mrc03 / IBM-HR-Analytics-Employee-Attrition-Performance Public

Code Issues Pull requests 1 Actions Projects Wiki Security Insights

master

IBM-HR-Analytics-Employee-Attrition-Performance / ibm-hr-attrition -ann.ipynb

mrc03 Update ibm-hr-attrition -ann.ipynb

1 contributor

4185 lines (4185 sloc) | 763 KB

IBM HR Analytics Employee Attrition & Performance.

[Please star/upvote it if you find it helpful.]

In [1]:

```
from IPython.display import Image
Image("image-logo.png")
```

Out[1]:

In [2]:

```
Image("image-hr.jpg")
```

Out[2]:

CONTENTS :

1) Exploratory Data Analysis

2) Corelation b/w Features

3) Feature Selection

4) Preparing Dataset

5) Making Predictions Using an Artificial Neural Network (ANN)

6) Hyperparameter Tuning

7) Conclusions

1) Exploratory Data Analysis

1.1) Importing Various Modules

In [3]:

```
# Ignore  the warnings
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
```

```
# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
import missingno as msno

#configure
# sets matplotlib to inline and displays graphs below the corressponding ce
% matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

#import the necessary modelling algos.
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB

#model selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score,precision_score,recall_score,con
from sklearn.model_selection import GridSearchCV

from imblearn.over_sampling import SMOTE

#preprocess.
from sklearn.preprocessing import MinMaxScaler,StandardScaler,Imputer,Label

# ann and dl libraraies
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from keras.utils import to_categorical

import tensorflow as tf
import random as rn
```

```
Using TensorFlow backend.
```

1.2) Reading the data from a CSV file

In [4]:

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

In [5]:

```
df.head()
```

Out[5]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educatic
0	41	Yes	Travel_Rarely	1102	Sales	1	
1	49	No	Travel_Frequently	279	Research & Development	8	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	

3	33	No	Travel_Frequently	1392	Research & Development	3
4	27	No	Travel_Rarely	591	Research & Development	2

5 rows × 35 columns

In [6]:
```
df.shape
```
Out[6]:
```
(1470, 35)
```

In [7]:
```
df.columns
```
Out[7]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorke
d',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

1.3) Missing Values Treatment

In [8]:
```
df.info()  # no null or Nan values.
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
Age                          1470 non-null int64
Attrition                    1470 non-null object
BusinessTravel               1470 non-null object
DailyRate                    1470 non-null int64
Department                   1470 non-null object
DistanceFromHome             1470 non-null int64
Education                    1470 non-null int64
EducationField               1470 non-null object
EmployeeCount                1470 non-null int64
EmployeeNumber               1470 non-null int64
EnvironmentSatisfaction      1470 non-null int64
Gender                       1470 non-null object
HourlyRate                   1470 non-null int64
JobInvolvement               1470 non-null int64
JobLevel                     1470 non-null int64
JobRole                      1470 non-null object
JobSatisfaction              1470 non-null int64
MaritalStatus                1470 non-null object
MonthlyIncome                1470 non-null int64
MonthlyRate                  1470 non-null int64
NumCompaniesWorked           1470 non-null int64
Over18                       1470 non-null object
OverTime                     1470 non-null object
PercentSalaryHike            1470 non-null int64
```

```
PerformanceRating           1470 non-null int64
RelationshipSatisfaction    1470 non-null int64
StandardHours               1470 non-null int64
StockOptionLevel            1470 non-null int64
TotalWorkingYears           1470 non-null int64
TrainingTimesLastYear       1470 non-null int64
WorkLifeBalance             1470 non-null int64
YearsAtCompany              1470 non-null int64
YearsInCurrentRole          1470 non-null int64
YearsSinceLastPromotion     1470 non-null int64
YearsWithCurrManager        1470 non-null int64
dtypes: int64(26), object(9)
memory usage: 402.0+ KB
```

In [9]:

```
df.isnull().sum()
```

Out[9]:

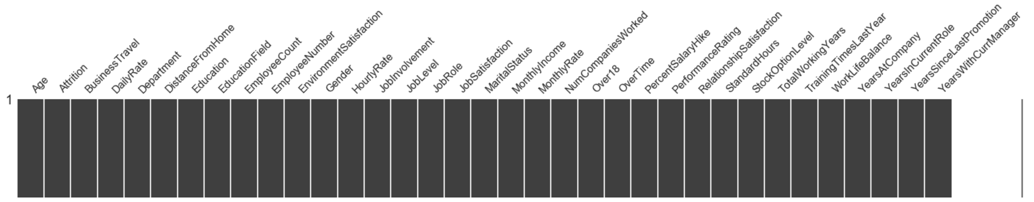
```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              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
```

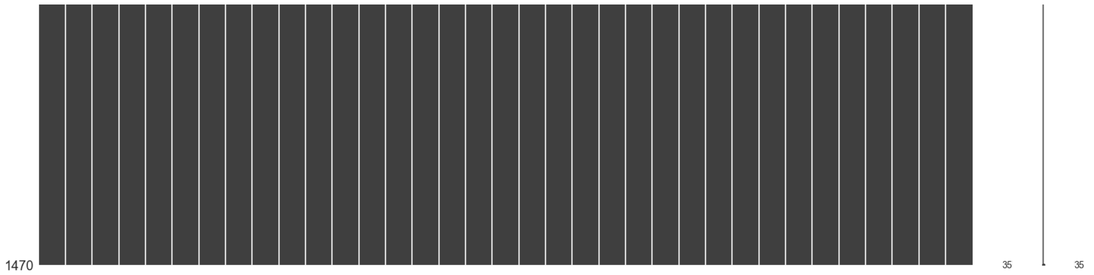
In [10]:

```
msno.matrix(df) # just to visualize.
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x249a858e978>
```





1.4) The Features and the 'Target'

In [11]:

```
df.columns
```

Out[11]:

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

In [12]:

```
df.head()
```

Out[12]:

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

5 rows × 35 columns

In all we have 34 features consisting of both the categorical as well as the numerical features. The target variable is the 'Attrition' of the employee which can be either a Yes or a No.

Hence this is a Binary Classification problem.

1.5) Univariate Analysis

In this section I have done the univariate analysis i.e. I have analysed the range or distribution of the values that various features take. To better analyze the results I have plotted various graphs and visualizations wherever necessary.

In [13]:

In [13]:
```
df.describe()
```

Out[13]:

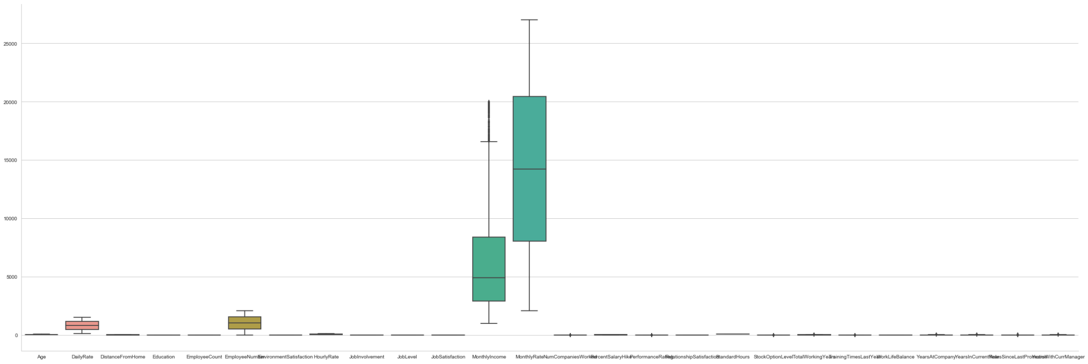
	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	Empl
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	
mean	36.923810	802.485714	9.192517	2.912925	1.0	
std	9.135373	403.509100	8.106864	1.024165	0.0	
min	18.000000	102.000000	1.000000	1.000000	1.0	
25%	30.000000	465.000000	2.000000	2.000000	1.0	
50%	36.000000	802.000000	7.000000	3.000000	1.0	
75%	43.000000	1157.000000	14.000000	4.000000	1.0	
max	60.000000	1499.000000	29.000000	5.000000	1.0	

8 rows × 26 columns

Let us first analyze the various numeric features. To do this we can actually plot a boxplot showing all the numeric features.

In [14]:
```
sns.factorplot(data=df,kind='box',size=10,aspect=3)
```

Out[14]: <seaborn.axisgrid.FacetGrid at 0x249a8401160>

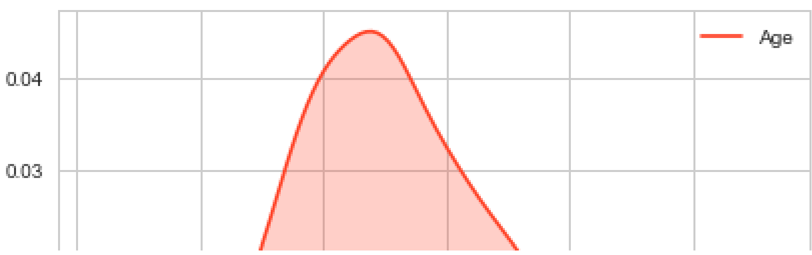


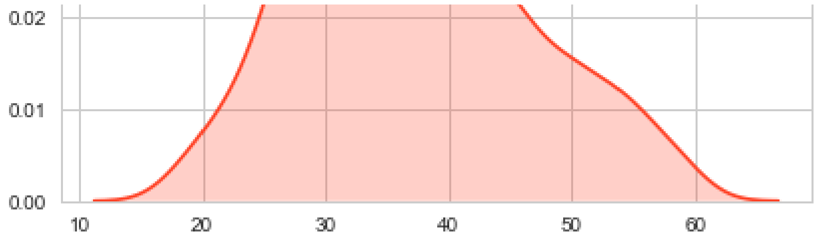
Note that all the features have pretty different scales and so plotting a boxplot is not a good idea. Instead what we can do is plot histograms of various continuously distributed features.

We can also plot a kdeplot showing the distribution of the feature. Below I have plotted a kdeplot for the 'Age' feature. Similarly we plot for other numeric features also. We can also use a distplot from seaborn library.

In [15]:
```
sns.kdeplot(df['Age'],shade=True,color='#ff4125')
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x249a8b549e8>





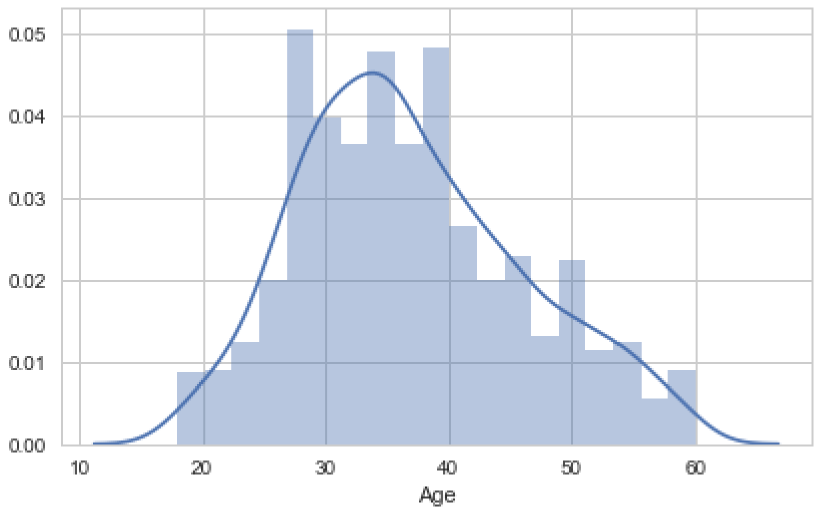
In [16]:

```
sns.distplot(df['Age'])
```

```
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: User
Warning: The 'normed' kwarg is deprecated, and has been replaced by the 'de
nsity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[16]:

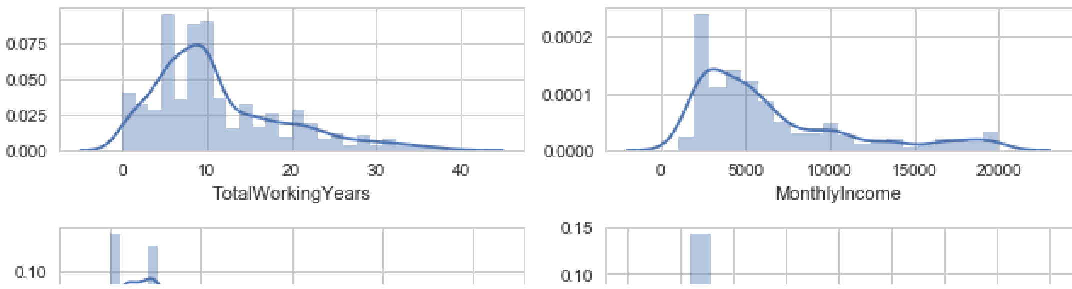
```
<matplotlib.axes._subplots.AxesSubplot at 0x249a8bb5da0>
```

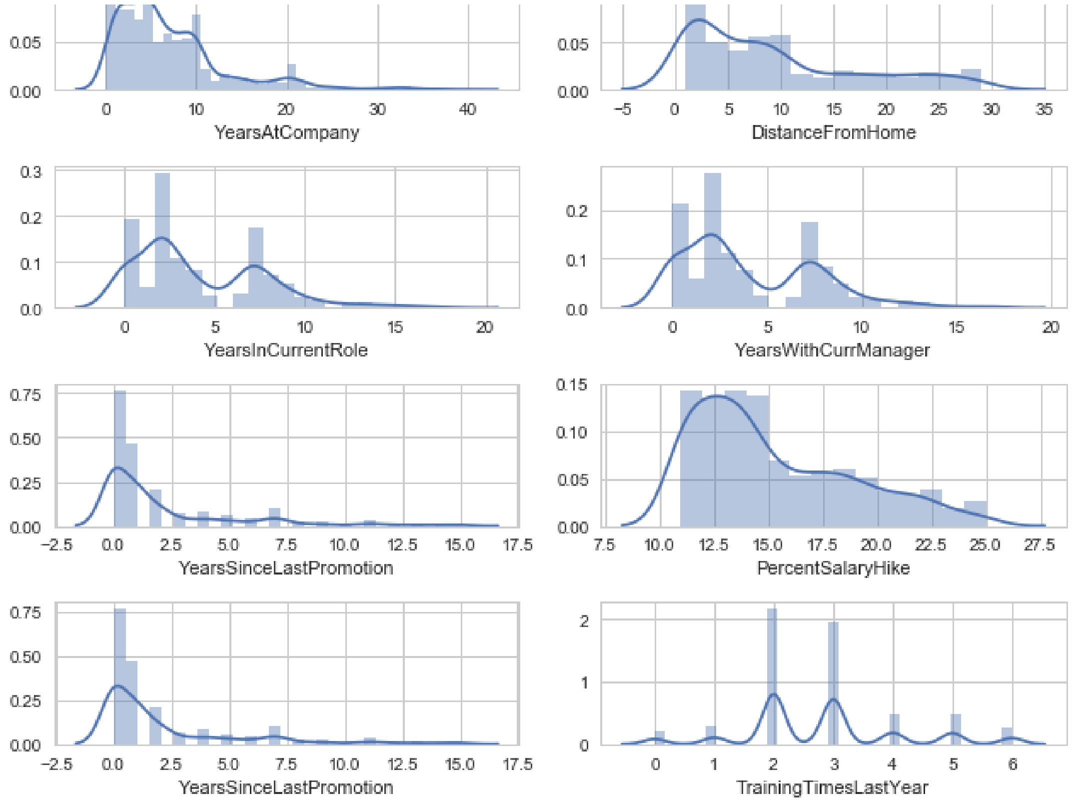


Similarly we can do this for all the numerical features. Below I have plotted the subplots for the other features.

In [17]:

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

fig,ax = plt.subplots(5,2, figsize=(9,9))
sns.distplot(df['TotalWorkingYears'], ax = ax[0,0])
sns.distplot(df['MonthlyIncome'], ax = ax[0,1])
sns.distplot(df['YearsAtCompany'], ax = ax[1,0])
sns.distplot(df['DistanceFromHome'], ax = ax[1,1])
sns.distplot(df['YearsInCurrentRole'], ax = ax[2,0])
sns.distplot(df['YearsWithCurrManager'], ax = ax[2,1])
sns.distplot(df['YearsSinceLastPromotion'], ax = ax[3,0])
sns.distplot(df['PercentSalaryHike'], ax = ax[3,1])
sns.distplot(df['YearsSinceLastPromotion'], ax = ax[4,0])
sns.distplot(df['TrainingTimesLastYear'], ax = ax[4,1])
plt.tight_layout()
plt.show()
```





Let us now analyze the various categorical features. Note that in these cases the best way is to use a count plot to show the relative count of observations of different categories.

In [18]:

```
cat_df=df.select_dtypes(include='object')
```

In [19]:

```
cat_df.columns
```

Out[19]:

```
Index(['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gend
er',
       'JobRole', 'MaritalStatus', 'Over18', 'OverTime'],
      dtype='object')
```

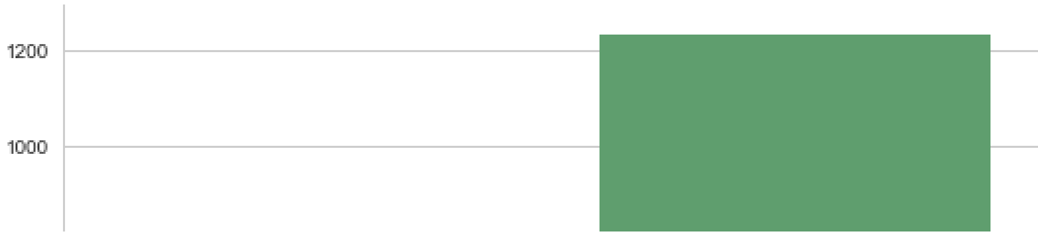
In [20]:

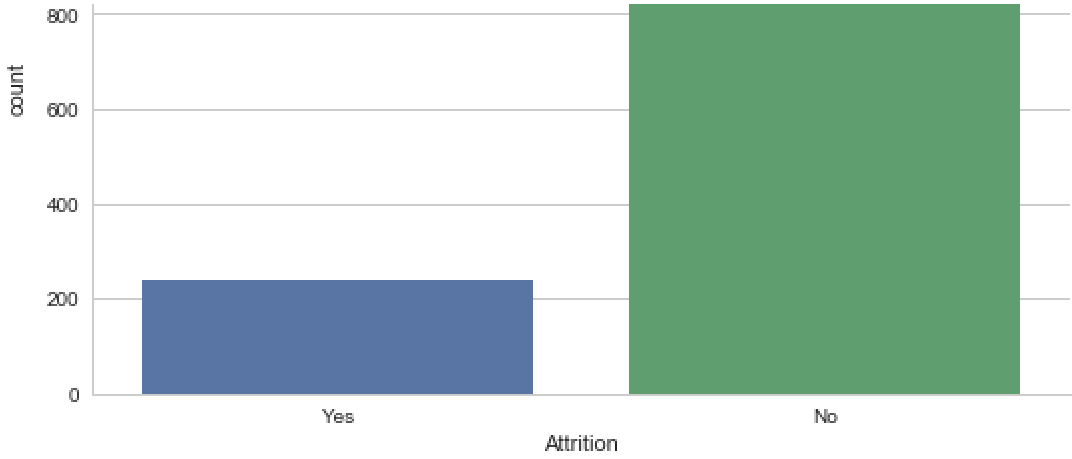
```
def plot_cat(attr,labels=None):
    if(attr=='JobRole'):
        sns.factorplot(data=df,kind='count',size=5,aspect=3,x=attr)
        return

    sns.factorplot(data=df,kind='count',size=5,aspect=1.5,x=attr)
```

I have made a function that accepts the name of a string. In our case this string will be the name of the column or attribute which we want to analyze. The function then plots the countplot for that feature which makes it easier to visualize.

In [21]:

```
plot_cat('Attrition')
```

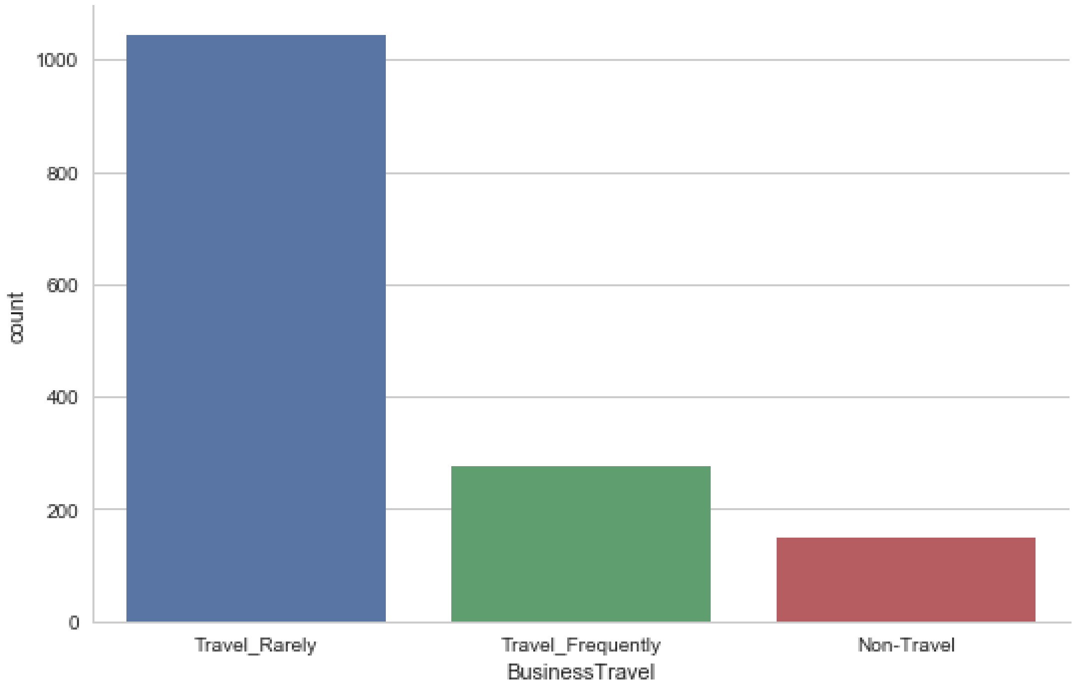




Note that the number of observations belonging to the 'No' category is way greater than that belonging to 'Yes' category. Hence we have skewed classes and this is a typical example of the 'Imbalanced Classification Problem'. To handle such types of problems we need to use the over-sampling or under-sampling techniques. I shall come back to this point later.

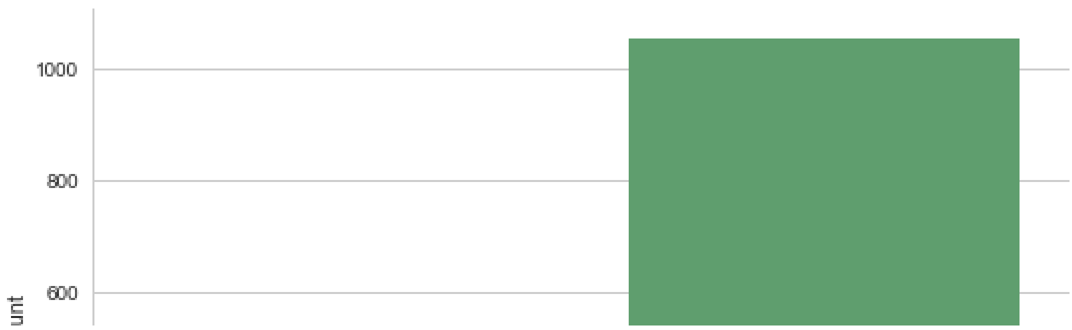
Let us now similalry analyze other categorical features.

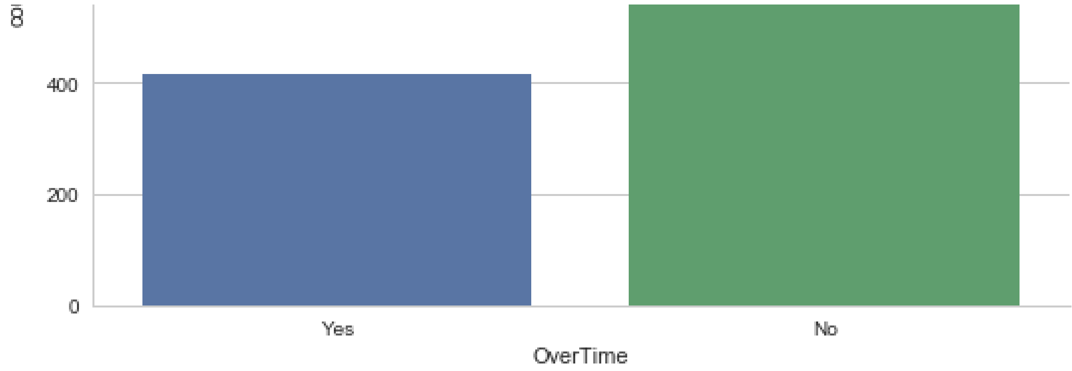
In [22]:
```
plot_cat('BusinessTravel')
```



The above plot clearly shows that most of the people belong to the 'Travel_Rarely' class. This indicates that most of the people did not have a job which asked them for frequent travelling.

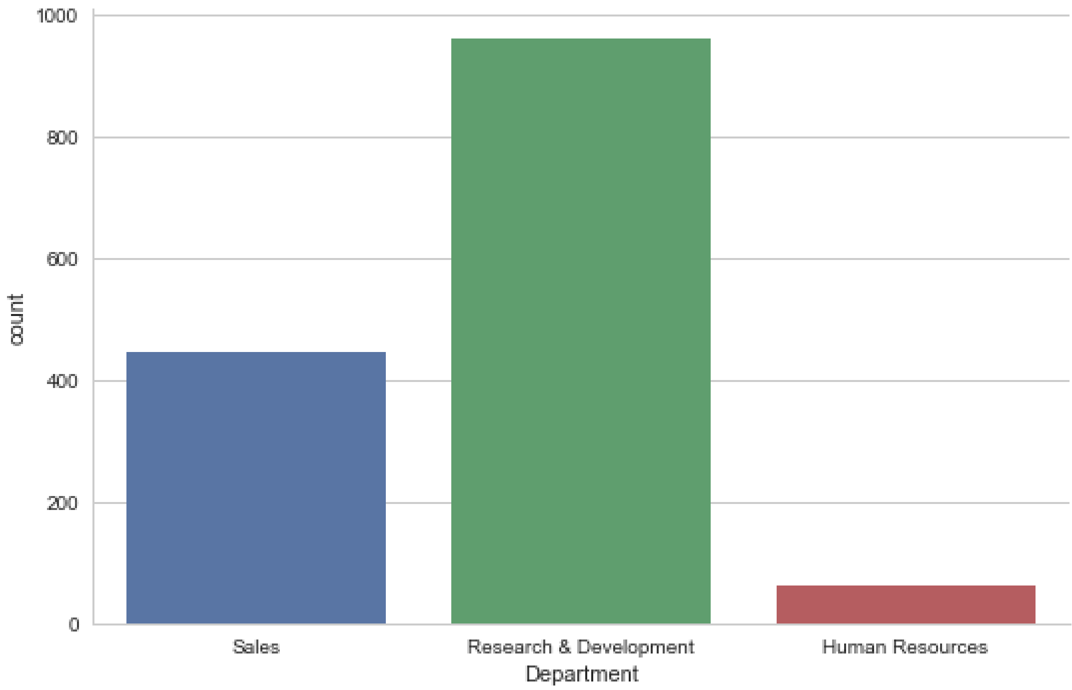
In [23]:
```
plot_cat('OverTime')
```





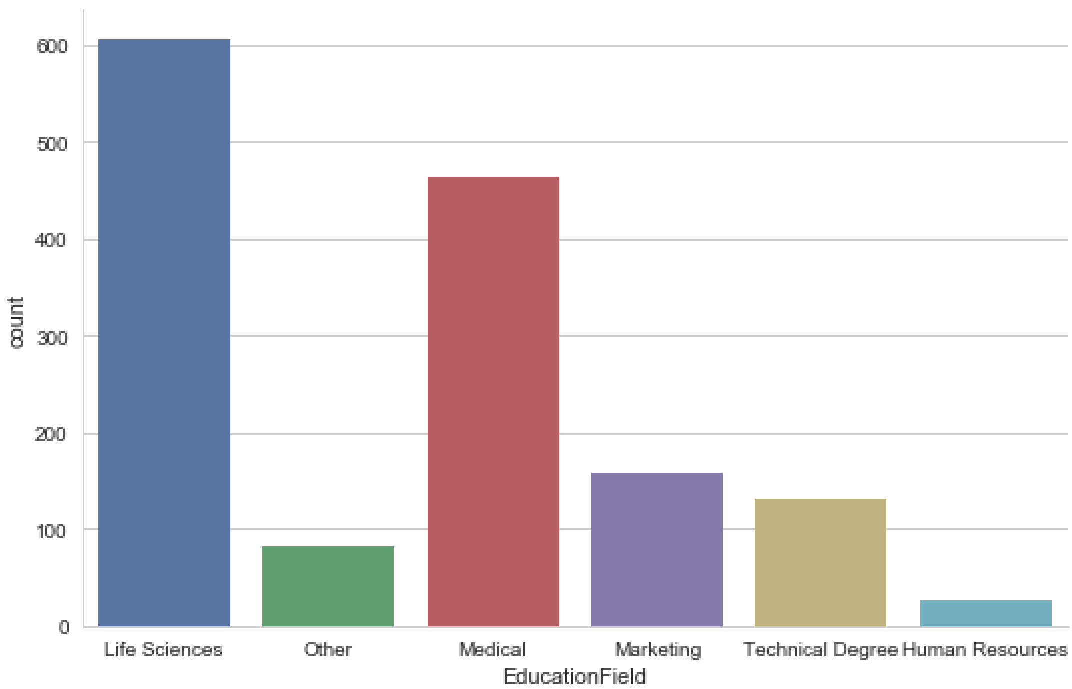
In [24]:

```
plot_cat('Department')
```



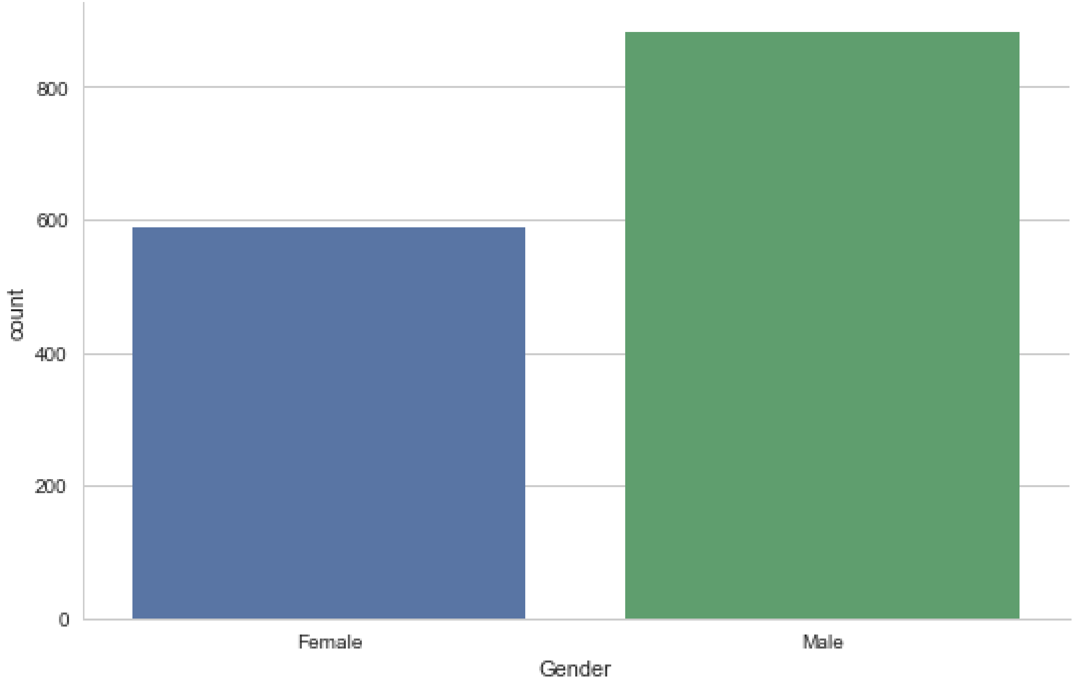
In [25]:

```
plot_cat('EducationField')
```



In [26]:

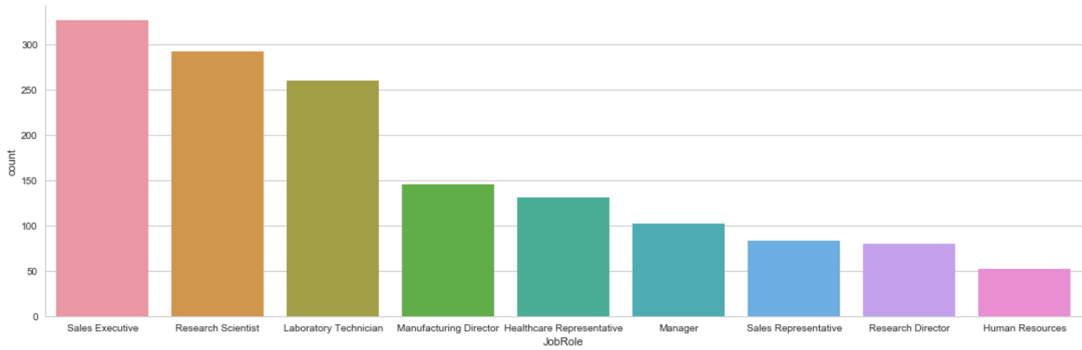
```
plot_cat('Gender')
```



Note that males are presnt in higher number.

In [27]:

```
plot_cat('JobRole')
```



Similarly we can continue for other categorical features.

Note that the same function can also be used to better analyze the numeric discrete features like 'Education' ,'JobSatisfaction' etc...

In [28]:

```
# just uncomment the following cell.
```

In [29]:

```
# num_disc=['Education','EnvironmentSatisfaction','JobInvolvement','JobSati
# for i in num_disc:
#     plot_cat(i)

# similarly we can intrepret these graphs.
```

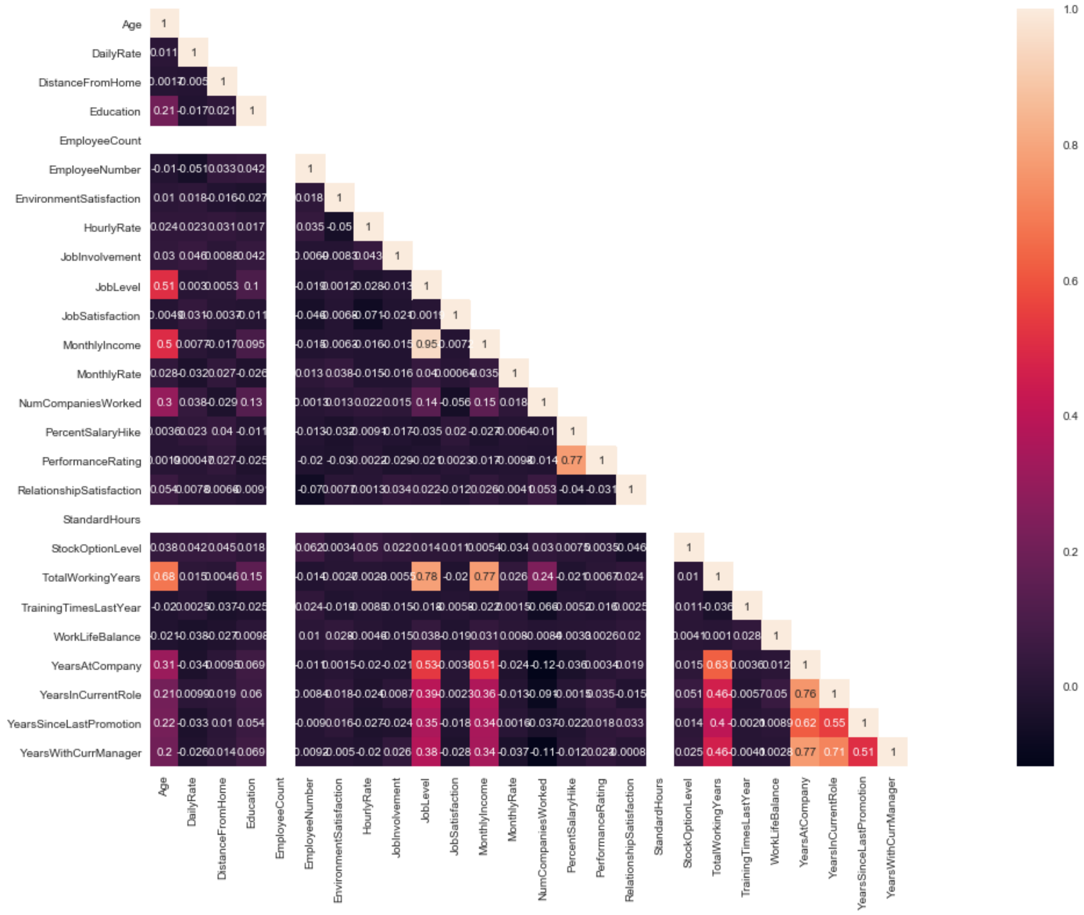
2) Corelation b/w Features

In [30]:

```
#corelation matrix.
cor_mat= df.corr()
mask = np.array(cor_mat)
mask[np.tril_indices_from(mask)] = False
fig=plt.gcf()
```

```
fig.set_size_inches(30,12)
sns.heatmap(data=cor_mat,mask=mask,square=True,annot=True,cbar=True)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x249ab7db908>



BREAKING IT DOWN

Firstly calling .corr() method on a pandas data frame returns a corelation data frame containing the corelation values b/w the various attributes. now we obtain a numpy array from the corelation data frame using the np.array method. nextly using the np.tril_indices.from() method we set the values of the lower half of the mask numpy array to False. this is bcoz on passing the mask to heatmap function of the seaborn it plots only those squares whose mask is False. therefore if we don't do this then as the mask is by default True then no square will appear. Hence in a nutshell we obtain a numpy array from the corelation data frame and set the lower values to False so that we can visualise the corelation. In order for a full square just use the [:] operator in mask in place of tril_ind... function. in next step we get the refernce to the current figure using the gcf() function of the matplotlib library and set the figure size. in last step we finally pass the necessary parameters to the heatmap function.

DATA=the corelation data frame containing the 'CORELATION' values.

MASK= explained earlier.

vmin,vmax= range of values on side bar

SQUARE= to show each individual unit as a square.

ANNOT- whether to dispaly values on top of square or not. In order to dispaly pass it either True or the cor_mat.

CBAR= whether to view the side bar or not.

SOME INFERENCES FROM THE ABOVE HEATMAP

1. Self relation ie of a feature to itself is equal to 1 as expected.
2. JobLevel is highly related to Age as expected as aged employees will generally tend to occupy higher positions in the company.
3. MonthlyIncome is very strongly related to joblevel as expected as senior employees will definately earn more.
4. PerformanceRating is highly related to PercentSalaryHike which is quite obvious.
5. Also note that TotalWorkingYears is highly related to JobLevel which is expected as senior employees must have worked for a larger span of time.
6. YearsWithCurrManager is highly related to YearsAtCompany.
7. YearsAtCompany is related to YearsInCurrentRole.

Note that we can drop some highly corelated features as they add redundancy to the model but since the corelation is very less in genral let us keep all the features for now. In case of highly corelated features we can use something like Principal Component Analysis(PCA) to reduce our feature space.

In [31]:

```
df.columns
```

Out[31]:

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

3) Feature Selection

3.1) Plotting the Features against the 'Target' variable.

3.1.1) Age

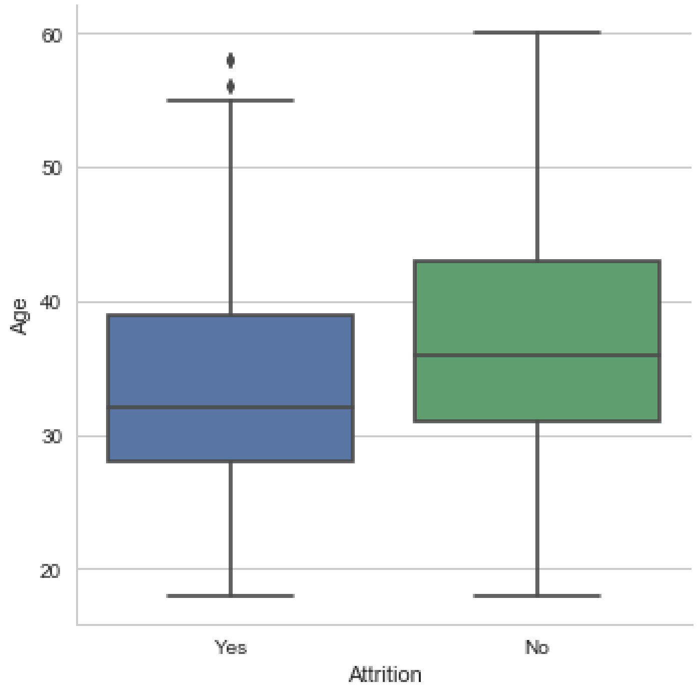
Note that Age is a continuous quantity and therefore we can plot it against the Attrition using a boxplot.

In [32]:

```
sns.factorplot(data=df,y='Age',x='Attrition',size=5,aspect=1,kind='box')
```

Out[32]:

```
<seaborn.axisgrid.FacetGrid at 0x249ab81e940>
```



Note that the median as well the maximum age of the peole with 'No' attrition is higher than that of the 'Yes' category. This shows that peole with higher age have lesser tendency to leave the organisation which makes sense as they may have settled in the organisation.

3.1.2) Department

Note that both Attrition(Target) as well as the Deaprtment are categorical. In such cases a cross-tabulation is the most reasonable way to analyze the trends; which shows clearly the number of observaftions for each class which makes it easier to analyze the results.

In [33]:

```
df.Department.value_counts()
```

Out[33]:

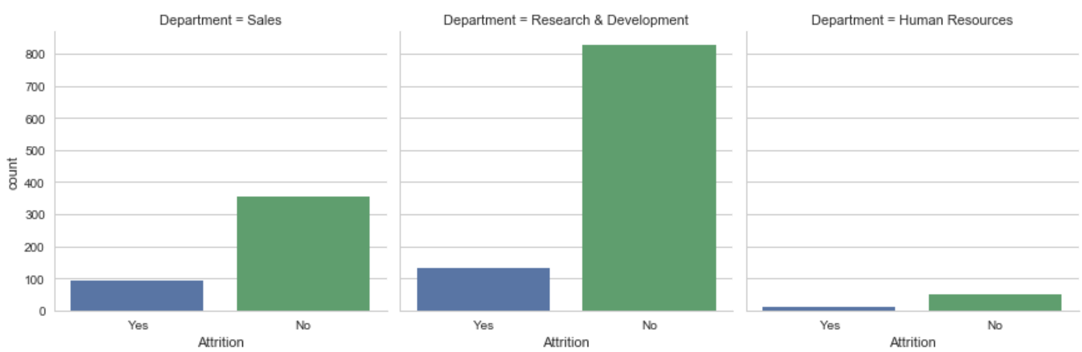
```
Research & Development    961
Sales                     446
Human Resources            63
Name: Department, dtype: int64
```

In [34]:

```
sns.factorplot(data=df,kind='count',x='Attrition',col='Department')
```

Out[34]:

```
<seaborn.axisgrid.FacetGrid at 0x249ac28d400>
```



In [35]:

```
pd.crosstab(columns=[df.Attrition],index=[df.Department],margins=True,norma
```

Out[35]:

Attrition	No	Yes
Department		
Human Resources	0.809524	0.190476
Research & Development	0.861602	0.138398
Sales	0.793722	0.206278
All	0.838776	0.161224

Note that most of the observations corresspond to 'No' as we saw previously also. About 81 % of the people in HR dont want to leave the organisation and only 19 % want to leave. Similar conclusions can be drawn for other departments too from the above cross-tabulation.

3.1.3) Gender

In [36]:

```
pd.crosstab(columns=[df.Attrition],index=[df.Gender],margins=True,normalize
```

Out[36]:

Attrition	No	Yes
Gender		
Female	0.852041	0.147959
Male	0.829932	0.170068
All	0.838776	0.161224

About 85 % of females want to stay in the organisation while only 15 % want to leave the organisation. All in all 83 % of employees want to be in the organisation with only being 16% wanting to leave the organisation or the company.

3.1.4) Job Level

In [37]:

```
pd.crosstab(columns=[df.Attrition],index=[df.JobLevel],margins=True,normali
```

Out[37]:

Attrition	No	Yes
JobLevel		
1	0.736648	0.263352
2	0.902622	0.097378
3	0.853211	0.146789
4	0.952830	0.047170
5	0.927536	0.072464
All	0.838776	0.161224

People in Joblevel 4 have a very high percent for a 'No' and a low percent for a 'Yes'. Similar inferences can be made for other job levels.

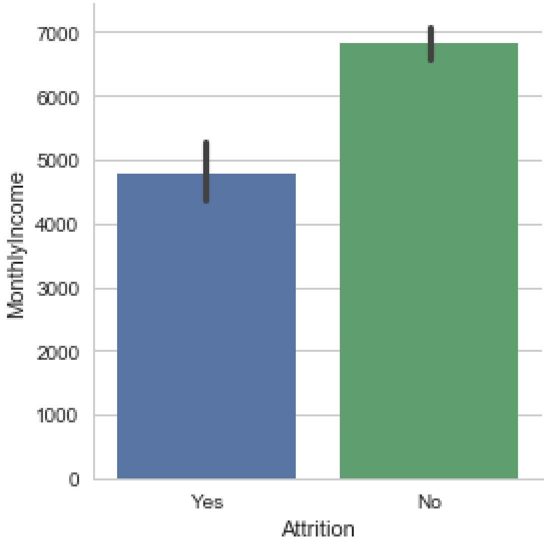
3.1.5) Monthly Income

In [38]:

```
sns.factorplot(data=df,kind='bar',x='Attrition',y='MonthlyIncome')
```

Out[38]:

```
<seaborn.axisgrid.FacetGrid at 0x249ac2ac198>
```



Note that the average income for 'No' class is quite higher and it is obvious as those earning well will certainly not be willing to exit the organisation. Similarly those employees who are probably not earning well will certainly want to change the company.

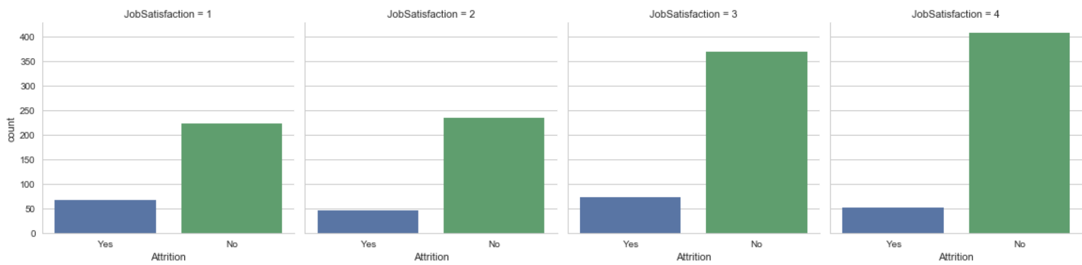
3.1.6) Job Satisfaction

In [39]:

```
sns.factorplot(data=df,kind='count',x='Attrition',col='JobSatisfaction')
```

Out[39]:

```
<seaborn.axisgrid.FacetGrid at 0x249abab04e0>
```



In [40]:

```
pd.crosstab(columns=[df.Attrition],index=[df.JobSatisfaction],margins=True,
```

Out[40]:

Attrition	No	Yes
JobSatisfaction		
1	0.771626	0.228374
2	0.835714	0.164286
3	0.834842	0.165158
4	0.886710	0.113290
All	0.838776	0.161224

Note this shows an interesting trend. Note that for higher values of job satisfaction(ie more a person is satisfied with his job) lesser percent of them say a 'Yes' which is quite obvious as highly contented workers will obvioulsy not like to leave the organisation.

3.1.7) Environment Satisfaction

In [41]:
```
pd.crosstab(columns=[df.Attrition],index=[df.EnvironmentSatisfaction],margi
```

Out[41]:

Attrition	No	Yes
EnvironmentSatisfaction		
1	0.746479	0.253521
2	0.850174	0.149826
3	0.863135	0.136865
4	0.865471	0.134529
All	0.838776	0.161224

Again we can notice that the relative percent of 'No' in people with higher grade of environment satisfacftion.

3.1.8) Job Involvement

In [42]:
```
pd.crosstab(columns=[df.Attrition],index=[df.JobInvolvement],margins=True,r
```

Out[42]:

Attrition	No	Yes
JobInvolvement		
1	0.662651	0.337349
2	0.810667	0.189333
3	0.855991	0.144009
4	0.909722	0.090278
All	0.838776	0.161224

3.1.9) Work Life Balance

In [43]:
```
pd.crosstab(columns=[df.Attrition],index=[df.WorkLifeBalance],margins=True,
```

Out[43]:

Attrition	No	Yes
WorkLifeBalance		
1	0.687500	0.312500
2	0.831395	0.168605
3	0.857783	0.142217
4	0.823529	0.176471
All	0.838776	0.161224

Again we notice a similar trend as people with better work life balance dont want to leave the organisation.

3.1.10) RelationshipSatisfaction

In [44]:
```
pd.crosstab(columns=[df.Attrition],index=[df.RelationshipSatisfaction],marg
```

Out[44]:

Attrition	No	Yes
RelationshipSatisfaction		
1	0.793478	0.206522
2	0.851485	0.148515
3	0.845316	0.154684
4	0.851852	0.148148
All	0.838776	0.161224

Notice that I have plotted just some of the important features against out 'Target' variable i.e. Attrition in our case. Similarly we can plot other features against the 'Target' variable and analye the trends i.e. how the feature effects the 'Target' variable.

3.2) Feature Selection

The feature Selection is one of the main steps of the preprocessing phase as the features which we choose directly effects the model performance. While some of the features seem to be less useful in terms of the context; others seem to equally useful. The better features we use the better our model will perform.

We can also use the Recusrive Feature Elimination technique (a wrapper method) to choose the desired number of most important features. The Recursive Feature Elimination (or RFE) works by recursively removing attributes and building a model on those attributes that remain.

It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

We can use it directly from the scikit library by importing the RFE module or function provided by the scikit. But note that since it tries different combinations or the subset of features;it is quite computationally expensive.

In [45]:
```
df.drop(['BusinessTravel','DailyRate','EmployeeCount','EmployeeNumber','Hou
           ,'NumCompaniesWorked','Over18','StandardHours', 'StockOptionLeve
```

4) Preparing Dataset

Before feeding our data into a ML model we first need to prepare the data. This includes encoding all the categorical features (either LabelEncoding or the OneHotEncoding) as the model expects the features to be in numerical form. Also for better performance we will do the feature scaling ie bringing all the features onto the same scale by using the StandardScaler provided in the scikit library.

4.1) Feature Encoding

I have used the Label Encoder from the scikit library to encode all the categorical features.

In [46]:
```
def transform(feature):
    le=LabelEncoder()
    df[feature]=le.fit_transform(df[feature])
    print(le.classes_)
```

In [47]:
```
cat_df=df.select_dtypes(include='object')
cat_df.columns
```

Out[47]:
```
Index(['Attrition', 'Department', 'EducationField', 'Gender', 'JobRole',
       'MaritalStatus', 'OverTime'],
      dtype='object')
```

In [48]:
```
for col in cat_df.columns:
    transform(col)
```

```
['No' 'Yes']
['Human Resources' 'Research & Development' 'Sales']
['Human Resources' 'Life Sciences' 'Marketing' 'Medical' 'Other'
 'Technical Degree']
['Female' 'Male']
['Healthcare Representative' 'Human Resources' 'Laboratory Technician'
 'Manager' 'Manufacturing Director' 'Research Director'
 'Research Scientist' 'Sales Executive' 'Sales Representative']
['Divorced' 'Married' 'Single']
['No' 'Yes']
```

In [49]:
```
df.head() # just to verify.
```

Out[49]:

	Age	Attrition	Department	DistanceFromHome	Education	EducationField	Environm
0	41	1	2	1	2	1	
1	49	0	1	8	1	1	
2	37	1	1	2	2	4	
3	33	0	1	3	4	1	
4	27	0	1	2	1	3	

5 rows × 24 columns

4.2) Feature Scaling

The scikit library provides various types of scalers including MinMax Scaler and the StandardScaler. Below I have used the StandardScaler to scale the data.

Note that the neural networks are quite sensitive towards the scale of the features. Hence it is always good to perform feature scaling on the data before feeding it into an Artificial Neural Network.

In [50]:
```
scaler=StandardScaler()
scaled_df=scaler.fit_transform(df.drop('Attrition',axis=1))
X=scaled_df
Y=df['Attrition'].as_matrix()
```

4.3) One Hot Encoding the Target

Note that there are two main things to watch out before feeding data into an ANN.

The first is that our data needs to be in the form of numpy arrays (ndarray).

The second that the target variable should be one hot encoded eg 2--> 0010 (assuming 0 based indexing) and so on.. In this way for a 'n' class classification problems our target variable will have n classes and hence after one hot encoding we shall have n labels with each label corressponding to a particular target class.

In [51]:
```
Y=to_categorical(Y)
Y
```

Out[51]:
```
array([[0., 1.],
       [1., 0.],
       [0., 1.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]], dtype=float32)
```

4.4) Splitting the data into training and validation sets

In [52]:
```
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_st
```

5) Making Predictions Using an Artificial Neural Network (ANN)

5.1) Handling the Imbalanced dataset

Note that we have a imbalanced dataset with majority of observations being of one type ('NO') in our case. In this dataset for example we have about 84 % of observations having 'No' and only 16 % of 'Yes' and hence this is an imbalanced dataset.

To deal with such a imbalanced dataset we have to take certain measures, otherwise the performance of our model can be significantly affected. In this section I have discussed two approaches to curb such datasets.

5.1.1) Oversampling the Minority or Undersampling the Majority Class

In an imbalanced dataset the main problem is that the data is highly skewed ie the number of observations of certain class is more than that of the other. Therefore what we do in this approach is to either increase the number of observations corressponding to the minority class (oversampling) or decrease the number of observations for the majority class (undersampling).

Note that in our case the number of observations is already pretty low and so oversampling will be more appropriate.

Below I have used an oversampling technique known as the SMOTE(Synthetic Minority Oversampling Technique) which randomly creates some 'Synthetic' instances of the minority class so that the net observations of both the class get balanced out.

One thing more to take of is to use the SMOTE before the cross validation step; just to ensure that our model does not overfit the data; just as in the case of feature selection.

In [53]:

```
# oversampler=SMOTE(random_state=42)
# x_train_smote,  y_train_smote = oversampler.fit_sample(x_train,y_train)
```

5.1.2) Using the Right Evaluation Metric

Another important point while dealing with the imbalanced classes is the choice of right evaluation metrics.

Note that accuracy is not a good choice. This is because since the data is skewed even an algorithm classifying the target as that belonging to the majority class at all times will achieve a very high accuracy. For eg if we have 20 observations of one type 980 of another ; a classifier predicting the majority class at all times will also attain a accuracy of 98 % but doesnt convey any useful information.

Hence in these type of cases we may use other metrics such as -->

'Precision'-- (true positives)/(true positives+false positives)

'Recall'-- (true positives)/(true positives+false negatives)

'F1 Score'-- The harmonic mean of 'precision' and 'recall'

'AUC ROC'-- ROC curve is a plot between 'senstivity' (Recall) and '1-specificity' (Specificity=Precision)

'Confusion Matrix'-- Plot the entire confusion matrix

5.2) Setting the random seeds

Note that in order to get exactly same results after training an artificial neural network at different instances of time we need to specify the random seed for the Keras backend engine which is TensorFlow in my case. Also I have specified the seeds for the python random module as well as for the numpy.

In order to adjust the weights oof an ANN ; the BackProp algorithm starts with a random weights and hence after a given no of epochs the results can be different if the random initialisation of weights is different in starting.

Hence to obtain the same results it is necessary to specify the random seed to get the reproducible results.

In [54]:
```
np.random.seed(42)
```

In [55]:
```
rn.seed(42)
```

In [56]:
```
tf.set_random_seed(42)
```

5.3) Building the Keras model

In [57]:
```
model=Sequential()
model.add(Dense(input_dim=23,units=8,activation='relu'))
model.add(Dense(units=16,activation='relu'))
model.add(Dense(units=2,activation='sigmoid'))
```

BREAKING IT DOWN

1. First we need to build a model. For this we use the Sequential model provided by the Keras which is nothing but a linear stack of layers.

1. Next we need to add the layers to our Sequential model. For this we use the model.add() function.

1. Note that for each layer we need to specify the number of units (or the number of neurons) and also the activation function used by the neurons.

   Note that activation function is used to model complex non-linear relationships and their are many choices. But generally it is preferred to use 'relu' function for the hidden layers and the 'sigmoid' or the 'logistic' function for the output layer. For a multi-class classification problem we can use the 'softmax' function as the activation function for the output layer.

1. Note that the first layer and ONLY the first layer expects the input dimensions in order to know the shape of the input numpy array.

1. Finally note that the number of units or neurons in the final layer is equal to the number of classes of the target variable. In other words for a 'n' class classification problem we shall have 'n' neurons in the output layer.

   Each neuron represents a specific target class. The output of each neuron in the final layer thus represents the probability of given observation being classified to that target class. The observation is classified to the target class; the neuron corressponding to which has the highest value.

5.4) Compiling the Keras model

In [58]:
```
model.compile(optimizer=Adam(lr=0.01),loss='binary_crossentropy',metrics=[
```

BREAKING IT DOWN

1. Now we need to compile the model. We have to specify the optimizer used by the model We have many choices like Adam, RMSprop etc.. Rfer to Keras doc for a comprehensive list of the optimizers available.

1. Next we need to specify the loss function for the neural network which we seek to minimize.

   I have used the 'binary_crossentropy' loss function since this is a binary classification problem. For a multi-class classification problems we may use the 'categorical_crossentropy'.

1. Next we need to specify the metric to evaluate our models performance. Here I have used accuracy.

5.5) Summary of the model

In [59]:

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 8)                 192       
_________________________________________________________________
dense_2 (Dense)              (None, 16)                144       
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 34        
=================================================================
Total params: 370
Trainable params: 370
Non-trainable params: 0
_________________________________________________________________
```

Provides overall description of the model.

5.6) Fitting the model on the training data and testing on the validation set

In [60]:

```
History=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=1(
```

```
Train on 1102 samples, validate on 368 samples
Epoch 1/10
1102/1102 [==============================] - 1s 456us/step - loss: 0.5200 - acc: 0.7772 - val_loss: 0.3944 - val_acc: 0.8696
Epoch 2/10
1102/1102 [==============================] - 0s 39us/step - loss: 0.3971 - acc: 0.8303 - val_loss: 0.3582 - val_acc: 0.8709
Epoch 3/10
1102/1102 [==============================] - 0s 50us/step - loss: 0.3580 - acc: 0.8457 - val_loss: 0.3515 - val_acc: 0.8791
Epoch 4/10
1102/1102 [==============================] - 0s 50us/step - loss: 0.3375 - acc: 0.8525 - val_loss: 0.3479 - val_acc: 0.8818
Epoch 5/10
1102/1102 [==============================] - 0s 42us/step - loss: 0.3272 - acc: 0.8652 - val_loss: 0.3498 - val_acc: 0.8668
Epoch 6/10
1102/1102 [==============================] - 0s 43us/step - loss: 0.3154 - acc: 0.8680 - val_loss: 0.3516 - val_acc: 0.8804
```

```
Epoch 7/10
1102/1102 [==============================] - 0s 45us/step - loss: 0.3115 -
acc: 0.8707 - val_loss: 0.3541 - val_acc: 0.8777
Epoch 8/10
1102/1102 [==============================] - 0s 46us/step - loss: 0.3064 -
acc: 0.8716 - val_loss: 0.3572 - val_acc: 0.8764
Epoch 9/10
1102/1102 [==============================] - 0s 49us/step - loss: 0.2978 -
acc: 0.8752 - val_loss: 0.3554 - val_acc: 0.8804
Epoch 10/10
1102/1102 [==============================] - 0s 46us/step - loss: 0.2969 -
acc: 0.8798 - val_loss: 0.3541 - val_acc: 0.8845
```

BREAKING IT DOWN

1. Lastly we need to fit our model onto the training data just as we do for traditional ML algorithms.

1. We have to specify the training(x_train ,y_train) and the testing (validation_data) sets.

1. We also need to specify the 'number of epochs'. An 'epoch' is one entire cycle of 'Forward & Backward propagation' through all the training examples.

1. Verbose is an optional parameter that just ensures how the output of each epoch is displayed on the screen.

1. We have assigned it to a 'History' variable to retrieve the model performance during each epoch in the future.

5.7) Making Predictions

In [61]:

```
model.predict_classes(x_test)
```

Out[61]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 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, 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, 0, 0, 0,
       0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [62]:

```
model.predict(x_test)
```

Out[62]:

```
array([[8.79308105e-01, 8.12292695e-02],
       [9.71148610e-01, 2.81063337e-02],
       [8.03762555e-01, 1.36793584e-01],
       [9.51201797e-01, 4.88448068e-02],
       [9.95227933e-01, 3.52992350e-03],
```

```
[7.21227765e-01, 2.44345397e-01],
[8.74907970e-01, 1.10384263e-01],
[9.63543653e-01, 3.81244868e-02],
[8.74568224e-01, 1.51544943e-01],
[9.97175455e-01, 2.85343896e-03],
[7.44774342e-01, 2.60270119e-01],
[9.98951912e-01, 2.53731292e-03],
[8.59844625e-01, 1.35331616e-01],
[9.62295294e-01, 3.26739624e-02],
[9.77151036e-01, 1.88689549e-02],
[8.93412769e-01, 7.41400793e-02],
[8.73111546e-01, 1.27169490e-01],
[9.96926963e-01, 4.83195670e-03],
[4.57652986e-01, 4.98768449e-01],
[9.80174661e-01, 1.69818159e-02],
[8.63179028e-01, 1.10279262e-01],
[9.59403634e-01, 4.53657545e-02],
[9.48211730e-01, 5.83794080e-02],
[6.20732188e-01, 3.88382256e-01],
[6.96785867e-01, 3.15230548e-01],
[9.91175532e-01, 1.94532089e-02],
[8.30274999e-01, 1.36309057e-01],
[9.98640120e-01, 1.23613828e-03],
[7.29659915e-01, 2.83266634e-01],
[9.75027621e-01, 2.25708690e-02],
[9.92654443e-01, 9.24290624e-03],
[9.98014808e-01, 2.06787162e-03],
[9.99390364e-01, 5.86275128e-04],
[9.72874880e-01, 3.01390812e-02],
[5.87521970e-01, 4.27139968e-01],
[9.99455988e-01, 7.81073177e-04],
[9.85460341e-01, 9.73227061e-03],
[6.51278257e-01, 3.51047367e-01],
[1.82692930e-01, 8.28915298e-01],
[9.82440233e-01, 1.34497872e-02],
[9.95299816e-01, 7.80557888e-03],
[8.15536082e-01, 1.59208342e-01],
[9.52694356e-01, 4.81933355e-02],
[9.02299583e-01, 1.06719755e-01],
[3.29582483e-01, 6.66044593e-01],
[9.95015919e-01, 5.66603290e-03],
[6.22996747e-01, 3.55247110e-01],
[5.17525315e-01, 5.41745245e-01],
[7.45204151e-01, 2.27850407e-01],
[2.94590387e-02, 9.70117033e-01],
[9.92687345e-01, 8.22171848e-03],
[6.09176815e-01, 4.09472913e-01],
[9.93230343e-01, 8.63955729e-03],
[9.09794688e-01, 7.56604597e-02],
[9.68378425e-01, 3.10450178e-02],
[9.50711429e-01, 4.76681776e-02],
[9.52055871e-01, 5.05449511e-02],
[9.99843121e-01, 1.93249842e-04],
[8.61165524e-01, 1.49787471e-01],
[9.29028809e-01, 7.59601519e-02],
[9.94296253e-01, 6.63177948e-03],
[1.59696117e-01, 8.30677450e-01],
[9.94229436e-01, 6.07449887e-03],
[9.91619229e-01, 8.56497232e-03],
[5.09674907e-01, 4.78069216e-01],
[2.93586701e-01, 7.43983269e-01],
[9.86606956e-01, 1.30643332e-02],
[5.34343600e-01, 4.70601022e-01],
[9.96183097e-01, 3.63092194e-03],
```

```
[9.65062261e-01, 3.40643339e-02],
[7.54825592e-01, 2.33094335e-01],
[9.87684488e-01, 1.26430495e-02],
[7.29179382e-01, 2.41721779e-01],
[9.07512009e-01, 6.74303323e-02],
[5.75408697e-01, 3.81445646e-01],
[9.76144254e-01, 2.45656427e-02],
[9.97126400e-01, 3.51211219e-03],
[9.12821949e-01, 8.68743211e-02],
[7.92953372e-01, 2.03814462e-01],
[1.92899585e-01, 8.19212496e-01],
[9.99045312e-01, 9.64513631e-04],
[9.74263012e-01, 2.50086430e-02],
[9.84625995e-01, 3.00977882e-02],
[9.81000543e-01, 1.91555191e-02],
[9.78358269e-01, 2.69385520e-02],
[8.39233935e-01, 1.59212068e-01],
[6.99977577e-01, 2.48416975e-01],
[9.58151102e-01, 4.57514599e-02],
[9.88833427e-01, 1.12588992e-02],
[9.98197377e-01, 1.64011354e-03],
[9.96814907e-01, 4.30727657e-03],
[8.50160539e-01, 1.47153422e-01],
[7.83366621e-01, 1.78318232e-01],
[8.19815516e-01, 1.77550718e-01],
[9.53814626e-01, 4.84316945e-02],
[9.18697417e-01, 6.25763983e-02],
[9.90967929e-01, 8.67833570e-03],
[7.30599165e-01, 2.28977576e-01],
[7.93860197e-01, 2.09334657e-01],
[8.39466214e-01, 1.68068141e-01],
[8.49789500e-01, 1.56958655e-01],
[9.98891890e-01, 1.15370518e-03],
[9.98813510e-01, 2.37192214e-03],
[9.89487529e-01, 1.19001595e-02],
[9.25520837e-01, 8.99397805e-02],
[9.49749053e-01, 5.57603724e-02],
[9.69693482e-01, 3.26183289e-02],
[8.85503948e-01, 1.40866548e-01],
[9.75944281e-01, 2.38660909e-02],
[9.86752510e-01, 1.31645948e-02],
[4.89995301e-01, 5.29593766e-01],
[3.90814334e-01, 6.12655222e-01],
[9.51851368e-01, 4.71307412e-02],
[8.93751442e-01, 9.74277332e-02],
[9.09294784e-01, 8.31167847e-02],
[8.50031495e-01, 1.39494330e-01],
[8.63894105e-01, 1.43573150e-01],
[9.96460497e-01, 5.51162334e-03],
[9.92702127e-01, 8.24081991e-03],
[7.53506362e-01, 2.44941309e-01],
[9.84539449e-01, 1.39413737e-02],
[9.92696583e-01, 8.05460848e-03],
[6.87694848e-01, 3.17304552e-01],
[9.68291998e-01, 3.57357524e-02],
[7.70896018e-01, 2.00537041e-01],
[4.44659203e-01, 5.30928910e-01],
[8.71135592e-01, 1.28886163e-01],
[9.33877707e-01, 7.73120597e-02],
[7.40758717e-01, 3.96508217e-01],
[9.76730824e-01, 2.27193069e-02],
[9.99721467e-01, 3.79375881e-04],
[4.34825659e-01, 4.91223603e-01],
[9.89936948e-01, 1.46112889e-02],
[5.56147397e-01, 4.60989237e-01]
```

```
       [5.36147397e-01, 4.60989237e-01],
       [9.96123493e-01, 3.56763974e-03],
       [9.61054444e-01, 3.44226845e-02],
       [8.50301802e-01, 1.23024650e-01],
       [9.16961253e-01, 8.11279938e-02],
       [8.02996933e-01, 1.99557394e-01],
       [9.31204617e-01, 8.59519839e-02],
       [3.65661502e-01, 6.45371735e-01],
       [8.31660807e-01, 1.90110713e-01],
       [9.65300798e-01, 3.65048908e-02],
       [9.91504133e-01, 8.67813081e-03],
       [9.72366393e-01, 3.71372104e-02],
       [8.61212969e-01, 1.40714228e-01],
       [9.38885748e-01, 7.11965933e-02],
       [4.68195915e-01, 4.50014472e-01],
       [9.22518849e-01, 5.84240146e-02],
       [5.14829040e-01, 5.03742695e-01],
       [9.80702221e-01, 1.97686739e-02],
       [9.72510397e-01, 3.42849083e-02],
       [6.46767497e-01, 3.71587396e-01],
       [3.99654776e-01, 6.06045306e-01],
       [7.64250696e-01, 2.32334733e-01],
       [7.12167919e-01, 2.57956803e-01],
       [8.48644078e-01, 1.55159444e-01],
       [9.10507739e-01, 8.50184858e-02],
       [9.98067915e-01, 1.99963292e-03],
       [8.94224942e-01, 1.02293231e-01],
       [8.89638782e-01, 1.11035667e-01],
       [9.94553447e-01, 1.02184210e-02],
       [9.99176919e-01, 1.42460584e-03],
       [4.69943196e-01, 4.32112426e-01],
       [9.98869598e-01, 1.28175353e-03],
       [5.99664032e-01, 3.20725411e-01],
       [7.43425488e-01, 2.38723144e-01],
       [9.88155842e-01, 1.42115531e-02],
       [9.46560264e-01, 7.01039433e-02],
       [9.56366837e-01, 3.85955907e-02],
       [9.56240475e-01, 4.76387404e-02],
       [9.91684198e-01, 1.23212198e-02],
       [6.66608393e-01, 3.29607219e-01],
       [7.26291537e-01, 2.26741582e-01],
       [9.97136950e-01, 2.60167266e-03],
       [6.69651031e-01, 2.37388879e-01],
       [9.75390136e-01, 2.93786079e-02],
       [9.24432337e-01, 6.61323518e-02],
       [9.84671891e-01, 1.59794800e-02],
       [6.14992499e-01, 3.72013420e-01],
       [8.13850343e-01, 1.93102062e-01],
       [9.81221616e-01, 2.51434259e-02],
       [9.99024987e-01, 8.67647352e-04],
       [9.23812270e-01, 8.12723264e-02],
       [9.84188914e-01, 1.72821563e-02],
       [8.99492443e-01, 1.06937274e-01],
       [8.32055330e-01, 1.48037717e-01],
       [9.69849527e-01, 4.52025086e-02],
       [9.97811913e-01, 3.22229904e-03],
       [9.80943620e-01, 1.96916610e-02],
       [9.95136797e-01, 5.59973018e-03],
       [6.34783208e-01, 3.29140365e-01],
       [8.94853592e-01, 1.12852819e-01],
       [6.85086071e-01, 3.16487283e-01],
       [9.37607229e-01, 3.65176015e-02],
       [8.44424009e-01, 1.47331089e-01],
       [9.11104679e-01, 9.70624387e-02],
       [9.86364067e-01, 1.49794435e-02],
```

```
[9.47316825e-01, 5.61361983e-02],
[9.24153626e-01, 7.75130391e-02],
[6.85104132e-01, 3.10857385e-01],
[9.79691803e-01, 2.24376656e-02],
[4.53556150e-01, 5.65017402e-01],
[9.94872749e-01, 4.62011900e-03],
[9.91144240e-01, 8.95536132e-03],
[8.35061014e-01, 1.66486412e-01],
[9.55800593e-01, 5.70388958e-02],
[9.96705711e-01, 3.27735650e-03],
[9.14399147e-01, 7.64124468e-02],
[8.97316456e-01, 7.97946602e-02],
[4.56830770e-01, 3.88112903e-01],
[9.14468884e-01, 7.99479857e-02],
[8.69894087e-01, 1.14719935e-01],
[9.17648673e-01, 8.43834579e-02],
[8.58454585e-01, 1.37823835e-01],
[9.57385480e-01, 4.48115170e-02],
[9.81703222e-01, 2.04096790e-02],
[9.99679089e-01, 2.75008380e-04],
[8.34236622e-01, 1.54364720e-01],
[9.67264593e-01, 3.81840877e-02],
[9.48131025e-01, 6.13971986e-02],
[3.97644043e-01, 6.33362710e-01],
[9.90680158e-01, 1.07658179e-02],
[9.13652003e-01, 7.97323659e-02],
[8.62745047e-01, 1.47280380e-01],
[5.74132085e-01, 4.53144282e-01],
[9.64415312e-01, 3.47326510e-02],
[9.42716897e-01, 5.79616055e-02],
[8.38283658e-01, 1.34430617e-01],
[9.56197858e-01, 3.71611156e-02],
[7.71372616e-01, 2.23186895e-01],
[6.27630472e-01, 3.61441016e-01],
[9.90081608e-01, 1.00290207e-02],
[9.86132503e-01, 1.26954671e-02],
[7.27667749e-01, 2.69219518e-01],
[9.87976968e-01, 1.11397579e-02],
[9.99513030e-01, 5.95606572e-04],
[9.83341932e-01, 1.52790975e-02],
[9.74663019e-01, 2.64192820e-02],
[9.94626224e-01, 6.61686249e-03],
[9.93840039e-01, 5.68089774e-03],
[9.92944777e-01, 5.42923436e-03],
[5.55332720e-01, 3.43325764e-01],
[9.96097028e-01, 4.21968754e-03],
[9.91992295e-01, 7.72065576e-03],
[4.06938970e-01, 5.77205539e-01],
[9.78851199e-01, 2.75729299e-02],
[5.89949787e-01, 4.40419048e-01],
[8.04836810e-01, 2.48432249e-01],
[8.23936999e-01, 1.50196806e-01],
[9.06539381e-01, 9.09800678e-02],
[9.77979660e-01, 2.28563342e-02],
[8.37549865e-01, 1.88284934e-01],
[9.99840975e-01, 1.90444916e-04],
[7.43559062e-01, 2.70907968e-01],
[9.98660564e-01, 2.01145792e-03],
[9.56107199e-01, 4.13455218e-02],
[9.87696350e-01, 1.41550153e-02],
[7.20583022e-01, 2.65296936e-01],
[9.98937190e-01, 1.39903126e-03],
[9.63266790e-01, 4.16957512e-02],
[9.65073287e-01, 3.43732312e-02],
```

```
       [8.04845154e-01, 2.03444317e-01],
       [6.46847665e-01, 3.30742389e-01],
       [9.82752681e-01, 1.80042591e-02],
       [7.80258358e-01, 1.61977351e-01],
       [9.96352792e-01, 4.83311480e-03],
       [9.30585742e-01, 6.94497153e-02],
       [8.76807272e-01, 1.27702057e-01],
       [8.51867795e-01, 1.59942165e-01],
       [7.66413927e-01, 2.14174390e-01],
       [9.89218354e-01, 1.16385920e-02],
       [9.59749043e-01, 4.30181772e-02],
       [3.34624410e-01, 6.97713614e-01],
       [9.84470963e-01, 1.25543280e-02],
       [9.68200922e-01, 3.46146971e-02],
       [9.77411866e-01, 2.12204326e-02],
       [9.93061483e-01, 6.97649550e-03],
       [9.88159180e-01, 1.24168657e-02],
       [9.60547447e-01, 4.48548272e-02],
       [9.84900832e-01, 1.94095671e-02],
       [6.37107849e-01, 4.03659552e-01],
       [5.60561955e-01, 3.82265061e-01],
       [9.39024866e-01, 3.02775539e-02],
       [8.95336211e-01, 1.12458080e-01],
       [9.02162254e-01, 9.88117084e-02],
       [7.99950123e-01, 1.58767059e-01],
       [9.50330138e-01, 4.29131500e-02],
       [9.81809378e-01, 1.73892900e-02],
       [9.92780626e-01, 7.41123548e-03],
       [9.04599905e-01, 9.84193608e-02],
       [8.29577327e-01, 1.71600536e-01],
       [9.83609021e-01, 1.66684389e-02],
       [9.87391174e-01, 9.60549526e-03],
       [6.61730051e-01, 2.35990256e-01],
       [8.64495754e-01, 1.34557664e-01],
       [9.93762314e-01, 7.15771411e-03],
       [9.88080978e-01, 1.30316662e-02],
       [9.65216994e-01, 3.66506390e-02],
       [9.94805872e-01, 4.97764535e-03],
       [9.94038463e-01, 9.52498149e-03],
       [7.02020943e-01, 2.89715648e-01],
       [8.48602593e-01, 1.04562961e-01],
       [9.82034087e-01, 2.08220538e-02],
       [4.66187507e-01, 5.36460280e-01],
       [9.74933982e-01, 2.74785906e-02],
       [8.67018700e-01, 1.52531832e-01],
       [9.95359123e-01, 4.45552729e-03],
       [9.26090777e-01, 7.03762248e-02],
       [2.29896605e-01, 7.81528056e-01],
       [7.67866313e-01, 1.95423618e-01],
       [9.44138765e-01, 5.05451001e-02],
       [9.49914217e-01, 4.51161265e-02],
       [9.80708957e-01, 2.11764444e-02],
       [9.56352532e-01, 5.13661020e-02],
       [7.68591821e-01, 2.00479433e-01],
       [9.66306269e-01, 3.44362780e-02],
       [9.02825356e-01, 8.93362463e-02],
       [9.89300251e-01, 1.08886687e-02],
       [9.89854634e-01, 1.23248463e-02],
       [9.93565381e-01, 7.23429443e-03],
       [9.94278312e-01, 5.75818820e-03],
       [9.92714584e-01, 6.34493772e-03],
       [9.58429933e-01, 3.90964709e-02],
       [9.45849538e-01, 5.38173318e-02],
       [9.32135105e-01, 9.96183828e-02],
       [9.24995244e-01, 8.06017593e-02],
```

```
       [9.14493322e-01, 8.96459147e-02],
       [9.01392162e-01, 9.89915282e-02],
       [9.90315259e-01, 1.15637779e-02],
       [9.10140455e-01, 9.37903225e-02],
       [9.34911430e-01, 5.37120737e-02],
       [9.97327924e-01, 2.71083019e-03],
       [9.83707726e-01, 1.55034289e-02],
       [9.97769475e-01, 2.02112878e-03],
       [9.74591672e-01, 2.54948307e-02],
       [9.92675602e-01, 7.07650185e-03],
       [9.67222869e-01, 3.35838571e-02],
       [8.64798367e-01, 1.44274265e-01],
       [9.54147577e-01, 4.78065088e-02],
       [8.49554300e-01, 1.51500165e-01],
       [8.64939928e-01, 1.03064910e-01],
       [9.96942222e-01, 2.83189001e-03],
       [9.99776185e-01, 4.12548310e-04],
       [9.90037322e-01, 6.72798045e-03],
       [5.62903762e-01, 4.25482720e-01],
       [5.11152029e-01, 4.60984409e-01],
       [9.94076133e-01, 1.15511790e-02],
       [7.27190912e-01, 2.45050192e-01],
       [9.27185893e-01, 5.19022420e-02],
       [7.81372428e-01, 1.90896928e-01],
       [9.50014830e-01, 5.41922413e-02],
       [9.91334379e-01, 1.14256665e-02],
       [9.95897651e-01, 4.19255951e-03],
       [9.76296723e-01, 3.12537774e-02],
       [8.59377384e-01, 1.35847583e-01],
       [5.14076412e-01, 4.33043063e-01],
       [5.33948660e-01, 4.43178862e-01],
       [9.99937773e-01, 6.55717886e-05],
       [6.26620233e-01, 3.63654613e-01],
       [6.65686548e-01, 2.81887591e-01],
       [7.91810095e-01, 1.65846244e-01],
       [6.25612140e-01, 3.57640088e-01],
       [8.68408918e-01, 1.34231463e-01],
       [7.64593303e-01, 2.42920578e-01],
       [9.96715188e-01, 3.86464642e-03],
       [9.52666879e-01, 4.88453731e-02],
       [9.81665671e-01, 1.87470540e-02]], dtype=float32)
```

5.8) Evaluating the Model Performance

In [63]:

```
model.evaluate(x_test,y_test)
```

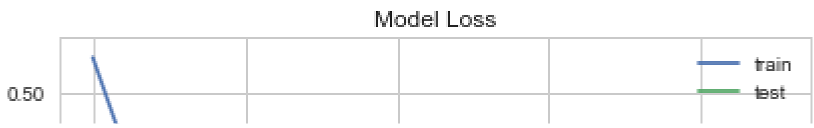
```
368/368 [==============================] - 0s 22us/step
```

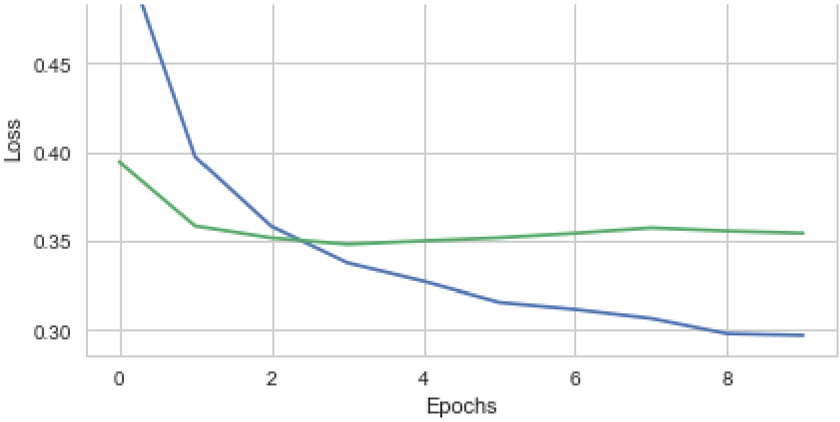
Out[63]:

```
[0.354127524987511, 0.8845108695652174]
```

In [64]:

```
plt.plot(History.history['loss'])
plt.plot(History.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```





In [65]:

```
plt.plot(History.history['acc'])
plt.plot(History.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```