Problem: We will explore several methods to see if certain characteristics or features can be used to predict whether an applicant defaults or pays his loan. Then we will use machine learning technics to develop a model to predict credit default. To reach this goal, we will implement a 3 steps project:

- Step 1-Identify and handle missing values

Identify missing values

Evaluate the missing values

Deal with the missing values

Correct data format

Get indicator variables and assign them to dummy variables

Binning in order to categorise the loan value

- Step 2- Analyse Individual Feature Patterns using Visualization and stastical technicques
- Step 3- Develop the Model

The data come from : [http://www.creditriskanalytics.net/datasets-private2.html (http://www.creditriskanalytics.net/datasets-private2.html)](http://www.creditriskanalytics.net/datasets-private2.html) The data set HMEQ reports characteristics and delinquency information for 5,960 home equity loans. A home equity loan is a loan where the obligor uses the equity of his or her home as the underlying collateral.

1-Identify and handle missing values

_IMPORT ' Home Equity Data set' from the following link: [http://www.creditriskanalytics.net/uploads/1/9/5/1/19511601/hmeq.csv (http://www.creditriskanalytics.net/uploads/1/9/5/1/19511601/hmeq.csv)](http://www.creditriskanalytics.net/uploads/1/9/5/1/19511601/hmeq.csv).

In [700]:

```
import numpy as np
import pandas as pd
url = 'http://www.creditriskanalytics.net/uploads/1/9/5/1/19511601/hmeq.csv'
df = pd.read_csv(url)
df.head()
#df.shape
```

Out[700]:

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE
0	1	1100	25860.0	39025.0	HomeImp	Other	10.5	0.0	0.0	94.366667
1	1	1300	70053.0	68400.0	HomeImp	Other	7.0	0.0	2.0	121.833333
2	1	1500	13500.0	16700.0	HomeImp	Other	4.0	0.0	0.0	149.466667
3	1	1500	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	0	1700	97800.0	112000.0	HomeImp	Office	3.0	0.0	0.0	93.333333

In [701]:

```
#First lets only use numeric data
#df=df._get_numeric_data()
```

In [702]:

```
print(df.dtypes)
```

```
BAD           int64
LOAN          int64
MORTDUE     float64
VALUE       float64
REASON       object
JOB          object
YOJ         float64
DEROG       float64
DELINQ      float64
CLAGE       float64
NINQ        float64
CLNO        float64
DEBTINC     float64
dtype: object
```

Evaluating for Missing Data

The missing values are auto-converted to Python's default **NaN**. We use Python's built-in functions to identify these missing values by using *.isnull()* method to detect missing data:

The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data.

In [703]:

```
missing_data = df.isnull()
missing_data.head()
```

Out[703]:

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINC
0	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False
3	False	False	True	True	True	True	True	True	True	True	True
4	False	False	False	False	False	False	False	False	False	False	False

In [704]:

```
# Identy the columns with missing values.

for column in missing_data.columns.values.tolist():
    print(column)
    #print (missing_data[column].value_counts())
    #print("")
```

```
BAD
LOAN
MORTDUE
VALUE
REASON
JOB
YOJ
DEROG
DELINQ
CLAGE
NINQ
CLNO
DEBTINC
```

In [705]:

```
for column in missing_data.columns.values.tolist():
    print (missing_data[column].value_counts())
    print("")
```

```
False    5960
Name: BAD, dtype: int64

False    5960
Name: LOAN, dtype: int64

False    5442
True      518
Name: MORTDUE, dtype: int64

False    5848
True      112
Name: VALUE, dtype: int64

False    5708
True      252
Name: REASON, dtype: int64

False    5681
True      279
Name: JOB, dtype: int64

False    5445
True      515
Name: YOJ, dtype: int64

False    5252
True      708
Name: DEROG, dtype: int64

False    5380
True      580
Name: DELINQ, dtype: int64

False    5652
True      308
Name: CLAGE, dtype: int64

False    5450
True      510
Name: NINQ, dtype: int64

False    5738
True      222
Name: CLNO, dtype: int64

False    4693
True     1267
Name: DEBTINC, dtype: int64
```

In [706]:

```
df.describe(include =['object'])
```

Out[706]:

	REASON	JOB
count	5708	5681
unique	2	6
top	DebtCon	Other
freq	3928	2388

Current sample is 5960. Only columns " BAD" and "LOAN" do not have missing values.

The next step now is to find a method to handle these missing values for the following features: MORTDUE,VALUE,REASON,JOB,YOJ,DEROG,DELINQ,CLAGE,NINQ,CLNO, DEBTINC.

Dealing with missing values : Several options are available. a) no rows or columns can be dropped as they do not have enough empty values. b) The more reasonable options to deal with these missings values are to replace them by either mean or frequency.

There are 2 categorical variables in the data which are "REASON" and "JOB". We will use the frequency to replace the missing values. The obove table shows the most frequent values for "REASON" and "JOB"are repectively "DebtCon" and "Other". This can be observed as well by using the "value_counts" method below. Hence we will replace the missing values by "DebtCon" and "Other".

In [707]:

```
#Number of missing values in Columns 'REASON' and 'JOB'

print('Number of NAN values for column REASON :', df["REASON"].isnull().sum())
print('Number of NAN values for column JOB :', df["JOB"].isnull().sum())
```

```
Number of NAN values for column REASON : 252
Number of NAN values for column JOB : 279
```

In [708]:

```
df['REASON'].value_counts()
```

Out[708]:

```
DebtCon    3928
HomeImp    1780
Name: REASON, dtype: int64
```

In [709]:

```
df['REASON'].value_counts().idxmax()
```

Out[709]:

```
'DebtCon'
```

In [710]:

```
df['JOB'].value_counts()
```

Out[710]:

```
Other      2388
ProfExe    1276
Office      948
Mgr         767
Self        193
Sales       109
Name: JOB, dtype: int64
```

In [711]:

```
df['JOB'].value_counts().idxmax()
```

Out[711]:

```
'Other'
```

In [712]:

```
df["REASON"].replace(np.nan, 'DebtCon', inplace=True)
df["JOB"].replace(np.nan, 'Other', inplace=True)
df.describe(include =['object'])
```

Out[712]:

	REASON	JOB
count	5960	5960
unique	2	6
top	DebtCon	Other
freq	4180	2667

In [713]:

```
print('Number of NAN values for column REASON :', df["REASON"].isnull().sum())
print('Number of NAN values for column JOB :', df["JOB"].isnull().sum())
```

```
Number of NAN values for column REASON : 0
Number of NAN values for column JOB : 0
```

In [714]:

```
df.head()
```

Out[714]:

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE
0	1	1100	25860.0	39025.0	HomeImp	Other	10.5	0.0	0.0	94.366667
1	1	1300	70053.0	68400.0	HomeImp	Other	7.0	0.0	2.0	121.833333
2	1	1500	13500.0	16700.0	HomeImp	Other	4.0	0.0	0.0	149.466667
3	1	1500	NaN	NaN	DebtCon	Other	NaN	NaN	NaN	NaN
4	0	1700	97800.0	112000.0	HomeImp	Office	3.0	0.0	0.0	93.333333

The next step is to replace the missing values in the remaining variables by their mean. These variables are : MORTDUE,VALUE,YOJ,DEROG,DELINQ,CLAGE,NINQ,CLNO, DEBTINC.

In [715]:

```
# Computing the mean for each remaining variable and replacing NaN by mean value in eac
h column
avg_MORTDUE = df["MORTDUE"].astype("float").mean(axis=0)
df['MORTDUE'].replace(np.nan, avg_MORTDUE, inplace = True)

avg_VALUE = df["VALUE"].astype("float").mean(axis=0)
df['VALUE'].replace(np.nan, avg_VALUE, inplace = True)

avg_YOJ = df["YOJ"].astype("float").mean(axis=0)
df['YOJ'].replace(np.nan, avg_YOJ, inplace = True)

avg_DEROG = df["DEROG"].astype("float").mean(axis=0)
df["DEROG"].replace(np.nan, avg_DEROG, inplace = True)

avg_DELINQ = df["DELINQ"].astype("float").mean(axis=0)
df["DELINQ"].replace(np.nan, avg_DELINQ, inplace = True)

avg_CLAGE = df["CLAGE"].astype("float").mean(axis=0)
df["CLAGE"].replace(np.nan, avg_CLAGE, inplace = True)

avg_NINQ = df["NINQ"].astype("float").mean(axis=0)
df["NINQ"].replace(np.nan, avg_NINQ, inplace = True)

avg_CLNO = df["CLNO"].astype("float").mean(axis=0)
df["CLNO"].replace(np.nan, avg_CLNO, inplace = True)

avg_DEBTINC = df["DEBTINC"].astype("float").mean(axis=0)
df["DEBTINC"].replace(np.nan, avg_DEBTINC, inplace = True)

print("avg_MORTDUE =", avg_MORTDUE)
print("avg_VALUE =", avg_VALUE)
print("avg_YOJ =", avg_YOJ)
print("avg_DEROG =", avg_DEROG)
print("avg_DELINQ =", avg_DELINQ)
print("avg_CLAGE =", avg_CLAGE)
print("avg_NINQ =", avg_NINQ)
print("avg_CLNO =", avg_CLNO)
print("avg_DEBTINC =", avg_DEBTINC)
```

```
avg_MORTDUE = 73760.817199559
avg_VALUE = 101776.04874145007
avg_YOJ = 8.922268135904499
avg_DEROG = 0.2545696877380046
avg_DELINQ = 0.4494423791821561
avg_CLAGE = 179.76627518656605
avg_NINQ = 1.1860550458715597
avg_CLNO = 21.29609620076682
avg_DEBTINC = 33.77991534872112
```

In [716]:

```
df.describe()
```

Out[716]:

	BAD	LOAN	MORTDUE	VALUE	YOJ	DEROG
count	5960.000000	5960.000000	5960.000000	5960.000000	5960.000000	5960.000000
mean	0.199497	18607.969799	73760.817200	101776.048741	8.922268	0.254570
std	0.399656	11207.480417	42481.395689	56843.931566	7.239301	0.794198
min	0.000000	1100.000000	2063.000000	8000.000000	0.000000	0.000000
25%	0.000000	11100.000000	48139.000000	66489.500000	3.000000	0.000000
50%	0.000000	16300.000000	69529.000000	90000.000000	8.000000	0.000000
75%	0.000000	23300.000000	88200.250000	119004.750000	12.000000	0.000000
max	1.000000	89900.000000	399550.000000	855909.000000	41.000000	10.000000

Check the dataset in order to ensure there are no missing data

In [717]:

```
df.head()
```

Out[717]:

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ
0	1	1100	25860.0000	39025.000000	HomeImp	Other	10.500000	0.00000	0.000000
1	1	1300	70053.0000	68400.000000	HomeImp	Other	7.000000	0.00000	2.000000
2	1	1500	13500.0000	16700.000000	HomeImp	Other	4.000000	0.00000	0.000000
3	1	1500	73760.8172	101776.048741	DebtCon	Other	8.922268	0.25457	0.449442
4	0	1700	97800.0000	112000.000000	HomeImp	Office	3.000000	0.00000	0.000000

In [718]:

```
No_missing_data = df.isnull()
No_missing_data.head()
```

Out[718]:

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINC
0	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False

In [719]:

```
for column in No_missing_data.columns.values.tolist():
    print (No_missing_data[column].value_counts())
    print("")
```

```
False    5960
Name: BAD, dtype: int64

False    5960
Name: LOAN, dtype: int64

False    5960
Name: MORTDUE, dtype: int64

False    5960
Name: VALUE, dtype: int64

False    5960
Name: REASON, dtype: int64

False    5960
Name: JOB, dtype: int64

False    5960
Name: YOJ, dtype: int64

False    5960
Name: DEROG, dtype: int64

False    5960
Name: DELINQ, dtype: int64

False    5960
Name: CLAGE, dtype: int64

False    5960
Name: NINQ, dtype: int64

False    5960
Name: CLNO, dtype: int64

False    5960
Name: DEBTINC, dtype: int64
```

Get indicator variables and assign it to data frame "dummy_variables

In [720]:

```
df.columns
```

Out[720]:

```
Index(['BAD', 'LOAN', 'MORTDUE', 'VALUE', 'REASON', 'JOB', 'YOJ', 'DEROG',
       'DELINQ', 'CLAGE', 'NINQ', 'CLNO', 'DEBTINC'],
      dtype='object')
```

In [721]:

```
df['REASON'].value_counts()
```

Out[721]:

```
DebtCon    4180
HomeImp    1780
Name: REASON, dtype: int64
```

In [722]:

```
dummy_variable_1 = pd.get_dummies(df["REASON"])
dummy_variable_1.head()
```

Out[722]:

	DebtCon	HomeImp
0	0	1
1	0	1
2	0	1
3	1	0
4	0	1

In [723]:

```
# Replace the dummy variables 'DebtCon' and 'HomeImp'their values
df["REASON"].replace("DebtCon", "1", inplace = True)
df["REASON"].replace("HomeImp", "1", inplace = True)
df.head(5)
```

Out[723]:

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ
0	1	1100	25860.0000	39025.000000	1	Other	10.500000	0.00000	0.000000
1	1	1300	70053.0000	68400.000000	1	Other	7.000000	0.00000	2.000000
2	1	1500	13500.0000	16700.000000	1	Other	4.000000	0.00000	0.000000
3	1	1500	73760.8172	101776.048741	1	Other	8.922268	0.25457	0.449442
4	0	1700	97800.0000	112000.000000	1	Office	3.000000	0.00000	0.000000

In [724]:

```
# correct data format
df["REASON"]=df["REASON"].astype(int, copy=True)
df.dtypes
```

Out[724]:

```
BAD          int64
LOAN         int64
MORTDUE    float64
VALUE      float64
REASON       int64
JOB         object
YOJ        float64
DEROG      float64
DELINQ     float64
CLAGE      float64
NINQ       float64
CLNO       float64
DEBTINC    float64
dtype: object
```

In [725]:

```
df.describe()
```

Out[725]:

	BAD	LOAN	MORTDUE	VALUE	REASON	YOJ	
count	5960.000000	5960.000000	5960.000000	5960.000000	5960.0	5960.000000	596
mean	0.199497	18607.969799	73760.817200	101776.048741	1.0	8.922268	
std	0.399656	11207.480417	42481.395689	56843.931566	0.0	7.239301	
min	0.000000	1100.000000	2063.000000	8000.000000	1.0	0.000000	
25%	0.000000	11100.000000	48139.000000	66489.500000	1.0	3.000000	
50%	0.000000	16300.000000	69529.000000	90000.000000	1.0	8.000000	
75%	0.000000	23300.000000	88200.250000	119004.750000	1.0	12.000000	
max	1.000000	89900.000000	399550.000000	855909.000000	1.0	41.000000	1

In [726]:

```
df.corr()['BAD'].sort_values()
```

Out[726]:

```
CLAGE      -0.165113
LOAN       -0.075099
YOJ        -0.058314
MORTDUE    -0.046034
VALUE      -0.028852
CLNO       -0.004067
DEBTINC     0.124324
NINQ        0.168851
DEROG       0.264068
DELINQ      0.341472
BAD         1.000000
REASON           NaN
Name: BAD, dtype: float64
```

In [727]:

```
df['JOB'].value_counts()
```

Out[727]:

```
Other      2667
ProfExe    1276
Office      948
Mgr         767
Self        193
Sales       109
Name: JOB, dtype: int64
```

In [728]:

```
dummy_variable_2 = pd.get_dummies(df["JOB"])
dummy_variable_2.head()
```

Out[728]:

	Mgr	Office	Other	ProfExe	Sales	Self
0	0	0	1	0	0	0
1	0	0	1	0	0	0
2	0	0	1	0	0	0
3	0	0	1	0	0	0
4	0	1	0	0	0	0

We now have the values 0 and 1 to represent the dummy variables, we will now insert this column back into our original dataset.

In [729]:

```
# merge data frame "df" and "dummy_variable_1"
df = pd.concat([df, dummy_variable_1], axis=1)

# drop original column "REASON" from "df"
df.drop("REASON", axis = 1, inplace=True)

# merge data frame "df" and "dummy_variable_2"
df = pd.concat([df, dummy_variable_2], axis=1)

# drop original column "JOB" from "df"
df.drop("JOB", axis = 1, inplace=True)
```

In [730]:

```
df.head()
```

Out[730]:

	BAD	LOAN	MORTDUE	VALUE	YOJ	DEROG	DELINQ	CLAGE	NI
0	1	1100	25860.0000	39025.000000	10.500000	0.00000	0.000000	94.366667	1.000
1	1	1300	70053.0000	68400.000000	7.000000	0.00000	2.000000	121.833333	0.000
2	1	1500	13500.0000	16700.000000	4.000000	0.00000	0.000000	149.466667	1.000
3	1	1500	73760.8172	101776.048741	8.922268	0.25457	0.449442	179.766275	1.186
4	0	1700	97800.0000	112000.000000	3.000000	0.00000	0.000000	93.333333	0.000

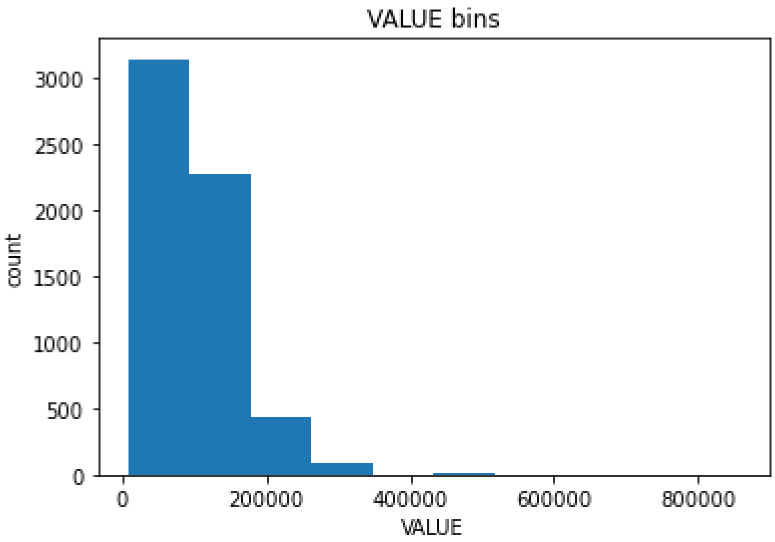
Categorise the loan value into 4 bins of equal size bandwith. First let's plot the distribution of loan Value

In [731]:

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

plt.pyplot.hist(df["VALUE"])

# set x/y labels and plot title
plt.pyplot.xlabel("VALUE")
plt.pyplot.ylabel("count")
plt.pyplot.title("VALUE bins")
```

Out[731]:

```
Text(0.5, 1.0, 'VALUE bins')
```



In [732]:

```
df['VALUE'].value_counts().idxmax()
```

Out[732]:

```
101776.04874145007
```

In [733]:

```
# 4 bins of equal size bandwith
bins = np.linspace(min(df["VALUE"]), max(df["VALUE"]), 5)
bins
```

Out[733]:

```
array([  8000.  , 219977.25, 431954.5 , 643931.75, 855909.  ])
```

In [734]:

```
# Let's see the bin group names and apply the  function "cut" to determine what each va
lue of "df['VALUE']" belongs to.
group_names = ['Low','Moderate', 'Medium', 'High']
df['VALUE_BINNED'] = pd.cut(df['VALUE'], bins, labels=group_names, include_lowest=True
)
df[['VALUE','VALUE_BINNED']].tail(5)
```

Out[734]:

	VALUE	VALUE_BINNED
5955	90185.0	Low
5956	92937.0	Low
5957	92924.0	Low
5958	91861.0	Low
5959	88934.0	Low

In [735]:

```
# checking the bin classification
df["VALUE_BINNED"].value_counts()
```

Out[735]:

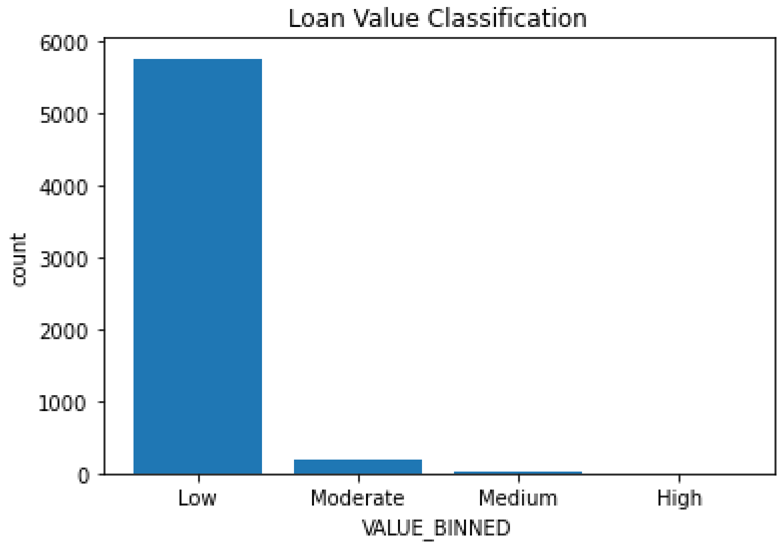
```
Low         5751
Moderate     193
Medium        12
High           4
Name: VALUE_BINNED, dtype: int64
```

In [736]:

```
#plotting the distribution for reach bin

%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot

pyplot.bar(group_names, df["VALUE_BINNED"].value_counts())

# set x/y labels and plot title
plt.pyplot.xlabel("VALUE_BINNED")
plt.pyplot.ylabel("count")
plt.pyplot.title("Loan Value Classification")
```

Out[736]:

```
Text(0.5, 1.0, 'Loan Value Classification')
```



In [737]:

```
# We have now a dataframe with the categorisation for each Loan in the last column' VAL
UE_BINNED'.
df.head()
```

Out[737]:

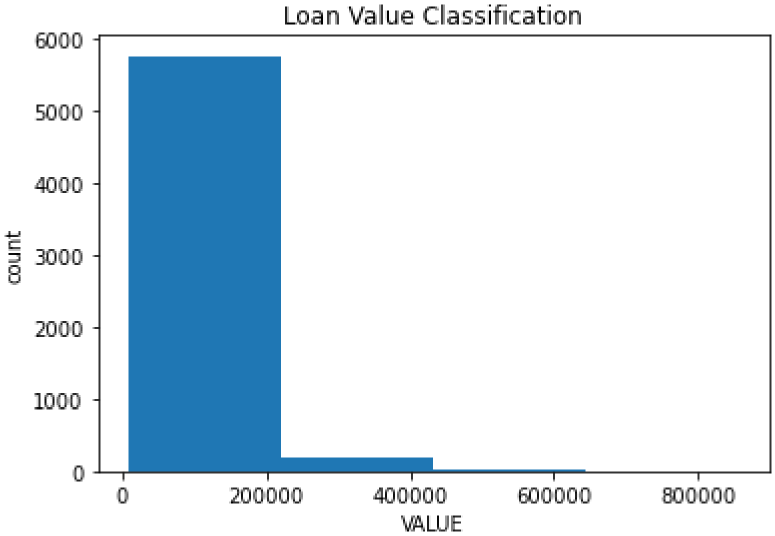
	BAD	LOAN	MORTDUE	VALUE	YOJ	DEROG	DELINQ	CLAGE	NII
0	1	1100	25860.0000	39025.000000	10.500000	0.00000	0.000000	94.366667	1.000C
1	1	1300	70053.0000	68400.000000	7.000000	0.00000	2.000000	121.833333	0.000C
2	1	1500	13500.0000	16700.000000	4.000000	0.00000	0.000000	149.466667	1.000C
3	1	1500	73760.8172	101776.048741	8.922268	0.25457	0.449442	179.766275	1.186C
4	0	1700	97800.0000	112000.000000	3.000000	0.00000	0.000000	93.333333	0.000C

In [738]:

```
# Visualising the newly created bins with a Histogram
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot

a = (0,1,2,3)

# draw historgram of attribute "horsepower" with bins = 4
plt.pyplot.hist(df["VALUE"], bins = 4)

# set x/y labels and plot title
plt.pyplot.xlabel("VALUE")
plt.pyplot.ylabel("count")
plt.pyplot.title("Loan Value Classification")
```

Out[738]:

```
Text(0.5, 1.0, 'Loan Value Classification')
```



In [739]:

```
# Normalising the DATA
#df_mean = (df- df.mean())/df.std()
#df_mean.head()
```

In [740]:

```
# Save the new dataframe into csv format
```

In [741]:

```
df.to_csv('clean_Loandata.csv')
```

In [748]:

```
# Let's separate y_data from x_data (features)
y_data = df['BAD']
# Drop bad debt from the x_data
x_data=df.drop('BAD',axis=1)
x_data
```

Out[748]:

	LOAN	MORTDUE	VALUE	YOJ	DEROG	DELINQ	CLAGE	NINQ
0	1100	25860.0000	39025.000000	10.500000	0.00000	0.000000	94.366667	1.000000
1	1300	70053.0000	68400.000000	7.000000	0.00000	2.000000	121.833333	0.000000
2	1500	13500.0000	16700.000000	4.000000	0.00000	0.000000	149.466667	1.000000
3	1500	73760.8172	101776.048741	8.922268	0.25457	0.449442	179.766275	1.186055
4	1700	97800.0000	112000.000000	3.000000	0.00000	0.000000	93.333333	0.000000
...	...	...	...	...	...	...	...	...
5955	88900	57264.0000	90185.000000	16.000000	0.00000	0.000000	221.808718	0.000000
5956	89000	54576.0000	92937.000000	16.000000	0.00000	0.000000	208.692070	0.000000
5957	89200	54045.0000	92924.000000	15.000000	0.00000	0.000000	212.279697	0.000000
5958	89800	50370.0000	91861.000000	14.000000	0.00000	0.000000	213.892709	0.000000
5959	89900	48811.0000	88934.000000	15.000000	0.00000	0.000000	219.601002	0.000000

5960 rows × 19 columns

This concludes the first part of this project. In the second part, the focus will be on the exploration and data analysis in order to identify the most relevant features that can be used to predict loan default

In [ ]:

```
```