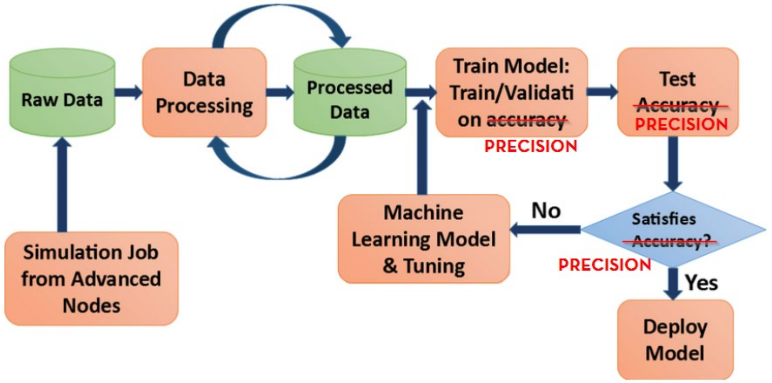
Employee Departure Predictions

Knowledge Transfer - Supplemental Material

University of Minnesota
Driven to Discover℠

The most important things to remember...

- There is no baseline model being used in other industries..this is all based on the data that we have available to us, and will be specific to our University.
- This type of work requires a significant amount of trial & error; it is iterative and *may* requires many **thousands** of attempts before we can release it to production.
- <u>Eduardo and Danny are no longer the experts!</u> Everyone on the team has a place to provide input into this process and improve the model performance.



A. Background & Problem Statement

Background of problem

Employee turnover is a significant problem for organizations, as it is difficult to predict when employees will resign and often can introduce noticeable voids in an organization’s workforce. As a result, it is imperative that organizations formulate proper recruitment, acquisition, and retention strategies, as well as implementing effective mechanisms for preventing and diminishing the impact of employee turnover, while understanding its underlying root causes.

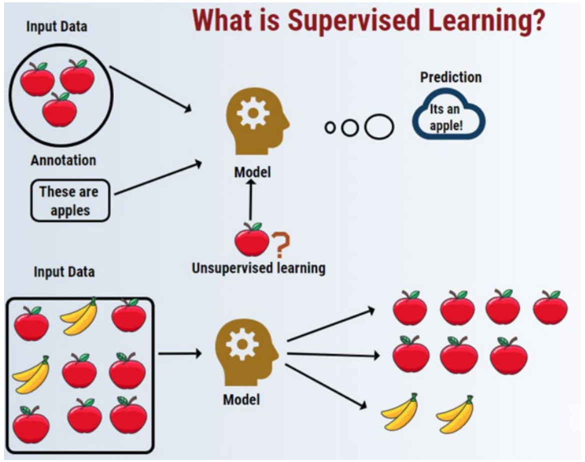
Problem statement

The Office of Human Resources at the University of Minnesota is looking to leverage machine learning & predictive modeling to predict (with *at least* a two-month lead time) which **full -time salaried faculty and professional staff** employees have the *highest* probability of resigning. The end goal of building a robust predictive model is to help HR leaders in each unit identify their employees who are most likely to resign in order to have intervention conversations and thereby increase the overall retention rate for their units and the University overall.

B. Why use supervised learning & classification?

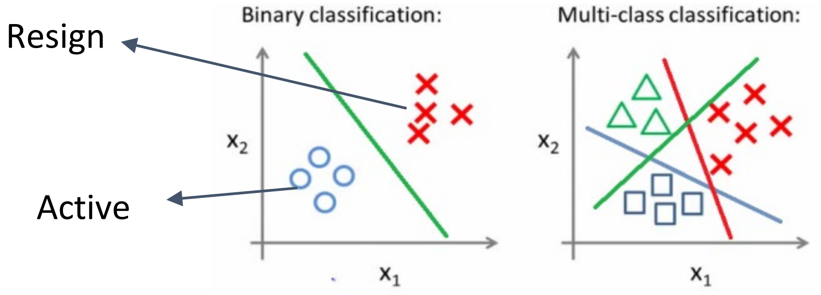
Supervised learning

In supervised learning, the training data you feed to the ML algorithm includes the desired outcomes or target; these go by many names, including **label** , **target** , **class** . If our data points were splattered across a dartboard, these would be our bulls-eyes. We have the historical records of every employee who has a Workforce Action Reason Description = 'Resignation', which we can use to assign a label and generate predictions based on this label.



Classification models tell you 'who'

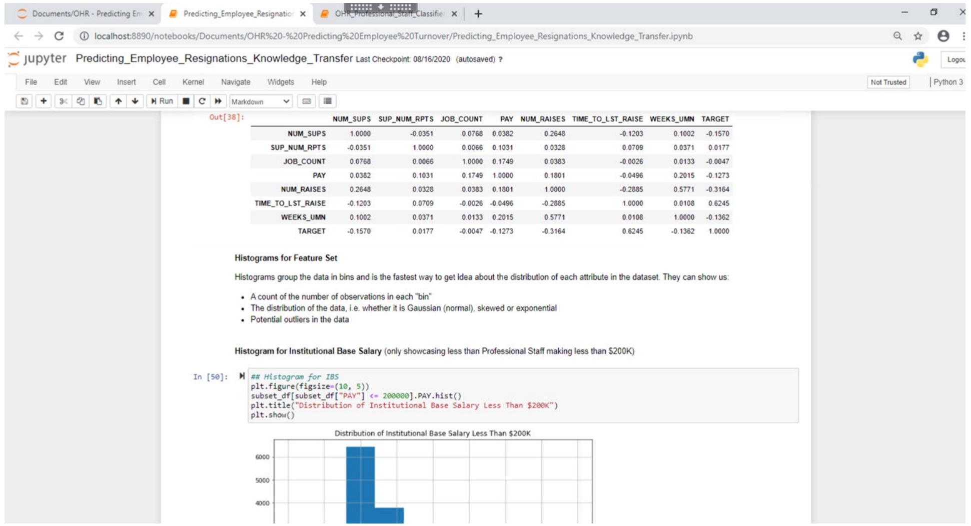
- Classify each observation into separate categories
  - 0 for Active employees, 1 for Resigned employees
- Different modeling techniques can be used
  - KNN, Decision Tree, Logistic Regression, Random Forest
- Used to predict what *class* (or group) a data observation (which **employee**) belongs to [resign or does not resign]
- Can consider timeframes (this is <u>important</u>!!)



C. Environment and Software Requirements

The main software/ hardware that is required is:

- Anaconda, free/ open source distribution of Python and R
- Python programming language
- SQL
- Jupyter Notebooks (example to the right)



Out[38]:

	NUM_SUPS	SUP_NUM_RPTS	JOB_COUNT	PAY	NUM_RAISES	TIME_TO_LST_RAISE	WEEKS_UMN	TARGET
NUM_SUPS	1.0000	-0.0351	0.0768	0.0382	0.2648	-0.1203	0.1002	-0.1570
SUP_NUM_RPTS	-0.0351	1.0000	0.0066	0.1031	0.0328	0.0709	0.0371	0.0177
JOB_COUNT	0.0768	0.0066	1.0000	0.1749	0.0383	-0.0026	0.0133	-0.0047
PAY	0.0382	0.1031	0.1749	1.0000	0.1801	-0.0496	0.2015	-0.1273
NUM_RAISES	0.2648	0.0328	0.0383	0.1801	1.0000	-0.2885	0.5771	-0.3164
TIME_TO_LST_RAISE	-0.1203	0.0709	-0.0026	-0.0496	-0.2885	1.0000	0.0108	0.6245
WEEKS_UMN	0.1002	0.0371	0.0133	0.2015	0.5771	0.0108	1.0000	-0.1362
TARGET	-0.1570	0.0177	-0.0047	-0.1273	-0.3164	0.6245	-0.1362	1.0000

Histograms for Feature Set

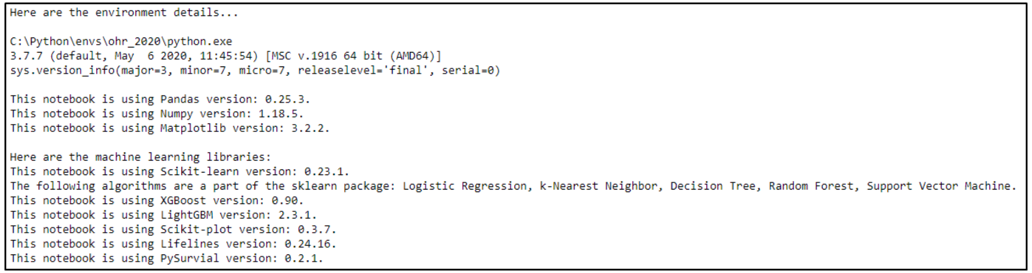
Histograms group the data in bins and is the fastest way to get idea about the distribution of each attribute in the dataset. They can show us:

- A count of the number of observations in each "bin"
- The distribution of the data, i.e. whether it is Gaussian (normal), skewed or exponential
- Potential outliers in the data

Histogram for Institutional Base Salary (only showcasing less than Professional Staff making less than $200K)

```
In [50]:
## Histogram for IBS
plt.figure(figsize=(10, 5))
subset_df[subset_df["PAY"] <= 200000].PAY.hist()
plt.title("Distribution of Institutional Base Salary Less Than $200K")
plt.show()
```

Documents/OHR - Predicting Em
Predicting_Employee_Resignatio
OHR_Professional_Staff_Classifie
localhost:8890/notebooks/Documents/OHR%20-%20Predicting%20Employee%20Turnover/Predicting_Employee_Resignations_Knowledge_Transfer.ipynb
jupyter Predicting_Employee_Resignations_Knowledge_Transfer Last Checkpoint: 08/16/2020 (autosaved)
File Edit View Insert Cell Kernel Navigate Widgets Help
Not Trusted
Python 3
Run
Markdown
Distribution of Institutional Base Salary Less Than $200K
6000
5000
4000



```
Here are the environment details...

C:\Python\envs\ohr_2020\python.exe
3.7.7 (default, May  6 2020, 11:45:54) [MSC v.1916 64 bit (AMD64)]
sys.version_info(major=3, minor=7, micro=7, releaselevel='final', serial=0)

This notebook is using Pandas version: 0.25.3.
This notebook is using Numpy version: 1.18.5.
This notebook is using Matplotlib version: 3.2.2.

Here are the machine learning libraries:
This notebook is using Scikit-learn version: 0.23.1.
The following algorithms are a part of the sklearn package: Logistic Regression, k-Nearest Neighbor, Decision Tree, Random Forest, Support Vector Machine.
This notebook is using XGBoost version: 0.90.
This notebook is using LightGBM version: 2.3.1.
This notebook is using Scikit-plot version: 0.3.7.
This notebook is using Lifelines version: 0.24.16.
This notebook is using PySurvial version: 0.2.1.
```

- A virtual machine that can be set up dedicated to solely running the algorithms is ideal
- There is a dependency on "hooking" into the data in EDW/ UM Analytics from the VM

D. Getting data from EDW

- Defining the employee population and filters
- Flattening the data set / table
    - Why do we need to do this for classification?
- Feature generation
    - What is a feature?
    - What is a “good” feature?
- Creating our target variables
    - 0, 1 (Hello, world!)
- Correlation matrix and feature interaction
- Implementing a time lag with cutoff dates

D. Getting data from EDW - Flattening the data set

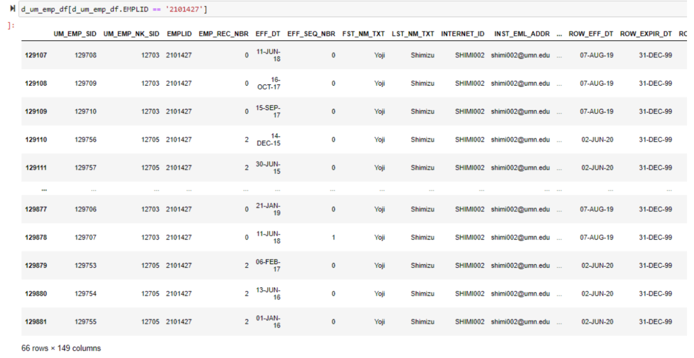
Why do we need to “flatten” the table?

The Employee Dimension is not at the right level of granularity. There are multiple rows per emplid (different dates, different job codes, different workforce actions, etc.). We need to generate **ONE** row per emplid because we need to classify the employee as “resigned” or “active”, and so we <u>cannot</u> have multiple observations for each emplid.

For **most** professional staff employees, this is pretty straightforward as they have a 1-to-1 relationship with their job.

For faculty and senior leaders with multiple job codes, this can be challenging.

Here is Yoji’s employee job history on the left. There are 66 rows, with different effective dates, different workforce actions, and Yoji has four distinct job codes. We want to be able to count Yoji as **ONE** employee, so we need to aggregate his history.

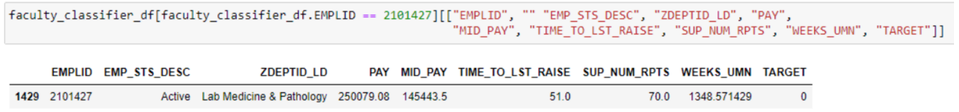


```
d_um_emp_df[d_um_emp_df.EMPLID == '2101427']
```

	UM_EMP_SID	UM_EMP_NK_SID	EMPLID	EMP_REC_NBR	EFF_DT	EFF_SEQ_NBR	FST_NM_TXT	LST_NM_TXT	INTERNET_ID	INST_EML_ADDR	...	ROW_EFF_DT	ROW_EXPIR_DT	RC
129107	129708	12703	2101427	0	11-JUN-18	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	07-AUG-19	31-DEC-99	
129108	129709	12703	2101427	0	16-OCT-17	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	07-AUG-19	31-DEC-99	
129109	129710	12703	2101427	0	15-SEP-17	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	07-AUG-19	31-DEC-99	
129110	129756	12705	2101427	2	14-DEC-15	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	02-JUN-20	31-DEC-99	
129111	129757	12705	2101427	2	30-JUN-15	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	02-JUN-20	31-DEC-99	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
129877	129706	12703	2101427	0	21-JAN-19	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	07-AUG-19	31-DEC-99	
129878	129707	12703	2101427	0	11-JUN-18	1	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	07-AUG-19	31-DEC-99	
129879	129753	12705	2101427	2	06-FEB-17	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	02-JUN-20	31-DEC-99	
129880	129754	12705	2101427	2	13-JUN-16	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	02-JUN-20	31-DEC-99	
129881	129755	12705	2101427	2	01-JAN-16	0	Yoji	Shimizu	SHIMI002	shimi002@umn.edu	...	02-JUN-20	31-DEC-99	

66 rows × 149 columns

Here is Yoji’s “row” of data after it has been aggregated; his IBS salary is summarized across all of his jobs, his ZDeptID belongs to his primary job row, and all of the other metrics are based on Yoji as **ONE** employee.



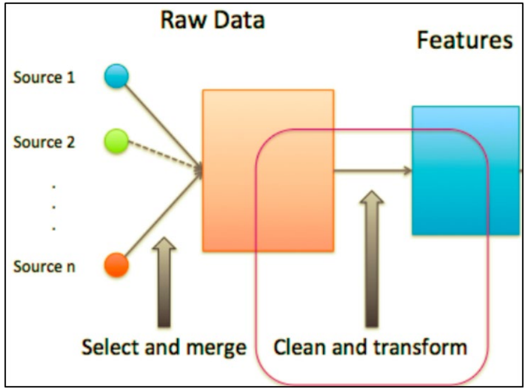
```
faculty_classifier_df[faculty_classifier_df.EMPLID == 2101427][["EMPLID", "" "EMP_STS_DESC", "ZDEPTID_LD", "PAY",
                                                                 "MID_PAY", "TIME_TO_LST_RAISE", "SUP_NUM_RPTS", "WEEKS_UMN", "TARGET"]]
```

	EMPLID	EMP_STS_DESC	ZDEPTID_LD	PAY	MID_PAY	TIME_TO_LST_RAISE	SUP_NUM_RPTS	WEEKS_UMN	TARGET
1429	2101427	Active	Lab Medicine & Pathology	250079.08	145443.5	51.0	70.0	1348.571429	0

D. Getting data from EDW - Feature Generation

What is a feature?

A feature is an individual measurable property or characteristic of a phenomenon being observed. Collecting and processing data can be an expensive and time consuming process. Therefore, choosing informative, discriminating and independent features is a crucial step for effective algorithms in classification.

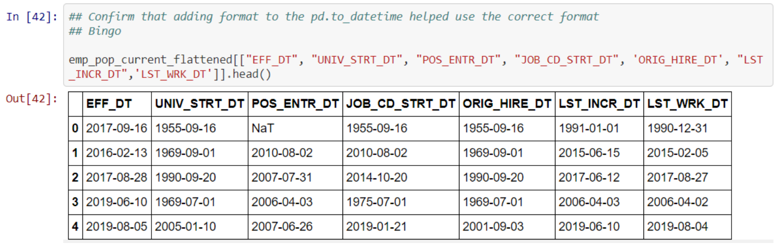


What is feature generation?

Feature generation is the process of creating new features from one or multiple features, for potential use in the statistical analysis. Usually this process is adding new information to the model to make it more accurate.

What is a "good feature"?

Anything that improves the performance of the model! By adding new features that encapsulate the feature interaction, new information becomes more accessible for the prediction model.



```
In [42]: ## Confirm that adding format to the pd.to_datetime helped use the correct format
         ## Bingo

         emp_pop_current_flattened[["EFF_DT", "UNIV_STRT_DT", "POS_ENTR_DT", "JOB_CD_STRT_DT", 'ORIG_HIRE_DT', "LST
         _INCR_DT",'LST_WRK_DT']].head()
```

Out[42]:

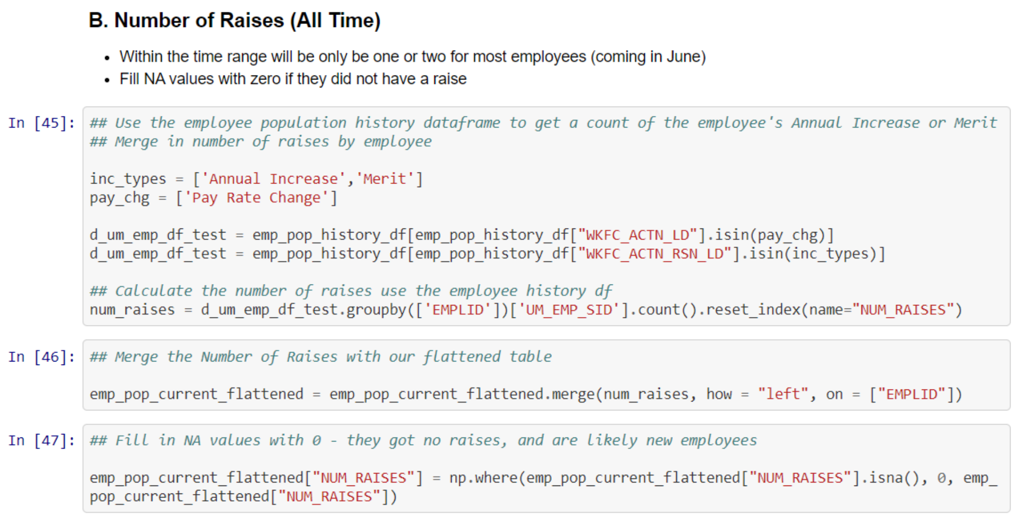
	EFF_DT	UNIV_STRT_DT	POS_ENTR_DT	JOB_CD_STRT_DT	ORIG_HIRE_DT	LST_INCR_DT	LST_WRK_DT
0	2017-09-16	1955-09-16	NaT	1955-09-16	1955-09-16	1991-01-01	1990-12-31
1	2016-02-13	1969-09-01	2010-08-02	2010-08-02	1969-09-01	2015-06-15	2015-02-05
2	2017-08-28	1990-09-20	2007-07-31	2014-10-20	1990-09-20	2017-06-12	2017-08-27
3	2019-06-10	1969-07-01	2006-04-03	1975-07-01	1969-07-01	2006-04-03	2006-04-02
4	2019-08-05	2005-01-10	2007-06-26	2019-01-21	2001-09-03	2019-06-10	2019-08-04

Here is some of the date fields from D_UM_EMP in their raw format. In order to make these more meaningful, they need to be transformed (by subtracting from today's date) and set to the same scale (either days, weeks, or years).

D. Getting data from EDW - Feature Generation, in depth example

Number of Raises (for entire history)

1. Filter the employee history data set on these specific Workforce Action/ Reason Descriptions:
   a. Action: Annual Increase, Merit
   b. Reason: Pay Rate Change
2. Group the data by emplid and count each UM_EMP_SID to get a unique count
3. For any NA values (i.e. employees that don't have any raises (new employees), fill them with 0. NA/ NULL values will get covered later.



B. Number of Raises (All Time)

- Within the time range will be only be one or two for most employees (coming in June)
- Fill NA values with zero if they did not have a raise

```
In [45]: ## Use the employee population history dataframe to get a count of the employee's Annual Increase or Merit
         ## Merge in number of raises by employee

         inc_types = ['Annual Increase','Merit']
         pay_chg = ['Pay Rate Change']

         d_um_emp_df_test = emp_pop_history_df[emp_pop_history_df["WKFC_ACTN_LD"].isin(pay_chg)]
         d_um_emp_df_test = emp_pop_history_df[emp_pop_history_df["WKFC_ACTN_RSN_LD"].isin(inc_types)]

         ## Calculate the number of raises use the employee history df
         num_raises = d_um_emp_df_test.groupby(['EMPLID'])['UM_EMP_SID'].count().reset_index(name="NUM_RAISES")
```

```
In [46]: ## Merge the Number of Raises with our flattened table

         emp_pop_current_flattened = emp_pop_current_flattened.merge(num_raises, how = "left", on = ["EMPLID"])
```

```
In [47]: ## Fill in NA values with 0 - they got no raises, and are likely new employees

         emp_pop_current_flattened["NUM_RAISES"] = np.where(emp_pop_current_flattened["NUM_RAISES"].isna(), 0, emp_
         pop_current_flattened["NUM_RAISES"])
```

- This was how it was executed using the CSV files. If we were to rebuild the feature engineering process in UM Analytics, we would want to rewrite this logic in a better way.

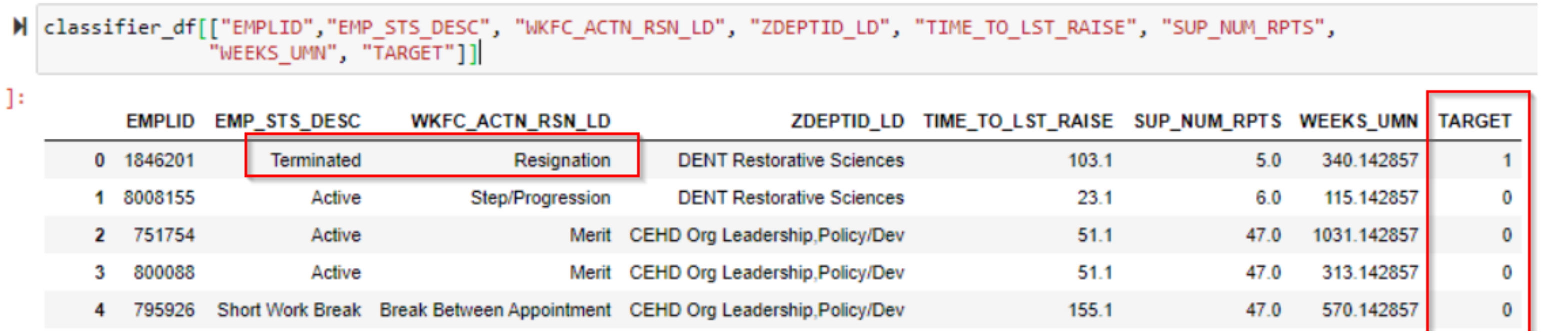
D. Getting data from EDW - Creating Target Variables

How do we create our target variables?

We assign a label of **1** for any employee with:

- Employee Status Code = 'T'
- Workforce Action Reason Code = 'RES' (Resignation)
- Workforce Action Reason Code != 'EIE' (Entered in Error)
- University Termination Date != ' ' or NULL

We assign a label of **0** for any employee that is still Active.



```
classifier_df[["EMPLID","EMP_STS_DESC", "WKFC_ACTN_RSN_LD", "ZDEPTID_LD", "TIME_TO_LST_RAISE", "SUP_NUM_RPTS",
              "WEEKS_UMN", "TARGET"]]
```

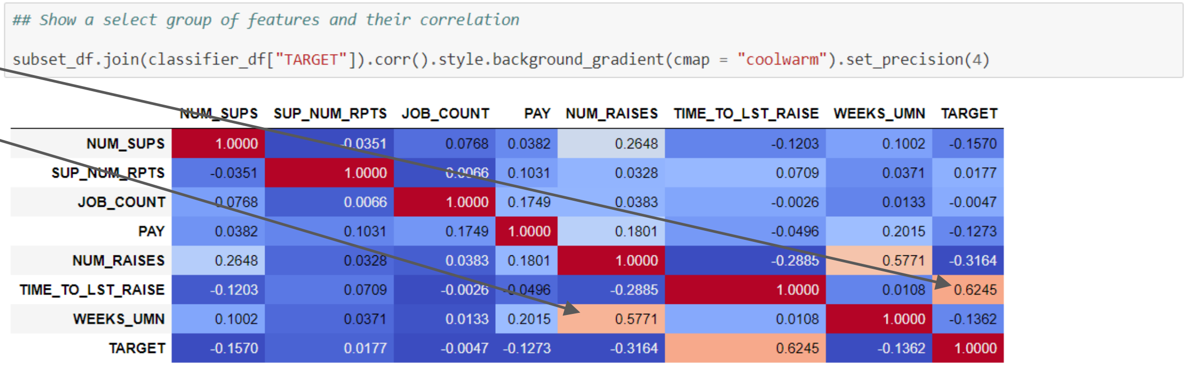
]:

	EMPLID	EMP_STS_DESC	WKFC_ACTN_RSN_LD	ZDEPTID_LD	TIME_TO_LST_RAISE	SUP_NUM_RPTS	WEEKS_UMN	TARGET
0	1846201	Terminated	Resignation	DENT Restorative Sciences	103.1	5.0	340.142857	1
1	8008155	Active	Step/Progression	DENT Restorative Sciences	23.1	6.0	115.142857	0
2	751754	Active	Merit	CEHD Org Leadership,Policy/Dev	51.1	47.0	1031.142857	0
3	800088	Active	Merit	CEHD Org Leadership,Policy/Dev	51.1	47.0	313.142857	0
4	795926	Short Work Break	Break Between Appointment	CEHD Org Leadership,Policy/Dev	155.1	47.0	570.142857	0

D. Getting data from EDW - Correlation Matrix and Feature Interaction

Correlation is an indication about the changes between two variables, or the way in which one set of data may correspond to another set. In machine learning, we can observe how our features correspond with the target variable. We can plot correlation matrices like the one below to show which variable is having a high or low correlation in respect to another variable or the **target** .

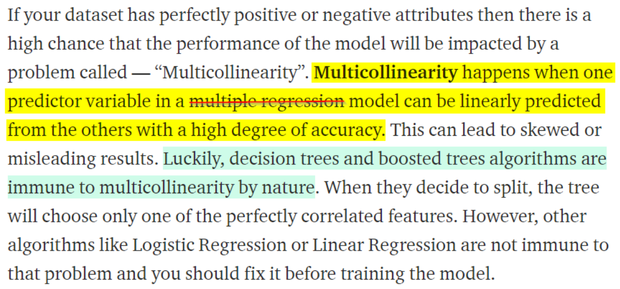
We (typically) do not want features correlated to **one another** or the **target** .
THIS IS BAD!



```
## Show a select group of features and their correlation

subset_df.join(classifier_df["TARGET"]).corr().style.background_gradient(cmap = "coolwarm").set_precision(4)
```

	NUM_SUPS	SUP_NUM_RPTS	JOB_COUNT	PAY	NUM_RAISES	TIME_TO_LST_RAISE	WEEKS_UMN	TARGET
NUM_SUPS	1.0000	-0.0351	0.0768	0.0382	0.2648	-0.1203	0.1002	-0.1570
SUP_NUM_RPTS	-0.0351	1.0000	0.0066	0.1031	0.0328	0.0709	0.0371	0.0177
JOB_COUNT	0.0768	0.0066	1.0000	0.1749	0.0383	-0.0026	0.0133	-0.0047
PAY	0.0382	0.1031	0.1749	1.0000	0.1801	-0.0496	0.2015	-0.1273
NUM_RAISES	0.2648	0.0328	0.0383	0.1801	1.0000	-0.2885	0.5771	-0.3164
TIME_TO_LST_RAISE	-0.1203	0.0709	-0.0026	-0.0496	-0.2885	1.0000	0.0108	0.6245
WEEKS_UMN	0.1002	0.0371	0.0133	0.2015	0.5771	0.0108	1.0000	-0.1362
TARGET	-0.1570	0.0177	-0.0047	-0.1273	-0.3164	0.6245	-0.1362	1.0000

And this is why it's bad:



If your dataset has perfectly positive or negative attributes then there is a high chance that the performance of the model will be impacted by a problem called — "Multicollinearity". **Multicollinearity** happens when one predictor variable in a ~~multiple regression~~ model can be linearly predicted from the others with a high degree of accuracy. This can lead to skewed or misleading results. Luckily, decision trees and boosted trees algorithms are immune to multicollinearity by nature. When they decide to split, the tree will choose only one of the perfectly correlated features. However, other algorithms like Logistic Regression or Linear Regression are not immune to that problem and you should fix it before training the model.

Correlation =/ =causation

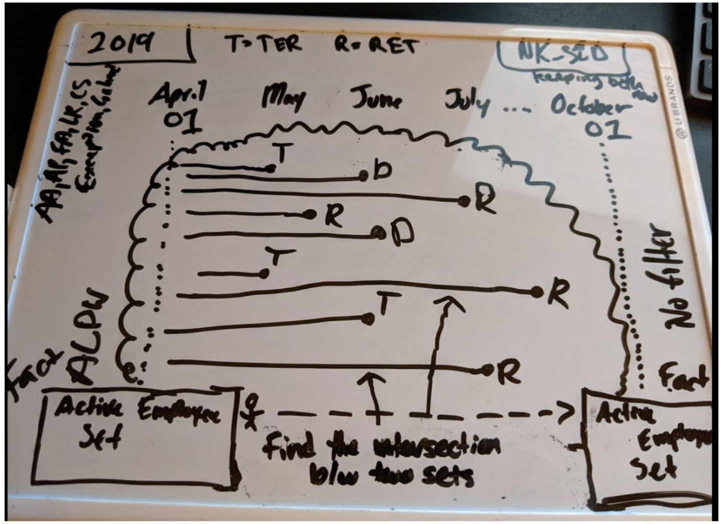
D. Getting data from EDW - Implement Time Lag

Why do we need to implement time lag?

- We need to look into the crystal ball into the future..but we can't do that with production data! We don't know who is going to quit *because they haven't quit...*yet.
- We need to set up a timeline and a cutoff so that we can make observations about individuals who quit, learn their features, and then make predictions.

How do we implement time lag?

- "Snap the line" on Time A to see who was active in past
- "Snap the line" on Time B to see who is **STILL** active from Time A
- Observe the employees who are dropping off between those two points
  - Are they terminating?
  - Are they retiring?
  - Are they moving job codes?



E. Training and Testing Split

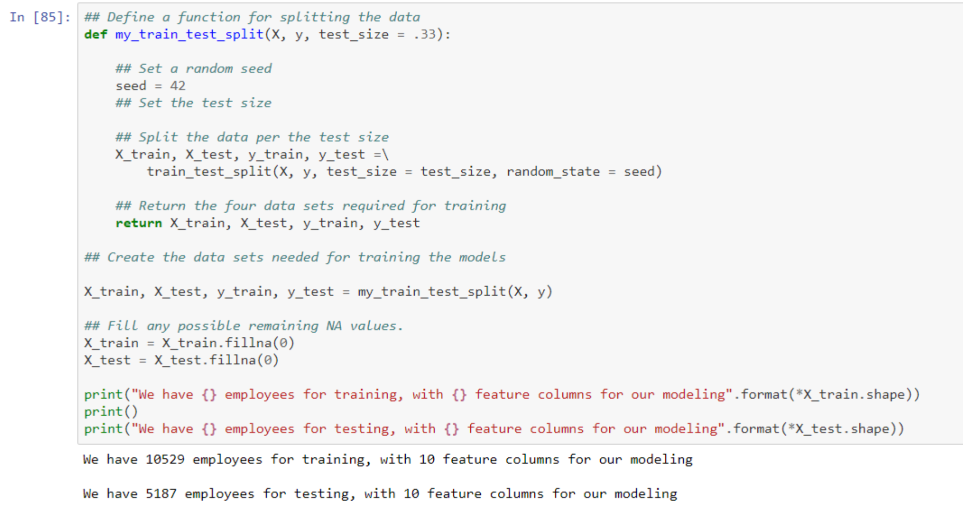
- Why do we need a *training* set?
- Why do we need a *testing* set?
- Why do we need a *holdout* set?
- How do we set different thresholds?

E. Training and Testing Split

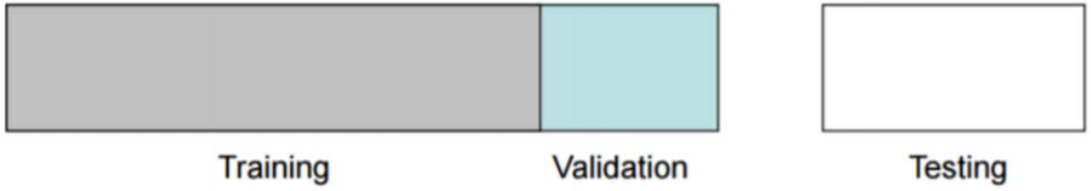
We calibrated and “fit” our classification models on a **training** dataset, a subset of employees that are used to “learn” latent patterns between our factors and whether an employees is active or resigned.

However, the training data used to train and tune your model will only produce the best model for that particular training set. This means that the model is unlikely to perform as well on new data. So what we do is create a **validation** set, which is essentially a subset of the training data. The validation set is how we can evaluate the model performance while fine-tuning the hyperparameters (which we cover in more depth later).

We set aside a **testing** set to evaluate how well the model performs on **unobserved** data points and to provide an estimate of how well the model will do on future data. This is never seen by the model until the very end!



```
In [85]: ## Define a function for splitting the data
         def my_train_test_split(X, y, test_size = .33):

             ## Set a random seed
             seed = 42
             ## Set the test size

             ## Split the data per the test size
             X_train, X_test, y_train, y_test =\
                 train_test_split(X, y, test_size = test_size, random_state = seed)

             ## Return the four data sets required for training
             return X_train, X_test, y_train, y_test

         ## Create the data sets needed for training the models

         X_train, X_test, y_train, y_test = my_train_test_split(X, y)

         ## Fill any possible remaining NA values.
         X_train = X_train.fillna(0)
         X_test = X_test.fillna(0)

         print("We have {} employees for training, with {} feature columns for our modeling".format(*X_train.shape))
         print()
         print("We have {} employees for testing, with {} feature columns for our modeling".format(*X_test.shape))

         We have 10529 employees for training, with 10 feature columns for our modeling

         We have 5187 employees for testing, with 10 feature columns for our modeling
```



E. Training and Testing Split, cont.

The training set is the material through which the computer learns how to process information. ML uses an algorithm(s) to "mimic" the human brain, which can take in diverse inputs and weigh them, in order to produce activations in individual neurons in the brain. ML uses "artificial" neurons to replicate (albeit not as well as the brain) this process with software; ML and neural network programs provide highly detailed models of how our human thought process works.

Lecture 9.8 — Neural Networks Learning | Autonomous Driving Example — [Andrew Ng]

F. Preprocessing the data

- Missing value imputation
  - Why is this needed?
- Numeric scaling
  - Why is this needed?
- Categorical encoding
  - One hot encoding
  - Why is this needed?
- Class imbalance
  - Why is this a problem and how do we deal with it?

F. Preprocessing the data - Imputing Missing Values

Missing values were imputed to guarantee that all algorithms would be able to handle the data. For most of the numeric features, any missing or NULL values were imputed (filled) with 0. Here are some examples from CAL Team 1:

- *Institutional Base Salary*: Where the emplid had no institutional base salary, Annual Rate Amnt was used instead.
- *Number of Raises*: Where there was no raises, set to 0.
- *Time to Last Raise*: Where Last Raise Date was empty and Job Code Start Date prior to prediction date, set to 0.

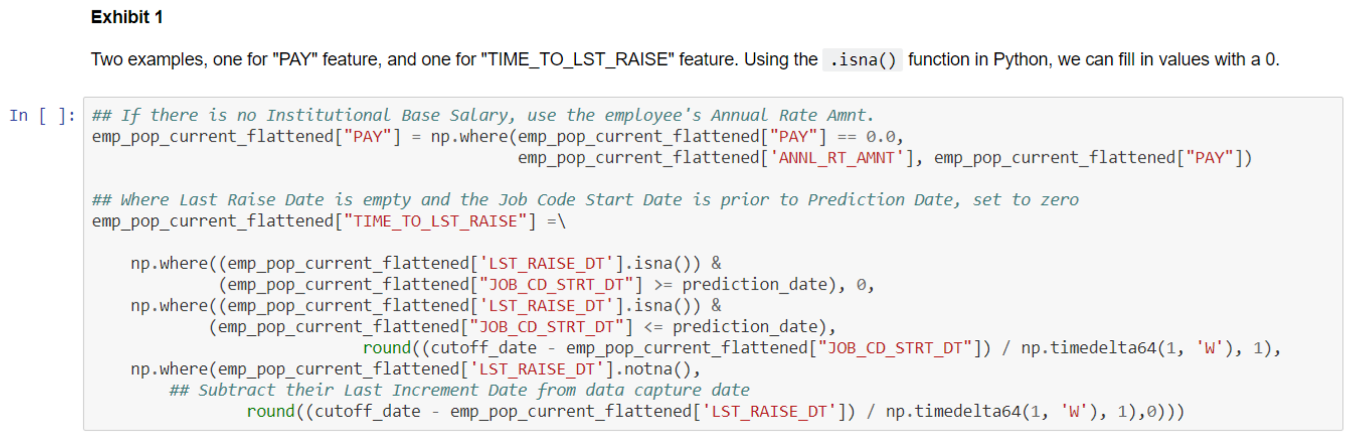


Exhibit 1

Two examples, one for "PAY" feature, and one for "TIME_TO_LST_RAISE" feature. Using the `.isna()` function in Python, we can fill in values with a 0.

```
In [ ]: ## If there is no Institutional Base Salary, use the employee's Annual Rate Amnt.
        emp_pop_current_flattened["PAY"] = np.where(emp_pop_current_flattened["PAY"] == 0.0,
                                                    emp_pop_current_flattened['ANNL_RT_AMNT'], emp_pop_current_flattened["PAY"])

        ## Where Last Raise Date is empty and the Job Code Start Date is prior to Prediction Date, set to zero
        emp_pop_current_flattened["TIME_TO_LST_RAISE"] =\

            np.where((emp_pop_current_flattened['LST_RAISE_DT'].isna()) &
                     (emp_pop_current_flattened["JOB_CD_STRT_DT"] >= prediction_date), 0,
            np.where((emp_pop_current_flattened['LST_RAISE_DT'].isna()) &
                    (emp_pop_current_flattened["JOB_CD_STRT_DT"] <= prediction_date),
                                    round((cutoff_date - emp_pop_current_flattened["JOB_CD_STRT_DT"]) / np.timedelta64(1, 'W'), 1),
            np.where(emp_pop_current_flattened['LST_RAISE_DT'].notna(),
                ## Subtract their Last Increment Date from data capture date
                        round((cutoff_date - emp_pop_current_flattened['LST_RAISE_DT']) / np.timedelta64(1, 'W'), 1),0)))
```

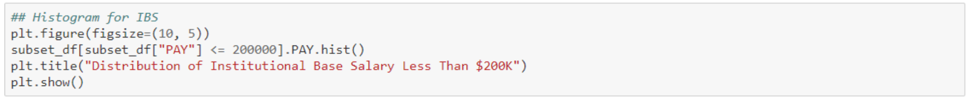
- This is done BEFORE running any of the models, and is very important

F. Preprocessing the data - Numeric Scaling

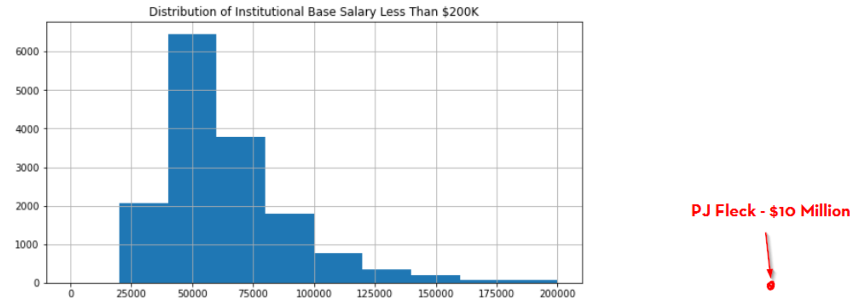
Why?

Feature scaling is a data mining approach to adjust the range of features and reducing disparate feature scales. This may help some machine learning classifiers perform better, because significant scale gaps among features are generally not favored within the optimization stage of these algorithms. In the OHR data, some features generally have significantly disparate scales, i.e. institutional base salary can range from $10,000 to $1,100,000. For this project, both normalization and standardization were performed on the final data set prior to model building. (see Exhibit 3)

Histogram for Institutional Base Salary (only showcasing less than Professional Staff making less than $200K)



```
## Histogram for IBS
plt.figure(figsize=(10, 5))
subset_df[subset_df["PAY"] <= 200000].PAY.hist()
plt.title("Distribution of Institutional Base Salary Less Than $200K")
plt.show()
```

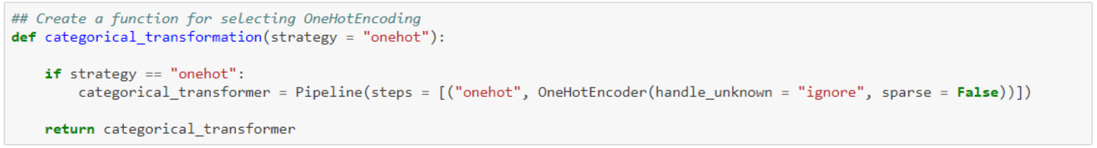


- Normalization is the process of scaling individual samples to have unit norm; many machine learning estimators might behave badly if the individual features do not more or less look like standard normally distributed data.

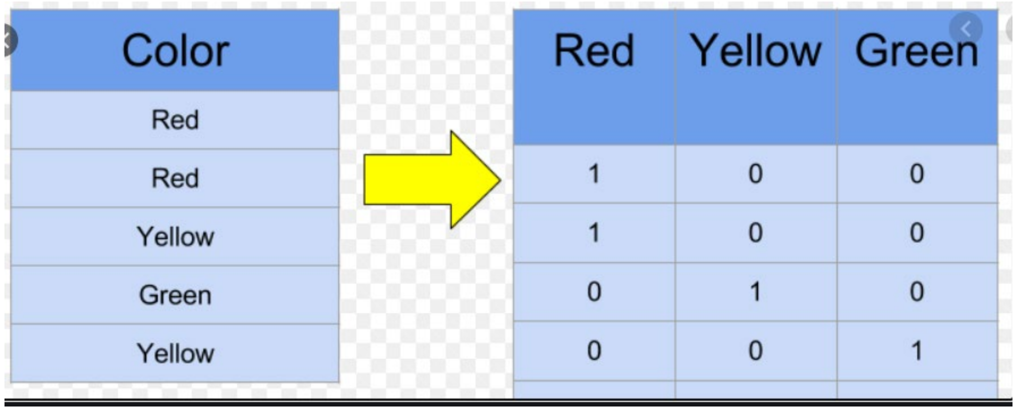
F. Preprocessing the data - Categorical Variables & Encoding

Encode categorical features as a one-hot numeric array. This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs with the standard kernels.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka 'one-of-K' or 'dummy) encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array (depending on the sparse parameter).



```
## Create a function for selecting OneHotEncoding
def categorical_transformation(strategy = "onehot"):

    if strategy == "onehot":
        categorical_transformer = Pipeline(steps = [("onehot", OneHotEncoder(handle_unknown = "ignore", sparse = False))])

    return categorical_transformer
```

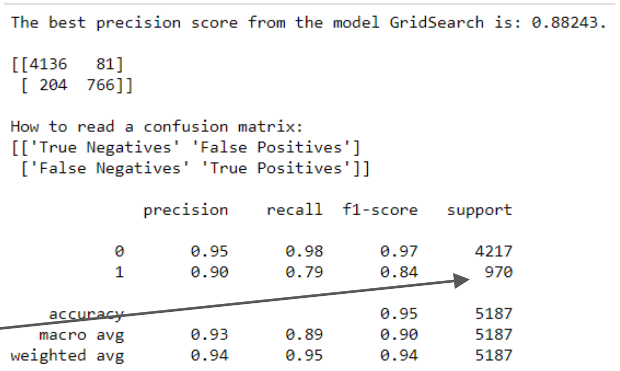


Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

F. Preprocessing the data - Class Imbalance

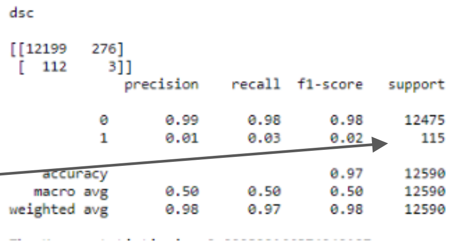
Why is this a problem?



```
The best precision score from the model GridSearch is: 0.88243.

[[4136   81]
 [ 204  766]]

How to read a confusion matrix:
[['True Negatives' 'False Positives']
 ['False Negatives' 'True Positives']]

              precision    recall  f1-score   support

           0       0.95      0.98      0.97      4217
           1       0.90      0.79      0.84       970

    accuracy                           0.95      5187
   macro avg       0.93      0.89      0.90      5187
weighted avg       0.94      0.95      0.94      5187
```

Entire 5-year history

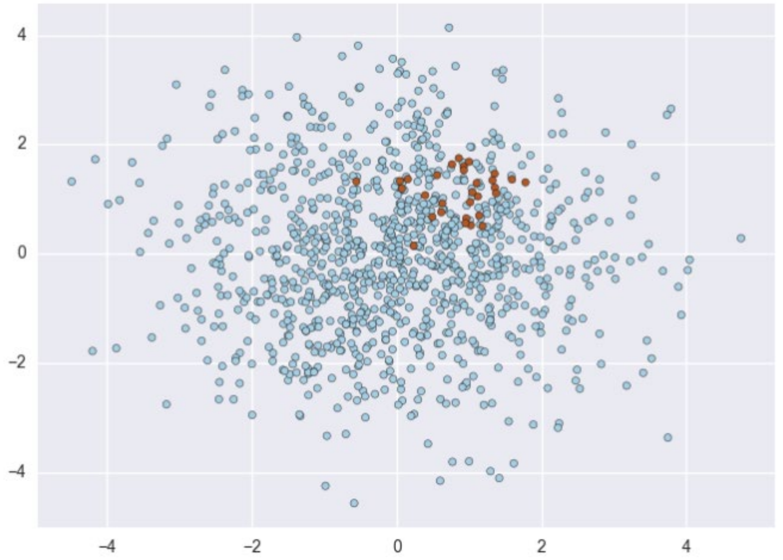
Lots of support



```
dsc

[[12199   276]
 [  112     3]]
              precision    recall  f1-score   support

           0       0.99      0.98      0.98     12475
           1       0.01      0.03      0.02       115

    accuracy                           0.97     12590
   macro avg       0.50      0.50      0.50     12590
weighted avg       0.98      0.97      0.98     12590
```

6 month time frame

Little support



Red is resigned employee

Rest are active employees.

How do we deal with it?

Balance the training set in some way:

- Oversample the minority class.
- Undersample the majority class.
- Synthesize new minority classes.

G. Fitting the models and hyperparameter tuning

- Cross validation / GridSearchCV
- Hyperparameter tuning
- Model decision making process (sample of LightGBM and DecisionTrees)

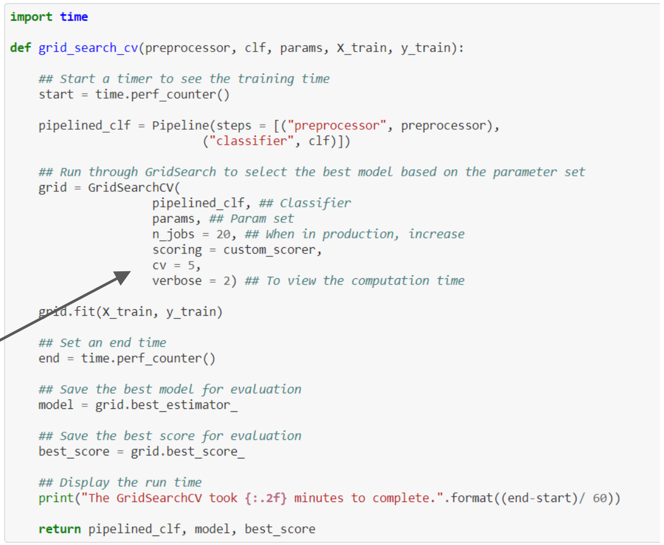
G. Fitting the models and hyperparameter tuning

Cross validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a **limited** data sample.

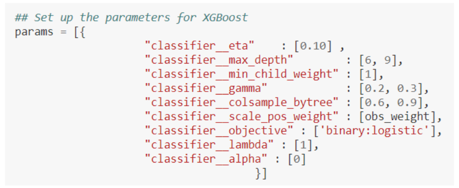
The procedure has a single parameter called *k* that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for *k* is chosen, it may be used in place of *k* in the reference to the model, such as **k = 5** becoming *5*-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data, and is also used to assess the generalization ability of an algorithm on a dataset. It can prevent a model from overfitting that is possibly caused by the high complexity of the model.



```python
import time

def grid_search_cv(preprocessor, clf, params, X_train, y_train):

    ## Start a timer to see the training time
    start = time.perf_counter()

    pipelined_clf = Pipeline(steps = [("preprocessor", preprocessor),
                                       ("classifier", clf)])

    ## Run through GridSearch to select the best model based on the parameter set
    grid = GridSearchCV(
                        pipelined_clf, ## Classifier
                        params, ## Param set
                        n_jobs = 20, ## When in production, increase
                        scoring = custom_scorer,
                        cv = 5,
                        verbose = 2) ## To view the computation time

    grid.fit(X_train, y_train)

    ## Set an end time
    end = time.perf_counter()

    ## Save the best model for evaluation
    model = grid.best_estimator_

    ## Save the best score for evaluation
    best_score = grid.best_score_

    ## Display the run time
    print("The GridSearchCV took {:.2f} minutes to complete.".format((end-start)/ 60))

    return pipelined_clf, model, best_score
```

Sample parameter grid



```python
## Set up the parameters for XGBoost
params = [{
                    "classifier__eta"    : [0.10] ,
                    "classifier__max_depth"         : [6, 9],
                    "classifier__min_child_weight" : [1],
                    "classifier__gamma"             : [0.2, 0.3],
                    "classifier__colsample_bytree" : [0.6, 0.9],
                    "classifier__scale_pos_weight" : [obs_weight],
                    "classifier__objective" : ['binary:logistic'],
                    "classifier__lambda" : [1],
                    "classifier__alpha" : [0]
                                    }]
```

GridSearchCV

- Exhaustive search over specified parameter values for an estimator.
- The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

G. Fitting the models and hyperparameter tuning

Hyperparameter tuning

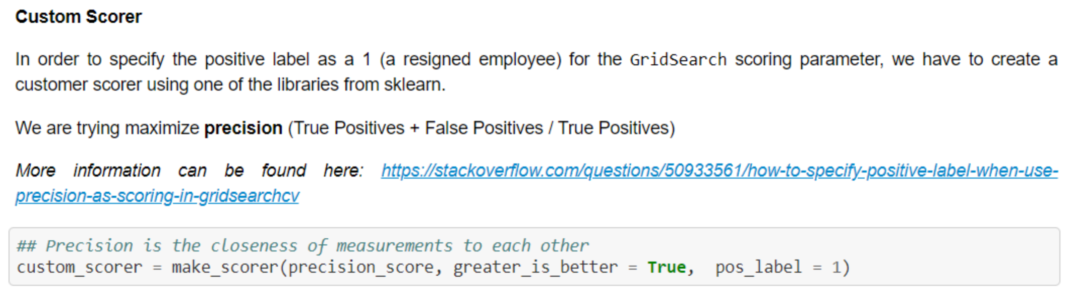
Hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process.

The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called *hyperparameters*, and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data. The *objective function* takes a tuple of hyperparameters and returns the associated loss.

Main approaches

- Grid Search (GridSearchCV, *sklearn*)
- Random Search (RandomizedSearchCV, *sklearn*)
- Bayes Optimization (Scikit-Optimize)
- Early-stopping (Neural Networks)

Custom scoring/objective function (from CAL Team 1)



Custom Scorer

In order to specify the positive label as a 1 (a resigned employee) for the `GridSearch` scoring parameter, we have to create a customer scorer using one of the libraries from sklearn.

We are trying maximize **precision** (True Positives + False Positives / True Positives)

More information can be found here: https://stackoverflow.com/questions/50933561/how-to-specify-positive-label-when-use-precision-as-scoring-in-gridsearchcv

```
## Precision is the closeness of measurements to each other
custom_scorer = make_scorer(precision_score, greater_is_better = True,  pos_label = 1)
```

H. Model selection & performance evaluation

- Performance metrics
- Testing against the "holdout set"
- Confusion matrix
- Lift curve
- Precision-recall curve

H. Model selection & performance evaluation

Performance Metrics, pt. 1

Precision/Recall/Accuracy - Background Information

During our initial discussions, one of the requirements for successful delivery is a highly "accurate" model. After further exploration, we see that only around 15% of employees resign during the five-year history of data.

Accuracy is simply the number of correct predictions over total predictions. Our model could achieve **85%** accuracy by simply predicting that that the employee will not resign every single time the model makes a prediction, since 1 - 15% = 85%. However, this model is almost completely <u>useless</u>, as our goal is to predict resigned employees. In this case, accuracy is not the best metric to strive for.

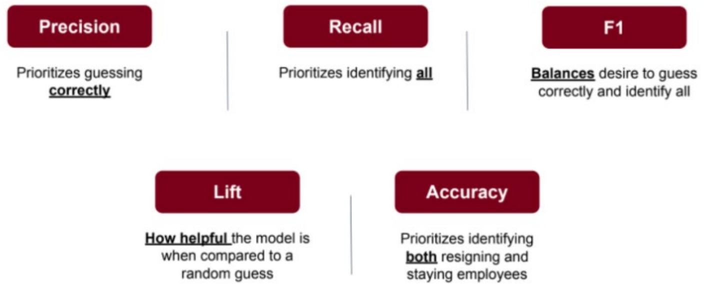
There are many other classification metrics that can be used instead:

- **Precision** : how many employees selected from our model will actually churn?
- **Recall** : how many employees who could potentially resign did our model actually identify?
- **Lift** : how much better is my model at identifying employees who resigned vs. random chance?

H. Model selection & performance evaluation

Performance Metrics, pt. 2

- **Accuracy** : The number of correct predictions divided by the total number of predictions, multiplied by 100.
- **Precision** : The precision is the ratio **tp / (tp + fp)** where tp is the number of true positives and fp the number of **false positives** . The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1 and the worst value is 0.
- **Recall** : The recall is the ratio **tp / (tp + fn)** where tp is the number of true positives and fn the number of **false negatives** . The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0.
- **F1**: The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is: F1 = 2 (precision recall) / (precision + recall)
- **Lift** : A measure of the performance of a classification model at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model. A targeting model is doing a good job if the response within the target is much better than the average for the population as a whole. Lift is simply the ratio of these values: target response divided by average response.

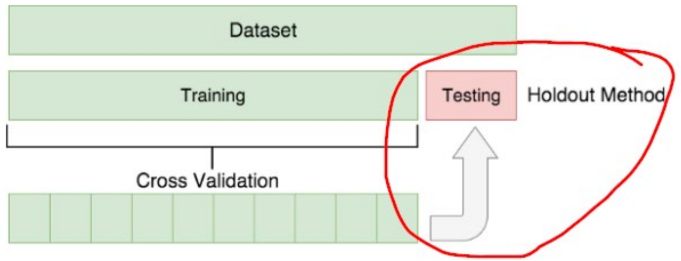

Precision
Prioritizes guessing correctly
Recall
Prioritizes identifying all
F1
Balances desire to guess correctly and identify all
Lift
How helpful the model is when compared to a random guess
Accuracy
Prioritizes identifying both resigning and staying employees

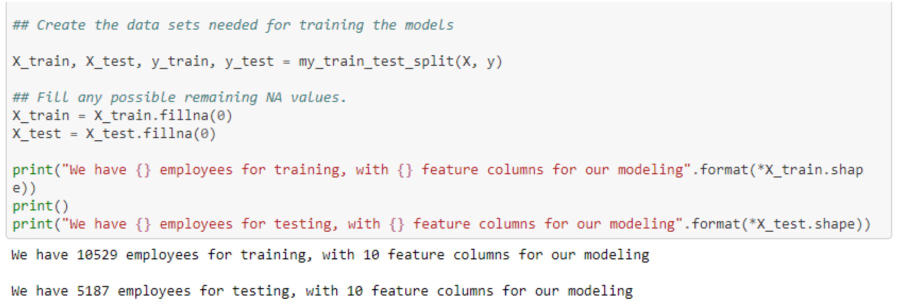
H. Model selection & performance evaluation

Testing against the “holdout set”

A **holdout** set is another word for the “testing” set. The training set is what the model is trained on, and the testing set is used to see how well that model performs on unseen data. One of the most common splits when using the hold-out method is using 80% of the data for training and the remaining 20% of the data for testing. A **spicier** version of this might be a 66%/ 33%split.

A holdout subset provides a final estimate of the machine learning model’s performance *after* it has been **trained** and **validated** . Holdout sets **should never** be used to make decisions about which algorithms to use or for improving or tuning algorithms.





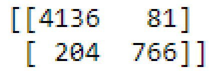
```
## Create the data sets needed for training the models

X_train, X_test, y_train, y_test = my_train_test_split(X, y)

## Fill any possible remaining NA values.
X_train = X_train.fillna(0)
X_test = X_test.fillna(0)

print("We have {} employees for training, with {} feature columns for our modeling".format(*X_train.shap
e))
print()
print("We have {} employees for testing, with {} feature columns for our modeling".format(*X_test.shape))
```

```
We have 10529 employees for training, with 10 feature columns for our modeling

We have 5187 employees for testing, with 10 feature columns for our modeling
```

H. Model selection & performance evaluation

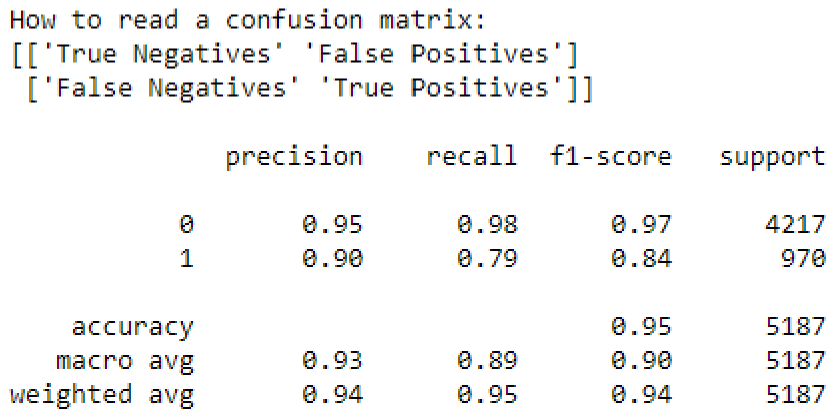
Confusion matrix

A confusion matrix is a table that is often used to **describe the performance of a classification model (or "classifier")** on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.



```
[[4136   81]
 [ 204  766]]
```

Sample confusion matrix



```
How to read a confusion matrix:
[['True Negatives' 'False Positives']
 ['False Negatives' 'True Positives']]

              precision    recall  f1-score   support

           0       0.95      0.98      0.97      4217
           1       0.90      0.79      0.84       970

    accuracy                           0.95      5187
   macro avg       0.93      0.89      0.90      5187
weighted avg       0.94      0.95      0.94      5187
```

- **true positives (TP)** : These are cases in which we predicted yes (the employee will resign), and the employee **did** resign.
- **true negatives (TN)** : We predicted no, and the employee is still activ
- **false positives (FP)** : We predicted yes, **but** the employee is still activ
- **false negatives (FN)** : We predicted no, but the employee **did** resign

In this example, our model predicted that for the Faculty employees, 184 employees would resign out of the 1,500 active employees in the testing data; of those predicted to resign, 162 were correct. This gets us a precision score of 88%.

H. Model selection & performance evaluation

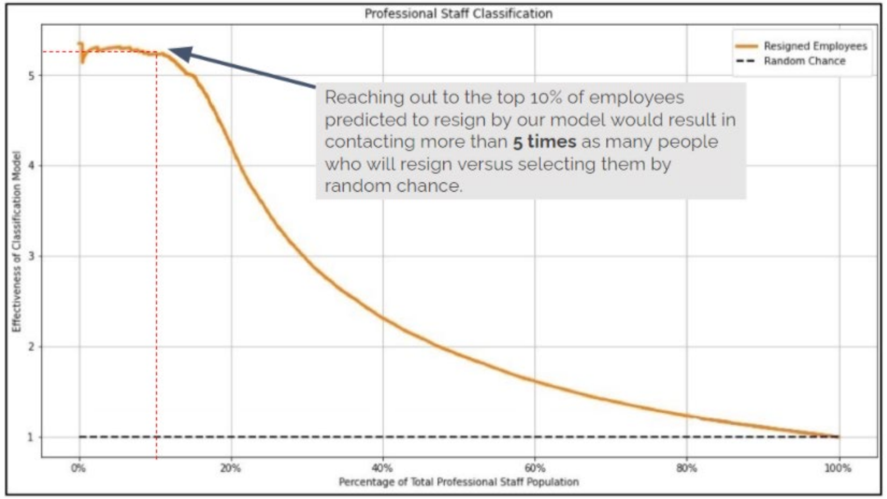
Lift curve

One useful way to think of a lift curve is to consider a data mining model that attempts to identify the likely responders to a mailing by assigning each case a "probability of responding" score. The lift curve helps us determine how effectively we can "skim the cream" by selecting a relatively small number of cases and getting a relatively large portion of the responders.

The lift will vary with the number of cases we choose to act on. A good classifier will give us a high lift when we act on only a few cases (i.e. use the prediction for the ones at the top).

In the case of predicting when an employee will churn, based on a **sample** Professional Staff classification model, by reaching out to the top 10% of employees predicted to resign by our model, it would result in our leaders contacting 5 times as many employees who would resign versus blinding drawing them out of a hat, or flipping a coin.

```
plot_lift_curve(lgb_model, X_test, y_test)
```



Reaching out to the top 10% of employees predicted to resign by our model would result in contacting more than **5 times** as many people who will resign versus selecting them by random chance.

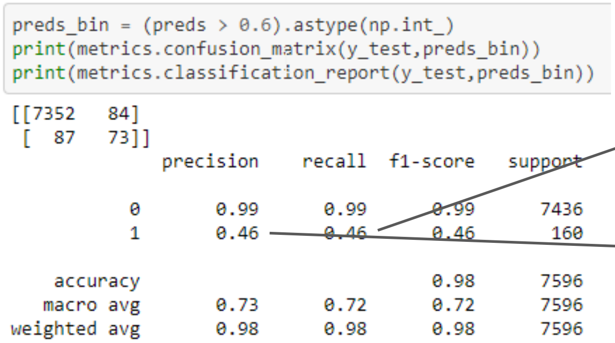
This point being called out is known as the '**Maximum Lift point**". The general rule is that the higher this point is, the better our model is performing, as there is a lot of real positive labels in a proportion of our population which has a very low probability of being positive (low chance of resigning).

H. Model selection & performance evaluation

Precision -recall curve

A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve.

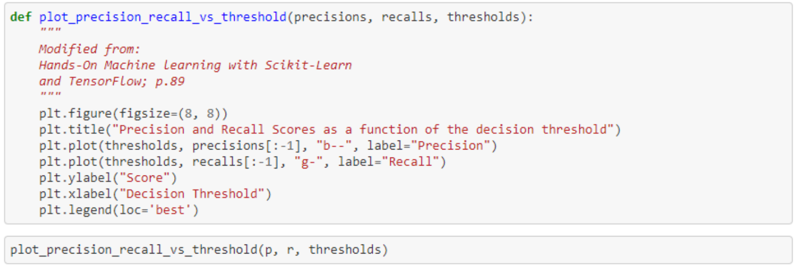
A no-skill classifier is one that cannot discriminate between the classes and would predict a random class or a constant class in all cases. The no-skill line changes based on the distribution of the positive to negative classes. It is a horizontal line with the value of the ratio of positive cases in the dataset. For a balanced dataset, this is 0.5.



```
preds_bin = (preds > 0.6).astype(np.int_)
print(metrics.confusion_matrix(y_test,preds_bin))
print(metrics.classification_report(y_test,preds_bin))
```

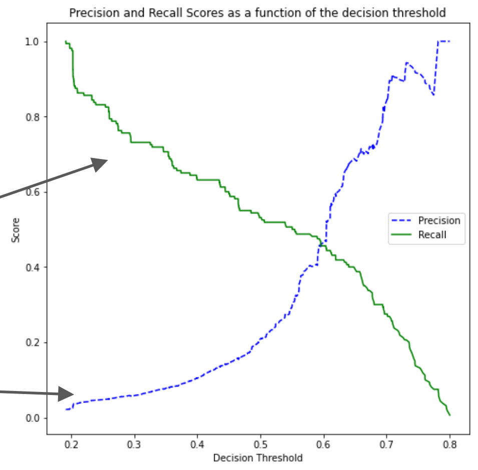
```
[[7352   84]
 [  87   73]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      7436
           1       0.46      0.46      0.46       160

    accuracy                           0.98      7596
   macro avg       0.73      0.72      0.72      7596
weighted avg       0.98      0.98      0.98      7596
```

Sample precision -recall curve



```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    """
    Modified from:
    Hands-On Machine learning with Scikit-Learn
    and TensorFlow; p.89
    """
    plt.figure(figsize=(8, 8))
    plt.title("Precision and Recall Scores as a function of the decision threshold")
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.ylabel("Score")
    plt.xlabel("Decision Threshold")
    plt.legend(loc='best')
```

```
plot_precision_recall_vs_threshold(p, r, thresholds)
```



I. Reference Materials

[2] Dawson, C. (2019, March 7). Churn Prediction and Prevention in Python. Retrieved from https://towardsdatascience.com/churn-prediction-and-prevention-in-python-2d454e5fd9a5

[8] Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. Retreived from https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[9] Radewagen, Rick & Wu, Ye. (2017, June 7). 7 Techniques to Handle Imbalanced Data. *KDnuggets*. Retreived from https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html

[10] Riso, Rebecca. (2019, March 27). Imbalanced Classes: Part 2; Avoiding Imbalanced Class Pitfalls in Classification. *Towards Data Science*. Retreived from https://towardsdatascience.com/imbalanced-class-sizes-and-classification-models-a-cautionary-tale-part-2-cf371500d1b3

[17] Zhao, Yue & Hryniewicki, Maciej & Cheng, Francesca & Fu, Boyang & Zhu, Xiaoyu. (2019). Employee Turnover Prediction with Machine Learning: A Reliable Approach. *Intelligent Systems and Applications (IntelliSys)*. Retreived from https://www.researchgate.net/publication/328796091_Employee_Turnover_Prediction_with_Machine_Learning_A_Reliable_Approach

QUESTIONS?

Appendices/More Information

Machine Learning Basics

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." Tom Mitchell, 1997

"[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed." - Arthur Samuel, 1959

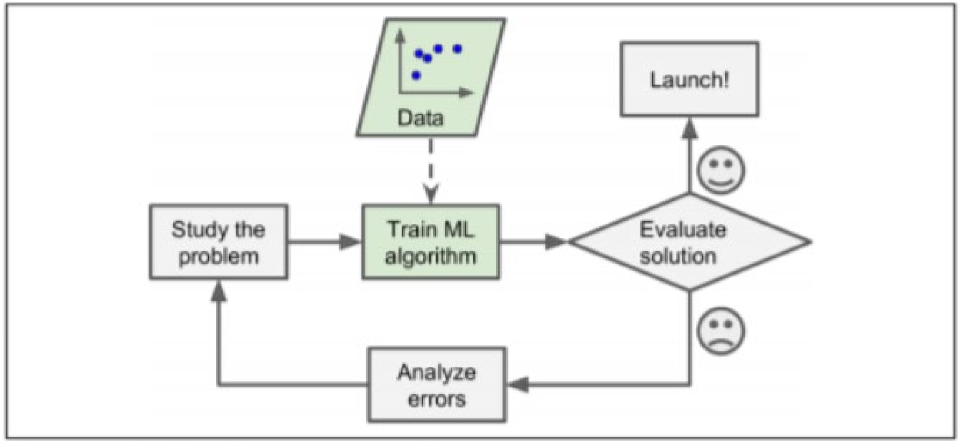


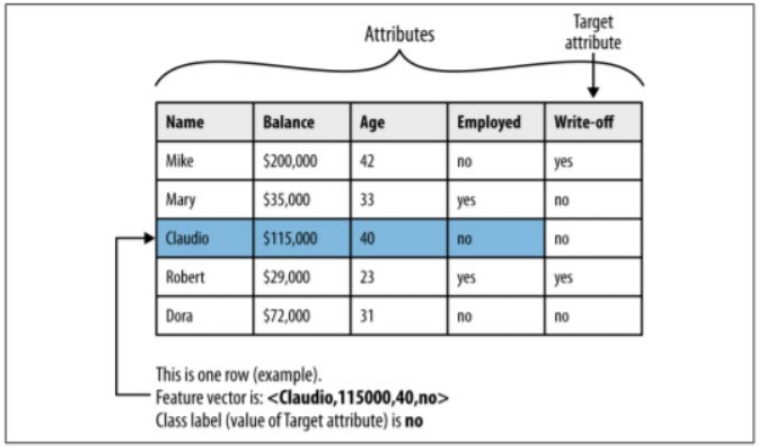
Figure 1-2. Machine Learning approach

Classification Modeling Basics

A typical supervised learning task is classification. An email spam filter is a good example of this: it is trained with many example emails along with their class (spam or not spam), and it must learn how to classify new emails.

Classification and class probability estimation

- How likely will a customer to respond to our mailing campaign?
- How likely is it that this patient will have diabetes based on previous medical history?
- How likely will this customer default on their loan based on their credit history (and that of others similar to this person)?


Attributes
Target attribute

Name	Balance	Age	Employed	Write-off
Mike	$200,000	42	no	yes
Mary	$35,000	33	yes	no
Claudio	$115,000	40	no	no
Robert	$29,000	23	yes	yes
Dora	$72,000	31	no	no

This is one row (example).
Feature vector is: **<Claudio,115000,40,no>**
Class label (value of Target attribute) is **no**

Figure 3-1. Data mining terminology for a supervised classification problem. The problem is supervised because it has a target attribute and some "training" data where we know the value for the target attribute. It is a classification (rather than regression) problem because the target is a category (yes or no) rather than a number.

"Name", "Balance", "Age", "Employed" are all **FEATURES, FACTORS, ATTRIBUTES.**

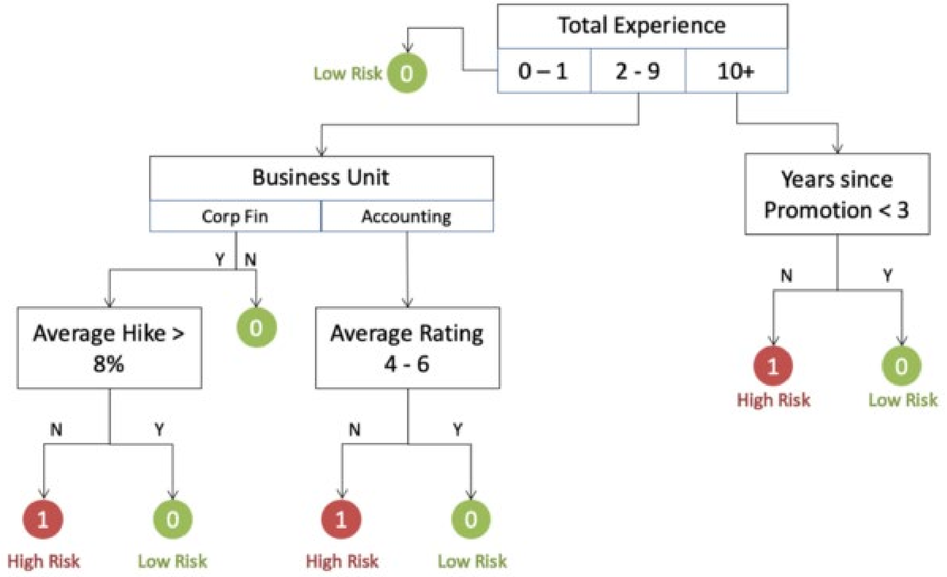
"Write-off" is a **TARGET** or **LABEL**.

How decision trees work, pt. I

Decision trees are powerful predictive algorithms that, with proper care, can also be very effective at explaining *why* someone might resign.

From the example, someone with 2-9 years of experience in the 'Corp Fin' business unit with an 'Average Hike' less than or equal to 8% is high risk of resigning.

Need to be cautious, though, as decision trees can become extremely complex and the *why* can become difficult to extract!


Total Experience
0 – 1
2 - 9
10+
Low Risk
0
Business Unit
Corp Fin
Accounting
Y
N
Years since Promotion < 3
N
Y
1
High Risk
0
Low Risk
Average Hike > 8%
0
Average Rating 4 - 6
N
Y
1
High Risk
0
Low Risk
N
Y
1
High Risk
0
Low Risk