Feature Engineering

Agenda

In this session, we will cover the following concepts with the help of a business use case:

- Features
- Feature engineering and it's methods
  - Applying domain expertise
  - Combining features
  - Encoding of categorical variables

Features

- In most cases, machine learning algorithms are fed a collection of numeric examples that are stacked on top of each other to form a two-dimensional feature matrix.
- This matrix has one **example** per row and one **function** per column.
- Understanding the task of these features is critical when doing some kind of learning on the dataset.
- Since the algorithms create their own internal transformations, deep learning features are normally simple.
- This method necessitates a vast volume of data and sacrifices interpretability; but, in image processing and natural language processing applications, these trade offs are often worthwhile.
- Feature engineering is needed for the majority of other use cases in order to translate data into a machine learning-ready format.
- The features chosen are critical for both the data's interpretability and the model's performance.
- A precise machine learning framework cannot be built without feature engineering.

What Is Feature Engineering?

- Feature Engineering repurposes already modified features to build new ones, making it possible for any Machine Learning algorithm to comprehend and master any pattern.
- It is the process of **creating new features** from raw data using domain knowledge to make the machine learning algorithms work.
- It is one of the fundamental concepts of machine learning and it is both difficult and expensive.
- Every new resource must enhance the performance in some way; otherwise, it would have the opposite effect, worsening the final outcome. When this happens, we must use feature selection instead of feature engineering.
- Acording to Andrew Ng, "Coming up with features is difficult, time-consuming, requires expert knowledge."
- According to Pedro Domingos, feature engineering is the art of coming up with new features with more predictive power using:
  - Experience
  - Domain expertise
  - Empirical processes
- The following are some of the most important components of feature engineering:
  1. Imputation
  2. Handling Outliers
  3. Binning
  4. One-Hot Encoding
  5. Grouping Operations
  6. Scaling

Most Important Techniques in Feature Engineering

- Applying domain expertise
- Combining features
- Encoding of categorical variables

Applying Domain Expertise

Domain expertise or domain knowledge is nothing but expertise in a particular field, such as Education, Healthcare, Consumer Goods, and Retail. A domain expert is someone who is not related to the technology aspect but has in-depth knowledge about the particular industry, how it is shaping up, the trends, and the things that might impact the industry.

For example, you are called in to develop a particular application for a consumer goods company, specifically an apparel and footwear one. The application that you build has to align to the industry and its various facets, and you being a technology expert wouldn't know much about it. This is where a domain expert will come in and explain how that industry works and what would be the best way to have the application built.

Combining Features

This is one of the most crucial part of feature engineering that involves analyzing the data set and understanding the variable(s) to combine one or more features to create new ones. This process involves intuitive decision-making and a small amount of research about the domain of the data set.

Combining features helps to increase the performance of the model which is fitted on the data.

Encoding of Categorical Variables

Let's first see what encoding is.

What Is Encoding?

Encoding is used to transform the categorical variable into numerical features.

For example, we have an attribute called gender in a data set where values are male and female. In this case, the numerical encoded version of the values will be 1 for male, 0 for female, or vice versa.

Since different kinds of categorical variables capture different amount of information, we need different techniques to encode them.

Label Encoding

Label encoding is a handy technique to encode categorical variables. However, such encoded nominal variables might end up being misinterpreted as ordinal.

Encoding Can Be Performed on the Following Types of Variables:

Nominal Variables

Consider the following three categorical variables and their values:

Color: Blue, Green, Red, Yellow

Educational Qualification: Primary School, Secondary School, Graduate, Post-Graduate, PhD

Salary Bracket: 0-50,000, 50,001-100,000, 100,001-150,000, 150,001-200,000

Although all three of them are categorical variables, they are different in the amount of information they convey. Let's look at them one by one.

Color conveys blue is different from red. That's all. The value of the variable is not meant to capture any relative difference among the values. Such variables are called `Nominal Variables`.

Ordinal Variables

Now consider educational qualifications.

The value "graduate" does not only convey that it is different from the value say "Primary school", it also implies that it is more in terms of qualification than "Primary School".

Such variables are called `Ordinal Variables` because they convey a sense of order.

In our example, Primary School < Secondary School < Graduate < Post-Graduate < PhD. (in terms of qualification).

Interval Variables

The third variable, salary bracket is similar to educational qualification by conveying order (a person earning 50,001-100,000 earns more salary than 0-50,000).

However, here, apart from knowing the order, we also know the interval between the values.

Here we can say that the averages of each of the values are separated by 50,000. Such variables are called `Interval variables`.

Techniques Used for Encoding Variables

There are two types of broadly used algorithms which perform the task of encoding of variables.

There are few libraries required to perform encoding variables:

- Pandas - It helps to retrieve datasets, handle missing data and perform data wrangling.
- NumPy - It helps to perform numerical operations in the dataset.
- sklearn.preprocessing - It helps in data transformation.

In [1]:
```
#Select the cell and click on run icon to import libraries.

import pandas as pd
import numpy as np

# Import label encoder
from sklearn import preprocessing
```

Instruction:

Download the **`Iris.csv`** dataset file from Course Resources and upload it in the lab using the Up arrow as shown below on the View Tab

Note 1:

- The **`iris_df`** is a dataframe that stores data imported from a CSV file in rows and columns format.
- The **head()** function helps to view the first few data present in the **`iris_df`** dataframe.

In [2]:
```
#Select the cell and click on run icon to retrieve and view the dataset.
# Import dataset

iris_df = pd.read_csv('Iris.csv')
iris_df.head()
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Observations from the above output:

> View few records of Iris dataset

Note 2:
The **info()** function helps to understand the dataset, the column name, total null values, and data type.

In [3]:
```
#Select the cell and click on run icon
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Observations from the above output:

> The variable **`species`** is a categorical Dtype **'object'**.

Note 3:

The **LabelEncoder()** function is used to convert categorical variables into numerical. According to our dataset, variable **`species`** is categorical, so convert **`species`** into numerical type.

In [4]:
```
#Select the cell and click on run icon to define LabelEnocder
label_encoder = preprocessing.LabelEncoder()
```

Note 4:

The **unique()** function is used to identify distinct rows present in the **`iris_df`** dataframe.

In [5]:
```
#Select the cell and click on run icon
iris_df['species'].unique()
```

Out[5]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

Observations from the above output:

> There are three categories of variable **`species`** such as *'setosa'*, *'versicolor'*, and *'virginica'* to be encoded.

Note 5:

- The **fit_transform()** method calculates the mean and variance of each feature and transforms all the features using the respective mean and variance.
- The **head()** function helps to view the first few data present in the **`iris_df`** dataframe.

In [6]:
```
#Select the cell and click on run icon
iris_df['species']= label_encoder.fit_transform(iris_df['species'])
iris_df.head()
```

Out[6]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Observations from the above output:

> The variable **`species`** is converted into numerical variable as 0's and 1's.

In [7]:
```
#Select the cell and click on run icon

iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    int32  
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

Observations from the above output:

> The variable **`species`** Dtype is converted to **'int64'**.
>
> > Using LabelEnocder, we can convert a categorical variable into a numerical variable.

One Hot Encoding

A data set with more dimensions requires more parameters for the model to understand, and that means more rows to reliably learn those parameters.

The effect of using One Hot Encoder is the addition of a number of columns (dimensions).

If the number of rows in the data set is fixed, the addition of extra dimensions without adding more information for the models to learn from can have a detrimental effect on the eventual model accuracy.

If the number of rows in the dataset is fixed and the addition of extra dimension without adding more information in the model would affect the model accuracy.

In the below example, you can see that category of Variable X is enocded into separate columns such as Variable X_Blue, Variable X_Yellow, Variable X_Red.

Encoding of categorical variables

One Hot Encoding:

Index	Variable X
1	Blue
2	Red
3	Yellow
4	Red
5	Red
6	Blue
7	Yellow

Index	Variable X_Blue	Variable X_Red	Variable X_Yellow
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0
5	0	1	0
6	1	0	0
7	0	0	1

There are few libraries required to perform **One Hot Encoding**:

- **datasets**: It helps to load the datasets from sklearn library.
- **OneHotEncoder**: It helps to encode categorical variable.

In [8]:
```
#Select the cell and click on run icon
from sklearn import datasets
from sklearn.preprocessing import OneHotEncoder
```

Note 6:

The given code helps to load the **Iris dataset** and create a dataframe **iris_data**, and **y** variable consists of target values.

In [48]:
```
#Select the cell and click on run icon
iris_data = datasets.load_iris()

iris_data = pd.DataFrame(data=np.c_[iris_data["data"], iris_data["target"]],
                         columns=iris_data["feature_names"] + ["target"])
y = iris_data.target.values
```

Note 7:

The **head()** function is used to view the first few data present in the **`iris_data`** dataframe.

In [10]:
```
#Select the cell and click on run icon
iris_data.head()
```

Out[10]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

Observations from the above output:

> There are four dependent variable and one independent variable present in the **`iris_data`** dataframe.

In [11]:
```
# Select the cell and click on run icon to view the target values
y
```

Out[11]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.])
```

Observations from the above output:

> The variable **`target`** is converted into an array.

Note 8:

- In OneHotEncoder, the categories set as 'auto' determines the categories automatically.
- The **reshape()** function reshapes an array without modifying its data.
- The **toarray()** function helps to represent the data in array format.

In [12]:
```
#Select the cell and click on run icon
onehotencoder = OneHotEncoder(categories='auto')
y = onehotencoder.fit_transform(y.reshape(-1,1))
print(y.toarray())
```

```
[[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
```

Observations from the above output:

> With the help of above output, you can see that variable **`target`** is converted into numerical variable as 0's and 1's into three separate categories .

Note 9:

The **get_dummies()** function is used for data manipulation and converts categorical variable to numerical variables .

In [13]:
```
#Select the cell and click on run icon
pd.get_dummies(iris_data.target).head()
```

Out[13]:

	0.0	1.0	2.0
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0

Observations from the above output:

> The target values are converted into dummy values in three different columns.

Hashing

Hashing is a technique that combines more than one category of a categorical variable into one single category.

There are three advantages in one-hot encoding or set-of word model:

- It's much simpler to code
- It doesn't require any dictionaries to prepare
- It's friendly to online learning where you can train on a data set that doesn't fit in memory because you need to see each example only once. One-hot encoding will not work well with online learning because to prepare dictionaries you need to see the whole data set first

Feature Hashing

Feature hasing is a important technique for handling sparse and high-dimensional features in machine learning.

- It is fast, simple, memory-efficient, and well-suited to online learning sceanrios.
- It converts unique tokens into integers.
- It operates on the exact strings that you provide as input and does not perform any linguistic analysis or preprocessing.

Techniques to merge categories:

Business Logic

Consider a column in the dataset corresponds to "zip codes". There are 182 zip codes in New York state and it is impractical to use each zip code as a separate category. So, to tackle this situation we can merge the zip codes according to localities. This helps to reduce the number of categories and results in meaningful aggregation of zip code.

Frequency

- It is not possible to apply business logic every time. In such cases, perform hashing using the frequency of occurrence.
- To combine levels using their frequency, we first look at the frequency distribution of each level and combine levels having frequency say less than 5% of total observation (can be changed based on distribution).
- This is an effective method to deal with rare levels.
- We can also combine levels by considering the response rate of each level. We can simply combine levels having similar response rates into the same group.

Now we'll see all the concepts explained with the help of a business use case.

There is a libraries required to perform encoding variables:

- FeatureHasher - Feature hashing can be employed in document classification and helps to extract important features from large text corpus.

In [14]:
```
#Select the cell and click on run icon
from sklearn.feature_extraction import FeatureHasher
```

Instruction:

Download the **`vgsales.csv`** dataset file from Course Resources and upload it in the lab using the Up arrow as shown below on the View Tab

Note 10:

- The **`game_df`** is dataframe to store the data imported from the csv as rows and columns table format.
- The encoding is specified as 'UTF-8' as the file type converion of the csv file.
- The **head()** function helps to view first few data present in the dataframe.

In [15]:
```
#Select the cell and click on run icon
game_df = pd.read_csv("vgsales.csv", encoding="utf-8")
game_df.head()
```

Out[15]:

	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
0	1	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
1	2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
2	3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
3	4	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	2.96	33.00
4	5	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37

Note 11:

The **columns** provides the name of the columns present in the **`game_df`** dataframe.

In [16]:
```
#Select the cell and click on run icon
game_df.columns
```

Out[16]:
```
Index(['Rank', 'Name', 'Platform', 'Year', 'Genre', 'Publisher', 'NA_Sales',
       'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales'],
      dtype='object')
```

Observations from the above output:

> The column names such as Rank, Name, Platform, Year, Genre, Publisher, NA_Sales, EU_Sales, JP_Sales, Other_Sales, and Global_Sales present in the **`game_df`** dataframe.

Note 12:

The **shape** represents the number of elements in each dimension and returns a tuple with each index having the number of corresponding elements.

In [17]:
```
#Select the cell and click on run icon
game_df.shape
```

Out[17]:
```
(16598, 11)
```

Observations we can draw from the above output:

> There are 16598 rows and 11 columns present in the **`game_df`** dataframe.

Note 13:

- The **iloc** is integer-index based that helps to extract the required number of rows or columns by specifying the integer index value present in the dataframe.
- Example: iloc[1:7] represents six rows, as it substracts one from seven.

In [18]:
```
#Select the cell and click on run icon
game_df[['Name', 'Platform', 'Year', 'Genre', 'Publisher']].iloc[1:7]
```

Out[18]:

	Name	Platform	Year	Genre	Publisher
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo
5	Tetris	GB	1989.0	Puzzle	Nintendo
6	New Super Mario Bros.	DS	2006.0	Platform	Nintendo

Observations we can draw from the above output:

> There are six rows present in the **`game_df`** dataframe as specified in the iloc.

Note 14:

- The **unique()** function extracts unique values present in a particular variable or dataframe.
- The **len()** function calculates the total number of elements present of a variable or dataframe.
- The **print()** function is used to display the result.

In [19]:
```
#Select the cell and click on run icon
u_generes = np.unique(game_df["Genre"])
print("Total game generes:", len(u_generes))
print(u_generes)
```

```
Total game generes: 12
['Action' 'Adventure' 'Fighting' 'Misc' 'Platform' 'Puzzle' 'Racing'
 'Role-Playing' 'Shooter' 'Simulation' 'Sports' 'Strategy']
```

Observations we can draw from the above output:

> The **'u_generes'** variable stores unique features such as Action, Adventure, Fighting, Misc, Platform, Puzzle, Racing, Role-Playing, Shooter, Simulation, Sports, and Strategy and the total count is 12 present in the **'genre'** column of **`game_df`** dataframe.

Note 15:

- The FeatureHasher() function extracts only important features from the dataset.

In [20]:
```
#Select the cell and click on run icon
fh = FeatureHasher(n_features=6, input_type='string')
hashed_features = fh.fit_transform(game_df["Genre"])
hashed_features = hashed_features.toarray()
new_game_df = pd.concat([game_df[['Name', 'Genre']], pd.DataFrame(hashed_features)], axis=1)

new_game_df.head()
```

Out[20]:

	Name	Genre	0	1	2	3	4	5
0	Wii Sports	Sports	-2.0	2.0	0.0	-2.0	0.0	0.0
1	Super Mario Bros.	Platform	0.0	2.0	2.0	-1.0	1.0	0.0
2	Mario Kart Wii	Racing	-1.0	0.0	0.0	0.0	0.0	-1.0
3	Wii Sports Resort	Sports	-2.0	2.0	0.0	-2.0	0.0	0.0
4	Pokemon Red/Pokemon Blue	Role-Playing	-1.0	1.0	2.0	0.0	1.0	-1.0

Grouping Operations

There are two types of grouping operations:

1. Categorical Grouping: Grouping based on aggregate function using lambda or grouping using a pivot table.

2. Numeric Grouping: Grouping numerical columns using mean and sum functions.

Problem Statements:

You work in HR analytics. You have been given a task to predict the employees who are going to leave the organinzation, so that their replacement process can be started within the available time frame.

The dataset is created by IBM data scientists to uncover the factors that lead to employee attrition and explore important questions such as 'show me a breakdown of distance from home by job role and attrition' or 'compare average monthly income by education and attrition'.

The name of the dataset attributes are self explanatory. Some required attributes are explained below.

- Education
  - Below College
  - College
  - Bachelor
  - Master
  - Doctor
- EnvironmentSatisfaction
  - Low
  - Medium
  - High
  - Very High
- JobInvolvement
  - Low
  - Medium
  - High
  - Very High
- JobSatisfaction
  - Low
  - Medium
  - High
  - Very High
- PerformanceRating
  - Low
  - Good
  - Excellent
  - Outstanding
- RelationshipSatisfaction
  - Low
  - Medium
  - High
  - Very High
- WorkLifeBalance
  - Bad
  - Good
  - Better
  - Best

There are few libraries required to perform encoding variables:

- pandas - It helps to retrieve datasets, handle missing data and in data wrangling.
- Numpy - It helps to perform numerical operations in the dataset.
- seaborn and matplotlib- It helps to data visualization.
- warnings - It helps to segregate the warnings.

Import libraries

In [21]:
```python
#Select the cell and click on run icon
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Instruction:

Download the `HR-Employee-Attrition.csv` dataset file from Course Resources and upload it in the lab using Up arrow shown below View Tab

Read the dataset

Note 16:

- The `df` is dataframe to store the data imported from the csv as rows and columns table format.
- The **head()** function helps to view first few data present in the `df` dataframe.

In [49]:
```python
#Select the cell and click on run icon
df = pd.read_csv("HR-Employee-Attrition.csv")
df.head()
```

Out[49]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7

5 rows × 35 columns

Note 17:

The **columns** provides the name of the columns present in the `df` dataframe.

In [50]:
```python
#Select the cell and click on run icon
df.columns
```

Out[50]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

Observations we can draw from the above output:

> It denotes the name of the columns present in the `df` dataframe.

Note 18:

The **shape** represents the number of elements in each dimension and returns a tuple with each index having the number of corresponding elements.

In [51]:
```python
#Select the cell and click on run icon
df.shape
```

Out[51]:
```
(1470, 35)
```

Observations we can draw from the above output:

> There are 1470 rows and 35 columns present in the `df` dataframe.

Check if there are any null values

In [52]:
```python
#Select the cell and click on run icon
df.isna().sum()
```

Out[52]:
```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

Observations:

> There are no null values present in the `df` dataframe.

Now, print the count of values for each column

In [26]:
```python
for i in df.columns:
    print (i , ":", df[i].value_counts())
    print (" _ "*40)
    print (" _ "*40)
```

```
Age : 35    78
34    77
36    69
31    69
29    68
32    61
30    60
33    58
38    58
40    57
37    50
27    48
28    48
42    46
39    42
45    41
41    40
26    39
44    33
46    33
43    32
50    30
25    26
24    26
49    24
47    24
55    22
51    19
53    19
48    19
54    18
52    18
22    16
56    14
23    14
58    14
21    13
20    11
59    10
19     9
18     8
60     5
57     4
Name: Age, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
Attrition : No     1233
Yes     237
Name: Attrition, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
BusinessTravel : Travel_Rarely        1043
Travel_Frequently     277
Non-Travel            150
Name: BusinessTravel, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
DailyRate : 691     6
408     5
530     5
1329    5
1082    5
        ..
650     1
279     1
316     1
314     1
628     1
Name: DailyRate, Length: 886, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
Department : Research & Development    961
Sales                      446
Human Resources             63
Name: Department, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
DistanceFromHome : 2     211
1     208
10     86
9      85
3      84
7      84
8      80
5      65
4      64
6      59
16     32
11     29
24     28
23     27
29     27
15     26
18     26
26     25
25     25
20     25
28     23
19     22
14     21
12     20
17     20
22     19
13     19
21     18
27     12
Name: DistanceFromHome, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
Education : 3    572
4    398
2    282
1    170
5     48
Name: Education, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
EducationField : Life Sciences       606
Medical             464
Marketing           159
Technical Degree    132
Other                82
Human Resources      27
Name: EducationField, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
EmployeeCount : 1    1470
Name: EmployeeCount, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
EmployeeNumber : 1       1
1391    1
1389    1
1387    1
1383    1
        ..
659     1
657     1
656     1
655     1
2068    1
Name: EmployeeNumber, Length: 1470, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
EnvironmentSatisfaction : 3    453
4    446
2    287
1    284
Name: EnvironmentSatisfaction, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
Gender : Male      882
Female    588
Name: Gender, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
HourlyRate : 66    29
98    28
42    28
48    28
84    28
      ..
31    15
53    14
68    14
38    13
34    12
Name: HourlyRate, Length: 71, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
JobInvolvement : 3    868
2    375
4    144
1     83
Name: JobInvolvement, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
JobLevel : 1    543
2    534
3    218
4    106
5     69
Name: JobLevel, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
JobRole : Sales Executive              326
Research Scientist           292
Laboratory Technician        259
Manufacturing Director       145
Healthcare Representative    131
Manager                      102
Sales Representative          83
Research Director             80
Human Resources               52
Name: JobRole, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
JobSatisfaction : 4    459
3    442
1    289
2    280
Name: JobSatisfaction, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
MaritalStatus : Married     673
Single      470
Divorced    327
Name: MaritalStatus, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
MonthlyIncome : 2342     4
6142     3
2741     3
2559     3
2610     3
         ..
7104     1
2773     1
19513    1
3447     1
4404     1
Name: MonthlyIncome, Length: 1349, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
MonthlyRate : 4223     3
9150     3
9558     2
12858    2
22074    2
         ..
14561    1
2671     1
5718     1
11757    1
10228    1
Name: MonthlyRate, Length: 1427, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
NumCompaniesWorked : 1    521
0    197
3    159
2    146
4    139
7     74
6     70
5     63
9     52
8     49
Name: NumCompaniesWorked, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
Over18 : Y    1470
Name: Over18, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
OverTime : No     1054
Yes     416
Name: OverTime, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
PercentSalaryHike : 11    210
13    209
14    201
12    198
15    101
18     89
17     82
16     78
19     76
22     56
20     55
21     48
23     28
24     21
25     18
Name: PercentSalaryHike, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
PerformanceRating : 3    1244
4     226
Name: PerformanceRating, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
RelationshipSatisfaction : 3    459
4    432
2    303
1    276
Name: RelationshipSatisfaction, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
StandardHours : 80    1470
Name: StandardHours, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
StockOptionLevel : 0    631
1    596
2    158
3     85
Name: StockOptionLevel, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
TotalWorkingYears : 10    202
6     125
8     103
9      96
5      88
7      81
1      81
4      63
12     48
3      42
15     40
16     37
11     36
13     36
21     34
17     33
2      31
14     31
20     30
18     27
19     22
23     22
22     21
24     18
25     14
28     14
26     14
0      11
29     10
31      9
32      9
30      7
33      7
27      7
36      6
34      5
37      4
35      3
40      2
38      1
Name: TotalWorkingYears, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
TrainingTimesLastYear : 2    547
3    491
4    123
5    119
1     71
6     65
0     54
Name: TrainingTimesLastYear, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
WorkLifeBalance : 3    893
2    344
4    153
1     80
Name: WorkLifeBalance, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
YearsAtCompany : 5     196
1     171
3     128
2     127
10    120
4     110
7      90
9      82
8      80
6      76
0      44
11     32
20     27
13     24
15     20
14     18
22     15
12     14
21     14
18     13
16     12
19     11
17      9
24      6
33      5
25      4
26      4
31      3
32      3
27      2
36      2
29      2
23      2
37      1
40      1
34      1
30      1
Name: YearsAtCompany, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
YearsInCurrentRole : 2     372
0     244
7     222
3     135
4     104
8      89
9      67
1      57
6      37
5      36
10     29
11     22
13     14
14     11
12     10
15      8
16      7
17      4
18      2
Name: YearsInCurrentRole, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
YearsSinceLastPromotion : 0     581
1     357
2     159
7      76
4      61
3      52
5      45
6      32
11     24
8      18
9      17
15     13
13     10
12     10
14      9
10      6
Name: YearsSinceLastPromotion, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
YearsWithCurrManager : 2     344
0     263
7     216
3     142
8     107
4      98
1      76
9      64
5      31
6      29
10     27
11     22
12     18
13     14
17      7
15      5
14      5
16      2
Name: YearsWithCurrManager, dtype: int64
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
 _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ 
```

Observations we can draw from the above output:

> It is an imbalanced data as value count in `attrition` column is imbalanced.
>
> > `Over18` is `Y` (Yes) across all the employees and it is not beneficial for further use.
> >
> > > `StandardHours` is 80 which is common for all and it is not beneficial for further use.
> > >
> > > > `EmployeeCount`, `EmployeeNumber` are unique which are also not going to help us predict our end result.

Now, print the unique values in each column to understand the above output more clearly

In [27]:
```python
for i in df.columns:
    print (i , ":", df[i].unique())
    print (" _ "*40)
    print (" _ "*40)
```

```
Age : [41 49 37 33 27 32 59 30 38 36 35 29 31 34 28 22 53 24 21 42 44 46 39 43
 50 26 48 55 45 56 23 51 40 54 58 20 25 19 57 52 47 18 60]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Attrition : ['Yes' 'No']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
BusinessTravel : ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
DailyRate : [1102  279 1373 1392  591 1005 1324 1358  216 1299  809  153  670 1346
  103 1389  334 1123 1219  371  673 1218  419  391  699 1282 1125  691
  477  705  924 1459  125  895  813 1273  869  890  852 1141  464 1240
 1357  994  721 1360 1065  408 1211 1229  626 1434 1488 1097 1443  515
  853 1142  655 1115  427  653  989 1435 1223  836 1195 1339  664  318
 1225 1328 1082  548  132  746  776  193  397  945 1214  111  573 1153
 1400  541  432  288  669  530  632 1334  638 1093 1217 1353  120  682
  489  807  827  871  665 1040 1420  240 1280  534 1456  658  142 1127
 1031 1189 1354 1467  922  394 1312  750  441  684  249  841  147  528
  594  470  957  542  802 1355 1150 1329  959 1033 1316  364  438  689
  201 1427  857  933 1181 1395  662 1436  194  967 1496 1169 1145  630
  303 1256  440 1450 1452  465  702 1157  602 1480 1268  713  134  526
 1380  140  629 1356  328 1084  931  692 1069  313  894  556 1344  290
  138  926 1261  472 1002  878  905 1180  121 1136  635 1151  644 1045
  829 1242 1469  896  992 1052 1147 1396  663  119  979  319 1413  944
 1323  532  818  854 1034  771 1401 1431  976 1411 1300  252 1327  832
 1017 1199  504  505  916 1247  685  269 1416  833  307 1311  128  488
  529 1210 1463  675 1385 1403  452  666 1158  228  996  728 1315  322
 1479  797 1070  442  496 1372  920  688 1449 1117  636  506  444  950
  889  555  230 1232  566 1302  812 1476  218 1132 1105  906  849  390
  106 1249  192  553  117  185 1091  723 1220  588 1377 1018 1275  798
  672 1162  508 1482  559  210  928 1001  549 1124  738  570 1130 1192
  343  144 1296 1309  483  810  544 1062 1319  641 1332  756  845  593
 1171  350  921 1144  143 1046  575  156 1283  755  304 1178  329 1362
 1371  202  253  164 1107  759 1305  982  821 1381  480 1473  891 1063
  645 1490  317  422 1485 1368 1448  296 1398 1349  986 1099 1116 1499
  983 1009 1303 1274 1277  587  413 1276  988 1474  163  267  619  302
  443  828  561  426  232 1306 1094  509  775  195  258  471  799  956
  535 1495  446 1245  703  823 1246  622 1287  448  254 1365  538  525
  558  782  362 1236 1112  204 1343  604 1216  646  160  238 1397  306
  991  482 1176  913 1076  727  885  243  806  817 1410 1207 1442  693
  929  562  608  580  970 1179  294  314  316  654  168  381  217  501
  650  141  804  975 1090  346  430  268  167  621  527  883  954  310
  719  725  715  657 1146  182  376  571  384  791 1111 1243 1092 1325
  805  213  118  676 1252  286 1258  932 1041  859  720  946 1184  436
  589  760  887 1318  625  180  586 1012  661  930  342 1230 1271 1278
  607  130  300  583 1418 1269  379  395 1265 1222  341  868 1231  102
  881 1383 1075  374 1086  781  177  500 1425 1454  617 1085  995 1122
  618  546  462 1198 1272  154 1137 1188  188 1333  867  263  938  129
  616  498 1404 1053  289 1376  231  152  882  903 1379  335  722  461
  974 1126  840 1134  248  955  939 1391 1206  287 1441  109 1066  277
  466 1055  265  135  247 1035  266  145 1038 1234 1109 1089  788  124
  660 1186 1464  796  415  769 1003 1366  330 1492 1204  309 1330  469
  697 1262 1050  770  406  203 1308  984  439  793 1451 1182  174  490
  718  433  773  603  874  367  199  481  647 1384  902  819  862 1457
  977  942 1402 1421 1361  917  200  150  179  696  116  363  107 1465
  458 1212 1103  966 1010  326 1098  969 1167  694 1320  536  373  599
  251  131  237 1429  648  735  531  429  968  879  640  412  848  360
 1138  325 1322  299 1030  634  524  256 1060  935  495  282  206  943
  523  507  601  855 1291 1405 1369  999 1202  285  404  736 1498 1200
 1439  499  205  683 1462  949  652  332 1475  337  971 1174  667  560
  172  383 1255  359  401  377  592 1445 1221  866  981  447 1326  748
  990  405  115  790  830 1193 1423  467  271  410 1083  516  224  136
 1029  333 1440  674 1342  898  824  492  598  740  888 1288  104 1108
  479 1351  474  437  884 1370  264 1059  563  457 1313  241 1015  336
 1387  170  208  671  711  737 1470  365  763  567  486  772  301  311
  584  880  392  148  708 1259  786  370  678  146  581  918 1238  585
  741  552  369  717  543  964  792  611  176  897  600 1054  428  181
  211 1079  590  305  953  478 1375  244  511 1294  196  734 1239 1253
 1128 1336  234  766  261 1194  431  572 1422 1297  574  355  207  706
  280  726  414  352 1224  459 1254 1131  835 1172 1266  783  219 1213
 1096 1251 1394  605 1064 1337  937  157  754 1168  155 1444  189  911
 1321 1154  557  642  801  161 1382 1037  105  582  704  345 1120 1378
  468  613 1023  628]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Department : ['Sales' 'Research & Development' 'Human Resources']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
DistanceFromHome : [ 1  8  2  3 24 23 27 16 15 26 19 21  5 11  9  7  6 10  4 25 12 18 29 22
 14 20 28 17 13]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Education : [2 1 4 3 5]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
EducationField : ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical Degree'
 'Human Resources']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
EmployeeCount : [1]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
EmployeeNumber : [   1    2    4 ... 2064 2065 2068]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
EnvironmentSatisfaction : [2 3 4 1]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Gender : ['Female' 'Male']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
HourlyRate : [ 94  61  92  56  40  79  81  67  44  84  49  31  93  50  51  80  96  78
  45  82  53  83  58  72  48  42  41  86  97  75  33  37  73  98  36  47
  71  30  43  99  59  95  57  76  87  66  55  32  52  70  62  64  63  60
 100  46  39  77  35  91  54  34  90  65  88  85  89  68  69  74  38]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
JobInvolvement : [3 2 4 1]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
JobLevel : [2 1 3 4 5]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
JobRole : ['Sales Executive' 'Research Scientist' 'Laboratory Technician'
 'Manufacturing Director' 'Healthcare Representative' 'Manager'
 'Sales Representative' 'Research Director' 'Human Resources']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
JobSatisfaction : [4 2 3 1]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
MaritalStatus : ['Single' 'Married' 'Divorced']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
MonthlyIncome : [5993 5130 2090 ... 9991 5390 4404]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
MonthlyRate : [19479 24907  2396 ...  5174 13243 10228]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
NumCompaniesWorked : [8 1 6 9 0 4 5 2 7 3]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Over18 : ['Y']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
OverTime : ['Yes' 'No']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
PercentSalaryHike : [11 23 15 12 13 20 22 21 17 14 16 18 19 24 25]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
PerformanceRating : [3 4]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
RelationshipSatisfaction : [1 4 2 3]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
StandardHours : [80]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
StockOptionLevel : [0 1 3 2]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
TotalWorkingYears : [ 8 10  7  6 12  1 17  5  3 31 13  0 26 24 22  9 19  2 23 14 15  4 29 28
 21 25 20 11 16 37 38 30 40 18 36 34 32 33 35 27]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
TrainingTimesLastYear : [0 3 2 5 1 4 6]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
WorkLifeBalance : [1 3 2 4]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
YearsAtCompany : [ 6 10  0  8  2  7  1  9  5  4 25  3 12 14 22 15 27 21 17 11 13 37 16 20
 40 24 33 19 36 18 29 31 32 34 26 30 23]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
YearsInCurrentRole : [ 4  7  0  2  5  9  8  3  6 13  1 15 14 16 11 10 12 18 17]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
YearsSinceLastPromotion : [ 0  1  3  2  7  4  8  6  5 15  9 13 12 10 11 14]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
YearsWithCurrManager : [ 5  7  0  2  6  8  3 11 17  1  4 12  9 10 15 13 16 14]
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Note 19:

The **drop()** function helps to drop the columns which are not important features in the dataset.

In [28]:

```
df=df.drop(['Over18','EmployeeNumber','EmployeeCount','StandardHours'],axis=1)
```

Note 20:

After dropping, check the number of columns and rows present in the dataset.

In [29]:

```
df.shape
```

Out[29]:

```
(1470, 31)
```

Note 21:

The **columns** provides the name of the columns present in the `df` dataframe.

In [30]:

```
df.columns
```

Out[30]:

```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager'],
      dtype='object')
```

Observations from the above output:

> We can see a lot columns which deal with satisfaction, let's observe them closely.

In [31]:

```
df[['RelationshipSatisfaction','JobSatisfaction','EnvironmentSatisfaction','JobInvolvement']]
```

Out[31]:

	RelationshipSatisfaction	JobSatisfaction	EnvironmentSatisfaction	JobInvolvement
0	1	4	2	3
1	4	2	3	2
2	2	3	4	2
3	3	3	4	3
4	4	2	1	3
...	...	...	...	...
1465	3	4	3	4
1466	1	1	4	2
1467	2	2	2	4
1468	4	2	4	2
1469	1	3	2	4

1470 rows × 4 columns

Observations from the above output:

> All the satisfactions are measured from 1 to 4 in incremental order, along with WorkLifeBalance.
>
> > Higher the value, higher the satisfaction.
> >
> > > So, we can combine all the columns that convey satisfaction detail and make them one.

Note 22:

Let us now calculate the mean of all the types of satisfaction. Based on a condition if the mean value is greater than 2.35 then it returns one else zero

In [32]:

```
df['TotalSatisfaction_mean'] = (df['RelationshipSatisfaction']  + df['EnvironmentSatisfaction']
                              + df['JobSatisfaction'] + df['JobInvolvement'] + df['WorkLifeBalance'])/5

def Satif(df) :
    if  df['TotalSatisfaction_mean'] > 2.35 :
        return 1
    else :
        return 0


df['Satif'] = df.apply(lambda df:Satif(df) ,axis = 1)
df['Satif']
```

Out[32]:

```
0       0
1       1
2       1
3       1
4       1
       ..
1465    1
1466    0
1467    1
1468    1
1469    1
Name: Satif, Length: 1470, dtype: int64
```

In [33]:

```
df.shape
```

Out[33]:

```
(1470, 33)
```

Let's keep all the columns for now.

Note 23:

Let us now create a separate column for job satisfaction.

In [34]:

```
df['JobSatisf_mean'] = (df['JobSatisfaction'] + df['JobInvolvement']) / 2
```

In [35]:

```
df.shape
```

Out[35]:

```
(1470, 34)
```

Note 24:

We can understant people who switch companies frequently have more tendency to leave a company. So lets create a new column called `MovingPeople`.

In [36]:

```
def MovingPeople(df) :
    if df['NumCompaniesWorked'] > 4:
        return 1
    else:
        return 0
df['MovingPeople'] = df.apply(lambda df:MovingPeople(df), axis = 1)
df['MovingPeople']
```

Out[36]:

```
0       1
1       0
2       1
3       0
4       1
       ..
1465    0
1466    0
1467    0
1468    0
1469    0
Name: MovingPeople, Length: 1470, dtype: int64
```

In [37]:

```
df.shape
```

Out[37]:

```
(1470, 35)
```

Note 25:

Create a column using `DistanceFromHome` column

In [38]:

```
def LongDis(df) :
    if  df['DistanceFromHome'] > 11:
        return 1
    else :
        return 0
df['LongDis'] = df.apply(lambda df:LongDis(df) ,axis = 1)
df['LongDis']
```

Out[38]:

```
0       0
1       0
2       0
3       0
4       0
       ..
1465    1
1466    0
1467    0
1468    0
1469    0
Name: LongDis, Length: 1470, dtype: int64
```

Note 26:

Create a column using `TrainingTimesLastYear` column

In [39]:

```
def MiddleTraining(df) :
    if df['TrainingTimesLastYear'] >= 3 and df['TrainingTimesLastYear'] <= 6:
        return 1
    else:
        return 0
df['MiddleTraining'] = df.apply(lambda df:MiddleTraining(df) ,axis = 1)
```

Note 27:

Create a column to view number of years worked in each company

In [40]:

```
df['Time_in_each_comp'] = (df['Age'] - 20) / ((df)['NumCompaniesWorked'] + 1)
df['Time_in_each_comp']
```

Out[40]:

```
0        2.333333
1       14.500000
2        2.428571
3        6.500000
4        0.700000
          ...
1465     3.200000
1466     3.800000
1467     3.500000
1468     9.666667
1469     4.666667
Name: Time_in_each_comp, Length: 1470, dtype: float64
```

In [41]:

```
df.shape
```

Out[41]:

```
(1470, 38)
```

Note 28:

Let us now understand the columns that are categorical and numerical in the `df` dataframe.

In [42]:

```
numeric_df= df.select_dtypes(include=[np.number])

categoric_df=df.select_dtypes(exclude=[np.number])
```

In [43]:

```
numericcol=numeric_df.columns.tolist()
categorycol=categoric_df.columns.tolist()

print ("Category :",categorycol)
print ("\n Numeric :",numericcol)
```

```
Category : ['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatu
s', 'OverTime']

 Numeric : ['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EnvironmentSatisfaction', 'HourlyRate', 'JobI
nvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'PercentSalar
yHike', 'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears', 'TrainingTime
sLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCur
rManager', 'TotalSatisfaction_mean', 'Satif', 'JobSatisf_mean', 'MovingPeople', 'LongDis', 'MiddleTraining', 'T
ime_in_each_comp']
```

Observations from the above output:

> 'DailyRate', 'HourlyRate' can be dropped as we have monthly rate
>
> > Attributes 'DistanceFromHome', 'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobSatisfaction', 'NumCompaniesWorked', 'RelationshipSatisfaction', and 'TrainingTimesLastYear' can be removed as these are considered in newly created column attributes

Note 29:

Let us now drop unnecessary columns present in the `df` dataframe.

In [44]:

```
df=df.drop(['DailyRate', 'DistanceFromHome', 'EnvironmentSatisfaction',
            'HourlyRate', 'JobInvolvement',
            'JobSatisfaction',  'NumCompaniesWorked',
            'RelationshipSatisfaction', 'TrainingTimesLastYear'],axis=1)
```

In [45]:

```
df.shape
```

Out[45]:

```
(1470, 29)
```

Note 30:

After extracting the important features from the dataframe.Now, we can convert the categorical features into numerical using **one hot encoding technique**.

In [46]:

```
data = pd.get_dummies(df, columns=categorycol, drop_first=True)
print(data.columns)
print(data.shape)
```

```
Index(['Age', 'Education', 'JobLevel', 'MonthlyIncome', 'MonthlyRate',
       'PercentSalaryHike', 'PerformanceRating', 'StockOptionLevel',
       'TotalWorkingYears', 'WorkLifeBalance', 'YearsAtCompany',
       'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager',
       'TotalSatisfaction_mean', 'Satif', 'JobSatisf_mean', 'MovingPeople',
       'LongDis', 'MiddleTraining', 'Time_in_each_comp', 'Attrition_Yes',
       'BusinessTravel_Travel_Frequently', 'BusinessTravel_Travel_Rarely',
       'Department_Research & Development', 'Department_Sales',
       'EducationField_Life Sciences', 'EducationField_Marketing',
       'EducationField_Medical', 'EducationField_Other',
       'EducationField_Technical Degree', 'Gender_Male',
       'JobRole_Human Resources', 'JobRole_Laboratory Technician',
       'JobRole_Manager', 'JobRole_Manufacturing Director',
       'JobRole_Research Director', 'JobRole_Research Scientist',
       'JobRole_Sales Executive', 'JobRole_Sales Representative',
       'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_Yes'],
      dtype='object')
(1470, 43)
```

In [47]:

```
data.head()
```

Out[47]:

	Age	Education	JobLevel	MonthlyIncome	MonthlyRate	PercentSalaryHike	PerformanceRating	StockOptionLevel	TotalWorkingYears	Work
0	41	2	2	5993	19479	11	3	0	8	
1	49	1	2	5130	24907	23	4	1	10	
2	37	2	1	2090	2396	15	3	0	7	
3	33	4	1	2909	23159	11	3	0	8	
4	27	1	1	3468	16632	12	3	1	6	

5 rows × 43 columns

Observations from the above output:

> We can see that all the variables present in the dataframe are in the numerical format.

`Note: The rest of the encoding methods will be used in further lessons`

The further operations come under feature Selection

Note: In this lesson, we saw the use of the feature engineering methods, but in the next lesson we are going to use one of these methods as a sub component for "Exploratory Data Analysis".

In [ ]: