In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [8]:

```
import pandas as pd
df = pd.read_csv(r"C:\Users\hsalka\Downloads\data\loans_FINAL.csv")
df
```

Out[8]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	day
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	563
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	276
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	471
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	269
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	406
5	1	credit_card	0.0788	125.13	11.904968	16.98	727	612
6	1	debt_consolidation	0.1496	194.02	10.714418	4.00	667	318
7	1	all_other	0.1114	131.22	11.002100	11.08	722	511
8	1	home_improvement	0.1134	87.19	11.407565	17.25	682	398
9	1	debt_consolidation	0.1221	84.12	10.203592	10.00	707	273
10	1	debt_consolidation	0.1347	360.43	10.434116	22.09	677	671
11	1	debt_consolidation	0.1324	253.58	11.835009	9.16	662	429
12	1	debt_consolidation	0.0859	316.11	10.933107	15.49	767	651
13	1	small_business	0.0714	92.82	11.512925	6.50	747	438
14	1	debt_consolidation	0.0863	209.54	9.487972	9.73	727	155
15	1	major_purchase	0.1103	327.53	10.738915	13.04	702	815
16	1	all_other	0.1317	77.69	10.522773	2.26	672	389
17	1	credit_card	0.0894	476.58	11.608236	7.07	797	651
18	1	debt_consolidation	0.1039	584.12	10.491274	3.80	712	276
19	1	major_purchase	0.1513	173.65	11.002100	2.74	667	112
20	1	all_other	0.0800	188.02	11.225243	16.08	772	488

21	credit1.policy	all_other purpose	int0.0863.rate	installment474.42	log10.819778.annual.inc	2.59dti	fico797	day119
22	1	credit_card	0.1355	339.60	11.512925	7.94	662	193
23	1	credit_card	0.0788	484.85	11.736069	7.05	782	564
24	1	debt_consolidation	0.1229	320.19	11.264464	8.80	672	376
25	1	all_other	0.0901	159.03	12.429216	10.00	712	155
26	1	all_other	0.0743	155.38	11.082143	0.28	802	464
27	1	debt_consolidation	0.1375	255.43	9.998798	14.29	662	131
28	1	all_other	0.0743	155.38	12.206073	0.28	772	451
29	1	all_other	0.0743	155.38	12.206073	3.72	812	677
...	...	...	...	...	...	...	...	...
9548	0	home_improvement	0.1607	87.99	10.778956	14.20	667	408
9549	0	home_improvement	0.2164	729.70	11.877569	8.63	667	828
9550	0	all_other	0.1459	137.86	10.085809	1.15	732	123
9551	0	home_improvement	0.1348	508.87	11.736069	16.85	707	744
9552	0	debt_consolidation	0.1311	337.45	10.691945	23.62	702	378
9553	0	debt_consolidation	0.1385	545.67	11.775290	10.80	697	411
9554	0	small_business	0.1533	870.71	11.842229	16.16	707	423
9555	0	home_improvement	0.1311	674.90	12.292250	9.94	717	573
9556	0	debt_consolidation	0.1385	136.42	11.002100	18.18	677	342
9557	0	credit_card	0.1025	466.35	12.206073	13.97	722	612
9558	0	debt_consolidation	0.1533	696.57	11.805595	17.21	682	279
9559	0	credit_card	0.1273	688.11	11.314475	21.13	732	588
9560	0	all_other	0.1867	547.36	11.407565	15.76	667	100
9561	0	all_other	0.0788	115.74	10.999095	10.17	722	441
9562	0	debt_consolidation	0.1348	508.87	10.933107	17.76	717	387
9563	0	debt_consolidation	0.1099	556.50	11.225243	17.84	727	684
9564	0	all_other	0.1385	511.56	12.323856	12.33	687	642
9565	0	all_other	0.1459	396.35	10.308953	21.04	697	339
9566	0	all_other	0.2164	551.08	11.002100	24.06	662	180
9567	0	all_other	0.1311	101.24	10.968198	8.23	687	279
9568	0	all_other	0.1979	37.06	10.645425	22.17	667	591
9569	0	home_improvement	0.1426	823.34	12.429216	3.62	722	323
9570	0	all_other	0.1671	113.63	10.645425	28.06	672	321
9571	0	all_other	0.1568	161.01	11.225243	8.00	677	72

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	day
9572	0	debt_consolidation	0.1565	69.98	10.110472	7.02	662	819
9573	0	all_other	0.1461	344.76	12.180755	10.39	672	104
9574	0	all_other	0.1253	257.70	11.141862	0.21	722	438
9575	0	debt_consolidation	0.1071	97.81	10.596635	13.09	687	345
9576	0	home_improvement	0.1600	351.58	10.819778	19.18	692	180
9577	0	debt_consolidation	0.1392	853.43	11.264464	16.28	732	474

9578 rows × 14 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries,  0 to 9577
Data columns (total 14 columns):
credit.policy          9578 non-null int64
purpose                9578 non-null object
int.rate               9578 non-null float64
installment            9578 non-null float64
log.annual.inc         9574 non-null float64
dti                    9578 non-null float64
fico                   9578 non-null int64
days.with.cr.line      9549 non-null float64
revol.bal              9578 non-null int64
revol.util             9516 non-null float64
inq.last.6mths         9549 non-null float64
delinq.2yrs            9549 non-null float64
pub.rec                9549 non-null float64
not.fully.paid         9578 non-null int64
dtypes: float64(9), int64(4), object(1)
memory usage: 1.0+ MB
```

In [4]:

```
df.head()
```

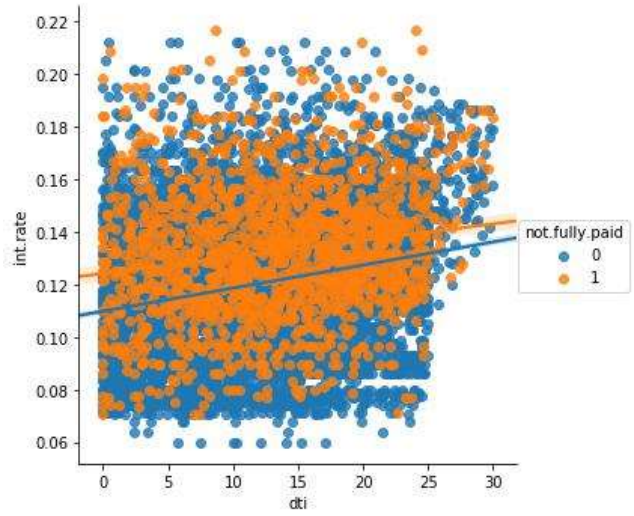
Out[4]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.wi
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.95
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.00
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.00
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.95
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.00

In [5]:

```
plt.figure(figsize = (15,8))
plot = sns.lmplot(x = "dti", y = "int.rate", data = df, hue = "not.fully.paid")
```

```
<Figure size 1080x576 with 0 Axes>
```



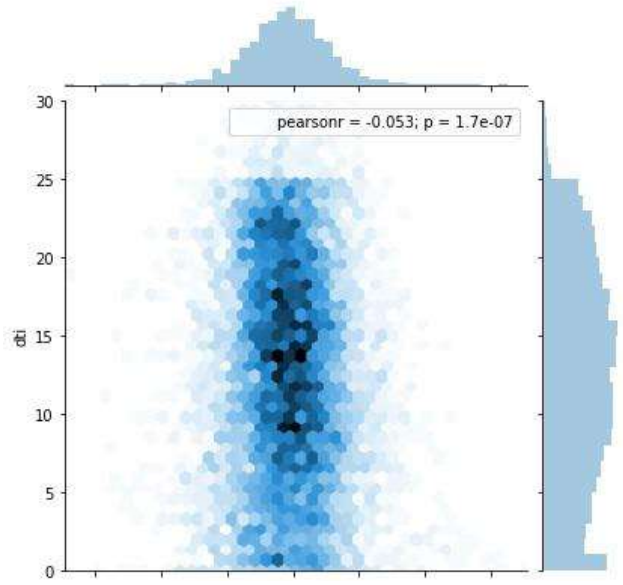
In [6]:

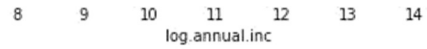
```
sns.jointplot(x = "log.annual.inc", y = "dti", data = df, kind = 'hex')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[6]:

```
<seaborn.axisgrid.JointGrid at 0x197dbacee10>
```

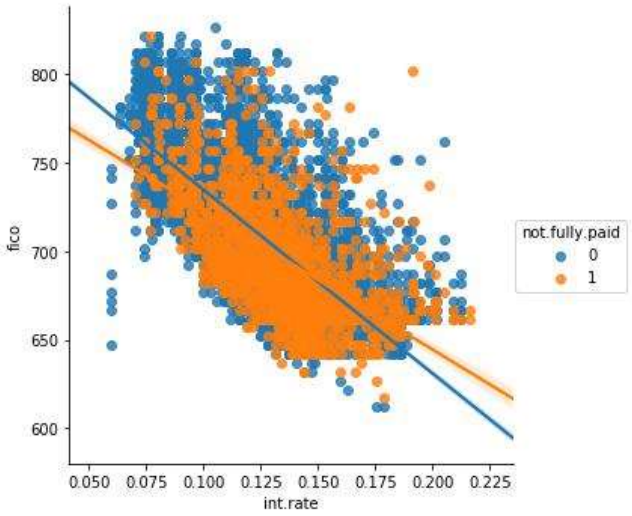




In [7]:

```
plt.figure(figsize = (15,8))
plot = sns.lmplot(x = "int.rate", y = "fico", data = df, hue = "not.fully.paid")
```

```
<Figure size 1080x576 with 0 Axes>
```

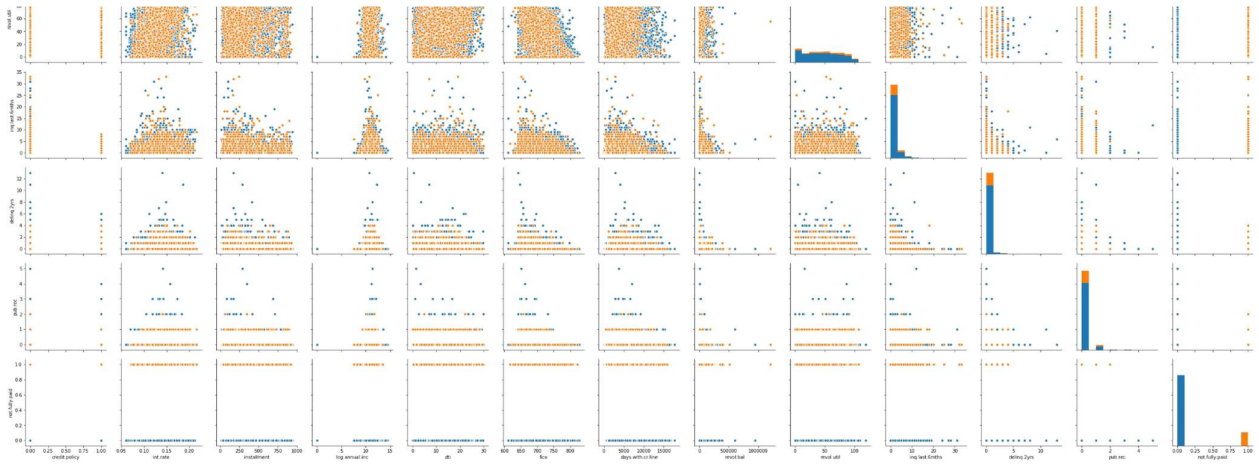


In [45]:

```
sns.pairplot(df, hue = 'not.fully.paid', size = 3)
```

Out[45]:

```
<seaborn.axisgrid.PairGrid at 0x197ee6b3da0>
```

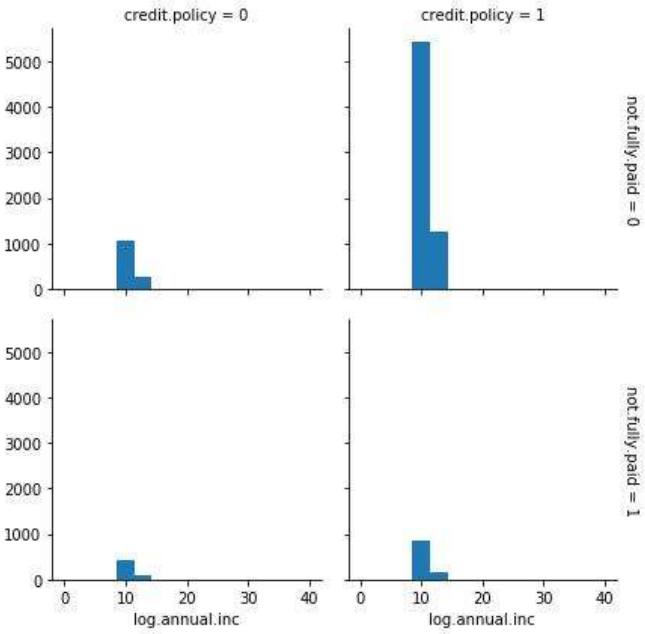


In [10]:

```
g = sns.FacetGrid(df, row = "not.fully.paid", col = "credit.policy", margin_titles = True)
g.map(plt.hist, "log.annual.inc", bins = np.linspace(0, 40, 15))
```

Out[10]:

```
<seaborn.axisgrid.FacetGrid at 0x197eb3e5828>
```



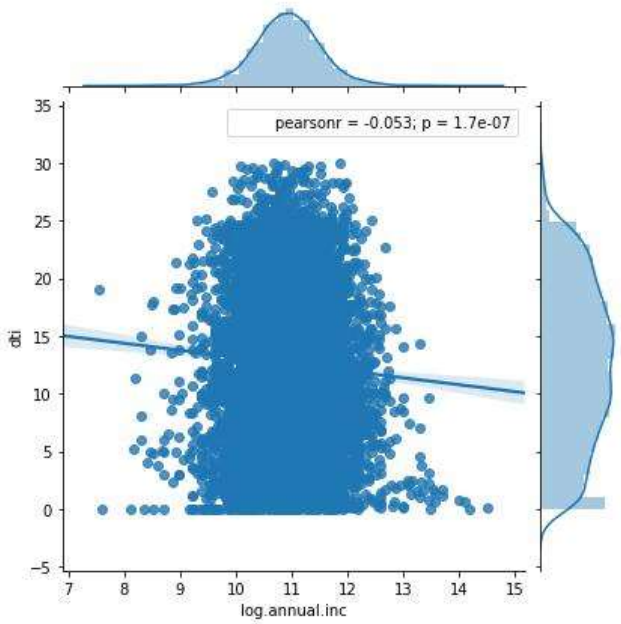
In [11]:

```
sns.jointplot(x = "log.annual.inc", y = "dti", data = df, kind = 'reg')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "

: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[11]:

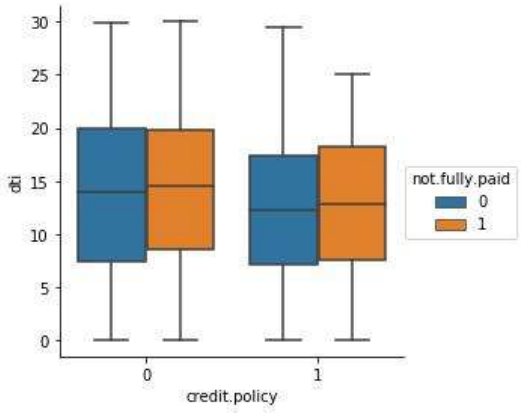
```
<seaborn.axisgrid.JointGrid at 0x197ed17d240>
```



In [12]:

```
l = sns.factorplot("credit.policy", "dti", "not.fully.paid", data = df, kind = "box")
l.set_axis_labels("Credit Policy", "Debt to Income Ratio", "FactorPlot of Distribution
of Debt to Income Ratio for Credit Policy")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-752aa08ff787> in <module>()
      1 l = sns.factorplot("credit.policy", "dti", "not.fully.paid", data = df, kind
= "box")
----> 2 l.set_axis_labels("Credit Policy", "Debt to Income Ratio", "FactorPlot of Dis
tribution of Debt to Income Ratio for Credit Policy")

TypeError: set_axis_labels() takes from 1 to 3 positional arguments but 4 were given
```

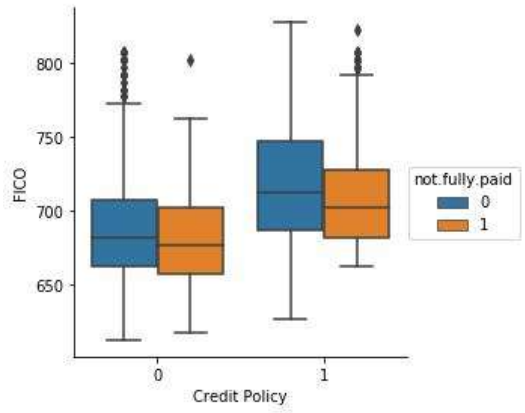


In [13]:

```
l = sns.factorplot("credit.policy", "fico", "not.fully.paid", data = df, kind = "box"
)
l.set_axis_labels("Credit Policy", "FICO")
```

Out[13]:

```
<seaborn.axisgrid.FacetGrid at 0x197ee464048>
```

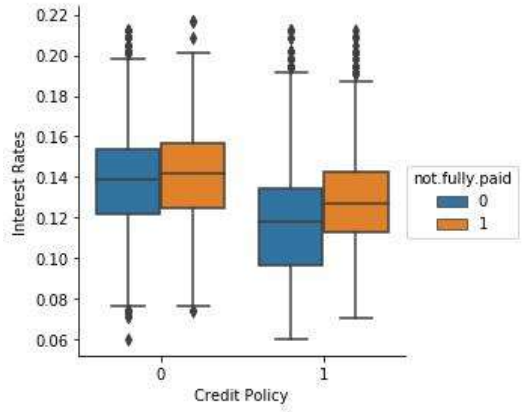


In [14]:

```
l = sns.factorplot("credit.policy", "int.rate", "not.fully.paid", data = df, kind = "
box")
l.set_axis_labels("Credit Policy", "Interest Rates")
```

Out[14]:

```
<seaborn.axisgrid.FacetGrid at 0x197ee4d1ac8>
```



Here I would like to figure out how many null values in the data and if they are significant enough by using isnull(). As shown below the numbers are not extremely significant and, as a result, I can choose to fill them in with a space or the mean/median value of the columns. I have decided to fill them in with the mean using the python function fillna()

In [16]:

```
df.isnull().sum()
```

Out[16]:

```
credit.policy         0
purpose               0
int.rate              0
```

```
installment            0
log.annual.inc         4
dti                    0
fico                   0
days.with.cr.line     29
revol.bal              0
revol.util            62
inq.last.6mths        29
delinq.2yrs           29
pub.rec               29
not.fully.paid         0
dtype: int64
```

In [17]:

```
df.fillna(df.mean())
```

Out[17]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	day
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	563
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	276
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	471
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	269
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	406
5	1	credit_card	0.0788	125.13	11.904968	16.98	727	612
6	1	debt_consolidation	0.1496	194.02	10.714418	4.00	667	318
7	1	all_other	0.1114	131.22	11.002100	11.08	722	511
8	1	home_improvement	0.1134	87.19	11.407565	17.25	682	398
9	1	debt_consolidation	0.1221	84.12	10.203592	10.00	707	273
10	1	debt_consolidation	0.1347	360.43	10.434116	22.09	677	671
11	1	debt_consolidation	0.1324	253.58	11.835009	9.16	662	429
12	1	debt_consolidation	0.0859	316.11	10.933107	15.49	767	651
13	1	small_business	0.0714	92.82	11.512925	6.50	747	438
14	1	debt_consolidation	0.0863	209.54	9.487972	9.73	727	155
15	1	major_purchase	0.1103	327.53	10.738915	13.04	702	815
16	1	all_other	0.1317	77.69	10.522773	2.26	672	389
17	1	credit_card	0.0894	476.58	11.608236	7.07	797	651
18	1	debt_consolidation	0.1039	584.12	10.491274	3.80	712	276
19	1	major_purchase	0.1513	173.65	11.002100	2.74	667	112
20	1	all_other	0.0800	188.02	11.225243	16.08	772	48

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	day
21	1	all_other	0.0863	474.42	10.819778	2.59	797	119
22	1	credit_card	0.1355	339.60	11.512925	7.94	662	193
23	1	credit_card	0.0788	484.85	11.736069	7.05	782	564
24	1	debt_consolidation	0.1229	320.19	11.264464	8.80	672	376
25	1	all_other	0.0901	159.03	12.429216	10.00	712	155
26	1	all_other	0.0743	155.38	11.082143	0.28	802	464
27	1	debt_consolidation	0.1375	255.43	9.998798	14.29	662	131
28	1	all_other	0.0743	155.38	12.206073	0.28	772	451
29	1	all_other	0.0743	155.38	12.206073	3.72	812	677
...	...	...	...	...	...	...	...	...
9548	0	home_improvement	0.1607	87.99	10.778956	14.20	667	408
9549	0	home_improvement	0.2164	729.70	11.877569	8.63	667	828
9550	0	all_other	0.1459	137.86	10.085809	1.15	732	123
9551	0	home_improvement	0.1348	508.87	11.736069	16.85	707	744
9552	0	debt_consolidation	0.1311	337.45	10.691945	23.62	702	378
9553	0	debt_consolidation	0.1385	545.67	11.775290	10.80	697	411
9554	0	small_business	0.1533	870.71	11.842229	16.16	707	423
9555	0	home_improvement	0.1311	674.90	12.292250	9.94	717	573
9556	0	debt_consolidation	0.1385	136.42	11.002100	18.18	677	342
9557	0	credit_card	0.1025	466.35	12.206073	13.97	722	612
9558	0	debt_consolidation	0.1533	696.57	11.805595	17.21	682	279
9559	0	credit_card	0.1273	688.11	11.314475	21.13	732	588
9560	0	all_other	0.1867	547.36	11.407565	15.76	667	100
9561	0	all_other	0.0788	115.74	10.999095	10.17	722	441
9562	0	debt_consolidation	0.1348	508.87	10.933107	17.76	717	387
9563	0	debt_consolidation	0.1099	556.50	11.225243	17.84	727	684
9564	0	all_other	0.1385	511.56	12.323856	12.33	687	642
9565	0	all_other	0.1459	396.35	10.308953	21.04	697	339
9566	0	all_other	0.2164	551.08	11.002100	24.06	662	180
9567	0	all_other	0.1311	101.24	10.968198	8.23	687	279
9568	0	all_other	0.1979	37.06	10.645425	22.17	667	591
9569	0	home_improvement	0.1426	823.34	12.429216	3.62	722	323
9570	0	all_other	0.1671	113.63	10.645425	28.06	672	321

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	day
9571	0	all_other	0.1568	161.01	11.225243	8.00	677	723
9572	0	debt_consolidation	0.1565	69.98	10.110472	7.02	662	819
9573	0	all_other	0.1461	344.76	12.180755	10.39	672	104
9574	0	all_other	0.1253	257.70	11.141862	0.21	722	438
9575	0	debt_consolidation	0.1071	97.81	10.596635	13.09	687	345
9576	0	home_improvement	0.1600	351.58	10.819778	19.18	692	180
9577	0	debt_consolidation	0.1392	853.43	11.264464	16.28	732	474

9578 rows × 14 columns

Next We will evaluate the number or percentage of people in this dataset who have or have not paid back the loan in full. If they have not paid back the loan, they would have a '1' in the not.fully.paid column or a '0' if they did fully pay back. Only 16% of the people have not fully paid back.

In [25]:

```
df.groupby('not.fully.paid').mean()
num_of_ones = (df['not.fully.paid'] == 1).sum()
num_of_zeros = (df['not.fully.paid'] != 1).sum()
prop_ones = (num_of_ones / (num_of_ones + num_of_zeros)) * 100
print(num_of_ones)
print(num_of_zeros)
print(prop_ones)
```

```
1533
8045
16.005429108373356
```

Next we will divide the data into training and test sets with knowledge concerning the not.fully.paid column as that is what we wish to predit/create a logistic regression model. Currently, we are using the 70%/30% priciple to divide the loans dataset. To help, I have imported the something from sklearn to split the data for me. I was experiencing issues at first while doing this, however, stumbled across the fact that one of the features, purpose, is categorical. I did n ot include this in the indep variable for train/test. I also took care of some nan values that for some reason were still in the dataset

In [21]:

```
indep_ var = df[['credit.policy','int.rate', 'installment', 'log.annual.inc', 'dti', '
fico', 'days.with.cr.line', 'revol.bal', 'revol.util', 'inq.last.6mths', 'delinq.2yrs
', 'pub.rec']]
dep_var = df['not.fully.paid']
df[['credit.policy','int.rate', 'installment', 'log.annual.inc', 'dti', 'fico', 'days
.with.cr.line', 'revol.bal', 'revol.util', 'inq.last.6mths', 'delinq.2yrs', 'pub.rec' ]] =
df[['credit.policy','int.rate', 'installment', 'log.annual.inc', 'dti', 'fico',
```

```
'days.with.cr.line', 'revol.bal', 'revol.util', 'inq.last.6mths', 'delinq.2yrs', 'pub
.rec']].fillna(0)
df[['credit.policy','int.rate', 'installment', 'log.annual.inc', 'dti', 'fico', 'days
.with.cr.line', 'revol.bal', 'revol.util', 'inq.last.6mths', 'delinq.2yrs', 'pub.rec'
]].isnull().sum()
df['not.fully.paid'] = df['not.fully.paid'].fillna(0)
df['not.fully.paid'].isnull().sum()
```

Out[21]:

```
0
```

Here used the sklearn function found in documentation: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html to divide 70/30 training and test data. There is also an implementation of logistic regression on unscaled training data below:

In [265]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
#Split Data using test_split
indep_ var_train, indep_var _test, dep_var_train, dep_var_test =
train_test_split(indep _var, dep_var, test_size = 0.3, random_state = 130)
indep_var_train.shape

#Logistic regression on the unscaled training data with all columns
reg = LogisticRegression()
reg.fit(indep_var_train, dep_var_train)

#Parameter
a = reg.get_params()
print(a)

#Function Parameters
par = reg.coef_
print(par)

#Finding the Accuracy using score function
score = reg.score(indep_var_test, dep_var_test)
print(score)

#Using predict and classification report to implement precision, recall, and f1-scor
e
pred_of_dep = reg.predict(indep_var_test)
report = classification_report(dep_var_test, pred_of_dep)
print(report)
```

```
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_sca
ling': 1, 'max_iter': 100, 'multi_class': 'ovr', 'n_jobs': 1, 'penalty': 'l2', 'rando
m_state': None, 'solver': 'liblinear', 'tol': 0.0001, 'verbose': 0, 'warm_start': Fal
se}
```

```
[[ 0.29522525 -0.07144138 -0.00705662  1.11042886  0.03221686  0.08960756
   0.13604877  0.05752726  0.00260927  0.03380929  0.16915648 -0.07525497
   1.78174461  0.06137697 -0.05746979 -0.12627367 -0.11294449 -0.19016788
   0.01739149  0.05584491]]
 0.9066666666666666
              precision    recall  f1-score   support

           0       0.94      0.88      0.91       153
           1       0.88      0.94      0.91       147

 avg / total       0.91      0.91      0.91       300
```

In [156]:

```
indep_var_test.shape, dep_var_test.shape
```

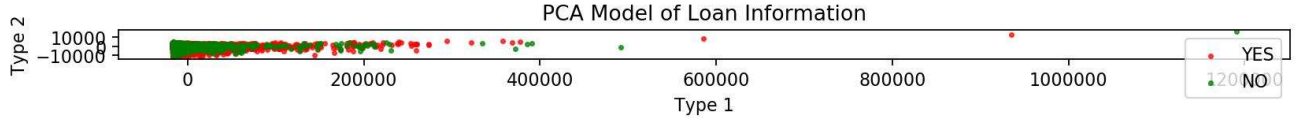
Out[156]:

```
((2874, 12), (2874,))
```

PCA Model to better Visualize Logistic Regression Implemented above

In [362]:

```
from sklearn.decomposition import PCA
data = df.drop('purpose', axis=1)
all_data = data.iloc[:,1:]
not_paid = df.iloc[:,-1]

graph = PCA(n_components=2).fit_transform(all_data)
all_train, all_test, pay_train, pay_test = train_test_split(graph, not_paid,
random_state=100)

#Plot
plt.figure(dpi=160)
plt.scatter(graph[not_paid.values==0,0], graph[not_paid.values==0,1], alpha=0.7, labe
l='YES', s=5, color='red')
plt.scatter(graph[not_paid.values==1,0], graph[not_paid.values==1,1], alpha=0.7, labe
l='NO', s=5, color='green')
plt.legend()
plt.title('PCA Model of Loan Information')
plt.xlabel('Type 1')
plt.ylabel('Type 2')
plt.gca().set_aspect('equal')
plt.show()
```



Here we recieve an 91% accuracy with the listed classification report and parameter. The

steps to increase accuracy are as follows: 1: Scale the Data 2: Find optimum parameters using random or Grid Search 3: Implement different Classication models instead of logistic regression 4: Use feature scaling to find the top ranked features and implement those ones specifically

Feature Scaling and Normalization Using the Library sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html This implementation was done after the first trial to improve accuracy

In [270]:

```
from sklearn.preprocessing import StandardScaler
indep_ var_train, indep_var _test, dep_var_train, dep_var_test =
train_test_split(indep _var, dep_var, test_size = 0.3, random_state = 130) scale =
StandardScaler()
new_scaled_indep_var_train = scale.fit_transform(indep_var_train)
new_scaled_indep_var_test = scale.fit_transform(indep_var_test)
```

Using the PCA - We can improve Logistic Regression and tweak it. Before, logistic regression was too slow (estimation > 1 minute). Therefore, we can improve this by applying PCA to make the model go faster, better visualize the data, and incrwease the precision

In [275]:

```
from sklearn.decomposition import PCA
from sklearn.linearmodel import LogisticRegression
# Make an instance of the Model
pca = PCA(.85)
pca.fit(new_scaled_indep_var_train)

new_scaled_indep_var_train = pca.transform(new_scaled_indep_var_train)
new_scaled_indep_var_test = pca.transform(new_scaled_indep_var_test)

# 'lbfgs' so it moves faster
Regr = LogisticRegression(solver = 'lbfgs')

Regr.fit(new_scaled_indep_var_train, dep_var_train)

Regr.predict(new_scaled_indep_var_test[0].reshape(1,-1))

Regr.predict(new_scaled_indep_var_test[0:9])

Regr.score(new_scaled_indep_var_test, dep_var_test)
```

Out[280]:

```
0.8399443284620738
```

After we have scaled the data like above, we now implement logistic regression one more time and compute the accuracy of the model. Unfortunately, the accuracy of the model

stayed relatively the same. Therefore, we will continue to establish new implementations to increase the accuracy of the model.

In [271]:

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

#Logistic regression on the scaled data with all columns
log = LogisticRegression()
log.fit(new_scaled_indep_var_train, dep_var_train)

#Accuracy of newly scaled data
score = log.score(new_scaled_indep_var_test, dep_var_test)
print(score)

#Parameter
b = log.get_params()
print(b)

#Function Parameters
para = log.coef_
print(para)

#Using predict and classification report to implement precision, recall, and f1-scor
e
pred_of_dep = log.predict(new_scaled_indep_var_test)
report = classification_report(dep_var_test, pred_of_dep)
print(report)
```

```
0.91
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_sca
ling': 1, 'max_iter': 100, 'multi_class': 'ovr', 'n_jobs': 1, 'penalty': 'l2', 'rando
m_state': None, 'solver': 'liblinear', 'tol': 0.0001, 'verbose': 0, 'warm_start': Fal
se}
[[ 0.40787451 -0.07057281 -0.00770694  1.56553829  0.03166561  0.08982892
   0.1342854   0.05512562  0.00323899  0.03122193  0.16677645 -0.07456801
   2.08589006  0.06175621  0.01496563 -0.13043532 -0.10832297 -0.18588641
   0.01741625  0.05540212]]
              precision    recall  f1-score   support

           0       0.95      0.87      0.91       153
           1       0.88      0.95      0.91       147

 avg / total       0.91      0.91      0.91       300
```

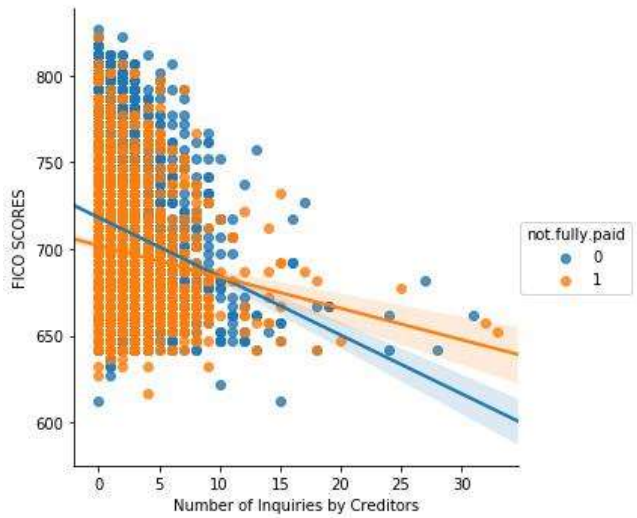
Graph

In [150]:

```
h = sns.lmplot(x = "inq.last.6mths", y = "fico", hue = "not.fully.paid", data = df, l
egend = True)
plt.xlabel("Number of Inquiries by Creditors")
plt.ylabel("FICO SCORES")
```

Out[150]:

```
Text(31.24,0.5,'FICO SCORES')
```



Now, we will implement the Grid Search method to find the optimum parameters

Judging from the implementation below, the best param is 1.0

In [260]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
params = np.array([0, 0.0001, 0.001, 0.01, 0.1, 1])
search = Ridge()
grid = GridSearchCV(estimator=search, param_grid=dict(alpha=params))
grid.fit(indep_var, dep_var)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_.alpha)
```

```
GridSearchCV(cv=None, error_score='raise',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'alpha': array([0.e+00, 1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn', scoring=None,
       verbose=0)
0.03082013814996594
1.0
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\ridge.py:112: LinAlgW
arning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number5.165064e-17
  overwrite_a=True).T
```

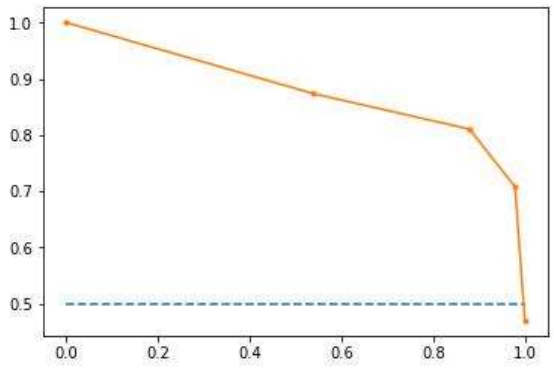
ROC Curve

In [261]:

```
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score from
sklearn.metrics import auc
from sklearn.metrics import average_precision_score
from matplotlib import pyplot
indep_var, dep_var = make_classification(n_samples=1000, n_classes=2, weights=[1,1],
random_state=1)
indep_ var_train, indep_var _test, dep_var_train, dep_var_test =
train_test_split(indep _var, dep_var, test_size = 0.3, random_state = 2)
KNN = KNeighborsClassifier(n_neighbors=3)
KNN.fit(indep_var_train, dep_var_train)
probability = KNN.predict_proba(indep_var_test)
probability = probability[:, 1]
```

```
pred = KNN.predict(indep_var_test)

precision, recall, thresholds = precision_recall_curve(dep_var_test, probability)

fscore = f1_score(dep_var_test, pred)
print(fscore)

accur = auc(recall, precision)
print(accur)

avprec = average_precision_score(dep_var_test, probability)
print(avprec)

pyplot.plot([0, 1], [0.5, 0.5], linestyle='--')

pyplot.plot(recall, precision, marker='.')

pyplot.show()
```

```
0.8435374149659864
0.8794715401570954
0.8270245226220369
```



Random Forest Model

In [276]:

```
from sklearn.ensemble import RandomForestClassifier
indep_ var_train, indep_var _test, dep_var_train, dep_var_test =
train_test_split(indep _var, dep_var, test_size = 0.3, random_state = 130)
def RFC(x, y):
    forest = RandomForestClassifier()
    forest.fit(x, y)
    return forest
train = RFC(indep_var_train,dep_var_train)
p = train.predict(indep_var_test)
```

In [277]:

```
params = train
print(params)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [278]:

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(dep_var_test,p))
```

```
             precision    recall  f1-score   support

          0       0.92      0.86      0.89       153
          1       0.86      0.92      0.89       147

avg / total       0.89      0.89      0.89       300
```

In [279]:

```
print(confusion_matrix(dep_var_test,p))
```

```
[[131  22]
 [ 12 135]]
```

In [280]:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(dep_var_train, train.predict(indep_var_train))
print(accuracy)
```

```
0.9928571428571429
```

Tweaking the Decision Forest parameters (play around with them to see if I can improve accuracy - I cannot)

In [281]:

```
Try_one = DecisionTreeClassifier(criterion = "gini", random_state = 300, max_depth=7,
min_samples_leaf=5)
Try_one.fit(indep_var_train,dep_var_train)
dep_pred_one = Try_one.predict(indep_var_test)
acc1 = accuracy_score(dep_var_test, dep_pred_one)
print(acc1)

Try_two = DecisionTreeClassifier(criterion = "entropy", random_state = 300,max_depth=
7, min_samples_leaf=5)
Try_two.fit(indep_var_train,dep_var_train)
dep_pred_two = Try_two.predict(indep_var_test)
acc2 = accuracy_score(dep_var_test, dep_pred_two)
print(acc2)
```

```
0.8666666666666667
0.85
```

Decision Tree Model

In [283]:

```
from sklearn.tree import DecisionTreeClassifier
indep_ var_train, indep_var _test, dep_var_train, dep_var_test =
train_test_split(indep _var, dep_var, test_size = 0.3, random_state = 130)
def DCT(x, y):
    decision = DecisionTreeClassifier()
    decision.fit(x, y)
    return decision
```

```
call = DCT(indep_var_train,dep_var_train)
pred = call.predict(indep_var_test)

call
```

Out[283]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [285]:

```
from sklearn.metrics import classification_report,confusion_matrix

print(classification_report(dep_var_test,pred))
```

```
             precision    recall  f1-score   support

          0       0.81      0.84      0.82       153
          1       0.82      0.80      0.81       147

avg / total       0.82      0.82      0.82       300
```

In [216]:

```
print(confusion_matrix(dep_var_test,pred))
```

```
[[1976  450]
 [ 338  110]]
```

In [236]:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(dep_var_train, call.predict(indep_var_train))
print(accuracy)
```

```
1.0
```

This accuracy above could possibly be an error in the actual way the functions were implemented above, the data wasn't split properly, or just a lucky coincidence. Nonetheless, it does require further consideration. Also, it is notable to say that although the validation is very high this doesn't mean the data overfit but it could be due to one of the reasons described above such as a malfunction or incorrect implementation. Due to this, I will not make it my "best" model.

Due to the number of columns as well as the fact that logistic regression didn't work on all columns, we will choose 2 of the highest ranked features using RFE and then use those to

implemenmt a another model. Below is the Recursive Feature Elimination. Note the choosing of two features was done to improve accuracy by using RFE to achieve a better model

In [254]:

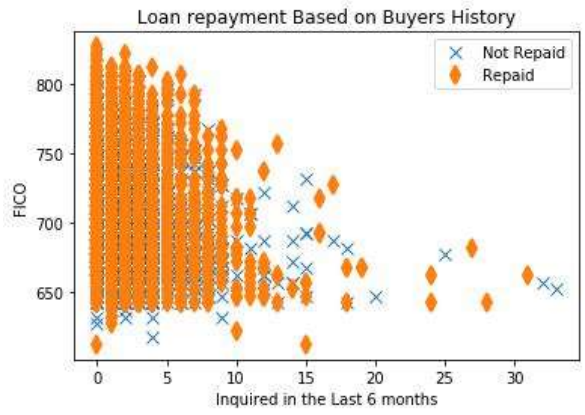
```
#Reursive Feature Elimination
scale = RFE(log, 2)
scale = scale.fit(new_scaled_indep_var_train, dep_var_train)
print(scale.ranking_)
#Based on the output FICO and inq.last.6mths were the top 2 features of importances
and will therefore be used in the implementation below for logistic regression from
sklearn.preprocessing import StandardScaler scale = StandardScaler()

ind_train = df[['fico', 'inq.last.6mths']]
scaled_ind_train = scale.fit_transform(ind_train)
scaled_ind_train = scaled_ind_train.reshape(-1,1)
```

```
[ 5  4  2  3  9  1 11  6 10  1  7  8]
```

Graphs of Scaled and Unscaled Data of Two Features

In [213]:

```
X = np.array([np.ones(len(df['inq.last.6mths'])), df['inq.last.6mths'].values, df['fi
co'].values]).T
Y = np.array([df['not.fully.paid'].values]).reshape(len(df['inq.last.6mths']),1)

ax = plt.gca()
ax.plot(df[df['not.fully.paid'] == 1]['inq.last.6mths'], df[df['not.fully.paid'] == 1
]['fico'], 'x', ms=5.0)
ax.plot(df[df['not.fully.paid'] == 0]['inq.last.6mths'], df[df['not.fully.paid'] == 0
]['fico'], 'd', mew=3, ms=5.0)
ax.set_title('Loan repayment Based on Buyers History')

ax.set_xlabel('Inquired in the Last 6 months')
ax.set_ylabel('FICO')
ax.legend(['Not Repaid', 'Repaid'], loc='upper right', frameon=True)
f = plt.gcf()
plt.show()
```

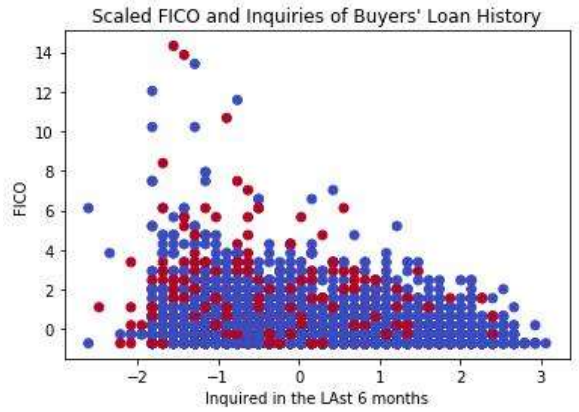


In [304]:

```
from sklearn.preprocessing import StandardScaler
cale = StandardScaler()
FICO = df[['fico']]
IL6M = df[['inq.last.6mths']]
FICO = cale.fit_transform(FICO)
IL6M = cale.fit_transform(IL6M)
y = df[['not.fully.paid']]
plt.scatter(FICO, IL6M, c = y, cmap = plt.cm.coolwarm)
plt.title("Scaled FICO and Inquiries of Buyers' Loan History")
plt.xlabel("Inquired in the LAst 6 months")
plt.ylabel("FICO")
```

Out[304]:

```
Text(0,0.5,'FICO')
```



Naive Bayes Implementation

In [325]:

```
X_ax = df.loc[:,['fico','inq.last.6mths']].values
Y_ax = df.loc[:,['not.fully.paid']].values
#Split accordingly to the two features we want to use
X_ax_train, X_ax_ test, Y_ax_train, Y_ax_test = train_test_split(X_ax, Y_ax,
test_size = 0.3, random_state = 130)
#Scale them
new = StandardScaler()
new_X_train = scale.fit_transform(X_ax_train)
new_X_test = scale.fit_transform(X_ax_test)
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
guess = NB.fit(new_X_train, Y_ax_train.ravel())
l = NB.predict(new_X_test)
print(classification_report(Y_ax_test,l))
```

	precision	recall	f1-score	support
0	0.85	0.96	0.90	2426

1	0.32	0.10	0.15	448
avg / total	0.77	0.83	0.79	2874