PDPs and ICE Plots

Module 3 | Chapter 1 | Notebook 4

In this exercise, we'll look at two more model agnostic methods for global interpretation. By the end of this lesson you will have:

- Investigated the average influence of a feature on the predictions.
- Visualized the influence of the values of a feature for each data point.
- Investigated the shared average influence of two feature on the predictions.

Partial dependence plots

Interpreting the results of a black-box model is a challenge. In the last lesson, you learned about the permutation feature importance, a model diagnostic method that can help you to take a closer look at your model. But it only gives us the importance of a feature and does not describe how the values the feature takes on affect our predictions.

So we'll look at two more methods in this lesson: *partial dependence plots* (PDPs) and *individual conditional expectation plots* (ICE plots). We'll apply these methods to our random forest, but since they can be used for all models, you could also use them to look at a neural network or SVM.

Run the following code cell to import, split and print the training data.

In [1]:
```
# read data
import pandas as pd
df_train = pd.read_csv('attrition_train.csv')

# split training data into features and target
features_train = df_train.iloc[:, 1:]
target_train = df_train.loc[:, 'attrition']

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

Column number	Column name	Type	Description
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
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

With the following cell you train the random forest from the last lesson:

In [2]:
```
# Import, instantiate and fit random forest classifier
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier(n_estimators=100, class_weight='balanced', max_depth
model_rf.fit(features_train, target_train)
```

Out[2]:
```
RandomForestClassifier(class_weight='balanced', max_depth=12, random_state=0)
```

In the last exercise we saw that the `'monthlyincome'` column was recognized as an important column by both the built-in feature importance and the permutation feature importance. However, we weren't able to say **how** `'monthlyincome'` affects the predictions, i.e. whether a low income has a greater impact than a high income, for example. In *Interpreting Decision Trees* we saw a tree that checks for very low incomes in one of its path and for very high incomes in the other. What about in the random forest?

First let's look at the income as a function of `'attrition'`. Use the `stripplot()` function from the `seaborn` module (link to documentation). This function has the following important parameters

```
sns.stripplot(x=str,                # Data on the x-axis
              y=str,                # Data on the y-axis
              data=pd.DataFrame, # Underlying DataFrame
              jitter=float          # Spread of overlapping points
```

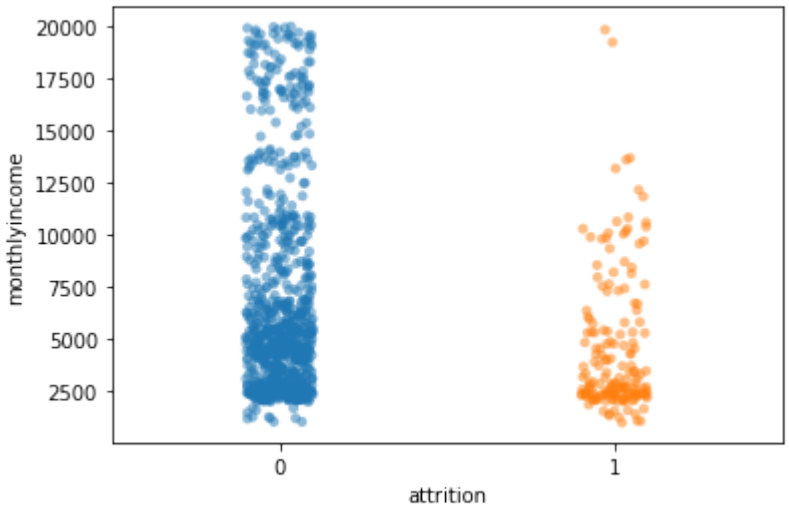
```
                alpha=float)            # Transparency of the points in the
plot
```

It draws scatterplots in strips, where one variable is categorical. This gives us an impression of the income distribution of employees who stay or leave.

Pass `df_train` to the `data` parameter. You should then assign the corresponding column names to the `x` and `y` parameters. Test both variants of `jitter` to see what this parameter does.

In [6]:
```
import seaborn as sns
sns.stripplot(x='attrition', y='monthlyincome', data=df_train, alpha=0.5, jitter=1)
```

Out[6]:
```
<AxesSubplot:xlabel='attrition', ylabel='monthlyincome'>
```



Your image should look something like this (with `jitter=1`).

Strip plot

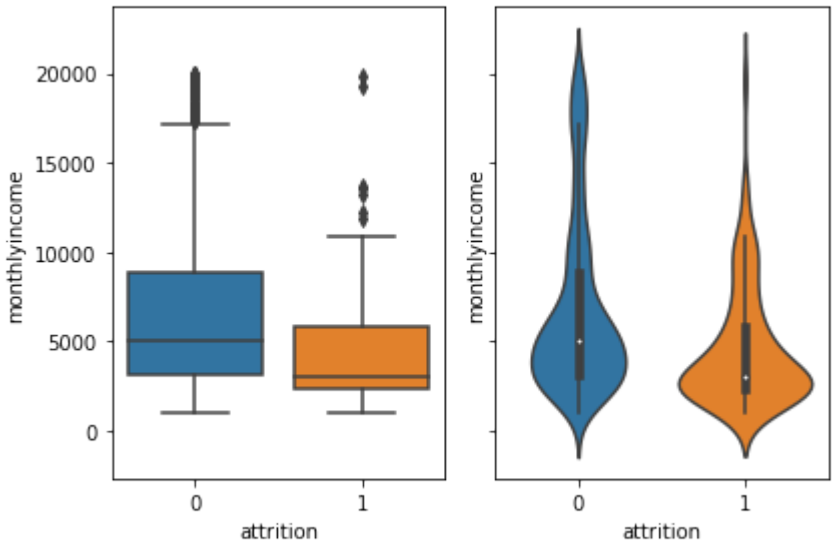
With this illustration we can see where points accumulate. `sns.stripplot()` creates a very intuitive representation to compare the distributions. It's therefore particularly suitable if you want to present the comparison of distributions for people without statistical knowledge. However, it also has the disadvantage that you can't identify much with a lot of data points. Then `sns.boxplot()` and `sns.violinplot()` are better alternatives then. How does the same representation look with them?

Create a visualization with two parts. Display a box plot on the left and a violin plot on the right. You can use the same parameters that you used with `sns.stripplot()` for `sns.boxplot()` and `sns.violinplot()`. In addition, you should assign the relevant axes to the `ax` parameter.

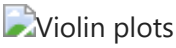
Tip: If the two images overlap, you can call `fig.tight_layout()` at the end to fix this.

In [11]:
```
import matplotlib.pyplot as plt

fig, axs = plt.subplots(ncols=2, sharey=True)
```

```
sns.boxplot(x="attrition", y="monthlyincome", data=df_train, ax=axs[0])
sns.violinplot(x="attrition", y="monthlyincome", data=df_train, ax=axs[1])

fig.tight_layout()
```



Your visualizations could look something like this:


Violin plots

It's no longer clear how many data points there are in the box plot. Instead, it shows additional characteristics of our distribution: The median (line in the box), the upper and lower quartiles (the two sides of the box) and outliers Outliers are defined here as points further away from the box than 1.5 times the length of the box (interquartile range). This distance is expressed as the length of the whiskers. If there are points above or below, they are drawn individually as points. If there are no more data points, the antenna is only drawn up to the last data point.

The violin plot shows the box plot as a small black box with a white median. On the other hand, it shows the density of the data points as a symmetrical width. To determine the density, a normal distribution is applied to each point. These are then added up. Since the normal distribution is not cut off, it can look as if there are still data points in places where there are none. For example, the colored area goes slightly into the negative range on the y-axis because the normal distributions still indicate possible data points here. Another disadvantage of the violin plot is that, like the box plot, it doesn't provide any information about the number of data points. With the `scale='count'` parameter, the width is made dependent on the number of data points at each position (try this out). You can find more settings in the box plot documentation and the Violin plot documentation.

Box plots and violin plots are good ways to compare distributions if the person making the comparison is familiar with these visualizations. For people who have not seen these illustrations and are not very interested in data, it's more difficult than it needs to be to focus on the main message.

All 3 figures show us that people with low income are more likely to leave the company. But there are also some people with a higher income. But what we don't see is what kind of influence certain incomes have on our model. **Partial dependence plots** (PDPs) can help us here. PDPs show the marginal influence a feature has on an average prediction. They can visually represent both linear and non-linear relationships between features and predictions.

To calculate the partial dependency of a selected feature, the value of this feature is replaced by a given value in all data points. Then the average of all predictions is calculated before the next value is tried out.

You can create PDPs with `pdp` from the `pdpbox` module. You can find the module documentation here. Import `pdp`.

In [12]:
```
from pdpbox import pdp
```

To create a PDP, we need 2 functions. The first one is the `pdp.pdp_isolate()` column. It calculates the prediction's partial dependence on the feature.

We need the following parameters:

```
pdp.pdp_isolate(model=model,           # a fitted sklearn model
                dataset=DataFrame,     # the dataset on which the model was
trained
                model_features=list,   # names of all the features the model
uses
                feature=`str`          # feature's column name in `dataset`
               )
```

Define a variable named `pdp_monthlyincome` and calculate the partial dependency for the `'monthlyincome'` feature with `pdp.pdp_isolate()` for `model_rf`.

In [13]:
```
pdp_monthlyincome = pdp.pdp_isolate(model=model_rf,          # a fitted sklearn model
                                    dataset=features_train,    # the dataset on which
                                    model_features=features_train.columns,  # names of
                                    feature='monthlyincome'         # feature's column
                                   )
```

`pdp_monthlyincome` is now a special object. We can visualize it with the second function `pdp.pdp_plot()`.

```
pdp.pdp_plot(pdp_isolate_out=instance of PDPIsolate, # output of
pdp.isolate()
             feature_name=str,                       # name of feature
(for title)
             center=bool,                            # center the plot
             plot_pts_dist=bool                      # display real
feature values
            )
```

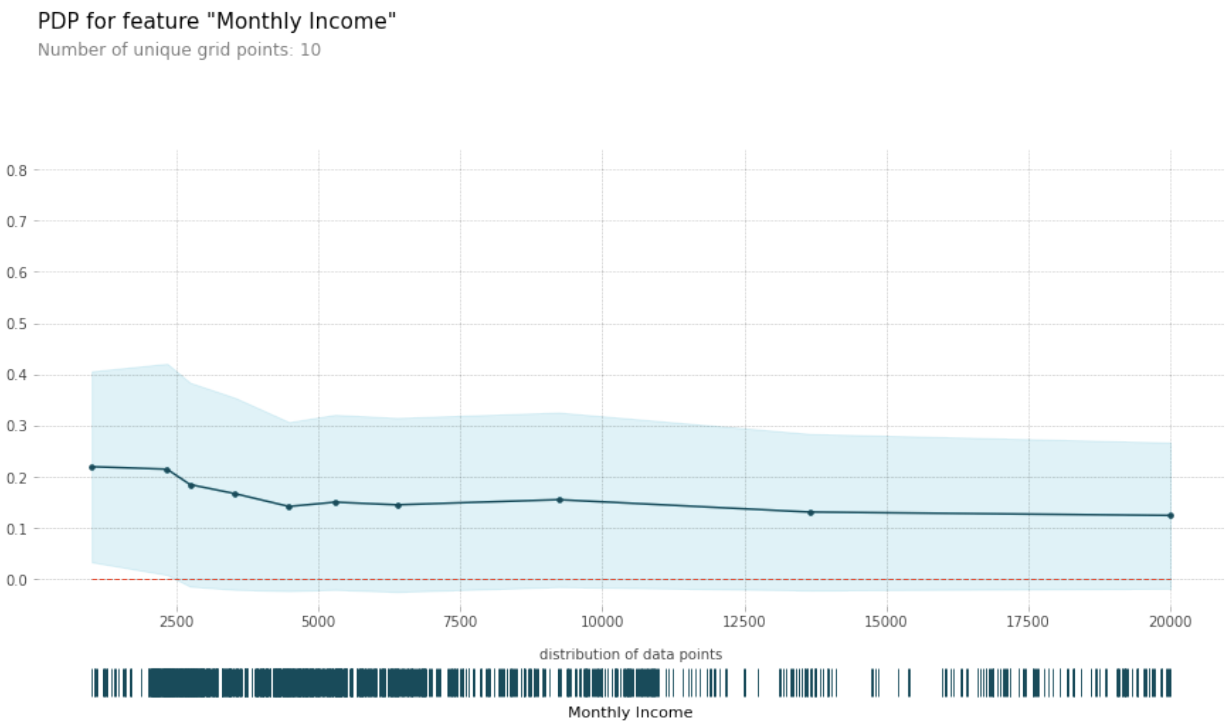
Use `pdp.pdp_plot()`, to display `pdp_monthlyincome`. Use `plot_pts_dist=True`. This means that the real values of `'monthlyincome'` are also displayed. This helps to identify whether the plot is relevant to the real problem at all points. Also specify `center=False`. So the y-axis shows the average forecasts instead of their difference.

If you encounter a warning message `findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.`, you can ignore it. It appears only the first time you create a PDP.

In [14]:
```
pdp.pdp_plot(pdp_isolate_out=pdp_monthlyincome, # output of pdp.isolate()
             feature_name='Monthly Income',     # name of feature (for title)
             center=False,                      # center the plot
             plot_pts_dist=True                 # display real feature values
            )
```

Out[14]:
```
(<Figure size 1080x684 with 3 Axes>,
 {'title_ax': <AxesSubplot:>,
  'pdp_ax': {'_pdp_ax': <AxesSubplot:>,
   '_count_ax': <AxesSubplot:title={'center':'distribution of data points'}, xlabel
='Monthly Income'>}})
```

```
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
```



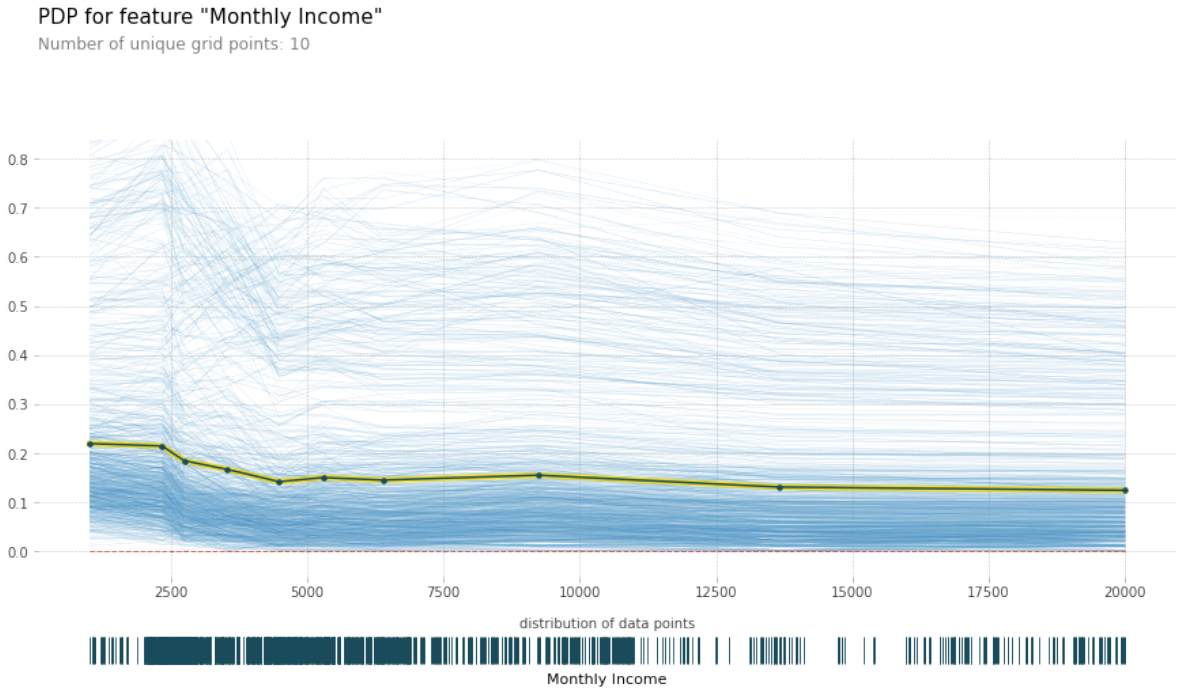
The y-axis shows the average prediction for all data points. The solid blue line shows them for the different values of `'monthlyincome'`. The light blue sleeve shows the standard deviation of the predictions. The blue points represent the values which the partial dependency was actually calculated for. We can define their number in `pdp.pdp_isolate()` with the `num_grid_points` parameter. The standard is 10. The dashes below the figure show the actual values of `'monthlyincome'`.

The PDP shows us that, according to the model, the average probability of leaving the company decreases the more you earn. With a very high income the probability decreases by about 10% to just over 10%. It's also interesting that the probability starts to rise again slightly above €4800.

As we already suspected when comparing the distributions, we can see here that it's mainly employees who don't earn very much who are prepared to leave the company. Unlike distributions, the PDP can show us a link to probability.

Congratulations: You've created your first partial dependence plot with `pdpbox`. Now you've got to know another model diagnostic method for interpreting black-box models. Next, we'll look at ICE plots, which are closely related.

Individual conditional expectation plot

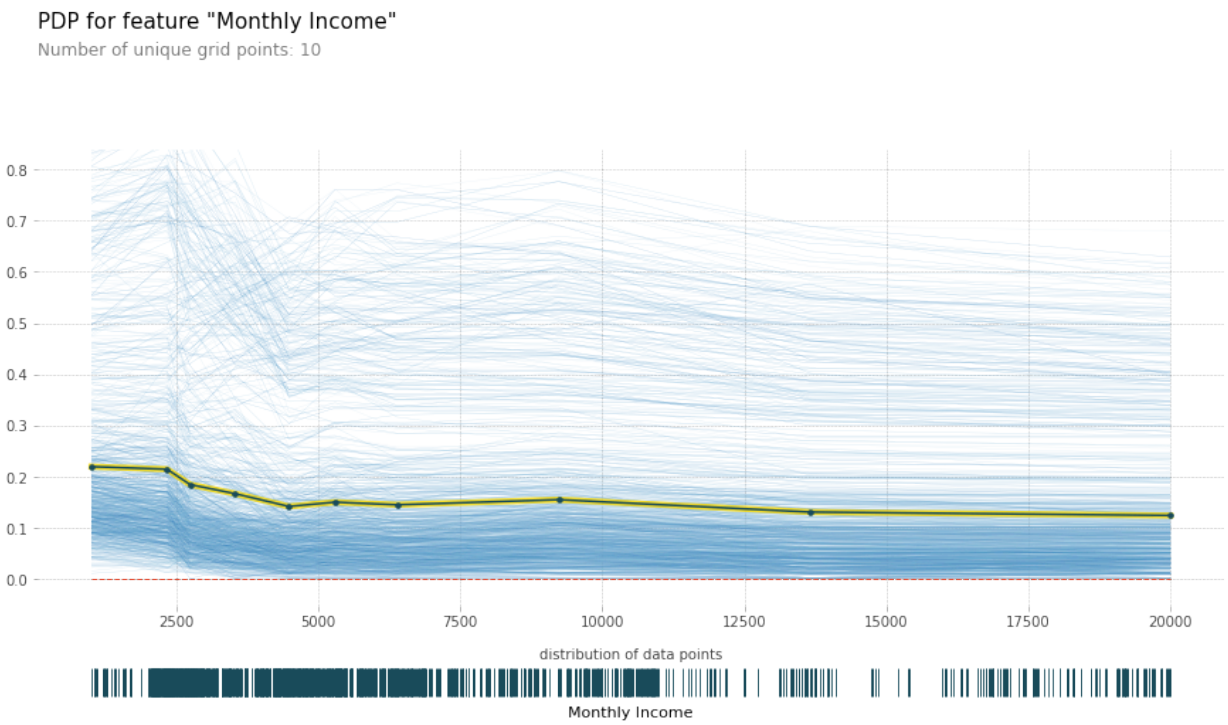


PDPs are a global method because they show the average effect of a feature value on the prediction. The local version of a PDP is the *individual conditional expectation* (ICE) *plot*. An ICE plot visualizes the dependence of the predictions on a feature separately for each data point. This way we get one line per data point instead of one line in total like in the PDP. The PDP corresponds to the average of the ICEs.

We can create an ICE plot with the same function - `pdp.pdp_plot()`. Use the same parameters as before. But this time you have to assign `True` to the new parameter `plot_lines` so that a line is created for each data point.

In [15]:
```
pdp.pdp_plot(pdp_isolate_out=pdp_monthlyincome,
             feature_name="Monthly Income",
             plot_lines=True,
```

```
                plot_pts_dist=True,
                center=False);
```

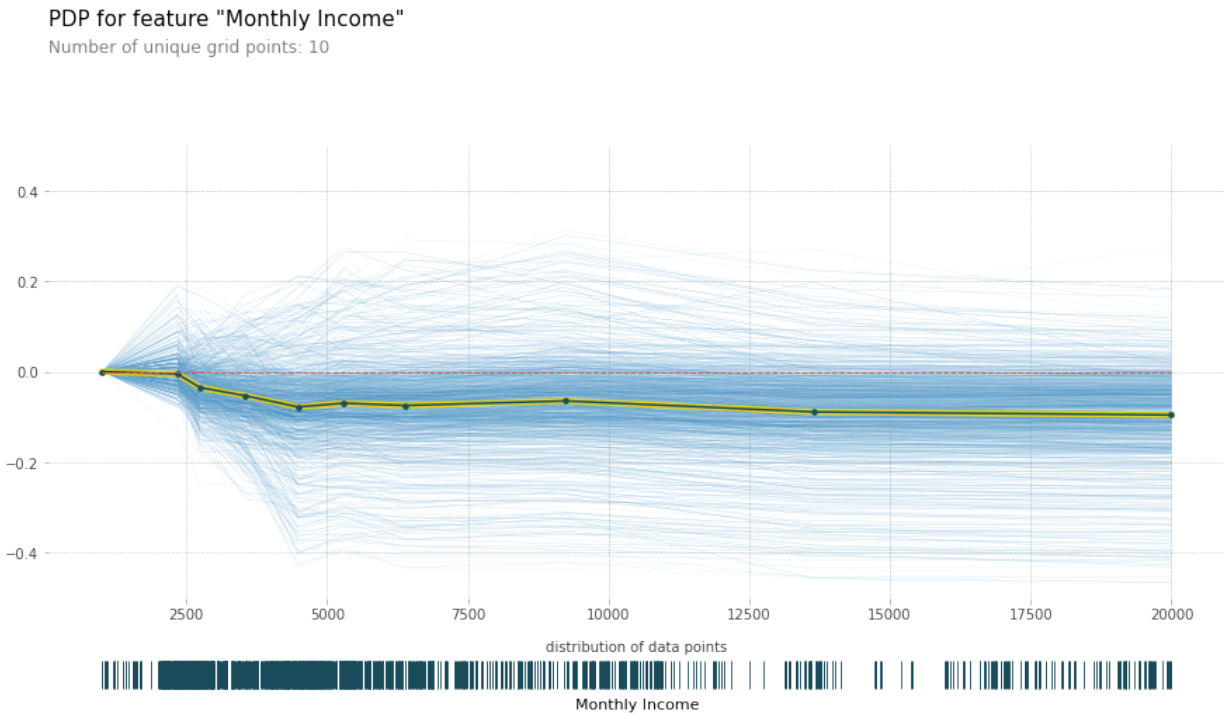


In addition to the average dependency, you will now see numerous thin blue lines. Each one stands for one employee in the data set. How this develops will reveal how a person's potential income would have influenced the probability of them quitting in our model.

With the `center=True` parameter, we can make sure that all lines start at 0. They are simply shifted by the value they have on the y-axis at the beginning. This makes an ICE plot a bit clearer. We would then call this a centered ICE plot. We can also modify the PDP and ICE plot from `pdpbox` itself, since `pdp.pdp_plot()` returns a *figure* object and a `dict` of *axes* objects.

Run the following code cell to create a centered ICE plot and adjust the y-axis limit

In [16]:
```
fig, ax_dict=pdp.pdp_plot(pdp_isolate_out=pdp_monthlyincome,
                feature_name="Monthly Income",
                plot_lines=True,
                plot_pts_dist=True,
                center=True)
ax_dict['pdp_ax']['_pdp_ax'].set_ylim(-0.5,0.5);
```



You can see that income does not have the same effect on all data points, which wasn't clear in the PDP. For a great deal of data points, the probability decreases considerably in the range from €2500 to €5000. These could be people who don't earn so much at the moment. It's also striking that the probability for some people with a salary of just under €2500 increases first before it falls again. Perhaps an income above first €2500 is often accompanied by another job level and people change employer to reach this job level in another company. The lines where the probability increases significantly with a high income are very strange. These could be the people who are already at the top end of the salary range. Or we could have found an error in our model that we would not have found without interpreting the model. To find this out, we could calculate the change in probability with increased income for each data point ourselves. Then we would find out whether a higher income leads to an increased probability of quitting even for low earners. In this case we could consider changing our model, for example by using a smaller depth of the individual trees.

We can also use the idea behind the ICE plot to come up with recommendations for action:

Scenario: The HR department gets in touch with you. They have reason to believe that a certain person is thinking about quitting. However, the department would like to keep this person if person. Run the following cell to import the data for this person.

In [17]:
```
features_aim = pd.read_csv('employee_single.csv', header=None, index_col=0).T
features_aim
```

Out[17]:

	age	gender	businesstravel	distancefromhome	education	joblevel	maritalstatus	monthlyincome
1	33.0	0.0	1.0	5.0	3.0	1.0	0.0	2851.0

According to our model, what is the probability that the person will leave the company soon?

In [18]:
```
model_rf.predict_proba(features_aim)
```

Out[18]:
```
array([[0.16173827, 0.83826173]])
```

Apparently, HR has good reason to be concerned. With our model, we get a probability of over 83% that the person will leave the company soon.

You are told that there are 3 ways to approach the person:

1. A salary increase of up to 20%
2. A promotion accompanied by a salary increase of 10%
3. Hire an additional person in the department so that they no longer need to do so much overtime

What is the most promising approach according to our model?

In [19]:
```
# First scenario:
features_aim_1 = features_aim.copy()
features_aim_1.loc[:, 'monthlyincome'] = features_aim_1.loc[:, 'monthlyincome']  * 1.2
print('20% raise improves chance of not leaving by:', model_rf.predict_proba(features_

# Second scenario:
features_aim_2 = features_aim.copy()
features_aim_2.loc[:, 'monthlyincome'] = features_aim_2.loc[:, 'monthlyincome']  * 1.1
features_aim_2.loc[:, 'joblevel'] = features_aim_2.loc[:, 'joblevel']  + 1
print('10% raise with promotion improves chance of not leaving by:', model_rf.predict_

# Third scenario:
features_aim_3 = features_aim.copy()
features_aim_3.loc[:, 'overtime'] = 0
print('reducing overtime improves chance of not leaving by:', model_rf.predict_proba(f
```

```
20% raise improves chance of not leaving by: [[-0.1684527  0.1684527]]
10% raise with promotion improves chance of not leaving by: [[-0.3065472  0.3065472]]
reducing overtime improves chance of not leaving by: [[-0.41098546  0.41098546]]
```

An increase in salary by itself reduces the probability by about 17%. The small salary increase as part of a promotion makes a difference of about 31%. However, according to our model, hiring another colleague to relieve this person would be the best option with a 45% reduction.

Congratulations: You can now enrich the PDP with the *individual conditional expectations*. This can help you find out how a feature affects individual data points. The HR department thanks you for your assessment! So far we've only looked at how a single feature affects our predictions. Next we'll look at how to create a PDP for 2 features.

PDPs with two features

One disadvantage of PDPs and ICE plots is that they don't take into account the interaction between the features.

For example, what happens when we want to look at joint changes in salary level and income? You can use the `pdp.pdp_interact()` function to do this.

Define a variable named `pdp_income_joblevel` and calculate the partial dependency of the features `'monthlyincome'` and `'joblevel'` with `pdp.pdp_interact()`. Use the same parameters as `pdp.pdp_isolate()` except that `features` is a list of the two features.

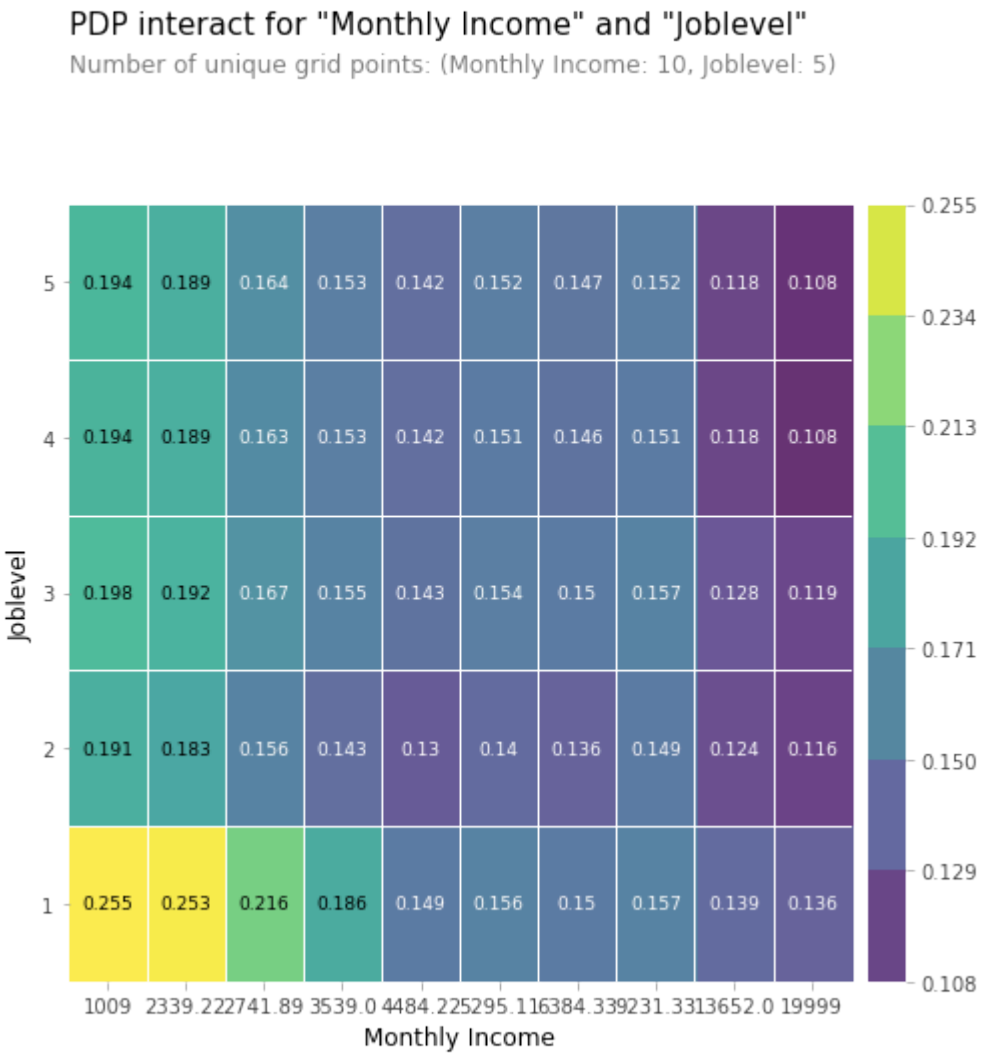
In [20]:
```
pdp_income_joblevel = pdp.pdp_interact(model=model_rf,
                                       dataset=features_train,
                                       model_features=features_train.columns,
                                       features=['monthlyincome', 'joblevel'])
```

Next, we'll pass `pdp_income_joblevel` to the `pdp_interact_plot()` function. It uses the following parameters:

```
pdp.pdp_interact_plot(pdp_interact_out=pdp_interact_out,  # output of
pdp.pdp_interact()
                      feature_names=list,                 # name of the
feature of interest (for plot title)
                      plot_type=str                       # type of
interaction plot
                      )
```

Pass `'grid'` to the `plot_type` parameter (the only alternative is `'contour'`, which is especially good for continuous features. Try it out afterwards if you like).

In [23]:
```
pdp.pdp_interact_plot(pdp_interact_out=pdp_income_joblevel,    # output of pdp.pdp_int
                      feature_names=['Monthly Income', 'Joblevel'], # name of the feature o
                      plot_type='grid'                        # type of interaction p
                      );
```



Now you see a matrix with color-coded values. They reflect the influence of the respective combination of `'monthlyincome'` and `'joblevel'` on the average prediction. If the person's income is very low and their job level is low, the probability of them quitting is highest on average (0.255).

Promotions without a salary increase only reduce the probability slightly. This reveals another downside to PDP plots: They suggest the existence of potentially impossible combinations. No one in the company with a management position will be earning €1000 per month, just as a person with a junior position will not earn €20000 per month. Always remember that a model does not perfectly reflect reality. It can only help us to better understand reality.

You might be thinking that we're running out of dimensions to display PDP plots for more than 2 features. Here we are limited by how we can represent things and the fact that we find it difficult to imagine more than 3 dimensions.

Congratulations: You have now created a PDP that shows the interaction of two variables on the predictions. Now you already know 3 model agnostic methods that help you to interpret your models. Have fun with your new skills!

Remember:

- Model agnostic methods help us interpret black-box models.
- *Partial dependence plots* (PDPs) show the average influence of feature values on the predictions
- *Independent conditional expectation plots* (ICE plots) visualize the influence of feature values for each data point separately
- PDP plots show the average of the ICE plots
- You can create PDPs and ICE plots with the `pdpbox` module.

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.
