In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
%matplotlib inline
```

Setting the working Directory

In [21]:
```
os.chdir("D:/Great Lakes PGPDSE/Great Lakes/13 Ensemble Techniques/Mini Projec
t")
```

Reading the data set

In [22]:
```
hr = pd.read_csv("HR_Employee_Attrition_Dat.csv")
hr.head()
```

Out[22]:

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

5 rows × 35 columns

In [23]:
```
hr.shape
```

Out[23]: (2940, 35)

There are 35 columns and 2940 data entries in the file

In [24]:
```
hr.dtypes
```

Out[24]:
```
Age                          int64
Attrition                   object
BusinessTravel              object
DailyRate                    int64
Department                  object
DistanceFromHome             int64
Education                    int64
EducationField              object
EmployeeCount                int64
EmployeeNumber               int64
EnvironmentSatisfaction      int64
Gender                      object
HourlyRate                   int64
JobInvolvement               int64
JobLevel                     int64
JobRole                     object
JobSatisfaction              int64
MaritalStatus               object
MonthlyIncome                int64
MonthlyRate                  int64
NumCompaniesWorked           int64
Over18                      object
OverTime                    object
PercentSalaryHike            int64
PerformanceRating            int64
RelationshipSatisfaction     int64
StandardHours                int64
StockOptionLevel             int64
TotalWorkingYears            int64
TrainingTimesLastYear        int64
WorkLifeBalance              int64
YearsAtCompany               int64
YearsInCurrentRole           int64
YearsSinceLastPromotion      int64
YearsWithCurrManager         int64
dtype: object
```

Checking summary statistics

In [25]:
```
hr.describe()
```

Out[25]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	Em
count	2940.000000	2940.000000	2940.000000	2940.000000	2940.0	294
mean	36.923810	802.485714	9.192517	2.912925	1.0	147
std	9.133819	403.440447	8.105485	1.023991	0.0	848
min	18.000000	102.000000	1.000000	1.000000	1.0	1.0
25%	30.000000	465.000000	2.000000	2.000000	1.0	735
50%	36.000000	802.000000	7.000000	3.000000	1.0	147
75%	43.000000	1157.000000	14.000000	4.000000	1.0	220
max	60.000000	1499.000000	29.000000	5.000000	1.0	294

8 rows × 26 columns

Checking for missing values

In [26]:
```
hr[hr.isnull().any(axis=1)]
```

Out[26]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education

0 rows × 35 columns

There are no missing values

In [27]:
```
hr.nunique()
```

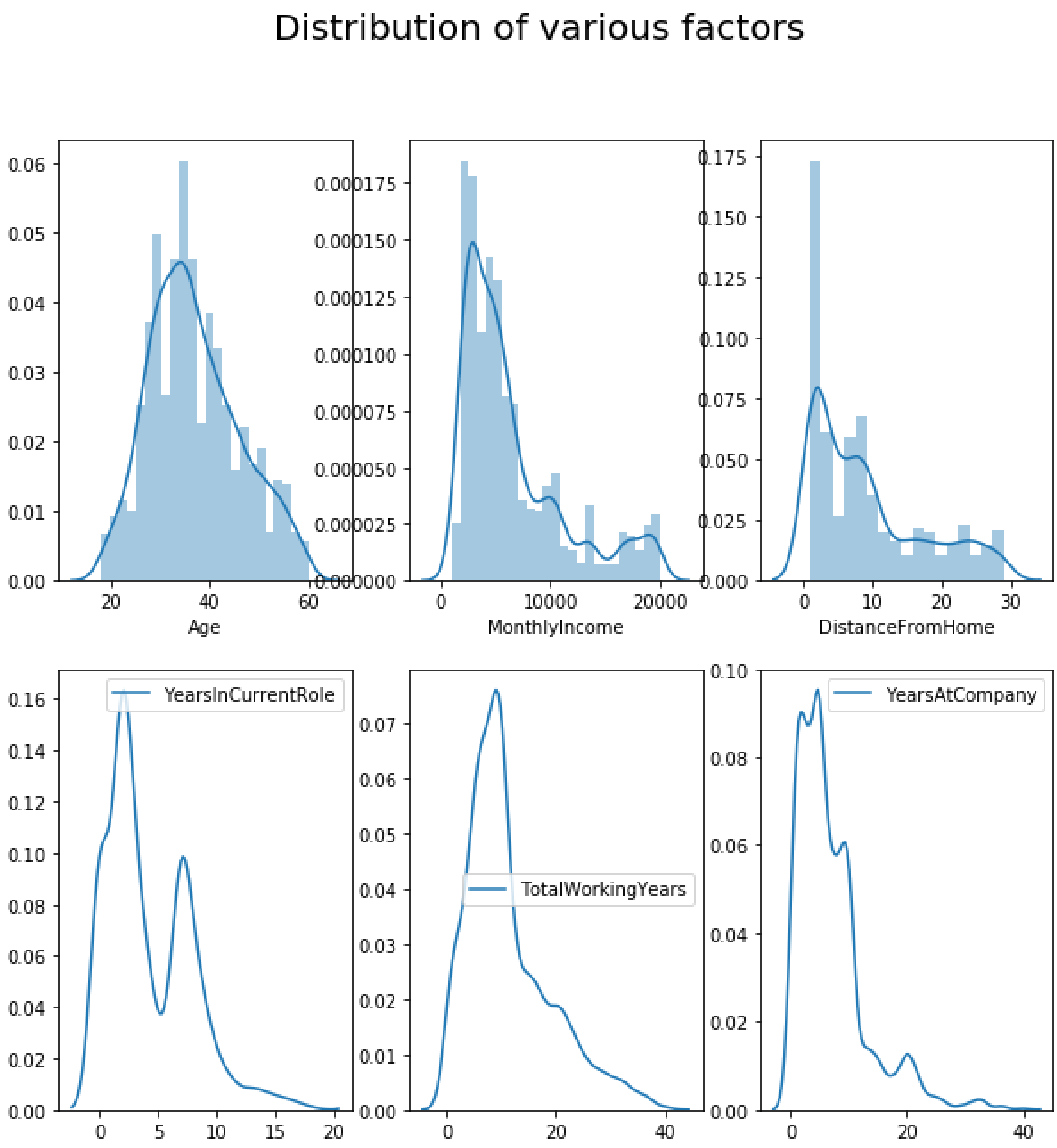
Out[27]:
```
Age                          43
Attrition                     2
BusinessTravel                3
DailyRate                   886
Department                    3
DistanceFromHome             29
Education                     5
EducationField                6
EmployeeCount                 1
EmployeeNumber             2940
EnvironmentSatisfaction       4
Gender                        2
HourlyRate                   71
JobInvolvement                4
JobLevel                      5
JobRole                       9
JobSatisfaction               4
MaritalStatus                 3
MonthlyIncome              1349
MonthlyRate                1427
NumCompaniesWorked           10
Over18                        1
OverTime                      2
PercentSalaryHike            15
PerformanceRating             2
RelationshipSatisfaction      4
StandardHours                 1
StockOptionLevel              4
TotalWorkingYears            40
TrainingTimesLastYear         7
WorkLifeBalance               4
YearsAtCompany               37
YearsInCurrentRole           19
YearsSinceLastPromotion      16
YearsWithCurrManager         18
dtype: int64
```

Here the value for columns, Over18, StandardHours and EmployeeCount same for all rows, we can eliminate these columns.

In [28]:
```
del hr["Over18"]
del hr["EmployeeCount"]
del hr["StandardHours"]
```

Ploting Histogram for various factors in the same plot space.

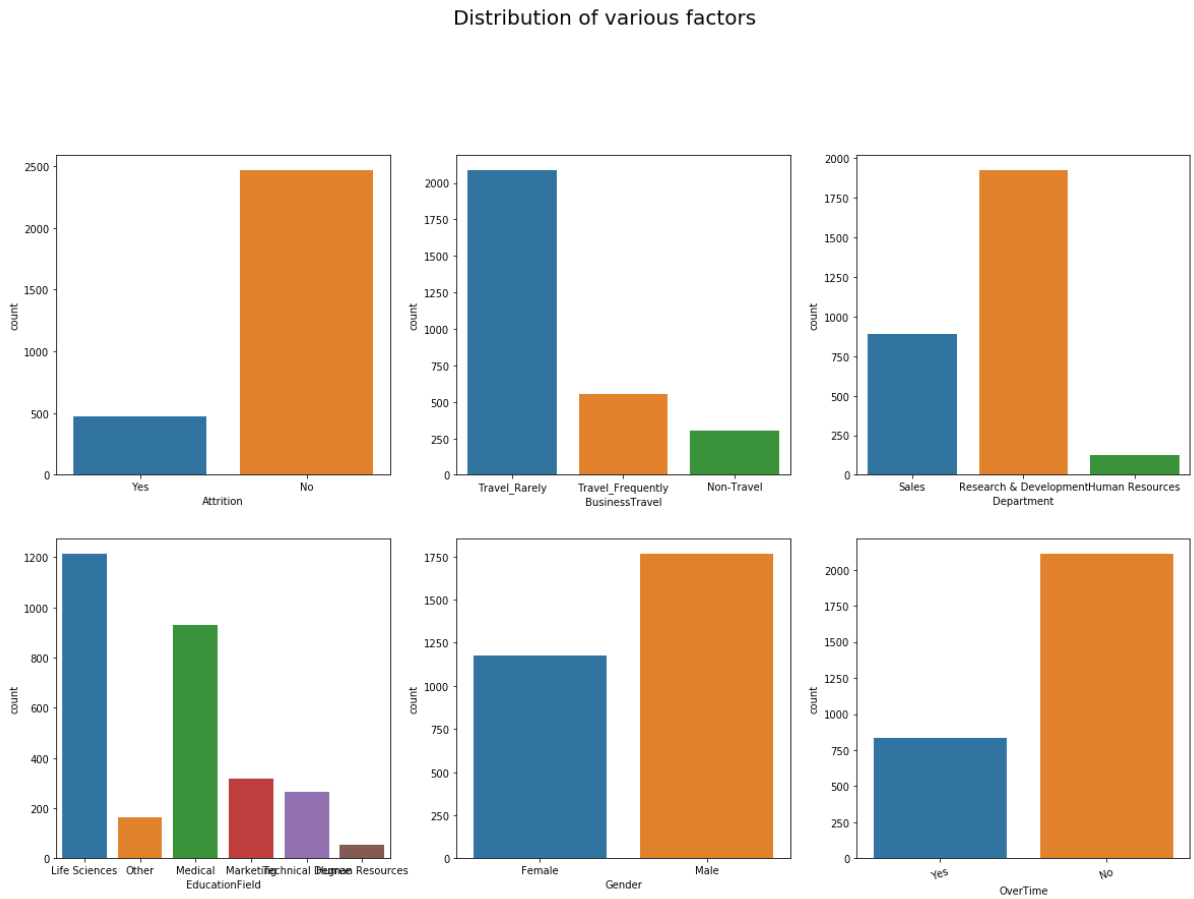
In [29]:
```
fig,ax = plt.subplots(2,3, figsize=(10,10))
plt.suptitle("Distribution of various factors", fontsize=20)
sns.distplot(hr['Age'], ax = ax[0,0])
sns.distplot(hr['MonthlyIncome'], ax = ax[0,1])
sns.distplot(hr['DistanceFromHome'], ax = ax[0,2])
sns.kdeplot(hr['YearsInCurrentRole'], ax = ax[1,0])
sns.kdeplot(hr['TotalWorkingYears'], ax = ax[1,1])
sns.kdeplot(hr['YearsAtCompany'], ax = ax[1,2])
plt.show()
```



Only Age is showing the normal distribution in the data , all other variables not showing the normal distribution

Checking the distribution of various factors

In [30]:
```
fig,ax = plt.subplots(2,3, figsize=(20,20))
plt.suptitle("Distribution of various factors", fontsize=20)
sns.countplot(hr['Attrition'], ax = ax[0,0])
sns.countplot(hr['BusinessTravel'], ax = ax[0,1])
sns.countplot(hr['Department'], ax = ax[0,2])
sns.countplot(hr['EducationField'], ax = ax[1,0])
sns.countplot(hr['Gender'], ax = ax[1,1])
sns.countplot(hr['OverTime'], ax = ax[1,2])
plt.xticks(rotation=20)
plt.subplots_adjust(bottom=0.4)
plt.show()
```

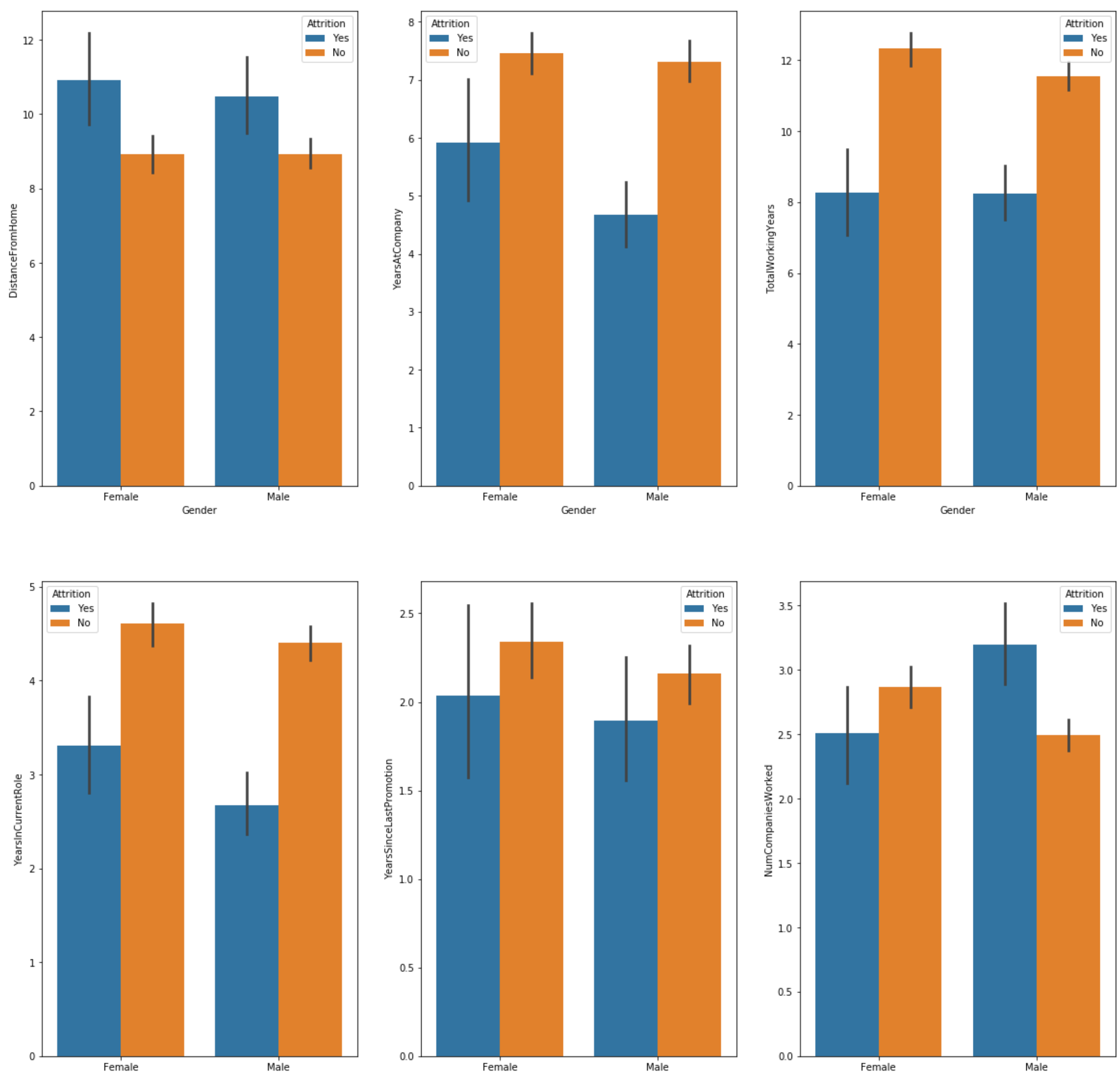


In the data we have, Attrition data is not distributed equally in this dataset. So this dataset is unbalanced data set.

Checking gender wise distribution

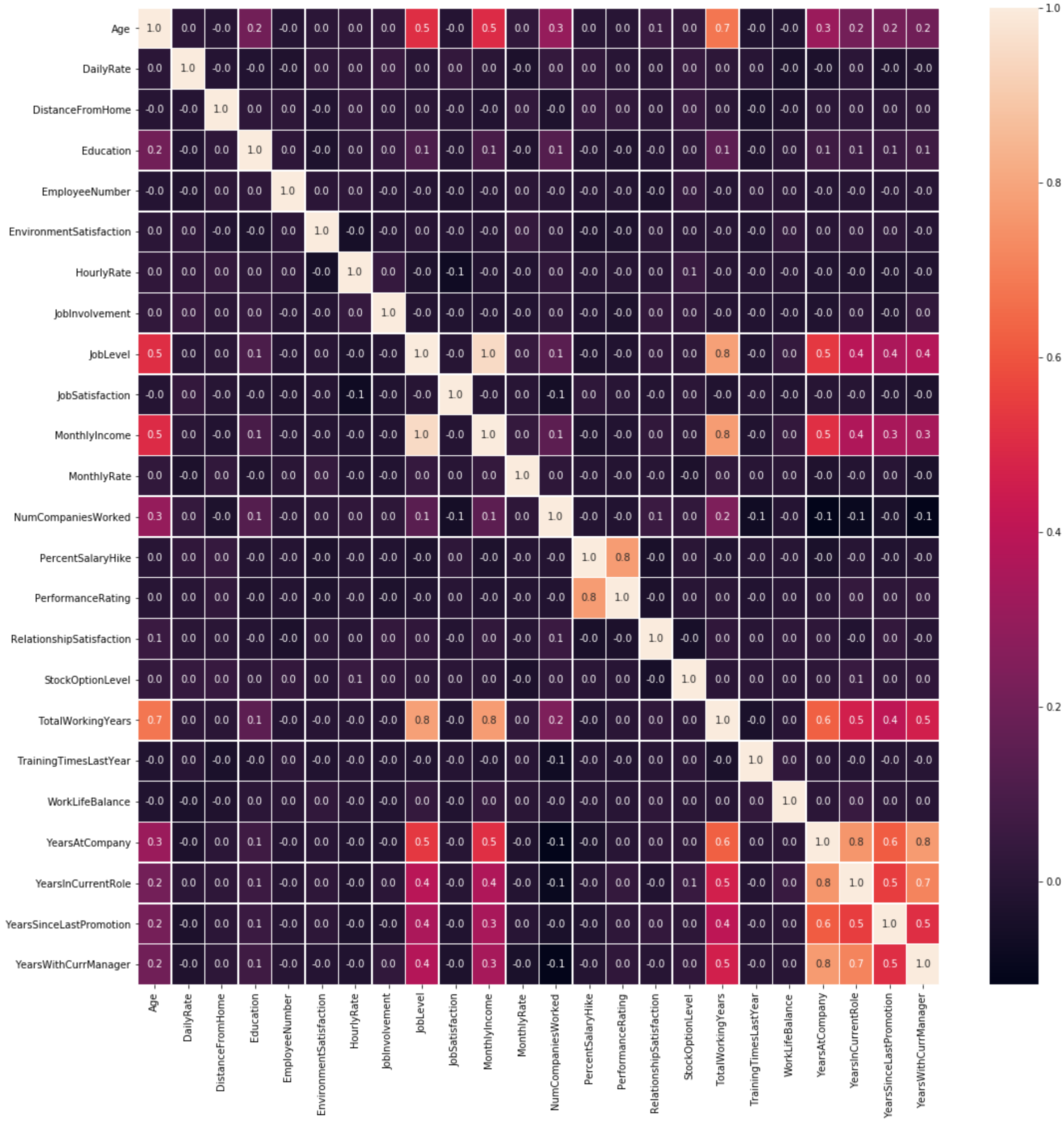
In [31]:
```
fig,ax = plt.subplots(2,3, figsize=(20,20))
plt.suptitle("Distribution of various factors", fontsize=20)
sns.barplot(hr['Gender'],hr['DistanceFromHome'],hue = hr['Attrition'], ax = ax
[0,0]);
sns.barplot(hr['Gender'],hr['YearsAtCompany'],hue = hr['Attrition'], ax = ax[0
,1]);
sns.barplot(hr['Gender'],hr['TotalWorkingYears'],hue = hr['Attrition'], ax = a
x[0,2]);
sns.barplot(hr['Gender'],hr['YearsInCurrentRole'],hue = hr['Attrition'], ax =
ax[1,0]);
sns.barplot(hr['Gender'],hr['YearsSinceLastPromotion'],hue = hr['Attrition'],
ax = ax[1,1]);
sns.barplot(hr['Gender'],hr['NumCompaniesWorked'],hue = hr['Attrition'], ax =
ax[1,2]);
plt.show()
```

Distribution of various factors



Ploting a correlation map for all numeric variables

In [32]:
```
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(hr.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
plt.show()
```



Converting Yes / No values in Attrition column to 1 / 0

In [34]:
```
cleanup_nums = {"Attrition":     {"Yes": 1, "No": 0}}
```

In [35]:
```
hr.replace(cleanup_nums, inplace=True)
hr.head()
```

Out[35]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educati
0	41	1	Travel_Rarely	1102	Sales	1	2
1	49	0	Travel_Frequently	279	Research & Development	8	1
2	37	1	Travel_Rarely	1373	Research & Development	2	2
3	33	0	Travel_Frequently	1392	Research & Development	3	4
4	27	0	Travel_Rarely	591	Research & Development	2	1

5 rows × 32 columns

Splitting to training and testing data

In [36]:
```
from sklearn.cross_validation import train_test_split
import random
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: De
precationWarning: This module was deprecated in version 0.18 in favor of the
model_selection module into which all the refactored classes and functions ar
e moved. Also note that the interface of the new CV iterators are different f
rom that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [38]:
```
np.random.seed(1010)
train,test = train_test_split( hr, test_size = 0.3)
```

Checking distribution of Attribution in each group

In [39]:
```
hr.Attrition.value_counts()
```

Out[39]:
```
0    2466
1     474
Name: Attrition, dtype: int64
```

In [14]:
```
474/(2466+474)
```

Out[14]:
```
0.16122448979591836
```

In [40]:
```
test.Attrition.value_counts()
```

Out[40]:
```
0    727
1    155
Name: Attrition, dtype: int64
```

In [43]:
```
155/(155+727)
```

Out[43]:
```
0.17573696145124718
```

In [41]:
```
train.Attrition.value_counts()
```

Out[41]:
```
0    1739
1     319
Name: Attrition, dtype: int64
```

In [44]:
```
319/(319+1739)
```

Out[44]:
```
0.15500485908649175
```

Split of attribution is approxsame for both train and test data

Spliting target variable y_train and y_test and independent variables X1 and X2 variables

In [45]:
```
X1 =  train[['Age', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager']]
X2 =  test[['Age', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager']]
```

In [46]:
```
y_train = train["Attrition"]
y_test = test["Attrition"]
```

Categorical Variable to Numerical Variables

In [47]:
```
X_train = pd.get_dummies(X1)
X_test = pd.get_dummies(X2)
X_train.columns
```

Out[47]:
```
Index(['Age', 'DailyRate', 'DistanceFromHome', 'Education',
       'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
       'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorke
d',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager',
       'BusinessTravel_Non-Travel', 'BusinessTravel_Travel_Frequently',
       'BusinessTravel_Travel_Rarely', 'Department_Human Resources',
       'Department_Research & Development', 'Department_Sales',
       'EducationField_Human Resources', 'EducationField_Life Sciences',
       'EducationField_Marketing', 'EducationField_Medical',
       'EducationField_Other', 'EducationField_Technical Degree',
       'Gender_Female', 'Gender_Male', 'JobRole_Healthcare Representative',
       'JobRole_Human Resources', 'JobRole_Laboratory Technician',
       'JobRole_Manager', 'JobRole_Manufacturing Director',
       'JobRole_Research Director', 'JobRole_Research Scientist',
       'JobRole_Sales Executive', 'JobRole_Sales Representative',
       'MaritalStatus_Divorced', 'MaritalStatus_Married',
       'MaritalStatus_Single', 'OverTime_No', 'OverTime_Yes'],
      dtype='object')
```

Applying Random Forest for the above data

Scaling the data

In [48]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [49]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [50]:
```
model = RandomForestClassifier()
```

In [51]:
```
model.fit(X_train,y_train)
```

Out[51]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [52]:
```
pred_y_train = model.predict(X_train)
pred_y_train
```

Out[52]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

Let us see the classification accuracy of our model

In [54]:
```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_train, pred_y_train)
score
```

Out[54]: 0.9941690962099126

AUC

In [55]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc,confusion_matrix

y_train_prob = model.predict_proba(X_train)
fpr, tpr, thresholds =  roc_curve(y_train, y_train_prob[:,1])
auc(fpr, tpr)
```

Out[55]: 0.9999296969216265

Checking for Test data

In [56]:
```
pred_y_test = model.predict(X_test)
pred_y_test

## Let us see the classification accuracy of our model
score_test = accuracy_score(y_test, pred_y_test)
score_test
```

Out[56]: 0.9319727891156463

In [57]:
```
y_test_prob = model.predict_proba(X_test)
fpr, tpr, thresholds =  roc_curve(y_test, y_test_prob[:,1])
auc(fpr, tpr)
```

Out[57]: 0.9500377157563118

In [58]:
```
from sklearn import model_selection
scores = model_selection.cross_val_score(model, X_train, y_train, cv = 10, sco
ring='roc_auc')
scores.mean()
```

Out[58]: 0.9422174385315734

In [59]:
```
scores.std()
```

Out[59]: 0.020981436334551497

So by cross validation we get the correct AUC of the model, that is 94.26% is the correct AUC with standard deviation of 0.020. For test data and model selection through cross validation is almost same, but auc for training data is high. so the model is overfitted.

Checking important variables and arranging in the descending order

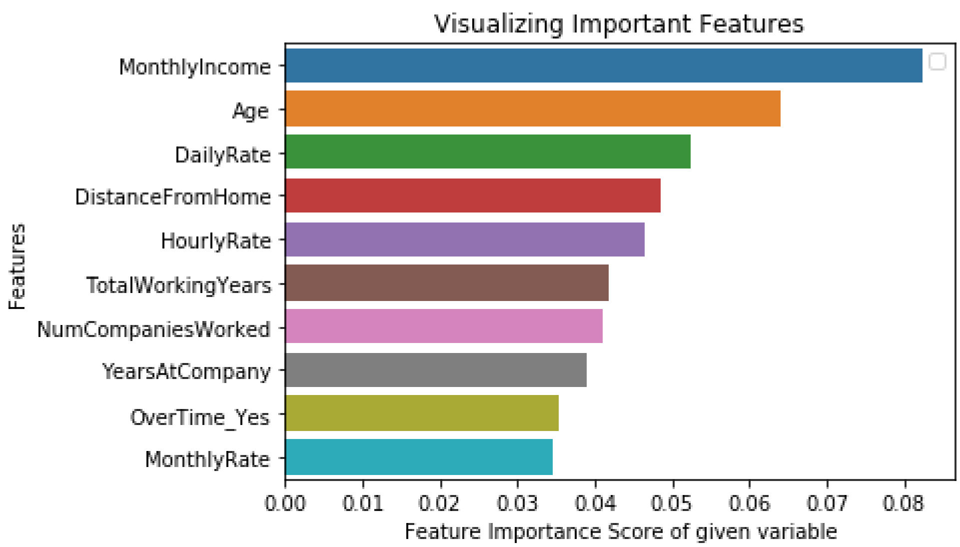
In [61]:
```
import pandas as pd
feature_imp = pd.Series(model.feature_importances_,index= ['Age', 'DailyRate',
 'DistanceFromHome', 'Education',
       'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
       'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked'
,
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager',
       'BusinessTravel_Non-Travel', 'BusinessTravel_Travel_Frequently',
       'BusinessTravel_Travel_Rarely', 'Department_Human Resources',
       'Department_Research & Development', 'Department_Sales',
       'EducationField_Human Resources', 'EducationField_Life Sciences',
       'EducationField_Marketing', 'EducationField_Medical',
       'EducationField_Other', 'EducationField_Technical Degree',
       'Gender_Female', 'Gender_Male', 'JobRole_Healthcare Representative',
       'JobRole_Human Resources', 'JobRole_Laboratory Technician',
       'JobRole_Manager', 'JobRole_Manufacturing Director',
       'JobRole_Research Director', 'JobRole_Research Scientist',
       'JobRole_Sales Executive', 'JobRole_Sales Representative',
       'MaritalStatus_Divorced', 'MaritalStatus_Married',
       'MaritalStatus_Single', 'OverTime_No', 'OverTime_Yes'] ).sort_values(as
cending=False)
feature_imp
```

```
Out[61]: MonthlyIncome                        0.082483
         Age                                  0.064031
         DailyRate                            0.052367
         DistanceFromHome                     0.048545
         HourlyRate                           0.046515
         TotalWorkingYears                    0.041863
         NumCompaniesWorked                   0.041041
         YearsAtCompany                       0.038896
         OverTime_Yes                         0.035306
         MonthlyRate                          0.034621
         EnvironmentSatisfaction              0.032266
         YearsSinceLastPromotion              0.031774
         YearsWithCurrManager                 0.030587
         WorkLifeBalance                      0.030251
         StockOptionLevel                     0.029649
         PercentSalaryHike                    0.029339
         YearsInCurrentRole                   0.028878
         TrainingTimesLastYear                0.024228
         JobSatisfaction                      0.024214
         RelationshipSatisfaction             0.022147
         JobInvolvement                       0.021971
         MaritalStatus_Single                 0.017762
         Education                            0.016408
         JobLevel                             0.015559
         BusinessTravel_Travel_Frequently     0.014292
         OverTime_No                          0.013723
         EducationField_Medical               0.010010
         JobRole_Sales Representative         0.009031
         Gender_Male                          0.008289
         JobRole_Laboratory Technician        0.008053
         EducationField_Marketing             0.007692
         BusinessTravel_Travel_Rarely         0.007366
         Department_Sales                     0.006842
         Gender_Female                        0.006837
         Department_Research & Development    0.006449
         EducationField_Life Sciences         0.006368
         EducationField_Technical Degree      0.005867
         MaritalStatus_Married                0.005426
         JobRole_Research Scientist           0.005175
         BusinessTravel_Non-Travel            0.005132
         EducationField_Other                 0.004881
         MaritalStatus_Divorced               0.004880
         JobRole_Sales Executive              0.004716
         PerformanceRating                    0.004259
         JobRole_Manufacturing Director       0.002619
         Department_Human Resources           0.002533
         EducationField_Human Resources       0.002338
         JobRole_Healthcare Representative    0.002277
         JobRole_Research Director            0.002218
         JobRole_Human Resources              0.001650
         JobRole_Manager                      0.000375
         dtype: float64
```

In [62]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
feature_imp=feature_imp[0:10,]

sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score of given variable')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```



Here most important variable is MonthlyIncome followed by Age and Daily rate

Parameter Tuning for the Random forest model

In [63]:
```
import time
from sklearn.grid_search import GridSearchCV
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: Depreca
tionWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are mov
ed. This module will be removed in 0.20.
  DeprecationWarning)
```

In [64]:
```
np.random.seed(42)
start = time.time()

param_dist = {

              'n_estimators':np.arange(10,20),
              'max_depth': [2, 3, 4],
              'bootstrap': [True, False],
              'max_features': ['auto', 'sqrt', 'log2', None],
              'criterion': ['gini', 'entropy']}

cv_rf = GridSearchCV(model, cv = 10,
                     param_grid=param_dist,
                     n_jobs = 3)

cv_rf.fit(X_train, y_train)
print('Best Parameters using grid search: \n',
      cv_rf.best_params_)
end = time.time()
print('Time taken in grid search: {0: .2f}'.format(end - start))
```

```
Best Parameters using grid search:
 {'bootstrap': True, 'criterion': 'gini', 'max_depth': 4, 'max_features': Non
e, 'n_estimators': 19}
Time taken in grid search:  236.21
```

In [65]:
```
classifier = cv_rf.best_estimator_
classifier.fit(X_train,y_train)
```

Out[65]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=4, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=19, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

AUC

In [66]:
```
y_train_prob = classifier.predict_proba(X_train)
fpr, tpr, thresholds = roc_curve(y_train, y_train_prob[:,1])
auc_d = auc(fpr, tpr)
auc_d
```

Out[66]: 0.8709199428201629

In [67]:
```
y_test_prob = classifier.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_test_prob[:,1])
auc_h = auc(fpr, tpr)
auc_h
```

Out[67]: 0.8300927363890491

Now the model seems more good. AUC for both data is almost same with 4% difference between train dataset and test dataset.

In [69]:
```
Prediction = classifier.predict_proba(X_train)
train["prob_score"] = Prediction[:,1]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

scoring and rank ordering

decile code

In [71]:
```
def deciles(x):
    decile = pd.Series(index=[0,1,2,3,4,5,6,7,8,9])
    for i in np.arange(0.1,1.1,0.1):
        decile[int(i*10)]=x.quantile(i)
    def z(x):
        if x<decile[1]: return(1)
        elif x<decile[2]: return(2)
        elif x<decile[3]: return(3)
        elif x<decile[4]: return(4)
        elif x<decile[5]: return(5)
        elif x<decile[6]: return(6)
        elif x<decile[7]: return(7)
        elif x<decile[8]: return(8)
        elif x<decile[9]: return(9)
        elif x<=decile[10]: return(10)
        else:return(np.NaN)
    s=x.map(z)
    return(s)
```

In [72]:
```
def Rank_Ordering(X,y,Target):
    X['decile']=deciles(X[y])
    Rank=X.groupby('decile').apply(lambda x: pd.Series([
        np.min(x[y]),
        np.max(x[y]),
        np.mean(x[y]),
        np.size(x[y]),
        np.sum(x[Target]),
        np.size(x[Target][x[Target]==0]),
        ],
        index=(["min_resp","max_resp","avg_resp",
                "cnt","cnt_resp","cnt_non_resp"])
        )).reset_index()
    Rank = Rank.sort_values(by='decile',ascending=False)
    Rank["rrate"] = Rank["cnt_resp"]*100/Rank["cnt"]
    Rank["cum_resp"] = np.cumsum(Rank["cnt_resp"])
    Rank["cum_non_resp"] = np.cumsum(Rank["cnt_non_resp"])
    Rank["cum_resp_pct"] = Rank["cum_resp"]/np.sum(Rank["cnt_resp"])
    Rank["cum_non_resp_pct"]=Rank["cum_non_resp"]/np.sum(Rank["cnt_non_resp"])
    Rank["KS"] = Rank["cum_resp_pct"] - Rank["cum_non_resp_pct"]
    Rank
    return(Rank)
```

Rank ordering for train data

In [74]:
```
Rank = Rank_Ordering(train,"prob_score","Attrition")
Rank
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

Out[74]:

	decile	min_resp	max_resp	avg_resp	cnt	cnt_resp	cnt_non_resp	rrate	cum
9	10	0.297405	0.950011	0.522539	206.0	161.0	45.0	78.155340	161.
8	9	0.191513	0.295821	0.232902	207.0	63.0	144.0	30.434783	224.
7	8	0.145209	0.190643	0.166641	205.0	32.0	173.0	15.609756	256.
6	7	0.118834	0.145195	0.132132	205.0	21.0	184.0	10.243902	277.
5	6	0.101833	0.118789	0.109218	206.0	8.0	198.0	3.883495	285.
4	5	0.092359	0.101824	0.096665	206.0	8.0	198.0	3.883495	293.
3	4	0.078050	0.092211	0.087245	206.0	8.0	198.0	3.883495	301.
2	3	0.059630	0.078050	0.067872	211.0	6.0	205.0	2.843602	307.
1	2	0.054601	0.059538	0.056443	203.0	5.0	198.0	2.463054	312.
0	1	0.047246	0.054419	0.050603	203.0	7.0	196.0	3.448276	319.

Response rate in top deciles is 80.25% and Highest ks value 59.94%. This looks bit overfitting.

Rank ordering for test data

In [75]:
```
Prediction_h = classifier.predict_proba(X_test)
test["prob_score"] = Prediction_h[:,1]

Rank_h = Rank_Ordering(test,"prob_score","Attrition")
Rank_h
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

Out[75]:

	decile	min_resp	max_resp	avg_resp	cnt	cnt_resp	cnt_non_resp	rrate	cum_
9	10	0.301090	0.936781	0.507944	89.0	64.0	25.0	71.910112	64.0
8	9	0.184474	0.300052	0.224480	88.0	28.0	60.0	31.818182	92.0
7	8	0.148623	0.183026	0.165121	89.0	18.0	71.0	20.224719	110.0
6	7	0.123403	0.147540	0.135241	87.0	13.0	74.0	14.942529	123.0
5	6	0.103882	0.123064	0.111437	92.0	15.0	77.0	16.304348	138.0
4	5	0.095047	0.103834	0.099146	84.0	4.0	80.0	4.761905	142.0
3	4	0.085524	0.094750	0.090676	89.0	3.0	86.0	3.370787	145.0
2	3	0.061764	0.085384	0.070817	87.0	1.0	86.0	1.149425	146.0
1	2	0.054601	0.061516	0.057357	92.0	6.0	86.0	6.521739	152.0
0	1	0.049909	0.054419	0.050659	85.0	3.0	82.0	3.529412	155.0

Here on testing the model on test data we are getting KS value of 49.50% and response rate of 70.22% in the top decile. This looks very good but compared to the training data this is less. The model looks bit over fitted.

Here the employees who all comes in the 10th,9th and 8th decile are most likely to leave the company. Let’s assume that training of new employee costs 1000 dollar and since we know which employee is likely to leave next month, and propose him/her a bonus program worth 500 to keep him for next 6 months, we are 500 dollar to keep him for next 6 months,we are 500 dollar on plus and keep experienced, well-trained employee under the hood, with higher morale. So better to take care of these employees and company should be prepared to find substitution for those.Also to find the better experienced resource company will get time to search better alternative resource.