WA3546 Lab Guide

ADP Data Program Projects

ascendient learning

Part of Accenture

Revision 1.1.0 published on 2025-06-09.

Ascendient, LLC
1 California Street Suite 2900
San Francisco, CA 94111
https://www.ascendientlearning.com/

USA: 1-877-517-6540, email: getinfousa@ascendientlearning.com
Canada: 1-877-812-8887 toll free, email: getinfo@ascendientlearning.com

Table of Contents

Client Card Number Validation Project

MODULE 1

This project will help you reinforce your knowledge and gain additional insights into the specific aspects of using the NumPy and pandas libraries.

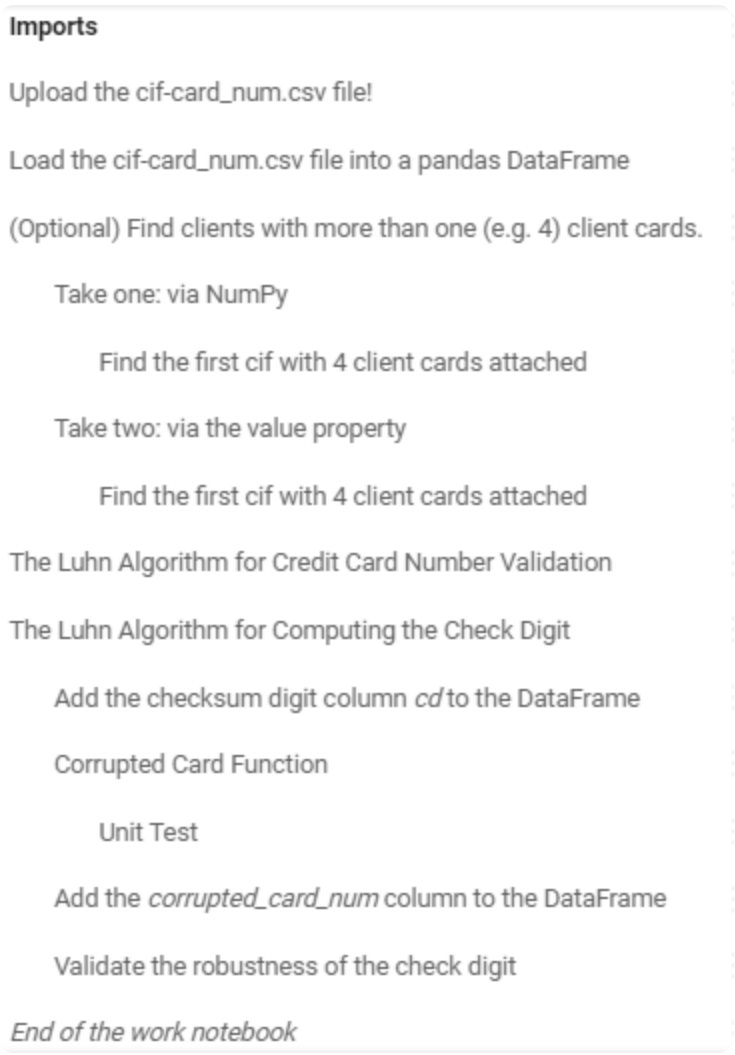
The project is centered around client card number validation, where the client card can be a credit card or some kind of ID.

The main algorithm used in this problem domain is the Luhn ("modulus 10") algorithm, which offers a simple yet sufficiently robust method for validating Luhn-encoded number sequences against data corruption or typing errors.

1.1. The Project Solution File

The solution notebook, **Client_Card_Number_Validation_Project_Solution.ipynb,** is located in the **LabFiles\client-card-validation-project\solutions** directory of the student package.

The ToC of the solution notebook is shown below:



1.2. The Input File

In this project, you will use the **cif-card_num.csv** file located in the LabFiles\client-card-validation-project\financial_data\data\ directory of your student package.

The file acts as a cross reference table between a client information file (CIF) id and client cards associated with it over a many-to-one relationship (a client can have multiple cards).

1.3. Project Instructions

Note: As you progress through the project, you may be asked questions marked **Q:** The answers are given in the project notebook.

1. Upload the input file to your notebook working environment.
2. Load the input file into a DataFrame called

   ```
   df_cif_card
   ```

   - Be aware that the input file does not have the header (the field names record). As a suggestion, assign these column names:
     `['cif_id', 'card_num']`
3. The **head()** DataFrame method then would show these records:

   ```
      cif_id      card_num
   0  10000000    9000000000000000
   1  10000001    9000000000000001
   2  10000002    9000000000000002
   3  10000003    9000000000000003
   4  10000004    9000000000000004
   ```

4. The **info()** method of the DataFrame should report these stats:

   ```
   <class 'pandas.core.frame.DataFrame'>
   RangeIndex: 100000 entries, 0 to 99999
   Data columns (total 2 columns):
    #   Column    Non-Null Count   Dtype
   ---  ------    --------------   -----
    0   cif_id    100000 non-null  int64
    1   card_num  100000 non-null  int64
   dtypes: int64(2)
   memory usage: 1.5 MB
   ```

   Notice the reported memory footprint of the DataFrame.

   Q: What does this command do?

   ```
   ((8 + 8) * df_cif_card.shape[0] / 1024 / 1024)
   ```

5. (Optional activity). Use pandas API and, if necessary, NumPy, to find CIFs with 4 client cards.

Hint: Use the **value_counts()** method.

There should be 17 such CIFs:

```
[10039552, 10086449, 10060898, 10005568, 10042682, 10030298, 10088053, 10056012,
10058648, 10006117, 10028230, 10007599, 10024468, 10087128, 10071590, 10076536,
10042754]

        cif_id      card_num
41671   10039552    9000000000041671
41672   10039552    9000000000041672
41673   10039552    9000000000041673
41674   10039552    9000000000041674
```

6. Write a *luhn_checksum_digit* generation function using the algorithm described here.

   The function should return a control checksum digit, e.g. *4* for the number *1789372997* as per the Wikipedia example.

   To help you with your coding exercise, below is a canonical example of the Luhn-based credit card number validation Python function that you can adapt to your needs:

```python
def luhn_check(cc_number):
    def get_digits_of(n):
        return [int(d) for d in str(n)]

    digits = get_digits_of(cc_number)

    odd_digits = digits[-1::-2]
    even_digits = digits[-2::-2]

    checksum = 0
    checksum += sum(odd_digits)

    for d in even_digits:
        checksum += sum(get_digits_of(2 * d))

    return checksum % 10 == 0

# Test
cc_number = 4532015112830366
yes_no = 'does not pass'
```

```
if luhn_check(cc_number):
    yes_no = 'passes'
```

7. Add the checksum digit column '*cd*' to the DataFrame using this command:

```
df_cif_card['cd'] = df_cif_card['card_num'].apply(luhn_checksum_digit)
```

   In many practical cases, the checksum digit is added at the end of the number from which it has been calculated.

   Now let's see if the method can withstand a single-digit corruption due to data entry or scanning error.

8. Write a function that simulates a *single* digit number corruption using the skeleton code below (the missing pieces you need to plug in are shown as **?**).

```
import random
random.seed(128)

def corrupt_card_number(card_number):
    card_number = ?(card_number)
    len_cn = len(str(card_number))
    pos = random.randint(0, len_cn - 1)
    digit = random.randint(?, ?)
    corrupt_cn = int(card_number[:pos] + ? + card_number[pos + 1:])
    return corrupt_cn
```

   The above function can generate a corrupted digit that equals the original digit at the same position making the number unchanged (and valid).

9. Add a new '*corrupted_card_num*' column to the DataFrame using this code:

```
random.seed(128)

df_cif_card['corrupted_card_num'] =
df_cif_card['card_num'].apply(corrupt_card_number)
```

   The DataFrame should now have the following header and trailing records:

```
      cif_id    card_num           cd   corrupted_card_num
0     10000000  9000000000000000   1    9000000600000000
1     10000001  9000000000000001   9    9000000000020001
2     10000002  9000000000000002   7    9000000000000000
3     10000003  9000000000000003   5    9000500000000003
4     10000004  9000000000000004   3    9000000090000004
```

```
...    ...  ...  ...  ...
99995 10094959  9000000000099995   4    9000000000094995
99996 10094960  9000000000099996   2    9000300000099996
99997 10094961  9000000000099997   10   9000000000099897
99998 10094962  9000000000099998   8    6000000000099998
99999 10094963  9000000000099999   6    9000000000499999
100000 rows x 4 columns
```

10. Use the function on a portion of the DataFrame records affected by "*true*" corruptions (that altered the card number).

11. Write code to verify the robustness of the Luhn-based algorithm by ascertaining that the checksum digit generated from the original card number and the one from the corrupted number are different.

    > ⓘ **Note**
    >
    > You may want to use these NumPy functions:
    > np.where()
    > np.vectorize()

    Did you expect the method to perform much worse?

1.4. End of Project

Data Preprocessing Project

MODULE 2

In this project, you will work through some of the tasks you can encounter in financial data processing environments. While most of the activities in the project are centered around pandas and NumPy APIs, these activities are generic and can be carried out using other data pre-processing tools.

2.1. The Project Solution File

The solution notebook, **Data_Preprocessing_Project_Solution.ipynb,** is located in the **LabFiles\data-preprocessing-project\solutions** directory of the student package.

To give you an idea of the project activities, here is the ToC of the solution notebook:

2.2. The Input File

In this project, you will use the **txn_2022-03-15.csv** file zipped into the *txn_2022-03-15.zip* archive that is located in the *LabFiles\data\financial_data* directory of your student package.

The input file contains a transaction history of a day's worth of mock client transactions conducted at a fictitious financial organization.

The transaction history file has the following fields:

- **CARD_NUM**: A 16-digit client card number
- **TIMESTAMP**: A date-time field with a millisecond precision
- **TXN_TYPE**: Transaction type: [OPEN_ACCNT | TFSA_OPEN | CLOSE_ACCNT | ADD_MERCHANT | DEL_MERCHANT | BILL_PAY | TRANSFER]
- **DC**: Transaction's financial designation: [DEBIT | CREDIT | N/A]
- **AMOUNT:** Transaction amount: [AMOUNT | N/A]
- **STATUS:** Transaction status: [SUCCESS | FAIL]

A client, identified by their client card, may have more than one transaction recorded in the history file.

A snippet of transactions recorded in the file is shown below:

```
9000000000012571,2022-03-15 9:25:44.637,SUCCESS,TRANSFER,CREDIT,1247.59
9000000000095770,2022-03-15 9:25:44.789,SUCCESS,TRANSFER,DEBIT,411.62
9000000000081696,2022-03-15 9:26:13.307,SUCCESS,OPEN_ACCNT,N/A,N/A
9000000000079072,2022-03-15 9:26:24.434,SUCCESS,ADD_MERCHANT,N/A,N/A
9000000000043518,2022-03-15 21:9:21.894,FAIL,BILL_PAY,DEBIT,992.35
9000000000058017,2022-03-15 23:20:54.261,SUCCESS,CLOSE_ACCNT,N/A,N/A
```

2.3. Project Instructions

> ⓘ Note

> As you progress through the project, you may be occasionally asked questions marked "**Q:**" (without quotes); the answers are given in the project notebook.

1. Upload the zip archive file to your notebook working environment and unzip it using this command:

```
!unzip txn_2022-03-15.zip
```

That would extract the input *txn_2022-03-15.csv* file.

2. Load the input file into a pandas DataFrame.

The **head()** method would show these records:

	CARD_NUM	TIMESTAMP	TXN_TYPE	DC	AMOUNT	STATUS
0	9000000000048093	2022-03-15 0:0:1.115	BILL_PAY	DEBIT	414.20	SUCCESS
1	9000000000089447	2022-03-15 0:0:1.181	BILL_PAY	DEBIT	505.31	SUCCESS
2	9000000000087039	2022-03-15 0:0:1.322	BILL_PAY	DEBIT	796.18	SUCCESS
3	9000000000014801	2022-03-15 0:0:1.330	BILL_PAY	DEBIT	80.83	SUCCESS
4	9000000000002550	2022-03-15 0:0:1.351	BILL_PAY	DEBIT	843.09	SUCCESS

The **info()** method called on the target DataFrame would return these stats:

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1036788 entries, 2022-03-15 00:00:01.115000 to 2022-03-15
23:59:59.922000
Data columns (total 5 columns):
 #   Column    Non-Null Count    Dtype
---  ------    --------------    -----
 0   CARD_NUM  1036788 non-null  int64
 1   TXN_TYPE  1036788 non-null  object
 2   DC        1036788 non-null  object
 3   AMOUNT    1014076 non-null  float64
 4   STATUS    1036788 non-null  object
dtypes: float64(1), int64(1), object(3)
memory usage: 79.7+ MB
```

Q: How many Null (NaN) values are there in the *DC* column?

Q: What is the type of the *TIMESTAMP* field?

3. Count the number of duplicate values in the *TIMESTAMP* field.

You should get:

```
5663
```

which indicates that, despite the millisecond resolution, the logging system of the financial transaction system still allows duplicates. This can suggest that the logging is performed using multiple threads with no global serialization lock. In some practical situations, this may pose a problem (e.g. for creating a a primary key or a unique constraint in a SQL table using these values). In our case, we are not constrained by this requirement and can go ahead and use the TIMESTAMP as the index of our DataFrame.

4. Find transactions with duplicate TIMESTAMP values.

   Hint: You may want to use the *duplicated()* DataFrame method to this end.

   Next, let's convert the TIMESTAMP into a *datetime* type to leverage the capability of this temporal type.

5. Convert the TIMESTAMP of type dtype('O') into a *datetime* type.

   Hint: Use the *pd.to_datetime()* method.

   Now the info() method would report this TIMESTAMP type:

   ```
   1 TIMESTAMP 1036788 non-null datetime64[ns]
   ```

   If you perform this time range operation (captured in the *Timedelta* temporal type) using the implicit *iloc()* indexing:

   ```
   df.TIMESTAMP [1050] - DataFrame.TIMESTAMP [1000]
   ```

   you would see the following result:

   ```
   Timedelta('0 days 00:00:04.062000')
   ```

   You essentially found the time delta between

   ```
   2022-03-15 00:01:28.456
   ```

   and

   ```
   2022-03-15 00:01:24.488
   ```


Now, let's use the TIMESTAMP as the DataFrame's DatetimeIndex (instead of the default auto-incremented RangeIndex)

You may want to make a copy of the original DataFrame object before proceeding and work on the copy keeping the original DataFrame around just in case.

6. Use this command:

```
<your new df>.index = <your df>.TIMESTAMP
```

7. Drop (in place) the TIMESTAMP column in *<your new df>*

   The DataFrame has some NaN values; let's see what we can do with those.

8. Select all the records that have 'OPEN_ACCNT' in the TXN_TYPE column.

   You should see the following output:

```
                        CARD_NUM         TXN_TYPE   DC  AMOUNT  STATUS
2022-03-15 00:01:25.901 9000000000061099 OPEN_ACCNT NaN NaN     SUCCESS
2022-03-15 00:02:21.584 9000000000072993 OPEN_ACCNT NaN NaN     SUCCESS
2022-03-15 00:02:53.527 9000000000031268 OPEN_ACCNT NaN NaN     SUCCESS
2022-03-15 00:03:29.669 9000000000073566 OPEN_ACCNT NaN NaN     SUCCESS
2022-03-15 00:03:49.524 9000000000038553 OPEN_ACCNT NaN NaN     SUCCESS
... ... ... ... ... ...
2022-03-15 23:57:06.138 9000000000049464 OPEN_ACCNT NaN NaN     SUCCESS
2022-03-15 23:57:33.658 9000000000099624 OPEN_ACCNT NaN NaN     SUCCESS
2022-03-15 23:58:16.396 9000000000001182 OPEN_ACCNT NaN NaN     SUCCESS
2022-03-15 23:59:44.997 9000000000001743 OPEN_ACCNT NaN NaN     SUCCESS
2022-03-15 23:59:48.128 9000000000018831 OPEN_ACCNT NaN NaN     SUCCESS
```

   As you can see, both the DC and AMOUNT fields have NaN's; in the input file, there are corresponding empty field values, e.g.:

```
9000000000031268,2022-03-15 0:2:53.527,OPEN_ACCNT,,,SUCCESS
```

9. Replace in place NaN values in the *DC* column with the '**NA**' string.

10. Replace in place NaN values in the *AMOUNT* column with the **0.0** float value.

11. Find the difference between the CREDIT and DEBIT amounts for the customer with the client card *9000000000031268*.

You should get the following value:

```
-2439.96
```

Finally, let's adapt the *AMOUNT* filed such that we can calculate the cumulative sum.

12. Create a function that takes a DataFrame record and makes the amount in the *AMOUNT* field negative for *DEBIT* transactions.

    The function's skeleton code is shown below for your reference (plug in your code in place of the '?' placeholders):

```
def deb_cred (record):
    #print (record)
    if record['DC'] == '?':
        record['?'] = - record['?']
    return record
```

13. Apply this function to the DataFrame via the *.apply()* method.

> ⓘ Note
>
> Running this method across all the DataFrame's records may take about 1.5 minutes.

Q: How many records per second does pandas process using the *apply* method, which is similar to a sequential cursor-oriented processing in relational databases?

14. Select all the transactions done with the *9000000000031268* client card as a DataFrame object.

    You should see the following output:

```
                        CARD_NUM         TXN_TYPE   DC      AMOUNT  STATUS
TIMESTAMP
2022-03-15 00:02:53.527 9000000000031268 OPEN_ACCNT NA      0.00    SUCCESS
2022-03-15 02:00:07.867 9000000000031268 BILL_PAY   DEBIT   -525.86 SUCCESS
2022-03-15 03:45:53.339 9000000000031268 TRANSFER   CREDIT  125.75  SUCCESS
2022-03-15 09:51:46.558 9000000000031268 TRANSFER   CREDIT  677.79  SUCCESS
2022-03-15 14:25:26.692 9000000000031268 BILL_PAY   DEBIT   -765.06 SUCCESS
2022-03-15 17:01:06.343 9000000000031268 BILL_PAY   DEBIT   -588.64 SUCCESS
```

```
2022-03-15 20:48:17.722 9000000000031268 BILL_PAY   DEBIT   -766.34 SUCCESS
2022-03-15 22:30:29.556 9000000000031268 BILL_PAY   DEBIT   -597.60 SUCCESS
```

Notice the negative amounts for DEBIT transactions.

15. Apply the *cumsum*() function to the AMOUNT fields of the DataFrame you created above for the *9000000000031268* client card.

    You should see the following output:

```
TIMESTAMP
2022-03-15 00:02:53.527     0.00
2022-03-15 02:00:07.867     -525.86
2022-03-15 03:45:53.339     -400.11
2022-03-15 09:51:46.558     277.68
2022-03-15 14:25:26.692     -487.38
2022-03-15 17:01:06.343     -1076.02
2022-03-15 20:48:17.722     -1842.36
2022-03-15 22:30:29.556     -2439.96
Name: AMOUNT, dtype: float64
```

    Which should correspond to the result you obtained in this previous instruction:

    Find the difference between the DEBIT amounts and CREDIT amounts for the customer with client card 9000000000031268.

2.4. End of Project

Exploratory Data Analysis Project

MODULE 3

In this project, you will use Exploratory Data Analysis (EDA) techniques to identify and analyze factors influencing employee attrition. This problem is part of the broader Human Capital Management (HCM) problem domain, which is a set of practices and strategies used by organizations to manage and optimize their workforce effectively.

Mainly, you will use the *seaborn* data visualization library API; in other cases, you will be instructed accordingly.

For input data, we will use the IBM HR Analytics Employee Attrition & Performance dataset . This is a dataset created at IBM to model attrition of the workforce. The dataset includes information on employee satisfaction, compensation, job seniority, tenure, and demographics, for a total of 35 attributes. For additional information about the dataset, visit this resource.

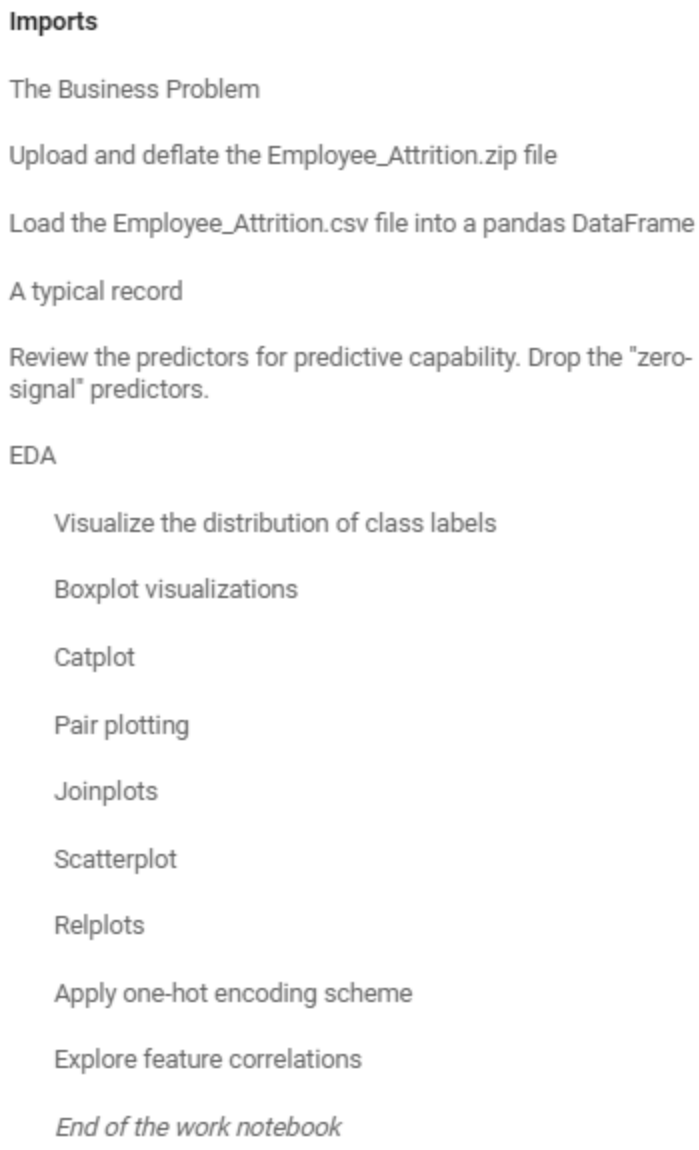
In this guide, we will use terms *column, feature, attribute,* and *variable* interchangeably.

> ⓘ Note
>
> As you progress through the project, you may be occasionally asked questions marked "**Q:**" (without quotes); the answers are given either in the project notebook or at the end of this project guide.

3.1. The Project Solution File

The solution notebook, **EDA_Project_Solution.ipynb,** is located in the **LabFiles\eda-project\solutions** directory of the student package.

The ToC of the solution notebook is shown below:



3.2. The Input File

The **Employee_Attrition.csv** file you will use in this project is zipped in the *Employee_Attrition.zip* archive that is located in the *LabFiles\eda-project\data\HCM* directory of your student package.

The file has 1470 records with 35 attributes.

A fragment of the file's content is shown below:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical

3.3. Load the Data

1. Upload the archive file to your notebook working environment and unzip it.
2. Load the *Employee_Attrition.csv* input file into a pandas DataFrame named **df**.
3. Call the DataFrame's *info()* method, you should see this output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
```

```
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

The first record is shown below for your reference:

```
Age                                41
Attrition                         Yes
BusinessTravel          Travel_Rarely
DailyRate                       1102
Department                     Sales
DistanceFromHome                   1
Education                          2
EducationField          Life Sciences
EmployeeCount                      1
EmployeeNumber                     1
EnvironmentSatisfaction            2
Gender                        Female
HourlyRate                        94
JobInvolvement                     3
JobLevel                           2
JobRole                Sales Executive
JobSatisfaction                    4
MaritalStatus                 Single
MonthlyIncome                   5993
MonthlyRate                    19479
NumCompaniesWorked                 8
Over18                             Y
OverTime                         Yes
```

```
PercentSalaryHike           11
PerformanceRating            3
RelationshipSatisfaction     1
StandardHours               80
StockOptionLevel             0
TotalWorkingYears            8
TrainingTimesLastYear        0
WorkLifeBalance              1
YearsAtCompany               6
YearsInCurrentRole           4
YearsSinceLastPromotion      0
YearsWithCurrManager         5
Name: 0, dtype: object
```

4. Identify columns with a single value. These are non-informative variables you may want to drop to simplify your analysis.

   The candidate columns are

   ```
   'EmployeeCount', 'Over18', 'StandardHours'
   ```

   Q: What can be said about the *EmployeeNumber* column?

   Heed the advice in the answer to this question in the **Answers** section at the end of this guide.

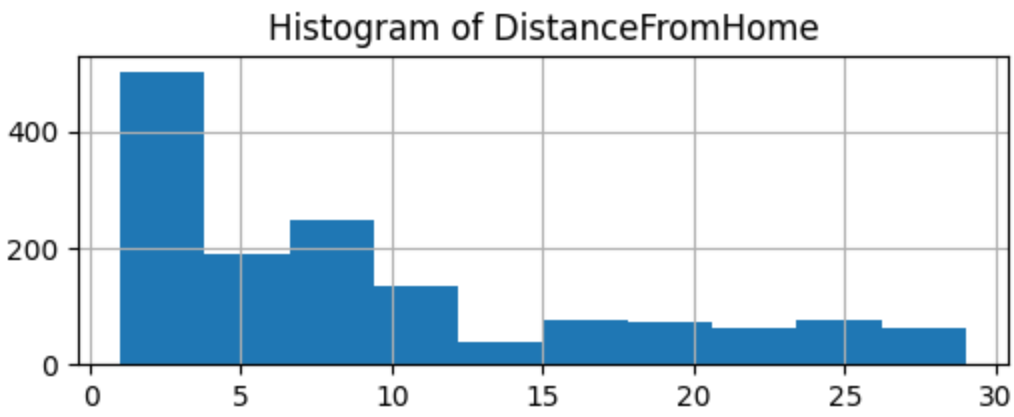
   > ⓘ **Note**
   >
   > The shape of your DataFrame after removing unnecessary for EDA columns should be (1470, 31).

3.4. Use pandas' Data Visualization API

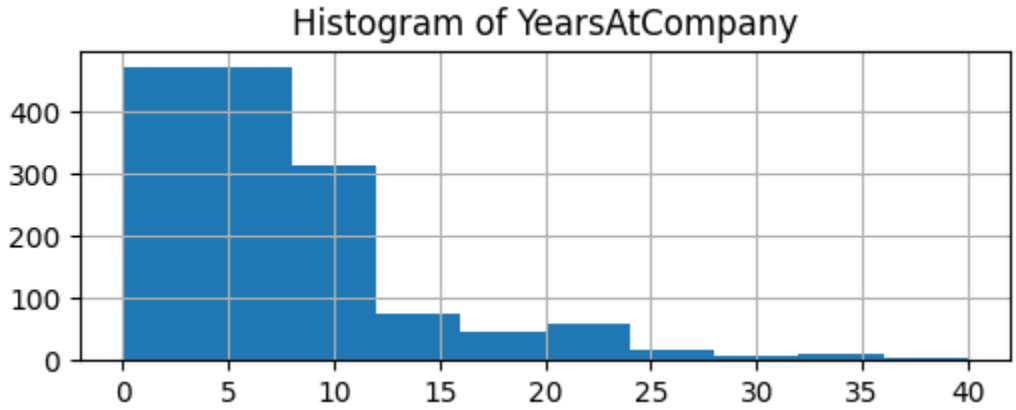
Now we can start our EDA activities.

1. Use the **hist()** DataFrame method to create a plot for *DistanceFromHome*:

   You should see the following output:



2. Repeat the above command for the *YearsAtCompany* feature:



Hint: Use the Series object's *name* attribute to print the histogram's title.

The **hist()** method can also render the categorical variables.

Feel free to explore other DataFrame columns like BusinessTravel

3.5. Analyze the Attrition Feature

1. Enter the following command to set the *seaborn* style:

```
sns.set_style("whitegrid")
```

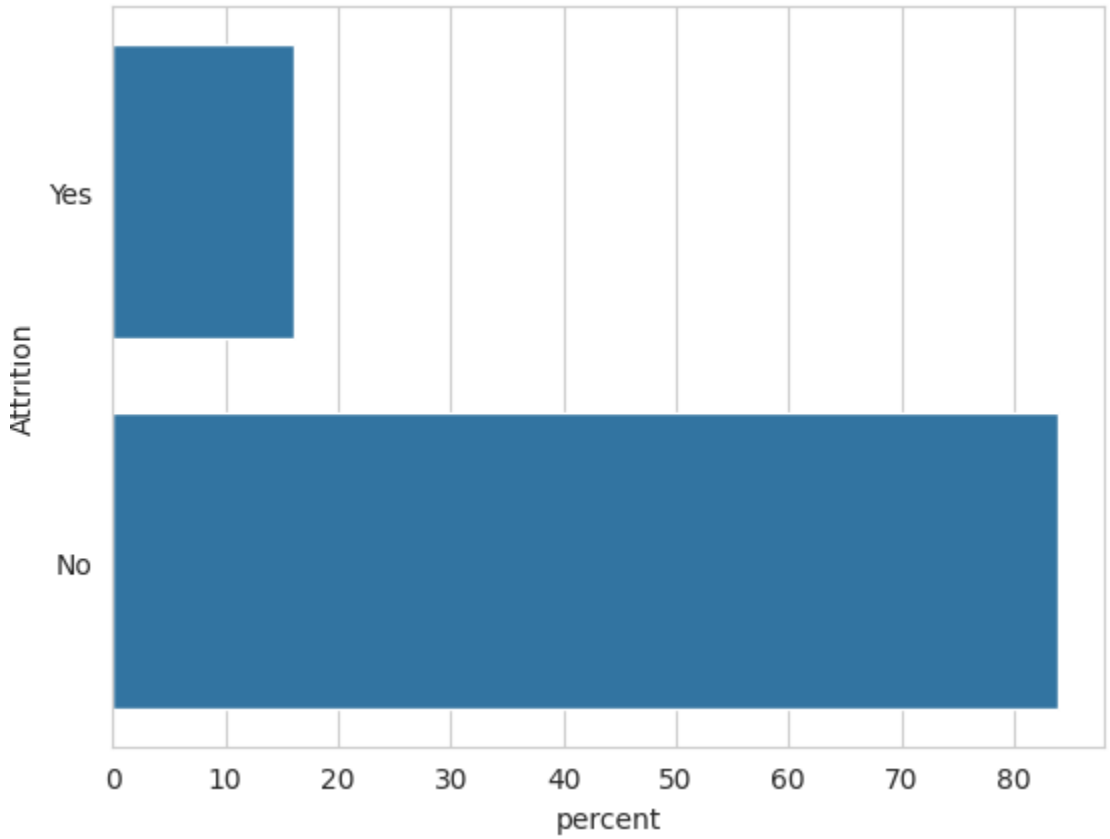
2. Enter the following command:

```
sns.countplot(df.Attrition, stat='percent')
```

To display the percentage of records in both attrition categories as histograms.

> ⓘ **Note**
>
> A count plot in *seaborn* can be thought of as a histogram across a categorical instead of quantitative variable. The plot orientation is determined automatically by the the data.

You should see the following plot:

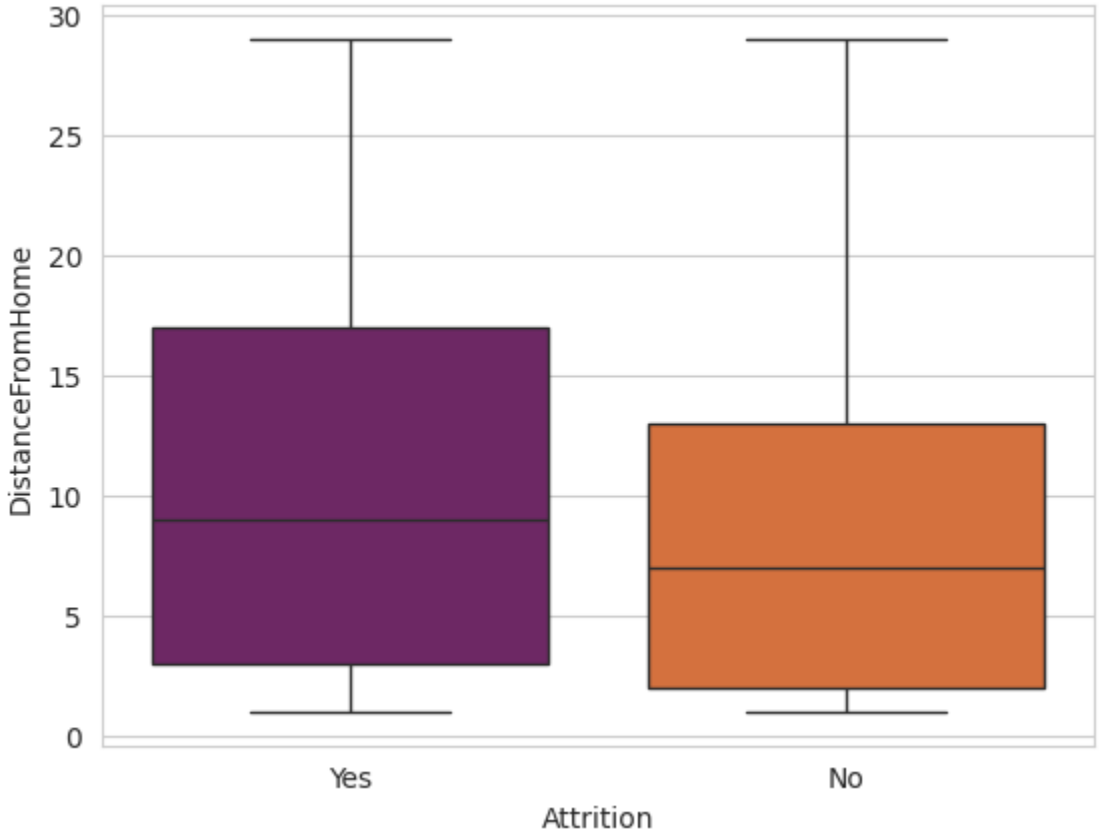


As you can see, there is about a 1:4 ratio of the data for the *Attrition* feature.

3.6. Boxplots

Now let's see what seaborn's graphing capabilities can do for us.

1. Create a boxplot that looks as follows:



> ⓘ Note
>
> Use the *infreno* palette.

Q: Does the *DistanceFromHome* variable influences decisions to leave the company?

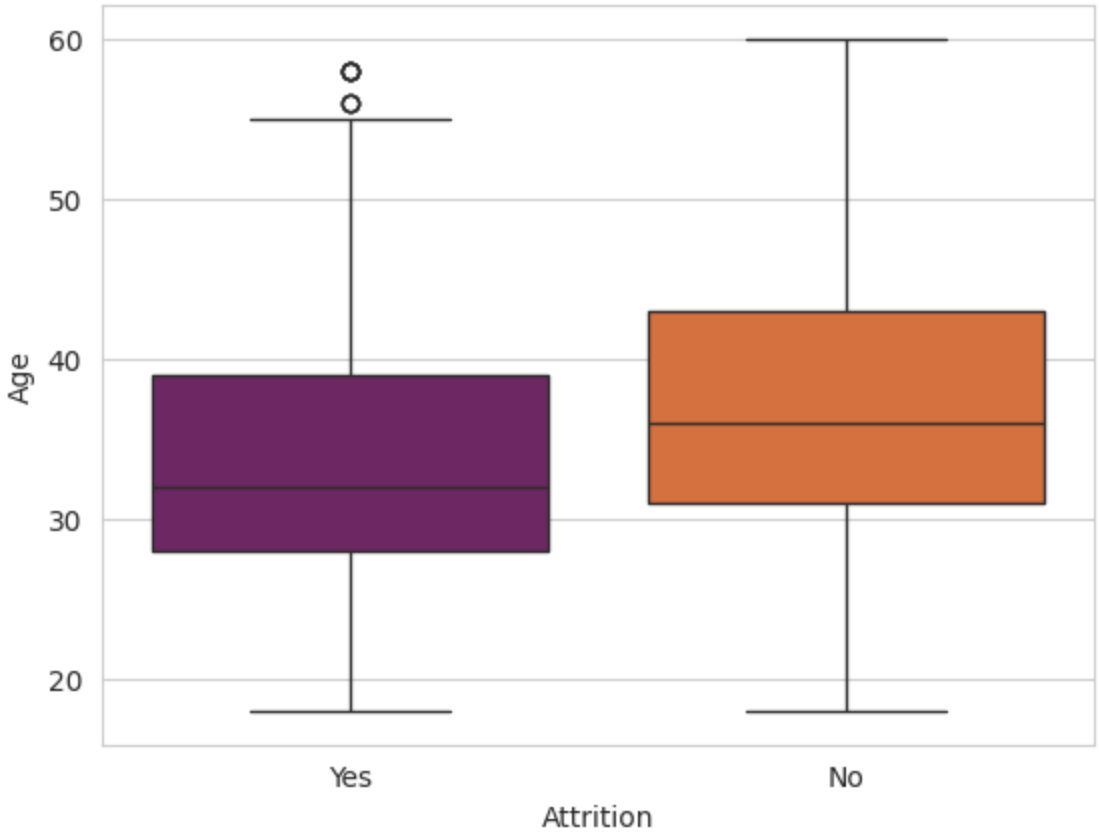
Q: What is the median of *DistanceFromHome* for each *Attrition* category?

Use this skeleton code for your reference:

```
df[df.Attrition == '?']['?'].median()
```

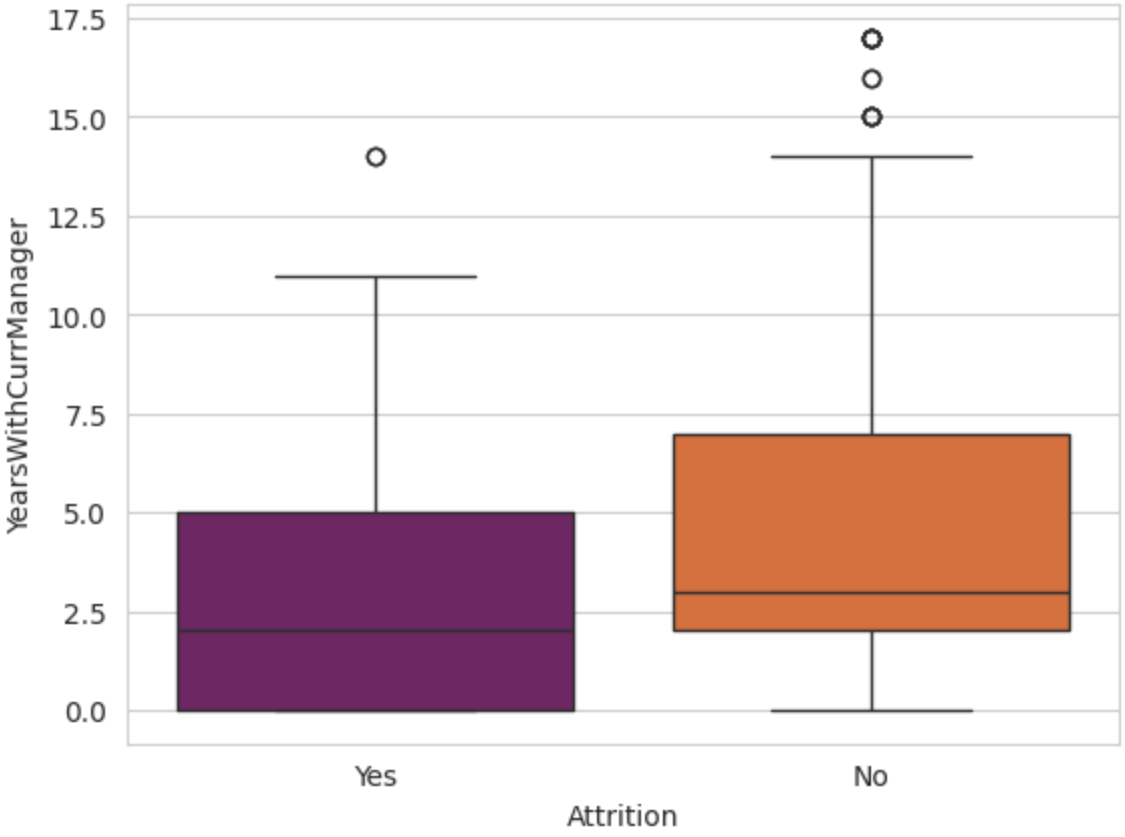
2. Use a boxplot-based visualization to corroborate that younger employees are more likely to leave the company.

   You can use this boxplot, for example:



   Is there evidence that employees tend to stick with their managers (provided that the managers provide genuine leadership and possess great “soft skills” like EQ and conflict resolution) and stay longer with the company?

   Would the boxplot below answer this question?

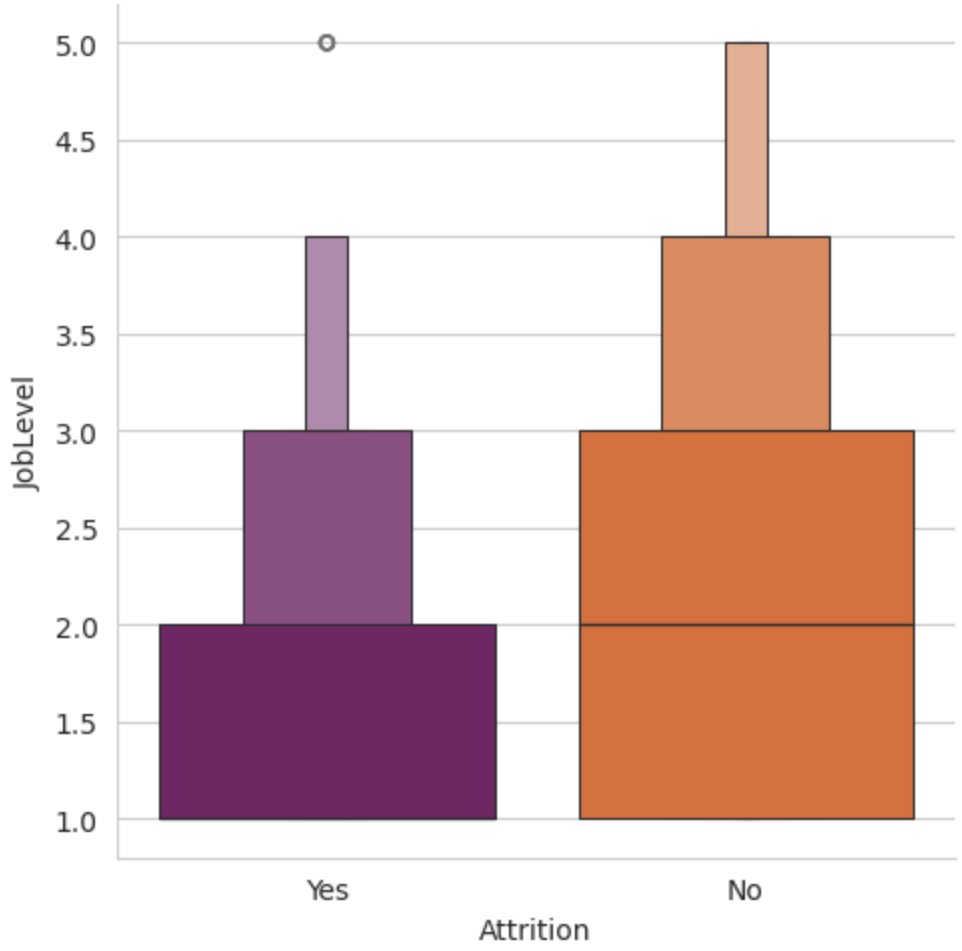


3.7. Catplots

Now let's see how the seaborn.catplot function can aid us in our EDA activities.

> ⓘ **Note**
>
> Catplots (categorical plots) in seaborn offer more diverse functionality compared to boxplots, including support for multi-plot grids to enable comparisons across different subset of the same data, as well as a wide range of plot types, such as *box, bar, strip, swarm, point,* and *violin* that are available via the *kind* parameter.

Q: How would you interpret the output of this catplot:

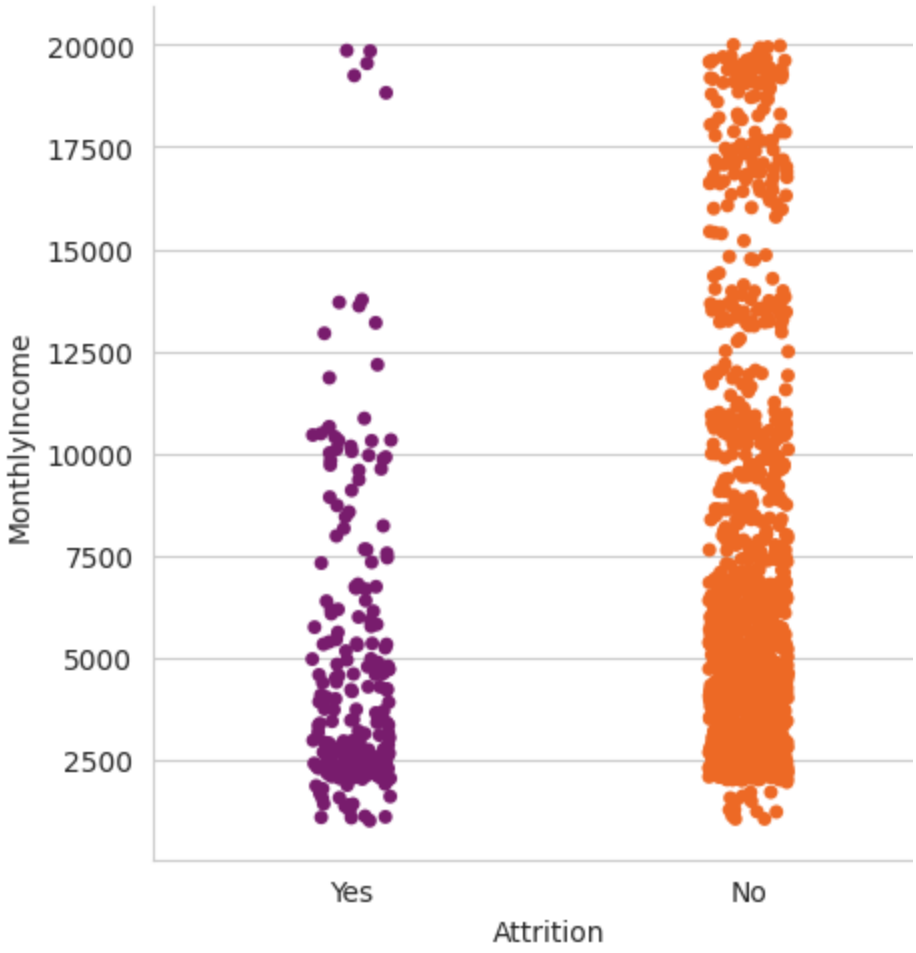


created with the following command:

```
sns.catplot (x=df.Attrition, y=df.JobLevel, palette='inferno', kind='boxen')
```

> ⓘ Note
>
> Answering this question is left as an exercise to the student.

Some useful information can be gleaned from the catplot below, e.g. indicating that there is a risk of losing senior employees in the high-income bracket.



3.8. Pairplots

In this section, we will explore the seaborn.pairplot function's capability to capture pairwise relationships in a dataset.

In its present form, the input dataset has three main attributes that refer to the employee's compensation/wages:

```
'MonthlyIncome', 'MonthlyRate', 'HourlyRate', and 'DailyRate'
```

which seem to be somehow not directly correlated. It may be due to the fact that some include bonuses, income from stock options, working overtime, and other forms of financial incentivization.

Let's see if we can visually see what relationships, if any, exist among those attributes.

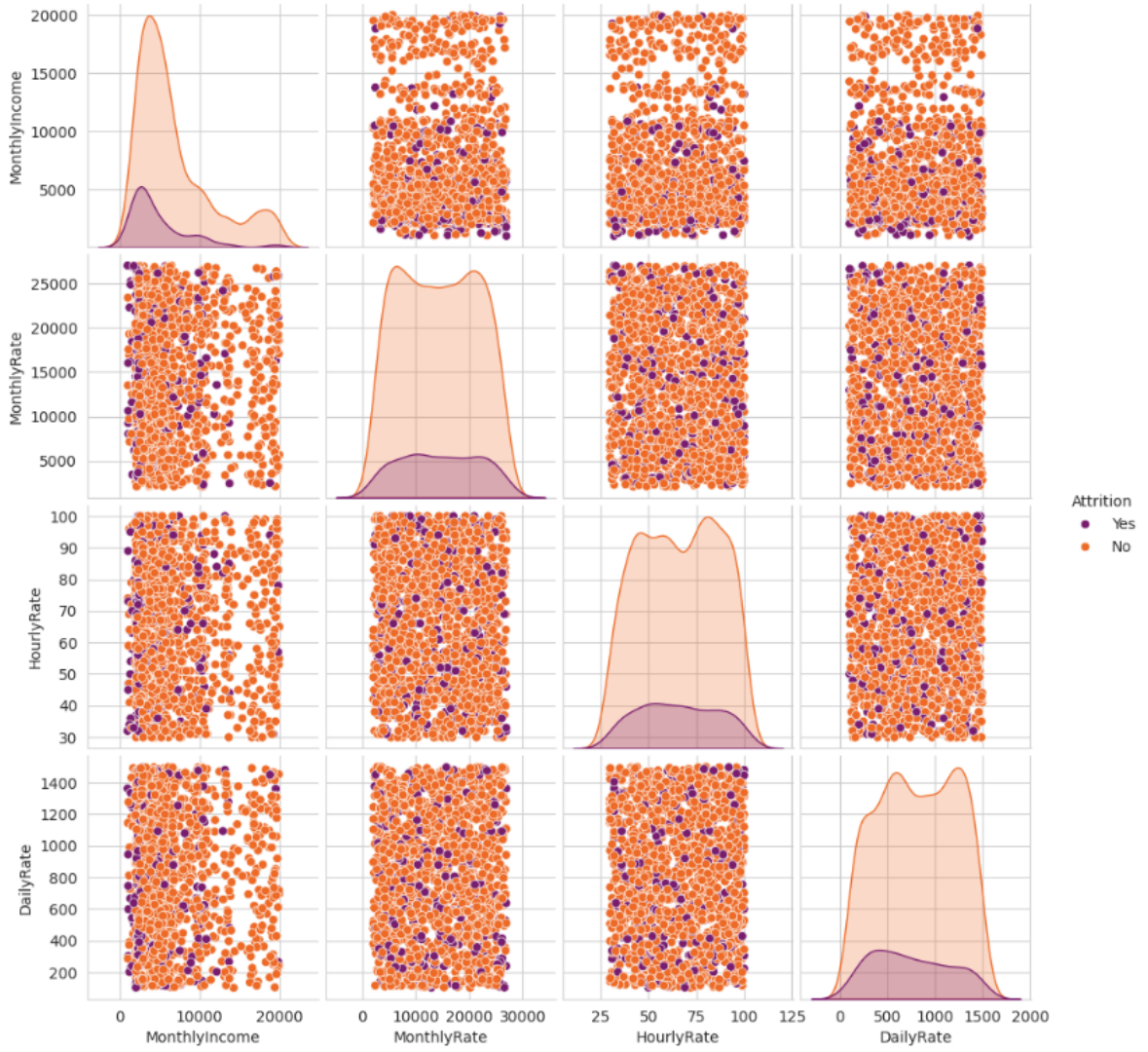
Below are the steps supplemented with some code you should follow to get started.

- Create a DataFrame with those compensation-related columns, plus the *Attrition* column.

```
df_comp = df[['MonthlyIncome', 'MonthlyRate', 'HourlyRate', 'DailyRate',
'Attrition']]
```

- Create a *pairplot* function using *Attrition* as the *hue* parameter.
- Use the *palette='inferno'* parameter setting.

  You should get the following pair plot:



> ⓘ **Note**
>
> *Seaborn* makes the *Attrition* column part of the plots' visualization configuration.

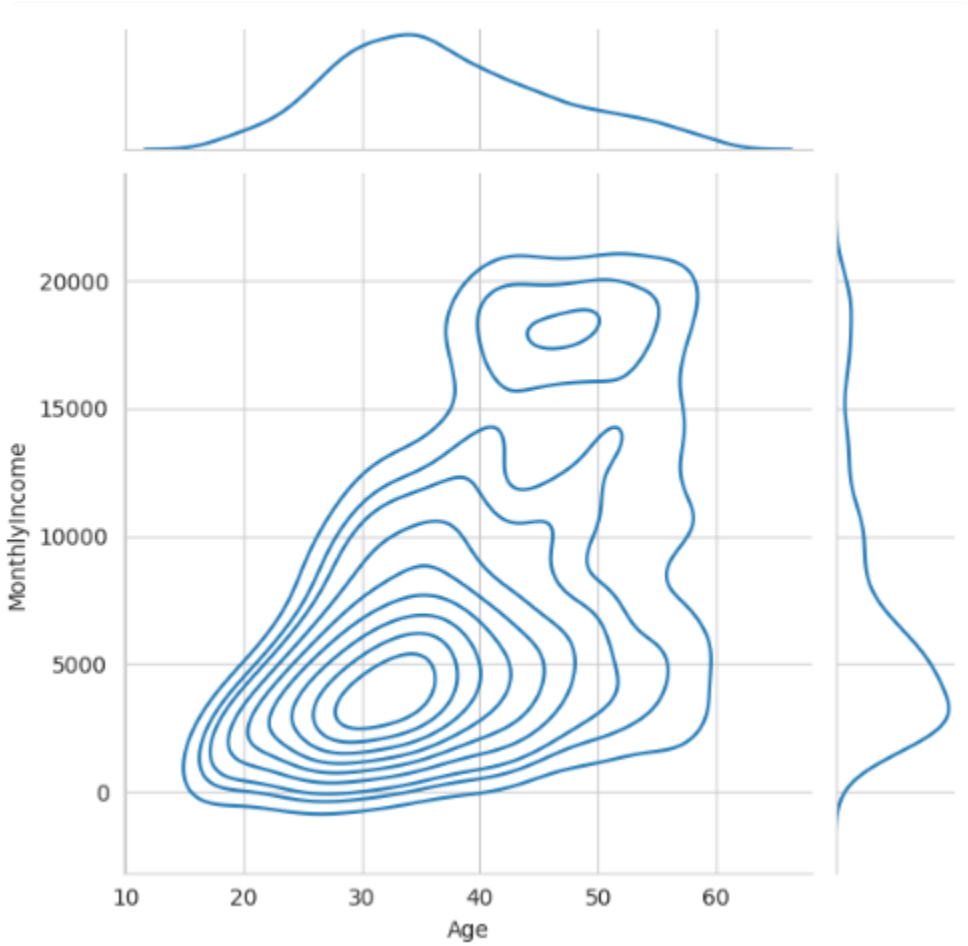
As you can see from the above plots, those variables have no clear-cut relationships.

Be aware that the *pairplot* function only works with numerical data (categorical string variables have to be converted into their equivalent numeric representations).

3.9. Jointplots

The seaborn.jointplot function is used to create a plot of two variables with bivariate and univariate graphs.

1. Create a *jointplot* that reproduces the one shown below.

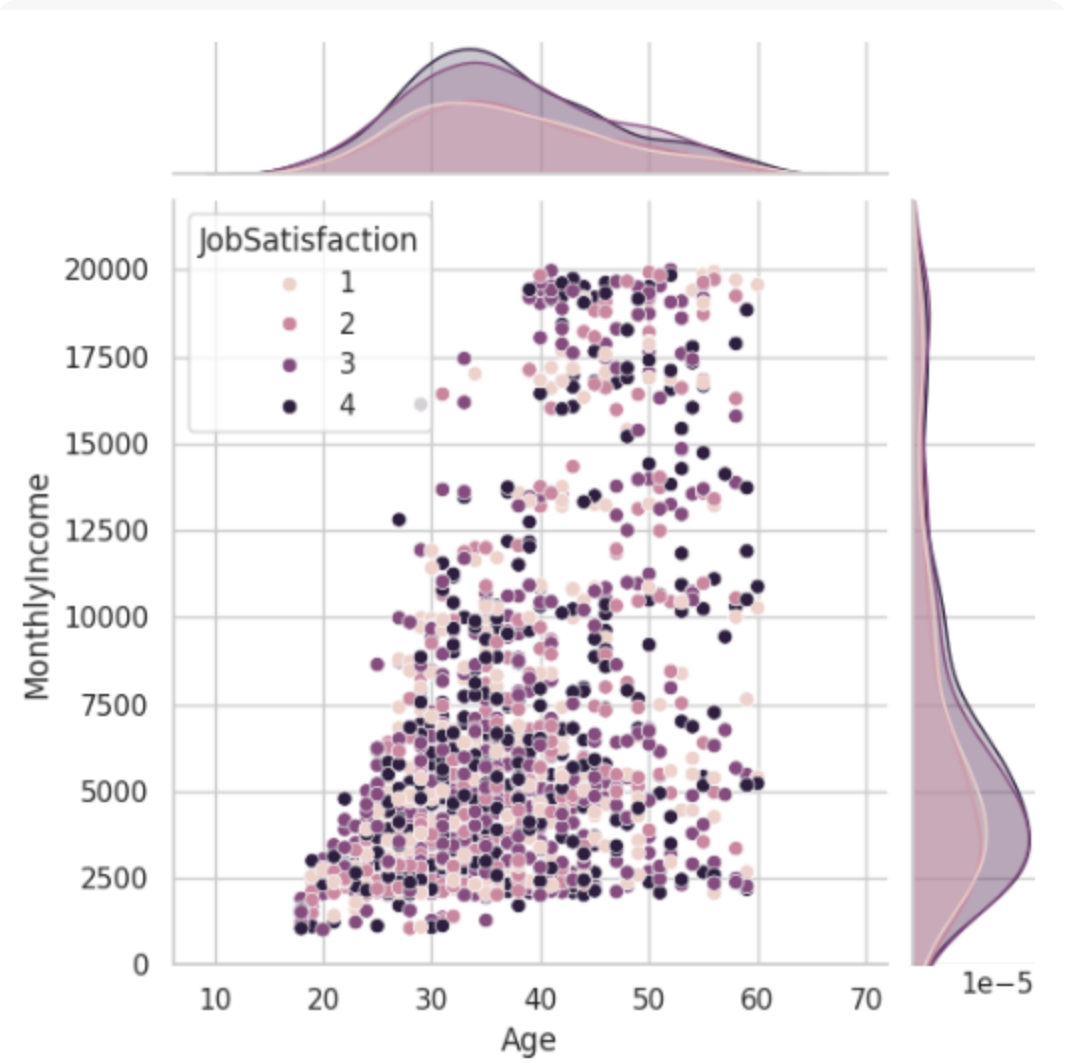


   Hint: Use the *kind="kde"* setting of the *jointplot* function.

2. Let's augment the impact of the pairplot by adding information about employees' job satisfaction. Use this code:

```
sns.jointplot(x="Age", y="MonthlyIncome", ylim=(0, 22000), data=df,
hue="JobSatisfaction")
```

   You should see the following plot:

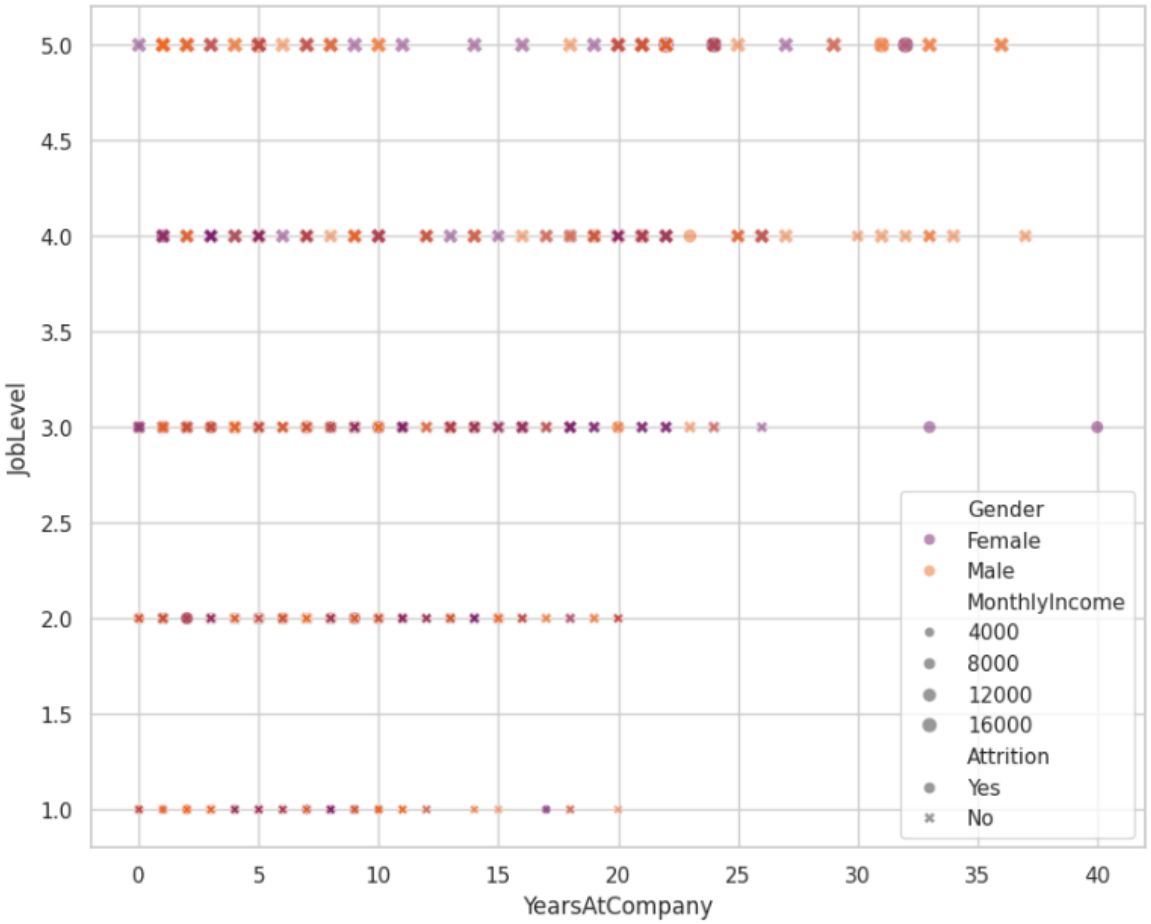


It appears that happy employees are all over the *Age-MonthlyIncome* map. Ditto unhappy ones ...

3.10. Scatterplots

1. Enter the following commands to create an informative plot using the seaborn.scatterplot function:

```
plt.figure(figsize=(10, 8))
sns.scatterplot(x="YearsAtCompany", y="JobLevel", hue="Gender",
size="MonthlyIncome", data=df, style="Attrition", palette='inferno', alpha=0.5)
```



The above scatterplot incorporates the following data attributes:

- Gender,
- MonthlyIncome,
- Attrition,
- JobLevel, and
- YearsAtCompany

Can you come up with similar scatterplot ideas?

3.11. Replots

The https://seaborn.pydata.org/generated/seaborn.relplot.html#seaborn.relplot function "*provides access to several different axes-level functions that show the relationship between two variables with semantic mappings of subsets*."

1. Enter the following command:

```
sns.relplot(
    data=df,
    x="YearsAtCompany", y="JobLevel", col="Gender",
    hue="Attrition", style="Attrition", size="MonthlyIncome",
    palette='inferno', alpha=0.5,)
```

   and analyze the produced visualization.

3.12. Heatmaps and Correlation

To support correlation analysis, we are going to apply the one-hot encoding scheme to the input data to convert the categorical variables in text form into their numerical representation.

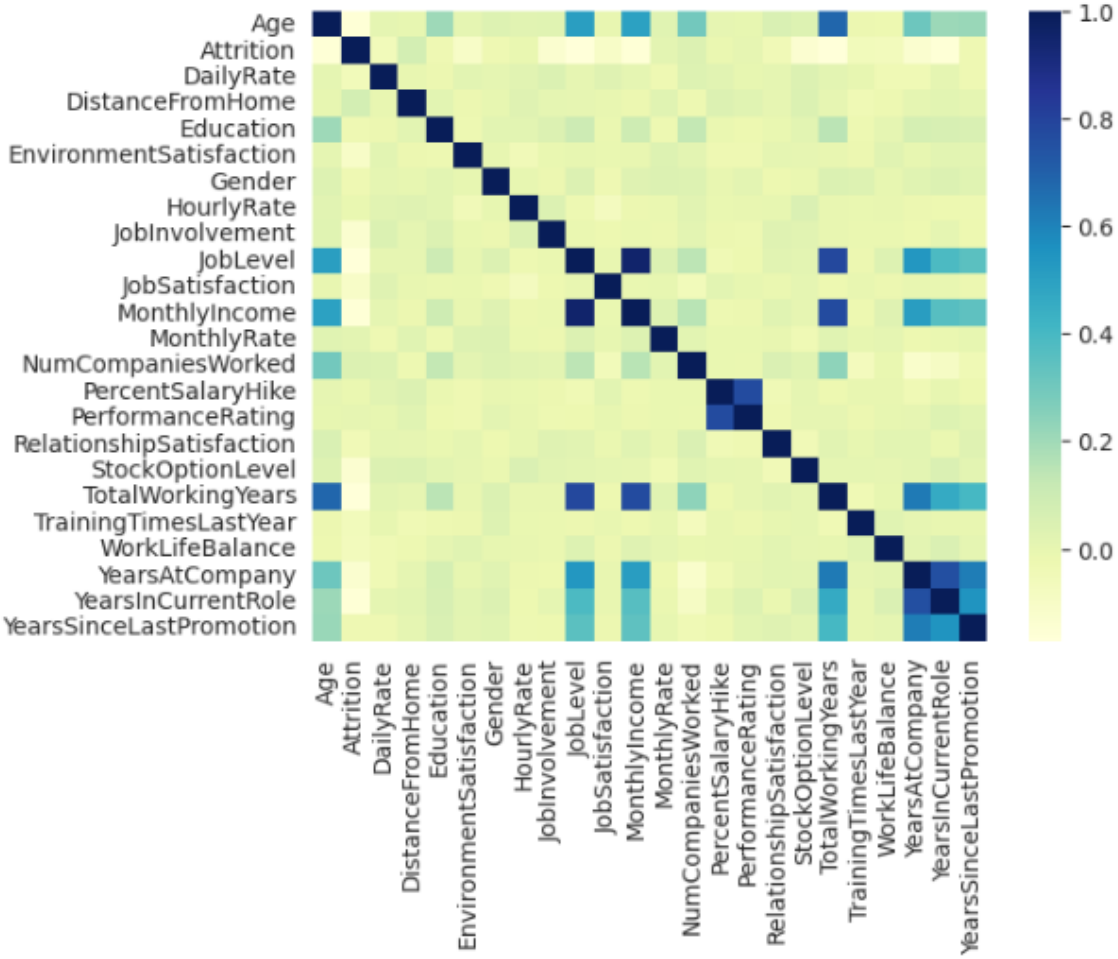
1. For the *Attribution* column, convert '**Yes**' to **1** and '**No**' to **0**.
2. Apply a similar transformation to the *Gender* column converting '**Female**' to **1** and '**Male**' to **0**
3. Use pandas' *get_dummies* function to apply the one-hot encoding scheme.
4. Enter the following command to create a DataFrame holding the correlation coefficients of all the feature pairs.

```
df_corr = df.corr()
df_corr
```

   A fragment of this DataFrame is shown below:

	Age	Attrition	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	Gender	HourlyRate	JobInvolvement	JobLevel
Age	1.000000	-0.159205	0.010661	-0.001686	0.208034	0.010146	0.036311	0.024287	0.029820	0.509604
Attrition	-0.159205	1.000000	-0.056652	0.077924	-0.031373	-0.103369	-0.029453	-0.006846	-0.130016	-0.169105
DailyRate	0.010661	-0.056652	1.000000	-0.004985	-0.016806	0.018355	0.011716	0.023381	0.046135	0.002966
DistanceFromHome	-0.001686	0.077924	-0.004985	1.000000	0.021042	-0.016075	0.001851	0.031131	0.008783	0.005303
Education	0.208034	-0.031373	-0.016806	0.021042	1.000000	-0.027128	0.016547	0.016775	0.042438	0.101589
EnvironmentSatisfaction	0.010146	-0.103369	0.018355	-0.016075	-0.027128	1.000000	-0.000508	-0.049857	-0.008278	0.001212
Gender	0.036311	-0.029453	0.011716	0.001851	0.016547	-0.000508	1.000000	0.000478	-0.017960	0.039403
HourlyRate	0.024287	-0.006846	0.023381	0.031131	0.016775	-0.049857	0.000478	1.000000	0.042861	-0.027853
JobInvolvement	0.029820	-0.130016	0.046135	0.008783	0.042438	-0.008278	-0.017960	0.042861	1.000000	-0.012630
JobLevel	0.509604	-0.169105	0.002966	0.005303	0.101589	0.001212	0.039403	-0.027853	-0.012630	1.000000

> **Note**
>
> For visual analysis of pairwise correlation, you can use the *seaborn.heatmap* function, e.g. the heatmap for the first 24 attributes is shown below.



5. Write a Python function that identifies all the feature pairs with a correlation coefficient greater than 0.5 and those with a correlation coefficient below -0.5

   You should see the following outputs (the '~' below is just a formatting character in the print statement):

```
Age~JobLevel :0.51
Age~TotalWorkingYears :0.68
JobLevel~MonthlyIncome :0.95
JobLevel~TotalWorkingYears :0.78
JobLevel~YearsAtCompany :0.53
JobLevel~JobRole_Manager :0.55
MonthlyIncome~TotalWorkingYears :0.77
MonthlyIncome~YearsAtCompany :0.51
MonthlyIncome~JobRole_Manager :0.62
PercentSalaryHike~PerformanceRating :0.77
TotalWorkingYears~YearsAtCompany :0.63
YearsAtCompany~YearsInCurrentRole :0.76
YearsAtCompany~YearsSinceLastPromotion :0.62
YearsAtCompany~YearsWithCurrManager :0.77
YearsInCurrentRole~YearsSinceLastPromotion :0.55
YearsInCurrentRole~YearsWithCurrManager :0.71
YearsSinceLastPromotion~YearsWithCurrManager :0.51
Department_Human Resources~EducationField_Human Resources :0.65
Department_Human Resources~JobRole_Human Resources :0.90
Department_Sales~EducationField_Marketing :0.53
Department_Sales~JobRole_Sales Executive :0.81
EducationField_Human Resources~JobRole_Human Resources :0.55
```

   and Analyze the produced output.

```
StockOptionLevel~MaritalStatus_Single :-0.64
BusinessTravel_Non-Travel~BusinessTravel_Travel_Rarely :-0.53
BusinessTravel_Travel_Frequently~BusinessTravel_Travel_Rarely :-0.75
Department_Research & Development~Department_Sales :-0.91
Department_Research & Development~JobRole_Sales Executive :-0.73
EducationField_Life Sciences~EducationField_Medical :-0.57
MaritalStatus_Married~MaritalStatus_Single :-0.63
OverTime_No~OverTime_Yes :-1.00
```

   Some pair relationships are self-explanatory: e.g. *OverTime_No~OverTime_Yes : -1.0* reflects their mutual exclusivity; ditto *MonthlyIncome~TotalWorkingYears : 0.77* . Others may be a bit more subtle ...

3.13. Answers

Q: What can be said about the *EmployeeNumber* column?

A: This is potentially sensitive information that most likely carries little to no signal for the EDA at hand. It should also be dropped.

Q: Does the *DistanceFromHome* variable influence the decision to leave the company?

A: Yes, it does have some influence, even though the *Attrition* feature has a great deal of data point overlapping.

Q: What is the median of *DistanceFromHome* for each *Attrition* category?

A:

```
df[df.Attrition == 'Yes']['DistanceFromHome'].median()  # 9.0
df[df.Attrition == 'No']['DistanceFromHome'].median()   # 7.0
```

3.14. End of Project

Predicting Employee Attrition Project

MODULE 4

In this project you will use the same dataset you worked on in the Exploratory Data Analysis (EDA) project. You will repeat some of the working environment setup activities from the EDA project and then further preprocess data for use in classification models, which you will create to predict employee attrition.

In this guide, we will use terms *column, feature, attribute,* and *variable* interchangeably.

Note: As you progress through the project, you may occasionally be asked questions marked "**Q**" (without quotes), the answers to which are given either in the project notebook or at the end of this project guide.

4.1. The Project Solution File

The solution notebook, **Predicting_Employee_Attrition_Project_Solution.ipynb,** is located in the** **LabFiles\predicting-employee-attrition-project\solutions** directory of the student package.

The ToC of the solution notebook is shown below:

Imports

Upload and deflate the Employee_Attrition.zip file

Load the Employee_Attrition.csv file into a pandas DataFrame

The Dataset Info Card

Extract the class label as the target variable y and the predictor variables as X

Review the predictors for predictive capability. Drop the "zero-signal" predictors

A typical record

Deal with the categorical variables -- apply one-hot encoding scheme

The Model Training and Performance Gauging function

Model: RandomForestClassifier

Save (serialize) the model

Deserialize the model

Analyze feature importance

Simplify the model

Drop the least important features (columns)

Using the employee feedback

Is OverallSatisfaction alone sufficient?

Model: LogisticRegression

Model: SVM

Scale the numeric column values.

4.2. The Input File

The **Employee_Attrition.csv** file you will use in this project is zipped in the *Employee_Attrition.zip* archive that is located in the **LabFiles\predicting-employee-attrition-project\data\HCM** directory of your student package.

The file has 1470 records with 35 attributes.

A fragment of the file's content is shown below:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical

4.3. Load the Data

1. Upload the archive file to your notebook working environment and unzip it.
2. Load the *Employee_Attrition.csv* input file into a pandas DataFrame named **df**.
3. Call the DataFrame's **info()** method; you should see this output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
```

```
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

4.4. Extract the Class Label and the Predictor Variables

1. Enter the following commands:

```
y = df['Attrition']
X = df.drop(['Attrition'], axis=1)
```

2. Convert the “Yes” and “No” values in **y** to 1 and 0

   Observe a 5:1 class label imbalance. The ratio should be maintained through the training and testing phases. Most scikit-learn estimators take care of class label imbalances via a parameter normally called *class_weight*, which instructs

the model to pay more attention to the minority class(es). We will look into this aspect of modeling in one of the next projects.

4.5. Review the Predictors. Drop the "Zero-Signal" Predictors

These activities are similar to those you performed in the EDA project.

1. Identify and drop columns with low-to-zero predictive capability.

   For the list of columns to be dropped, see the **Answers** section at the end of this project guide.

   > ⓘ Note
   >
   > The shape of your DataFrame after removing the qualifying columns should be (**1470, 30**).

4.6. Handle Categorical Variables

1. Apply the one-hot encoding scheme to the **X** DataFrame.

   > ⓘ Note
   >
   > This should add 21 new columns.

   The updated list of columns is shown below for your reference:

```
Index(['Age', 'DailyRate', 'DistanceFromHome', 'Education',
       'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
       'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager',
       'BusinessTravel_Non-Travel', 'BusinessTravel_Travel_Frequently',
```

```
       'BusinessTravel_Travel_Rarely', 'Department_Human Resources',
       'Department_Research & Development', 'Department_Sales',
       'EducationField_Human Resources', 'EducationField_Life Sciences',
       'EducationField_Marketing', 'EducationField_Medical',
       'EducationField_Other', 'EducationField_Technical Degree',
       'Gender_Female', 'Gender_Male', 'JobRole_Healthcare Representative',
       'JobRole_Human Resources', 'JobRole_Laboratory Technician',
       'JobRole_Manager', 'JobRole_Manufacturing Director',
       'JobRole_Research Director', 'JobRole_Research Scientist',
       'JobRole_Sales Executive', 'JobRole_Sales Representative',
       'MaritalStatus_Divorced', 'MaritalStatus_Married',
       'MaritalStatus_Single', 'OverTime_No', 'OverTime_Yes'],
dtype='object'
```

4.7. Uniform Classification Model Setup

In subsequent sections of this project, you will create three classification models using the following scikit-learn estimators:

- RandomForestClassifier
- LogisticRegression, and
- SVM

All of the estimators will share the following setup and preprocessing steps:

- The train-test split of the input data using the *sklearn.model_selection.train_test_split* function into a 70% - 30% ratio with a random state of *1011*.
- All the listed above models should be created with the *random_state=1011* parameters
- Use the model performance functions from the *sklearn.metrics* module:
  - accuracy_score,
  - confusion_matrix, and
  - classification_report

You may want to create a generic function that executes the standard machine learning cycle using the above artifacts and functions:

- Trains the model
- Test the model
- Evaluate the model's performance

4.8. The RandomForestClassifier Model

Using the uniform model setup from the previous guide part, create an instance of *RandomForestClassifier* estimator with 100 estimators (decision trees) and evaluate its performance.

You should see the following model performance aggregated report (your report's layout may look different, but all the reporting functions from the model's setup part above must be present):

```
=======================================================
Accuracy Score: 85.3%
-------------------------------------------------------
Classification report:
               precision    recall  f1-score    support
           0       0.86      0.99      0.92        368
           1       0.75      0.16      0.27         73

    accuracy                           0.85        441
   macro avg       0.80      0.58      0.59        441
weighted avg       0.84      0.85      0.81        441
-------------------------------------------------------
Confusion matrix:
[[364 4]
 [ 61 12]]
=======================================================
```

Q: Are you satisfied with the results? The accuracy score metric seems high enough — Yahoo!

Q: Is the model overfitting? How can you find this out?

To improve the current model, you may want to use *GridSearchCV* or *RandomizedSearchCV* to find the best combination of the model's hyperparameters that would boost its performance. We will leave this exercise for a later project; for now here is one attempt at improving the model:

1. Use the following model configuration and go through the usual machine learning cycle:

```
RandomForestClassifier(n_estimators=50, max_depth=11, min_samples_split=10,
min_samples_leaf=3, random_state=1011)
```

   You should see the following (slightly improved) performance report:

```
Accuracy Score: 85.3%
-------------------------------------------------------
Classification report:
               precision    recall  f1-score    support
           0       0.86      1.00      0.93        368
           1       1.00      0.16      0.34        73

    accuracy                           0.87        441
   macro avg       0.93      0.60      0.63        441
weighted avg       0.89      0.87      0.81        441
-------------------------------------------------------
Confusion matrix:
[[368 0]
 [ 58 15]]
=======================================================
```

> ⓘ Note
>
> In the subsequent steps, we will be using this model object.

4.9. Save (Serialize) and Deserialize the Model

Let's practice model saving and restoring,

1. Save and reload the last model using the **pickle** library.

> ⓘ Note

> Use the **get_params()** model's method to ensure that the model has been fully restored.

4.10. Analyze Feature Importance

1. Use the ***feature_importance*** attribute of the trained RandomForestClassifier model to sort the features in descending order of their importance.

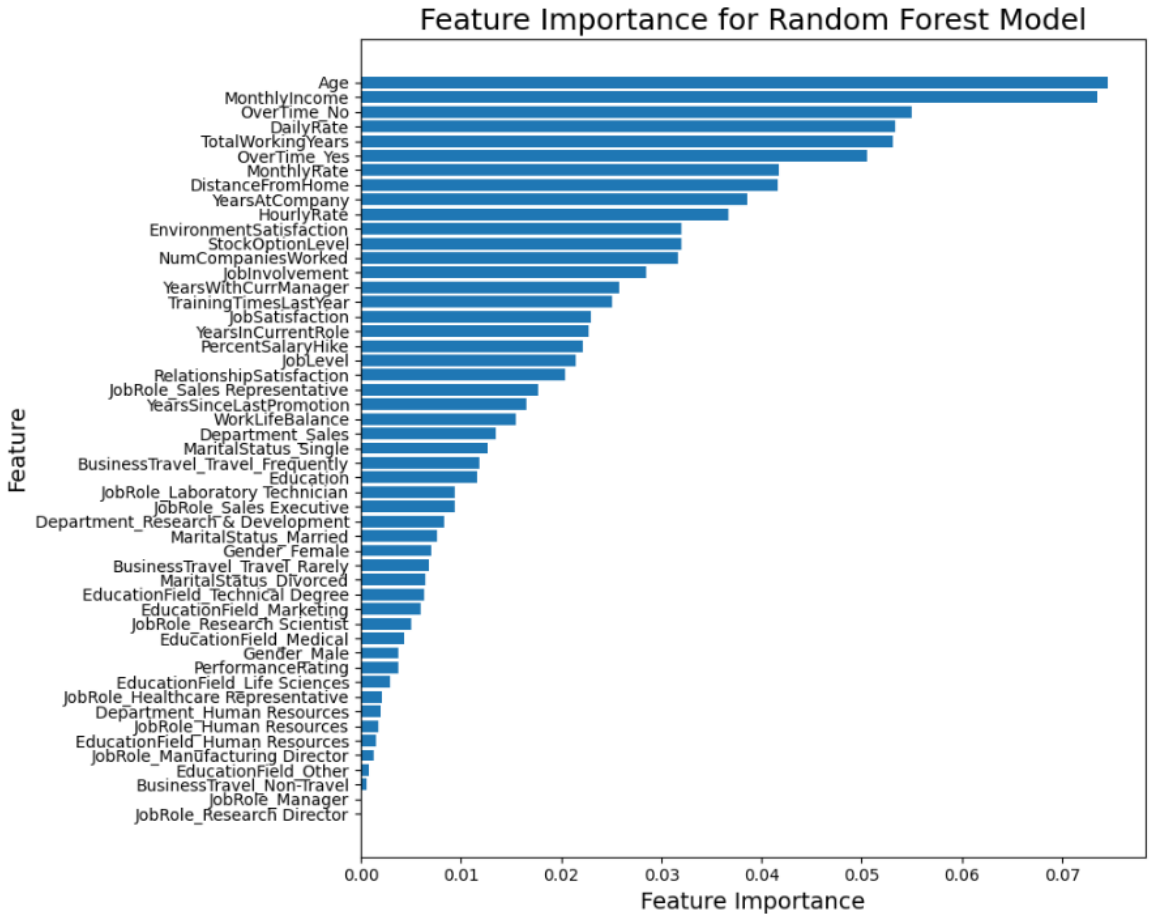
   You should see the following output (the 6 most important features as seen by the *RandomForestClassifier* estimator are shown in bold):

```
[('Age', 0.07459800633683424),
 ('MonthlyIncome', 0.07351117485510157),
 ('OverTime_No', 0.05495831276899056),
 ('DailyRate', 0.053391040543012726),
 ('TotalWorkingYears', 0.053102060061303025),
 ('OverTime_Yes', 0.05056739961979184),
 ('MonthlyRate', 0.041688880259845885),
 ('DistanceFromHome', 0.04158521581437315),
 ('YearsAtCompany', 0.0386176696143964),
 ('HourlyRate', 0.03664643302676625),
 ('EnvironmentSatisfaction', 0.03200239588073458),
 ('StockOptionLevel', 0.03199926390544927),
 ('NumCompaniesWorked', 0.031705401486171354),
 ('JobInvolvement', 0.028468873743186726),
 ('YearsWithCurrManager', 0.025832240739274585),
 ('TrainingTimesLastYear', 0.025054168020240224),
 ('JobSatisfaction', 0.02299880020089436),
 ('YearsInCurrentRole', 0.02278734017303065),
 ('PercentSalaryHike', 0.022197887732989856),
 ('JobLevel', 0.02143930078775593),
 ('RelationshipSatisfaction', 0.020415506446201485),
 ('JobRole_Sales Representative', 0.017699324163287174),
 ('YearsSinceLastPromotion', 0.016572158231252344),
 ('WorkLifeBalance', 0.015478529059305867),
 ('Department_Sales', 0.013492620025165153),
 ('MaritalStatus_Single', 0.012634718320443624),
 ('BusinessTravel_Travel_Frequently', 0.011779932673240564),
 ('Education', 0.011638240975073917),
 ('JobRole_Laboratory Technician', 0.009400447611299852),
 ('JobRole_Sales Executive', 0.009382343339013246),
 ('Department_Research & Development', 0.008309674151251315),
```

```
('MaritalStatus_Married', 0.007612073907501589),
('Gender_Female', 0.006971012153789172),
('BusinessTravel_Travel_Rarely', 0.006840403769827584),
('MaritalStatus_Divorced', 0.006479815742613668),
('EducationField_Technical Degree', 0.006263658757015706),
('EducationField_Marketing', 0.006006577347687884),
('JobRole_Research Scientist', 0.004993855155985105),
('EducationField_Medical', 0.004338780668327801),
('Gender_Male', 0.003724500678732356),
('PerformanceRating', 0.0036957009292013275),
('EducationField_Life Sciences', 0.0028772981328390374),
('JobRole_Healthcare Representative', 0.0020624039114219253),
('Department_Human Resources', 0.001974541313431451),
('JobRole_Human Resources', 0.001771844808000089),
('EducationField_Human Resources', 0.0015621064257977917),
('JobRole_Manufacturing Director', 0.0013211868323581013),
('EducationField_Other', 0.0008526016395809736),
('BusinessTravel_Non-Travel', 0.0005534974478339274),
('JobRole_Manager', 0.0001364581158541729),
('JobRole_Research Director', 6.321696522767151e-06)]
```

2. Visualize the feature importance as horizontal bars (use the Matplotlib's **barh()** function and the **tight_layout()** plotting layout)

   You should see the following plot:



4.11. Simplify the Dataset

One approach to improving model predictive capability is to simplify the training dataset by dropping less important features. This can be seen as a form of model regularization. Let's see if this approach could help us.

1. Drop 9 least important features (a rather invasive step, we must admit):

```
['JobRole_Manufacturing Director', 'BusinessTravel_Non-Travel',
    'EducationField_Other', 'JobRole_Human Resources',
    'Department_Human Resources', 'EducationField_Human Resources',
    'JobRole_Manager', 'JobRole_Healthcare Representative',
    'JobRole_Research Director']
```

When you run the model, you will see that the model has lost some of its predictive capacity, meaning that those “trailing” features also contribute to the accuracy of the model. The 'JobRole_Human Resources', 'Department_Human Resources', and 'EducationField_Human Resources' features are really important!

4.12. Use the Employee Feedback

The employee feedback is mainly captured in these three features:

- EnvironmentSatisfaction
- JobSatisfaction, and
- RelationshipSatisfaction

Let’s see if they are objective enough to reflect the employee’s genuine satisfaction on the job and that they can be used to predict employees' eventual decision to leave for greener pastures.

1. Combine the above three features into a single synthetic feature as an arithmetic mean; name it *OverallSatisfaction*.

2. Drop the three constituent features.

   When you run the model, you will see that the model has not improved, meaning that either

   - The employee’s feedback does not reflect their real sentiments, or
   - This way of blending the feedback was not adequate

Most likely, true is the former, which you can also corroborate by using this feature as the only attribute of the input dataset.

Company leadership and the HR department should critically review the employees' responses and seek ways to better engage them in a more objective feedback loop.

4.13. The Logistic Regression Model

1. Create an instance of the *LogisticRegression* scikit-learn estimator.
2. Train and test it.

   You should see that the performance of the model leaves much to be desired …

4.14. The SVM Model

1. Using the *MinMaxScaler()* class, scale the data to its default (0,1) range.

   Q: Why do you have to do that?

2. Create an instance of the SVC estimator with these parameters:

   ```
   SVC(C=1.0, cache_size=512, random_state=1011)
   ```

   Train and test the model. The performance of the first-cut SVC is below par …

3. Relax model regularization by increasing the C parameter to 2.5.

   > ⓘ **Note**
   >
   > According to the SVC documentation page:

   "The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty."

   ```
   model = SVC(C=2.5, cache_size=512, random_state=1011)
   ```

   You should see a noticeable improvement in the model's performance.

   ```
   ========================================================
   Accuracy Score: 87.1%
   --------------------------------------------------------
   Classifiction report:
                 precision    recall  f1-score   support
   ```

```
           0       0.89      0.97      0.93       368
           1       0.70      0.38      0.50        73

    accuracy                           0.87       441
   macro avg       0.79      0.68      0.71       441
weighted avg       0.86      0.87      0.85       441

------------------------------------------------------
Confusion matrix:
[[356  12]
 [ 45  28]]
======================================================
```

As you can see, the C hyperparameter effectively contributes to the model fine-tuning effort.

Our SVM model, however, creates more false negatives (6 in our case above), which basically raises false flags to management and HR that someone may have plans to leave the company while they are not. In our situation, this is a less harmful outcome compared to missing those employees who disguise themselves as being loyal to the company when they are not.

4.15. Answers

- You should drop the following columns:
  `'EmployeeCount', 'EmployeeNumber', 'Over18', and 'StandardHours'`

Q: Are you satisfied with the results? The accuracy score metric seems high enough!

A: The accuracy score is misleading in this case as its high value is influenced by the prevalence of the true positive observations of the employees who have been predicted to stay with the company (you will recall that there is a significant class imbalance.)

The true model performance can be read from the related confusion matrix:

```
[[364 4]
 [ 61 12]]
```

The classification report also corroborates the obviously poor model performance:


```
-------------------------------------------------------
Classifiction report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       368
           1       0.75      0.16      0.27        73
```

There are only 73 test observations for the Attrition=1 category, and the majority of them are misclassified.

Q: Is the model overfitting?

A: Yes, the model overfits the train portion of the dataset. You can quickly verify this with this command:

```
confusion_matrix(y_*train*, <your model>.predict(X_train))
array([[865, 0],
       [ 0, 164]]
```

Overfitting prevents the model from generalizing beyond the training observations.

Q: Why do you have to do that (scale the data for use in SVM models)?

A: The SVM algorithm is very sensitive to data scaling; you need to scale (transform) your training and test data to ranges like [-1,1] or [0,1].

4.16. End of Project

Fine-Tuning Classification Models — Capstone Project

MODULE 5

In this project, you will apply the skills you have learned so far in this program and add additional techniques, methods, and tools to your data scientist's toolbox.

You will use the same dataset you worked on in the *Predicting Employee Attrition* project. You will be required to repeat some of the data pre-processing steps and activities to set up your working environment, including . Creating the **X** and **y** datasets from the uploaded input file . Dropping "zero-signal" predictors . Splitting the data into the training and test data sets . Applying one-hot encoding

5.1. The Project Solution File

The solution notebook,
Fine_Tuning_Classification_Models_Capstone_Project_Solution.ipynb, is located in the **LabFiles\fine-tuning-classification-models-capstone-project\solutions** directory of the student package.

5.2. The Input File

The **Employee_Attrition.csv** file you will use in this project is zipped in the *Employee_Attrition.zip* archive that is located in the *LabFiles\fine-tuning-classification-models-capstone-project\data\HCM* directory of your student package.

The file has 1470 records with 35 attributes.

A fragment of the file's content is shown below:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical

5.3. Fine-Tuning Classification Models

The main objective of this project is to identify the best set of hyperparameters to fine-tune the *RandomForestClassifier* models using the sklearn.model_selection.GridSearchCV class.

Once you have identified the set of best parameters as returned by the *best_params* attribute of the trained *GridSearchCV* object, see if you can further improve the fine-tuned model by leveraging the *class_weight* model parameter's functionality that takes care of the exiting class imbalance.

5.4. Use the XGBoost Library

In this project part, explore the capabilities of the XGBoost machine learning library.

XGBoost offers the scikit-learn estimator interface allowing scikit-learn users to quickly switch between different underlying machine learning algorithm implementations. The library is already preinstalled in the Colab environment.

Use the *xgb.XGBClassifier* estimator in your notebook and compare the results with the previously fine-tuned *sklearn.ensemble.RandomForestClassifier* based model.

5.5. End of Project

Cluster Modeling Project

MODULE 6

This project aims to test your knowledge and understanding of the unsupervised machine learning.

You will use a modified version of the dataset you worked on in the *Predicting Employee Attrition* project to see if it lends itself to clustering modeling and analysis of employee attrition.

6.1. The Project Solution File

The solution notebook, **Cluster_Modeling_Project_Solution.ipynb,** is located in the **LabFiles\cluster_modeling_project\solutions** directory of the student package.

The ToC of the solution notebook is shown below:

6.2. The Input Files

You will use the following files located in the *LabFiles\cluster_modeling_project\data\HCM* directory of your student package:

```
X.pkl
y.pkl
```

6.3. Load the Data

1. Upload the input files into your notebook working environment.

   The files are a DataFrame and a Python list (of attrition labels) in the pickle binary format.

2. Deserialize the **X** and **y** files.

6.4. The kMeans Algorithm

1. Enter the following command to create an instance of the *sklearn.cluster.KMeans* class with the following parameters:

   ```
   n_clusters=2,
   init='k-means++',
   n_init='auto',
   random_state=1011,
   verbose=True
   ```

2. Train (fit) the model on **X**.

   You should see that the model converges quite fast — in mere 19 iterations. Is it a good sign? Let's evaluate the model's performance.

   The trained model has the following parameters:

   ```
   {'algorithm': 'lloyd',
    'copy_x': True,
   ```

```
 'init': 'k-means++',
 'max_iter': 300,
 'n_clusters': 2,
 'n_init': 'auto',
 'random_state': 1011,
 'tol': 0.0001,
'verbose': True}
```

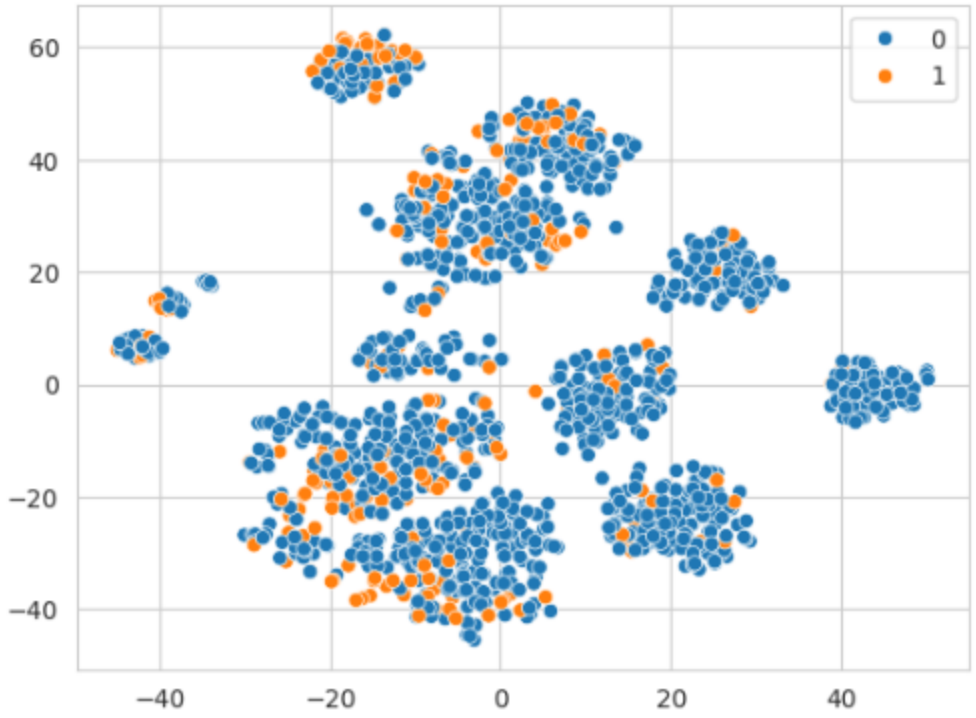
The last captured inertia (the within-cluster sum-of-squares criterion) value that kMeans tries to minimize when identifying the centroids is:

```
50865906789.80997
```

Let's visualize the input data by using the t-SNE algorithm as implemented in the *sklearn.manifold.TSNE* class.

3. Scale the data (the **X** DataFrame) using the *sklearn.preprocessing.StandardScaler* class and use the scaled data as input to the TSNE manifold transformer (create it with two components).

   You should see the following 2-D plot:



> ⓘ Note

> The above plot was created using the seaborn's *scatterplot* method, *hue=y*, and the *sns.set_style("whitegrid")* style.

As you can see, TSNE does not see the data grouped around two centroids — rather, there are about 9 or so groups of data points.

4. Calculate the count of the labels predicted by the kMeans model in each category (leave (1s)/stay(0s)).

   You should see the following output (it does not matter at the moment which labels are 0s and which ones 1s):

   ```
   [724, 746]
   ```

   meaning that the model, essentially, works like a 50/50 random label generator.

   The ground-truth labels have this composition of 0s and 1s:

   ```
   [1233, 237]
   ```

5. Using the *sklearn.metrics.accuracy_score* function, find the accuracy of the model.

   You should see that it is around 50%. A coin flip clustering strategy ...

6.5. The silhouette_score Function

To complete the kMeans cluster modeling analysis, use the *sklearn.metrics.silhouette_score* function to try to find the optimum number of clusters (we use 2 clusters given the nature of the problem we are modeling).

For this array of possible cluster numbers: [2,3,4,5,6,8,9, 10, 11, 12] , you would see the following results (your output formatting will differ):

```
2:: inertia: 50865906789.81, silhouette score: 0.466
3:: inertia: 40731317995.9, silhouette score: 0.342
4:: inertia: 25207948422.07, silhouette score: 0.417
```



```
5:: inertia: 22168389962.79, silhouette score: 0.35
6:: inertia: 15636517042.74, silhouette score: 0.386
8:: inertia: 11677410789.28, silhouette score: 0.4
9:: inertia: 10272448954.22, silhouette score: 0.404
10:: inertia: 9350763149.56, silhouette score: 0.372
11:: inertia: 8281619645.76, silhouette score: 0.376
12:: inertia: 7702595327.97, silhouette score: 0.357
```

You can see a bump in the silhouette score at 9 clusters, which more or less corroborates the TSNE's view of the data.

The interpretation of what those 9 clusters actually are is left as an exercise to the reader. It may well be something to do with the proverb about a cat's nine lives or 9 stages an employee goes through before deciding to leave the company …

Veteran data practitioners may share their insight that if you torture the data long enough, it will confess …

6.6. The BIRCH Algorithm

See if you can get better results using the *sklearn.cluster.Birch* class as it offers more leeway in terms of the available hyperparameters, including the *threshold* and *branching factor*.

You may get a slightly better performance compared with the kMeans algorithm, still nothing to post on the Kaggle competition board …

6.7. The Agglomerative Clustering Algorithm

As a final attempt to get a more meaningful data clustering result, let's see what the *sklearn.cluster.AgglomerativeClustering* class can do for us.

To save you some time towards the end of this project, here is the model's configuration worthy of looking at:


```
AgglomerativeClustering(
    n_clusters=2, linkage='complete', metric='cosine')
```

If you score this model's predictions (*labels*) with the ground-truth labels (*y*), you will get a rather good accuracy score:

```
0.6496
```

So what was the trick the *Agglomerative Clustering* algorithm pulled to give us such a treat?

Here is one opinion.

Agglomerative Clustering works by recursively merging cluster pairs of sample data using the *linkage* distance. Originally, it may well detect 9 or so distinct clusters we saw earlier, and then it iteratively applies, figuratively speaking, the M&A (Merges and Acquisition) business practice to organically grow a model from fragmented data groups.

Still, the model is below par from the confusion matrix's perspective:

```
array([[927, 306],
    [209, 28]])
```

With a large proportion of mislabeled observations (about 50% of the data: 306 + 209), we cannot be completely satisfied with the results and use the model for classification purposes ...

In many cases, including ours, the problem is not with the algorithms but, rather, with the data. The current dataset has a great deal of overlap in attribute ranges in both categories, and, to add insult to injury, the employee responses captured in the **Satisfaction* attributes *(['EnvironmentSatisfaction', 'JobSatisfaction', 'RelationshipSatisfaction'])* are not good indicators of employees' real sentiments. Obviously, the HR department need to jump to action and get to the bottom of the problem.

6.8. End of Project

Dimensionality Reduction Project

MODULE 7

In this project, you will have a chance to test and expand your knowledge of dimensionality reduction unsupervised machine learning algorithms and techniques.

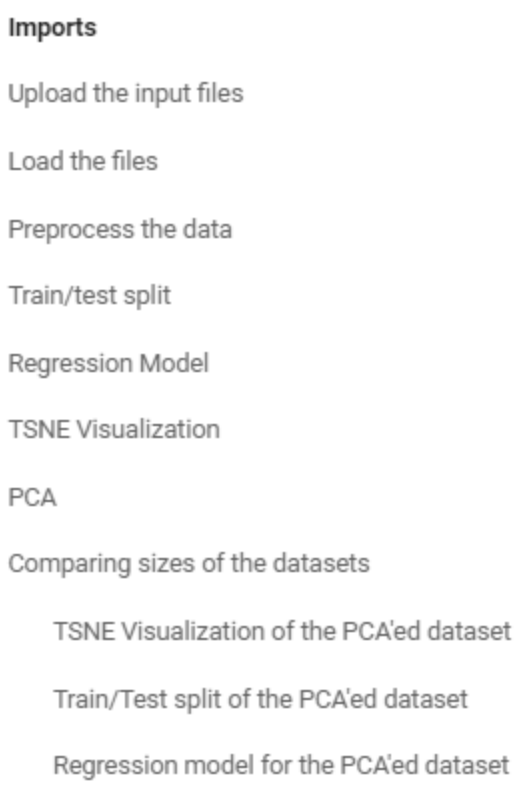
You will work through steps required to (substantially) reduce the size of the input data using the PCA algorithm without significantly affecting the quality of downstream tasks — a regression model will be utilized for this purpose.

For input data, you will use the dataset you worked on in the *Cluster Modeling* project.

7.1. The Project Solution File

The solution notebook, **Dimensionality_Reduction_Project_Solution.ipynb,** is located in the** **LabFiles\dimensionality-reduction-project** directory of the student package.

The ToC of the solution notebook is shown below:



7.2. The Input Files

You will use the following files located in the *LabFiles\dimensionality-reduction-project\data\HCM* directory of your student package:

```
X.pkl
y.pkl
```

7.3. Load the Data

1. Upload the input files into your notebook working environment.
2. Deserialize the **X** and **y** files.
3. Check the shape of the **X** DataFrame — It should be (1470, 51)

7.4. Preprocess the Data

The overall project plan is to compare the performance of regression models (with the *MonthlyIncome* column designated as the target variable) in two scenarios:

- Using the original dataset, and
- Using the dataset that has been transformed (reduced in size) by the PCA algorithm.

We are going to remove all the records related to employees who left the company.

With the **X** and **y** datasets, the quickest way to achieve this is to follow these steps:

1. Create a new column in the **X** DataFrame named '*Attrition*'. Fill it with the values from the **y** dataset (that holds the original labels).

2. Create a DataFrame that holds all the records that satisfy the **Attrition == 0** predicate.

3. Drop the *Attrition* column.

   We are going to drop a few variables related to employees' earnings and wages, leaving only the *MonthlyIncome* column as the target variable for regression analysis purposes.

4. Identify columns that have *Rate* in the name.

   You should have:

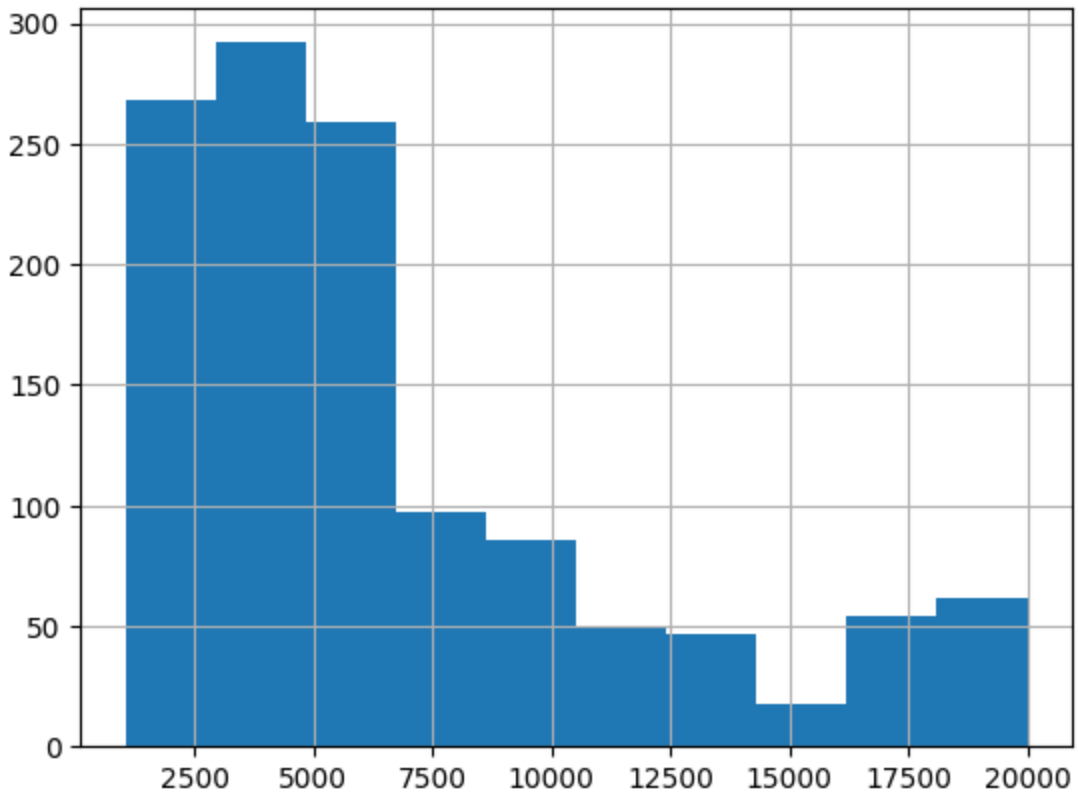
   ```
   ['DailyRate', 'HourlyRate', 'MonthlyRate']
   ```

5. Drop those rate-related columns.

6. Check the shape of the new DataFrame. Now it should be (1233, 48)

7. Create a histogram for the *MonthlyIncome* column's values.

   You should see the following plot:



8. Extract the *MonthlyIncome* column as the target variable **y** and drop the column in the DataFrame; call the updated DataFrame **X**

9. Split **X** into a train and a test datasets with the test size of 30% and random_state=1011.

7.5. The Regression Model

We are going to build the regression model using the XGBoost Regressor known for its stability and high performance.

1. Enter the following command to create an instance of *XGBRegressor* that implements the standard scikit-learn estimator interface.

```
xgb.XGBRegressor (objective='reg:squarederror', random_state=1011)
```

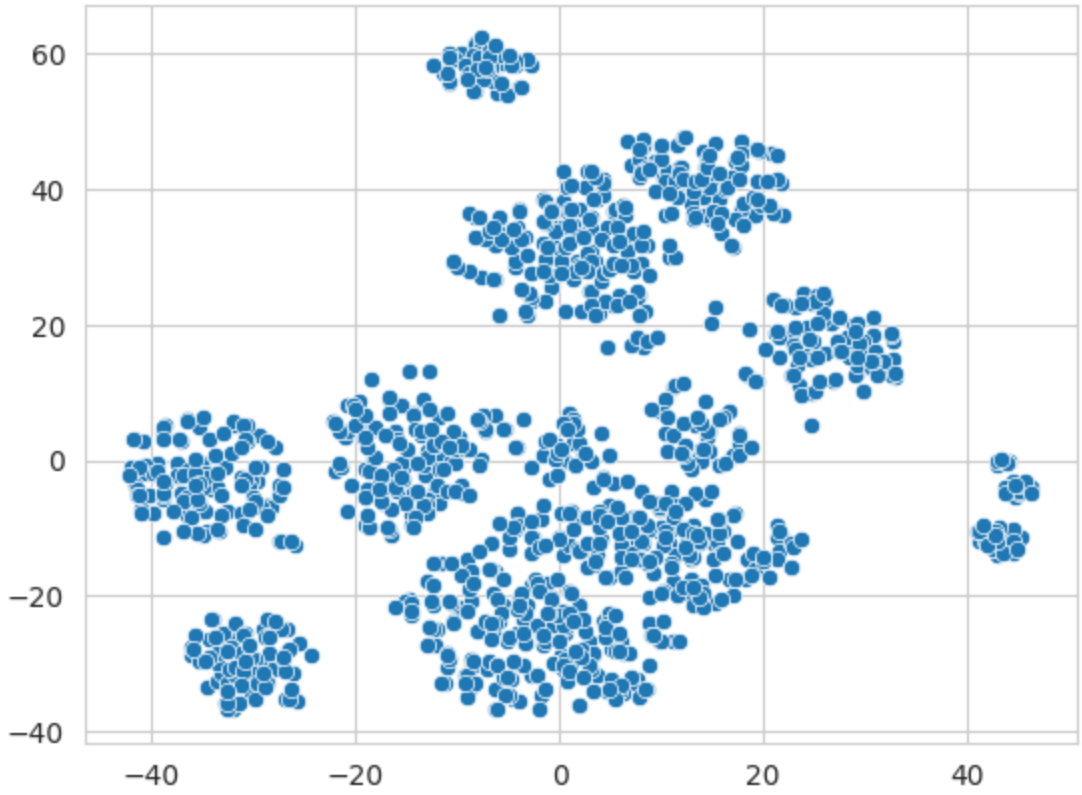
2. Train the model on the train subset of the X dataset.

3. Use the *score()* method of the trained model to evaluate the coefficient of determination (R^2) of the predictions.

   You should get a result around **0.94.**

4. Use the *sklearn.manifold.TSNE* implementation of the TSNE algorithm to reduce the dimensionality and visualize the X dataset.

   You should see the following output:



   Later, we will compare this plot with the distribution of the data points in a dataset preprocessed with the PCA algorithm.

7.6. PCA

> ⓘ Note

> The input DataFrame at this point should have 47 columns/attributes.

1. Enter the following commands:

```
pca = PCA(random_state=1011)
pca.fit(X)
print (pca.components_)
```

   You should see a 47x47 PCA component matrix.

   Let's assume that we need to reduce the input dataset to 1/3 of its original size without losing much of the contained information for a downstream machine learning task that will be represented by a regression model similar to what we used earlier in the project.

2. Create a PCA model with 16 components (1/3 of the 48 original attributes).

   Calculate the size of the NumPy array that backs the **X** DataFrame (available via the *to_numpy()* DataFrame method) and that of the new 16-component PCA matrix.

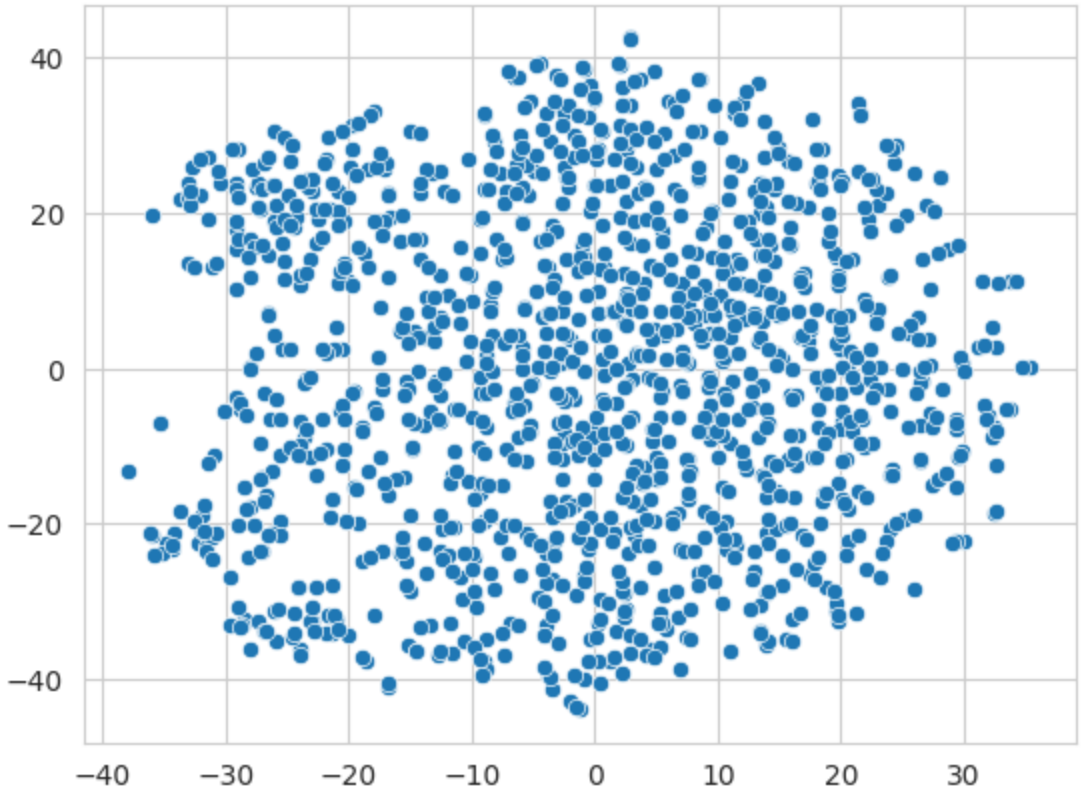
   You should get values similar to the one shown below:

```
Original X size: 452.7421875 KB
PCA X size: 154.125 KB
```

   > ⓘ **Note**
   >
   > NumPy arrays report their memory footprint via the *nbytes* attribute.

3. Create a TSNE visualization of the PCA'ed dataset.

   You should see the following plot:



Compare this plot with the one plotted against the original **X**. The new plot shows a higher density and homogeneity of the data distribution.

XGBoost uses an algorithm that is based on an ensemble of gradient-boosted decision trees and fitting the model on the PCA-processed dataset takes noticeably longer, which may be explained by the fact that there are no obvious candidates where to start/continue feature splitting.

4. Create, fit, and evaluate a new XGBoost-based regression model.

   You should see the following (rounded) output: **0.73.**

   Not a hefty price tag to pay for a 2/3 size reduction in the input dataset.

7.7. End of Project