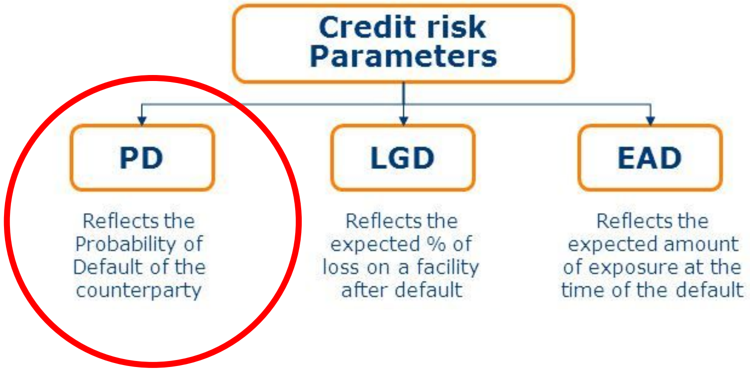
Loan Default Prediction

Denizhan Özyurt
Ezgi Berksoy
Ömer Bahar
Ozan Yağcılar
Zeki Berkay Saygın

Kaggle Case: Loan Default Prediction

- **Case:** Kaggle Competition from Imperial College of London, 5 years ago (link)
- **Objective:** predicting whether the loan will default or not by looking at 776 anonymous features
- **Data:** 105,471 observations for 776 features and “loss” label. 653 of these variables were in float type, 99 was in integer type and 19 was in object type.
- **Why Loan Default Prediction:** knowledge-abundant and challenging topic as there are a great number of people working on the topic
- **Performance:** We used Random Forest Model and got cross validation score of 90.66%. That’s worse than the most, but at least we made it work.



Method: Random Forest Classification

- **Selecting the right model:** We discussed upon 4 different models. Since we had a lot of features and our features were anonymized already, it was a good idea to pick something that is inherently randomized. Also Mr. Dalaman suggested to Random Forest as well. :-)
- **Library:** We used random forest functions from scikit-learn libraries

Logistic Regression

K-nearest Neighbour Class.

Random Forest Class.

Support Vector Machine

Decision Trees: Supervised learning algorithms used for classification

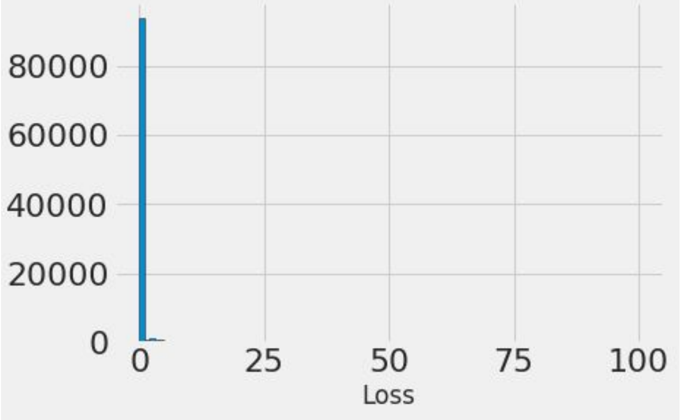
Random Forest: Builds an ensemble of decision trees to find the most accurate and stable prediction

Statistical Analysis and Data Exploration

- **Massive Data:** 105,471 observations with 778 variables and an output variable of loss
- **Main Statistics:** We looked at the following statistics to understand the data: count, mean, standard deviation, min, 25%, 50%, 75% and max. But we couldn't understand much since our data was anonymized.
- **Output:** Our output (y_variable) is losses which is also shown in the histogram on right

	id	f1	f2	f3	f4	f5	f6	f7
count	105471.000000	105471.000000	105471.000000	105471.000000	105471.000000	105471.000000	105471.000000	105289.0000
mean	52736.000000	134.603171	8.246883	0.499066	2678.488874	7.354533	47993.704317	2974.336018
std	30446.999458	14.725467	1.691535	0.288752	1401.010943	5.151112	35677.136048	2546.551085
min	1.000000	103.000000	1.000000	0.000006	1100.000000	1.000000	0.000000	1.000000
25%	26368.500000	124.000000	8.000000	0.248950	1500.000000	4.000000	11255.000000	629.000000
50%	52736.000000	129.000000	9.000000	0.498267	2200.000000	4.000000	76530.000000	2292.000000
75%	79103.500000	148.000000	9.000000	0.749494	3700.000000	10.000000	80135.000000	4679.000000
max	105471.000000	176.000000	11.000000	0.999994	7900.000000	17.000000	88565.000000	9968.000000

Histogram



Raw Data (778 variables, 105K observations)

	id	f1	f2	f3	f5	f6	f7	f8	f13	f14	f15	f16	f19	f25	f33	f34	f35	f36	f37	f38	f44	f69	f7
0	1	126	10	0.686842	3	13699	7201.0	4949.0	7	0.7607	0.7542	612922	0.5171	66	0	0	0	5	0	0	1.037424	-0.01	0.0
1	2	121	10	0.782776	3	84645	240.0	1625.0	7	0.6555	0.6555	245815	0.3909	50	0	0	0	6	0	0	-0.915138	-0.04	-0.0
2	3	126	10	0.500080	3	83607	1800.0	1527.0	7	0.7542	0.7542	1385872	0.5508	54	0	0	0	13	0	0	-1.332533	-0.03	-0.0
3	4	134	10	0.439874	3	82642	7542.0	1730.0	7	0.8017	0.7881	704687	0.5923	55	0	0	0	4	0	0	-0.947279	0.02	0.0
4	5	109	9	0.502749	4	79124	89.0	491.0	6	0.5263	0.5263	51985	0.3044	21	0	0	0	26	0	0	-0.950251	-0.20	-0.2
5	6	126	9	0.691954	4	14448	1514.0	4176.0	6	0.8070	0.7480	764587	0.5212	64	0	0	0	22	0	0	-0.564620	-0.03	-0.0
6	7	121	9	0.985674	4	13026	4565.0	263.0	6	0.7739	0.7739	542244	0.5378	64	0	0	0	23	0	0	-0.249323	-0.08	-0.0
7	8	128	9	0.385778	4	79244	6597.0	3592.0	6	0.8596	0.7967	17175731	0.5868	75	0	0	0	17	0	0	-0.928703	0.01	0.0
8	9	126	9	0.745471	4	78920	3058.0	112.0	6	0.8684	0.8049	1560191	0.5705	72	0	0	0	7	0	0	-1.211526	0.03	0.0
9	10	127	9	0.580561	4	83442	684.0	1141.0	6	0.8534	0.8534	339011	0.6883	66	0	0	0	5	0	0	0.911271	0.00	-0.0
10	11	115	9	0.611158	3	6901	685.0	2437.0	14	0.8276	0.8136	70132	0.5577	53	0	0	0	15	0	0	0.302215	-0.04	0.0
11	12	120	9	0.801255	3	13026	4566.0	982.0	14	0.7177	0.7177	0	0.4852	37	0	0	0	90	0	0	-1.030597	0.03	0.0
12	13	130	9	0.574090	3	8563	5264.0	3566.0	14	0.8051	0.8051	12158352	0.5797	57	0	0	0	81	0	0	-1.428818	0.05	0.0
13	14	119	9	0.445187	3	2456	8236.0	983.0	14	0.7876	0.7607	2228468	0.5312	41	0	0	0	70	0	0	-1.742888	-0.06	-0.0
14	15	116	9	0.092193	3	83726	3059.0	1731.0	14	0.8793	0.8644	704838	0.6848	60	0	0	0	60	0	0	-0.946539	-0.09	-0.0
15	16	123	9	0.466685	3	9128	6864.0	6316.0	14	0.7983	0.7724	6559743	0.5845	62	0	0	0	70	0	0	-1.698270	-0.02	-0.0
16	17	130	9	0.437883	3	11255	3679.0	2969.0	14	0.9292	0.8974	1452127	0.7569	77	0	0	0	54	0	0	-1.675873	0.00	0.0
17	18	125	9	0.681169	3	14184	332.0	1528.0	14	0.8319	0.8049	259490	0.5890	70	0	0	0	62	0	0	-1.245031	0.05	0.0
18	19	130	9	0.655732	3	113	6611.0	2795.0	14	0.7881	0.7881	4963974	0.5372	39	0	0	0	22	0	0	-1.007784	0.07	0.1
19	20	114	9	0.810728	3	9836	241.0	1626.0	14	0.7966	0.7966	118935	0.5769	64	0	0	0	20	0	0	-1.151689	-0.09	-0.0
20	21	129	9	0.417462	3	13317	7543.0	1801.0	14	0.8879	0.8729	2683633	0.7265	78	0	0	0	8	0	0	-1.014823	0.00	0.0
21	22	127	9	0.014417	3	85723	931.0	6317.0	14	0.6379	0.6271	84375	0.3401	23	0	0	0	9	0	0	-0.632459	0.05	0.0
22	23	116	9	0.258737	3	13008	2617.0	152.0	14	0.7034	0.7034	77232	0.4498	42	0	0	0	6	0	0	1.071693	-0.08	-0.0
23	24	117	9	0.958126	3	76973	2039.0	2076.0	14	0.5913	0.5862	954067	0.3007	17	0	0	0	4	0	0	-0.937407	-0.05	-0.0
24	25	116	9	0.035919	4	77014	3535.0	2304.0	14	0.6552	0.6441	0	0.3710	25	0	0	0	16	0	0	-0.623102	-0.08	-0.0
25	26	123	9	0.830775	4	842	686.0	3750.0	14	0.8333	0.8130	365865	0.5630	73	0	0	0	46	0	0	-1.101829	0.04	0.0
26	27	125	9	0.197708	4	14694	2040.0	167.0	14	0.6613	0.6613	0	0.4936	40	0	0	0	27	0	0	-1.198865	-0.07	-0.0
27	28	120	9	0.788092	4	6901	242.0	652.0	14	0.7350	0.7049	1014197	0.4853	53	0	0	0	32	0	0	-1.190926	-0.03	-0.0
28	29	120	9	0.277109	4	79649	3770.0	743.0	14	0.8049	0.8049	2491476	0.5803	63	0	0	0	15	0	0	-0.674224	-0.05	-0.0

Cleaning the Data

1) **Standardize the features:** During the data exploration we realized the data is very unstandard. Since the data is anonymous we standardized the data by transforming it into a standard scalar vector

2) **Drop NAs:** 525 of our variables (776 total) had at least one of their observations missing. Some features like f662 has 17.9% of it's observations missing. We filled NAs with the mean of all observations remaining in that feature.

3) **Removing Collinearity:** We realized many of the features were proxying for similar things as we realized the correlation of many features was very high with each other. In order to reduce this collinearity, we decide to eliminate some of these features. We calculated a correlation matrix. In order to eliminate most features we can (to gain some computing power) we set a threshold like 0.6, and eliminated one of each features that his correlation higher than 0.6.

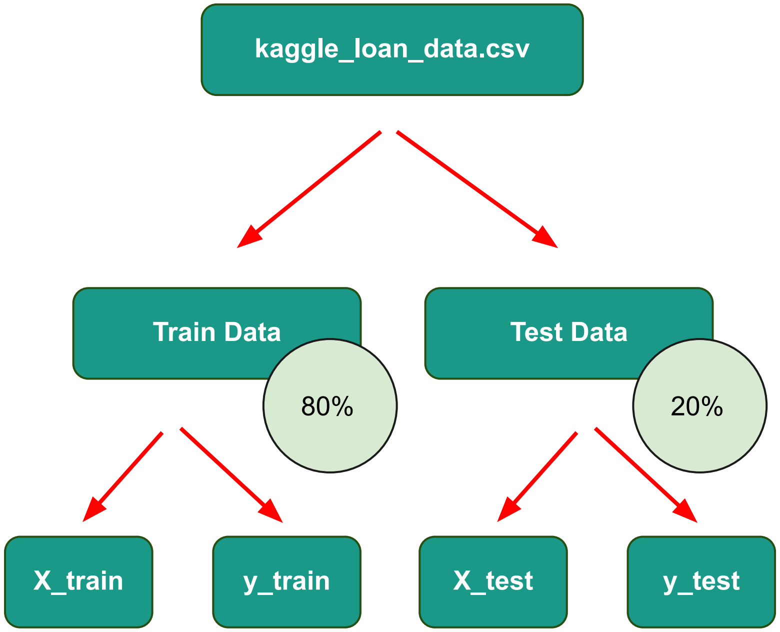
```
Your selected dataframe has 771 columns.
There are 525 columns that have missing values.
```

Out[4]:

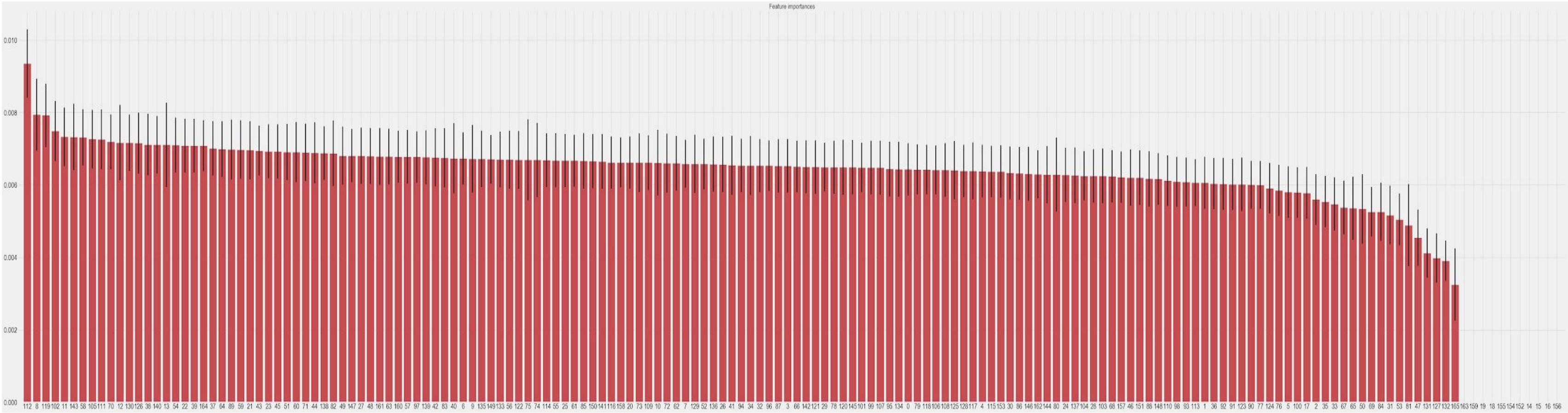
	Missing Values	% of Total Values
f662	18833	17.9
f663	18833	17.9
f159	18736	17.8
f160	18736	17.8
f170	18417	17.5
f169	18417	17.5
f618	18407	17.5
f619	18407	17.5
f331	18067	17.1
f330	18067	17.1

Random Forest Model

- **Data:** First, we randomly split the data into two as the following: 80% Train, 20% Test.
- **Train:** Later we trained the model with the train data by using "ExtraTreesClassifier" function from scikit-learn library with n_estimators = 250).
- **Features:** We fitted the model and calculated the importance score of each feature. Later we listed the features according to their importance scores.
- **Prioritization of Features:** Using an extra tree classifier we analysed the importance of each feature and realized that some features have no impact on the classification, except one feature, mainly FX, rest of the features has pretty similar impact and the overall impact for each feature is low, meaning the combination of features is more important than a couple of main features.



List of Features, sorted in their importance




```
# Build a forest and compute the feature importances
forest = ExtraTreesClassifier(n_estimators=250,
                              random_state=0)
```

```
forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in
forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]
```

Final Results

- **Testing the Model with Test Data:** After fitting the train data (X_train, y_train) with a random forest model, we tested the success of our model by looking at Mean Absolute Errors with a cross validation score on the test data we previously separated randomly.
- **Cross Validation Score:** Our cross validation score was 0.9066, which was higher than some solution attempts posted on Kaggle.
- **Graph Limitations:** As our decision tree had 166 features, it was not intuitive to plot the graph for this.

Cross Validation Calculation with Mean Absolute Error

```
# Function to calculate mean absolute error

def cross_val(X_train, y_train, model):
    from sklearn.model_selection import cross_val_score
    accuracies = cross_val_score(estimator = model, X =
X_train, y = y_train, cv = 5)

    return accuracies.mean()
```

Resources:

Kaggle Competition

- https://www.kaggle.com/c/loan-default-prediction/overview

Dropbox Link for Our Data (kaggle_loan_data.csv)

- https://www.dropbox.com/s/t67zsa44iebunwa/kaggle_loan_data.csv?dl=0

Scikit Learn Random Forest Classifier Documentations

- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://scikit-learn.org/stable/modules/cross_validation.html

Random Forest Blogs on Medium.com

- https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d
- https://medium.com/datadriveninvestor/k-fold-cross-validation-6b8518070833
- https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd