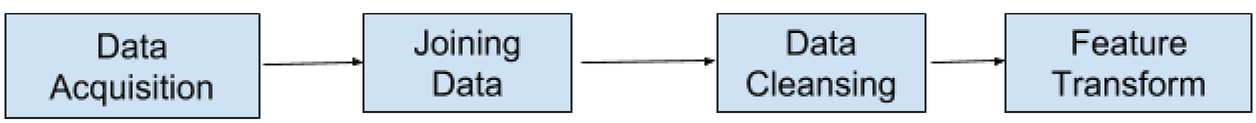
Data Wrangling

Data wrangling is the process of cleaning and unifying messy and complex data sets for easy access and analysis.

Goals of Data Wrangling are:

- Reveal a “deeper intelligence” within your data, by gathering data from multiple sources
- Provide accurate, actionable data in the hands of business analysts in a timely matter
- Reduce the time spent collecting and organizing unruly data before it can be utilized
- Enable data scientists and analysts to focus on the analysis of data, rather than the wrangling
- Drive better decision-making skills by senior leaders in an organization

Key steps in Data Wrangling



The data is provided by Home Credit Group, a service dedicated to provided lines of credit (loans) to the unbanked population.

Project Objective

Predicting whether or not a client will repay a loan or have difficulty is a critical business need, and Home Credit is hosting this competition on Kaggle to see what sort of models the machine learning community can develop to help them in this task. application_train/application_test: the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.

These data are readily and publicly available at https://www.kaggle.com/c/home-credit-default-risk/data and appear as below:

SK_ID_CURR	TARGET	NAME_CONT	CODE_GEND	FLAG_OWN_	FLAG_OWN_	CNT_CHILDR	AMT_INCOM	AMT_CREDIT	AMT_ANNUI	AMT_GOODS	NAME_TYPE_	NAME_INCO	NAME_EDUC	NAME_FAMI	NAME_HOUS	REGION_POI	DAYS_BIRTH	DAYS_EM
100002	1	Cash loans	M	N	Y	0	202500	406597.5	24700.5	351000	Unaccompar	Working	Secondary /	Single / not r	House / apar	0.018801	-9461	-6
100003	0	Cash loans	F	N	N	0	270000	1293502.5	35698.5	1129500	Family	State servant	Higher educa	Married	House / apar	0.003541	-16765	-11
100004	0	Revolving loa	M	Y	Y	0	67500	135000	6750	135000	Unaccompar	Working	Secondary /	Single / not r	House / apar	0.010032	-19046	-2
100006	0	Cash loans	F	N	Y	0	135000	312682.5	29686.5	297000	Unaccompar	Working	Secondary /	Civil marriag	House / apar	0.008019	-19005	-30
100007	0	Cash loans	M	N	Y	0	121500	513000	21865.5	513000	Unaccompar	Working	Secondary /	Single / not r	House / apar	0.028663	-19932	-30
100008	0	Cash loans	M	N	Y	0	99000	490495.5	27517.5	454500	Spouse, part	State servant	Secondary /	Married	House / apar	0.035792	-16941	-15
100009	0	Cash loans	F	Y	Y	1	171000	1560726	41301	1395000	Unaccompar	Commercial	Higher educa	Married	House / apar	0.035792	-13778	-31
100010	0	Cash loans	M	Y	Y	0	360000	1530000	42075	1530000	Unaccompar	State servant	Higher educa	Married	House / apar	0.003122	-18850	-4
100011	0	Cash loans	F	N	Y	0	112500	1019610	33826.5	913500	Children	Pensioner	Secondary /	Married	House / apar	0.018634	-20099	3652

The datasets required extensive data wrangling for it involved not only fundamental steps of data preparation but also feature engineering and data imputation to be run during Machine Learning:

1. **Extracting Data**
2. **Identifying Target Dataset among Multiple Data Sources**
3. **Identifying Missing Data**
4. **Identifying Data Types of the Feature Set into Non-Categorical and Categorical**
5. **Casting Data Types per Need**
6. **Feature Engineering (Date timestamp, One Hot Encoding)**

Here is the detailed codebook showing above steps:

https://github.com/rashi-n/Machine-Learning-Projects/blob/master/Capstone%20Projects/Capstone%20II%20Project/Data%20Wrangling.ipynb

Following the highlights from the codebook:

I. Among multiple datasets for Credit Accounts and Transactions, 'Application_train' and 'Application_Test' datasets were deduced to import as they seemed to carry most of the features that may be significant in analysis & predictions of the project objective. The main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.:

```
In [2]: #Importing the dataset
        df_train = pd.read_csv('application_train.csv')
        df_test=pd.read_csv('application_test.csv')

In [3]: #Shape of dataset
        df_train.shape

Out[3]: (307511, 122)
```

II. Dataset carries nearly 307K records with 121 features and one ‘TARGET’ variable to infer predictions on each transaction.

III. There were 67 columns that had missing values:

```
Your selected dataframe has 122 columns.
There are 67 columns that have missing values.
```

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MODE	210199	68.4
LIVINGAPARTMENTS_MEDI	210199	68.4
LIVINGAPARTMENTS_AVG	210199	68.4
FLOORSMIN_MODE	208642	67.8
FLOORSMIN_MEDI	208642	67.8
FLOORSMIN_AVG	208642	67.8
YEARS_BUILD_MODE	204488	66.5
YEARS_BUILD_MEDI	204488	66.5
YEARS_BUILD_AVG	204488	66.5
OWN_CAR_AGE	202929	66.0
LANDAREA_AVG	182590	59.4
LANDAREA_MEDI	182590	59.4
LANDAREA_MODE	182590	59.4

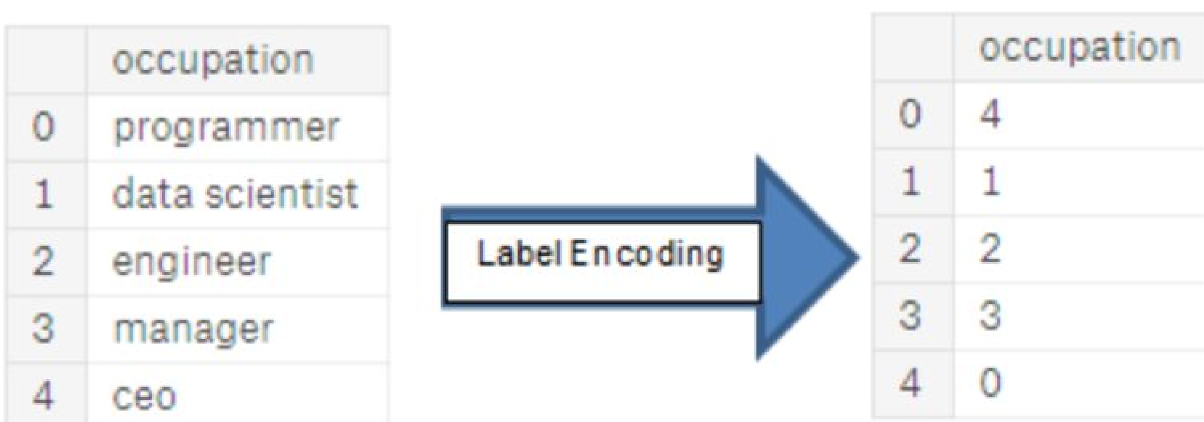
IV. Of the 122 features, 106 were non-categorical and 16 were categorical:

```
# Number of each type of column
df_train.dtypes.value_counts()
```

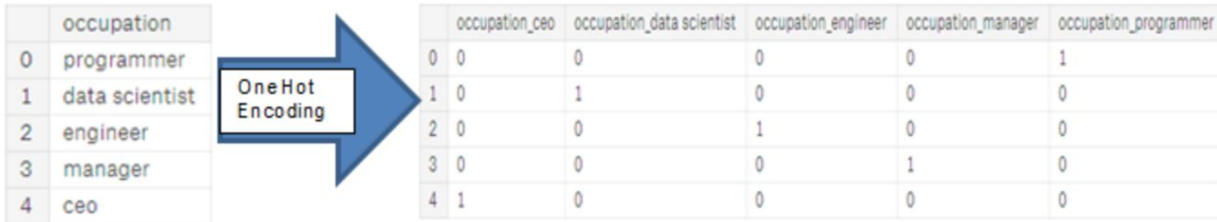
```
float64     65
int64       41
object      16
dtype: int64
```

V. The features were appropriately cast into right data types and categorical features were Label Encoded and One Hot Encoded. A machine learning model unfortunately cannot deal with categorical variables (except for some models such as LightGBM). Therefore, we have to find a way to encode (represent) these variables as numbers before handing them off to the model. There are two main ways to carry out this process:

Label encoding: assign each unique category in a categorical variable with an integer. No new columns are created. An example is shown below image



One-hot encoding: create a new column for each unique category in a categorical variable. Each observation receives a 1 in the column for its corresponding category and a 0 in all other new columns.



	occupation
0	programmer
1	data scientist
2	engineer
3	manager
4	ceo

	occupation_ceo	occupation_data scientist	occupation_engineer	occupation_manager	occupation_programmer
0	0	0	0	0	1
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	1	0	0	0	0

The problem with label encoding is that it gives the categories an arbitrary ordering. The value assigned to each of the categories is random and does not reflect any inherent aspect of the category. In the example above, programmer receives a 4 and data scientist a 1, but if we did the same process again, the labels could be reversed or completely different. The actual assignment of the integers is arbitrary. Therefore, when we perform label encoding, the model might use the relative value of the feature (for example programmer = 4 and data scientist = 1) to assign weights which is not what we want. If we only have two unique values for a categorical variable (such as Male/Female), then label encoding is fine, but for more than 2 unique categories, one-hot encoding is the safe option.

There is some debate about the relative merits of these approaches, and some models can deal with label encoded categorical variables with no issues. Here is a good Stack Overflow discussion. I think (and this is just a personal opinion) for categorical variables with many classes, one-hot encoding is the safest approach because it does not impose arbitrary values to categories. The only downside to one-hot encoding is that the number of features (dimensions of the data) can explode with categorical variables with many categories. To deal with this, we can perform one-hot encoding followed by PCA or other dimensionality reduction methods to reduce the number of dimensions (while still trying to preserve information).

In this notebook, we will use Label Encoding for any categorical variables with only 2 categories and One-Hot Encoding for any categorical variables with more than 2 categories. This process may need to change as we get further into the project, but for now, we will see where this gets us. (We will also not use any dimensionality reduction in this notebook but will explore in future iter

Let's implement the policy described above: for any categorical variable (`dtype == object`) with 2 unique categories, we will use label encoding, and for any categorical variable with more than 2 unique categories, we will use one-hot encoding.

For label encoding, we use the Scikit-Learn `LabelEncoder` and for one-hot encoding, the pandas `get_dummies(df)` function.

3 columns were label encoded.

VI. **Aligning Training and Testing Data**

There need to be the same features (columns) in both the training and testing data. One-hot encoding has created more columns in the training data because there were some categorical variables with categories not represented in the testing data. To remove the columns in the training data that are not in the testing data, we need to align the dataframes. First we extract the target column from the training data (because this is not in the testing data but we need to keep this information). When we do the align, we must make sure to set axis = 1 to align the dataframes based on the columns and not on the rows!

('Training Features shape: ', (307511, 240))

('Testing Features shape: ', (48744, 239))

The training and testing datasets now have the same features which is required for machine learning. The number of features has grown significantly due to one-hot encoding. At some point we probably will want to try dimensionality reduction (removing features that are not relevant) to reduce the size of the datasets.

VII. Followed two approaches to treat missing Data

It is important to understand how to deal with missing data. As we learn more data science/statistics, you'll learn about data imputation. Here, we'll learn to find missing data points and then we'll drop those points from the dataset so as not to affect our analysis with bias: an important part of data wrangling and data cleaning. We'll try to find which columns in 'df_train.csv' contain missing values and drop those missing values so you'll have tidy data.

Missing Values Strategy # 1 - Identify Features with Missing Values -> Replace with NaN -> Remove all Features with Missing Value -> Assess Model using Logistic Regression

Missing Values Strategy # 2 - Identify Features with Missing Values -> Replace with NaN -> Impute all Features with Missing Value -> Assess Model using Logistic Regression

Followed both the approaches and created Logistic REgression model with following accuracies respectively:

Accuracy of logistic regression classifier on test set: 0.93 with Srategy #1

Accuracy of logistic regression classifier on test set: 0.92 with Strategy#2

```
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))
```

```
Accuracy of logistic regression classifier on test set: 0.92
```

Now our baseline model is ready to start Exploratory & Inferential Data Statistics.