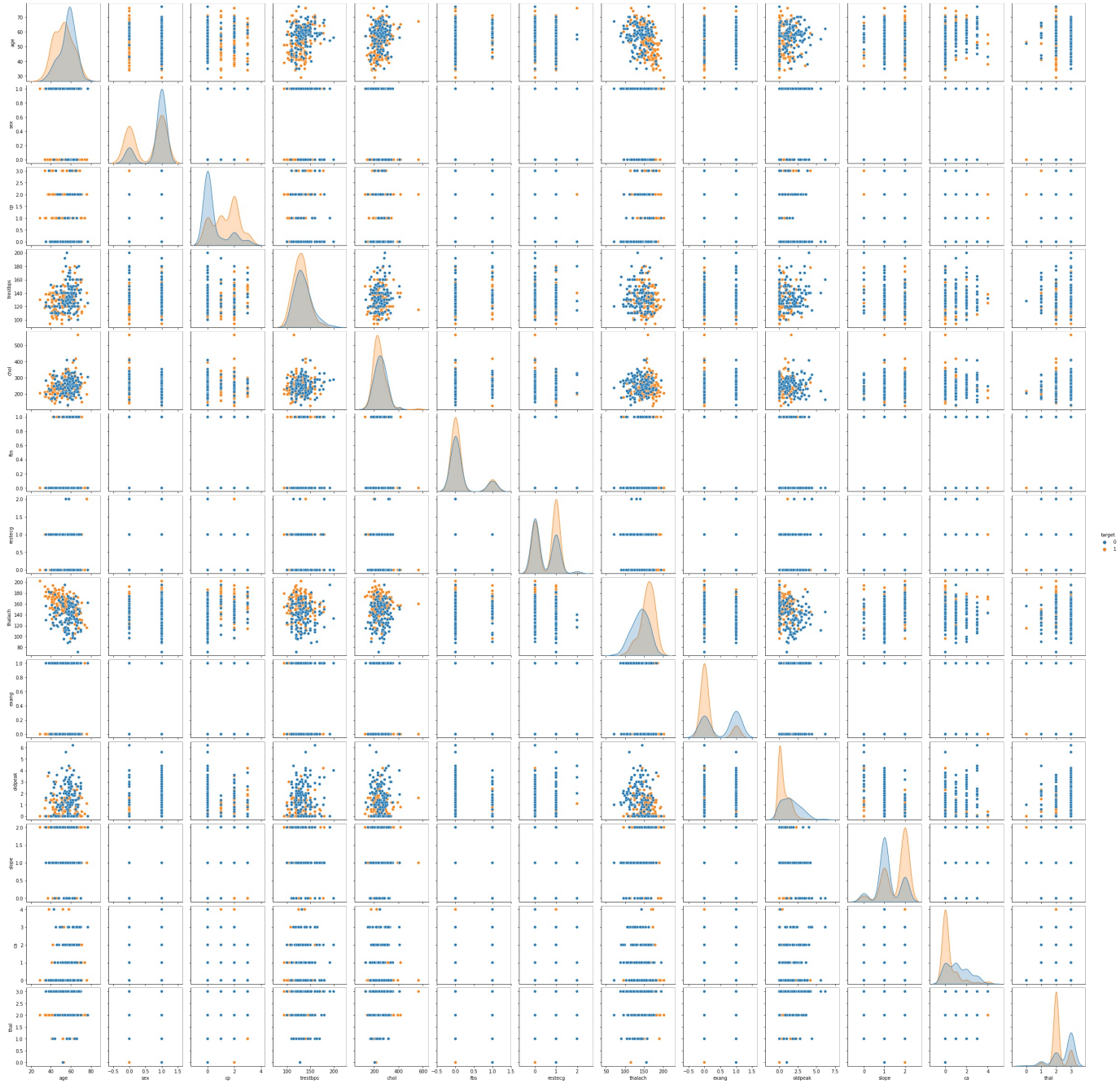
In [ ]:
```
dataset.hist(figsize=(16, 20), xlabelsize=8, ylabelsize=8)
```

Out[ ]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3f11d10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3ecd390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3e83990>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3e39f90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3dfe2d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3db57d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3de9d50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3dab1d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3dab210>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3d63810>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3cdb150>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3c92650>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3c46b10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3bf2b90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3bc0590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f89a3b75a90>]],
      dtype=object)
```



In [ ]:
```
sns.pairplot(dataset, hue='target')
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f89a512fbd0>
```



Correlation heatmap

In [ ]:
```
dataset.corr()
```

Out[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	t
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	0.068
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	0.210
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	-0.161
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	0.062
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	0.098
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	-0.032
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	-0.011
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	-0.096
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	0.206

oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682	0.210
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155	-0.104
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000	0.151
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832	1.000
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391724	-0.344

In [ ]:
```
f, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(dataset.corr(),annot=True,cmap='PiYG',linewidths=.5)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89921d6f10>
```

Splitting the data - Train Test split

In [ ]:
```
from sklearn.model_selection import train_test_split
x = dataset.drop("target",axis=1)
y= dataset["target"]

X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=0.20,random_state=0)
```

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

Out[ ]:
```
(242, 13)
```

In [ ]:
```
X_test.shape
```

Out[ ]:
```
(61, 13)
```

In [ ]:
```
Y_train.shape
```

Out[ ]:
```
(242,)
```

In [ ]:
```
Y_test.shape
```

Out[ ]:
```
(61,)
```

In [ ]:
```
from sklearn.metrics import accuracy_score
```

Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
model_logistic_reg = LogisticRegression()
model_logistic_reg.fit(X_train,Y_train)
Y_pred_logistic_reg = model_logistic_reg.predict(X_test)
```

In [ ]:
```
Y_pred_logistic_reg.shape
```

Out[ ]:
```
(61,)
```

In [ ]:
```
print("Predicted Values : ",Y_pred_logistic_reg)
```

```
Predicted Values :  [0 1 1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 1 1 1 1 0
 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1]
```

In [ ]:
```
Y_test[0:10] #You can check accuracy by observing predicted results and test data.
```

Out[ ]:
```
225    0
152    1
228    0
201    0
52     1
245    0
175    0
168    0
223    0
217    0
Name: target, dtype: int64
```

In [ ]:
```
accuracy_score_logistic_reg = round(accuracy_score(Y_pred_logistic_reg,Y_test)*100,2)
print("The accuracy score achieved using Logistic Regression is: "+str(accuracy_score_logistic_reg)+" %")
```

```
The accuracy score achieved using Logistic Regression is: 85.25 %
```

SVM

In [ ]:
```
from sklearn import svm
model_svm = svm.SVC(kernel='linear')
model_svm.fit(X_train, Y_train)
Y_pred_svm = model_svm.predict(X_test)
```

In [ ]:
```
Y_pred_svm.shape
```

Out[ ]:
```
(61,)
```

In [ ]:
```
print("Predicted Values : ",Y_pred_svm)
```

```
Predicted Values :  [0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 1 1 1 0
 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1]
```

In [ ]:
```
Y_test[0:10] #You can check accuracy by observing predicted results and test data.
```

Out[ ]:
```
225    0
```

```
152    1
228    0
201    0
52     1
245    0
175    0
168    0
223    0
217    0
Name: target, dtype: int64
```

In [ ]:
```
accuracy_score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)
print("The accuracy score achieved using Linear SVM is: "+str(accuracy_score_svm)+" %")
```

```
The accuracy score achieved using Linear SVM is: 81.97 %
```

K Nearest Neighbors

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
Y_pred_knn=knn.predict(X_test)
```

In [ ]:
```
Y_pred_knn.shape
```

Out[ ]:
```
(61,)
```

In [ ]:
```
print("Predicted Values : ",Y_pred_knn)
```

```
Predicted Values :  [0 0 1 0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0
 1 0 1 0 1 1 0 0 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0]
```

In [ ]:
```
Y_test[0:10] #You can check accuracy by observing predicted results and test data.
```

Out[ ]:
```
225    0
152    1
228    0
201    0
52     1
245    0
175    0
168    0
223    0
217    0
Name: target, dtype: int64
```

In [ ]:
```
accuracy_score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)
print("The accuracy score achieved using KNN is: "+str(accuracy_score_knn)+" %")
```

```
The accuracy score achieved using KNN is: 67.21 %
```

Decision Tree

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
max_accuracy = 0
for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,Y_train)
    Y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(X_train,Y_train)
```

```
Y_pred_dt = dt.predict(X_test)
```

In [ ]:
```
print(Y_pred_dt.shape)
```

```
(61,)
```

In [ ]:
```
print("Predicted Values : ",Y_pred_dt)
```

```
Predicted Values :  [0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 1 1 1 0
 1 0 0 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1]
```

In [ ]:
```
Y_test[0:10] #You can check accuracy by observing predicted results and test data.
```

Out[ ]:
```
225    0
152    1
228    0
201    0
52     1
245    0
175    0
168    0
223    0
217    0
Name: target, dtype: int64
```

In [ ]:
```
accuracy_score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
print("The accuracy score achieved using Decision Tree is: "+str(accuracy_score_dt)+" %")
```

```
The accuracy score achieved using Decision Tree is: 81.97 %
```

Random Forest

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
max_accuracy = 0
for x in range(2000):
    rf = RandomForestClassifier(random_state=x)
    rf.fit(X_train,Y_train)
    Y_pred_rf = rf.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

rf = RandomForestClassifier(random_state=best_x)
rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)
```

In [ ]:
```
Y_pred_rf.shape
```

Out[ ]:
```
(61,)
```

In [ ]:
```
print("Predicted Values : ",Y_pred_rf)
```

```
Predicted Values :  [0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 1 1 1 0 0
 1 0 0 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1]
```

In [ ]:
```
Y_test[0:10] #You can check accuracy by observing predicted results and test data.
```

Out[ ]:
```
225    0
152    1
228    0
201    0
52     1
245    0
175    0
```

```
168    0
223    0
217    0
Name: target, dtype: int64
```

In [ ]:
```
accuracy_score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
print("The accuracy score achieved using Random Forest is: "+str(accuracy_score_rf)+" %")
```

```
The accuracy score achieved using Random Forest is: 90.16 %
```

Summary of accuracy scores

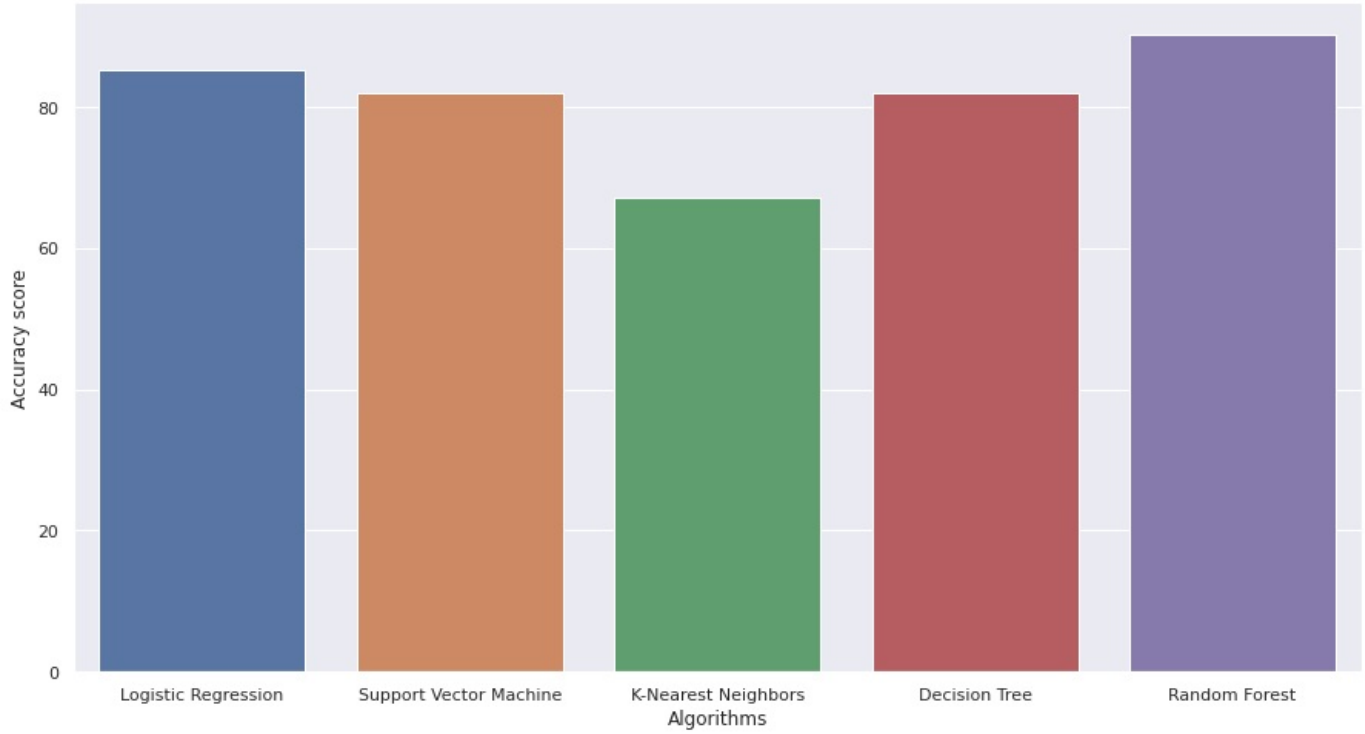
In [ ]:
```
all_accuracy_scores = [accuracy_score_logistic_reg,accuracy_score_svm,accuracy_score_knn,accuracy_score_dt,accura
algorithms_used = ["Logistic Regression","Support Vector Machine","K-Nearest Neighbors","Decision Tree","Random F
```

In [ ]:
```
for i in range(len(algorithms_used)):
    print("\nThe accuracy score achieved using "+algorithms_used[i]+" is: "+str(all_accuracy_scores[i])+" %")
```

```
The accuracy score achieved using Logistic Regression is: 85.25 %

The accuracy score achieved using Support Vector Machine is: 81.97 %

The accuracy score achieved using K-Nearest Neighbors is: 67.21 %

The accuracy score achieved using Decision Tree is: 81.97 %

The accuracy score achieved using Random Forest is: 90.16 %
```

In [ ]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms_used,all_accuracy_scores)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f898ac16590>
```



Here we can see that Random Forest is better than other algorithms.

**Name - Anjali.N

**Project name - Breast cancer prediction

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

from sklearn.model_selection import train_test_split from sklearn.model_selection import KFold, cross_val_score, cross_val_predict from sklearn import metrics

from sklearn import svm from sklearn.linear_model import LogisticRegression from sklearn.ensemble import RandomForestClassifier from sklearn.neighbors import KNeighborsClassifier from sklearn.tree import DecisionTreeClassifier

Load Dataset

In [2]:
```
data = pd.read_csv('data.csv')

data.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0

5 rows × 33 columns

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
id                         569 non-null int64
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
Unnamed: 32                0 non-null float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Refine Dataset

In [4]:
```
data.drop(['id', 'Unnamed: 32'], axis=1, inplace=True)

'''
Benign: 양성 (0)
Malignant: 악성 (1)
'''
data['diagnosis'] = data['diagnosis'].map({'B': 0, 'M': 1})

data.head()
```

Out[4]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimet
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	

5 rows × 31 columns

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

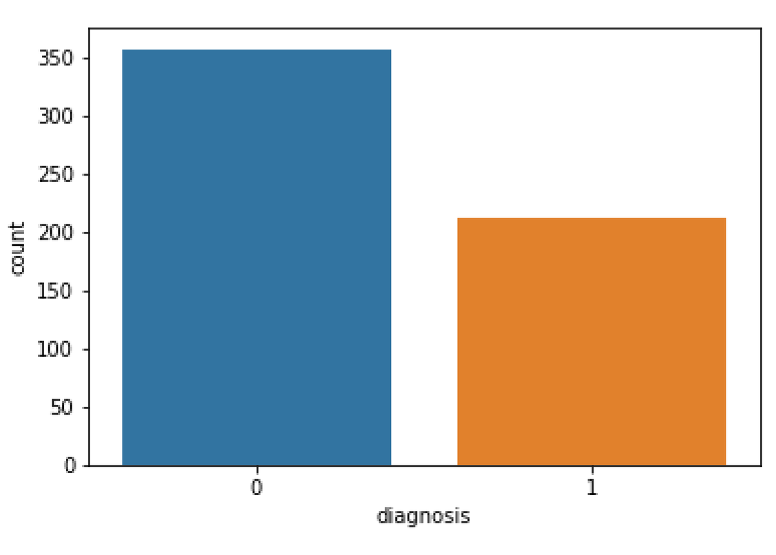
Out[5]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	...	16.269190	25.677223
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	...	4.833242	6.146258
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	...	7.930000	12.020000
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	...	13.010000	21.080000
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	...	14.970000	25.410000
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	...	18.790000	29.720000
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	...	36.040000	49.540000

8 rows × 31 columns

In [6]:
```
sns.countplot(data['diagnosis'])
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x122262240>
```



Split Train and Test

In [7]:
```
train, test = train_test_split(data, test_size=0.2, random_state=2019)

x_train = train.drop(['diagnosis'], axis=1)
y_train = train.diagnosis

x_test = test.drop(['diagnosis'], axis=1)
y_test = test.diagnosis

print(len(train), len(test))
```

```
455 114
```

SVM

In [8]:
```
model = svm.SVC(gamma='scale')
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print('SVM: %.2f' % (metrics.accuracy_score(y_pred, y_test) * 100))
```

```
SVM: 91.23
```

DecisionTreeClassifier

In [9]:
```
model = DecisionTreeClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print('DecisionTreeClassifier: %.2f' % (metrics.accuracy_score(y_pred, y_test) * 100))
```

```
DecisionTreeClassifier: 87.72
```

KNeighborsClassifier

In [10]:
```
model = KNeighborsClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print('KNeighborsClassifier: %.2f' % (metrics.accuracy_score(y_pred, y_test) * 100))
```

```
KNeighborsClassifier: 92.98
```

LogisticRegression

In [11]:
```
model = LogisticRegression(solver='lbfgs', max_iter=2000)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print('LogisticRegression: %.2f' % (metrics.accuracy_score(y_pred, y_test) * 100))
```

```
LogisticRegression: 94.74
```

RandomForestClassifier

In [12]:
```
model = RandomForestClassifier(n_estimators=100)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print('RandomForestClassifier: %.2f' % (metrics.accuracy_score(y_pred, y_test) * 100))
```

```
RandomForestClassifier: 95.61
```

Compute Feature Importances

In [13]:
```
features = pd.Series(
    model.feature_importances_,
    index=x_train.columns
).sort_values(ascending=False)

print(features)
```

```
perimeter_worst            0.149118
radius_worst               0.122345
area_worst                 0.115274
concave points_mean        0.111324
concave points_worst       0.110002
concavity_mean             0.054926
area_mean                  0.046717
concavity_worst            0.046431
area_se                    0.041391
perimeter_mean             0.034161
radius_mean                0.033442
perimeter_se               0.017655
smoothness_worst           0.014702
compactness_mean           0.010849
texture_worst              0.010581
compactness_worst          0.009898
symmetry_worst             0.009830
fractal_dimension_worst    0.009228
texture_mean               0.009160
concave points_se          0.006606
compactness_se             0.004309
smoothness_mean            0.004203
fractal_dimension_se       0.004196
smoothness_se              0.004004
concavity_se               0.003812
radius_se                  0.003742
texture_se                 0.003539
fractal_dimension_mean     0.003089
symmetry_mean              0.002861
symmetry_se                0.002605
dtype: float64
```

Extract Top 5 Features

In [14]:
```
top_5_features = features.keys()[:5]

print(top_5_features)
```

```
Index(['perimeter_worst', 'radius_worst', 'area_worst', 'concave points_mean',
       'concave points_worst'],
      dtype='object')
```

SVM (Top 5)

In [32]:
```
model = svm.SVC(gamma='scale')
model.fit(x_train[top_5_features], y_train)

y_pred = model.predict(x_test[top_5_features])

print('SVM(Top 5): %.2f' % (metrics.accuracy_score(y_pred, y_test) * 100))
```

```
RandomForestClassifier(Top 5): 93.81
```

Cross Validaiton (Tedious)

In [22]:
```
model = svm.SVC(gamma='scale')

cv = KFold(n_splits=5, random_state=2019)

accs, scores = [], []

for train_index, test_index in cv.split(data[top_5_features]):
    x_train = data.iloc[train_index][top_5_features]
    y_train = data.iloc[train_index].diagnosis

    x_test = data.iloc[test_index][top_5_features]
    y_test = data.iloc[test_index].diagnosis

    model.fit(x_train, y_train)

    y_pred = model.predict(x_test)

    accs.append(metrics.accuracy_score(y_test, y_pred))

print(accs)
```

```
[0.7807017543859649, 0.8947368421052632, 0.9736842105263158, 0.9298245614035088, 0.9380530973451328]
```

Cross Validation (Simple)

In [23]:
```
model = svm.SVC(gamma='scale')

cv = KFold(n_splits=5, random_state=2019)

accs = cross_val_score(model, data[top_5_features], data.diagnosis, cv=cv)

print(accs)
```

```
[0.78070175 0.89473684 0.97368421 0.92982456 0.9380531 ]
```

Test All Models

In [24]:
```
models = {
    'SVM': svm.SVC(gamma='scale'),
    'DecisionTreeClassifier': DecisionTreeClassifier(),
    'KNeighborsClassifier': KNeighborsClassifier(),
    'LogisticRegression': LogisticRegression(solver='lbfgs', max_iter=2000),
    'RandomForestClassifier': RandomForestClassifier(n_estimators=100)
}

cv = KFold(n_splits=5, random_state=2019)

for name, model in models.items():
    scores = cross_val_score(model, data[top_5_features], data.diagnosis, cv=cv)

    print('%s: %.2f%%' % (name, np.mean(scores) * 100))
```

```
SVM: 90.34%
DecisionTreeClassifier: 91.38%
KNeighborsClassifier: 88.40%
LogisticRegression: 90.69%
RandomForestClassifier: 93.33%
```

Normalize Dataset

In [25]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[top_5_features])

models = {
    'SVM': svm.SVC(gamma='scale'),
    'DecisionTreeClassifier': DecisionTreeClassifier(),
    'KNeighborsClassifier': KNeighborsClassifier(),
    'LogisticRegression': LogisticRegression(solver='lbfgs', max_iter=2000),
    'RandomForestClassifier': RandomForestClassifier(n_estimators=100)
}

cv = KFold(n_splits=5, random_state=2019)

for name, model in models.items():
    scores = cross_val_score(model, scaled_data, data.diagnosis, cv=cv)

    print('%s: %.2f%%' % (name, np.mean(scores) * 100))
```

```
SVM: 93.85%
DecisionTreeClassifier: 90.68%
KNeighborsClassifier: 93.15%
LogisticRegression: 93.85%
RandomForestClassifier: 93.15%
```