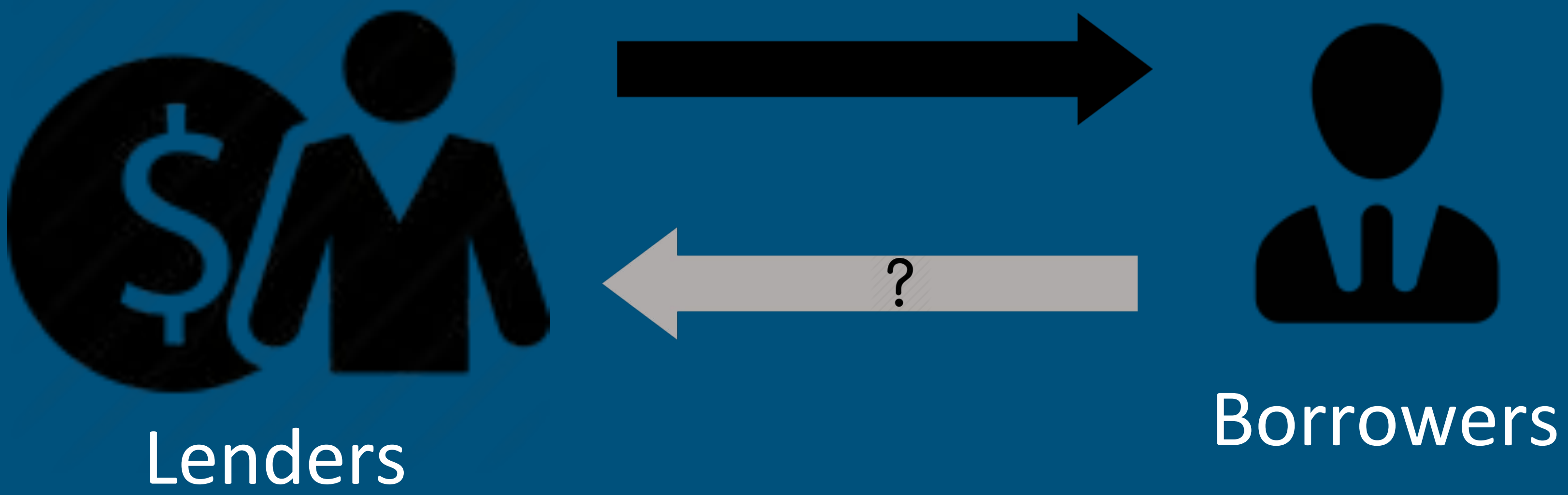
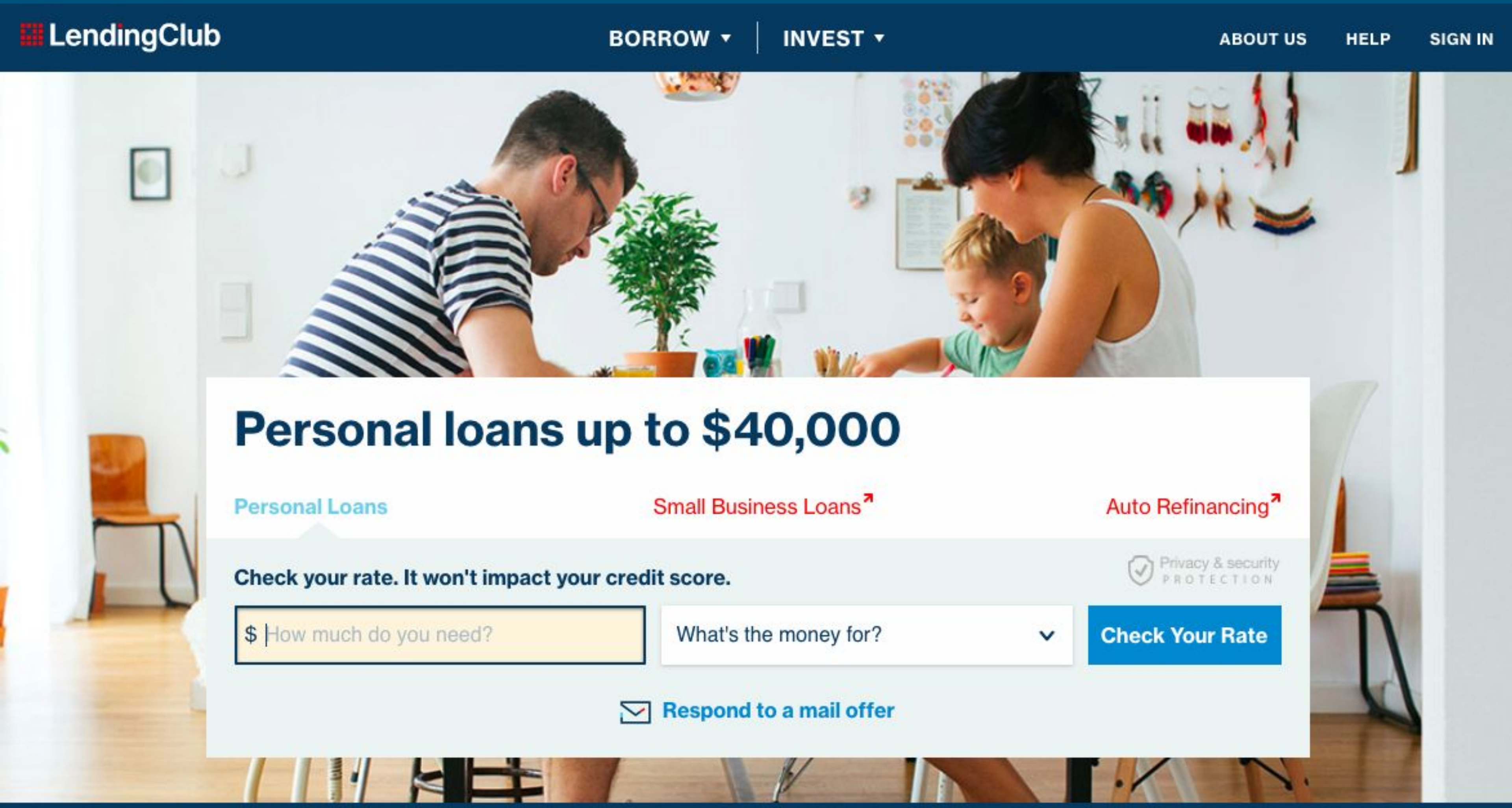
Predicting Loan Defaults

Kenneth Chee, Chen Wanlun, Fong Yew Loong, Tan Jiaqi

Problem Statement



Source of Data



Data

- Loan data for all loans issued through the period 2008 to 2011.

- Contains over 50 dimensions and 20,000 observations!

Variables

- Dependent Variable:
    - Loan status - Charged Off or Fully Paid.

- Independent Variable:
    - Quantitative: Loan Amount, Number of accounts the borrowers is now delinquent, employment length (and more!)
    - Categorical: Term of Loan, Purpose of Loan

Data Cleaning – Removing Variables

- Original dataset with 51 variables
- Removed 30 variables
- 21 variables remaining

Data Cleaning – Removing Variables

- Variables with no predictive value (all observations with same value):
  - Eg. Loan application type, Policy code, Initial list status, Payment plan
- Variables with sparse data (only a handful of observations with values recorded):
  - Eg. Months since most recent rating, Months since last delinquency, Months since last record

Data Cleaning – Removing Variables

- Variables with all NAs:
  - Eg. URL
- Variables with text data:
  - Eg. Description by borrower of loan, Title of loan
- Variables with address data:
  - Zipcode is censored so data not meaningful
- Variables with meaningless data for future predictions:
  - Eg. Issue date

Data Cleaning – Removing Variables

- Variables with data that borrower/bank cannot know before taking up loan:
  - Eg. Last payment date, Total payment to date, Principal received to date
- Variables that assume loanee has already defaulted:
  - Eg. Recoveries, Collection Recovery Fee
- Other variables:
  - Subgrade (already part of grade categories)
  - Earliest Credit Line (irrelevant)

Data Cleaning – Dealing with Categorical Variables

- Transformation into dummies

In [77]:
```
df3_grade=pd.get_dummies(df3['grade'])
df3_grade.head()
```

Out[77]:

	A	B	C	D	E	F	G
0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0
3	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0

Data Cleaning – Dealing with Percentage Variables

- Transformation into numeric

```
In [4]: df3['int_rate']

Out[4]: 0        10.65%
        1        15.27%
        2        15.96%
        3         7.90%
        4        18.64%
        5        21.28%
        6        12.69%
        7        13.49%
        8        10.65%
        9        16.29%
        10       15.27%
        11        6.03%
        12       11.71%
        13        6.03%
        14       12.42%
        15       11.71%
```

```
In [75]: df3['int_rate'] = df3['int_rate'].replace({'\%':''}, regex = True)
         df3['revol_util'] = df3['revol_util'].replace({'\%':''}, regex = True)
```

Data Cleaning – Others

- Scaling variables

```
In [107]: df3['annual_inc']= df3['annual_inc']/1000
          df3.head()
```

Out[107]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	emp_length	home_ownership	annual_inc	verification_status	loan_status
0	5000	5000	4975.0	36 months	10.65	162.87	B	10+ years	RENT	24.000	Verified	0
1	2500	2500	2500.0	60 months	15.27	59.83	C	< 1 year	RENT	30.000	Source Verified	1
2	2400	2400	2400.0	36 months	15.96	84.33	C	10+ years	RENT	12.252	Not Verified	0 sr
3	5000	5000	5000.0	36 months	7.90	156.46	A	3 years	RENT	36.000	Source Verified	0
4	3000	3000	3000.0	36 months	18.64	109.43	E	9 years	RENT	48.000	Source Verified	0

- Dropping rows with NA values

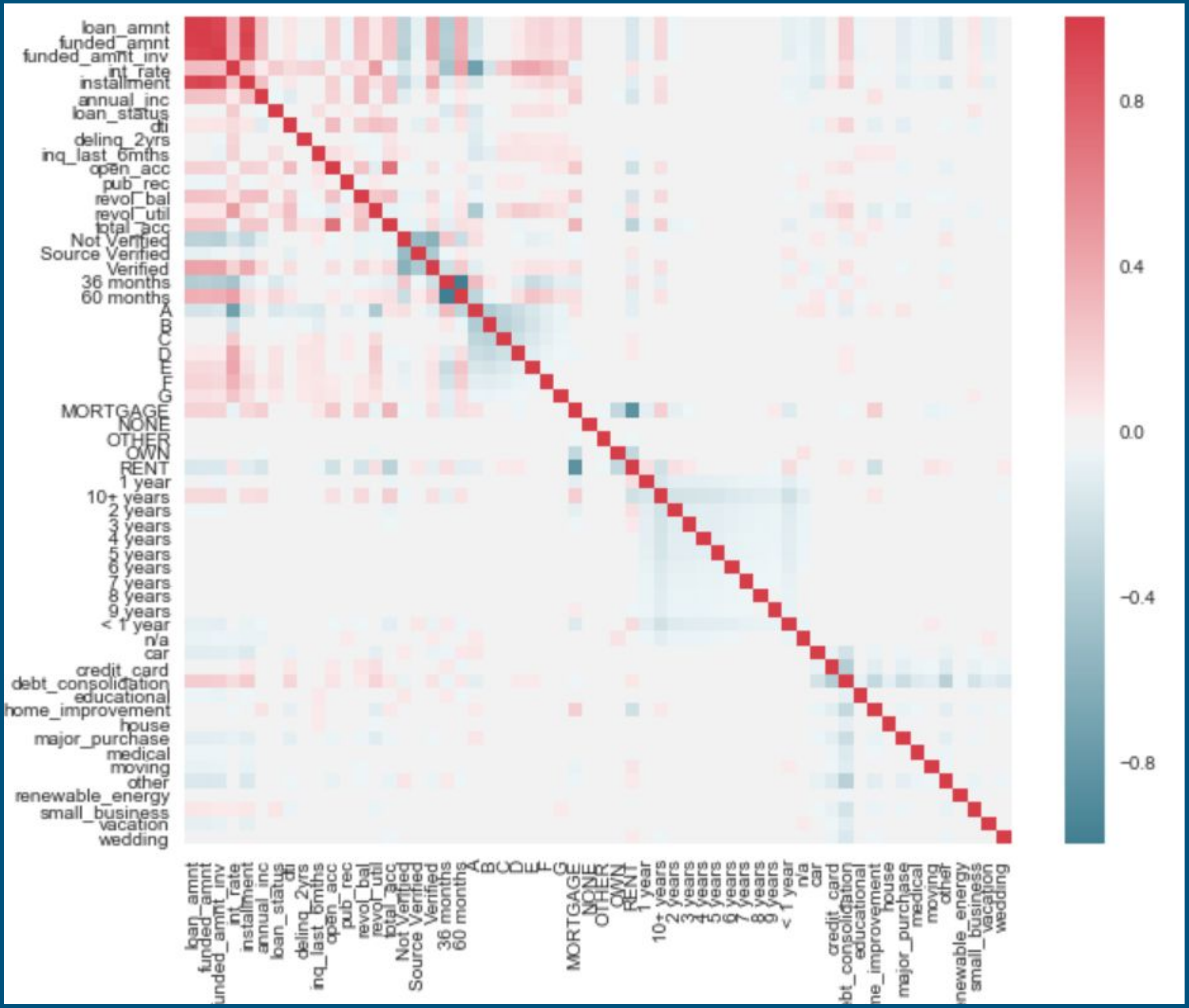
```
In [ ]: df3_perf = df3_perf.dropna(axis=0, how='any')
        df3_perf.head()
```

Data Cleaning – Standardisation of variables

- Important for feature selection later
- Need to determine which are most important variables

```
In [455]: from sklearn.linear_model import LinearRegression
          from sklearn.preprocessing import StandardScaler
          scaler = StandardScaler()
          X_scaled = scaler.fit_transform(X_final)
```

Data Exploration – Correlation Plot

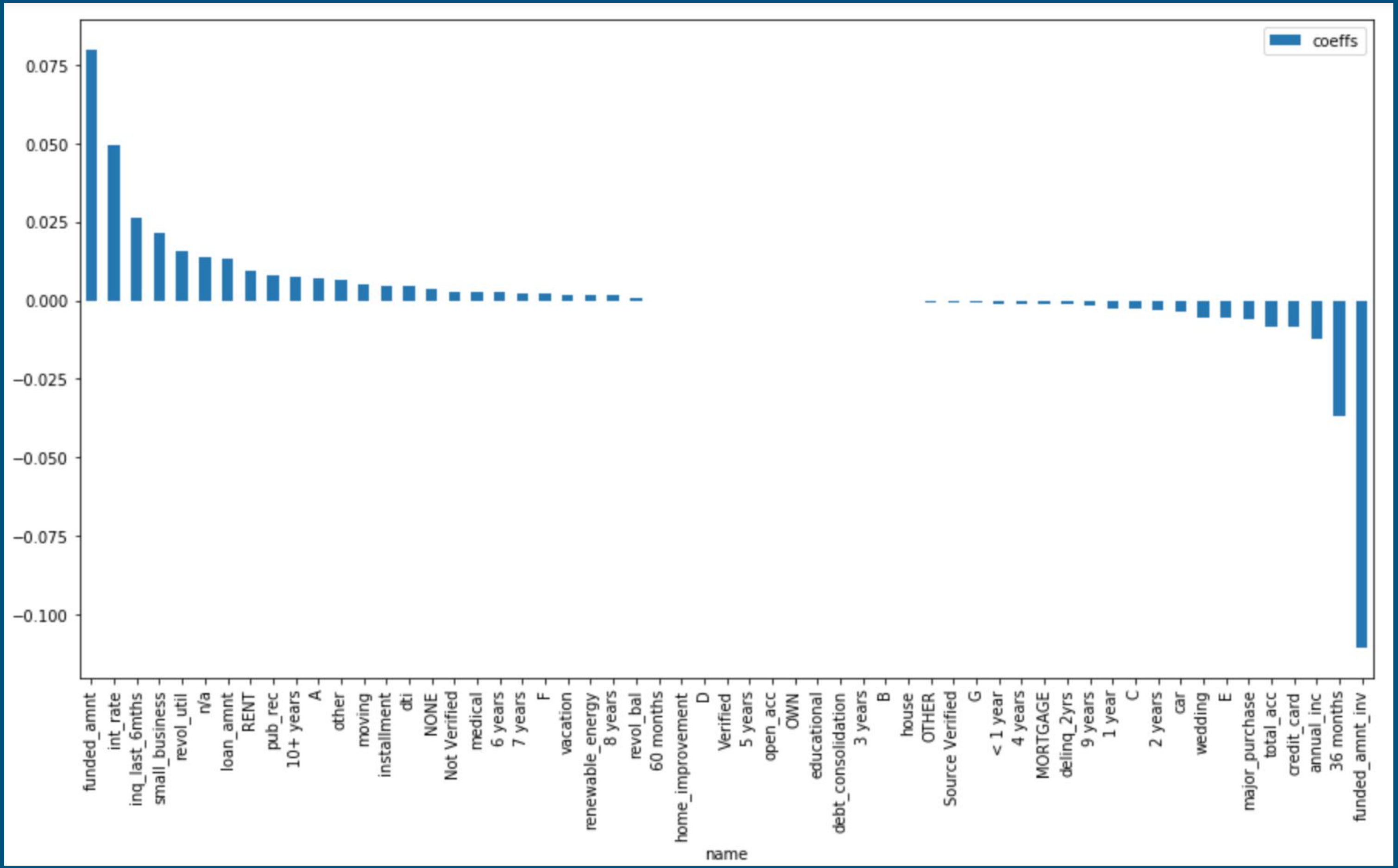


Data Exploration – Correlation Plot

- Defaulting most positively correlated to:
  - Installment
  - Incidents of deliquency
  - No of open credit lines
- Defaulting most negatively correlated to:
  - Short term loans
  - Good loan grades

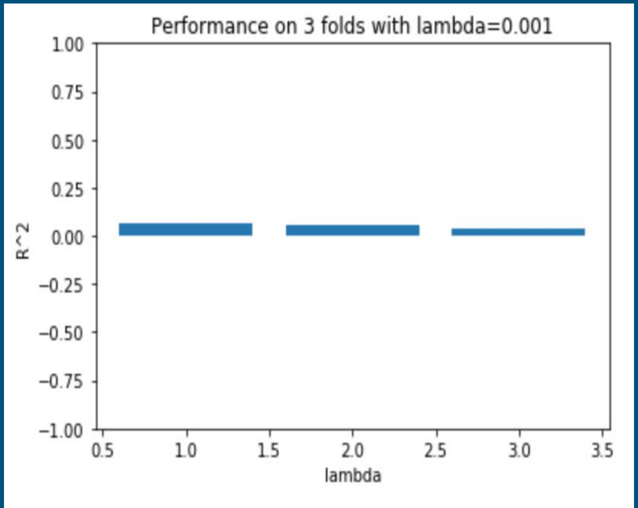
Features Selection

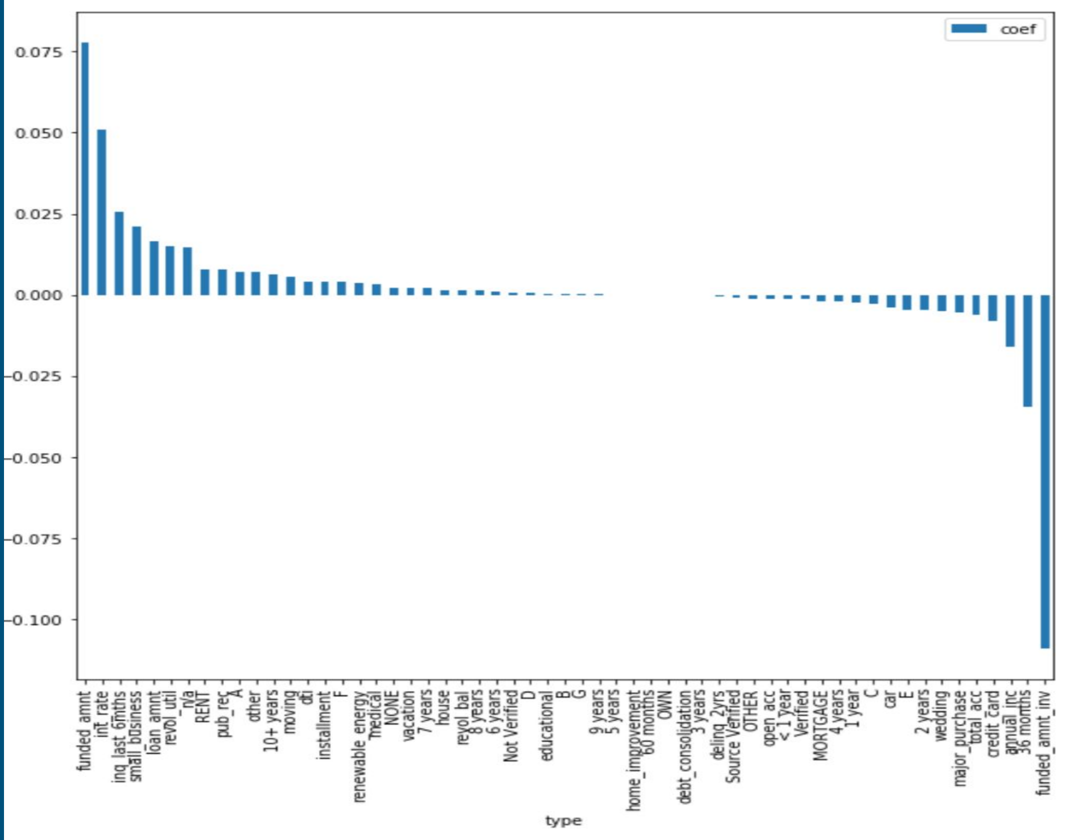
- Insignificant features like Verified, OWN



Cross Validation

- Across 3 folds





Combining categories

- Combined non-significant variables with significant ones
  - E.g. home ownership, verification status

```
df3 = df3.replace(['MORTGAGE', 'OTHER', 'NONE', 'RENT'], 'Do not own')
df3 = df3.replace('OWN', 'Own')
df3_home_ownership=pd.get_dummies(df3['home_ownership'])
df3_home_ownership = df3_home_ownership.drop(['Do not own'],1)

df3_home_ownership.head()
```

	Own
0	0
1	0
2	0
3	0
4	0

Combining categories

- Dropped reference categories for dummies
  - No value-add

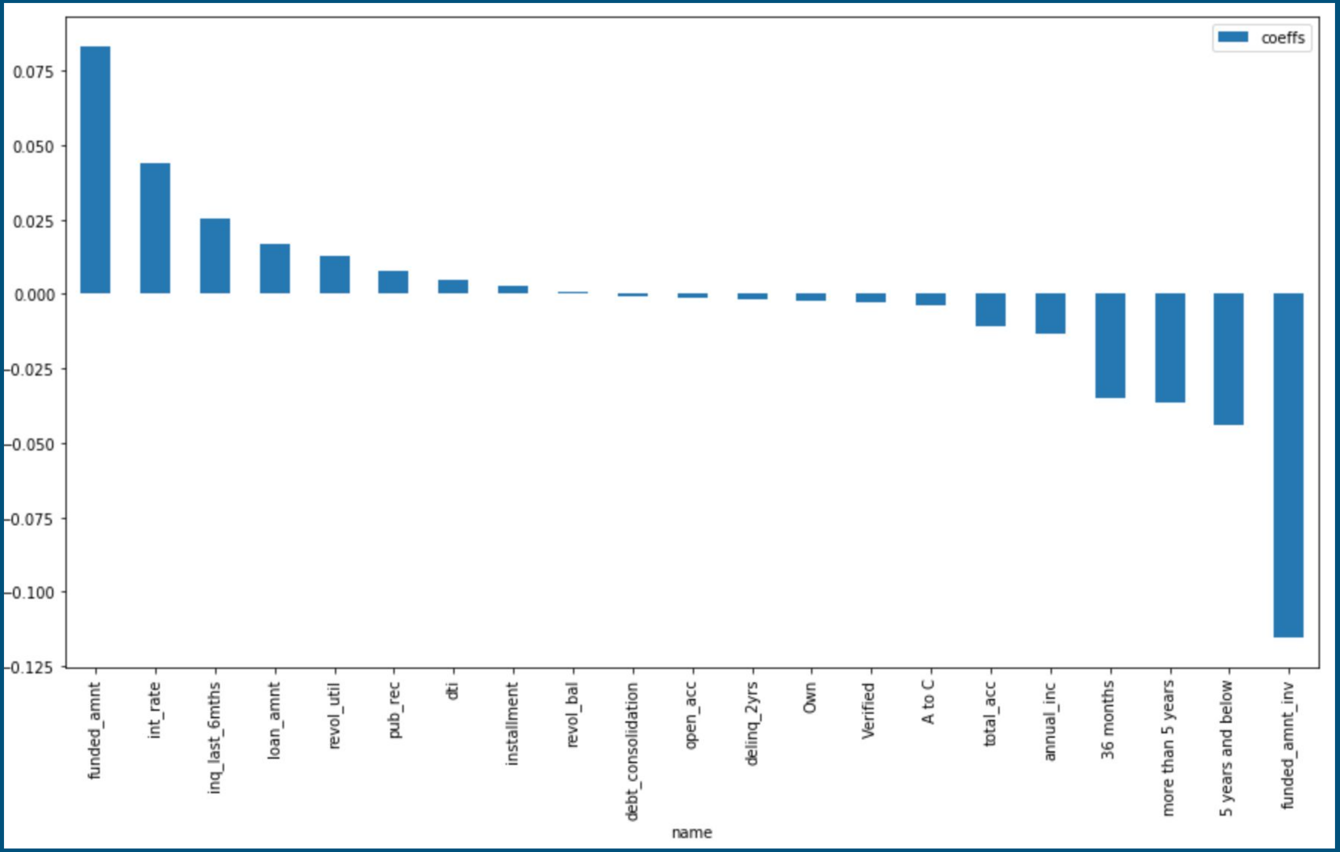
```
df3_term=pd.get_dummies(df3['term'])
df3_term = df3_term.drop([' 60 months'],1)

df3_term.head()
```

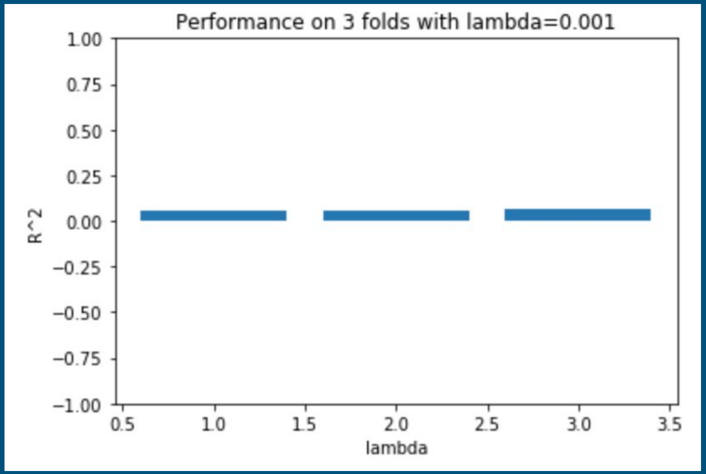
	36 months
0	1
1	0
2	1
3	1
4	1

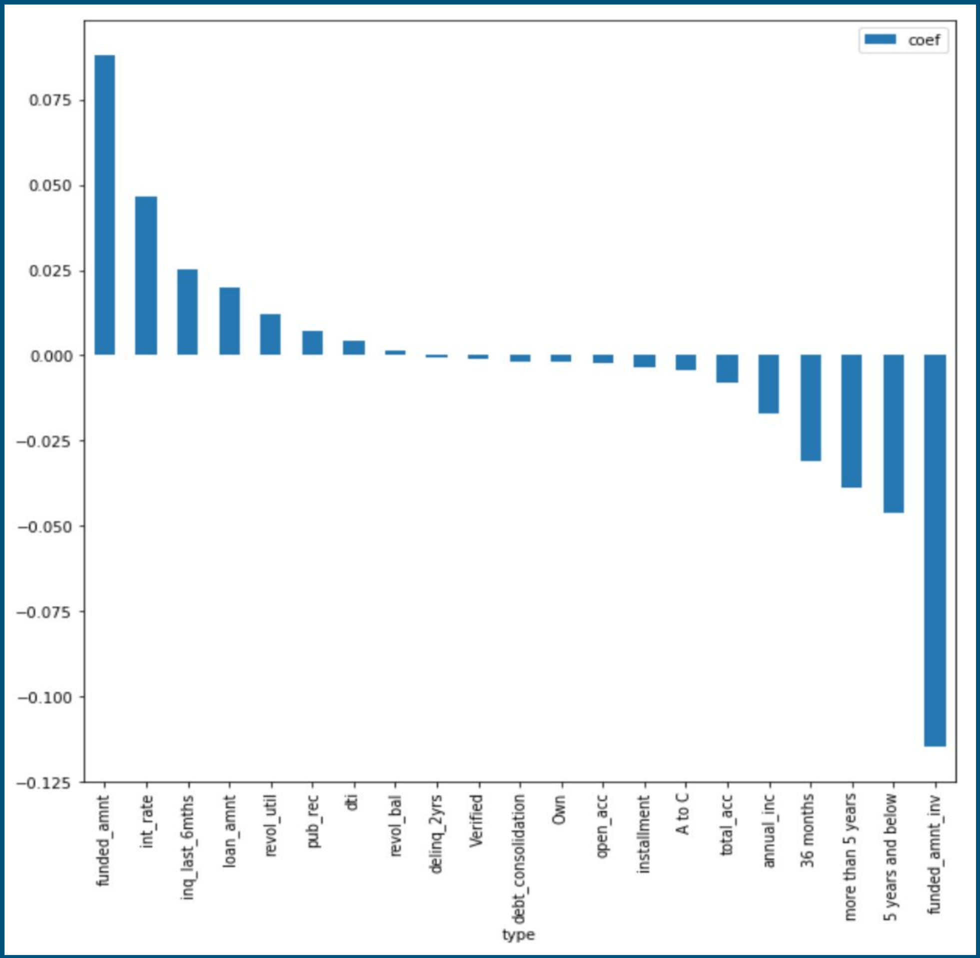
Validating Revised Categories

- All variables significant



Cross Validation 2





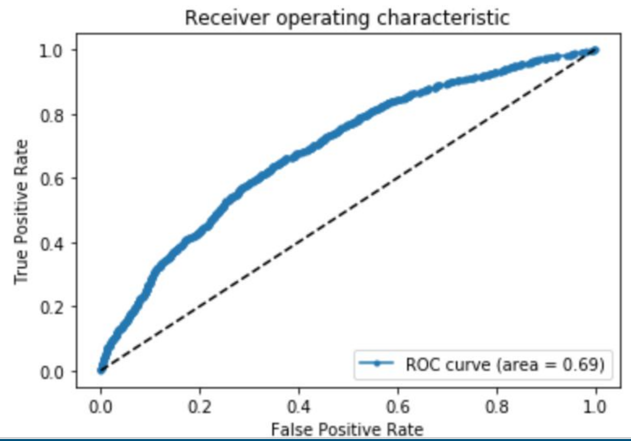
Baseline Logistic Model

- Optimise using Ridge regularisation

```
best_alpha=alphas[np.argmax(scores)]
# Generate ROC  for LR with l2 penalty and C=best_alpha
fpr,tpr,roc_auc, thresholds = generate_auc(X,y,LogisticRegression,C=best_alpha,penalty='l2')
def generate_ROCplot(fpr,tpr,label,roc_auc):
    plt.clf()
    plt.plot(fpr, tpr, '.-',label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()

# Plots ROC
generate_ROCplot(fpr,tpr,'LR',roc_auc)
```

```
Area under the ROC curve : 0.691122
```



Check for Robustness

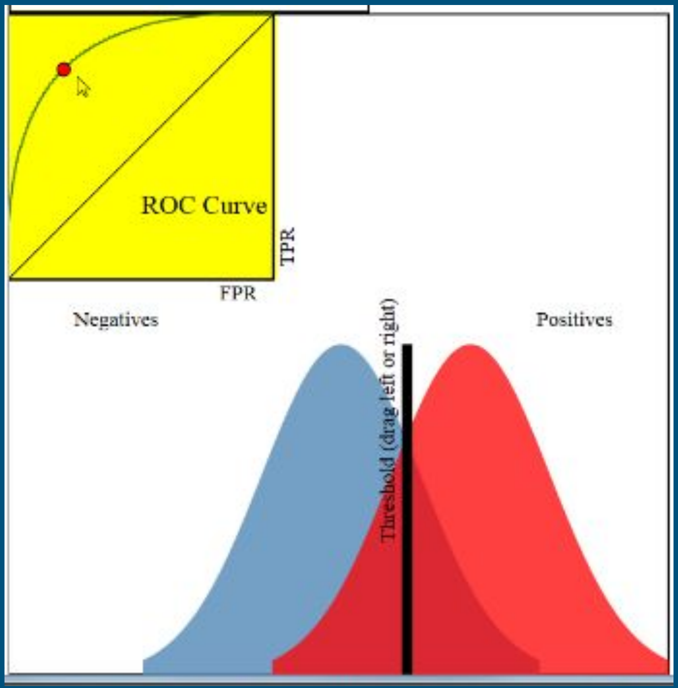
- Generalizable across 5 random folds
  - AUC ROC of around 0.69-0.70

```
regr2 = LogisticRegression(C=best_alpha,penalty='l2')
regr2.fit(X_train,y_train)
from sklearn.model_selection import cross_val_score
scores2 = cross_val_score(regr2, X_train, y_train, cv=5, scoring = "roc_auc")
scores2
```

```
array([ 0.69191326,  0.70022915,  0.70510038,  0.69118042,  0.69397267])
```

Selection Method - AUC ROC

- Binary Classifier
- Drawbacks of Using Prediction Accuracy
- Separate Distributions instead



Support Vector Machine

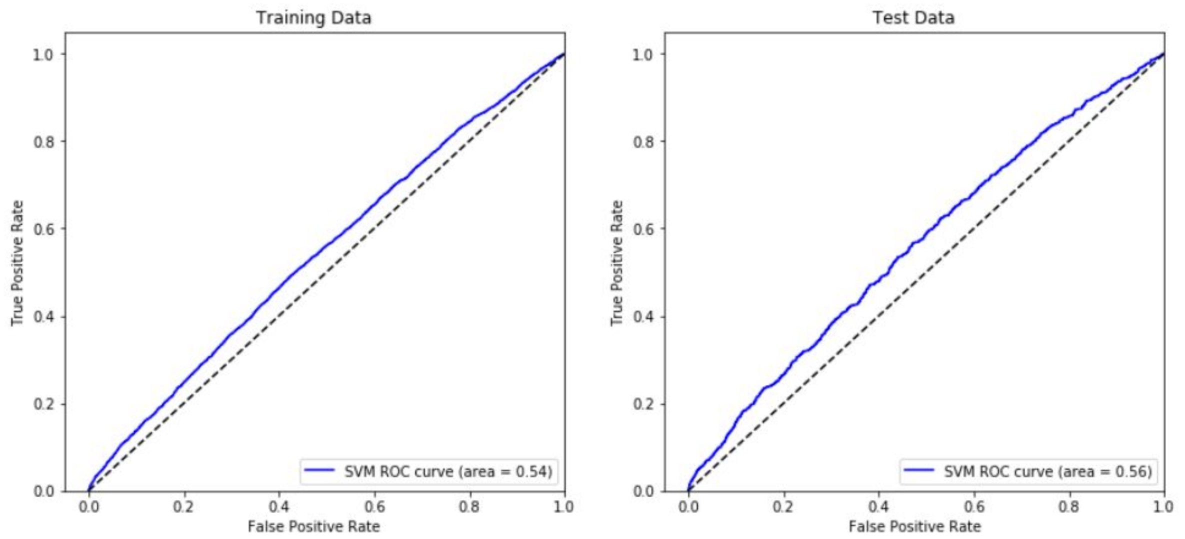
- 3 Parameters to select:
  - Penalization term for 'Slackness'
  - Kernel
  - Coefficient on Kernel

- GridsearchCV to find parameters

- Operating time: O(nfeatures x nobservations^3)

Support Vector Machine

- Smaller training set rbf kernel < Larger training set linear kernel



In [65]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, X, y, cv=3, scoring="roc_auc")
scores
```

Out[65]:
```
array([ 0.52356987,  0.54363675,  0.57254598])
```

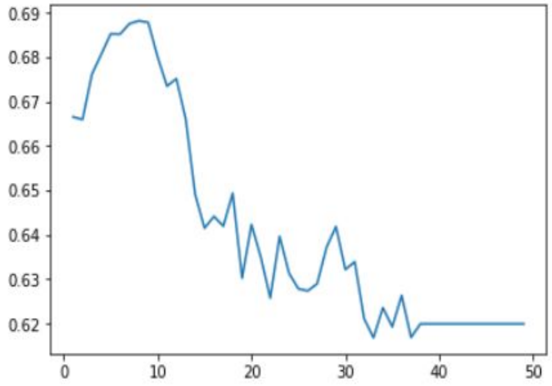
Random Forest

- 1 parameter: Optimal depth

```
In [73]: scores=[]
depths = range(1,50)
for n in depths:
    fpr, tpr, roc_auc, thresholds= generate_auc(X,y,RandomForestClassifier, max_depth=n, random_state=42)
    scores.append(roc_auc)

n_opt=depths[np.argmax(scores)]

plt.plot(depths,scores)
plt.show()
print('Optimal Decision Tree Depth: %.10f' % n_opt)
```



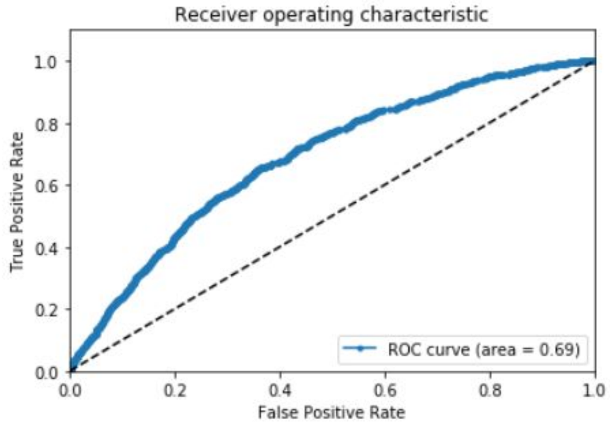

```
Optimal Decision Tree Depth: 8.0000000000
```

Random Forest

- Result

In [76]:
```python
# Generate ROC
fpr,tpr,roc_auc, thresholds = generate_auc(X,y,RandomForestClassifier, max_depth=n_opt, random_state=42)
def generate_ROCplot(fpr,tpr,label,roc_auc):
    plt.clf()
    plt.plot(fpr, tpr, '.-',label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()

# Plots ROC
generate_ROCplot(fpr,tpr,'LR',roc_auc)
print("Area under the ROC curve : %f" % roc_auc)
```




```
Area under the ROC curve : 0.688219
```

In [77]:
```python
# Cross Validate model
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, X, y, cv=5, scoring="roc_auc")
scores
```

Out[77]:
```
array([ 0.69228543,  0.67787542,  0.6776415 ,  0.65840256,  0.68749189])
```

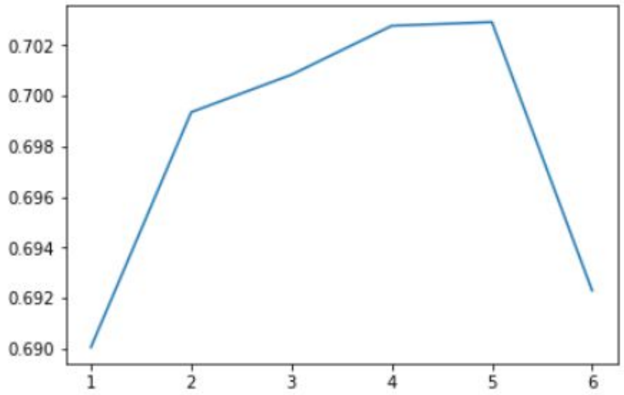
Boosting

- 2 parameters: Depth and Learning rate

```
In [82]: scores = []
depths = range(1,7)
for n in depths:
    fpr, tpr, roc_auc, thresholds = generate_auc(X, y, GradientBoostingClassifier, max_depth = n, random_state = 42)
    scores.append(roc_auc)


n_opt = depths[np.argmax(scores)]

plt.plot(depths,scores)
plt.show()
print("Optimal Boosting Tree Depth:", n_opt)
```




```
Optimal Boosting Tree Depth: 5
```

```
Learning rate:  0.0025
Accuracy score (training): 0.8495
Accuracy score (validation): 0.8533

Learning rate:  0.005
Accuracy score (training): 0.8548
Accuracy score (validation): 0.8529

Learning rate:  0.01
Accuracy score (training): 0.8623
Accuracy score (validation): 0.8520

Learning rate:  0.02
Accuracy score (training): 0.8737
Accuracy score (validation): 0.8529

Learning rate:  0.025
Accuracy score (training): 0.8805
Accuracy score (validation): 0.8526

Learning rate:  0.05
Accuracy score (training): 0.9136
Accuracy score (validation): 0.8484

Learning rate:  0.1
Accuracy score (training): 0.9634
Accuracy score (validation): 0.8473
```

Boosting

- Highest AUC ROC (although generally comparable)
- Out of curiosity...AUC vs Prediction Accuracy

```
Confusion Matrix:
[[4500    3]
 [ 771    3]]

Classification Report
             precision    recall  f1-score   support

          0       0.85      1.00      0.92      4503
          1       0.50      0.00      0.01       774

avg / total       0.80      0.85      0.79      5277
```

In [86]:

```
scores_gb = gb.decision_function(X_test)
fpr_gb, tpr_gb, _ = roc_curve(y_test, scores_gb)
roc_auc_gb = auc(fpr_gb, tpr_gb)

print("Area under ROC curve = {:0.4f}".format(roc_auc_gb))
```

```
Area under ROC curve = 0.6967
```

In [87]:

```
from sklearn.externals import joblib
joblib.dump(gb, 'loan_boosting.pkl')
```

Out[87]:

```
['loan_boosting.pkl']
```

Thank you