David_Riser_DataChallenge1

February 20, 2019

1 Data Challenge 1: David Riser

Our first data challenge asks us to predict employee turnover, decide the main factors which drive it, and offer advice about how to improve employee retention.

1.1 Table of Contents:

1.2 1. Problem Statement and Objectives

1.2.1 1.1 Basic Problem Statement

Employee attrition (an employee quits) is expensive for the company who loses the employee, often costing them over $100K. For this data challenge we've been asked to predict which employees are at risk of quitting, and provide some actionable insights on how to improve the situation. We've also been asked which factors (features) are the best predictors of employee attrition.

1.2.2 Data

Several companies have provided HR data on employee attrition (24,702 employee records), we will explore the structure of this data in section 2.1.

1.2.3 Objectives

- Ensure the data are valid. Before building models and making predictions and creating knowledge, i'll do my best to verify the integrity of the data and data collection.
- Provide a model that predicts employee attrition.
- Offer suggestions on actionable ways to improve employee attrition rates, validate these suggestions if possible.

1.2.4 1.2 Metrics for Success

The problem statement implies that losing employees is very costly. Depending on what our suggestion is for retaining employees, it's likely less costly to the company to offer incentives to at risk employees than it is to lose them. For this reason, I believe it is important to not miss true positives and that I can tolerate some level of false positives. For this reason, i'll be optimizing the recall of the model. We'll also dive a little deeper into more informative metrics provided that there is time.

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

        %matplotlib inline
```

1.3 2. Loading and Cleaning

In this section the data is loaded into memory and we aim to clean the data for our machine learning model. Additionally, we seek to decide if the data is reasonably representative of the ground truth and will actually build a useful model.

1.3.1 2.1 Loading with Pandas

```
In [2]: data = pd.read_csv('./employee_retention_data.csv')
```

```
In [3]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24702 entries, 0 to 24701
Data columns (total 7 columns):
employee_id    24702 non-null float64
company_id     24702 non-null int64
dept           24702 non-null object
seniority      24702 non-null int64
salary         24702 non-null float64
join_date      24702 non-null object
quit_date      13510 non-null object
dtypes: float64(2), int64(2), object(3)
memory usage: 1.3+ MB
```

We can see the basic structure of the data from a basic computer perspective here, the dates are stored as objects (strings) and will need to be converted to numerical representations. I chose to convert each date into a year, month, day format using three columns.

1.3.2 2.2 Outliers and NaN Values

In this section, I want to inspect the dataset and check for values which are obviously outliers and then make a decision on what to do with them. Additionally, i'll handle NaN values in this section.

In [4]:
```
data.describe()
```

Out[4]:
```
         employee_id    company_id     seniority         salary
count   24702.000000  24702.000000  24702.000000   24702.000000
mean   501604.403530      3.426969     14.127803  138183.345478
std    288909.026101      2.700011      8.089520   76058.184573
min        36.000000      1.000000      1.000000   17000.000000
25%    250133.750000      1.000000      7.000000   79000.000000
50%    500793.000000      2.000000     14.000000  123000.000000
75%    753137.250000      5.000000     21.000000  187000.000000
max    999969.000000     12.000000     99.000000  408000.000000
```

Here we can see that something doesn't make sense. The seniority variable which describes the number of years of experience of an employee is actually 99 for some employees. Let's see how many.

In [5]:
```
data[data['seniority'] > 50]
```

Out[5]:
```
       employee_id  company_id       dept  seniority    salary   join_date  \
24700      97289.0          10   engineer         98  266000.0  2011-12-13
24701     604052.0           1  marketing         99  185000.0  2011-07-26

        quit_date
24700  2015-01-09
24701  2013-12-06
```

Since these represent such a small fraction of the entire sample, I am going to remove them from this analysis. The employee from company 10 may be important because of the relatively small size of company 10. If that turns out to be the case, we could try guessing the seniority of these employees by looking at their department and salary. For now let's continue without these two employees. Let's look for missing values now.

```
In [6]: data.isna().sum()
```

```
Out[6]: employee_id        0
        company_id         0
        dept               0
        seniority          0
        salary             0
        join_date          0
        quit_date      11192
        dtype: int64
```

Here it's clear that the dataset is somewhat clean already. The missing values in `quit_date` correspond to employees which have not quit and are still employed as of 12/31/2015. In the cleaning phase, we will use that to create a binary target variable called `churn_status` to represent the employees status. Let's define our cleaning function and proceed to clean the dataset.

```
In [7]: def process_date(date_string):
            ''' Tokenize dates into arrays of integers.

                Arguments:
                date_string - String containing a date in the
                format year/month/day.

                Ouptut:
                If the input is null, the
                function returns an array of zeros.  Otherwise
                the array is returned with [year, month, day].
            '''
            if type(date_string) == str:
                tokens = date_string.split('-')
                tokens = [int(t) for t in tokens]

                if len(tokens) != 3:
                    return np.array([0, 0, 0])
                else:
                    return np.array(tokens)

            return np.array(([0, 0, 0]))

        def get_year(date_string):
            return process_date(date_string)[0]

        def get_month(date_string):
```

```
        return process_date(date_string)[1]

    def get_day(date_string):
        return process_date(date_string)[2]

    def clean_dataset(data):
        ''' Clean employee dataset of NaN values and add target.

            This function uses the insights gained above to clean
            the employee/company dataset.

            Arguments:
            data - The input dataframe which contains the employee data.

            Output:
            data - The modified dataframe is returned.
        '''

        data = data[data['seniority'] < 90]

        # Add target as float 0/1.
        data['churn_status'] = data['quit_date'].notna().round()

        # Decompose join date string into 3 integers.
        data['join_year'] = data['join_date'].apply(get_year)
        data['join_month'] = data['join_date'].apply(get_month)
        data['join_day'] = data['join_date'].apply(get_day)

        # Decompose quit date string into 3 integers.
        data['quit_year'] = data['quit_date'].apply(get_year)
        data['quit_month'] = data['quit_date'].apply(get_month)
        data['quit_day'] = data['quit_date'].apply(get_day)

        # Drop useless columns
        data.drop(columns = ['join_date', 'quit_date'], inplace = True)

        return data
```

In [8]:
```
data = clean_dataset(data)
```

```
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/ipykernel_launcher.py:49:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/ipykernel_launcher.py:52:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/ipykernel_launcher.py:53:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/ipykernel_launcher.py:54:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/ipykernel_launcher.py:57:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/ipykernel_launcher.py:58:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/ipykernel_launcher.py:59:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/pandas/core/frame.py:3697
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  errors=errors)
```

In [9]:
```
data.head()
```

Out[9]:

	employee_id	company_id	dept	seniority	salary
0	13021.0	7	customer_service	28	89000.0
1	825355.0	7	marketing	20	183000.0
2	927315.0	4	marketing	14	101000.0
3	662910.0	7	customer_service	20	115000.0
4	256971.0	2	data_science	23	276000.0

	churn_status	join_year	join_month	join_day	quit_year	quit_month
0	1.0	2014	3	24	2015	10
1	1.0	2013	4	29	2014	4
2	0.0	2014	10	13	0	0
3	1.0	2012	5	14	2013	6

```
4              1.0        2011           10         17        2014              8

   quit_day
0        30
1         4
2         0
3         7
4        22
```

Before moving to feature engineering, let's quickly see if the classes are balanced.

```
In [10]: data['churn_status'].value_counts()
```

```
Out[10]: 1.0    13508
         0.0    11192
         Name: churn_status, dtype: int64
```

Not too bad.

1.4 3. Feature Engineering

Let's use all of our creativity to add some features from the domain that might be useful.

```
In [11]: def feature_engineer(data):

             # Total number of days worked at the company
             # for those who quit (I will fix the others below).
             data['days_worked'] = 365 * (data['quit_year'] - data['join_year']) + \
                 365.25 / 12.0 * (data['quit_month'] - data['join_month']) + \
                 (data['quit_day'] - data['join_day'])

             # Add some days worked for peolpe who didn't quit.
             indexer = data['days_worked'] < 0

             data['days_worked'].loc[indexer] = 365 * (2015 - data['join_year'].loc[indexer]) +
                 365.25 / 12.0 * (12 - data['join_month'].loc[indexer]) + \
                 (31 - data['join_day'].loc[indexer])

             # By plotting salary and seniority one can see that there are clearly
             # three steps for every company.
             data['career_level'] = np.digitize(data['seniority'], np.array([0, 5, 15, 99]))

             # Encode the departments into columns
             department = data['dept']
             data = pd.get_dummies(data, columns = ['dept'], prefix = 'dept')
             data['dept'] = department

             # Compare the individual employee to the average in their division
             # by computing z-score
```

```
data['average_salary_in_division'] = data.groupby(['company_id', 'dept', 'career_l
data['std_salary_in_division'] = data.groupby(['company_id', 'dept', 'career_leve
data['internal_salary_zscore'] = (data['salary'] - data['average_salary_in_divisio
data['internal_salary_zscore'].fillna(0, inplace = True)
data.drop(columns = ['average_salary_in_division', 'std_salary_in_division'], inpl

# Compare the individual employee to the average in their division
# by computing z-score
data['average_salary_in_profession'] = data.groupby(['dept', 'career_level']).tran
data['std_salary_in_profession'] = data.groupby(['dept', 'career_level']).transfo
data['profession_salary_zscore'] = (data['salary'] - data['average_salary_in_profe
data['profession_salary_zscore'].fillna(0, inplace = True)
data.drop(columns = ['average_salary_in_profession', 'std_salary_in_profession'],

# Figure out how long the person worked before coming to this company
# Need to insert zeros if people haven't worked before coming here.
# This variable has problems check it out.
data['previous_days_worked'] = 365.25 * data['seniority'] - data['days_worked']
data['previous_days_worked'] = data['previous_days_worked'].apply(lambda x: 0.0 i

# How much of their careers have they worked here?
data['career_fraction_in_company'] = data['days_worked'] / (data['days_worked'] +

# How much of the department years of experience
# does each person have?
data['dept_experience_fraction'] = data['seniority'] / data.groupby(['company_id'
data['dept_experience_fraction'].fillna(0, inplace = True)

# How does their experience compare to the top dog?
data['dept_experience_disparity'] = data.groupby(['company_id', 'dept']).transform
data['dept_experience_disparity'].fillna(data['seniority'].max(), inplace = True)

# Compare the individual employee to the average in their division
# by computing z-score of seniority
data['average_seniority_in_profession'] = data.groupby(['dept', 'career_level']).t
data['std_seniority_in_profession'] = data.groupby(['dept', 'career_level']).trans
data['profession_seniority_zscore'] = (data['seniority'] - data['average_seniority
data['profession_seniority_zscore'].fillna(0, inplace = True)
data.drop(columns = ['average_seniority_in_profession', 'std_seniority_in_professi
data['profession_seniority_zscore'].fillna(0, inplace = True)

# Compare the individual employee to the average in their division
# by computing z-score of seniority
#data['average_seniority_in_division'] = data.groupby(['company_id' ,'dept', 'car
#data['std_seniority_in_division'] = data.groupby(['company_id', 'dept', 'career_
#data['division_seniority_zscore'] = (data['salary'] - data['average_seniority_in_
#data['division_seniority_zscore'].fillna(0, inplace = True)
#data.drop(columns = ['average_seniority_in_division', 'std_seniority_in_division
```

```
    #data['division_seniority_zscore'].fillna(0, inplace = True)

    # Has the employee taken a step up during their
    # time in the company.  This happens from 4-5 and
    # from 14-15 according to our visual analysis below.
    current_experience = data['seniority'].values
    previous_experience = data['previous_days_worked'].values / 365.25
    data['step_up_1'] = (current_experience >= 5) & (previous_experience < 5)
    data['step_up_2'] = (current_experience >= 15) & (previous_experience < 14)
    data['step_up_1'] = data['step_up_1'].astype(int)
    data['step_up_2'] = data['step_up_2'].astype(int)

    return data
```

In [12]:
```
data = feature_engineer(data)
data.head(12)
```

```
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/pandas/core/indexing.py:18
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  self._setitem_with_indexer(indexer, value)
```

Out[12]:

	employee_id	company_id	seniority	salary	churn_status	join_year	\
0	13021.0	7	28	89000.0	1.0	2014	
1	825355.0	7	20	183000.0	1.0	2013	
2	927315.0	4	14	101000.0	0.0	2014	
3	662910.0	7	20	115000.0	1.0	2012	
4	256971.0	2	23	276000.0	1.0	2011	
5	509529.0	4	14	165000.0	1.0	2012	
6	88600.0	4	21	107000.0	0.0	2013	
7	716309.0	2	4	30000.0	0.0	2014	
8	172999.0	9	7	160000.0	1.0	2012	
9	504159.0	1	7	104000.0	0.0	2012	
10	892155.0	6	13	72000.0	1.0	2012	
11	904158.0	2	17	230000.0	0.0	2015	

	join_month	join_day	quit_year	quit_month	...	dept	\
0	3	24	2015	10	...	customer_service	
1	4	29	2014	4	...	marketing	
2	10	13	0	0	...	marketing	
3	5	14	2013	6	...	customer_service	
4	10	17	2014	8	...	data_science	
5	1	30	2013	8	...	data_science	
6	10	21	0	0	...	customer_service	
7	3	5	0	0	...	customer_service	
8	12	10	2015	10	...	engineer	

```
9             6         12          0           0     ...                sales
10           11         12       2015           2     ...     customer_service
11            5         11          0           0     ...            marketing

    internal_salary_zscore  profession_salary_zscore  previous_days_worked  \
0                -0.761004                 -1.200686             9642.9375
1                 1.330631                  0.105023             6965.0000
2                -0.347946                 -0.945573             4669.6250
3                 1.976543                  0.394777             6916.5625
4                -0.837331                  0.101311             7361.6250
5                 0.408610                 -0.391116             4535.4375
6                 1.183250                 -0.096135             6869.3750
7                 0.081826                  0.704173              796.0625
8                 0.311569                 -0.620292             1509.6250
9                -1.966423                 -0.768153             1260.1250
10                1.547717                  0.168272             3912.1875
11                1.777681                  1.961723             5976.1875

    career_fraction_in_company  dept_experience_fraction  \
0                     0.057110                  0.004165
1                     0.046543                  0.010325
2                     0.086805                  0.003837
3                     0.053174                  0.002975
4                     0.123694                  0.002769
5                     0.113046                  0.003540
6                     0.104413                  0.001946
7                     0.455125                  0.000180
8                     0.409553                  0.002677
9                     0.507138                  0.000456
10                    0.176078                  0.001806
11                    0.037535                  0.002253

    dept_experience_disparity  profession_seniority_zscore  step_up_1  \
0                           1                     1.550613          0
1                           9                    -0.319690          0
2                          15                     1.565323          0
3                           9                    -0.329178          0
4                           6                     0.389992          0
5                          15                     1.630788          0
6                           8                    -0.094204          0
7                          25                     1.367156          0
8                          22                    -0.868630          1
9                          22                    -0.829365          1
10                         16                     1.206759          0
11                         12                    -1.009213          0

   step_up_2
0          0
```

```
1          0
2          0
3          0
4          0
5          0
6          0
7          0
8          0
9          0
10         0
11         0

[12 rows x 29 columns]
```

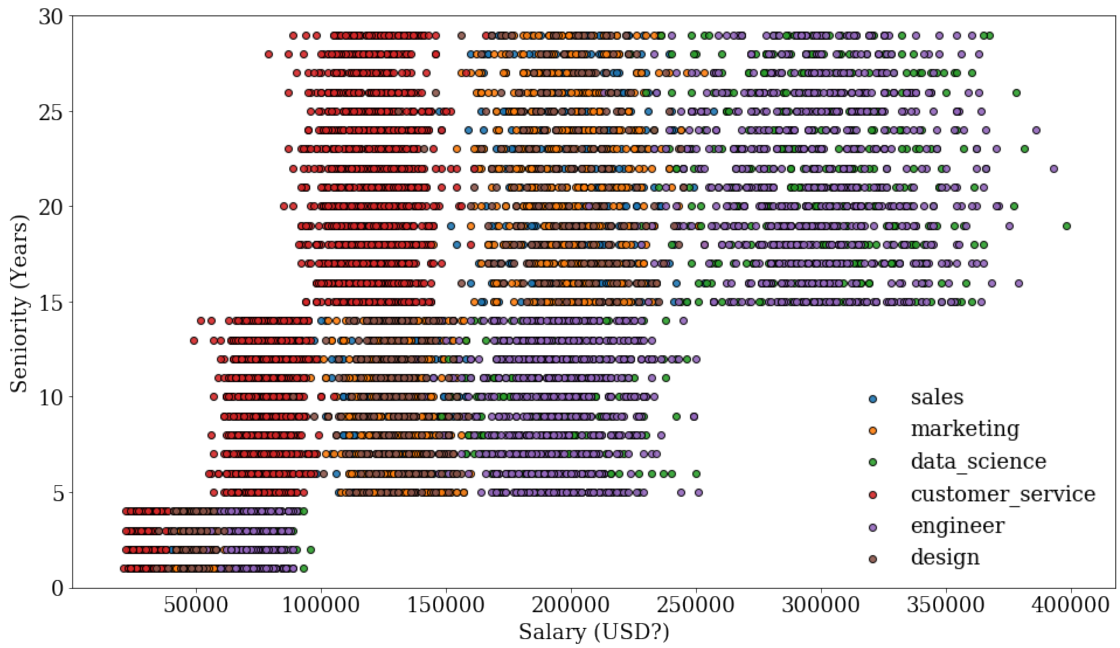
1.5 4. Visual Exploration

I am going to set some style preferences before the plotstorm begins.

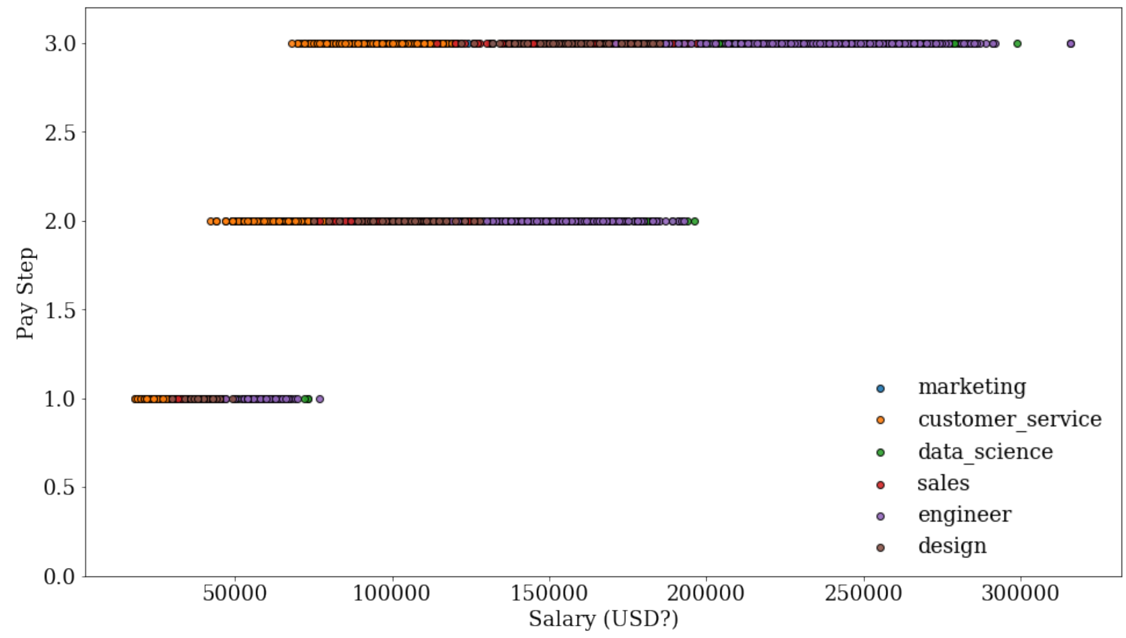
```
In [13]: plt.rc('font', family = 'serif')
         plt.rc('font', size = 18)
```

```
In [14]: plt.figure(figsize = (16, 9))

         company_data = data.query('company_id == 1')

         for department in company_data['dept'].unique():
             plt.scatter(
                 company_data.query('dept == "{}"'.format(department))['salary'],
                 company_data.query('dept == "{}"'.format(department))['seniority'],
                 edgecolor = 'k',
                 alpha = 0.9,
                 label = department
             );

         plt.ylim([0, 30])
         plt.legend(frameon = False)
         plt.xlabel('Salary (USD?)')
         plt.ylabel('Seniority (Years)')
```

```
Out[14]: Text(0, 0.5, 'Seniority (Years)')
```

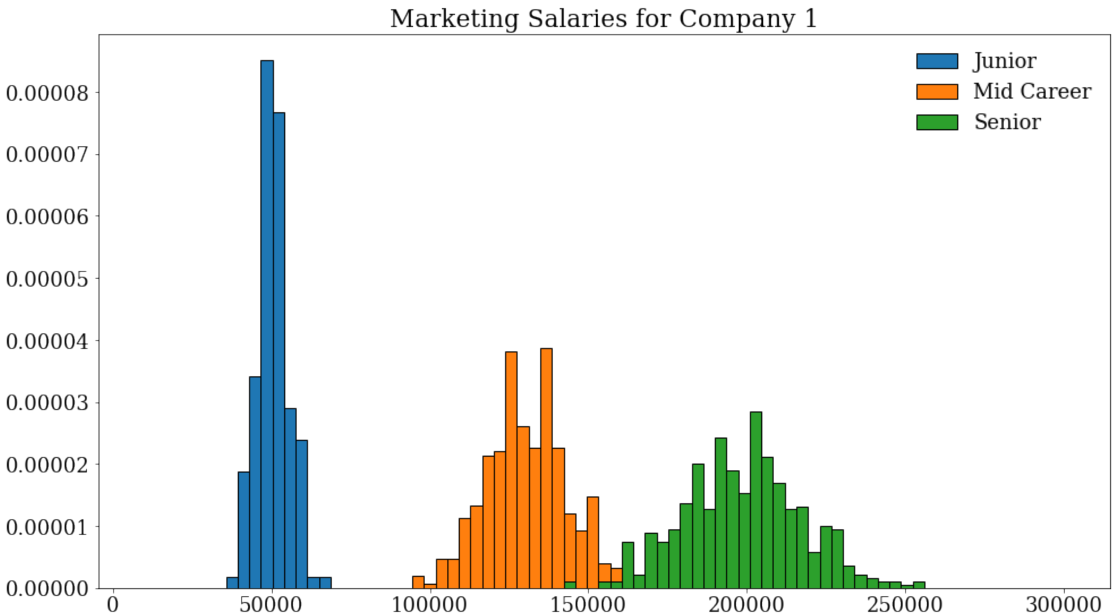



```
In [15]: plt.figure(figsize = (16, 9))

         company_data = data.query('company_id == 3')

         for department in company_data['dept'].unique():
             plt.scatter(
                 company_data.query('dept == "{}"'.format(department))['salary'],
                 company_data.query('dept == "{}"'.format(department))['career_level'],
                 edgecolor = 'k',
                 alpha = 0.9,
                 label = department
             );

         plt.ylim([0, 3.2])
         plt.legend(frameon = False)
         plt.xlabel('Salary (USD?)')
         plt.ylabel('Pay Step')

Out[15]: Text(0, 0.5, 'Pay Step')
```



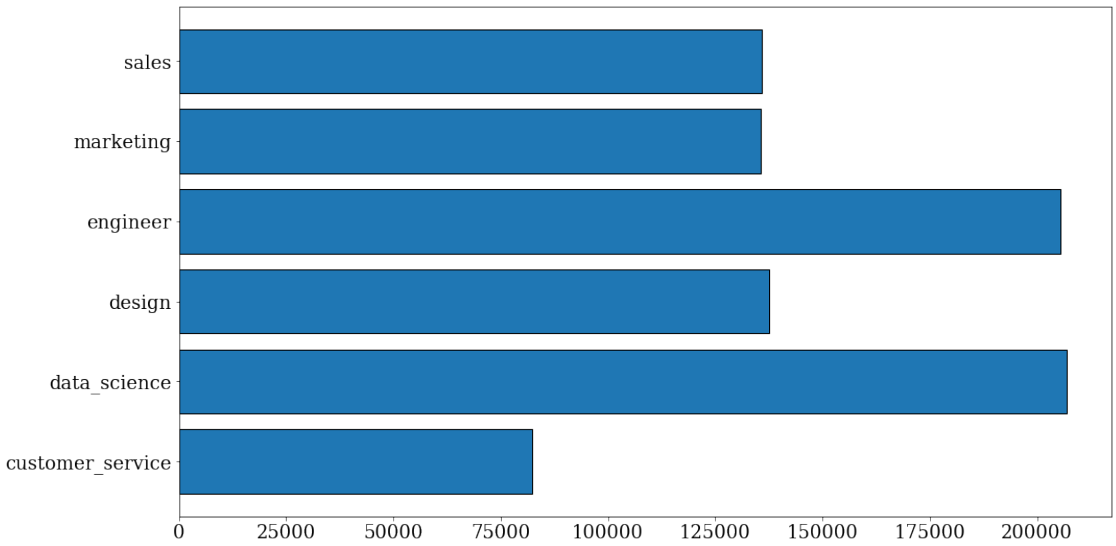

```
In [16]: plt.figure(figsize = (16, 9))
         plt.hist(
             data.query('company_id == 1 and dept_marketing == 1 and career_level == 1')['sala
             bins = np.linspace(10000, 300000, 80),
             edgecolor = 'k',
             density = True,
             label = 'Junior'
         );
         plt.hist(
             data.query('company_id == 1 and dept_marketing == 1 and career_level == 2')['sala
             bins = np.linspace(10000, 300000, 80),
             edgecolor = 'k',
             density = True,
             label = 'Mid Career'
         );
         plt.hist(
             data.query('company_id == 1 and dept_marketing == 1 and career_level == 3')['sala
             bins = np.linspace(10000, 300000, 80),
             edgecolor = 'k',
             density = True,
             label = 'Senior'
         );
         plt.legend(frameon = False)
         plt.title('Marketing Salaries for Company 1')
```

```
Out[16]: Text(0.5, 1.0, 'Marketing Salaries for Company 1')
```



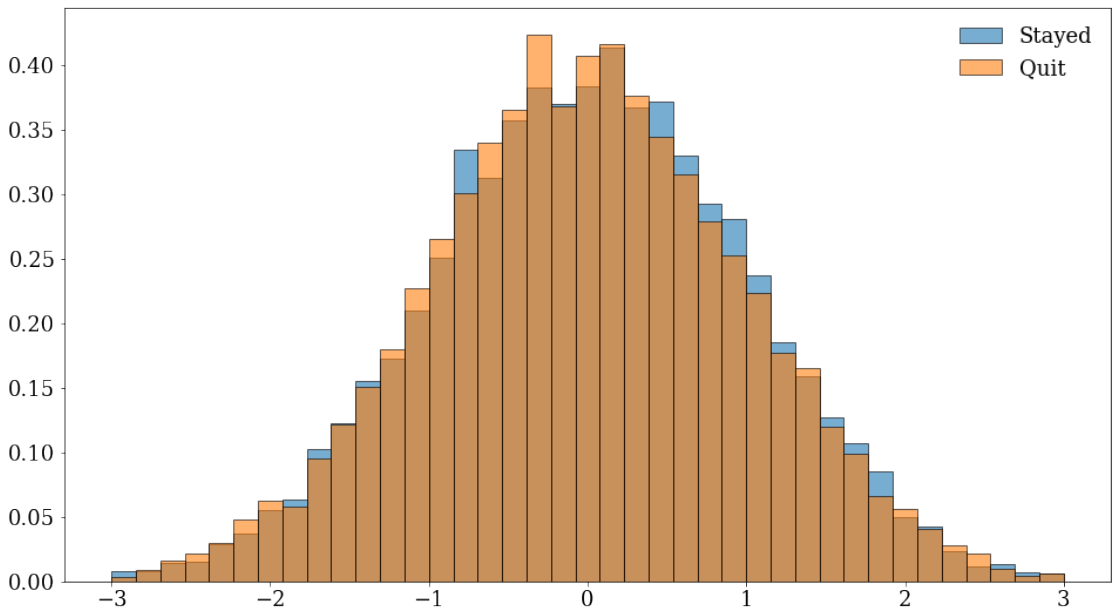
This figure reveals something interesting. Employees within company one with the same level of experience are not compensated the same. Perhaps this information can be used as a predictor. One could imagine using a z-score for each employee, reflecting how far away they are from the mean salary in their bracket.

```
In [17]: global_salary_by_department = data.groupby('dept').aggregate(
             {
                 'salary' : np.mean
             }
         )

         plt.figure(figsize = (16, 9))
         plt.barh(
             global_salary_by_department.index,
             global_salary_by_department.values.reshape(len(global_salary_by_department)),
             edgecolor = 'k'
         )

Out[17]: <BarContainer object of 6 artists>
```



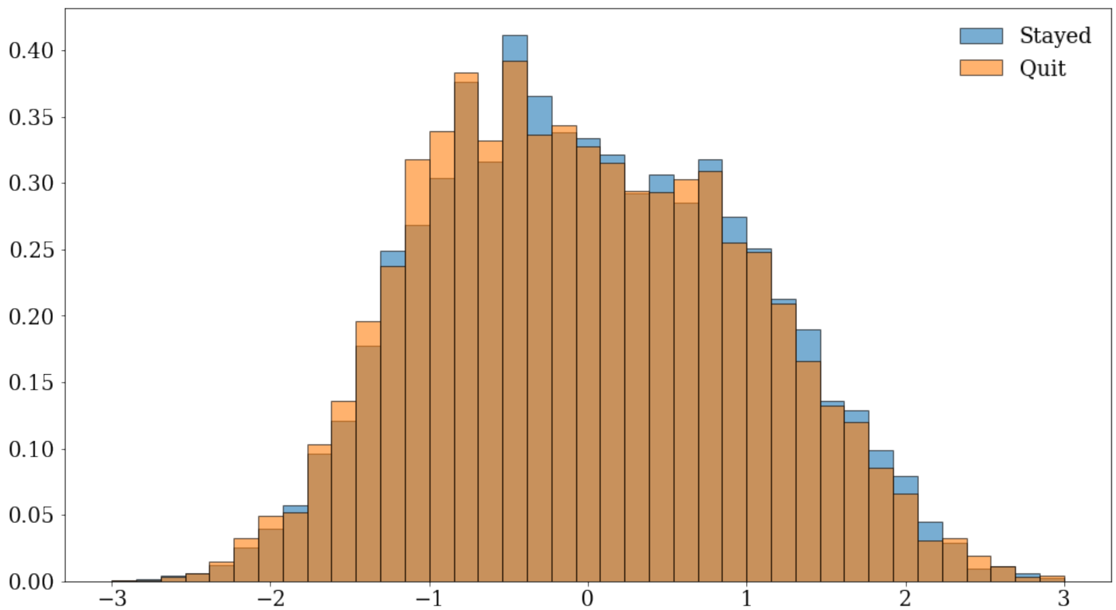
In [18]:
```
plt.figure(figsize = (16, 9))
plt.hist(
    data.query('churn_status < 0.5')['internal_salary_zscore'],
    bins = np.linspace(-3, 3, 40),
    edgecolor = 'k',
    alpha = 0.6,
    label = 'Stayed',
    density = True
);
plt.hist(
    data.query('churn_status > 0.5')['internal_salary_zscore'],
    bins = np.linspace(-3, 3, 40),
    edgecolor = 'k',
    alpha = 0.6,
    label = 'Quit',
    density = True
);
plt.legend(frameon = False)
```

Out[18]:
```
<matplotlib.legend.Legend at 0x116de7470>
```



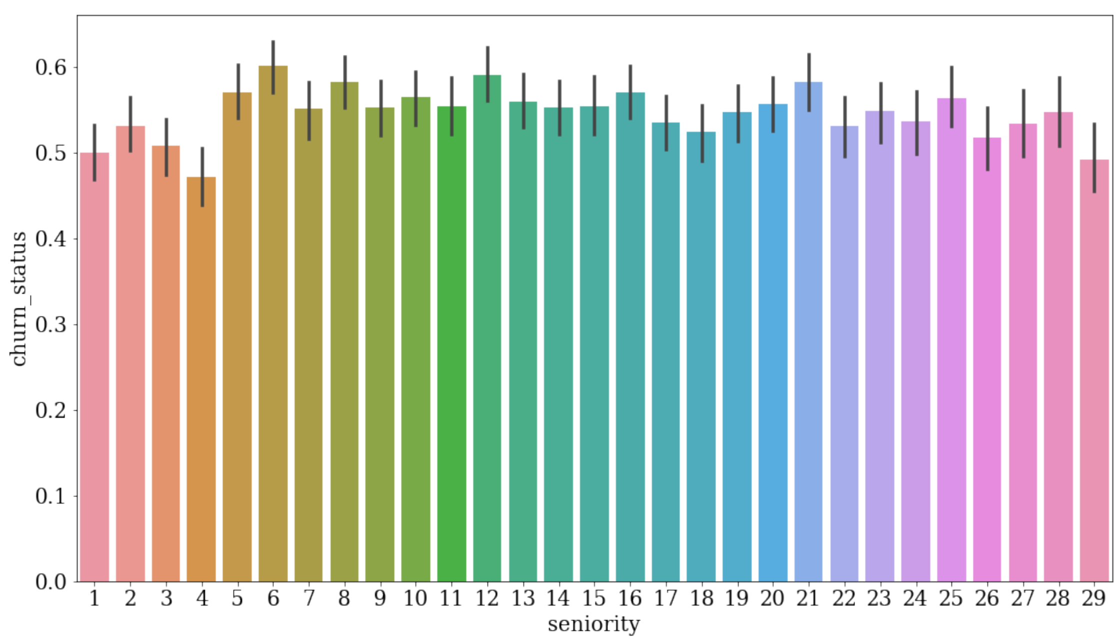

```
In [19]: plt.figure(figsize = (16, 9))
         plt.hist(
             data.query('churn_status < 0.5')['profession_salary_zscore'],
             bins = np.linspace(-3, 3, 40),
             edgecolor = 'k',
             alpha = 0.6,
             label = 'Stayed',
             density = True
         );
         plt.hist(
             data.query('churn_status > 0.5')['profession_salary_zscore'],
             bins = np.linspace(-3, 3, 40),
             edgecolor = 'k',
             alpha = 0.6,
             label = 'Quit',
             density = True
         );
         plt.legend(frameon = False)
```

```
Out[19]: <matplotlib.legend.Legend at 0x116ef0940>
```



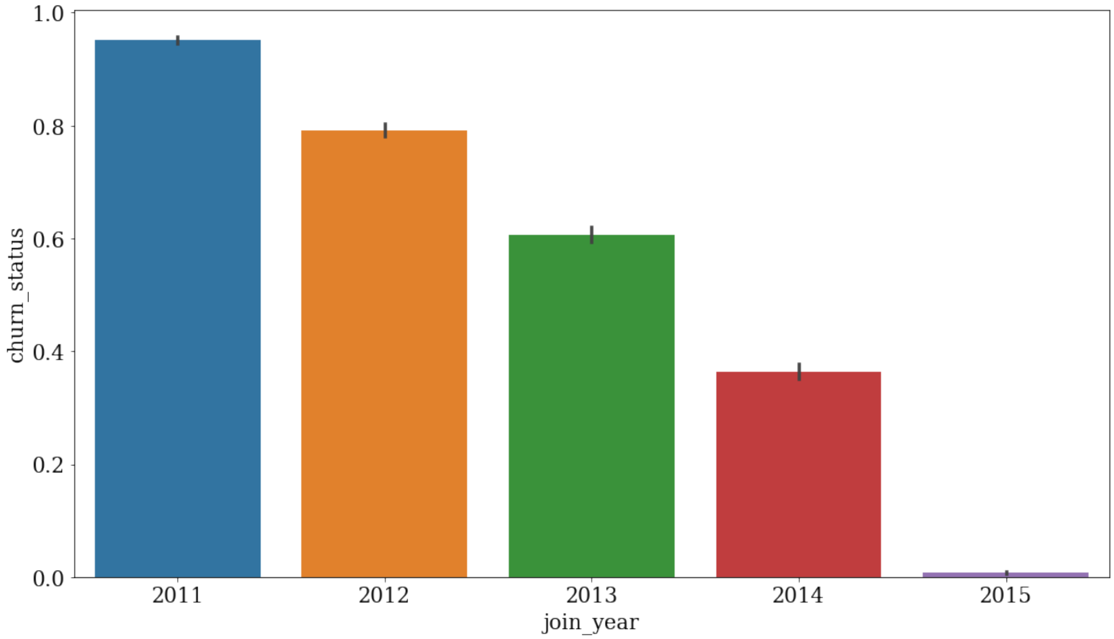

```
In [20]: plt.figure(figsize = (16, 9))
         sns.barplot(data['seniority'], data['churn_status'])
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x116e17940>
```



```
In [21]: plt.figure(figsize = (16, 9))
         sns.barplot(data['join_year'], data['churn_status'])
```

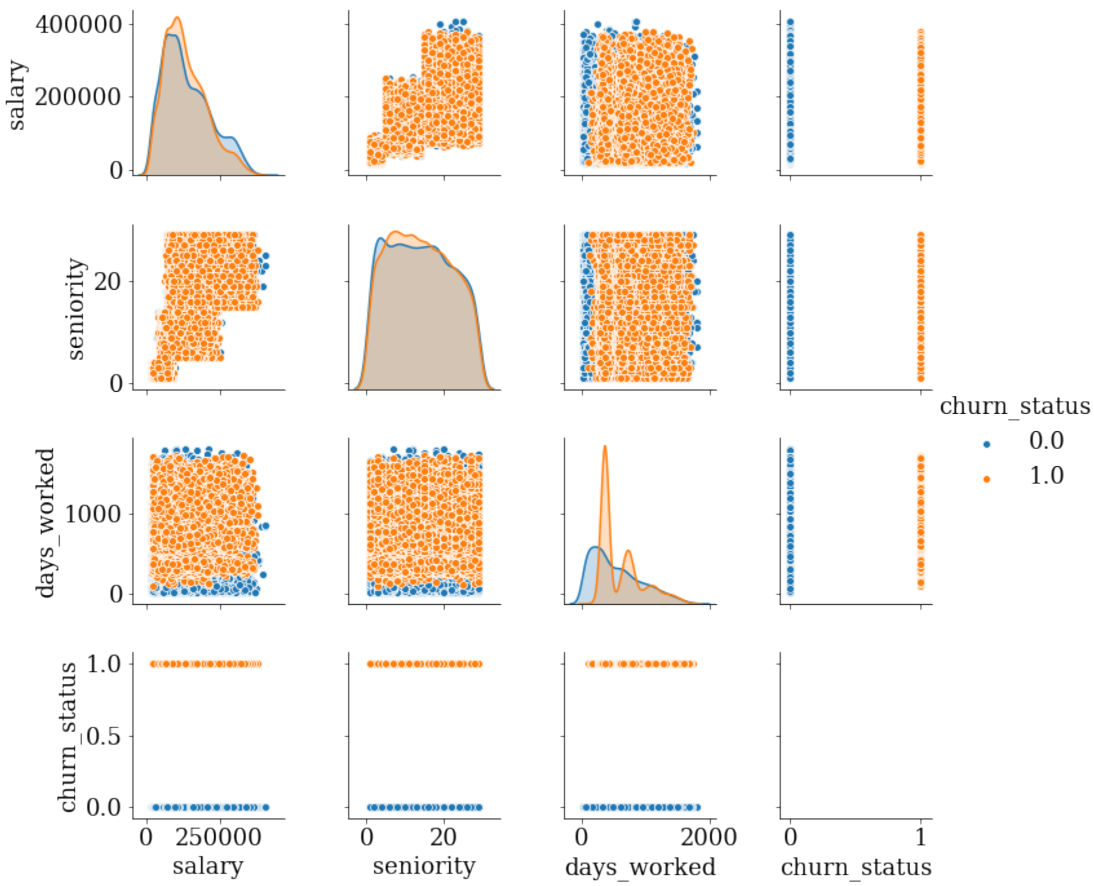
```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x115f35128>
```




```
In [22]: sns.pairplot(data[['salary', 'seniority', 'days_worked', 'churn_status']], hue = 'chu
```

```
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/statsmodels/nonparametric/
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/Users/davidriser/anaconda3/envs/insight/lib/python3.7/site-packages/statsmodels/nonparametric/
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

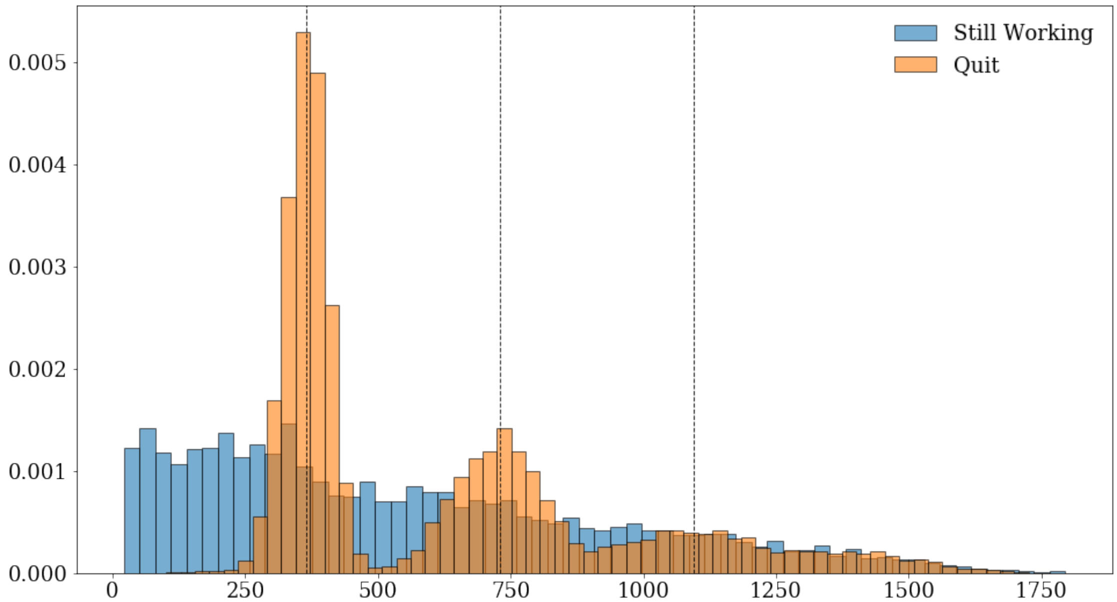
```
Out[22]: <seaborn.axisgrid.PairGrid at 0x1a19561b38>
```




```
In [23]: plt.figure(figsize = (16, 9))
         plt.hist(
             data[data['churn_status'] == 0]['days_worked'],
             bins = 60,
             edgecolor = 'k',
             alpha = 0.6,
             density = True,
             label = 'Still Working'
         );
         plt.hist(
             data[data['churn_status'] == 1]['days_worked'],
             bins = 60,
             edgecolor = 'k',
             alpha = 0.6,
             density = True,
             label = 'Quit'
         );
         plt.legend(frameon = False)
         plt.axvline(365.0, linestyle = '--', linewidth = 1, color = 'k', alpha = 0.8)
```

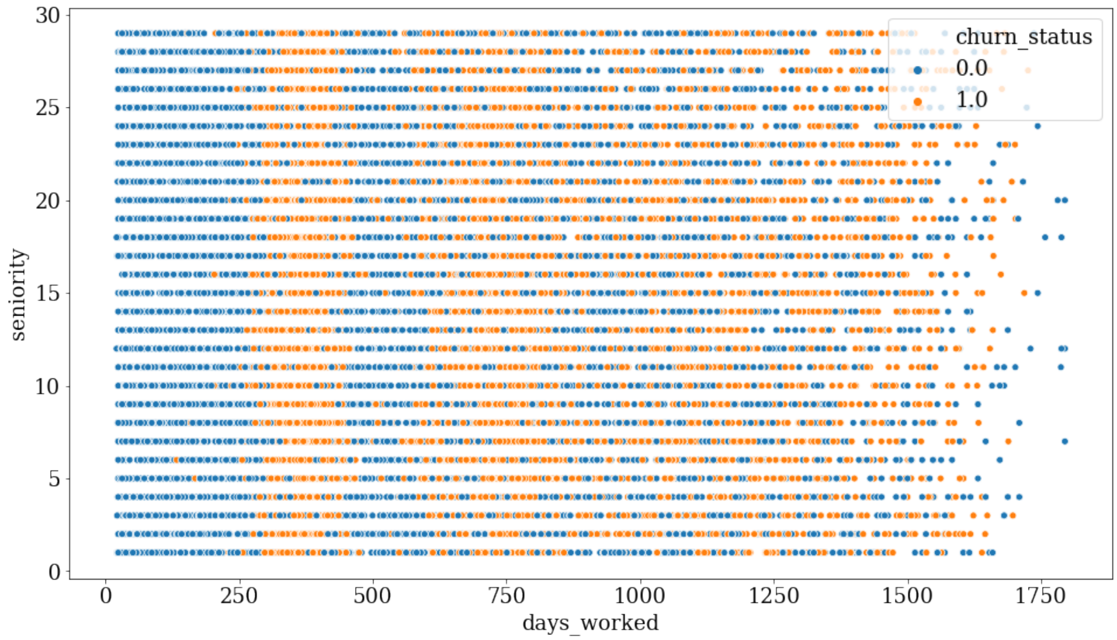
```
    plt.axvline(2 * 365.0, linestyle = '--', linewidth = 1, color = 'k', alpha = 0.8)
    plt.axvline(3 * 365.0, linestyle = '--', linewidth = 1, color = 'k', alpha = 0.8)
```

Out[23]: <matplotlib.lines.Line2D at 0x1a1a7e05c0>



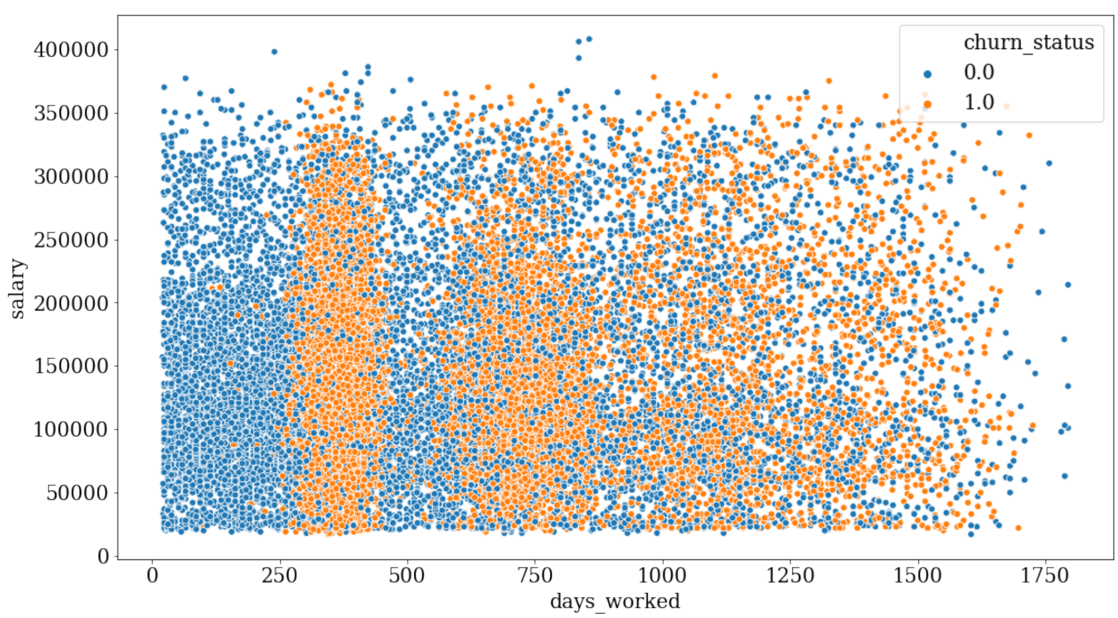
In [24]:
```
plt.figure(figsize = (16, 9))
sns.scatterplot(
    x = 'days_worked',
    y = 'seniority',
    data = data,
    hue = 'churn_status'
)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a7b6208>

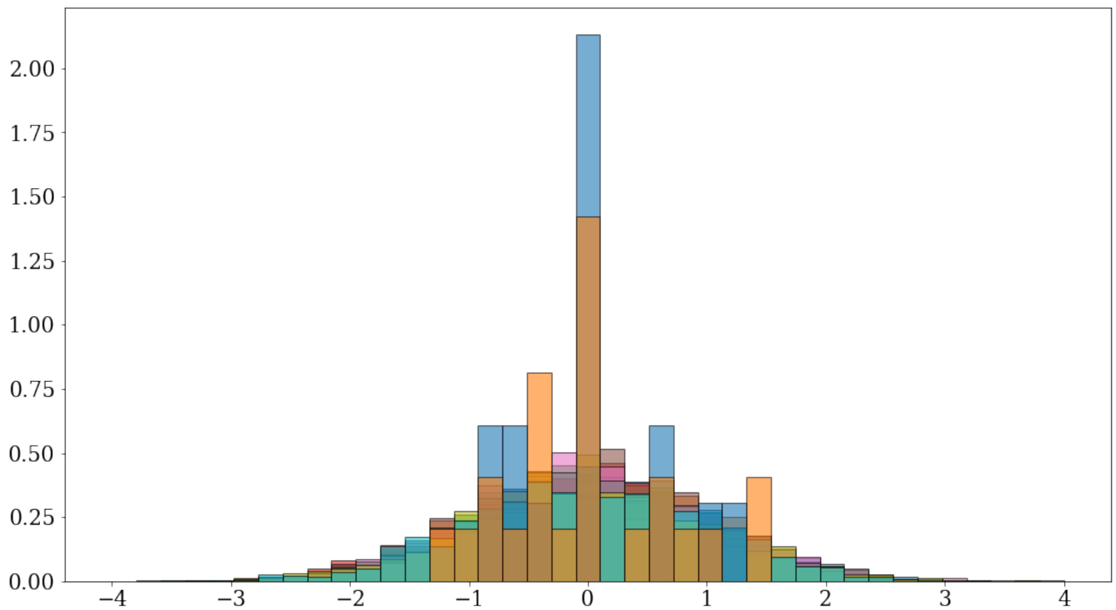


In [25]:
```
plt.figure(figsize = (16, 9))
sns.scatterplot(
    x = 'days_worked',
    y = 'salary',
    data = data,
    hue = 'churn_status'
)
```

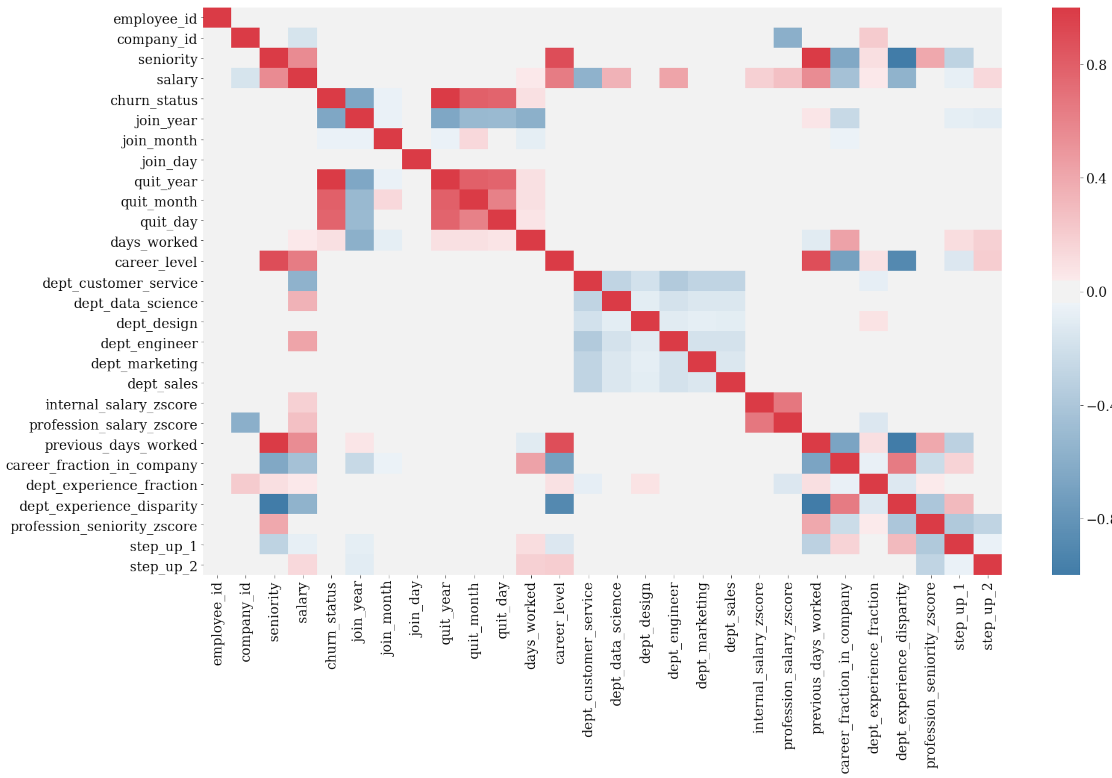
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1af19828>



```
In [26]: plt.figure(figsize = (16, 9))

         for company_id in data['company_id'].unique():
             company_data = data.query('company_id == {}'.format(company_id))
             plt.hist(
                 company_data['internal_salary_zscore'],
                 bins = np.linspace(-4, 4, 40),
                 edgecolor = 'k',
                 alpha = 0.6,
                 density = True
             );
```




```
In [27]: plt.figure(figsize = (24, 14))
         sns.heatmap(data.corr(), cmap = sns.diverging_palette(240, 10, n = 256))

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1af40a20>
```



1.6 5. Modeling

In this section predictions are made for employee attrition based on the cleaned data that has features added to it. First, I am going to define which features to take into account. I'm leaving out things like the start and end date of employment, as that information has been transfered into the `days_worked` variable and I don't think it would be meaningful to include start days, months, and years.

```
In [28]: from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
         from sklearn.model_selection import train_test_split
         from sklearn.naive_bayes import GaussianNB
         from sklearn.preprocessing import StandardScaler
```

```
In [29]: target = 'churn_status'
         features = ['career_level', 'company_id', 'days_worked',
                     'dept_customer_service', 'dept_data_science', 'dept_design',
                      'dept_engineer', 'dept_marketing', 'dept_sales',
                      'salary', 'seniority', 'internal_salary_zscore',
                     'profession_salary_zscore', 'dept_experience_fraction', 'previous_days_worl
                     'career_fraction_in_company', 'profession_seniority_zscore',
                     'dept_experience_disparity', 'step_up_1', 'step_up_2']
```

```
In [30]: for feat in features:
             if feat not in data.columns:
                 raise ValueError('There is no feature {}.'.format(feat))
```

Now we're going to do something important and define the metrics that we want to track. Based on these we can take a decision on what action to recommend.

```
In [31]: from sklearn.metrics import confusion_matrix
```

```
In [32]: def business_metric(y_pred, y_true,
                             cost_of_loss = 100000,
                             cost_of_intervention = 10000,
                             probability_of_success = 0.2):
             ''' Important business metric. This would be fleshed out with someone who knows b

             confusion_mat = confusion_matrix(y_true, y_pred)

             true_negatives = confusion_mat[0,0]
             false_negatives = confusion_mat[1,0]
             false_positives = confusion_mat[0,1]
             true_positives = confusion_mat[1,1]

             # We predicted that the person will quit
             # but actually they will not. We incur the
             # cost of the intervention.
             cost = false_positives * cost_of_intervention

             # We predict that the employee will not quit
             # but they are going to quit.  We incur the full
             # cost of loss.
             cost += false_negatives * cost_of_loss

             # We predict they will quit, and they actually will
             # but by doing our intervention we save N * probability_of_success
             # of them and only incur the intervention cost for them.
             cost += true_positives * (cost_of_intervention + (1 - probability_of_success) * c

             # True negatives are free so do nothing for them.

             return float(cost / len(y_true))
```

```
In [33]: metrics = {
             'accuracy' : accuracy_score,
             'precision' : precision_score,
             'recall' : recall_score,
             'f1_score' : f1_score,
             'business_metric' : business_metric
         }
```

We know from visual exploration of the data that the number of days worked is important and that employee attrition spikes around the one and two year employment times. Let's build a simple model that just guesses employees are going to quit if they're inside of a time window `quit_window` from one or two years of employment.

```
In [34]: class NaiveBaselineModel(object):
             '''
             NaiveBaselineModel - Guess that any employee close
             to 1 or 2 years of employment is likely to quit.

             '''
             def __init__(self, quit_window = 60):
                 self.quit_window = quit_window

             def fit(x_train, y_train):
                 pass

             def predict(self, days_worked):
                 predictions = np.zeros(len(days_worked))
                 predictions[np.where((days_worked > (365 - self.quit_window)) & (days_worked 
                 predictions[np.where((days_worked > (2 * 365 - self.quit_window)) & (days_wor
                 return predictions

             def predict_proba(self, days_worked):
                 return self.predict(days_worked)
```

Let's also write a function useful for training our models.

```
In [35]: def train_evaluate(model, x_train, x_test, y_train, y_test, metrics = {}, return_mode
             model.fit(x_train, y_train)

             y_pred_test = model.predict(x_test)
             y_pred_train = model.predict(x_train)

             results = {}
             for metric_name in metrics.keys():
                 results[metric_name + '_test'] = metrics[metric_name](y_pred_test, y_test)
                 results[metric_name + '_train'] = metrics[metric_name](y_pred_train, y_train)

             if return_model:
                 return model, results
             else:
                 return results
```

```
In [36]: baseline_model = NaiveBaselineModel(60)
         y_pred_baseline = baseline_model.predict(data['days_worked'].values)

         print(accuracy_score(data['churn_status'].values, y_pred_baseline))
```

```
print(precision_score(data['churn_status'].values, y_pred_baseline))
print(recall_score(data['churn_status'].values, y_pred_baseline))
```

```
0.7002834008097166
0.7865928081870247
0.6202250518211431
```

Standardization of the features.

```
In [37]: transformer = StandardScaler()
         x = transformer.fit_transform(data[features].values)
         y = data[target].values

         x_train, x_test, y_train, y_test = train_test_split(x, y)
```

```
In [38]: default_models = {
             'Logistic Regression' : LogisticRegression(),
             'Naive Bayes' : GaussianNB(),
             'Random Forest' : RandomForestClassifier(),
             'Gradient Boosted Trees' : GradientBoostingClassifier()
         }
```

```
In [39]: default_metrics = {}
         for model in default_models.keys():
             default_metrics[model] = train_evaluate(
                 model = default_models[model],
                 x_train = x_train,
                 x_test = x_test,
                 y_train = y_train,
                 y_test = y_test,
                 metrics = metrics
             )
```

```
In [40]: for key, value in default_metrics.items():
             print(key)

             for k, v in value.items():
                 print('\t', k, v)
```

```
Logistic Regression
         accuracy_test 0.6050202429149798
         accuracy_train 0.6087989203778678
         precision_test 0.8733912002394493
         precision_train 0.8628897413199568
         recall_test 0.5914065666801783
         recall_train 0.5998222343771366
         f1_score_test 0.7052567975830815
         f1_score_train 0.7076997539628119
```

```
        business_metric_test 52644.534412955465
        business_metric_train 53306.34278002699
Naive Bayes
        accuracy_test 0.5815384615384616
        accuracy_train 0.5833198380566802
        precision_test 0.7390002993115834
        precision_train 0.7347300088521688
        recall_test 0.5905285816790241
        recall_train 0.5979827089337176
        f1_score_test 0.6564743419303377
        f1_score_train 0.6593406593406593
        business_metric_test 52879.35222672065
        business_metric_train 53561.133603238864
Random Forest
        accuracy_test 0.6981376518218624
        accuracy_train 0.9861808367071525
        precision_test 0.7117629452259803
        precision_train 0.9832792367463362
        recall_test 0.7252211039951204
        recall_train 0.99147079242289
        f1_score_test 0.7184290030211479
        f1_score_train 0.9873580246913579
        business_metric_test 51713.36032388664
        business_metric_train 49532.52361673414
Gradient Boosted Trees
        accuracy_test 0.7525506072874494
        accuracy_train 0.7710121457489879
        precision_test 0.876384316073032
        precision_train 0.893577259761975
        recall_test 0.7242146920603513
        recall_train 0.7419354838709677
        f1_score_test 0.7930660888407368
        f1_score_train 0.8107263965732644
        business_metric_test 51169.230769230766
        business_metric_train 51684.21052631579
```

It seems like the gradient boosted tree is working best for this dataset (without any tuning). Let's write a function to do k-fold cross validation and tune the parameters.

```
In [41]: from sklearn.model_selection import KFold, StratifiedKFold

In [42]: def train_evaluate_kfold(model_builder, x, y,
                                  metrics = {}, k = 5, stratify = False, model_params = {}):
             ''' Train k-fold cross validation and return results
                 in a DateFrame. '''

             if stratify:
```

```
            kf = StratifiedKFold(k, y)
        else:
            kf = KFold(k)

        results = {}

        for name in metrics.keys():
            results[name + '_test'] = np.zeros(k)
            results[name + '_train'] = np.zeros(k)

        fold_idx = 0
        for train_idx, test_idx in kf.split(x, y):
            model = model_builder(**model_params)


            current_result = train_evaluate(
                model = model,
                x_train = x[train_idx],
                x_test = x[test_idx],
                y_train = y[train_idx],
                y_test = y[test_idx],
                metrics = metrics
            )

            for res in current_result.keys():
                results[res][fold_idx] = current_result[res]

            fold_idx += 1

        return pd.DataFrame(results)
```

In [43]:
```
kfold_results = train_evaluate_kfold(
        model_builder = GradientBoostingClassifier,
        x = x,
        y = y,
        k = 5,
        metrics = metrics,
        stratify = False,
        model_params = dict()
    )
```

In [44]:
```
kfold_results.head(5)
```

Out[44]:

	accuracy_test	accuracy_train	precision_test	precision_train \
0	0.770445	0.771964	0.885192	0.893159
1	0.769028	0.771913	0.919428	0.914854
2	0.753239	0.772368	0.879735	0.897674
3	0.758907	0.775506	0.879955	0.897965

```
4       0.758704        0.773988        0.899236         0.904359

   recall_test  recall_train  f1_score_test  f1_score_train  \
0     0.746992      0.741466       0.810241        0.810274
1     0.724844      0.734809       0.810622        0.815006
2     0.728380      0.740500       0.796935        0.811547
3     0.727528      0.745546       0.796515        0.814688
4     0.729849      0.738969       0.805737        0.813341

   business_metric_test  business_metric_train
0          52123.481781           51347.672065
1          50698.380567           51707.995951
2          52004.048583           51416.497976
3          50672.064777           51703.947368
4          52495.951417           51263.663968
```

Now that training is easy, let's tune the hyperparameters of the model using this cross validation function.

```
In [45]: parameter_limits = {
             'n_estimators' : [20, 400],
             'max_depth' : [2, 5]
         }

         parameter_values = {
             'loss' : ['deviance', 'exponential']
         }

         parameter_generators = {
             'n_estimators' : lambda x: np.random.randint(parameter_limits['n_estimators'][0],
             'max_depth' : lambda x: np.random.randint(parameter_limits['max_depth'][0], parame
             'loss' : lambda x: np.random.choice(parameter_values['loss'], x)
         }
```

```
In [46]: from tqdm import tqdm
```

```
In [47]: def stack_dicts(dictionaries):
             ''' Assuming same keys for all dictionaries, stack them into one with lists of va

             keys = dictionaries[0].keys()

             output_dict = {}
             for key in keys:
                 output_dict[key] = [dictionaries[i][key] for i in range(len(dictionaries))]

             return output_dict

         def tune_hyperparameters(model_builder, n_iter, parameter_generators,
                                  x, y, metrics = {}, k = 5, stratify = False):
```

```
        ''' Use random search to find the metric values for different parameter sets. '''

        # Get some parameters to test
        parameters = {}
        for parameter in parameter_generators.keys():
            parameters[parameter] = parameter_generators[parameter](n_iter)

        # Summary of results
        summary = []

        # Test them
        for iteration in tqdm(range(n_iter)):

            # Build current iteration
            current_parameters = {}
            for parameter in parameters.keys():
                current_parameters[parameter] = parameters[parameter][iteration]

            # Find results
            current_results = train_evaluate_kfold(
                model_builder = GradientBoostingClassifier,
                x = x,
                y = y,
                k = 5,
                metrics = metrics,
                stratify = False,
                model_params = current_parameters
            )

            # Add scores by averaging folds
            current_parameters.update(current_results.mean().to_dict())
            summary.append(current_parameters)

        return pd.DataFrame(stack_dicts(summary))
```

```
In [48]: hyperparams_results = tune_hyperparameters(
        model_builder = GradientBoostingClassifier,
        n_iter = 100,
        x = x,
        y = y,
        k = 4,
        metrics = metrics,
        stratify = False,
        parameter_generators = parameter_generators
    )
```

```
100%|| 100/100 [40:41<00:00, 21.41s/it]
```

```
In [49]: hyperparams_results.sort_values('recall_test', ascending = False)
```

Out[49]:

	n_estimators	max_depth	loss	accuracy_test	accuracy_train	\
10	338	4	deviance	0.833077	0.884676	
30	320	4	deviance	0.829352	0.879089	
73	304	4	deviance	0.826356	0.874281	
67	374	4	exponential	0.822874	0.873704	
82	240	4	deviance	0.817206	0.858016	
36	386	3	deviance	0.814453	0.845466	
59	312	4	exponential	0.813684	0.858988	
54	230	4	deviance	0.814251	0.853836	
88	311	4	exponential	0.811538	0.858654	
12	287	4	exponential	0.808381	0.852368	
75	343	3	deviance	0.807692	0.837024	
57	266	4	exponential	0.806356	0.845496	
29	189	4	deviance	0.805506	0.841913	
14	285	3	deviance	0.800243	0.825364	
34	173	4	deviance	0.803198	0.836144	
2	371	3	exponential	0.795466	0.824960	
93	223	4	exponential	0.796761	0.830830	
24	159	4	deviance	0.798947	0.829686	
43	217	4	exponential	0.796437	0.829271	
33	255	3	deviance	0.795506	0.818340	
7	205	4	exponential	0.793806	0.826022	
74	232	3	deviance	0.792470	0.812996	
66	145	4	deviance	0.795668	0.823239	
25	332	3	exponential	0.788704	0.816437	
79	192	4	exponential	0.791417	0.821387	
15	330	3	exponential	0.788259	0.815739	
99	326	3	exponential	0.788057	0.815051	
69	202	4	exponential	0.790405	0.822763	
61	126	4	deviance	0.791457	0.815951	
21	211	3	deviance	0.787328	0.807257	
..	...	...	...	...	...	
56	244	2	exponential	0.758704	0.768411	
23	131	3	exponential	0.758502	0.771235	
40	129	3	exponential	0.757409	0.770182	
11	236	2	exponential	0.757085	0.766427	
63	123	3	exponential	0.757652	0.769514	
9	126	3	exponential	0.757247	0.770283	
80	117	3	exponential	0.757206	0.768472	
16	117	3	exponential	0.757166	0.768472	
62	158	2	deviance	0.755668	0.763887	
49	157	2	deviance	0.755547	0.763735	
27	159	2	deviance	0.755506	0.764069	
48	151	2	deviance	0.755182	0.762611	
84	55	4	deviance	0.763279	0.775506	
35	80	3	deviance	0.757247	0.767571	

0	51	4	deviance	0.760567	0.772794
72	191	2	exponential	0.751498	0.758816
77	185	2	exponential	0.750972	0.757996
44	164	2	exponential	0.745870	0.753583
90	45	4	deviance	0.756275	0.766771
46	52	4	exponential	0.754008	0.764879
87	50	3	deviance	0.749352	0.755800
89	96	2	exponential	0.738947	0.744504
53	94	2	exponential	0.738138	0.744211
8	68	2	deviance	0.740891	0.744717
32	76	2	exponential	0.737976	0.742794
85	31	4	exponential	0.742672	0.747257
18	59	2	exponential	0.735830	0.739109
83	22	2	exponential	0.715749	0.717389
45	22	2	exponential	0.715749	0.717389
42	22	2	exponential	0.715749	0.717389

	precision_test	precision_train	recall_test	recall_train	f1_score_test \
10	0.888468	0.931629	0.821186	0.867366	0.853390
30	0.887558	0.929040	0.816511	0.860932	0.850457
73	0.886526	0.926390	0.813021	0.855697	0.848082
67	0.886613	0.927564	0.808167	0.854081	0.845469
82	0.887880	0.922246	0.800008	0.835322	0.841557
36	0.885276	0.910736	0.797768	0.824950	0.839128
59	0.885518	0.921974	0.796602	0.836825	0.838585
54	0.887728	0.922338	0.796199	0.829500	0.839389
88	0.884132	0.922420	0.794533	0.836087	0.836814
12	0.883244	0.919699	0.790826	0.829054	0.834402
75	0.883051	0.907031	0.790132	0.815668	0.833917
57	0.884956	0.916678	0.787356	0.821512	0.833223
29	0.890155	0.919669	0.783687	0.815022	0.833467
14	0.884734	0.905199	0.779772	0.801298	0.828847
34	0.892381	0.918263	0.779698	0.808247	0.832165
2	0.880608	0.904328	0.775809	0.801251	0.824793
93	0.886897	0.914753	0.774274	0.803237	0.826713
24	0.893053	0.917171	0.774118	0.800497	0.829271
43	0.887193	0.913733	0.773792	0.801775	0.826542
33	0.885414	0.904070	0.773567	0.792879	0.825607
7	0.887109	0.913662	0.770593	0.797658	0.824683
74	0.886359	0.902845	0.769389	0.786761	0.823625
66	0.895564	0.916857	0.769022	0.792533	0.827376
25	0.879554	0.901590	0.767967	0.791741	0.819874
79	0.887921	0.912940	0.767317	0.792175	0.823134
15	0.880318	0.902087	0.767093	0.790623	0.819680
99	0.880162	0.901867	0.766928	0.789869	0.819515
69	0.887862	0.913329	0.766062	0.793698	0.822407
61	0.895141	0.915374	0.764242	0.784301	0.824388
21	0.885700	0.902141	0.763467	0.779985	0.819934

..	...	...	...	...	...
56	0.885387	0.891753	0.730651	0.738850	0.800504
23	0.886038	0.895902	0.730105	0.740389	0.800473
40	0.884840	0.894666	0.729364	0.739711	0.799539
11	0.884806	0.891694	0.729084	0.736661	0.799319
63	0.886643	0.896215	0.728985	0.738354	0.800022
9	0.886196	0.896272	0.728701	0.739190	0.799676
80	0.886726	0.895993	0.728466	0.737288	0.799744
16	0.886726	0.895993	0.728420	0.737288	0.799717
62	0.884156	0.890192	0.727806	0.734420	0.798291
49	0.884156	0.890396	0.727676	0.734174	0.798210
27	0.884075	0.890045	0.727657	0.734676	0.798168
48	0.885042	0.890821	0.726965	0.732781	0.798135
84	0.912021	0.921747	0.725646	0.735075	0.808188
35	0.895173	0.902468	0.725349	0.733770	0.801273
0	0.913118	0.922821	0.722426	0.731768	0.806591
72	0.886786	0.891957	0.722242	0.728180	0.796007
77	0.886349	0.891362	0.721831	0.727504	0.795582
44	0.885066	0.891660	0.716849	0.722649	0.792002
90	0.919597	0.927087	0.715742	0.723926	0.804912
46	0.913218	0.921852	0.715650	0.723799	0.802378
87	0.903624	0.908115	0.714086	0.719159	0.797670
89	0.892252	0.896970	0.707193	0.711250	0.788931
53	0.893431	0.898174	0.706000	0.710554	0.788631
8	0.903946	0.907335	0.705486	0.708063	0.792340
32	0.900477	0.904155	0.703631	0.707157	0.789825
85	0.933825	0.936859	0.697829	0.701329	0.798741
18	0.918464	0.920949	0.695878	0.698244	0.791774
83	0.947891	0.949380	0.670076	0.671530	0.784842
45	0.947891	0.949380	0.670076	0.671530	0.784842
42	0.947891	0.949380	0.670076	0.671530	0.784842

	f1_score_train	business_metric_test	business_metric_train
10	0.898339	50888.663968	50372.672065
30	0.893677	50925.910931	50428.542510
73	0.889628	50955.870445	50476.619433
67	0.889296	50990.688259	50482.388664
82	0.876620	51047.368421	50639.271255
36	0.865690	51074.898785	50764.777328
59	0.877323	51082.591093	50629.554656
54	0.873451	51076.923077	50681.072874
88	0.877121	51104.048583	50632.894737
12	0.872022	51135.627530	50695.748988
75	0.858899	51142.510121	50849.190283
57	0.866480	51155.870445	50764.473684
29	0.864182	51164.372470	50800.303644
14	0.850060	51217.004049	50965.789474
34	0.859739	51187.449393	50857.995951

2	0.849652	51264.777328	50969.838057
93	0.855372	51251.821862	50911.133603
24	0.854862	51229.959514	50922.570850
43	0.854094	51255.060729	50926.720648
33	0.844801	51264.372470	51036.032389
7	0.851724	51281.376518	50959.210526
74	0.840779	51294.736842	51089.473684
66	0.850157	51262.753036	50987.044534
25	0.843072	51332.388664	51055.060729
79	0.848272	51305.263158	51005.566802
15	0.842646	51336.842105	51062.044534
99	0.842120	51338.866397	51068.927126
69	0.849316	51315.384615	50991.801619
61	0.844745	51304.858300	51059.919028
21	0.836585	51346.153846	51146.862348
..	...	...	...
56	0.808118	51632.388664	51535.323887
23	0.810732	51634.412955	51507.085020
40	0.809821	51645.344130	51517.611336
11	0.806777	51648.582996	51555.161943
63	0.809633	51642.914980	51524.291498
9	0.810158	51646.963563	51516.599190
80	0.808900	51647.368421	51534.716599
16	0.808900	51647.773279	51534.716599
62	0.804821	51662.753036	51580.566802
49	0.804756	51663.967611	51582.085020
27	0.804916	51664.372470	51578.744939
48	0.804088	51667.611336	51593.319838
84	0.817883	51586.639676	51464.372470
35	0.809397	51646.963563	51543.724696
0	0.816260	51613.765182	51491.497976
72	0.801775	51704.453441	51631.275304
77	0.801127	51709.716599	51639.473684
44	0.798283	51760.728745	51683.603239
90	0.813002	51656.680162	51551.720648
46	0.810900	51679.352227	51570.647773
87	0.802655	51725.910931	51661.437247
89	0.793375	51829.959514	51774.392713
53	0.793406	51838.056680	51777.327935
8	0.795379	51810.526316	51772.267206
32	0.793581	51839.676113	51791.497976
85	0.802150	51792.712551	51746.862348
18	0.794279	51861.133603	51828.340081
83	0.786276	52061.943320	52045.546559
45	0.786276	52061.943320	52045.546559
42	0.786276	52061.943320	52045.546559

[100 rows x 13 columns]

Since running these random trials takes time, let's try to be smart and save them to a folder so that we don't lose them and we can gradually increase the number.

```
In [50]: import glob, os

         def check_and_create_directory(path):
             if not os.path.exists(path):
                 os.mkdir(path)
```

```
In [51]: random_samples_path = './samples'

         # Low chance of repeat
         sample_filename = '{}/{}.csv'.format(random_samples_path, np.random.randint(0, 1e9))
         print('Saving samples to {}'.format(sample_filename))

         check_and_create_directory(random_samples_path)
         hyperparams_results.to_csv(sample_filename, index = False)
```

```
Saving samples to ./samples/339227563.csv
```

```
In [52]: previous_results = []

         for f in glob.glob(random_samples_path + '/*.csv'):
             previous_results.append(pd.read_csv(f))

         if len(previous_results) > 0:
             samples = pd.concat(previous_results)
         else:
             samples = hyperparams_results
```

```
In [53]: samples.sort_values('recall_test', ascending = False)
```

```
Out[53]:     n_estimators  max_depth         loss  accuracy_test  accuracy_train  \
         23           393          4     deviance       0.837126        0.893745
         82           347          4     deviance       0.835506        0.887186
         40           342          4     deviance       0.834656        0.886265
         10           338          4     deviance       0.833077        0.884676
         28           346          4     deviance       0.832348        0.885294
         55           345          4     deviance       0.831619        0.884676
         15           339          4     deviance       0.832389        0.884393
         19           331          4     deviance       0.830607        0.881599
         30           320          4     deviance       0.829352        0.879089
         26           314          4     deviance       0.827571        0.877176
         73           304          4     deviance       0.826356        0.874281
         64           369          4  exponential       0.824534        0.874838
         79           364          4  exponential       0.821822        0.873006
         67           374          4  exponential       0.822874        0.873704
         62           344          4  exponential       0.817854        0.867561
```

35	335	4	exponential	0.817206	0.865688
60	334	4	exponential	0.816032	0.864939
78	333	4	exponential	0.816073	0.863947
9	242	4	deviance	0.818866	0.859140
82	240	4	deviance	0.817206	0.858016
56	398	3	deviance	0.814332	0.848026
36	386	3	deviance	0.814453	0.845466
59	312	4	exponential	0.813684	0.858988
54	230	4	deviance	0.814251	0.853836
88	311	4	exponential	0.811538	0.858654
86	220	4	deviance	0.814049	0.852257
91	218	4	deviance	0.812834	0.849575
12	287	4	exponential	0.808381	0.852368
75	343	3	deviance	0.807692	0.837024
73	277	4	exponential	0.805223	0.849514
..	...	...	...	...	...
53	177	2	exponential	0.745830	0.754666
63	115	2	deviance	0.746194	0.753644
90	45	4	deviance	0.756275	0.766771
46	52	4	exponential	0.754008	0.764879
44	106	2	deviance	0.745749	0.752287
87	50	3	deviance	0.749352	0.755800
32	131	2	exponential	0.740891	0.748047
36	130	2	exponential	0.741174	0.748036
6	49	3	exponential	0.745020	0.752358
89	96	2	exponential	0.738947	0.744504
70	80	2	deviance	0.742591	0.748279
84	37	3	deviance	0.745992	0.750648
51	101	2	exponential	0.738543	0.744899
53	94	2	exponential	0.738138	0.744211
8	68	2	deviance	0.740891	0.744717
75	35	4	deviance	0.748866	0.755314
32	76	2	exponential	0.737976	0.742794
27	72	2	exponential	0.736518	0.741599
46	30	4	exponential	0.743360	0.747166
4	44	2	deviance	0.735870	0.736680
85	31	4	exponential	0.742672	0.747257
37	30	2	deviance	0.735101	0.736356
18	59	2	exponential	0.735830	0.739109
80	21	3	exponential	0.735951	0.737480
30	21	3	exponential	0.735951	0.737480
83	22	2	exponential	0.715749	0.717389
42	22	2	exponential	0.715749	0.717389
45	22	2	exponential	0.715749	0.717389
59	21	2	exponential	0.695830	0.695921
0	20	2	exponential	0.695547	0.695698

precision_test precision_train recall_test recall_train f1_score_test \

23	0.888646	0.935392	0.826721	0.878257	0.856460
82	0.891029	0.934367	0.822956	0.869183	0.855564
40	0.890737	0.934255	0.821950	0.867903	0.854890
10	0.888468	0.931629	0.821186	0.867366	0.853390
28	0.888041	0.932078	0.820346	0.867963	0.852766
55	0.887452	0.931911	0.819707	0.867164	0.852142
15	0.889934	0.933254	0.819317	0.865826	0.853100
19	0.888494	0.931024	0.817676	0.863246	0.851536
30	0.887558	0.929040	0.816511	0.860932	0.850457
26	0.888279	0.929336	0.813637	0.857932	0.849245
73	0.886526	0.926390	0.813021	0.855697	0.848082
64	0.884619	0.926645	0.811667	0.856346	0.846462
79	0.882646	0.925924	0.809199	0.854202	0.844195
67	0.886613	0.927564	0.808167	0.854081	0.845469
62	0.882073	0.924218	0.804131	0.847468	0.841186
35	0.882875	0.923460	0.802807	0.845299	0.840815
60	0.882652	0.923276	0.801327	0.844359	0.839910
78	0.883022	0.922902	0.801168	0.843197	0.839993
9	0.891108	0.924709	0.800447	0.835350	0.843264
82	0.887880	0.922246	0.800008	0.835322	0.841557
56	0.883408	0.912540	0.798702	0.827356	0.838786
36	0.885276	0.910736	0.797768	0.824950	0.839128
59	0.885518	0.921974	0.796602	0.836825	0.838585
54	0.887728	0.922338	0.796199	0.829500	0.839389
88	0.884132	0.922420	0.794533	0.836087	0.836814
86	0.891946	0.922189	0.793726	0.827439	0.839884
91	0.892010	0.922243	0.792095	0.823768	0.839007
12	0.883244	0.919699	0.790826	0.829054	0.834402
75	0.883051	0.907031	0.790132	0.815668	0.833917
73	0.881479	0.918796	0.787863	0.825756	0.831913
..	...	...	...	...	...
53	0.885705	0.892465	0.716606	0.723508	0.792147
63	0.888439	0.894186	0.715999	0.721782	0.792877
90	0.919597	0.927087	0.715742	0.723926	0.804912
46	0.913218	0.921852	0.715650	0.723799	0.802378
44	0.888622	0.893642	0.715481	0.720535	0.792629
87	0.903624	0.908115	0.714086	0.719159	0.797670
32	0.885447	0.892161	0.711493	0.716562	0.788913
36	0.886732	0.893211	0.711379	0.716188	0.789343
6	0.902077	0.907460	0.710113	0.715832	0.794597
89	0.892252	0.896970	0.707193	0.711250	0.788931
70	0.904602	0.908246	0.706890	0.711351	0.793519
84	0.915985	0.918882	0.706571	0.710287	0.797707
51	0.893250	0.899745	0.706447	0.710726	0.788867
53	0.893431	0.898174	0.706000	0.710554	0.788631
8	0.903946	0.907335	0.705486	0.708063	0.792340
75	0.930253	0.935584	0.704983	0.709565	0.802033
32	0.900477	0.904155	0.703631	0.707157	0.789825

27	0.903510	0.907638	0.701167	0.704835	0.789452
46	0.931850	0.934801	0.699279	0.701881	0.798849
4	0.909127	0.909829	0.698689	0.699269	0.790090
85	0.933825	0.936859	0.697829	0.701329	0.798741
37	0.910530	0.911477	0.697498	0.698459	0.789859
18	0.918464	0.920949	0.695878	0.698244	0.791774
80	0.938959	0.938790	0.690167	0.691732	0.795440
30	0.938959	0.938790	0.690167	0.691732	0.795440
83	0.947891	0.949380	0.670076	0.671530	0.784842
42	0.947891	0.949380	0.670076	0.671530	0.784842
45	0.947891	0.949380	0.670076	0.671530	0.784842
59	0.973032	0.973241	0.647654	0.647805	0.777653
0	0.978228	0.978143	0.646482	0.646616	0.778450

	f1_score_train	business_metric_test	business_metric_train
23	0.905915	50848.178138	50281.983806
82	0.900591	50864.372470	50347.570850
40	0.899851	50872.874494	50356.781377
10	0.898339	50888.663968	50372.672065
28	0.898869	50895.951417	50366.497976
55	0.898363	50903.238866	50372.672065
15	0.898269	50895.546559	50375.506073
19	0.895847	50913.360324	50403.441296
30	0.893677	50925.910931	50428.542510
26	0.892198	50943.724696	50447.672065
73	0.889628	50955.870445	50476.619433
64	0.890089	50974.089069	50471.052632
79	0.888593	51001.214575	50489.372470
67	0.889296	50990.688259	50482.388664
62	0.884163	51040.890688	50543.825911
35	0.882636	51047.368421	50562.550607
60	0.882040	51059.109312	50570.040486
78	0.881230	51058.704453	50579.959514
9	0.877755	51030.769231	50628.036437
82	0.876620	51047.368421	50639.271255
56	0.867849	51076.113360	50739.170040
36	0.865690	51074.898785	50764.777328
59	0.877323	51082.591093	50629.554656
54	0.873451	51076.923077	50681.072874
88	0.877121	51104.048583	50632.894737
86	0.872240	51078.947368	50696.862348
91	0.870226	51091.093117	50723.684211
12	0.872022	51135.627530	50695.748988
75	0.858899	51142.510121	50849.190283
73	0.869766	51167.206478	50724.291498
..	...	...	...
53	0.799137	51761.133603	51672.773279
63	0.798782	51757.489879	51682.995951

```
90          0.813002          51656.680162          51551.720648
46          0.810900          51679.352227          51570.647773
44          0.797803          51761.943320          51696.558704
87          0.802655          51725.910931          51661.437247
32          0.794774          51810.526316          51738.967611
36          0.794957          51807.692308          51739.068826
6           0.800313          51769.230769          51695.850202
89          0.793375          51829.959514          51774.392713
70          0.797817          51793.522267          51736.639676
84          0.801214          51759.514170          51712.955466
51          0.794133          51834.008097          51770.445344
53          0.793406          51838.056680          51777.327935
8           0.795379          51810.526316          51772.267206
75          0.807027          51730.769231          51666.295547
32          0.793581          51839.676113          51791.497976
27          0.793456          51854.251012          51803.441296
46          0.801724          51785.829960          51747.773279
4           0.790756          51860.728745          51852.631579
85          0.802150          51792.712551          51746.862348
37          0.790852          51868.421053          51855.870445
18          0.794279          51861.133603          51828.340081
80          0.796403          51859.919028          51844.635628
30          0.796403          51859.919028          51844.635628
83          0.786276          52061.943320          52045.546559
42          0.786276          52061.943320          52045.546559
45          0.786276          52061.943320          52045.546559
59          0.777805          52261.133603          52260.222672
0           0.778553          52263.967611          52262.449393

[200 rows x 13 columns]
```

Let's now take a look at the parameter set which has the highest recall and use that to train our final model.

```
In [54]: best_parameters = {
             'loss' : 'deviance',
             'n_estimators' : 360,
             'max_depth' : 4
         }
```

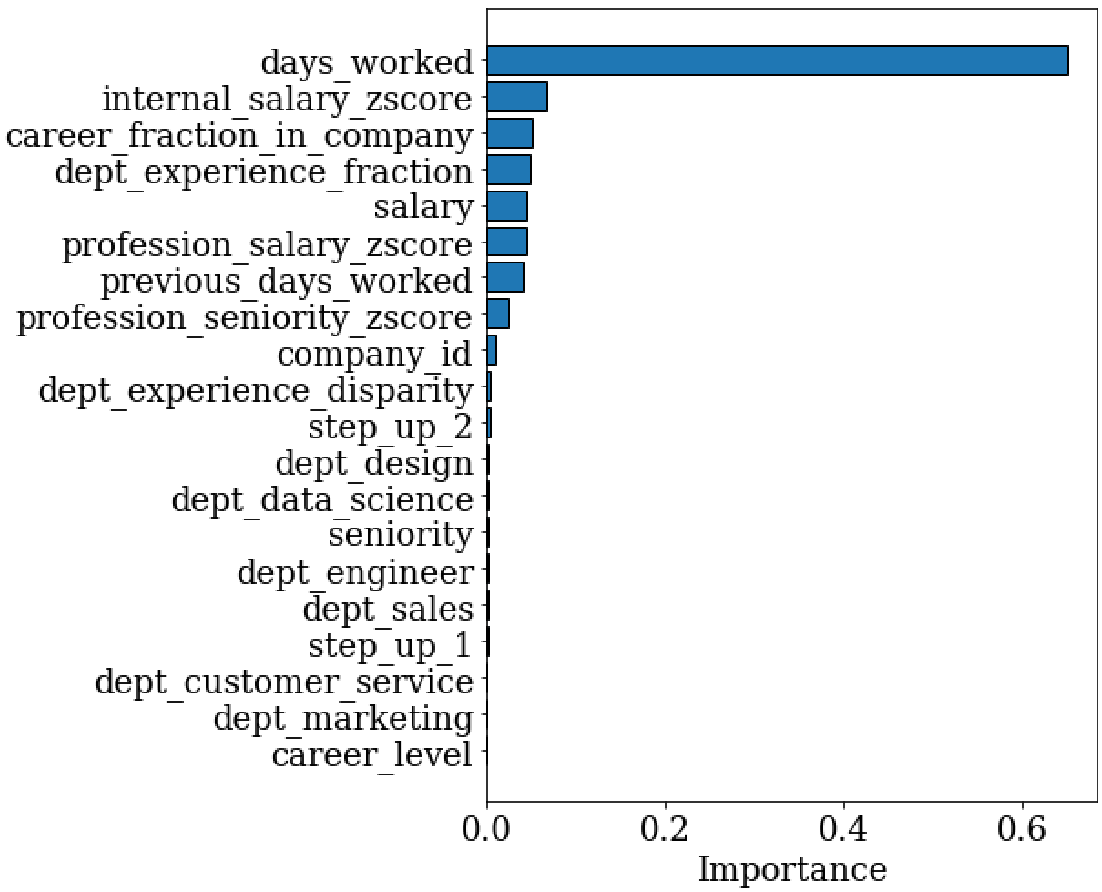
```
In [55]: model, results = train_evaluate(
             model = GradientBoostingClassifier(**best_parameters),
             x_train = x_train,
             x_test = x_test,
             y_train = y_train,
             y_test = y_test,
             metrics = metrics,
             return_model = True
         )
```

In [56]: results

Out[56]:
```
{'accuracy_test': 0.8241295546558705,
 'accuracy_train': 0.8907422402159244,
 'precision_test': 0.8790781203232565,
 'precision_train': 0.9355758827579423,
 'recall_test': 0.8115501519756839,
 'recall_train': 0.8741843580553258,
 'f1_score_test': 0.8439655172413792,
 'f1_score_train': 0.9038388445458,
 'business_metric_test': 50453.44129554656,
 'business_metric_train': 50486.909581646425}
```

In [57]:
```
idx = np.argsort(model.feature_importances_)

plt.figure(figsize = (6, 8))
plt.barh(
    [features[i] for i in idx],
    [model.feature_importances_[i] for i in idx],
    edgecolor = 'k'
)
plt.xlabel('Importance')
```

Out[57]:
```
Text(0.5, 0, 'Importance')
```



2 6. Interpretation of Model

The model trainined above heavily relies on knowing how many days an employee has worked so far. The trend is clearly evident from the visual exploration; people quit around one and two years on the job. Our simple baseline model took advantage of this fact as well. The second most important factor in attrition appears to be the relative salary gap between peers working in the same division.

2.0.1 6.1 Recommended Actions

- Track employees and focus on those around the one and two year mark, the model can predict those at risk employees for you.
- Offer these employees a raise that brings their salary closer to the mean for their division and career level. The degree to which this raise can be offered without affecting the bottom line depends on how much you expect to lose by replacing this candidate with a less experienced one (hard to quantify).
- Start tracking job satisfaction and hours worked per week. Stronger predictions could be made if the company collected job satisfaction surveys and tracked the hours each employee is working on per week.

- The business metric defined is probably not perfect but it follows the recall, so maximizing recall seems sensible.

```
In [58]: np.sum(data['churn_status'].values) * 100000 / len(data)
```

```
Out[58]: 54704.45344129555
```

The number above is the cost (per employee) of replacing all those employees who quit if we offer no incentives. When compared with our best business metric (cost per employee) we see that we can save 4,000 - 5,000 per employee.