

```
import re
import os

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

```
In [2]: # File Path
data_path = "combined_results.csv"
```

```
In [3]: # reading in csv and converting to dataframes
data = pd.read_csv(data_path, error_bad_lines=False, low_memory=False)
```

```
In [4]: data
```

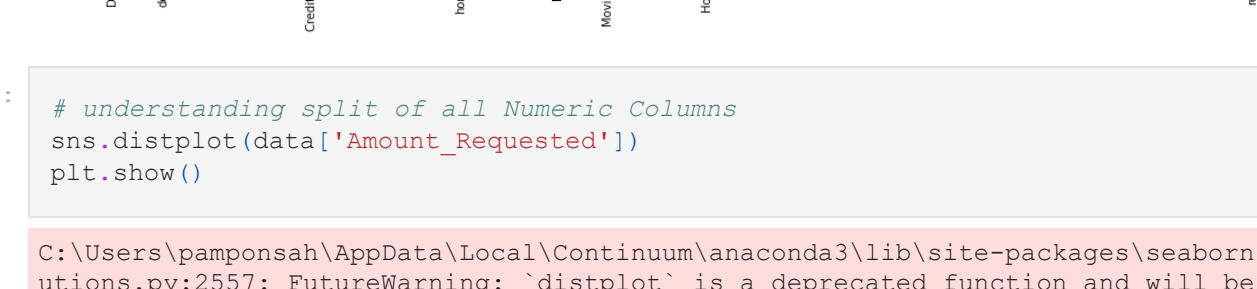
	Amount_Requested	Risk_Score	debt_to_income_ratio	State	Emp_length	Purpose	Loan_Status
0	50000.0	6690.0	21.80	OK	8	other	Accepted
1	150000.0	7040.0	18.29	FL	2	debt_consolidation	Accepted
2	112000.0	6690.0	43.97	NH	0	medical	Accepted
3	250000.0	6690.0	12.89	AL	10	debt_consolidation	Accepted
4	30000.0	7640.0	0.58	WA	9	major_purchase	Accepted
...
1081515	30000.0	5350.0	1.54	CT	5	Medical expenses	Rejected
1081516	60000.0	5800.0	5.55	TX	0	Car financing	Rejected
1081517	30000.0	5610.0	0.00	OR	0	Debt consolidation	Rejected
1081518	25000.0	5730.0	8.55	NY	0	Other	Rejected
1081519	25000.0	5670.0	0.00	AL	2	Car financing	Rejected

1081520 rows x 7 columns

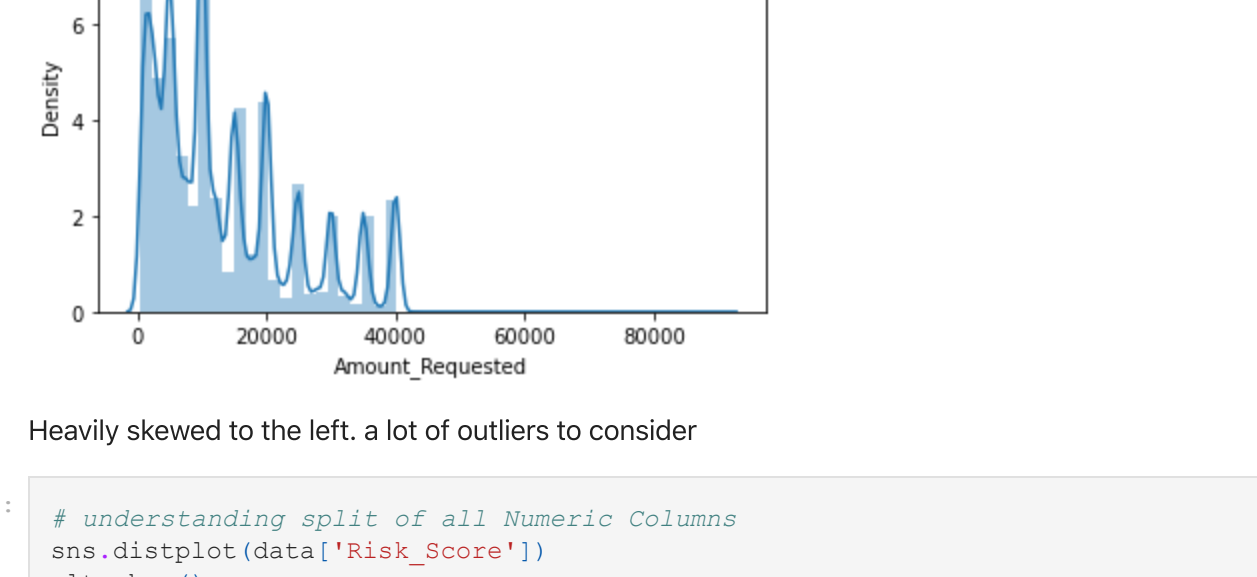
Cont'd Exploratory Data analysis

Univariate exploration

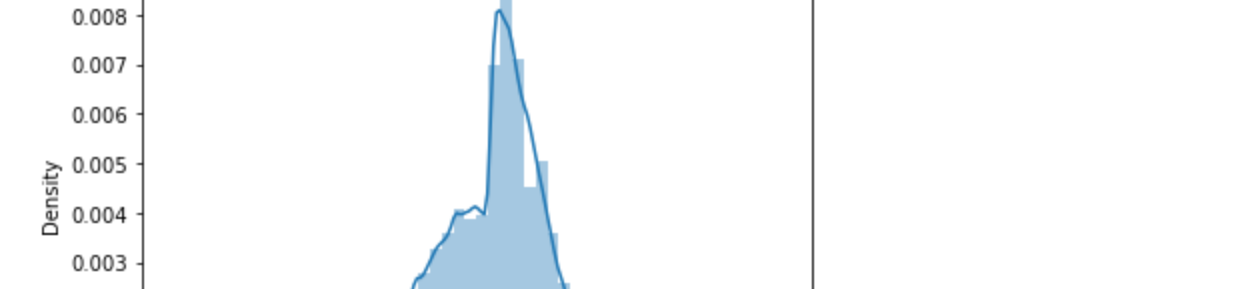
```
In [5]: # Understanding the split of our Accept/Reject - Our Target columns
print(data['Loan_Status'].value_counts(normalize=True))
data['Loan_Status'].value_counts().plot.bar(figsize=(7,5), title="Dependent Variable - Loan Status")
```



```
In [6]: # understanding split of all categorical Columns
data['State'].value_counts().plot.bar(figsize=(20,5), title="State")
plt.show()
data['Purpose'].value_counts().plot.bar(figsize=(20,5), title="Loan Purpose")
plt.show()
```

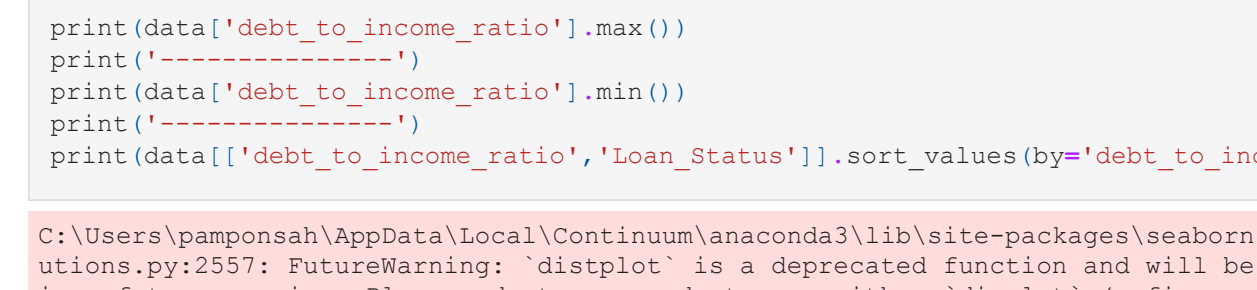


```
In [7]: # understanding split of all Numeric Columns
sns.distplot(data['Amount_Requested'])
plt.show()
```



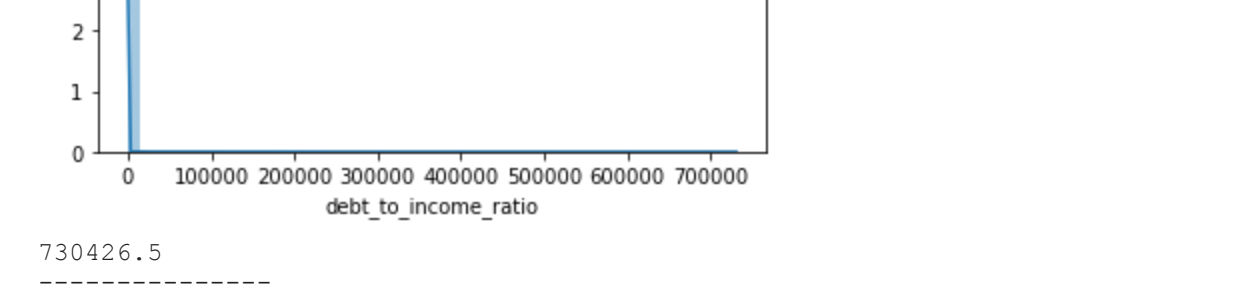
Heavily skewed to the left, a lot of outliers to consider

```
In [8]: # understanding split of all Numeric Columns
sns.distplot(data['Risk_Score'])
plt.show()
```



Distribution is good

```
In [9]: # understanding split of all Numeric Columns
sns.distplot(data['debt_to_income_ratio'])
plt.show()
print(data['debt_to_income_ratio'].max())
print('-----')
print(data['debt_to_income_ratio'].min())
print('-----')
print(data[['debt_to_income_ratio', 'Loan_Status']].sort_values(by='debt_to_income_ratio'))
```



```
In [10]: # dti > 10
dti_values_over100 = data[data.debt_to_income_ratio > 100]
```

```
In [11]: print(dti_values_over100.info())
dti_values_over100.head()
```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 22081 entries, 493 to 1081423
Data columns (total 7 columns):
Column Non-Null Count Dtype
--
0 Amount_Requested 22081 non-null float64
1 Risk_Score 22081 non-null float64
2 debt_to_income_ratio 22081 non-null object
3 State 22081 non-null object
4 Emp_length 22081 non-null int64
5 Purpose 22081 non-null object
6 Loan_Status 22081 non-null object
dtypes: float64(3), int64(1), object(3)
memory usage: 1.3+ MB
None

	Amount_Requested	Risk_Score	debt_to_income_ratio	State	Emp_length	Purpose	Loan_Status
493	280000.0	7040.0	112.20	NV	0	debt_consolidation	Accepted
643	400000.0	7090.0	104.46	OH	4	debt_consolidation	Accepted
1229	100000.0	7140.0	17.17	TX	9	debt_consolidation	Accepted
1683	90000.0	6840.0	106.66	IL	4	credit_card	Accepted
1879	350000.0	7140.0	102.47	UT	0	debt_consolidation	Accepted

```
In [12]: dti_values_over100['Loan_Status'].value_counts()
```

Rejected 2119
Accepted 888
Name: Loan_Status, dtype: int64
It appears that a lot of DTI values Over 100 (95%+) are those who were rejected. Due to the nature that being a severe outlier, I think it would be best to reject anyone whose DTI is over 100.
Let's remove them.

```
In [13]: clean_dti_data = data[data.debt_to_income_ratio < 100]
```

```
In [14]: clean_dti_data
```

	Amount_Requested	Risk_Score	debt_to_income_ratio	State	Emp_length	Purpose	Loan_Status
0	50000.0	6690.0	21.80	OK	8	other	Accepted
1	150000.0	7040.0	18.29	FL	2	debt_consolidation	Accepted
2	112000.0	6690.0	43.97	NH	0	medical	Accepted
3	250000.0	6690.0	12.89	AL	10	debt_consolidation	Accepted
4	30000.0	7640.0	0.58	WA	9	major_purchase	Accepted
...
1081515	30000.0	5350.0	1.54	CT	5	Medical expenses	Rejected
1081516	60000.0	5800.0	5.55	TX	0	Car financing	Rejected
1081517	30000.0	5610.0	0.00	OR	0	Debt consolidation	Rejected
1081518	25000.0	5730.0	8.55	NY	0	Other	Rejected
1081519	25000.0	5670.0	0.00	AL	2	Car financing	Rejected

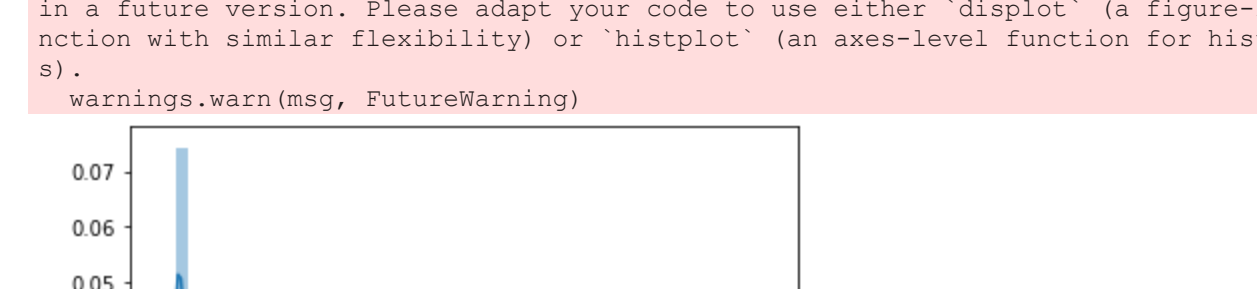
1034236 rows x 7 columns

```
In [15]: sns.distplot(clean_dti_data['debt_to_income_ratio'])
plt.show()
```

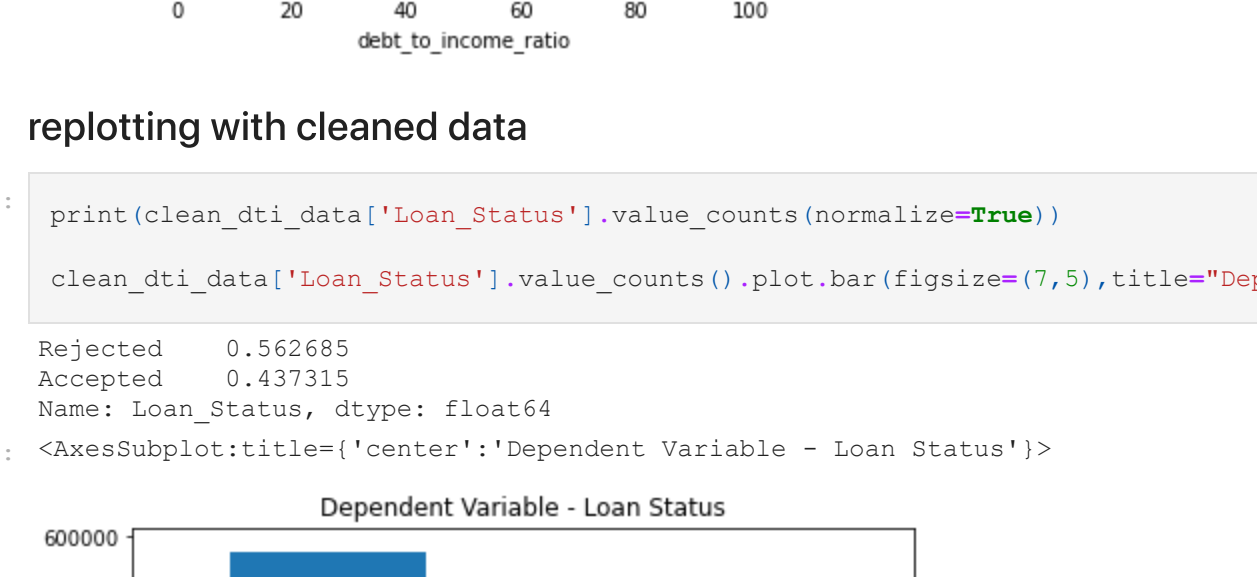


replotting with cleaned data

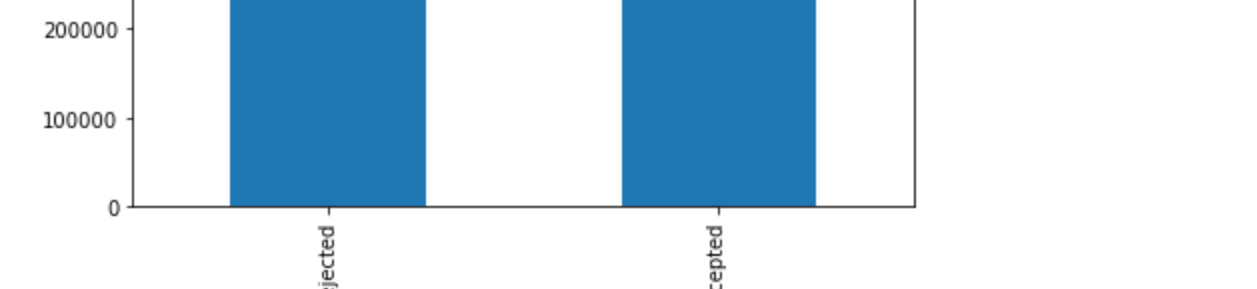
```
In [16]: print(clean_dti_data['Loan_Status'].value_counts(normalize=True))
clean_dti_data['Loan_Status'].value_counts().plot.bar(figsize=(7,5), title="Dependent Variable - Loan Status")
```



```
In [17]: # understanding split of all categorical Columns in cleaned data
clean_dti_data['State'].value_counts().plot.bar(figsize=(20,5), title="State")
plt.show()
clean_dti_data['Purpose'].value_counts().plot.bar(figsize=(20,5), title="Loan Purpose")
plt.show()
```



```
In [18]: # understanding split of all Numeric Columns in cleaned data
sns.distplot(clean_dti_data['Amount_Requested'])
plt.show()
```



```
In [19]: # understanding split of all Numeric Columns
sns.distplot(clean_dti_data['Risk_Score'])
plt.show()
```



Majority of the data remains relatively unchanged after cleaning out DTI

However, I need to recategorize purpose of loan

```
In [20]: # purpose of Loans need to be recategorized
clean_dti_data['Purpose'].value_counts()
```

```
In [21]: clean_dti_data.loc[(clean_dti_data.Purpose == 'debt_consolidation'), 'Purpose'] = 'Debt consolidation'
clean_dti_data.loc[(clean_dti_data.Purpose == 'other'), 'Purpose'] = 'Other'
clean_dti_data.loc[(clean_dti_data.Purpose == 'car'), 'Purpose'] = 'Car financing'
clean_dti_data.loc[(clean_dti_data.Purpose == 'home_improvement'), 'Purpose'] = 'Home improvement'
clean_dti_data.loc[(clean_dti_data.Purpose == 'moving'), 'Purpose'] = 'Moving and relocation'
clean_dti_data.loc[(clean_dti_data.Purpose == 'major_purchase'), 'Purpose'] = 'Major purchase'
clean_dti_data.loc[(clean_dti_data.Purpose == 'medical'), 'Purpose'] = 'Medical expenses'
clean_dti_data.loc[(clean_dti_data.Purpose == 'house'), 'Purpose'] = 'Home buying'
clean_dti_data.loc[(clean_dti_data.Purpose == 'vacation'), 'Purpose'] = 'Vacation'
clean_dti_data.loc[(clean_dti_data.Purpose == 'small_business'), 'Purpose'] = 'Business'
clean_dti_data.loc[(clean_dti_data.Purpose == 'renewable_energy'), 'Purpose'] = 'Green loan'
```

C:\Users\pamponasah\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\indexing.py:1720: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row_indexer, col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

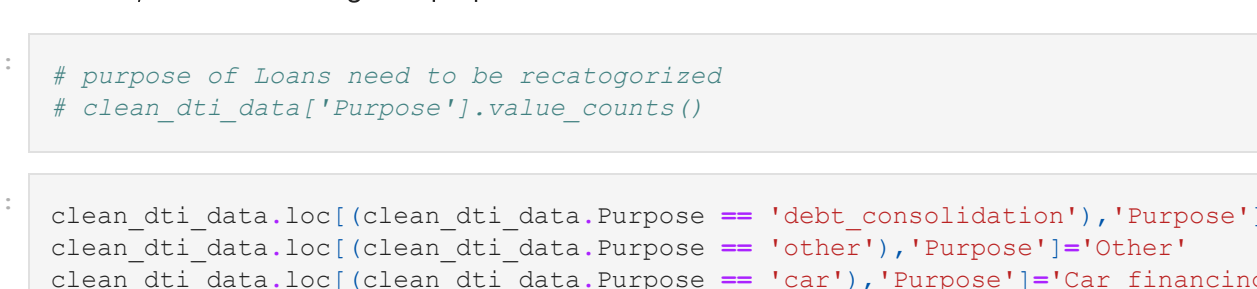
```
In [22]: # cleaned Purpose column
clean_dti_data['Purpose'].value_counts()
```

Debt consolidation	463923
Other	179281
Credit card	118031
Credit card refinancing	77889
Home improvement	42409
Car financing	35979
Major purchase	26423
Medical expenses	24098
Home buying	23977
Moving and relocation	20437
Business	13010
Vacation	7780
Green loan	1060
Learning and training	34
wedding	5

Name: Purpose, dtype: int64

Bivariate exploration

```
In [23]: # calculate correlation matrix
corr = clean_dti_data.corr() # plot the heatmap
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, cmap=
```



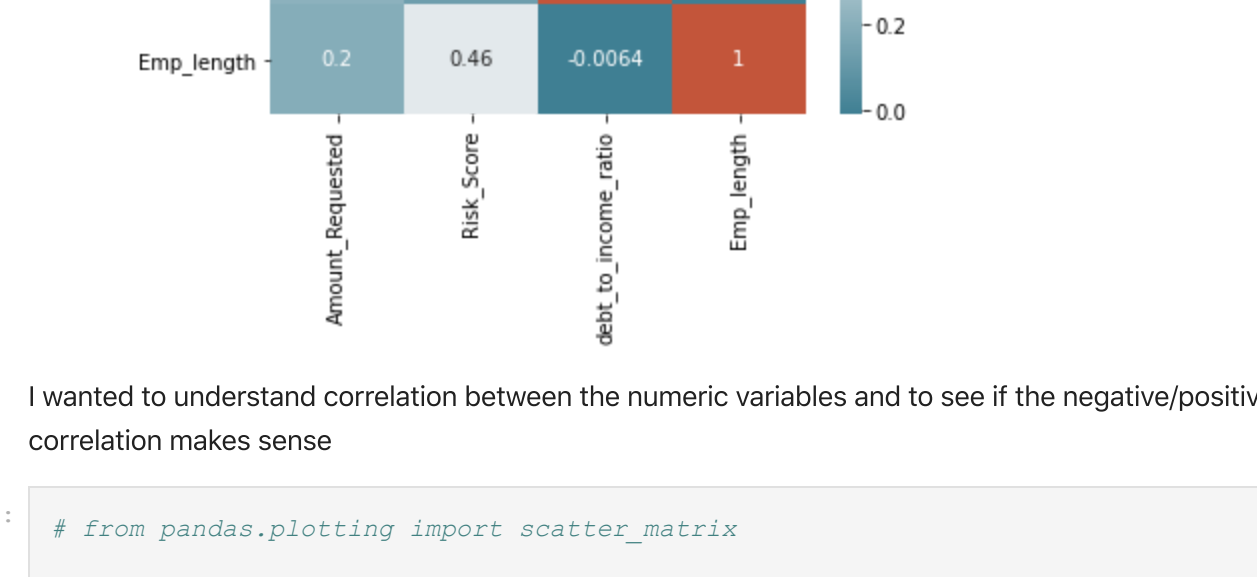
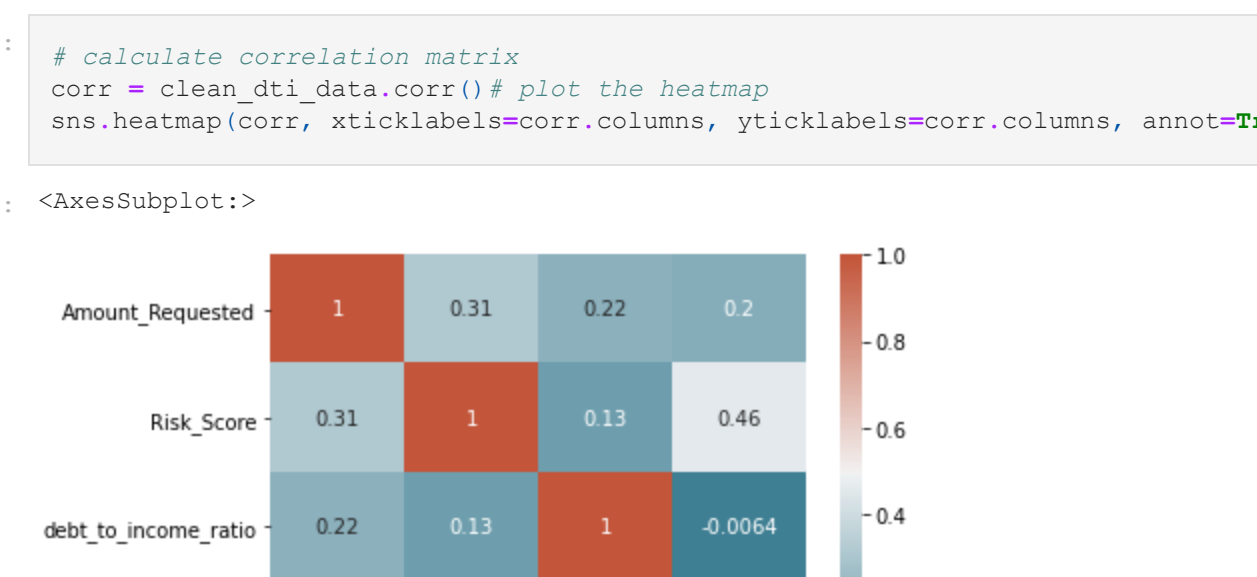
I wanted to understand correlation between the numeric variables and to see if the negative/positive correlation makes sense

```
In [24]: # attributes = ['Amount_Requested', 'Risk_Score', 'debt_to_income_ratio']
# scatter_matrix(clean_dti_data[attributes], figsize=(12,8))
```

```
In [25]: clean_dti_data.info()
```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1034236 entries, 0 to 1081519
Data columns (total 7 columns):
Column Non-Null Count Dtype
--
0 Amount_Requested 1034236 non-null float64
1 Risk_Score 1034236 non-null float64
2 debt_to_income_ratio 1034236 non-null object
3 State 1034236 non-null object
4 Emp_length 1034236 non-null int64
5 Purpose 1034236 non-null object
6 Loan_Status 1034236 non-null object
dtypes: float64(3), int64(1), object(3)
memory usage: 63.1+ MB

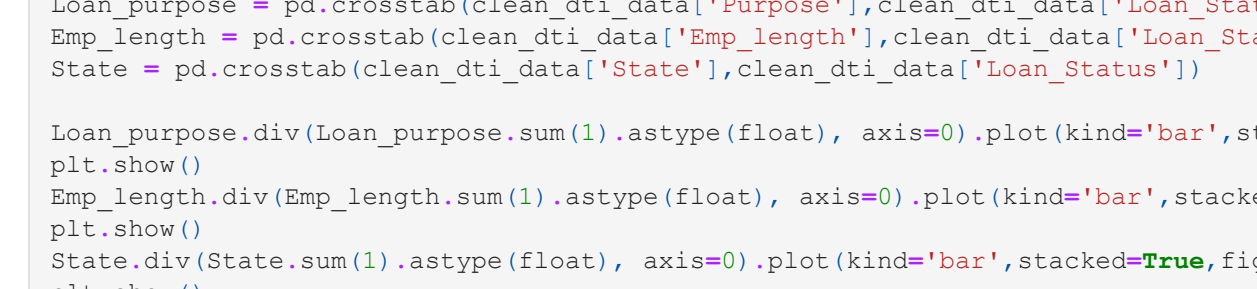
```
In [26]: # Splitting all the categorical variables into subsets of the data to see how they in
Loan_purpose = pd.crosstab(clean_dti_data['Purpose'], clean_dti_data['Loan_Status'])
Emp_length = pd.crosstab(clean_dti_data['Emp_length'], clean_dti_data['Loan_Status'])
State = pd.crosstab(clean_dti_data['State'], clean_dti_data['Loan_Status'])
Loan_purpose.div(Loan_purpose.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True, title="Loan Purpose")
Emp_length.div(Emp_length.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True, title="Emp_length")
State.div(State.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True, title="State")
plt.show()
```



```
In [27]: # Plot Numeric variables vs target
clean_dti_data.groupby('Loan_Status')['Amount_Requested'].mean().plot.bar()
bins=[0,10000,20000,30000]
groups=['Low', 'Average', 'High']
clean_dti_data['Amount_Requested_bins'] = pd.cut(clean_dti_data['Amount_Requested'], bins, labels=groups, include_lowest=True)
Amount_Requested_bins = pd.crosstab(clean_dti_data['Amount_Requested_bins'], clean_dti_data['Loan_Status'])
Amount_Requested_bins.div(Amount_Requested_bins.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True, title="Amount_Requested_bins")
plt.show()
```

C:\Users\pamponasah\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row_indexer, col_indexer] = value instead

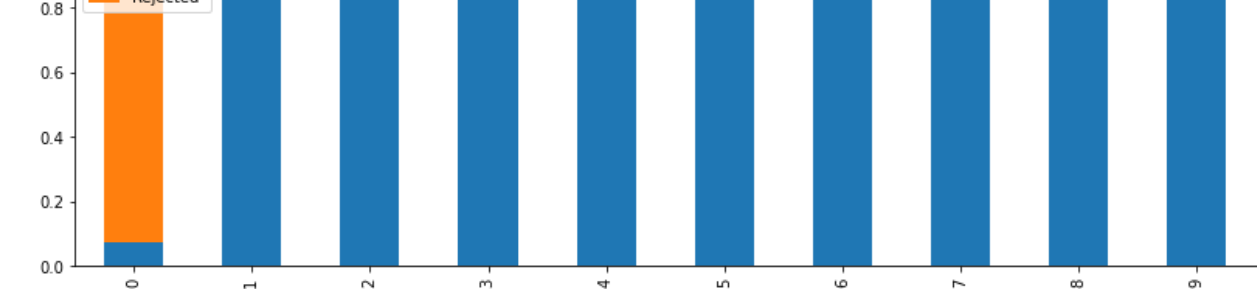
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy



```
In [28]: # plot risk scores
bins=[0,629,719,850]
groups=['Bad', 'Fair', 'Good', 'Excellent']
clean_dti_data['Risk_Score_bins'] = pd.cut(clean_dti_data['Risk_Score'], bins, labels=groups, include_lowest=True)
Risk_Score_bins = pd.crosstab(clean_dti_data['Risk_Score_bins'], clean_dti_data['Loan_Status'])
Risk_Score_bins.div(Risk_Score_bins.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True, title="Risk_Score_bins")
plt.show()
```

C:\Users\pamponasah\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row_indexer, col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy



```
In [29]: # Lets redo the correlation, but this time let's turn the loan Status into a binary variable
clean_dti_data = clean_dti_data.drop(['Risk_Score_bins', 'Amount_Requested_bins'], axis=1)
```

Create a final table to work with

```
In [30]: model_data = clean_dti_data.copy()
model_data
```

	Amount_Requested	Risk_Score	debt_to_income_ratio	State	Emp_length	Purpose	Loan_Status
0	50000.0	6690.0	21.80	OK	8	Other	Accepted
1	150000.0	7040.0	18.29	FL	2	Debt consolidation	Accepted
2	112000.0	6690.0	43.97	NH	0	Medical expenses	Accepted
3	250000.0	6690.0	12.89	AL	10	Debt consolidation	Accepted
4	30000.0	7640.0	0.58	WA	9	Major purchase	Accepted
...
1081515	30000.0	5350.0	1.54	CT	5	Medical expenses	Rejected
1081516	60000.0	5800.0	5.55	TX	0	Car financing	Rejected
1081517	30000.0	5610.0	0.00	OR	0	Debt consolidation	Rejected
1081518	25000.0	5730.0	8.55	NY	0	Other	Rejected
1081519	25000.0	5670.0	0.00	AL	2	Car financing	Rejected

1034236 rows x 7 columns

```
In [31]: # Turning Loan status to a binary variable
model_data['Loan_Status'].replace('Accepted', 1, inplace=True)
model_data['Loan_Status'].replace('Rejected', 0, inplace=True)
```

```
In [32]: new_corr = model_data.corr()
f, ax = plt.subplots(figsize=(9,6))
sns.heatmap(new_corr, vmax=.8, square=True, cmap="BuPu", annot=True)
```



It looks like all correlation are correctly assumed

Best Correlations:

Employment Length

Risk Score

Debt to Income Ratio

Model building

```
In [33]: # identify dependent and independent variable
y = model_data.Loan_Status
X = model_data.drop('Loan_Status', axis=1)
# might decide to drop state
```

```
In [34]: X = pd.get_dummies(X)
```

```
In [35]: X
```

	Amount_Requested	Risk_Score	debt_to_income_ratio	Emp_length	State_AK	State_AL	State_AR	...
0	50000.0	6690.0	21.80	8	0	0	0	...
1	150000.0	7040.0	18.29	2	0	0	0	...
2	112000.0	6690.0	43.97	0	0	0	0	...
3	250000.0	6690.0	12.89	10	0	1	0	...
4	30000.0	7640.0	0.58	9	0	0	0	...
...
1081515	30000.0	5350.0	1.54	5	0	0	0	...
1081516	60000.0	5800.0	5.55	0	0	0	0	...
1081517	30000.0	5610.0	0.00	0	0	0	0	...
1081518	25000.0	5730.0	8.55	0	0	0	0	...
1081519	25000.0	5670.0	0.00	2	0	1	0	...

1034236 rows x 70 columns

```
In [36]: # Split to test/train split and scale
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

Stochastic Gradient Descent (SGD) classifier

This classifier has the advantage of being capable of handling very large datasets efficiently. This is in part because SGD deals with training instances independently.

Let's create an SGDClassifier and train it on the whole training set:

```
In [37]: from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train)
```

```
Out[37]: SGDClassifier(random_state=42)
```

```
In [101]: # for testing purposes - Lets see how good/bad the model is at predicting
y_train_acc = y_train == 1
```