Introduction

Lending companies work by analyzing the financial history of their loan applicants and choosing whether the applicant is too risky to be given a loan. If the applicant is not, the company then determines the terms of the loan. To acquire these applicants, companies can organically receive them through their websites/apps, often with the help of advertisement campaigns. Other times, lending companies partner with peer-to-peer (P2P) lending marketplaces, in order to acquire leads of possible applicants. Some example marketplaces include Upstart, Lending Tree and Lending Club. In this project, we are going to assess the 'quality' of the leads our company receives from these marketplaces.

Market: The target audience is the set of loan applicants who reached out through an intermediary marketplace.

Product: A loan.

Goal: Develop a model to predict for 'quality' applicants. In this case study, 'quality' applicants are those who reach a key part of the loan application process.

Importing Essential Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import random
import time
```

In [2]:
```
dataset = pd.read_csv('Financial Data.csv')
```

Exploratory Data Analysis (EDA)

In [3]:
```
dataset.head()
```

Out[3]:

	entry_id	age	pay_schedule	home_owner	income	months_employed	years_employed	current_a
0	7629673	40	bi-weekly	1	3135	0	3	
1	3560428	61	weekly	0	3180	0	6	
2	6934997	23	weekly	0	1540	6	0	
3	5682812	40	bi-weekly	0	5230	0	6	
4	5335819	33	semi-monthly	0	3590	0	5	

5 rows × 21 columns

In [4]:
```
dataset.columns
```

Out[4]:
```
Index(['entry_id', 'age', 'pay_schedule', 'home_owner', 'income',
       'months_employed', 'years_employed', 'current_address_year',
```

```
       'personal_account_m', 'personal_account_y', 'has_debt',
       'amount_requested', 'risk_score', 'risk_score_2', 'risk_score_3',
       'risk_score_4', 'risk_score_5', 'ext_quality_score',
       'ext_quality_score_2', 'inquiries_last_month', 'e_signed'],
      dtype='object')
```

In [5]:
```
dataset.describe()
```

Out[5]:

	entry_id	age	home_owner	income	months_employed	years_employed
count	1.790800e+04	17908.000000	17908.000000	17908.000000	17908.000000	17908.000000
mean	5.596978e+06	43.015412	0.425173	3657.214653	1.186006	3.526860
std	2.562473e+06	11.873107	0.494383	1504.890063	2.400897	2.259732
min	1.111398e+06	18.000000	0.000000	905.000000	0.000000	0.000000
25%	3.378999e+06	34.000000	0.000000	2580.000000	0.000000	2.000000
50%	5.608376e+06	42.000000	0.000000	3260.000000	0.000000	3.000000
75%	7.805624e+06	51.000000	1.000000	4670.000000	1.000000	5.000000
max	9.999874e+06	96.000000	1.000000	9985.000000	11.000000	16.000000

In [6]:
```
dataset.isnull().sum()
```

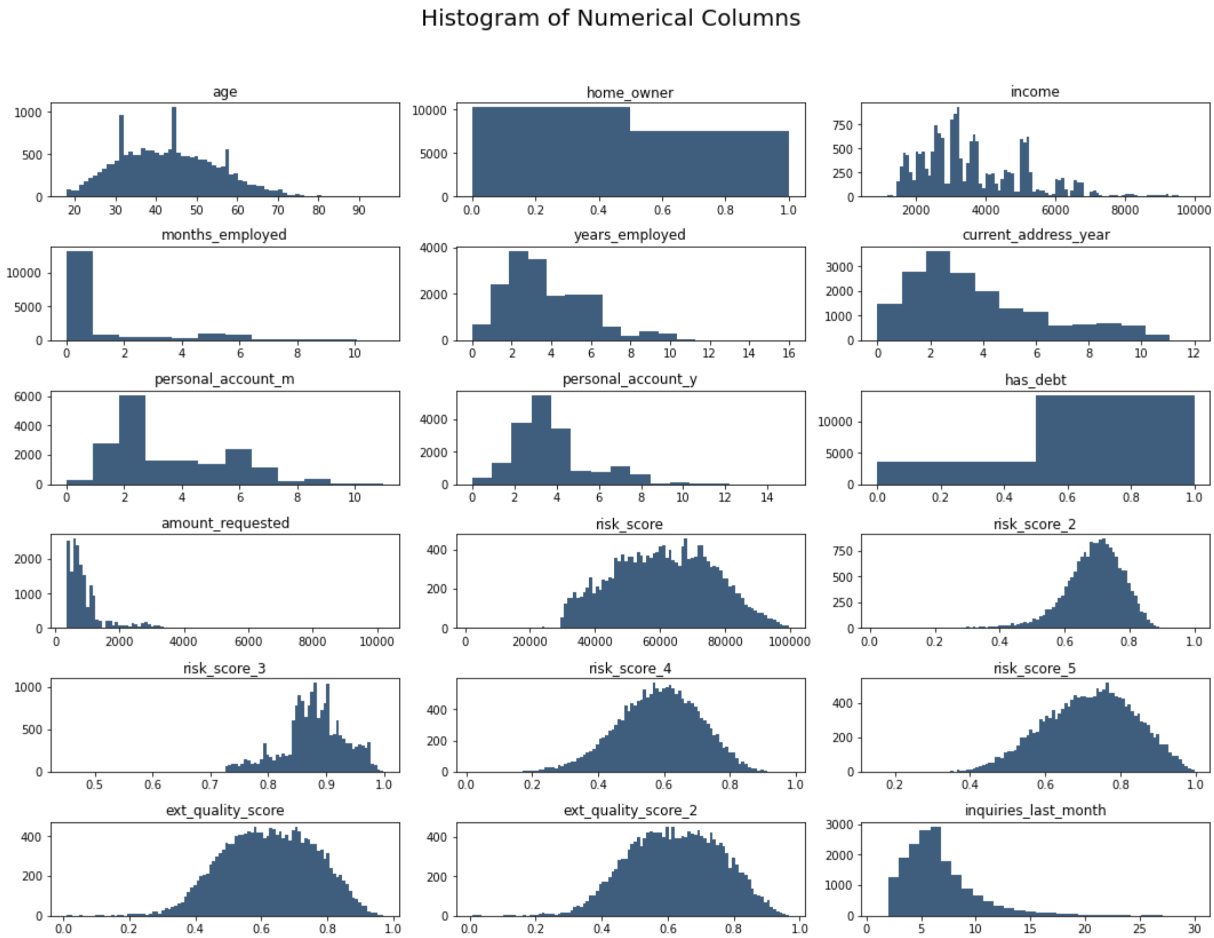
Out[6]:
```
entry_id                0
age                     0
pay_schedule            0
home_owner              0
income                  0
months_employed         0
years_employed          0
current_address_year    0
personal_account_m      0
personal_account_y      0
has_debt                0
amount_requested        0
risk_score              0
risk_score_2            0
risk_score_3            0
risk_score_4            0
risk_score_5            0
ext_quality_score       0
ext_quality_score_2     0
inquiries_last_month    0
e_signed                0
dtype: int64
```

NOTE that the data is already cleaned because these set of people who are users coming from an intermediary market place. So any data that is missing or is not to be used is probably cleaned before it reaches to us. So we can have very good expectations that the data we are getting from the P2P market place is cleaned and saves us from Data Cleaning. 😃

In [7]:
```
dataset2 = dataset.drop(columns = ['entry_id', 'pay_schedule', 'e_signed'])
```

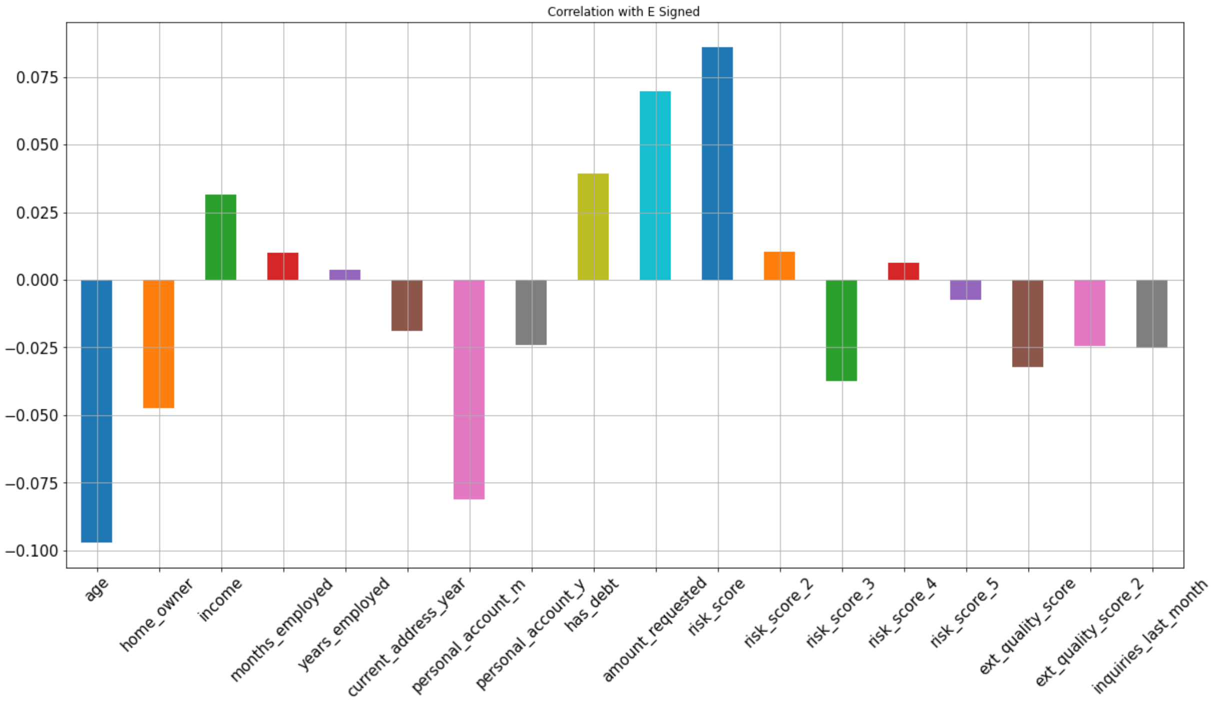
In [8]:
```
fig = plt.figure(figsize = (15, 12))
plt.suptitle('Histogram of Numerical Columns', fontsize = 20)
for i in range(dataset2.shape[1]):
    plt.subplot(6, 3, i + 1)
```

```
    f = plt.gca()
    f.set_title(dataset2.columns.values[i])

    vals = np.size(dataset2.iloc[:, i].unique())
    if vals >= 100:
        vals = 100

    plt.hist(dataset2.iloc[:, i], bins = vals, color = '#3F5D7D')
plt.tight_layout(rect = [0, 0.03, 1, 0.95])
#plt.savefig('1.Histogram of Numerical Columns.png')
```



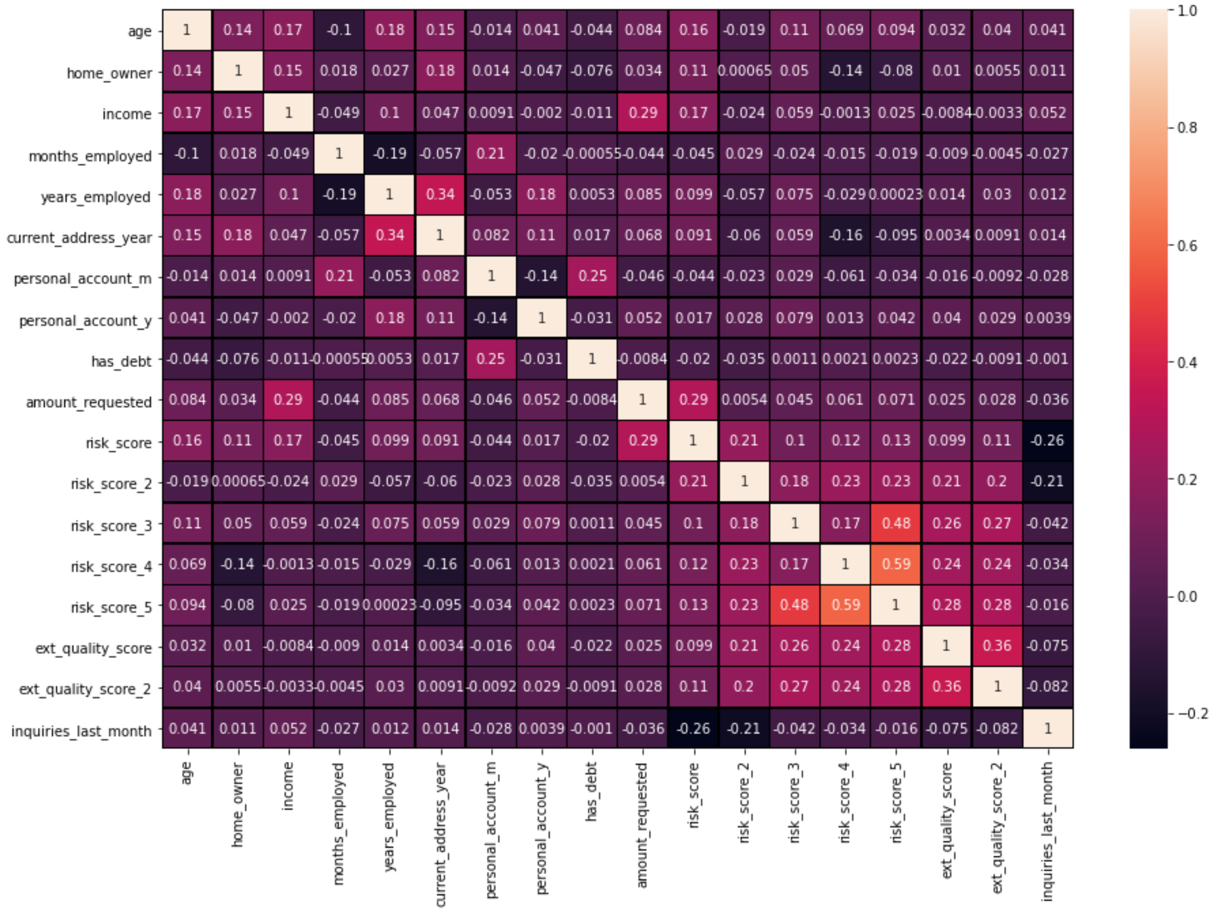
In [9]:

```
dataset2.corrwith(dataset.e_signed).plot.bar(figsize = (20, 10), title = "Correlatio
                                          fontsize = 15, rot = 45, grid = True, colo
#plt.savefig('2.Correlation of Features with the Response Variable.png')
```



In [10]:

```
f, ax = plt.subplots(figsize = (15, 10))
sns.heatmap(dataset2.corr(), annot = True, linewidths = 0.5, linecolor = 'black')
#plt.savefig('3.Heatmap showing Correlation of all the Features.png')
```



Data Preprocessing

In [11]:

```
#random.seed(100)
dataset = dataset.drop(columns = ['months_employed'])
dataset['personal_account_months'] = (dataset.personal_account_m + (dataset.personal
dataset = dataset.drop(columns = ['personal_account_m', 'personal_account_y'])
```

In [12]:
```
dataset.head()
```

Out[12]:

	entry_id	age	pay_schedule	home_owner	income	years_employed	current_address_year	has_de
0	7629673	40	bi-weekly	1	3135	3	3	
1	3560428	61	weekly	0	3180	6	3	
2	6934997	23	weekly	0	1540	0	0	
3	5682812	40	bi-weekly	0	5230	6	1	
4	5335819	33	semi-monthly	0	3590	5	2	

In [13]:
```
dataset.dtypes
```

Out[13]:
```
entry_id                  int64
age                       int64
pay_schedule             object
home_owner                int64
income                    int64
years_employed            int64
current_address_year      int64
has_debt                  int64
amount_requested          int64
risk_score                int64
risk_score_2            float64
risk_score_3            float64
risk_score_4            float64
risk_score_5            float64
ext_quality_score       float64
ext_quality_score_2     float64
inquiries_last_month      int64
e_signed                  int64
personal_account_months   int64
dtype: object
```

One-Hot Encoding

In [14]:
```
dataset = pd.get_dummies(dataset)
dataset.columns
```

Out[14]:
```
Index(['entry_id', 'age', 'home_owner', 'income', 'years_employed',
       'current_address_year', 'has_debt', 'amount_requested', 'risk_score',
       'risk_score_2', 'risk_score_3', 'risk_score_4', 'risk_score_5',
       'ext_quality_score', 'ext_quality_score_2', 'inquiries_last_month',
       'e_signed', 'personal_account_months', 'pay_schedule_bi-weekly',
       'pay_schedule_monthly', 'pay_schedule_semi-monthly',
       'pay_schedule_weekly'],
      dtype='object')
```

In [15]:
```
dataset = dataset.drop(columns = ['pay_schedule_semi-monthly'])
```

Removing Extra Columns

In [16]:
```
y = dataset['e_signed']
users = dataset['entry_id']
dataset = dataset.drop(columns = ['e_signed', 'entry_id'])
```

Splitting into Train & Test Set

In [17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dataset, y, test_size = 0.2, ran
```

Feature Scaling

In [18]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train2 = pd.DataFrame(sc.fit_transform(X_train))
X_test2 = pd.DataFrame(sc.transform(X_test))

X_train2.columns = X_train.columns.values
X_test2.columns = X_test.columns.values

X_train2.index = X_train.index.values
X_test2.index = X_test.index.values

X_train = X_train2
X_test = X_test2
```

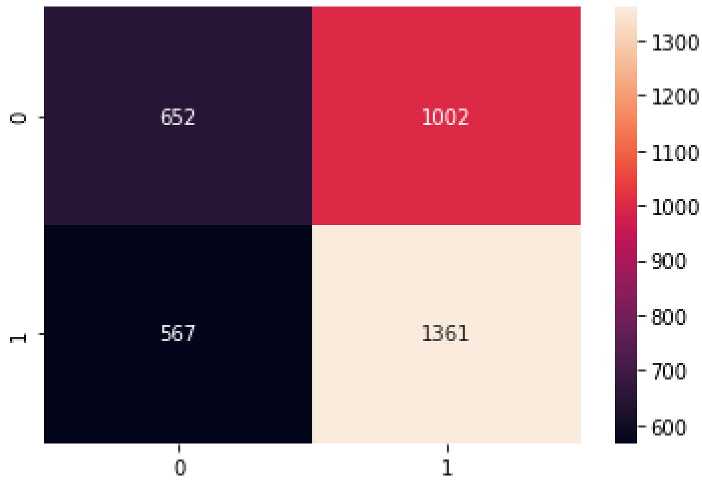
Model Building

Logistic Regression

In [19]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, solver = 'liblinear', penalty = 'l
classifier.fit(X_train, y_train)


# Predicitng the Test Set
y_pred = classifier.predict(X_test)


# Checking Acccuracy
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
acc = (accuracy_score(y_test, y_pred))
print(acc)
print(classification_report(y_test, y_pred))
```

```
0.5619765494137353
              precision    recall  f1-score   support

           0       0.53      0.39      0.45      1654
           1       0.58      0.71      0.63      1928

    accuracy                           0.56      3582
   macro avg       0.56      0.55      0.54      3582
weighted avg       0.56      0.56      0.55      3582
```

In [20]:
```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot = True, fmt = 'g')
#plt.savefig('4.Logistic Regression - Confusion Matrix.png')
```



In [21]:

```
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

results = pd.DataFrame([['Linear Regression (Lasso)', acc, prec, rec, f1]],
             columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

Support Vector Classification SVC

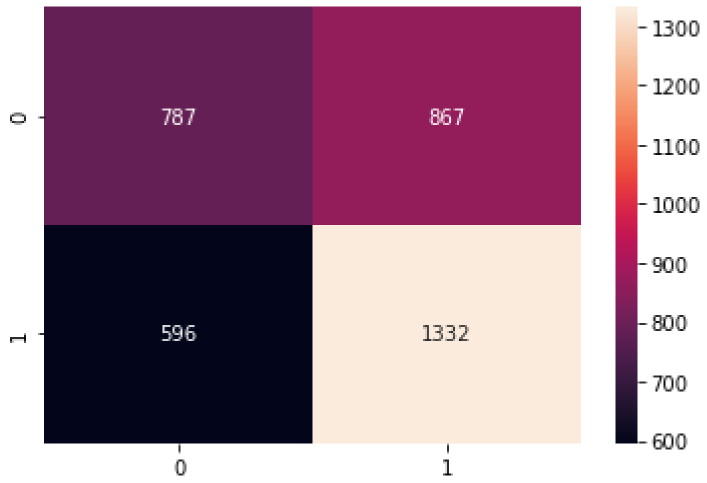
In [22]:

```
from sklearn.svm import SVC
classifier = SVC(random_state = 0, kernel = 'rbf')
classifier.fit(X_train, y_train)


# Predicitng the Test Set
y_pred = classifier.predict(X_test)


# Checking Acccuracy
acc = (accuracy_score(y_test, y_pred))
print(acc)
print(classification_report(y_test, y_pred))
```

```
0.5915689558905639
              precision    recall  f1-score   support

           0       0.57      0.48      0.52      1654
           1       0.61      0.69      0.65      1928

    accuracy                           0.59      3582
   macro avg       0.59      0.58      0.58      3582
weighted avg       0.59      0.59      0.59      3582
```

In [23]:

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot = True, fmt = 'g')
#plt.savefig('5.Support Vector Classification - Confusion Matrix.png')
```



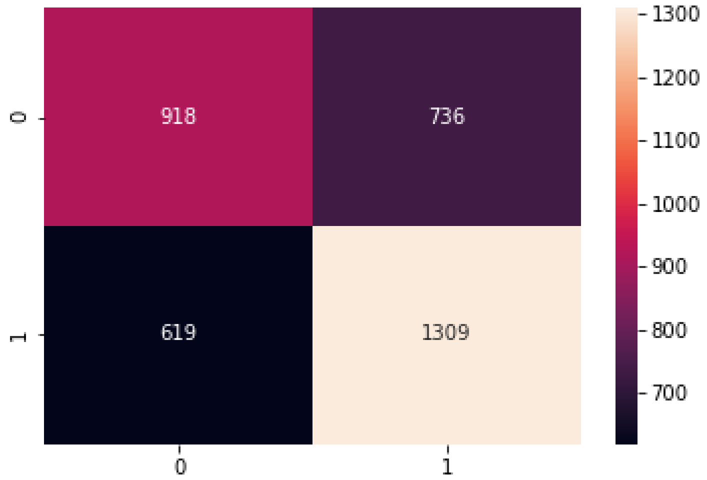
In [24]:
```
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['SVC (RBF)', acc, prec, rec, f1]],
              columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
results = results.append(model_results, ignore_index = True)
```

Random Forest Classifier

In [25]:
```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state = 0, n_estimators = 100, criterion
classifier.fit(X_train, y_train)


# Predicitng the Test Set
y_pred = classifier.predict(X_test)


# Checking Acccuracy
acc = (accuracy_score(y_test, y_pred))
print(acc)
print(classification_report(y_test, y_pred))
```

```
0.6217197096594081
              precision    recall  f1-score   support

           0       0.60      0.56      0.58      1654
           1       0.64      0.68      0.66      1928

    accuracy                           0.62      3582
   macro avg       0.62      0.62      0.62      3582
weighted avg       0.62      0.62      0.62      3582
```

In [26]:
```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot = True, fmt = 'g')
#plt.savefig('6.Random Forest Classifier - Confusion Matrix.png')
```



In [27]:

```
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['Random Forest (n = 100)', acc, prec, rec, f1]],
             columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
results = results.append(model_results, ignore_index = True)
```

In [28]:

```
results
```

Out[28]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Linear Regression (Lasso)	0.561977	0.575963	0.705913	0.634351
1	SVC (RBF)	0.591569	0.605730	0.690871	0.645505
2	Random Forest (n = 100)	0.621720	0.640098	0.678942	0.658948

K-Fold Cross Validation for our Random Forest Algorithm

In [29]:

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv =
print("Random Forest Classifier Mean Accuracy: %0.2f" % accuracies.mean())
print("Random Forest Classifier Standard Deviation: %0.2f" % (accuracies.std() * 2))
```

```
Random Forest Classifier Mean Accuracy: 0.63
Random Forest Classifier Standard Deviation: 0.03
```

Parameter Tuning of Random Forest Algorithm

In [30]:

```
parameters = {'max_depth': [3, None], 'max_features': [1, 5, 10], 'min_samples_split
              'min_samples_leaf': [1, 5, 10], 'bootstrap': [True, False], 'criterion
```

In [31]:

```
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator = classifier, param_grid = parameters, scoring

t0 = time.time()
grid_search = grid_search.fit(X_train, y_train)
t1 = time.time()
print('Took %0.2f Seconds' % (t1 - t0))
```

```
Took 3986.89 Seconds
```

In [32]:

```
rf_best_accuracy = grid_search.best_score_
rf_best_parameters = grid_search.best_params_
rf_best_accuracy, rf_best_parameters
```

Out[32]:
```
(0.6353512282315882,
 {'bootstrap': True,
  'criterion': 'gini',
  'max_depth': None,
  'max_features': 10,
  'min_samples_leaf': 5,
  'min_samples_split': 2})
```

Parameter Tuning has not given much of an improvement but maybe by a few points.

In [33]:
```
y_pred = grid_search.predict(X_test)


# Checking Acccuracy
acc = (accuracy_score(y_test, y_pred))
print(acc)
print(classification_report(y_test, y_pred))
```

```
0.6395868230039085
              precision    recall  f1-score   support

           0       0.62      0.57      0.59      1654
           1       0.65      0.70      0.68      1928

    accuracy                           0.64      3582
   macro avg       0.64      0.63      0.63      3582
weighted avg       0.64      0.64      0.64      3582
```

In [34]:
```
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['Random Forest (Grid Search)', acc, prec, rec, f1]],
             columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
results = results.append(model_results, ignore_index = True)
```

In [35]:
```
results
```

Out[35]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Linear Regression (Lasso)	0.561977	0.575963	0.705913	0.634351
1	SVC (RBF)	0.591569	0.605730	0.690871	0.645505
2	Random Forest (n = 100)	0.621720	0.640098	0.678942	0.658948
3	Random Forest (Grid Search)	0.639587	0.653494	0.703320	0.677492

Final Result

In [36]:
```
final_results = pd.concat([y_test, users], axis = 1).dropna()
final_results['predictions'] = y_pred
final_results = final_results[['entry_id', 'e_signed', 'predictions']]
```

In [37]:
```
final_results
```

Out[37]:

	entry_id	e_signed	predictions
8	6493191	1.0	0
9	8908605	1.0	0

	entry_id	e_signed	predictions
12	6889184	1.0	1
16	9375601	0.0	1
18	8515555	1.0	1
...	...	...	...
17881	5028251	1.0	1
17888	8958068	0.0	0
17890	3605941	0.0	1
17901	1807355	0.0	1
17907	1498559	1.0	1

3582 rows × 3 columns

Conclusion

Our model has given us an accuracy of around 64%. With this, we have an algorithm that can help predict whether a user will complete the E-Signing step of the loan application. One way to leverage this model is to target those predicted to not reach the e-sign phase with customized onboarding. This means that when a lead arrives from the marketplace, they may receive a different onboarding experience based on how likely they are to finish the general onboarding process. This can help our company minimize how many people drop off from the funnel. This funnel of screens is as effective as we, as a company, build it. Therefore, user drop-off in this funnel falls entirely on our shoulders. So, with new onboarding screens built intentionally to lead users to finalize the loan application, we can attempt to get more than 40% of those predicted to not finish the process to complete the e-sign step. If we can do this, then we can drastically increase profits. Many lending companies provide hundreds of loans every day, gaining money for each one. As a result, if we can increase the number of loan takers, we are increasing profits. All with a simple model!