Example of an end-to-end machine learning project in Data Science for beginners.

"This is a tad bit lengthy article since I had to make sure I do not miss out on any of the important points to build a complete ML project but you have definitely made the right choice to invest your valuable time in something informative here."

Today I am going to write about a complete end-to-end project for HR Analytics which should serve as a guiding path for many Data Science aspirants. When I began my journey into studying Data Science, I was overwhelmed by the content that is available online and most of them scattered into chunks emphasizing on a deeper knowledge that a newbie will most likely not be able to comprehend or connect through. I agree that there are already many projects available into some deep web beneath multiple clicks and paths hidden like a core of an onion protected by multiple layers. But here I am just trying to do my bit in making things easier for a new comer to understand the basic architecture that's required in the real world for creating a Data Science project.

So, without any further ado please allow me to explain the agenda for this blog post. In this article, I have jotted down all the techniques in the form of sub-topics that I will be explaining one by one. And those pointers are as follows:

1. Problem Definition
2. Data Analysis
3. EDA
4. Pre-processing Data
5. Building Machine Learning Models
6. Concluding Remarks

Let's start with the problem definition or a short introduction on the project that I have chosen to elaborate and why it was made in the first place.

1. Problem Definition

The project I am using for this article is for HR Analytics which is basically a fictional dataset that was created by the Data Scientists at IBM. You can find the dataset here on the Kaggle website. Basically, every year a lot of companies hire a number of employees. The companies invest time and money in training those employees, not just this but there are training programs within the companies for their existing employees as well. The aim of these programs is to increase the effectiveness of their employees. But where does HR Analytics fit in this? and is it just about improving the performance of employees?

Human Resource Analytics (HR Analytics) is an area in the field of analytics that refers to applying analytic processes to the human resource department of an organization in the hope of improving employee performance and therefore getting a better return on investment.

!

Attrition affecting companies is a major problem since high employee attrition is its cost to an organization. Job postings, hiring processes, paperwork and new hire trainings are some of the common expenses of losing employees and replacing them. Additionally, regular employee turnover prohibits an organization from increasing its collective knowledge base and experience over time. This is especially concerning if your business is customer-facing, as customers often prefer to interact with familiar people. Errors and issues are more likely if you constantly have new workers too.

Therefore the major goal of this project is to identify the “Attrition” rate as a simple Yes or a No tag making this to be a classification problem!

```
import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings("ignore")
import joblib

import pandas as pd
import numpy as np
import seaborn as sns
import missingno
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import zscore

from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

First we are going to import all the necessary dependencies here that will be used in our project and obtain the rest as and when required. Before we begin with any process, we need to get the dataset in our Jupyter Notebook that can be achieved by a single step.

```
df = pd.read_csv("WA_Fn-UseC_-HR-Employee-Attrition.csv")
```

This gives us our entire dataset stored in the variable name “df” for our dataframe.

2. Data Analysis

For data analysis part we can simply eye ball the contents for our dataset trying to make sense of some columns, it’s related values and anything that comes to your mind.

```
df
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	RelationsI
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	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1465	36	No	Travel_Frequently	884	Research & Development	23	2	Medical	1	2061	...	
1466	39	No	Travel_Rarely	613	Research & Development	6	1	Medical	1	2062	...	
1467	27	No	Travel_Rarely	155	Research & Development	4	3	Life Sciences	1	2064	...	
1468	49	No	Travel_Frequently	1023	Sales	2	3	Medical	1	2065	...	
1469	34	No	Travel_Rarely	628	Research & Development	8	3	Medical	1	2068	...	

1470 rows × 35 columns

In the above line of code we can see that the total number of rows present in our data is 1470 and the total number of columns are 35. Since it is a dataset with reasonably higher number of rows and columns the visualization gets truncated.

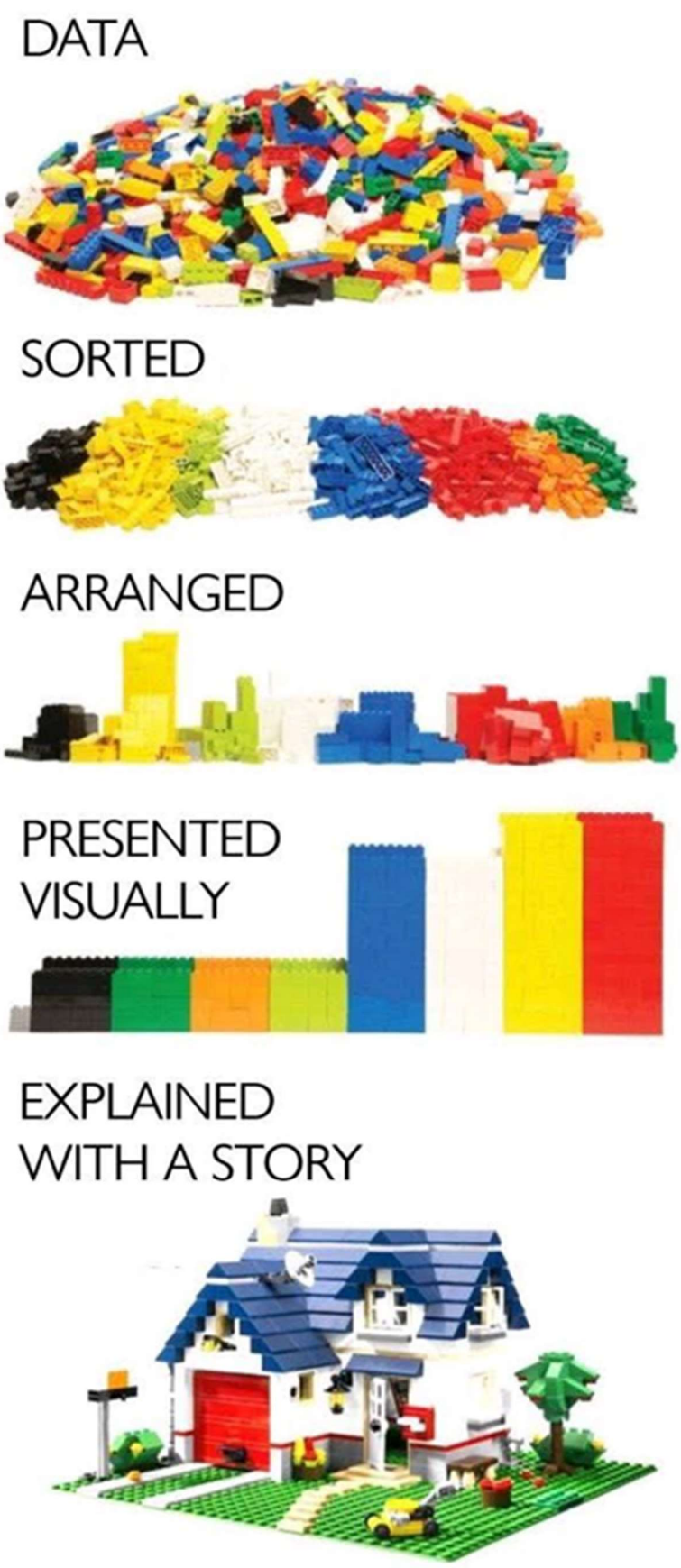
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

You can write this piece of code and then try to check the dataframe “df” again. It will show you the entire row and column information on your Jupyter Notebook directly.

3. EDA

EDA also known as Exploratory Data Analysis is considered the most important aspect in Data Science by many Data Scientist including me. After following a huge number of expert Data Scientist on various platforms I can confirm one thing that it boils down to a single important thing of conveying a story on how you were able to achieve each and every step via your code showing the provided problem statement, the observation, the challenges faced and what was done to tackle or rectify those issues.

Building a good model comes only when you understand clearly what you are doing and why



you are doing it. Making sure that you have a clean data in proper processed format to feed into your model and get appropriate result. Because no amount of Machine Learning model usage and hyper parameter tuning is going to help if you have not invested time to sort out and fix your data that's the only input you have at hand.

There is this famous saying amongst Data Scientist and also people who work with data that goes something like "Garbage in Garbage out" simply translating to mean, if you are trying to build an automated model or make sense out of data that you have you need to clean it and work on it thoroughly before you even start utilizing it further. I cannot stress enough on the process of clean data collection than anything but in real world there is no such thing as clean data. That's where data analysis comes into picture and using multiple methods from data cleaning to data pre-processing and then data engineering brings us closer to get our desired label prediction.

Enough of the story telling let me show you the code since talk is cheap, but also necessary at times to explain what really is going on. The first thing I am going to take a look at is the

missing data information in our dataset by using the codes below.

```
df.isna().sum()
```

```
missingno.bar(df, figsize = (25,5), color="tab:green")
```

These two codes give us the missing values information in a tabular and visual format that looks something like this.

```
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
```

Now that we were able to confirm our dataset being free of any missing data we will drop any duplicates that might be present using the code below.

```
print("We had {} Rows and {} Columns before dropping duplicates.".format(df.shape[0], df.shape[1]))
df.drop_duplicates(inplace=True)
print("We have {} Rows and {} Columns after dropping duplicates.".format(df.shape[0], df.shape[1]))
```

```
We had 1470 Rows and 35 Columns before dropping duplicates.
We have 1470 Rows and 35 Columns after dropping duplicates.
```

With the `drop_duplicates` option I was trying to get rid of all the duplicate data present in our dataset. However, we can see that there are no duplicate data existing in our dataset. Next, we move on to using the describe method to take a look at the count value, mean data, standard

deviation information and the minimum, maximum, 25% quartile, 50% quartile and 75% quartile details. As the describe method works best for numeric data all the object (text) type data gets ignored. Take a look at the below code and you will get an idea on how to use it.

```
df.describe().T
```

```
# visualizing the statistical description of numeric datatype columns

plt.figure(figsize = (15,9))
sns.heatmap(round(df.describe()[1:].transpose(),2), linewidth = 2, annot= True, fmt = ".4f", cmap="hot")
plt.title("Satistical Report of Numerical Columns")
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)
plt.show()
```

Once you have used the code the output provided are in transpose format to accommodate all the columns from our dataset in tabular as well as visual format.

	count	mean	std	min	25%	50%	75%	max
Age	1470.0	36.923810	9.135373	18.0	30.00	36.0	43.00	60.0
DailyRate	1470.0	802.485714	403.509100	102.0	465.00	802.0	1157.00	1499.0
DistanceFromHome	1470.0	9.192517	8.106864	1.0	2.00	7.0	14.00	29.0
Education	1470.0	2.912925	1.024165	1.0	2.00	3.0	4.00	5.0
EmployeeCount	1470.0	1.000000	0.000000	1.0	1.00	1.0	1.00	1.0
EmployeeNumber	1470.0	1024.865306	602.024335	1.0	491.25	1020.5	1555.75	2068.0
EnvironmentSatisfaction	1470.0	2.721769	1.093082	1.0	2.00	3.0	4.00	4.0
HourlyRate	1470.0	65.891156	20.329428	30.0	48.00	66.0	83.75	100.0
JobInvolvement	1470.0	2.729932	0.711561	1.0	2.00	3.0	3.00	4.0
JobLevel	1470.0	2.063946	1.106940	1.0	1.00	2.0	3.00	5.0
JobSatisfaction	1470.0	2.728571	1.102846	1.0	2.00	3.0	4.00	4.0
MonthlyIncome	1470.0	6502.931293	4707.956783	1009.0	2911.00	4919.0	8379.00	19999.0
MonthlyRate	1470.0	14313.103401	7117.786044	2094.0	8047.00	14235.5	20461.50	26999.0
NumCompaniesWorked	1470.0	2.693197	2.498009	0.0	1.00	2.0	4.00	9.0
PercentSalaryHike	1470.0	15.209524	3.659938	11.0	12.00	14.0	18.00	25.0
PerformanceRating	1470.0	3.153741	0.360824	3.0	3.00	3.0	3.00	4.0
RelationshipSatisfaction	1470.0	2.712245	1.081209	1.0	2.00	3.0	4.00	4.0
StandardHours	1470.0	80.000000	0.000000	80.0	80.00	80.0	80.00	80.0
StockOptionLevel	1470.0	0.793878	0.852077	0.0	0.00	1.0	1.00	3.0
TotalWorkingYears	1470.0	11.279592	7.780782	0.0	6.00	10.0	15.00	40.0
TrainingTimesLastYear	1470.0	2.799320	1.289271	0.0	2.00	3.0	3.00	6.0
WorkLifeBalance	1470.0	2.761224	0.706476	1.0	2.00	3.0	3.00	4.0
YearsAtCompany	1470.0	7.008163	6.126525	0.0	3.00	5.0	9.00	40.0
YearsInCurrentRole	1470.0	4.229252	3.623137	0.0	2.00	3.0	7.00	18.0
YearsSinceLastPromotion	1470.0	2.187755	3.222430	0.0	0.00	1.0	3.00	15.0
YearsWithCurrManager	1470.0	4.123129	3.568136	0.0	2.00	3.0	7.00	17.0

When we are able to draw insights from the describe method we can take a look at the datatype information using the code below and that shall give us the list of all the columns marking them to be either integer, float or object datatype depending on the values present inside the columns.

Code:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 413.4+ KB
```

This is the output that I get explaining the datatypes of all the columns present in our dataframe. We also get an opportunity to drop or remove any unwanted columns from the dataframe here.

One of the things that I like to do is separate the object datatype and numeric datatype values that allows for easier processing in further steps. The code to do that is a simple for loop usage.

```
object_datatype = []
for x in df.dtypes.index:
    if df.dtypes[x] == 'O':
        object_datatype.append(x)
object_datatype
```

```
integer_datatype = []
for x in df.dtypes.index:
    if df.dtypes[x] == 'int64':
        integer_datatype.append(x)
integer_datatype
```

	Unique Values
Age	43
Attrition	2
BusinessTravel	3
DailyRate	886
Department	3
DistanceFromHome	29
Education	5
EducationField	6
EnvironmentSatisfaction	4
Gender	2
HourlyRate	71
JobInvolvement	4
JobLevel	5
JobRole	9
JobSatisfaction	4
MaritalStatus	3
MonthlyIncome	1349
MonthlyRate	1427
NumCompaniesWorked	10
OverTime	2
PercentSalaryHike	15
PerformanceRating	2
RelationshipSatisfaction	4
StockOptionLevel	4
TotalWorkingYears	40
TrainingTimesLastYear	7
WorkLifeBalance	4
YearsAtCompany	37
YearsInCurrentRole	19
YearsSinceLastPromotion	16
YearsWithCurrManager	18

This allows us to store the column names in a list format within the variables namely object_datatype and integer_datatype.

After I have bifurcated the datatype column names in two separate lists, we will take a look at the overall unique values for all the columns and then the data numbers for only object datatype columns using the below codes.

```
df.nunique().to_frame("Unique Values
```

```
for col in object_datatype:
    print(col)
    print(df[col].value_counts())
    print("="*120)
```

These line of codes provide us the output where we get an entire list of column names with unique data covered in the dataset rows providing a numerical data and then a description of those values for categorical object datatype columns.

Considering the separation of object data I then take a visual on how many rows or count of rows these values cover in our data set. Usage of various visualization techniques allows me to optimize and analyse the columns further. It gives me an idea as to where data pre processing will be needed and where removal of those data will benefit. Honestly all this can only be acquired from practising on different projects and as everyone says

the more you work the more you acquire knowledge in that field working like a 6th sense in such project creation. I am giving this example here but it does not mean that these are the only steps when it comes to creating a project. The architecture or the backbone of the project will remain the same however depending on what data you are working on the usage of techniques with all differ.

For example in this project I did not get any missing value therefore I did not worry about handling them but there are datasets which have a lot of missing data which are then filled using various methods and are at times even discarded as a last resort if it is only going to make our machine learning model biased towards one data value or category.

Let me go head and list down all the visualization codes and their output for your reference.

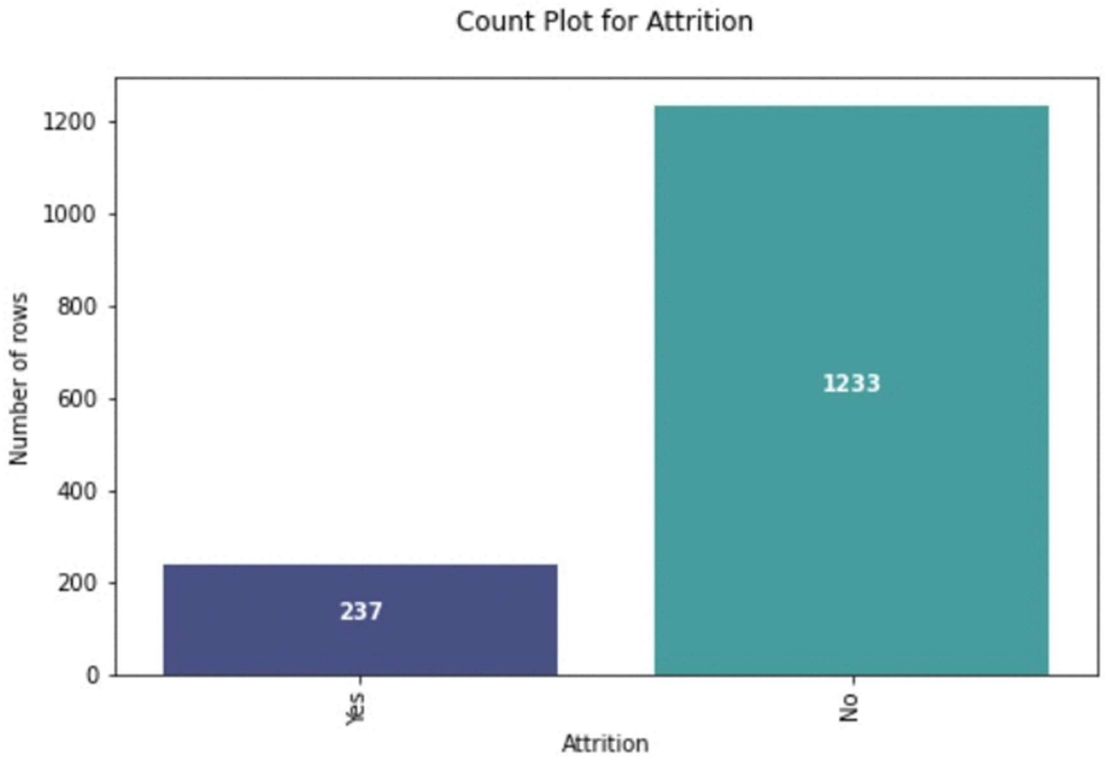
Code:

```
plt.figure(figsize=(8,5))
col_name = 'Attrition'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```

Output:



Code:

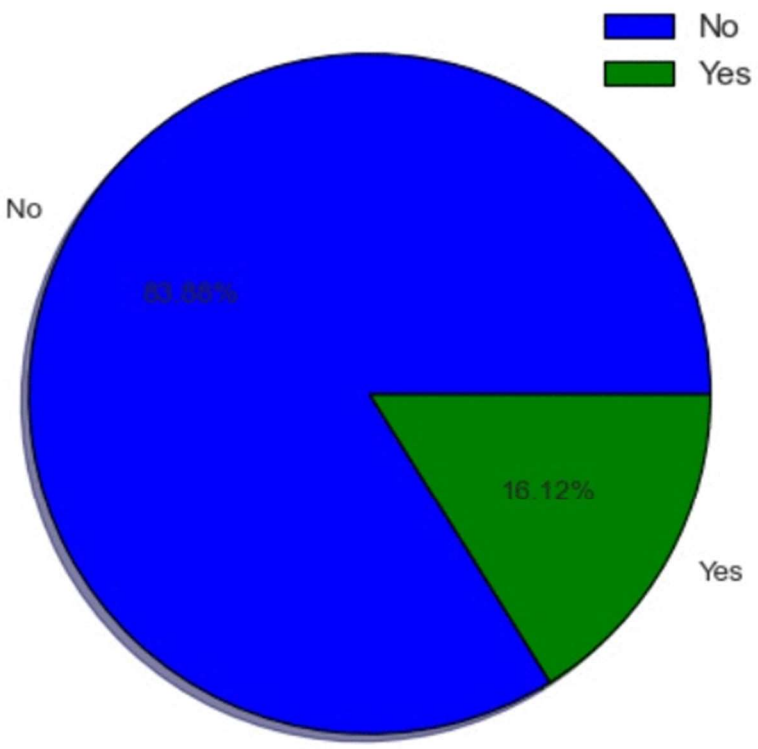
```
low_unique_val = ['Attrition', 'Gender', 'OverTime', 'PerformanceRating', 'BusinessTravel', 'Department',
                  'MaritalStatus', 'EnvironmentSatisfaction', 'JobInvolvement', 'JobSatisfaction',
                  'RelationshipSatisfaction', 'StockOptionLevel', 'WorkLifeBalance', 'Education', 'JobLevel',
                  'EducationField', 'TrainingTimesLastYear', 'JobRole']

def generate_pie(x):
    plt.style.use('seaborn-white')
    plt.figure(figsize=(10,8))
    plt.pie(x.value_counts(), labels=x.value_counts().index, shadow=True, autopct='%1.2f%%')
    plt.legend(prop={'size':14})
    plt.axis('equal')
    plt.tight_layout()
    fig = plt.gcf()
    fig.set_size_inches(5,5)
    return plt.show()

plt.style.use('classic')
for i in df[low_unique_val]:
    print("{} column category details ->".format(i))
    generate_pie(df[i])
```

Output:

Attrition column category details ->



Codes:

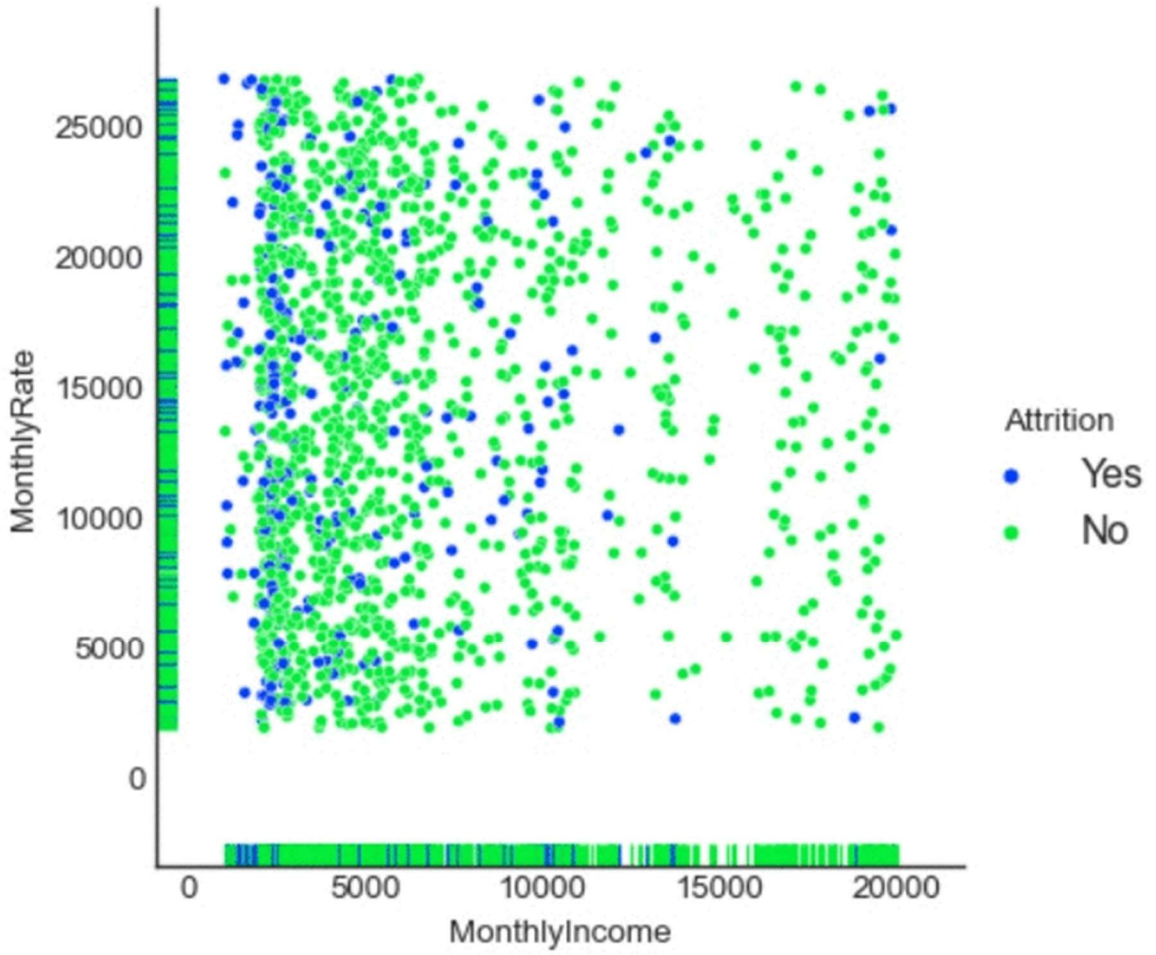
```
plt.style.use('seaborn-bright')

sns.relplot(data=df, x='MonthlyIncome', y='MonthlyRate', hue='Attrition')
sns.rugplot(data=df, x='MonthlyIncome', y='MonthlyRate', hue='Attrition', legend=False)
```

```
mul_val_col = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate', 'JobLevel', 'NumCompaniesWorked',
               'PercentSalaryHike', 'TotalWorkingYears', 'YearsAtCompany', 'YearsInCurrentRole',
               'YearsSinceLastPromotion', 'YearsWithCurrManager']

for z in df[mul_val_col]:
    fig = plt.figure(figsize=(5,5))
    ax=sns.kdeplot(df.loc[(df['Attrition'] == 'No'), z], color='b', shade=True, label='No attrition')
    ax=sns.kdeplot(df.loc[(df['Attrition'] == 'Yes'), z], color='r', shade=True, label='Attrition')
    plt.title('Attrition rate with respect to {}'.format(z))
    plt.legend()
    plt.show()
```

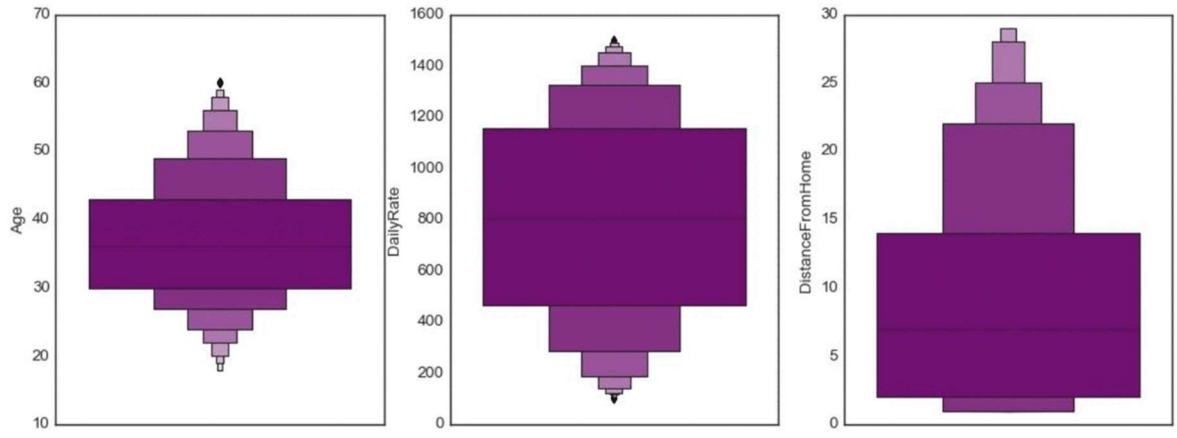
Output:



Code:

```
fig, ax = plt.subplots(ncols=3, nrows=8, figsize=(15,50))
index = 0
ax = ax.flatten()
for col, value in df[integer_datatype].items():
    sns.boxenplot(y=col, data=df, ax=ax[index], color="purple")
    index += 1
plt.show()
```

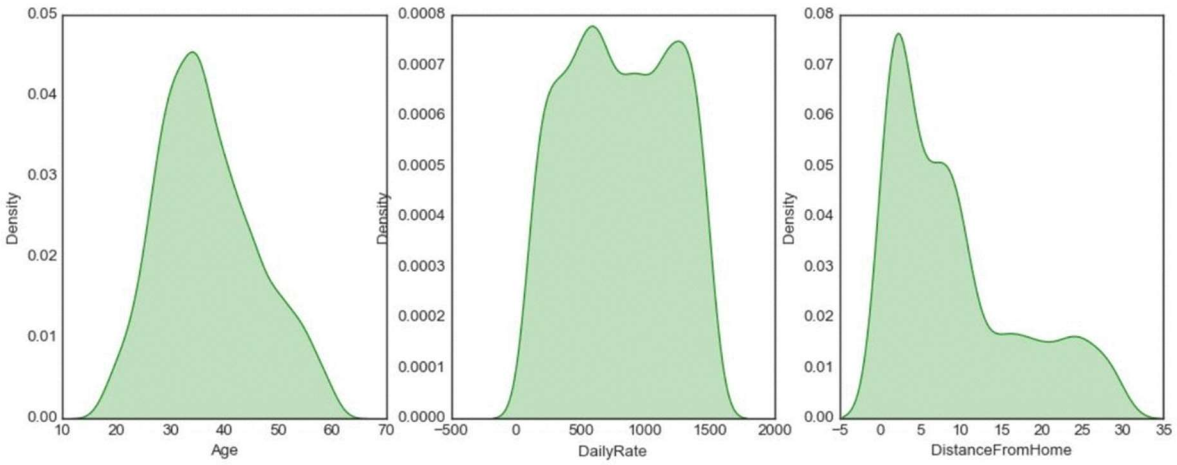
Output:



Code:

```
fig, ax = plt.subplots(ncols=3, nrows=8, figsize=(15,50))
index = 0
ax = ax.flatten()
for col, value in df[integer_datatype].items():
    sns.distplot(value, ax=ax[index], hist=False, color="g", kde_kws={"shade": True})
    index += 1
plt.show()
```

Output:



You can see that with the help of above codes and getting the outputs I was able to take a look at all the column values/counts, the boxen plots gave me a view on the presence of outliers and the distribution plots showed me the skewness information that will needed to be treated. These are like the challenges that will need to be dealt with before I even think of building my

Classification Machine Learning models.

4. Pre-processing Data

In the pre-processing step I am going to tackle all the miss fits and fix them one by one starting with the problem that out dataset has object datatype values where as our Machine Learning models can only understand numeric values. I am making use of the encoding methods to convert all the object datatype values. For our label I am using Label Encoder while for other categorical feature columns I am using the Ordinal Encoder. Instead of the Ordinal Encoder I could have used The One Hot Encoder but as I mentioned it is all about preference with trial and errors. The One Hot Encoder method increases the number columns while application of Ordinal Encoder on data values offering an order deems a better option to me. I have also seen many people apply Label Encoder on feature columns as well and it does not make any sense to me since the name itself says Label Encoder how much of a specification is required to understand that it is only for our label(s) columns. Hope my readers won't make the same mistake!

Code:

```
# Label Encoding

le = LabelEncoder()
df["Attrition"] = le.fit_transform(df["Attrition"])
df.head()
```

```
# Ordinal Encoding

oe = OrdinalEncoder()

def ordinal_encode(df, column):
    df[column] = oe.fit_transform(df[column])
    return df

oe_col = ['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'OverTime']
df=ordinal_encode(df, oe_col)
df.head()
```

Output:

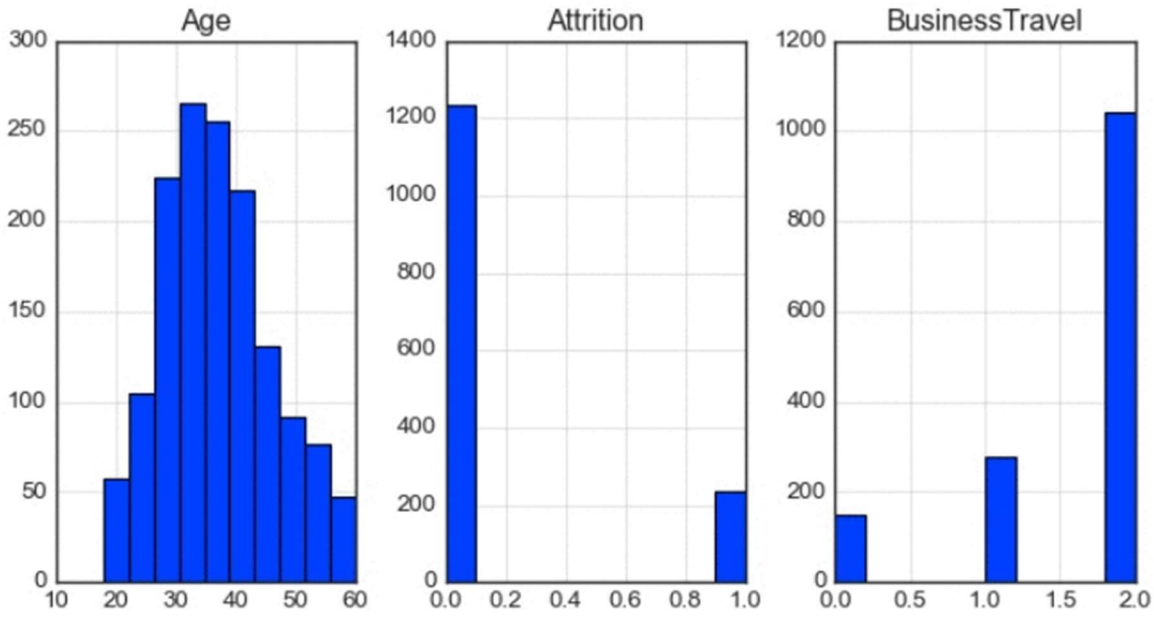
	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfaction	Gender	HourlyRate	JobIn
0	41	1	2.0	1102	2.0	1	2	1.0	2	0.0	94	
1	49	0	1.0	279	1.0	8	1	1.0	3	1.0	61	
2	37	1	2.0	1373	1.0	2	2	4.0	4	1.0	92	
3	33	0	1.0	1392	1.0	3	4	1.0	4	0.0	56	
4	27	0	2.0	591	1.0	2	1	3.0	1	1.0	40	

After I have encoded all the columns in our dataset, I am using a Histogram to view the data distribution. Since Histograms only consider numeric data it should be able to identify all the information from our encoded dataframe.

Code:

```
df.hist(figsize=(20,35))
plt.show()
```

Output:

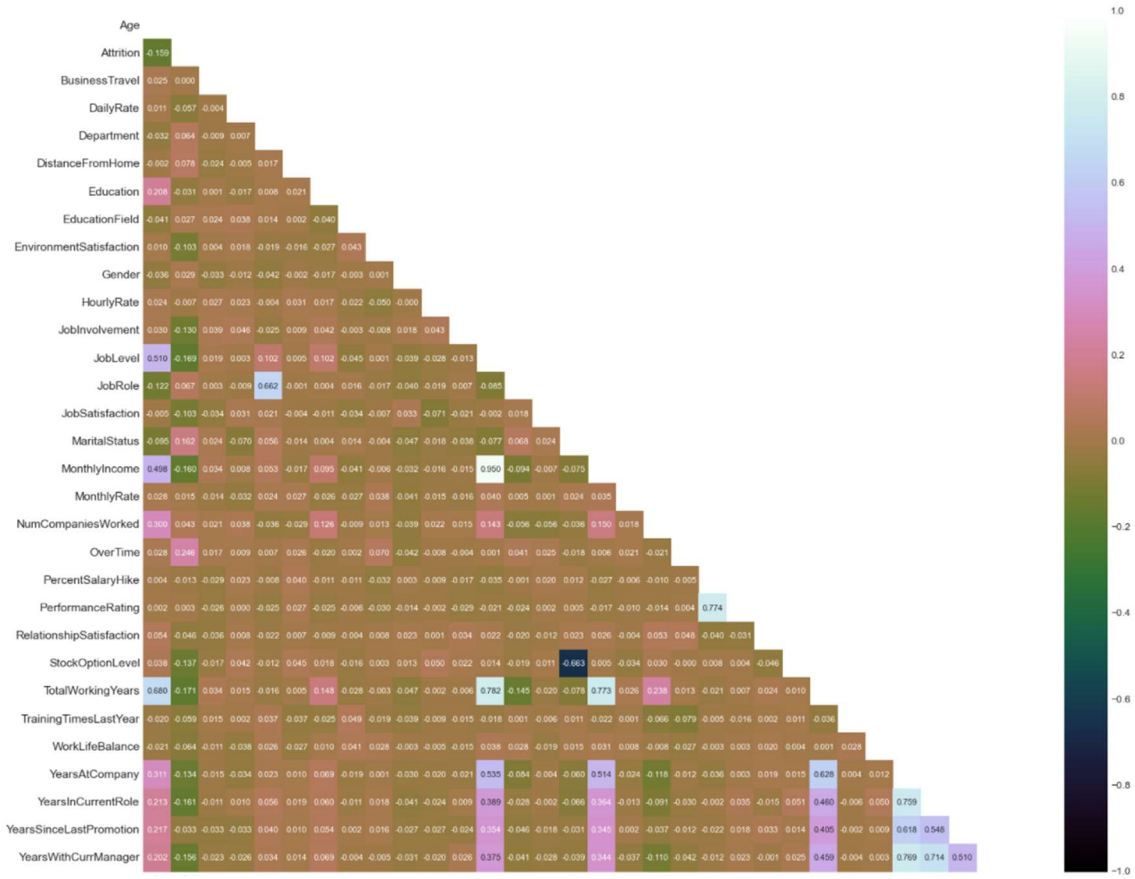


I now feel the need to check for correlation details in our dataset through a Heatmap. For those who still feel a confusion on correlation details let me break it down in two simple points that there are Positive correlation - A correlation of +1 indicates a perfect positive correlation, meaning that both variables move in the same direction together and Negative correlation - A correlation of –1 indicates a perfect negative correlation, meaning that as one variable goes up, the other goes down. The code to see this information is displayed below.

Code:

```
upper_triangle = np.triu(df.corr())
plt.figure(figsize=(26,18))
sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, square=True, fmt='0.3f',
            annot_kws={'size':10}, cmap="cubehelix", mask=upper_triangle)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

Output:

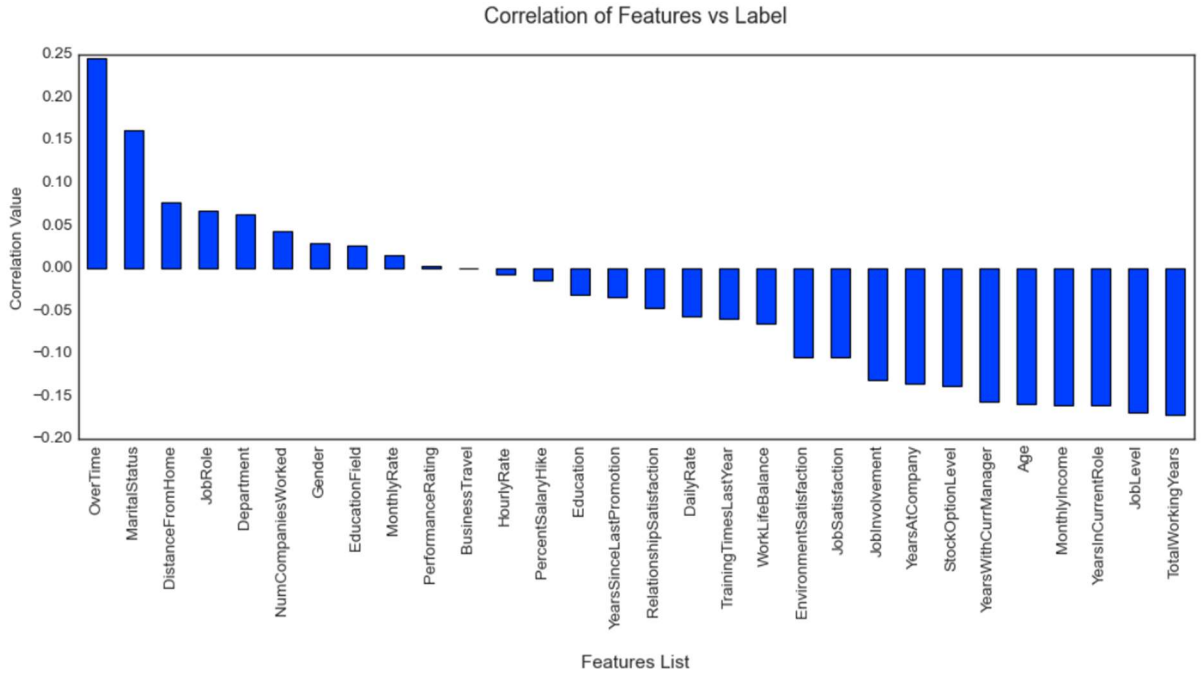


Trust me even on the Jupyter Notebook it looks all tiny due to the high number of columns. So, in such scenarios I majorly look at the colours to understand if there is any multicollinearity issue among the feature columns and if there is still any column that I can drop. But to clearly view the correlation between our label and feature columns I use a Bar Plot comparison and you can find it's code here.

Code:

```
df_corr = df.corr()
plt.figure(figsize=(15,5))
df_corr['Attrition'].sort_values(ascending=False).drop('Attrition').plot.bar()
plt.title("Correlation of Features vs Label\n", fontsize=16)
plt.xlabel("\nFeatures List", fontsize=14)
plt.ylabel("Correlation Value", fontsize=12)
plt.show()
```

Output:



In the above Bar Plot we are able to clearly define the feature columns that are positively correlated with our label and the feature columns that are negatively correlated with our label. Now coming back to the outlier and skewness concern in our dataset I will be using the Z score and Log transformation methods.

Code:

```
z = np.abs(zscore(df))
threshold = 3
df1 = df[(z<3).all(axis = 1)]

print ("Shape of the dataframe before removing outliers: ", df.shape)
print ("Shape of the dataframe after removing outliers: ", df1.shape)
print ("Percentage of data loss post outlier removal: ", (df.shape[0]-df1.shape[0])/df.shape[0]*100)

df=df1.copy() # reassigning the changed dataframe name to our original dataframe name
```

```
Shape of the dataframe before removing outliers:  (1470, 31)
Shape of the dataframe after removing outliers:  (1387, 31)
Percentage of data loss post outlier removal:  5.646258503401361
```

As for the usage of Z score, I was able to lose only about 5% of data but when I used the IQR method it took away I believe 30% of the data. And as Data Scientist retaining valuable data always takes priority and fixing it rather than simply deleting it unless it is the last resort. After this I am using the Log transformation to deal with the skewness since the acceptable range lies between +/-0.5 value for each column.

Code:

```
for col in integer_datatype:
    if df.skew().loc[col]>0.55:
        df[col]=np.log1p(df[col])
```

After dealing with the data concerns I will then split our columns into feature and label. I am storing the feature columns in X and the target label column in the Y variable.

Code:

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

But there was an imbalance between the label classes. If you would notice the value displayed in the count plot earlier, there was a huge difference between the “Yes” and “No” data. Therefore, I will have to resolve it as the imbalance can make our machine learning model biased towards the “No” value.

Code:

```
oversample = SMOTE()
X, Y = oversample.fit_resample(X, Y)
```

Then I will also scale the feature columns that is stored in the X variable to avoid any kind of biasness over column values. Some integers cover thousands place and some cover hundreds or tens place then it can make the machine learning model assume the column with thousands place has a higher importance when in real that won’t be true due to difference in unit range.

Code:

```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

I would like to share a simple piece of code that allows us to choose a fitting random state for

the machine learning models.

Code:

```
maxAccu=0
maxRS=0

for i in range(1, 1000):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=i)
    lr=LogisticRegression()
    lr.fit(X_train, Y_train)
    pred = lr.predict(X_test)
    acc_score = (accuracy_score(Y_test, pred))*100

    if acc_score>maxAccu:
        maxAccu=acc_score
        maxRS=i

print("Best accuracy score is", maxAccu,"on Random State", maxRS)
```

Then I will use the train test split to bifurcate our entire data set into training data and testing data. Here I am using 75% data for training purpose and 25% data for testing purpose. Some people provide training and test data separately as well and hence it completely depends on you how you want to use this step.

Code:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=75)
```

Now at this critical step before building my machine learning model I take a look at the importance of my feature columns. This gives me an insight on how the feature columns are involved and what kind of weightage they have in predicting my target label.

Code:

```
rf=RandomForestClassifier()
rf.fit(X_train, Y_train)
importances = pd.DataFrame({'Features':X.columns, 'Importance':np.round(rf.feature_importances_,3)})
importances = importances.sort_values('Importance', ascending=False).set_index('Features')
importances.plot.bar(color='teal')
importances
```

Output:

Features	Importance
OverTime	0.148
JobLevel	0.072
StockOptionLevel	0.056
MaritalStatus	0.048
JobInvolvement	0.043
TotalWorkingYears	0.042
MonthlyIncome	0.038
JobSatisfaction	0.038
Department	0.034
Age	0.031
YearsWithCurrManager	0.030
YearsInCurrentRole	0.029
DailyRate	0.029
DistanceFromHome	0.027
BusinessTravel	0.027
MonthlyRate	0.027
YearsAtCompany	0.026
NumCompaniesWorked	0.026
Gender	0.026
EnvironmentSatisfaction	0.025
TrainingTimesLastYear	0.025
HourlyRate	0.024
JobRole	0.023
PercentSalaryHike	0.020
EducationField	0.019
RelationshipSatisfaction	0.017
YearsSinceLastPromotion	0.017
WorkLifeBalance	0.016
Education	0.013
PerformanceRating	0.005

Once we have invested enough time in doing EDA and Pre-processing our data comes the step for which all the previous hard work was performed. That is to finally start building our Machine Learning model for classification purpose.

5. Building Machine Learning Models

In order to build a classification method I have imported the necessary libraries and created a function that contains all our machine learning model creation and its evaluation metrics steps.

This makes our job easier since later on we just need to feed the model's name and get the result without repeating/rewriting the same code again and again.

Code:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
import lightgbm as lgb

from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

```
# Classification Model Function

def classify(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=75)

    # Training the model
    model.fit(X_train, Y_train)

    # Predicting Y_test
    pred = model.predict(X_test)

    # Accuracy Score
    acc_score = (accuracy_score(Y_test, pred))*100
    print("Accuracy Score:", acc_score)

    # Classification Report
    class_report = classification_report(Y_test, pred)
    print("\nClassification Report:\n", class_report)

    # Cross Validation Score
    cv_score = (cross_val_score(model, X, Y, cv=5).mean())*100
    print("Cross Validation Score:", cv_score)

    # Result of accuracy minus cv scores
    result = acc_score - cv_score
    print("\nAccuracy Score - Cross Validation Score is", result)
```

Output:

```
# Logistic Regression

model=LogisticRegression()
classify(model, X, Y)
```

```
Accuracy Score: 88.77374784110536

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.90      0.89       297
           1       0.89      0.88      0.88       282

    accuracy                           0.89       579
   macro avg       0.89      0.89      0.89       579
weighted avg       0.89      0.89      0.89       579

Cross Validation Score: 84.63264318164893

Accuracy Score - Cross Validation Score is 4.141104659456431
```

Created the Logistic Regression Model and checked for it's evaluation metrics.

It is always advisable to build more than 5 machine learning models so that you can choose from the best performing model and then apply hyper parameter tuning to make it perform even better. I am going to use the Extra Trees Classifier as my choice of classification model as I see it is doing better than the other models I used.

Code:

```
# Choosing Extra Trees Classifier

fmod_param = {'criterion' : ["gini", "entropy"],
              'max_depth' : [30, 40, 50],
              'n_estimators' : [300, 350, 400],
              'min_samples_split' : [3, 4, 5],
              'random_state' : [42, 75, 111, 680, 895]
             }
```

```
GSCV = GridSearchCV(ExtraTreesClassifier(), fmod_param, cv=5)
```

```
GSCV.fit(X_train,Y_train)
```

```
GSCV.best_params_
```

Output:

```
{'criterion': 'entropy',
 'max_depth': 30,
 'min_samples_split': 3,
 'n_estimators': 400,
 'random_state': 75}
```

After applying the above steps to get the best parameters list, I simply have to plug it into my final model and receive the output of it. I have created an ROC curve plot and Confusion matrix for the final model.

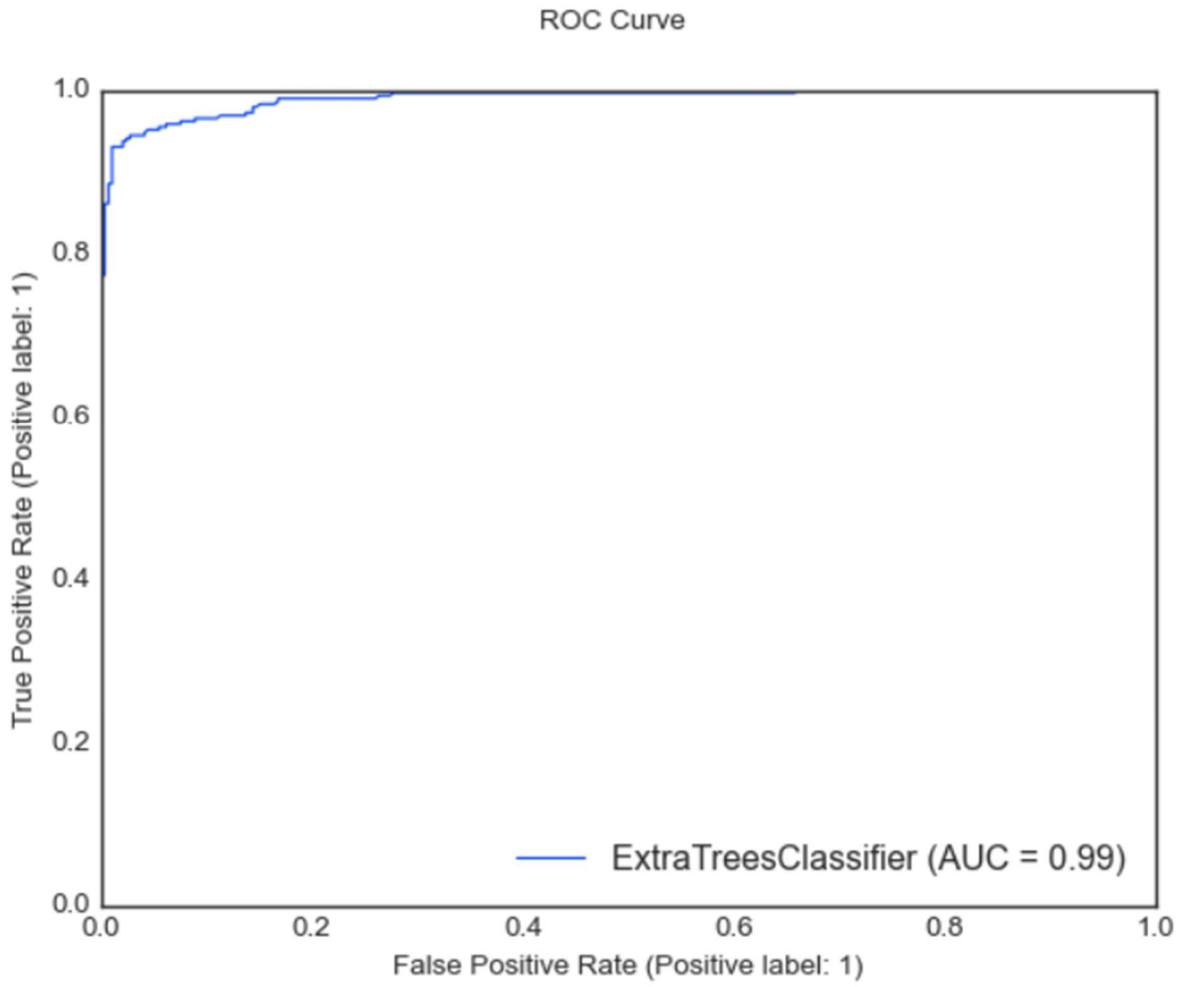
Code:

```
Final_Model = ExtraTreesClassifier(criterion="entropy", max_depth=30, min_samples_split=3,
                                   n_estimators=400, random_state=75)
Classifier = Final_Model.fit(X_train, Y_train)
fmod_pred = Final_Model.predict(X_test)
fmod_acc = (accuracy_score(Y_test, fmod_pred))*100
print("Accuracy score for the Best Model is:", fmod_acc)
```

```
Accuracy score for the Best Model is: 95.50949913644214
```

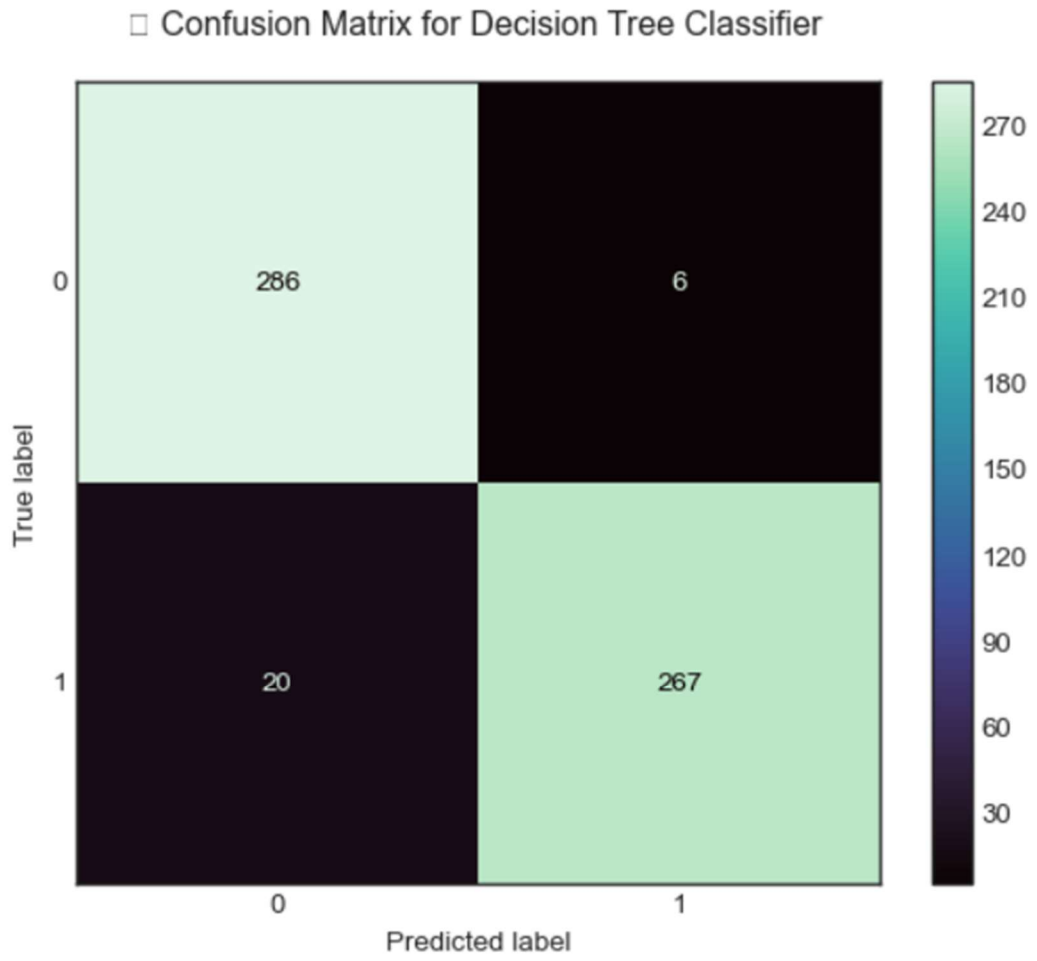
Code:

```
disp = metrics.plot_roc_curve(Final_Model, X_test, Y_test)
disp.figure_.suptitle("ROC Curve")
plt.show()
```



Code:

```
class_names = df.columns
metrics.plot_confusion_matrix(Classifier, X_test, Y_test, cmap='mako')
plt.title('\t Confusion Matrix for Decision Tree Classifier \n')
plt.show()
```



Accidentally I forgot to edit the print statement for my confusion matrix and it still shows the earlier Decision Tree model name instead of the Extra Trees model name and has the values stored for the latter. This also proves that your best performing model can keep changing at times even without changing your code and simply running it multiple times. But I am sure you can understand that the print statement can be changed as per your liking and the important stuff was on the utilization of the code showing the correct result.

Once you have gone through all the previous steps and you are satisfied with outcome you can then save the final model using either joblib or pickle. I have used the joblib method to save and then load my model from the same saved filename.

Code:

Saving the model

```
filename = "FinalModel_Blog.pkl"
joblib.dump(Final_Model, filename)
```

```
['FinalModel_Blog.pkl']
```

Code:

Loading the model

```
load_model = joblib.load(filename)
result = load_model.score(X_test, Y_test)*100
print(result)
```

```
95.50949913644214
```

6. Concluding Remarks

Kindly allow me to provide a quick recap on all the steps that we went through starting from understanding the Problem Definition then going through the Data Analysis and EDA processes. We went through the necessary Pre-processing Data steps before the final Building Machine Learning Models step came into picture.

What I do is code my entire project on my own and then take a peek at the internet to look through other's coding style for inspiration and understand if I can incorporate anything to improvise further on accuracy or beautify the visuals. However, I have seen many people doing the complete opposite whereupon they don't practise or create their own unique coding style first and rather copy paste lines from the web and perform some sort of messy patch work and when asked to explain might not be capable of conveying functioning or usage of those code blocks.

Before wrapping up my only advise to everyone is "No pain No gain" you will have to get your hands dirty with building your own code and trying out all the permutations and combinations. Create a self-made unique data story telling commandment list and follow it along with the standard project life cycle. Hope this at length article helps you in gaining the initial knowledge on building your first project from scratch.

Disclaimer: I am a newbie myself in this field with some accumulated knowledge over a year's time studying Data Science and felt like since sharing is caring someone who's stepping in now can benefit from my experience. I am also open to get some feedback from anyone that will help me in improving too! The content that I have written is solely my view of the project but it's definitely inspired by others over the internet who have worked on similar projects before me.