EDA - Employee Attrition

Module 2 | Chapter 3 | Notebook 2

In this notebook, we'll take a closer look at the data that we'll be processing in this chapter to familiarize ourselves with them. You'll noticed that we can reduce the number of features using PCA without losing much information.

The attrition data

Scenario: You work for an international global logistics company. Due to the tense situation on the labor market, the company is finding it increasingly difficult to attract new talent. For this reason, management has decided to limit the turnover of existing employees. You should predict which employees are likely to want to leave the company so that measures can be taken to encourage them to stay.

You can find the data about employees who leave in *attrition_train.csv*. You've also been provided test data in the file *attrition_test.csv*.

- Import the training data and store it as a `DataFrame` named `df_train`. Store the features in the variable `features_train` and store the target vector `'attrition'` in the variable `target_train`. Then print the first 5 rows of `df_train`.

In [1]:
```
import pandas as pd
df_train = pd.read_csv('attrition_train.csv')
features_train = df_train.iloc[:,1:]
target_train = df_train.iloc[:,0]
df_train.head()
```

Out[1]:

	attrition	age	gender	businesstravel	distancefromhome	education	joblevel	maritalstatus	mont
0	0	30	0	1	5.0	3	2	1	
1	1	33	0	1	5.0	3	1	0	
2	0	45	1	1	24.0	4	1	0	
3	0	28	1	1	15.0	2	1	1	
4	0	30	1	1	1.0	3	1	2	

5 rows × 21 columns

The code for that looks like this:

Column number	Column name	Type	Description
0	`'attrition'`	categorical	Whether the employee has left the company (`1`) or not (`0`)
1	`age`	continuous (`int`)	The person's age in years
2	`'gender'`	categorical (nominal, `int`)	Gender: male (`1`) or female (`0`)
3	`'businesstravel'`	categorical (ordinal, `int`)	How often the employee is on a business trip: often (`2`), rarely (`1`) or never (`0`)
4	`'distancefromhome'`	continuous (`int`)	Distance from home address to work address in kilometers
5	`'education'`	categorical (ordinal, `int`)	Level of education: doctorate (`5`), master (`4`), bachelor (`3`), apprenticeship(`2`), Secondary school qualifications (`1`)
6	`'joblevel'`	categorical (ordinal, `int`)	Level of responsibility: Executive (`5`), Manager (`4`), Team leader (`3`), Senior employee (`2`), Junior employee (`1`)
7	`'maritalstatus'`	categorical (nominal, `int`)	Marital status: married (`2`), divorced (`1`), single (`0`)
8	`'monthlyincome'`	continuous (`int`)	Gross monthly salary in EUR
9	`'numcompaniesworked'`	continuous (`int`)	The number of enterprises where the employee worked before their current position
10	`'over18'`	categorical (`int`)	Whether the employee is over 18 years of age (`1`) or not (`0`)
11	`'overtime'`	categorically (`int`)	Whether or not they have accumulated overtime in the past year (`1`) or not (`0`)
12	`'percentsalaryhike'`	continuous (`int`)	Salary increase in percent within the last twelve months
13	`'standardhours'`	continuous (`int`)	contractual working hours per two weeks
14	`'stock option levels'`	categorical (ordinal, `int`)	options on company shares: very many (`4`), many (`3`), few (`2`), very little (`1`), none (`0`)
15	`'trainingtimeslastyear'`	continuous (`int`)	Number of training courses taken in the last 12 months
16	`'totalworkingyears'`	continuous (`int`)	Number of years worked: Number of years on the job market and as an employee
17	`'years_atcompany'`	continuous (`int`)	Number of years at the current company Number of years in the current company

Column number	Column name	Type	Description
18	`'years_currentrole'`	continuous (`int`)	Number of years in the current position
19	`'years_lastpromotion'`	continuous (`int`)	Number of years since the last promotion
20	`'years_withmanager'`	continuous (`int`)	Number of years working with current manager

Each row in `df_train` represents an employee

Let's explore the data set a little bit to get a first impression of the data. What is the ratio of employees who have left to those who stayed in the company? We can determine this quickly with the `pd.crosstab()` function, for example. It creates a contingency table for a column. `pd.crosstab()` therefore counts how often each value of a column occurs and returns this as a new `DataFrame`. `pd.crosstab()` takes the parameters `index`, `columns` and `normalize`. You assign the feature you're interested in to `index`, e.g. `df_train.loc[:, 'attrition']`. `columns` is the name of the columns in the contingency table. You can assign this a `str` such as `'count'`. The `normalize` parameter specifies whether you want the contingency table to contain whole numbers or proportions. With `normalize='columns'` the proportions are calculated column by column.

Create the contingency table for `df_train.loc[:, 'attrition']` and calculate the shares column by column. Name the resulting `DataFrame` `attrition_prop_train` and then print it.

In [2]:
```
attrition_prop_train = pd.crosstab(index=target_train, columns='count', normalize='col
attrition_prop_train
```

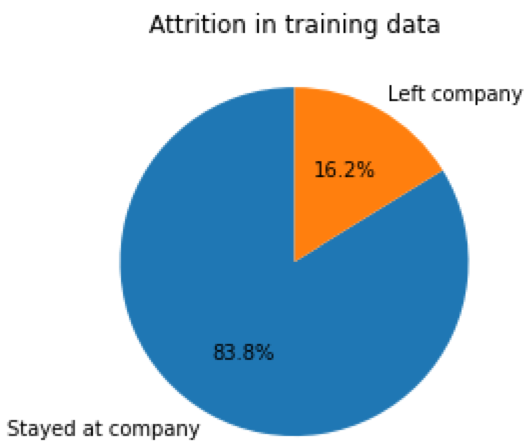
Out[2]:

col_0	count
attrition	
0	0.837707
1	0.162293

Now visualize the proportions in `attrition_prop_train` with a pie chart.

In [3]:
```
# module import
import matplotlib.pyplot as plt

# initialize figure and axes
fig, ax = plt.subplots()

# draw pie chart
attrition_prop_train.plot(kind='pie',
                          y='count',
                          labels=['Stayed at company', 'Left company'],  # set labels
                          autopct='%1.1f%%',  # print values to
                          legend=False,  # do not print legend
```

```
                          startangle=90,  # rotate pie
                          title='Attrition in training data',  # set title
                          ax=ax)

# optimize pie chart
ax.set_ylabel('')  # do not print "count"
ax.set_aspect('equal')  # draw a circle, not an ellipse
```



After a bit of tweaking, a pie chart could look like this:

Attrition in training data

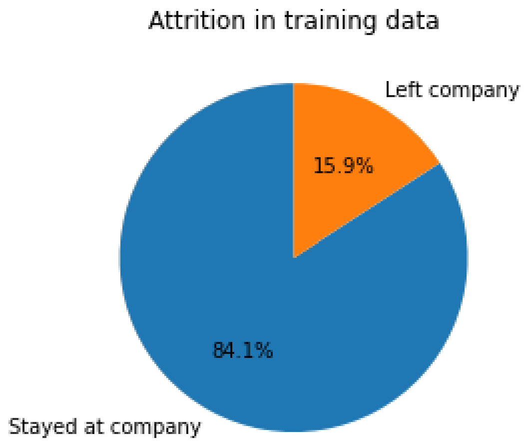
It looks like we're dealing with a very unbalanced data set. There are a lot more people who stay in the company (84%) than people who have left (16%). A classification algorithm could take advantage of this and simply always predict that employees will stay with the company. This strategy would achieve a remarkable accuracy of 84%. With unbalanced data sets, you have to be particularly careful that your classification model is not deceiving you in this way.

Is the ratio in the test data similar? This is important so that the training data is representative of the test data. Use a pie chart to visualize the percentage of employees who stayed and those who left in the test data in the *attrition_test.csv* file.

In [4]:

```
import pandas as pd
df_test = pd.read_csv('attrition_test.csv')
features_test = df_test.iloc[:,1:]
target_test = df_test.iloc[:,0]

attrition_prop_test = pd.crosstab(index=target_test, columns='count', normalize='colum

# initialize figure and axes
fig, ax = plt.subplots()

# draw pie chart
attrition_prop_test.plot(kind='pie',
                          y='count',
                          labels=['Stayed at company', 'Left company'],  # set labels
                          autopct='%1.1f%%',  # print values to
                          legend=False,  # do not print legend
                          startangle=90,  # rotate pie
                          title='Attrition in training data',  # set title
```

```
                                    ax=ax)

# optimize pie chart
ax.set_ylabel('')  # do not print "count"
ax.set_aspect('equal')  # draw a circle, not an ellipse
```



The category sizes are roughly the same, with 15.9% of employees leaving the company. The test data set appears to be similar to the training data set on first glance.

What about the rest of the columns? Print the eight-value summary of the training data. This way you get a good overview. Pay special attention to the `'std'` information, which indicates the data's dispersion. If the dispersion is zero, the column doesn't contain any information that a machine learning model can learn.

In [5]:
```
df_train.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
attrition	1029.0	0.162293	0.368899	0.0	0.0	0.0	0.0	1.0
age	1029.0	36.788144	9.053226	18.0	30.0	36.0	42.0	60.0
gender	1029.0	0.611273	0.487698	0.0	0.0	1.0	1.0	1.0
businesstravel	1029.0	1.083576	0.531875	0.0	1.0	1.0	1.0	2.0
distancefromhome	1029.0	9.114674	8.066146	1.0	2.0	7.0	14.0	29.0
education	1029.0	2.934888	1.000310	1.0	2.0	3.0	4.0	5.0
joblevel	1029.0	2.055394	1.108792	1.0	1.0	2.0	3.0	5.0
maritalstatus	1029.0	1.121477	0.873193	0.0	0.0	1.0	2.0	2.0
monthlyincome	1029.0	6464.963071	4744.912070	1009.0	2867.0	4809.0	8321.0	19999.0
numcompaniesworked	1029.0	2.733722	2.534785	0.0	1.0	2.0	4.0	9.0
over18	1029.0	1.000000	0.000000	1.0	1.0	1.0	1.0	1.0
overtime	1029.0	0.274052	0.446252	0.0	0.0	0.0	1.0	1.0
percentsalaryhike	1029.0	15.217687	3.658480	11.0	12.0	14.0	18.0	25.0
standardhours	1029.0	80.000000	0.000000	80.0	80.0	80.0	80.0	80.0
stockoptionlevels	1029.0	0.800777	0.849673	0.0	0.0	1.0	1.0	3.0
trainingtimeslastyear	1029.0	2.776482	1.242327	0.0	2.0	3.0	3.0	6.0
totalworkingyears	1029.0	11.149660	7.716836	0.0	6.0	9.0	15.0	40.0
years_atcompany	1029.0	6.976676	6.133838	0.0	3.0	5.0	9.0	40.0
years_currentrole	1029.0	4.188533	3.636368	0.0	2.0	3.0	7.0	18.0
years_lastpromotion	1029.0	2.185617	3.211959	0.0	0.0	1.0	3.0	15.0
years_withmanager	1029.0	4.089407	3.550603	0.0	2.0	3.0	7.0	17.0

The two following columns seem to stand out:

- `'over18'`
- `'standardhours'`

They both only contain one unique value (`1` or `80`) and therefore don't provide us with any information to predict which employees will leave. Remove these two columns from the training and test data.

In [6]:
```
df_train = df_train.drop(['over18','standardhours'], axis=1)
df_test = df_test.drop(['over18','standardhours'], axis=1)
```

Next, visualize the training data with a scatterplot matrix, see *EDA - Understanding Data (Module 1, Chapter 1)*.

In [7]:
```
import seaborn as sns
```

```python
# draw correlogram
sns.pairplot(df_train)
```

Out[7]: <seaborn.axisgrid.PairGrid at 0x7fe63c7c7c40>

Remember that each *axes* represents the scatterplot of two columns. The histograms on the diagonal represent the distributions of the columns. The scatterplot matrix is mirrored along this diagonal.

One way to approach this is to look for unusual distributions in the histograms. With the scatter plots you can pay special attention to strong positive or negative correlations. If all the points lie along a diagonal, then these features are correlated.

In this case, no features or combinations of features seem to be particularly striking.

Congratulations: You've explored the data that we'll be using in this chapter. We've already removed two features because they don't contain any useful information. Next, we'll see if a principal component analysis can be used to further reduce the number of features.

Principal Component Analysis of the "years" features

Since the scatter plots in the matrix are not so easy to get an overview of, it could be a good idea to summarize them with correlation coefficients. Calculate the correlation coefficients of all feature combinations of the continuous features within the training data. Their names are summarized in `num_cols` :

In [8]:
```
num_cols = ['age',
            'distancefromhome',
            'monthlyincome',
            'numcompaniesworked',
            'percentsalaryhike',
            'trainingtimeslastyear',
            'totalworkingyears',
            'years_atcompany',
            'years_currentrole',
            'years_lastpromotion',
            'years_withmanager']
```

Use `num_cols` to calculate the correlation coefficients of all numerical features in the training data.

In [9]:
```
features_train.loc[:, num_cols].corr()
```

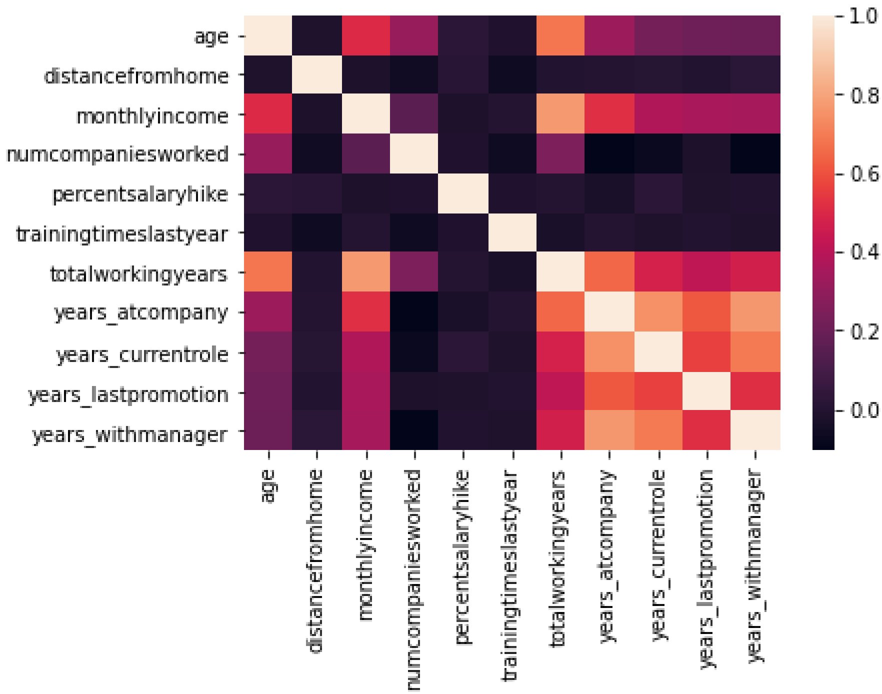
Out[9]:

	age	distancefromhome	monthlyincome	numcompaniesworked	percentsa
age	1.000000	-0.015626	0.501744	0.311860	
distancefromhome	-0.015626	1.000000	-0.020628	-0.052172	
monthlyincome	0.501744	-0.020628	1.000000	0.155551	-
numcompaniesworked	0.311860	-0.052172	0.155551	1.000000	-
percentsalaryhike	0.024361	0.013394	-0.017632	-0.009058	
trainingtimeslastyear	-0.010961	-0.062188	0.001075	-0.061857	-
totalworkingyears	0.682786	-0.002480	0.767788	0.247163	
years_atcompany	0.327015	0.003062	0.521506	-0.103507	-
years_currentrole	0.226492	0.012362	0.377688	-0.069373	
years_lastpromotion	0.210601	-0.003676	0.355648	-0.019731	-
years_withmanager	0.202227	0.020429	0.349325	-0.101654	-

It is not so easy to keep track of all these figures. We can use a heat map to get an overview of the situation, see *Preparing Continuous Features (chapter 2)*.

In [10]:
```
sns.heatmap(features_train.loc[:, num_cols].corr())
```

Out[10]: <AxesSubplot:>



A number of features that correlate fairly strongly with each other are particularly noticeable:

- `'totalworkingyears'`
- `'years_atcompany'`
- `'years_currentrole'`
- `'years_lastpromotion'`
- `'years_withmanager'`

Let's summarize the relevant column names in `col_correlated`:

In [11]:
```
col_correlated = ['totalworkingyears',
                  'years_atcompany',
                  'years_currentrole',
                  'years_lastpromotion',
                  'years_withmanager']
```

These features all correlate with each other, with a correlation coefficient of at least 0.4. In *Module 1, Chapter 4* we learned that in this kind of situation, a principal component analysis can extract the most important information. This allows us to reduce the number of features.

We use the following components for the principal component analysis:

- `Pipeline` from `sklearn.pipeline`
- `StandardScaler` from `sklearn.preprocessing`
- `PCA` (from `sklearn.decomposition`)

Import it.

In [12]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Now define `Pipeline()`, called `std_pca`, which should first standardize the data and then reduce it with a principal component analysis, see *Principal Component Analysis as Part of the Data Pipeline (Module 1, Chapter 4)*. You should retain 80% of the variance in the data, see *Data Compression with PCA (Module 1, Chapter 4)*. We have chosen the value 80% because it allows us to compress the data into a manageable number of features without losing too much information.

In [13]:
```
std_pca = Pipeline([('std', StandardScaler()),
                    ('pca', PCA(n_components=0.8))])
```

`col_correlated` contains the following columns:

- `'totalworkingyears'`
- `'years_atcompany'`
- `'years_currentrole'`
- `'years_lastpromotion'`
- `'years_withmanager'`

Now use the `my_transformer.fit_transform()` method to first standardize these columns in `df_train` and then reduce them with a principal component analysis: Store the resulting array in `arr_years_train` and check how many columns it contains.

In [14]:
```
arr_years_train = std_pca.fit_transform(features_train.loc[:, col_correlated])
arr_years_train.shape
```

Out[14]:
```
(1029, 2)
```

We were able to reduce the five correlating columns to just two. It's best to replace the columns that we have reduced. First remove them from `df_train` by executing the following cell.

In [15]:
```
features_train = features_train.drop(col_correlated, axis=1)
```

Now add two new columns to `features_train` which represent the reduced columns. Call them `'pca_years_0'` and `'pca_years_1'`.

In [16]:
```
features_train.loc[:, 'pca_years_0'] = arr_years_train[:, 0]
features_train.loc[:, 'pca_years_1'] = arr_years_train[:, 1]
```

We should now do the same for the test data. Remember to transform the test data with the same settings as you did with the training data before it. Delete the `col_correlated` columns and replace them with new principal components named `'pca_years_0'` and `'pca_years_1'`.

In [17]:
```
# pca
arr_years_test = std_pca.transform(features_test.loc[:, col_correlated])

# remove old features
features_test = features_test.drop(col_correlated, axis=1)
```

```
# add pca features
features_test.loc[:, 'pca_years_0'] = arr_years_test[:, 0]
features_test.loc[:, 'pca_years_1'] = arr_years_test[:, 1]
```

Use the following code cell for a few sanity checks to see if the principal component analysis went as expected. Answer the following questions yourself:

- Do the new columns `'pca_years_0'` and `'pca_years_1'` correlate with each other in the training data or in the test data? They shouldn't.
- Do the new columns `'pca_years_0'` and `'pca_years_1'` correlate with other continuous columns very strongly in the training data or in the test data? They shouldn't.
- Is the the new `'pca_years_0'` column on a similar scale in training data and test data? This should be the case.
- Is the scale of the new column `'pca_years_1'` similar in training data and test data? This should be the case.

Tip: If you use the Jupyter Lab `display()` function instead of the `print()` function, you can display several DataFrames in the same code cell much more clearly.

In [18]:
```
display(features_train.loc[:, :].corr().tail(2))
display(features_test.loc[:, :].corr().tail(2))
display(features_train.describe().T.tail(2))
display(features_test.describe().T.tail(2))
```

	age	gender	businesstravel	distancefromhome	education	joblevel	maritalstatus
pca_years_0	0.388596	-0.069862	0.005704	0.007735	0.088830	0.589057	0.105036
pca_years_1	0.547528	-0.012325	0.023576	-0.009973	0.105514	0.500732	0.009050

	age	gender	businesstravel	distancefromhome	education	joblevel	maritalstatus
pca_years_0	0.369259	0.022902	-0.004098	0.028190	0.110760	0.572272	0.018434
pca_years_1	0.530655	-0.032935	-0.026095	-0.006873	0.102878	0.512633	0.056226

	count	mean	std	min	25%	50%	75%	max
pca_years_0	1029.0	-4.488365e-17	1.841763	-2.481086	-1.339399	-0.615637	1.033808	6.816757
pca_years_1	1029.0	8.199898e-18	0.783462	-1.862920	-0.445751	-0.192191	0.277034	3.056317

	count	mean	std	min	25%	50%	75%	max
pca_years_0	441.0	0.063398	1.844224	-2.481086	-1.257373	-0.533122	1.087543	5.955334
pca_years_1	441.0	0.032717	0.823529	-1.617410	-0.472658	-0.201257	0.269724	3.271972

The new columns only correlate relatively strongly with the `'age'`, `'joblevel'` and `'monthlyincome'` columns. This is not surprising considering the origin of the new columns. As you would expect with a principal component analysis, the new columns don't correlate with each other. They also have very similar scales in the training data to in the test data. The sanity checks were successful. You successfully reduced the number of columns.

Now let's have a look at what our cleaned and reduced `features_train` DataFrame looks like. Then print the number of rows and columns as well as the first five rows of the `DataFrame`.

In [19]:
```
print(features_train.shape)
features_train.head()
```

```
(1029, 17)
```

Out[19]:

	age	gender	businesstravel	distancefromhome	education	joblevel	maritalstatus	monthlyincome
0	30	0	1	5.0	3	2	1	6118.0
1	33	0	1	5.0	3	1	0	2851.0
2	45	1	1	24.0	4	1	0	2177.0
3	28	1	1	15.0	2	1	1	2207.0
4	30	1	1	1.0	3	1	2	3833.0

We were able to reduce the relevant features for the training set from 19 to 15!

Congratulations: You have successfully prepared the data of this chapter and used knowledge from previous chapters. So now we can turn to a new classification algorithm: Decision trees.

Remember:

- You can generally remove features that only have one value.
- The training data should be similar to the test data.
- If features are correlated, principal component analysis is a useful tool for reducing the number of features.

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.
