

datapreproc

October 2, 2019

1 STANDARD SCALER

```
[17]: from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
import numpy as np
import pandas as pd
np.set_printoptions(suppress=True)
```

```
[2]: views = pd.DataFrame([1295., 25., 19000., 5., 1., 300.], columns=['views'])
views
```

```
[2]:      views
0    1295.0
1      25.0
2   19000.0
3         5.0
4         1.0
5     300.0
```

```
[3]: ss = StandardScaler()
views['zscore'] = ss.fit_transform(views[['views']])
views
```

```
[3]:      views    zscore
0    1295.0 -0.307214
1      25.0 -0.489306
2   19000.0  2.231317
3         5.0 -0.492173
4         1.0 -0.492747
5     300.0 -0.449877
```

```
[4]: vw = np.array(views['views'])
(vw[0] - np.mean(vw)) / np.std(vw)
```

```
[4]: -0.30721413311687235
```

2 MINMAX SCALAR

```
[5]: mms = MinMaxScaler()  
views['minmax'] = mms.fit_transform(views[['views']])  
views
```

```
[5]:
```

	views	zscore	minmax
0	1295.0	-0.307214	0.068109
1	25.0	-0.489306	0.001263
2	19000.0	2.231317	1.000000
3	5.0	-0.492173	0.000211
4	1.0	-0.492747	0.000000
5	300.0	-0.449877	0.015738

```
[6]: (vw[0] - np.min(vw)) / (np.max(vw) - np.min(vw))
```

```
[6]: 0.06810884783409653
```

3 ROBUST SCALAR

```
[8]: rs = RobustScaler()  
views['robust'] = rs.fit_transform(views[['views']])  
views
```

```
[8]:
```

	views	zscore	minmax	robust
0	1295.0	-0.307214	0.068109	1.092883
1	25.0	-0.489306	0.001263	-0.132690
2	19000.0	2.231317	1.000000	18.178528
3	5.0	-0.492173	0.000211	-0.151990
4	1.0	-0.492747	0.000000	-0.155850
5	300.0	-0.449877	0.015738	0.132690

```
[9]: quartiles = np.percentile(vw, (25., 75.))  
iqr = quartiles[1] - quartiles[0]  
(vw[0] - np.median(vw)) / iqr
```

```
[9]: 1.0928829915560916
```

4 STANDARD SCALING

```
[14]: import pandas as pd  
X_train=pd.read_csv('X_train.csv')  
Y_train=pd.read_csv('Y_train.csv')  
X_test=pd.read_csv('X_test.csv')  
Y_test=pd.read_csv('Y_test.csv')  
X_train.head()
```

```
[14]: Loan_ID Gender Married Dependents Education Self_Employed \
0 LP001032 Male No 0 Graduate No
1 LP001824 Male Yes 1 Graduate No
2 LP002928 Male Yes 0 Graduate No
3 LP001814 Male Yes 2 Graduate No
4 LP002244 Male Yes 0 Graduate No

ApplicantIncome CoapplicantIncome LoanAmount Loan_Amount_Term \
0 4950 0.0 125 360
1 2882 1843.0 123 480
2 3000 3416.0 56 180
3 9703 0.0 112 360
4 2333 2417.0 136 360

Credit_History Property_Area
0 1 Urban
1 1 Semiurban
2 1 Semiurban
3 1 Urban
4 1 Urban
```

5 Label ENCODER

```
[14]: from sklearn.preprocessing import LabelEncoder
```

```
[23]: lb=LabelEncoder()
X_train['Education'].unique()
X_train['en_education']=lb.fit_transform(X_train['Education'])
X_train['en_education'].unique()
```

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

```
[27]: X_2 = X.apply(lb.fit_transform)
X_2.head()
```

```
[27]: Loan_ID Gender Married Dependents Education Self_Employed \
0 11 1 0 0 0 0
1 151 1 1 1 0 0
2 367 1 1 0 0 0
3 150 1 1 2 0 0
4 236 1 1 0 0 0

Property_Area
0 2
1 1
2 1
3 2
4 2
```

6 ONE HOT ENCODING

```
[24]: X = X_train.select_dtypes(include=[object])
      X.head(3)
```

```
[24]:   Loan_ID  Gender  Married  Dependents  Education  Self_Employed  Property_Area
0  LP001032   Male      No          0  Graduate             No          Urban
1  LP001824   Male     Yes          1  Graduate             No      Semiurban
2  LP002928   Male     Yes          0  Graduate             No      Semiurban
```

```
[25]: X.shape
```

```
[25]: (384, 7)
```

```
[28]: from sklearn.preprocessing import OneHotEncoder
```

```
[32]: enc=OneHotEncoder()
      enc.fit(X_2)
      onehotlabels = enc.transform(X_2).toarray()
      onehotlabels.shape
```

/home/nbuser/anaconda3_501/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py:371: FutureWarning: The handling of integer data will change in version 0.22. Currently, the categories are determined based on the range [0, max(values)], while in the future they will be determined based on the unique values.

If you want the future behaviour and silence this warning, you can specify "categories='auto'".

In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the OneHotEncoder directly.

```
warnings.warn(msg, FutureWarning)
```

```
[32]: (384, 399)
```

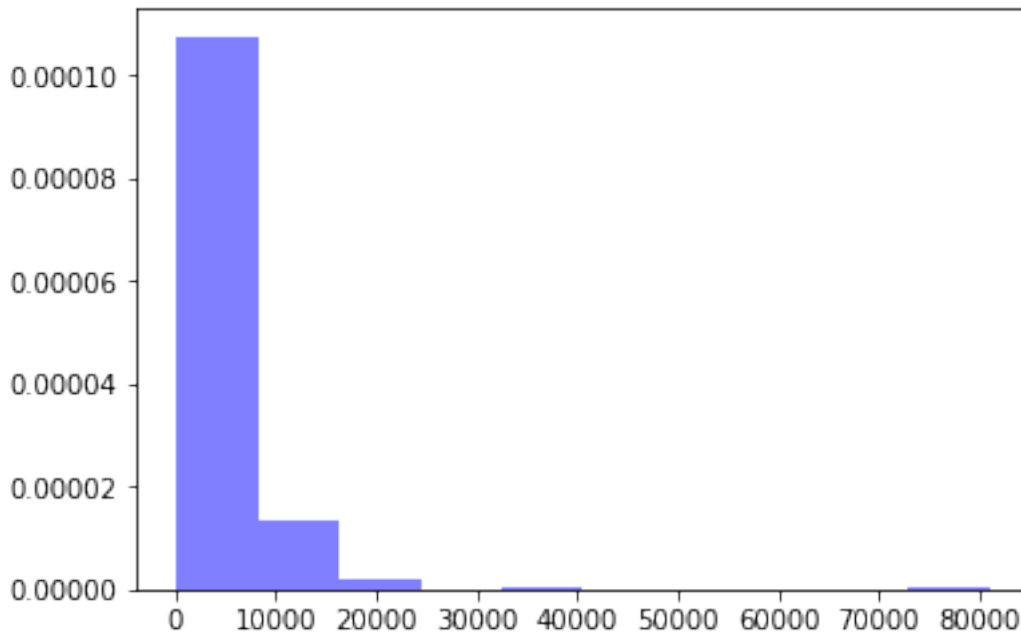
```
[30]: onehotlabels
```

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

7 EXPLORATIVE DATA ANALYSIS

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

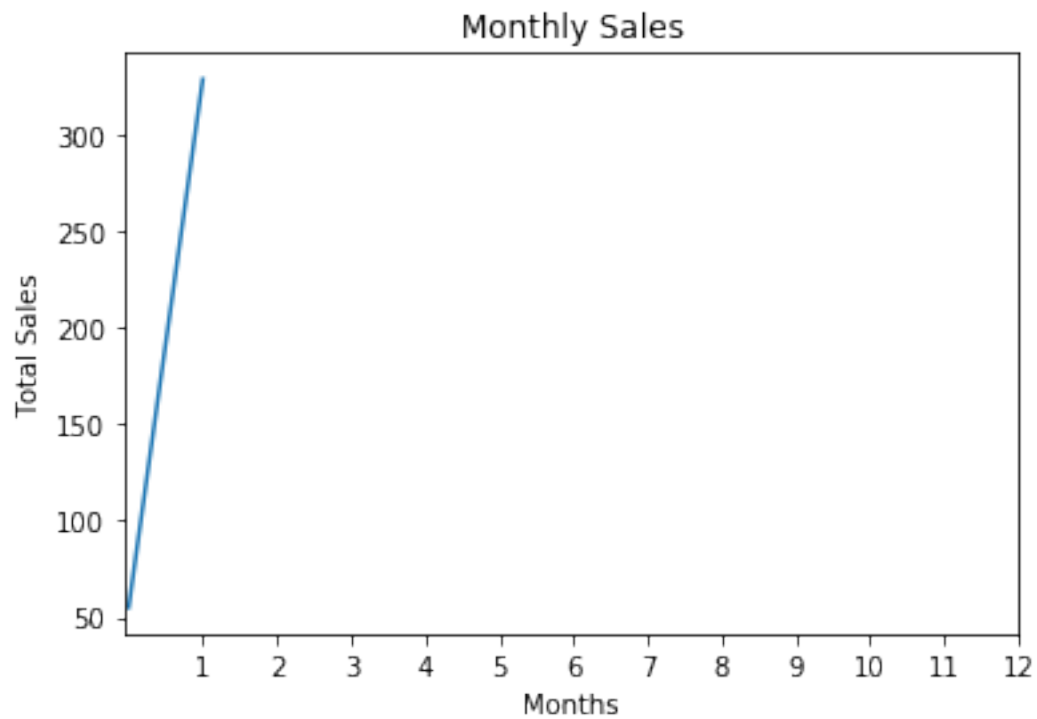
```
[37]: num_bins = 10
plt.hist(X_train['ApplicantIncome'], num_bins, normed=1, facecolor='blue',
         alpha=0.5)
plt.show()
```



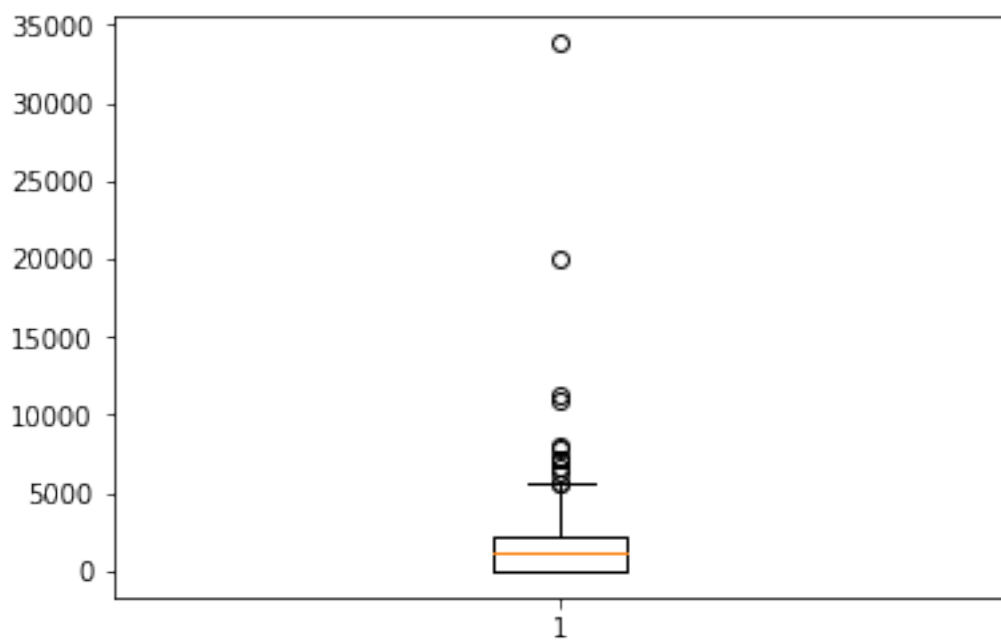
```
[39]: sales_by_month = X_train.groupby('Credit_History').size()
print(sales_by_month)
#Plotting the Graph
plot_by_month = sales_by_month.plot(title='Monthly
    ↳Sales',xticks=(1,2,3,4,5,6,7,8,9,10,11,12))
plot_by_month.set_xlabel('Months')
plot_by_month.set_ylabel('Total Sales')
```

```
Credit_History
0      55
1     329
dtype: int64
```

```
[39]: Text(0, 0.5, 'Total Sales')
```



```
[40]: y= list(X_train.CoapplicantIncome)
plt.boxplot(y)
plt.show()
```



8 FEATURE SELECTION CORELATION

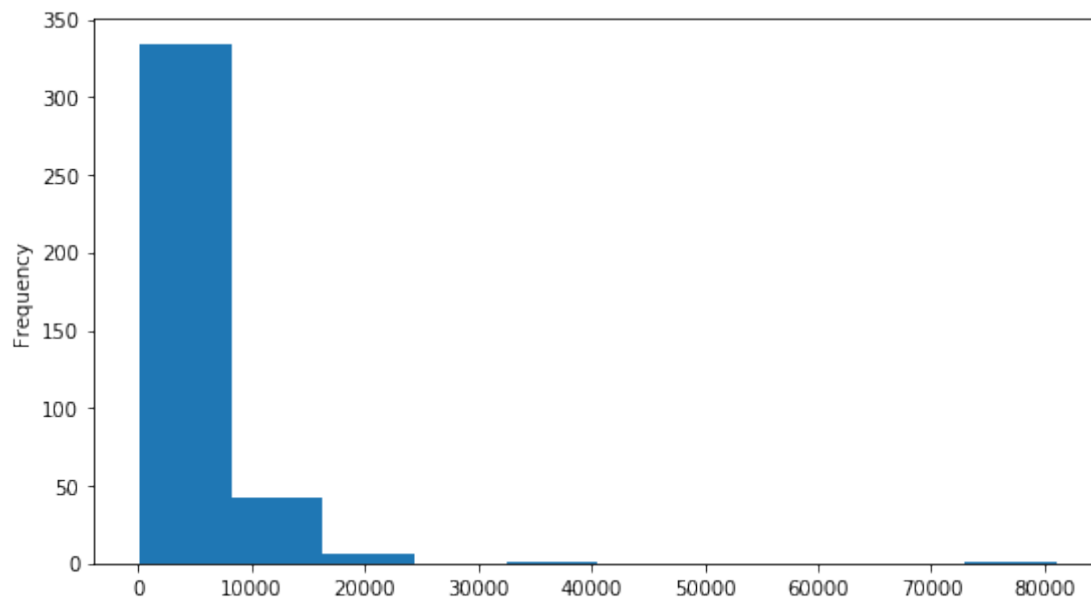
```
[2]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm
```

```
[6]: X_train['ApplicantIncome'].describe()
```

```
[6]: count      384.000000
mean      5149.718750
std       5304.921764
min        150.000000
25%       2898.750000
50%       3893.500000
75%       5819.500000
max      81000.000000
Name: ApplicantIncome, dtype: float64
```

```
[7]: plt.figure(figsize = (9, 5))
X_train['ApplicantIncome'].plot(kind = "hist")
```

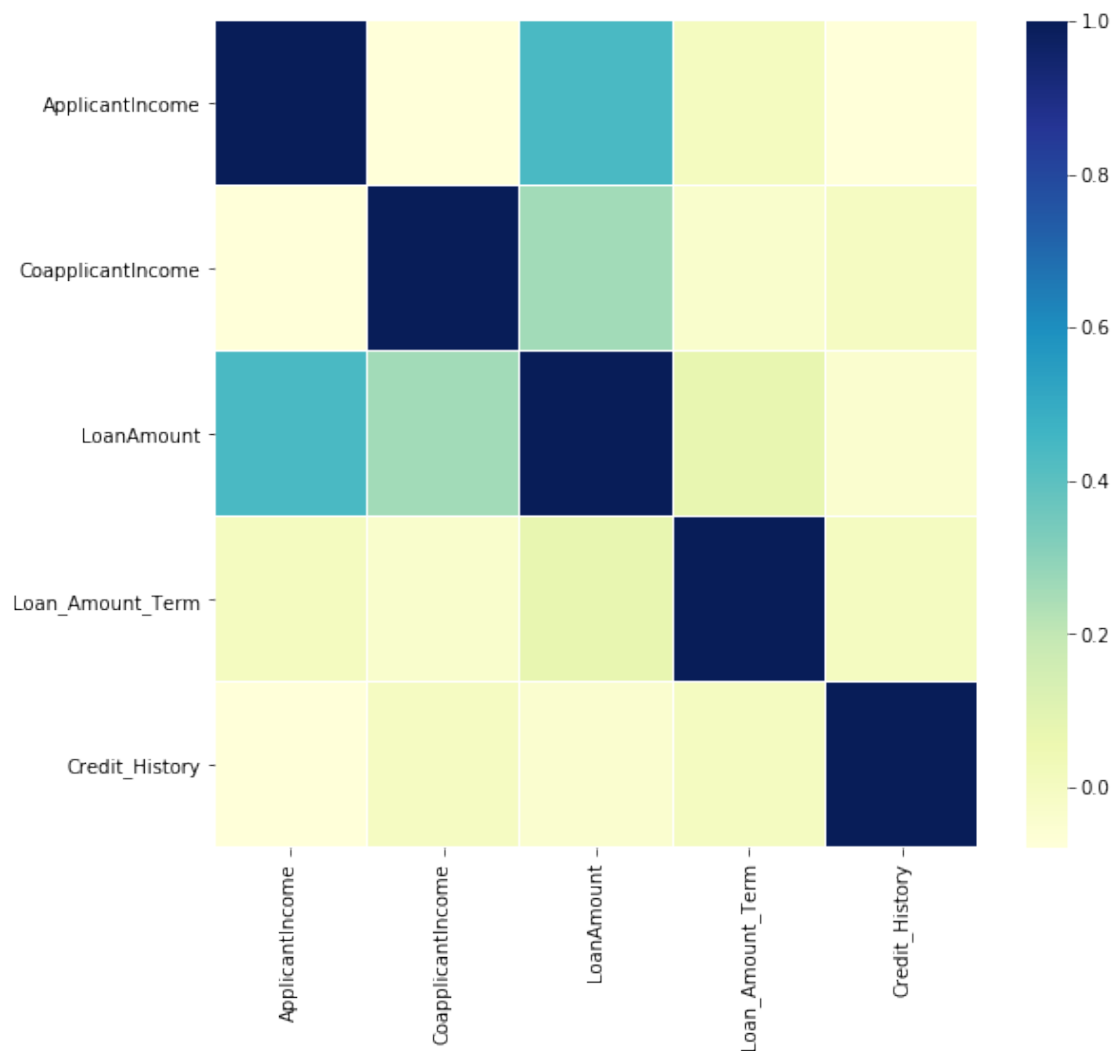
```
[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f756cd2e7f0>
```



```
[9]: corrmatrix = X_train.corr()

f, ax = plt.subplots(figsize=(9, 8))
sns.heatmap(corrmatrix, ax=ax, cmap="YlGnBu", linewidths=0.1)
```

[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f756c5f79e8>

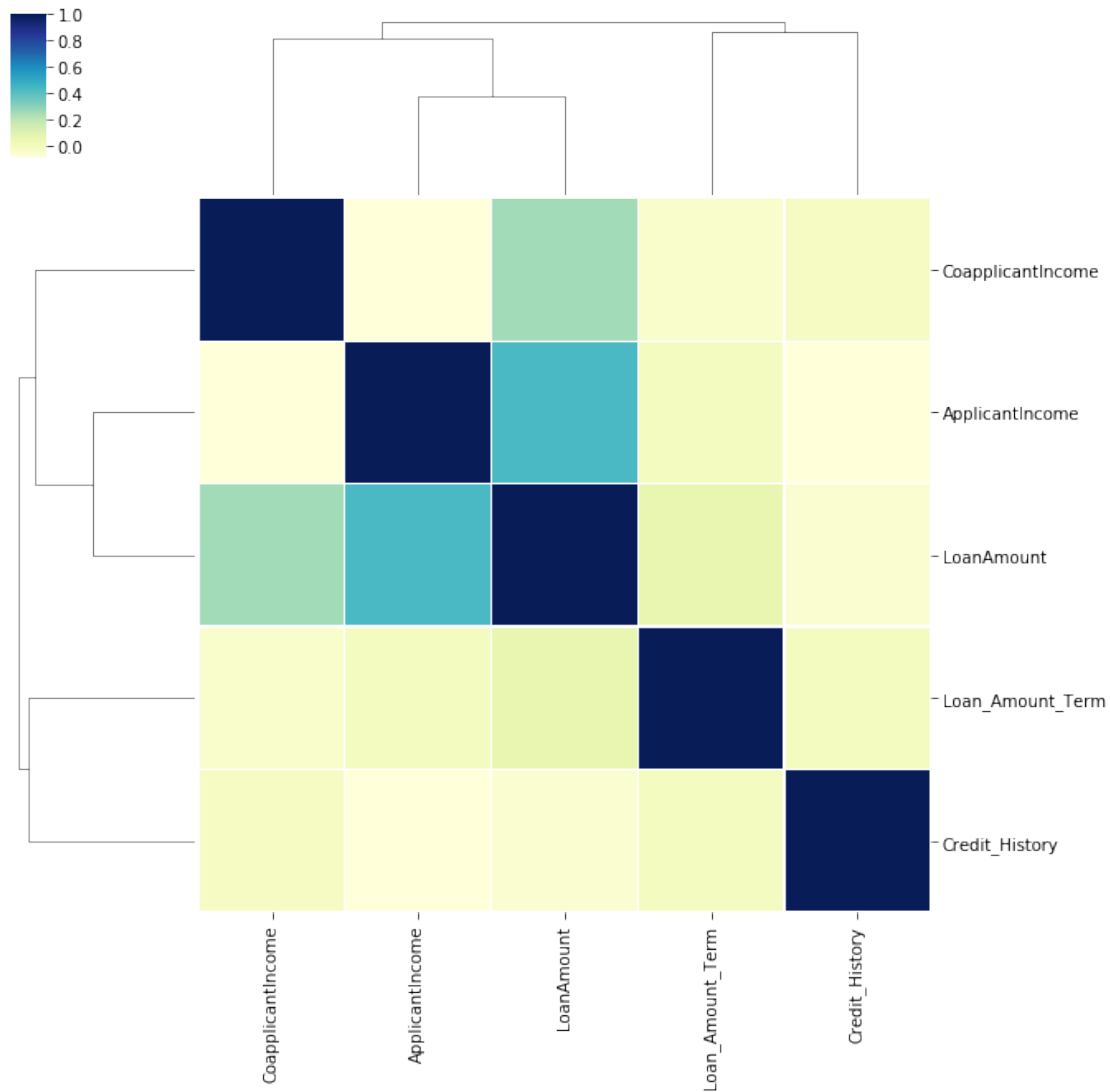


```
[12]: corrmatrix = X_train.corr()

cg = sns.clustermap(corrmatrix, cmap="YlGnBu", linewidths=0.1)
plt.setp(cg.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)

cg
```

[12]: <seaborn.matrix.ClusterGrid at 0x7f756c197320>



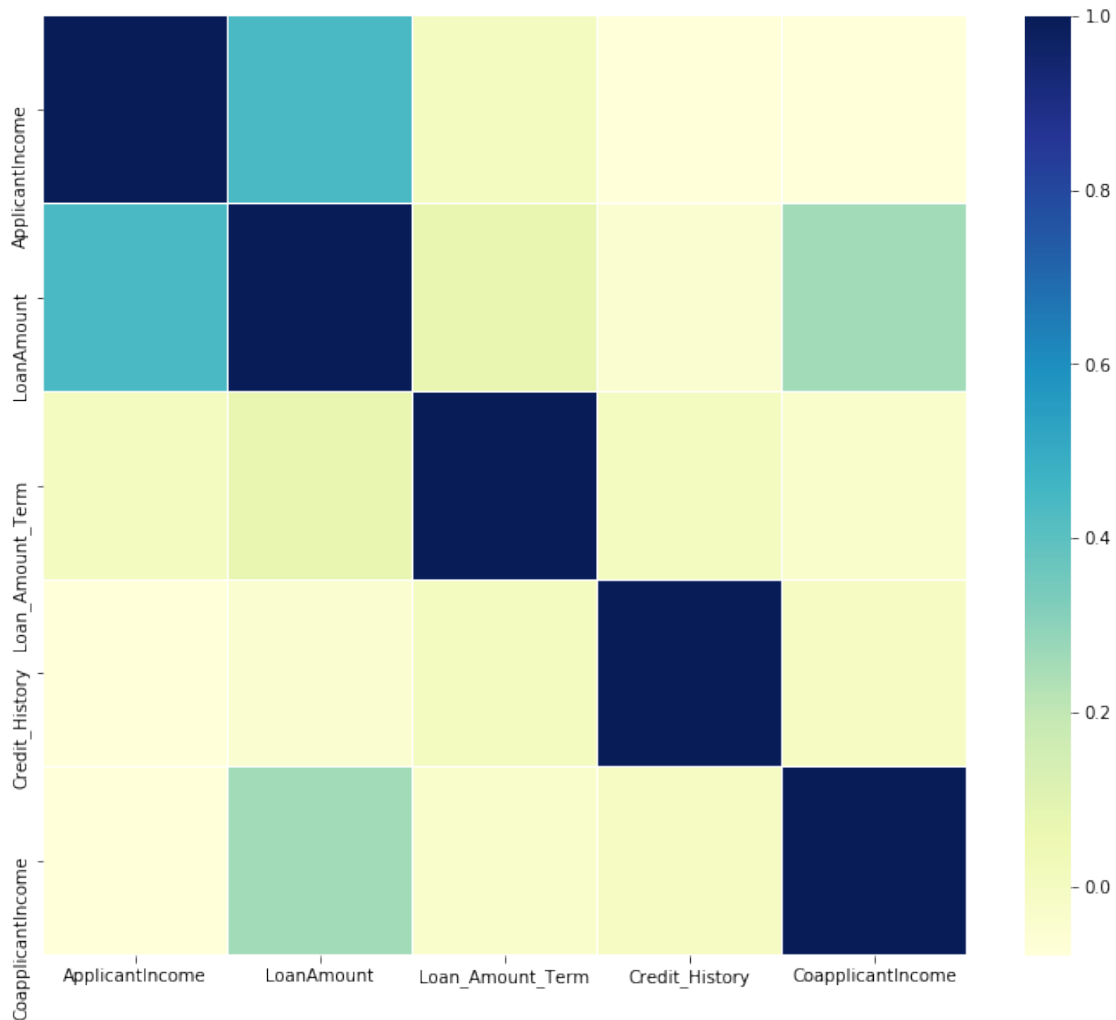
```
[15]: k = 15

cols = corrmatrix.nlargest(k, 'ApplicantIncome')['ApplicantIncome'].index

cm = np.corrcoef(X_train[cols].values.T)
f, ax = plt.subplots(figsize=(12, 10))

sns.heatmap(cm, ax = ax, cmap = "YlGnBu",
            linewidths = 0.1, yticklabels = cols.values,
            xticklabels = cols.values)
```

```
[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f756b5f0b38>
```



```
[16]: import scipy.stats
```

```
[18]: scipy.stats.chisquare(X_train["Property_Area"].value_counts())
```

```
[18]: Power_divergenceResult(statistic=4.359375, pvalue=0.113076861638956)
```

```
[19]: cont = pd.crosstab(X_train["Property_Area"], X_train["ApplicantIncome"])
```

```
[20]: scipy.stats.chi2_contingency(cont)
```

```
[20]: (690.1927583949571,
      0.3050310535570292,
      672,
      array([[0.29427083, 0.29427083, 0.29427083, ..., 0.29427083, 0.29427083,
              0.29427083],
            [0.38020833, 0.38020833, 0.38020833, ..., 0.38020833, 0.38020833,
              0.38020833],
            [0.32552083, 0.32552083, 0.32552083, ..., 0.32552083, 0.32552083,
              0.32552083],
```

```
0.32552083]]))
```

9 CHISQUARED!

```
[21]: import numpy as np
a1 = [6, 4, 5, 10]
a2 = [8, 5, 3, 3]
a3 = [5, 4, 8, 4]
a4 = [4, 11, 7, 13]
a5 = [5, 8, 7, 6]
a6 = [7, 3, 5, 9]
dice = np.array([a1, a2, a3, a4, a5, a6])
```

```
[23]: dice
```

```
[23]: array([[ 6,  4,  5, 10],
           [ 8,  5,  3,  3],
           [ 5,  4,  8,  4],
           [ 4, 11,  7, 13],
           [ 5,  8,  7,  6],
           [ 7,  3,  5,  9]])
```

```
[24]: from scipy import stats

stats.chi2_contingency(dice)
```

```
[24]: (16.490612061288754,
      0.35021521809742745,
      15,
      array([[ 5.83333333,  5.83333333,  5.83333333,  7.5         ],
             [ 4.43333333,  4.43333333,  4.43333333,  5.7         ],
             [ 4.9         ,  4.9         ,  4.9         ,  6.3         ],
             [ 8.16666667,  8.16666667,  8.16666667, 10.5         ],
             [ 6.06666667,  6.06666667,  6.06666667,  7.8         ],
             [ 5.6         ,  5.6         ,  5.6         ,  7.2         ]]))
```

```
[26]: chi2_stat, p_val, dof, ex = stats.chi2_contingency(dice)
print("===Chi2 Stat===")
print(chi2_stat)
print("\n")
print("===Degrees of Freedom===")
print(dof)
print("\n")
print("===P-Value===")
print(p_val)
print("\n")
print("===Contingency Table===")
print(ex)
```

```
===Chi2 Stat===  
16.490612061288754
```

```
===Degrees of Freedom===  
15
```

```
===P-Value===  
0.35021521809742745
```

```
===Contingency Table===  
[[ 5.83333333  5.83333333  5.83333333  7.5      ]  
 [ 4.43333333  4.43333333  4.43333333  5.7      ]  
 [ 4.9         4.9         4.9         6.3      ]  
 [ 8.16666667  8.16666667  8.16666667 10.5     ]  
 [ 6.06666667  6.06666667  6.06666667  7.8      ]  
 [ 5.6         5.6         5.6         7.2     ]]
```

```
[ ]:
```