## Explain the problem you are trying to solve

The problem of predicting an Online Shopper's Intention involves with forecasting visitors and understanding with the target variable "revenue". An online shopping will generate revenue for businesses, it is an important job for retailers to create strategies, design, and communicate with the customers to maximize revenue. The goal of this assignment is to predict analytics to improve the effectiveness of online sales and marketing efforts, driving the growth of revenue.

## Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
```

*Link: [https://archive.ics.uci.edu/dataset/468/online+shoppers+purchasing+intention+dataset](https://archive.ics.uci.edu/dataset/468/online+shoppers+purchasing+intention+dataset)*

## Load dataset

```
LE = LabelEncoder()
OrigData = pd.read_csv('Data.csv')
Data = pd.read_csv('Data.csv')
Data
```

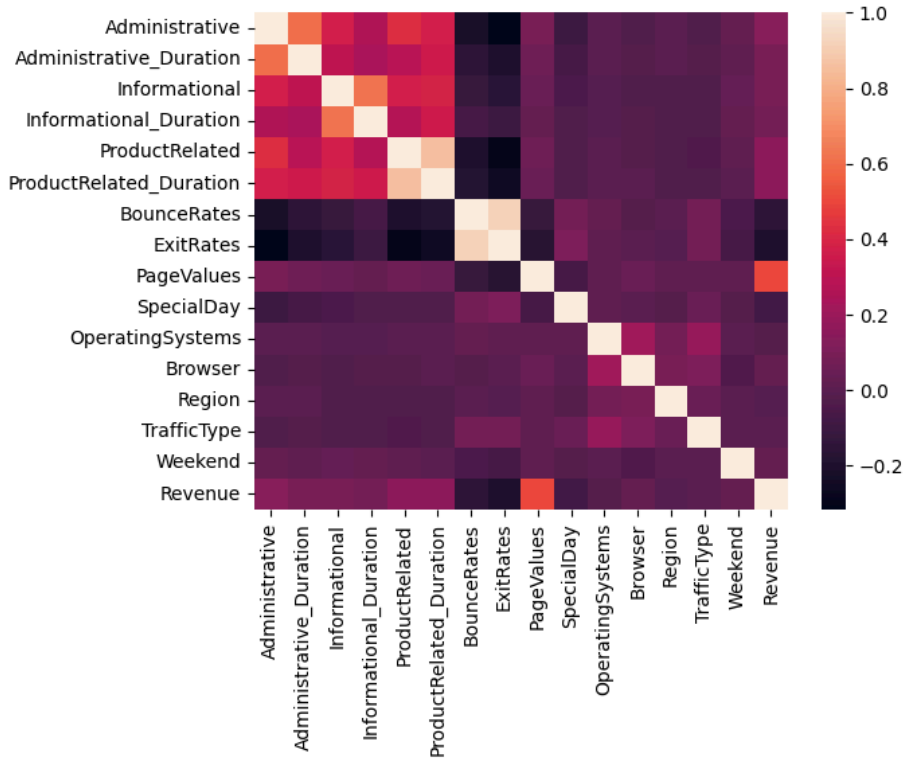
| | Administrative | Administrative_Duration | Informational | Informational_Duration | ProductRelated | ProductRelated_Duration | Bo |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 0.0 | 0 | 0.0 | 1 | 0.000000 | |
| **1** | 0 | 0.0 | 0 | 0.0 | 2 | 64.000000 | |
| **2** | 0 | 0.0 | 0 | 0.0 | 1 | 0.000000 | |
| **3** | 0 | 0.0 | 0 | 0.0 | 2 | 2.666667 | |
| **4** | 0 | 0.0 | 0 | 0.0 | 10 | 627.500000 | |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **12325** | 3 | 145.0 | 0 | 0.0 | 53 | 1783.791667 | |
| **12326** | 0 | 0.0 | 0 | 0.0 | 5 | 465.750000 | |
| **12327** | 0 | 0.0 | 0 | 0.0 | 6 | 184.250000 | |
| **12328** | 4 | 75.0 | 0 | 0.0 | 15 | 346.000000 | |
| **12329** | 0 | 0.0 | 0 | 0.0 | 3 | 21.250000 | |

12330 rows × 18 columns

Next steps: View recommended plots

```
sns.heatmap(Data.corr())
```

```
<ipython-input-64-942c4db03ddf>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future v
  sns.heatmap(Data.corr())
<Axes: >
```




**Remarks:** In this cell, as we can see in the heatmap, almost all of them are have a negative value, because the dataset has not yet preprocess, in this case, we can see that some of them are not correlated.

## Checking values and data types

```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

```
for i in Data:
  if Data[i].dtypes == 'object':
    Data[i] = LE.fit_transform(Data[i])
  elif Data[i].dtypes == 'bool':
    Data[i] = LE.fit_transform(Data[i])
  else:
    pass
Data
```

| | Administrative | Administrative_Duration | Informational | Informational_Duratio |
|---|---|---|---|---|
| **0** | 0 | 0.0 | 0 | 0. |
| **1** | 0 | 0.0 | 0 | 0. |
| **2** | 0 | 0.0 | 0 | 0. |
| **3** | 0 | 0.0 | 0 | 0. |
| **4** | 0 | 0.0 | 0 | 0. |
| **...** | ... | ... | ... | . |
| **12325** | 3 | 145.0 | 0 | 0. |
| **12326** | 0 | 0.0 | 0 | 0. |
| **12327** | 0 | 0.0 | 0 | 0. |
| **12328** | 4 | 75.0 | 0 | 0. |
| **12329** | 0 | 0.0 | 0 | 0. |

12330 rows × 18 columns

Next steps: View recommended plots

## Check if the target variable is balance

```
count = Data['Revenue'].value_counts()
count
```

```
0    10422
1     1908
Name: Revenue, dtype: int64
```

## Balance the values of class

```
TheZero = Data[(Data.Revenue == 0)]
TheOne = Data[(Data.Revenue == 1)]
```

```
TheZeroDownside = TheZero.sample(len(TheOne), random_state = 123)
DataBalanced = pd.concat([TheZeroDownside, TheOne])
```

```
count = DataBalanced['Revenue'].value_counts()
count
```

```
0    1908
1    1908
Name: Revenue, dtype: int64
```

```
db = DataBalanced
db
```

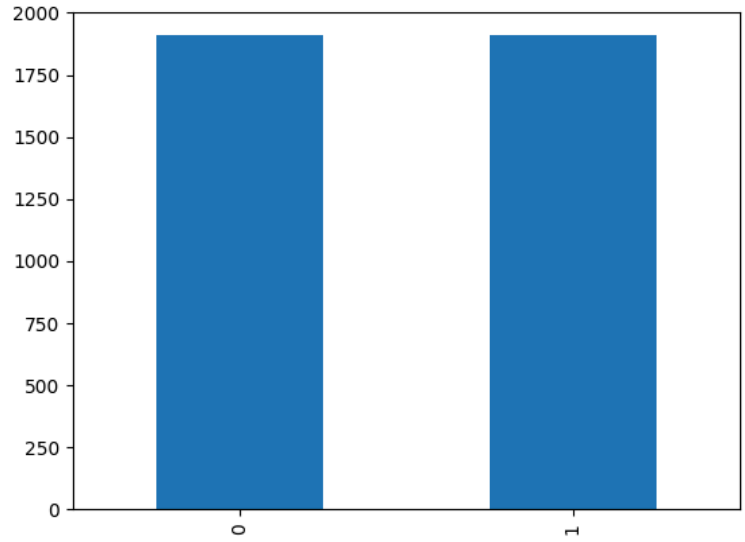
| | Administrative | Administrative_Duration | Informational | Informational_Duratio |
|---|---|---|---|---|
| **12097** | 1 | 0.000000 | 0 | 0.0 |
| **11805** | 2 | 45.500000 | 0 | 0.0 |
| **11969** | 0 | 0.000000 | 0 | 0.0 |
| **8879** | 6 | 143.000000 | 0 | 0.0 |
| **6428** | 1 | 54.400000 | 0 | 0.0 |
| **...** | ... | ... | ... | . |
| **12272** | 6 | 133.466667 | 0 | 0.0 |
| **12276** | 7 | 139.575000 | 0 | 0.0 |
| **12311** | 1 | 0.000000 | 2 | 211.2 |
| **12312** | 7 | 150.357143 | 1 | 9.0 |
| **12313** | 3 | 16.000000 | 3 | 86.0 |

3816 rows × 18 columns

Next steps: View recommended plots

```
db['Revenue'].value_counts(normalize = True)
db['Revenue'].value_counts().plot(kind = 'bar')
```

```
<Axes: >
```




## Checking the datas

```
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3816 entries, 12097 to 12313
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           3816 non-null   int64
 1   Administrative_Duration  3816 non-null   float64
 2   Informational            3816 non-null   int64
 3   Informational_Duration   3816 non-null   float64
 4   ProductRelated           3816 non-null   int64
 5   ProductRelated_Duration  3816 non-null   float64
 6   BounceRates              3816 non-null   float64
 7   ExitRates                3816 non-null   float64
 8   PageValues               3816 non-null   float64
 9   SpecialDay               3816 non-null   float64
 10  Month                    3816 non-null   int64
 11  OperatingSystems         3816 non-null   int64
 12  Browser                  3816 non-null   int64
 13  Region                   3816 non-null   int64
 14  TrafficType              3816 non-null   int64
 15  VisitorType              3816 non-null   int64
 16  Weekend                  3816 non-null   int64
 17  Revenue                  3816 non-null   int64
```

```
dtypes: float64(7), int64(11)
memory usage: 566.4 KB
```

```
db.describe()
```

| | Administrative | Administrative_Duration | Informational | Informational_Duratio |
|---|---|---|---|---|
| **count** | 3816.00000 | 3816.000000 | 3816.000000 | 3816.00000( |
| **mean** | 2.77044 | 98.766276 | 0.615042 | 42.38891 |
| **std** | 3.51096 | 190.651493 | 1.374119 | 148.06081( |
| **min** | 0.00000 | 0.000000 | 0.000000 | 0.00000( |
| **25%** | 0.00000 | 0.000000 | 0.000000 | 0.00000( |
| **50%** | 1.00000 | 29.782576 | 0.000000 | 0.00000( |
| **75%** | 4.00000 | 116.210417 | 1.000000 | 0.00000( |
| **max** | 26.00000 | 2720.500000 | 12.000000 | 2256.91666 |

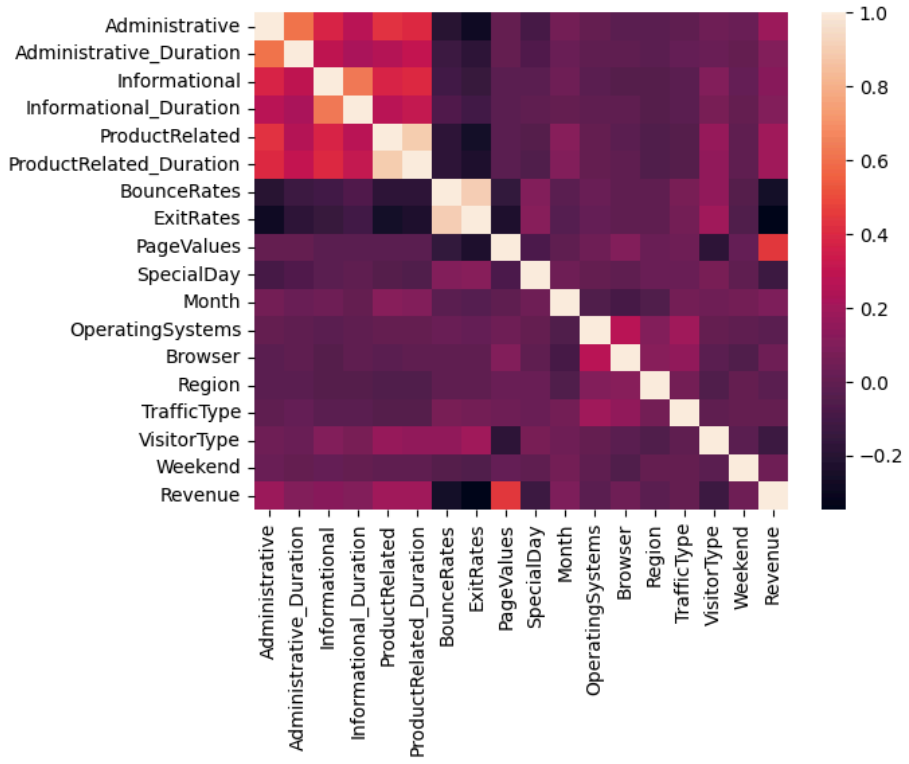
```
db.corr()
```

| Rates | ExitRates | PageValues | SpecialDay | Month | OperatingSystems | Browser | Regi |
|---|---|---|---|---|---|---|---|
| 09274 | -0.280111 | 0.001300 | -0.083790 | 0.058932 | 0.006624 | -0.015291 | -0.0220 |
| 31245 | -0.183475 | 0.009511 | -0.065767 | 0.022707 | -0.001231 | -0.006903 | -0.0195 |
| 01005 | -0.145086 | -0.027665 | -0.030679 | 0.042796 | -0.016384 | -0.046182 | -0.0408 |
| 57219 | -0.101017 | -0.018191 | -0.011194 | 0.009478 | -0.003497 | -0.011163 | -0.0361 |
| 74129 | -0.257073 | -0.022429 | -0.045378 | 0.113630 | 0.003659 | -0.015527 | -0.0587 |
| 58176 | -0.237680 | -0.016866 | -0.049207 | 0.104885 | 0.006215 | -0.014768 | -0.0501 |
| 00000 | 0.909399 | -0.162134 | 0.098529 | -0.023849 | 0.023777 | -0.009643 | -0.0003 |
| 09399 | 1.000000 | -0.232189 | 0.119542 | -0.035723 | 0.011362 | -0.005215 | -0.0045 |
| 52134 | -0.232189 | 1.000000 | -0.073258 | -0.007229 | 0.036132 | 0.096966 | 0.0284 |
| 98529 | 0.119542 | -0.073258 | 1.000000 | 0.043811 | 0.001707 | -0.006185 | 0.0219 |
| 23849 | -0.035723 | -0.007229 | 0.043811 | 1.000000 | -0.055241 | -0.089773 | -0.0527 |
| 23777 | 0.011362 | 0.036132 | 0.001707 | -0.055241 | 1.000000 | 0.264856 | 0.1112 |
| 09643 | -0.005215 | 0.096966 | -0.006185 | -0.089773 | 0.264856 | 1.000000 | 0.1148 |
| 00374 | -0.004570 | 0.028435 | 0.021996 | -0.052785 | 0.111239 | 0.114811 | 1.0000 |
| 58531 | 0.054498 | 0.044787 | 0.022986 | 0.052638 | 0.190231 | 0.155408 | 0.0617 |
| 47699 | 0.192309 | -0.180194 | 0.074512 | 0.045126 | 0.009319 | -0.030540 | -0.0481 |
| 42920 | -0.058121 | 0.016516 | -0.003003 | 0.049161 | -0.002235 | -0.049256 | 0.0083 |
| 54269 | -0.346233 | 0.447635 | -0.120711 | 0.089455 | -0.028837 | 0.042455 | -0.0249 |

**Remarks:** In this cell, most of them are still negative value, some of the value has improve but some are not, I notice that the lowest value that I see is 0.003091 and it is TrafficTypes, means that TrafficType does not correlated to Revenue which is our Target Variable.

```
sns.heatmap(db.corr())
```

```
<Axes: >
```




## Removing the columns that is not correlated to the target variables

```
#Column Region does not correlate to revenue
#Drop Region
db = db.drop(columns = 'TrafficType')
#db
```

## Split data

```
X = db.drop(columns = 'Revenue')
y = db['Revenue']
```

## Normalizing data

```
Sc = StandardScaler()
Xnorm = Sc.fit_transform(X)
```

```
scale = MinMaxScaler(feature_range=(0, 1))
Xnormed = scale.fit_transform(X)
```

## Spliting the training and test

```
X_train, X_test, y_train, y_test = train_test_split(Xnormed, y, test_size = 0.3, random_state = 123)
```

## Train Multilayer Perceptron

```
mlp = MLPClassifier(hidden_layer_sizes=(100,50),
                    activation='tanh',
                    solver='adam',
                    max_iter=1000,
                    verbose = True,
                    random_state=123)
mlp.fit(X_train, y_train)
```