GAURAV SAHU

GITHUB LINK: https://github.com/GauravSahu13

IBM HR EMPLOYEE ATTRITION

In [2]:

```
import pandas as pd
import seaborn as sns
import numpy as np
from statistics import mean
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [22]:

```
!pip install wget
```

```
Collecting wget
  Downloading wget-3.2.zip (10 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9655 sha256=85a0ea154a55e0b2
7af40c2394a7f138f6a082774359f02a038b7806dc11f98f
  Stored in directory: /root/.cache/pip/wheels/8b/f1/7f/5c94f0a7a505ca1c81cd1d9208ae2064675d97
582078e6c769
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```



```
# Load csv file
df = pd.read_csv('HR-Employee-Attrition.csv')
df.head()
```

Out[23]:

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

5 rows × 35 columns

In [24]:

```
df.shape
```

Out[24]: (1470, 35)

In [25]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [26]:

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

Out[26]:

```
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 [27]:
```
df.nunique()
```

Out[27]:
```
Age                           43
Attrition                      2
BusinessTravel                 3
DailyRate                    886
Department                     3
DistanceFromHome              29
Education                      5
EducationField                 6
EmployeeCount                  1
EmployeeNumber              1470
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
```

In [28]:
```
df.duplicated().sum()
```

Out[28]:
```
0
```

In [29]:
```
df.describe().T
```

Out[29]:

	count	mean	std	min	25%	50%	75%	ma
Age	1470.0	36.923810	9.135373	18.0	30.00	36.0	43.00	60
DailyRate	1470.0	802.485714	403.509100	102.0	465.00	802.0	1157.00	1499
DistanceFromHome	1470.0	9.192517	8.106864	1.0	2.00	7.0	14.00	29
Education	1470.0	2.912925	1.024165	1.0	2.00	3.0	4.00	5
EmployeeCount	1470.0	1.000000	0.000000	1.0	1.00	1.0	1.00	1
EmployeeNumber	1470.0	1024.865306	602.024335	1.0	491.25	1020.5	1555.75	2068
EnvironmentSatisfaction	1470.0	2.721769	1.093082	1.0	2.00	3.0	4.00	4
HourlyRate	1470.0	65.891156	20.329428	30.0	48.00	66.0	83.75	100

JobInvolvement	1470.0	2.729932	0.711561	1.0	2.00	3.0	3.00	4
JobLevel	1470.0	2.063946	1.106940	1.0	1.00	2.0	3.00	5
JobSatisfaction	1470.0	2.728571	1.102846	1.0	2.00	3.0	4.00	4
MonthlyIncome	1470.0	6502.931293	4707.956783	1009.0	2911.00	4919.0	8379.00	19999
MonthlyRate	1470.0	14313.103401	7117.786044	2094.0	8047.00	14235.5	20461.50	26999
NumCompaniesWorked	1470.0	2.693197	2.498009	0.0	1.00	2.0	4.00	9
PercentSalaryHike	1470.0	15.209524	3.659938	11.0	12.00	14.0	18.00	25
PerformanceRating	1470.0	3.153741	0.360824	3.0	3.00	3.0	3.00	4
RelationshipSatisfaction	1470.0	2.712245	1.081209	1.0	2.00	3.0	4.00	4
StandardHours	1470.0	80.000000	0.000000	80.0	80.00	80.0	80.00	80
StockOptionLevel	1470.0	0.793878	0.852077	0.0	0.00	1.0	1.00	3
TotalWorkingYears	1470.0	11.279592	7.780782	0.0	6.00	10.0	15.00	40
TrainingTimesLastYear	1470.0	2.799320	1.289271	0.0	2.00	3.0	3.00	6
WorkLifeBalance	1470.0	2.761224	0.706476	1.0	2.00	3.0	3.00	4
YearsAtCompany	1470.0	7.008163	6.126525	0.0	3.00	5.0	9.00	40
YearsInCurrentRole	1470.0	4.229252	3.623137	0.0	2.00	3.0	7.00	18
YearsSinceLastPromotion	1470.0	2.187755	3.222430	0.0	0.00	1.0	3.00	15
YearsWithCurrManager	1470.0	4.123129	3.568136	0.0	2.00	3.0	7.00	17

In [30]:

```
df.columns
```

Out[30]:

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

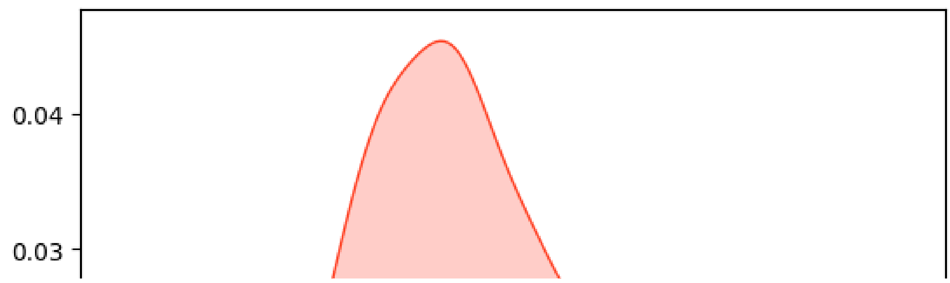
In all we have 34 features consisting of both the categorical as well as the numerical features. The target variable is the 'Attrition' of the employee which can be either a Yes or a No.

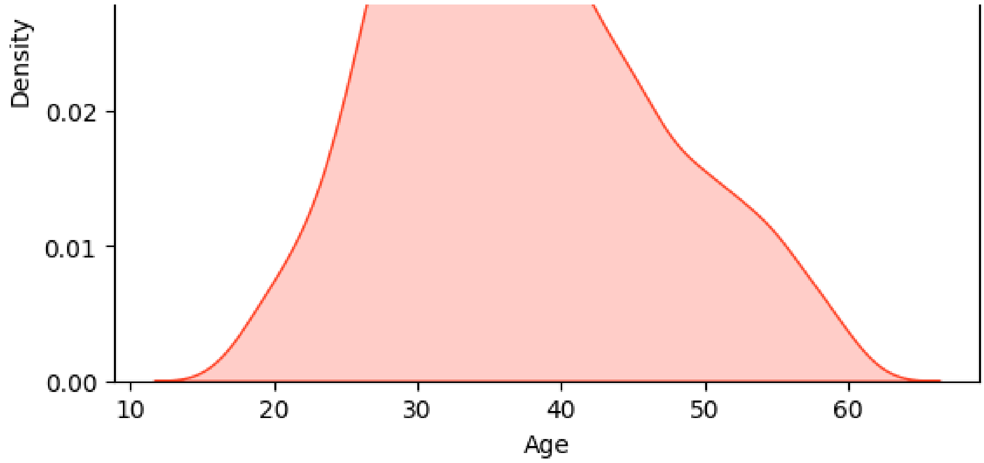
Univariate Analysis

In [31]:

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

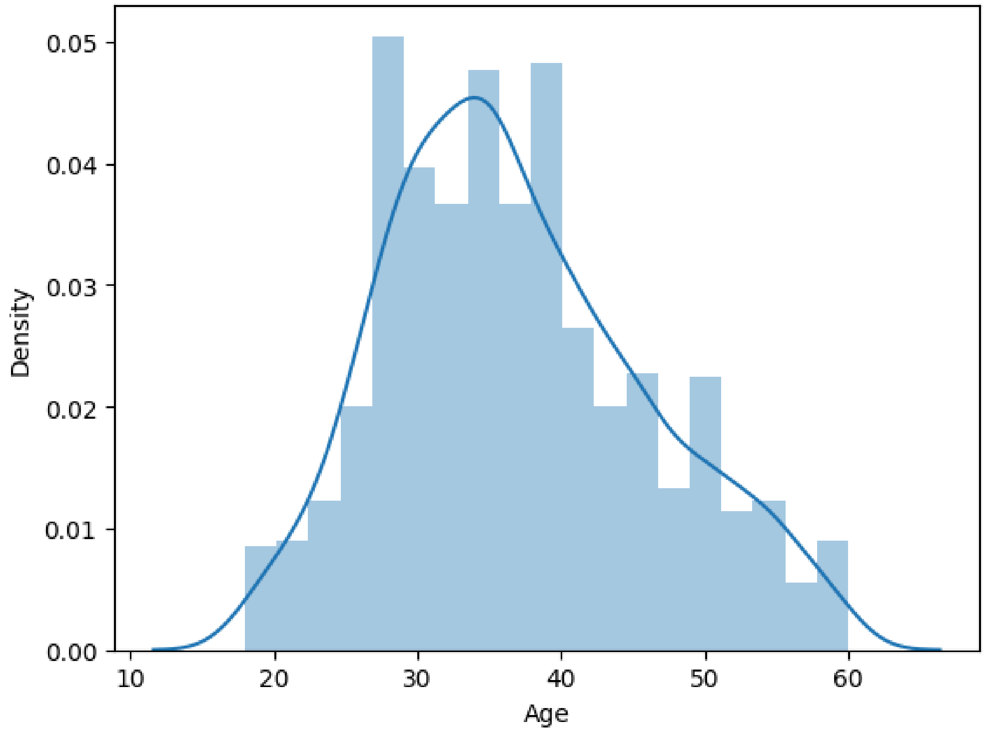
Out[31]: <Axes: xlabel='Age', ylabel='Density'>



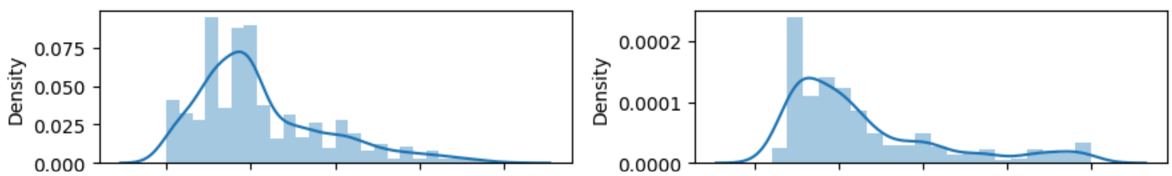


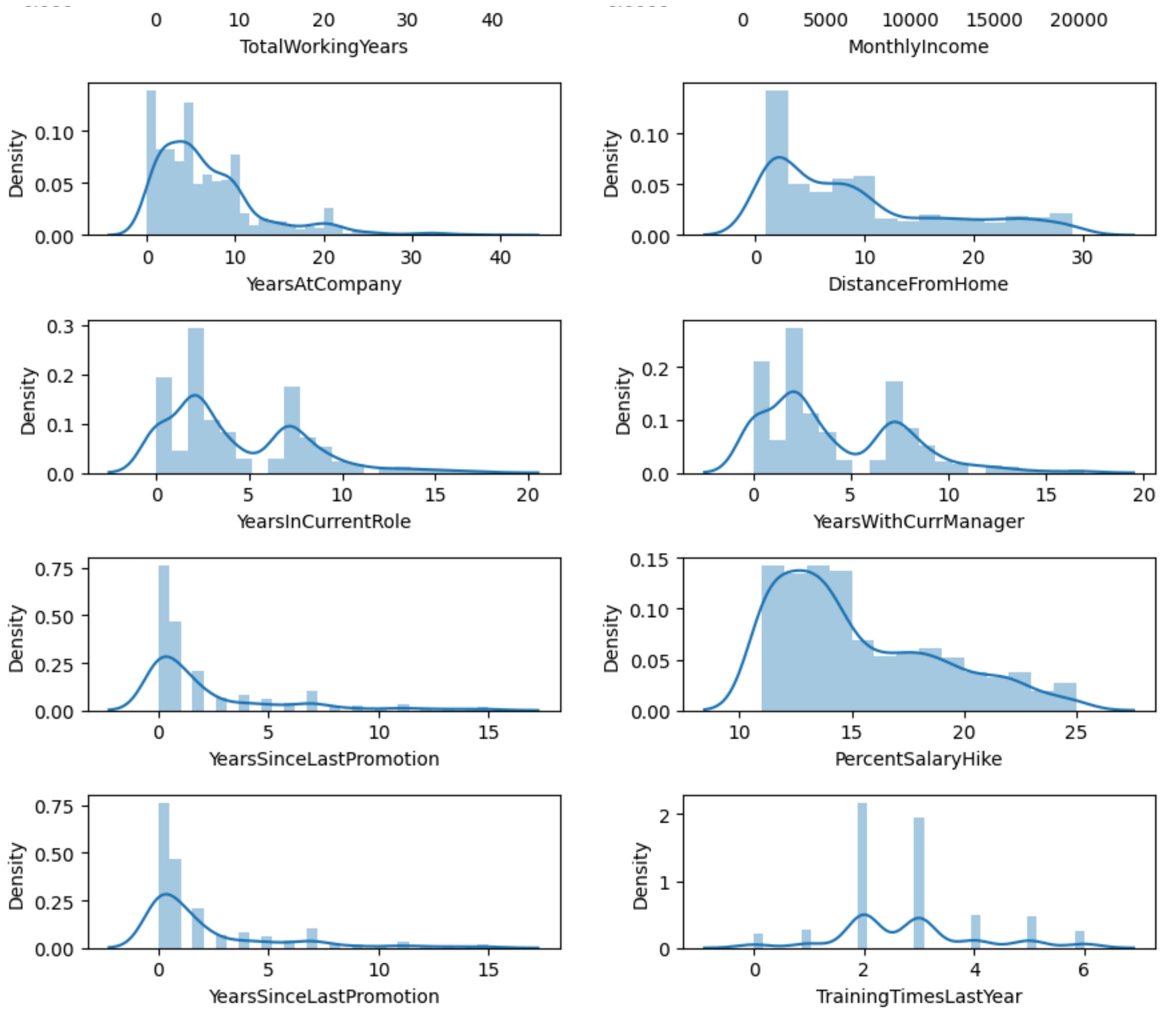
In [32]:
```
sns.distplot(df['Age'])
```

Out[32]: <Axes: xlabel='Age', ylabel='Density'>



In [33]:
```
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()
```





In [34]:

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

Out[34]:

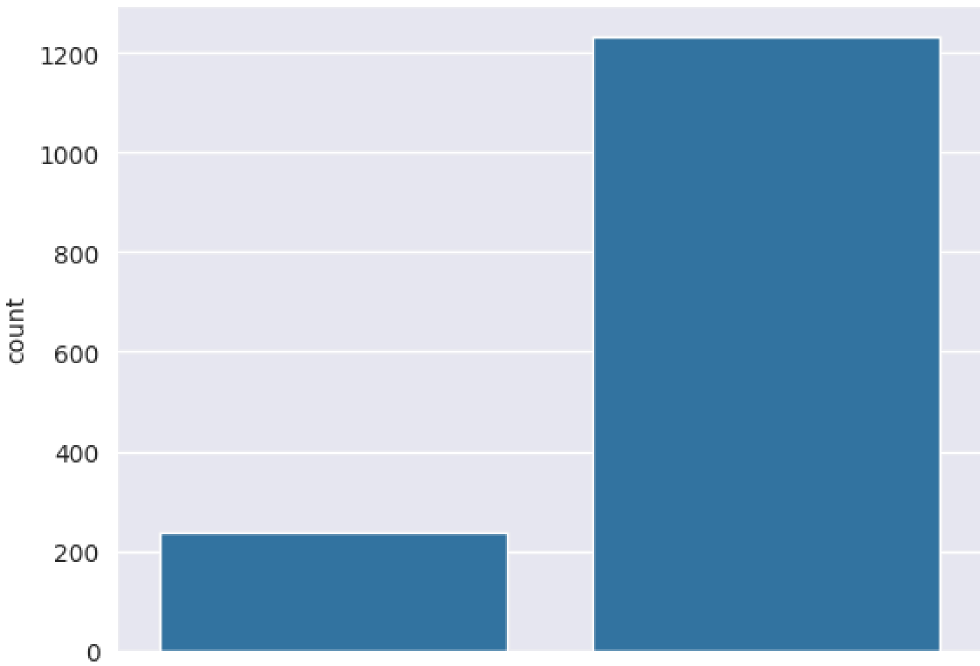
```
Index(['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender',
       'JobRole', 'MaritalStatus', 'Over18', 'OverTime'],
      dtype='object')
```

In [35]:

```
sns.set_style('darkgrid')
sns.countplot(x ='Attrition', data = df)
```

Out[35]:

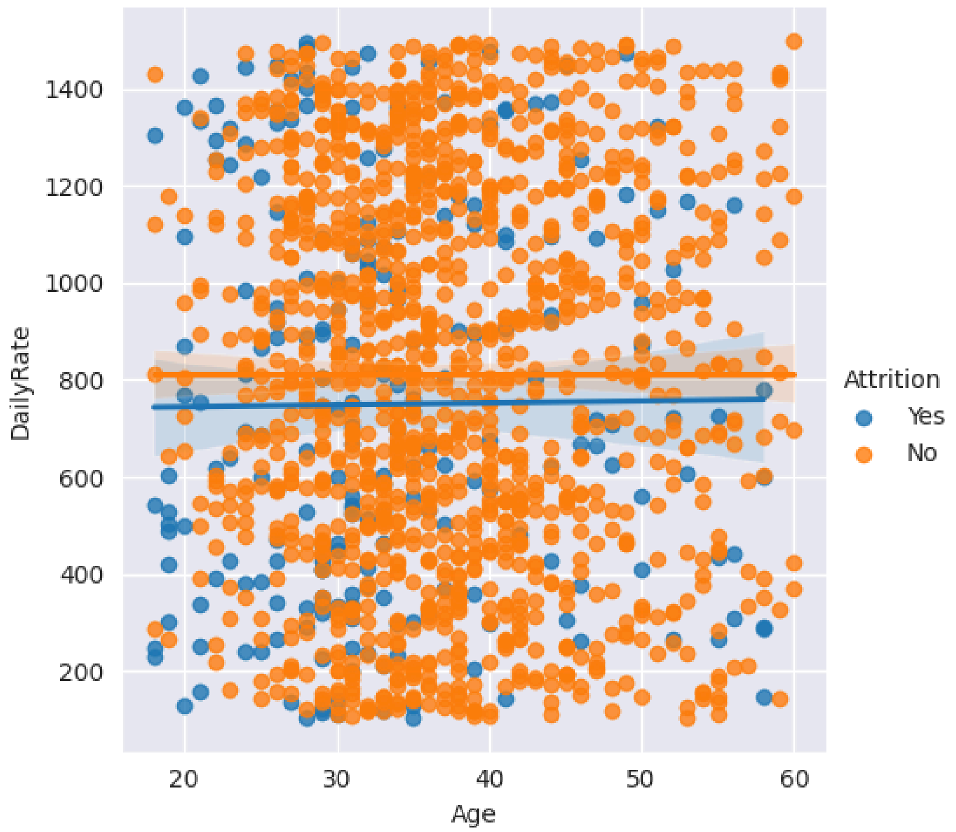
```
<Axes: xlabel='Attrition', ylabel='count'>
```



Yes No
Attrition

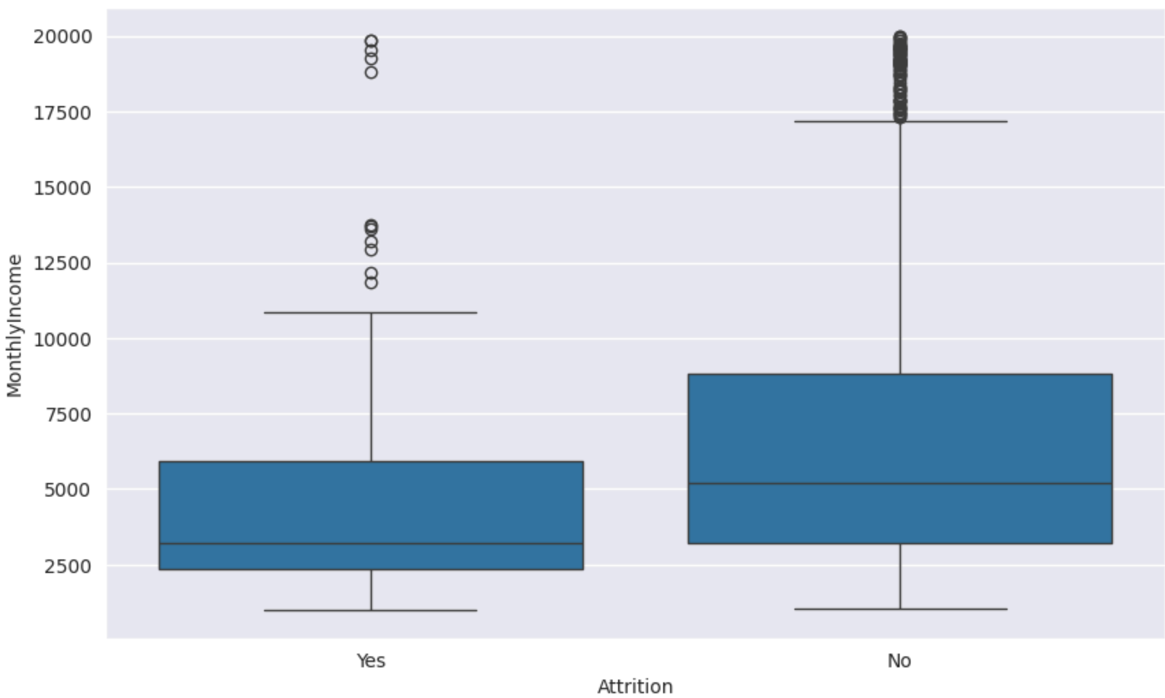
In [36]:
```
sns.lmplot(x = 'Age', y = 'DailyRate', hue = 'Attrition', data = df)
```

Out[36]: <seaborn.axisgrid.FacetGrid at 0x7ef6c7797400>



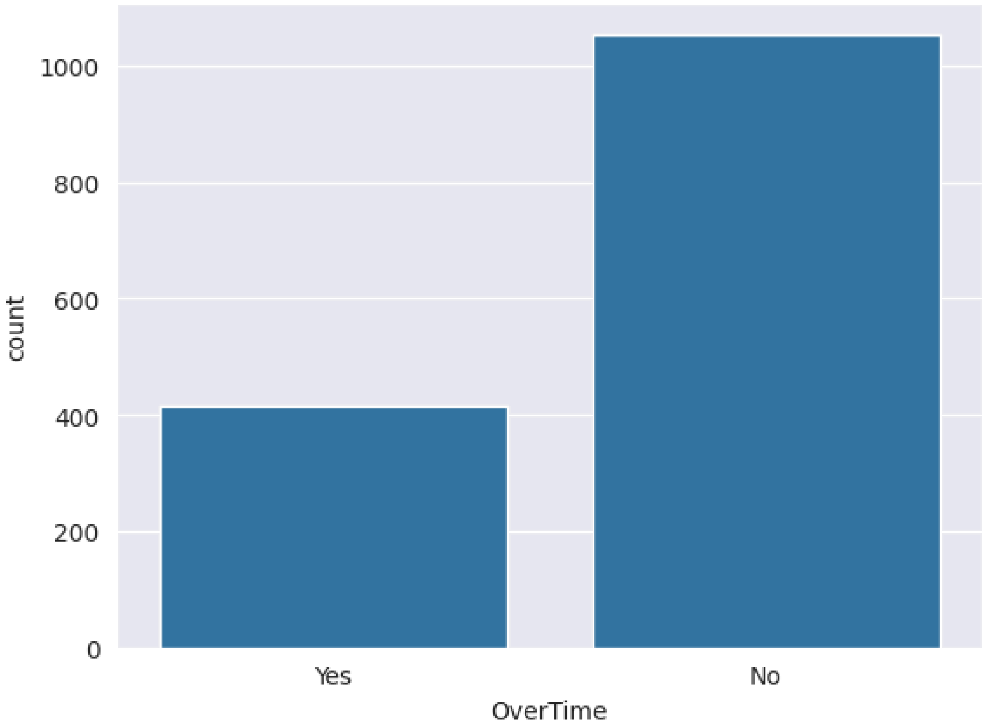
In [37]:
```
plt.figure(figsize =(10, 6))
sns.boxplot(y ='MonthlyIncome', x ='Attrition', data = df)
```

Out[37]: <Axes: xlabel='Attrition', ylabel='MonthlyIncome'>



In [38]:
```
sns.countplot(x ='OverTime', data = df)
```

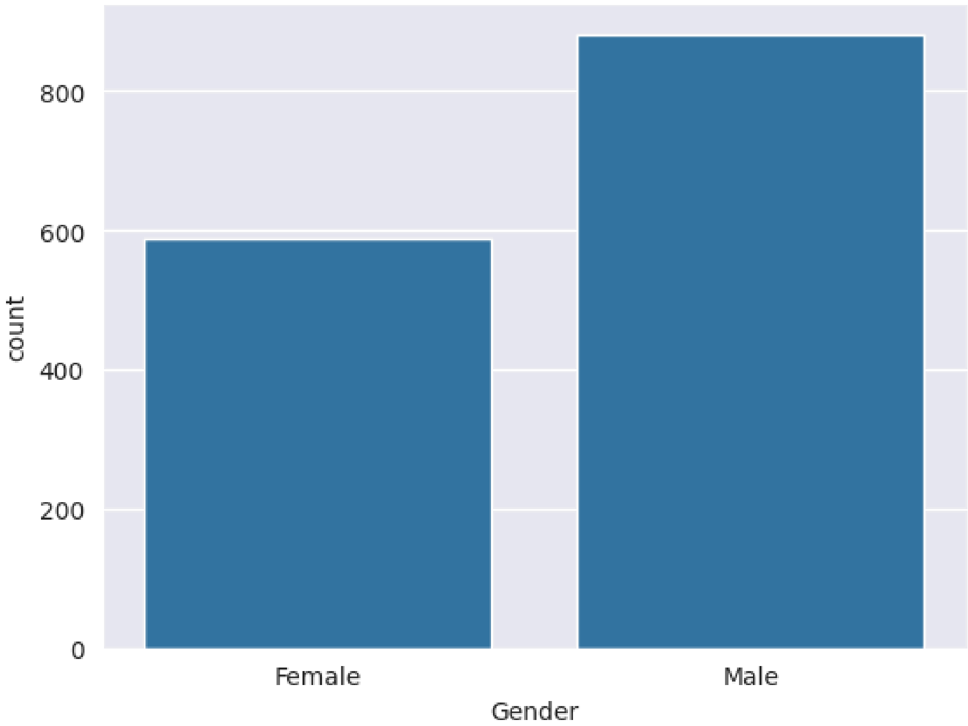
Out[38]: <Axes: xlabel='OverTime', ylabel='count'>



In [39]:

```
sns.countplot(x ='Gender', data = df)
```

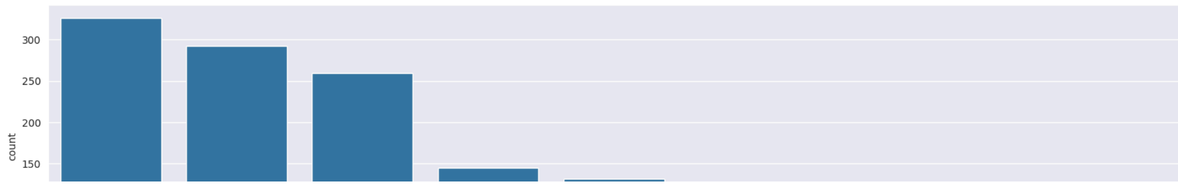
Out[39]: <Axes: xlabel='Gender', ylabel='count'>

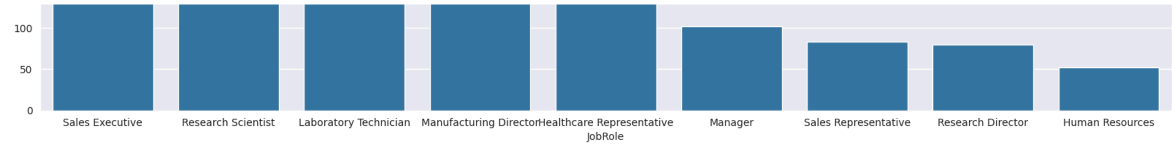


In [40]:

```
fig, ax = plt.subplots(figsize=(20, 5))
sns.countplot(x ='JobRole', data = df)
```

Out[40]: <Axes: xlabel='JobRole', ylabel='count'>

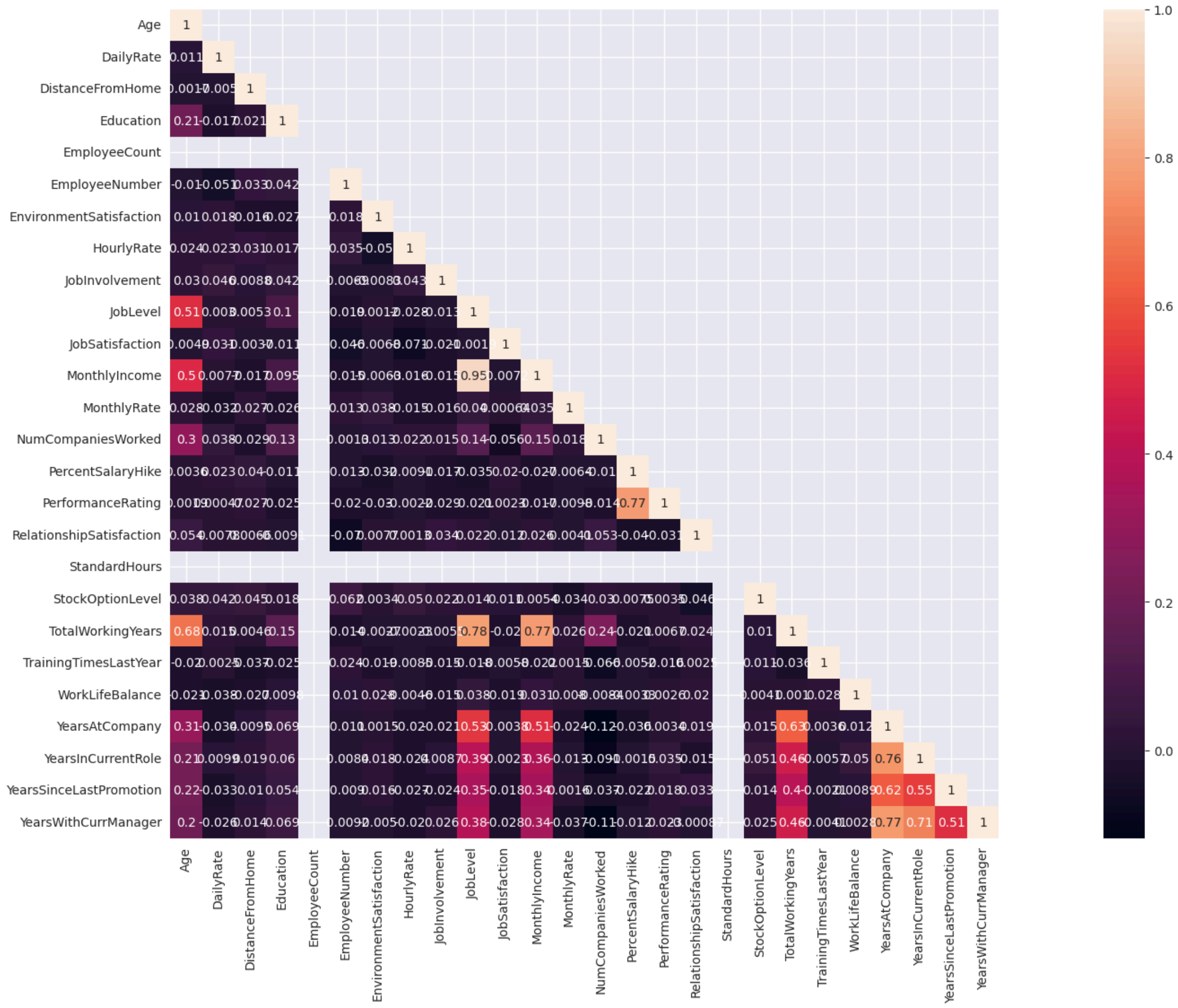




In [41]:

```
#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[41]: <Axes: >



In [42]:

```
# Feature Selection
df.drop(['BusinessTravel','DailyRate','EmployeeCount','EmployeeNumber','HourlyRate','Monthly
        ,'NumCompaniesWorked','Over18','StandardHours', 'StockOptionLevel','TrainingTimes
```

In [43]:

```
df.shape
```

Out[43]: (1470, 24)

Feature Encoding

In [44]:

```
#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,confusion_matrix,roc
from sklearn.model_selection import GridSearchCV

from imblearn.over_sampling import SMOTE

#preprocess.
from sklearn.preprocessing import MinMaxScaler,StandardScaler,LabelEncoder,OneHotEncoder
```

In [45]:

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

In [46]:

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

Out[46]:

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

In [47]:

```
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 [48]:

```
y = df.iloc[:, 1]
X = df
X.drop('Attrition', axis = 1, inplace = True)
```

In [49]:

```
X.head()
```

Out[49]:

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

5 rows × 23 columns

In [50]:

```
#Splitting data into Training and Test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=20)
```

In [51]:
```
X_train.shape, y_train.shape
```

Out[51]: ((1249, 23), (1249,))

In [52]:
```
### Crating a standard scaler object
scaler=StandardScaler()
scaler
```

Out[52]: StandardScaler()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [53]:
```
### using fit_transform to Standardize the train data
X_train=scaler.fit_transform(X_train)
X_train
```

Out[53]:
```
array([[ 1.41518706, -0.5031381 ,  0.9840021 , ...,  1.835334  ,
        -0.38987004,  1.92450052],
       [-0.21943252, -0.5031381 ,  1.97915888, ...,  1.56178601,
         2.34238001, -0.88651708],
       [-0.87328035,  1.40695153,  1.73036969, ...,  1.01469003,
         0.21729664,  1.64339876],
       ...,
       [ 1.08826314, -0.5031381 ,  2.47673727, ...,  0.74114204,
         2.03879667,  1.362297  ],
       [-1.41815354, -0.5031381 ,  1.73036969, ...,  0.19404605,
        -0.38987004, -0.0432118 ],
       [-1.09122963, -2.41322773,  1.60597509, ..., -1.1736939 ,
        -0.69345337, -1.16761884]])
```

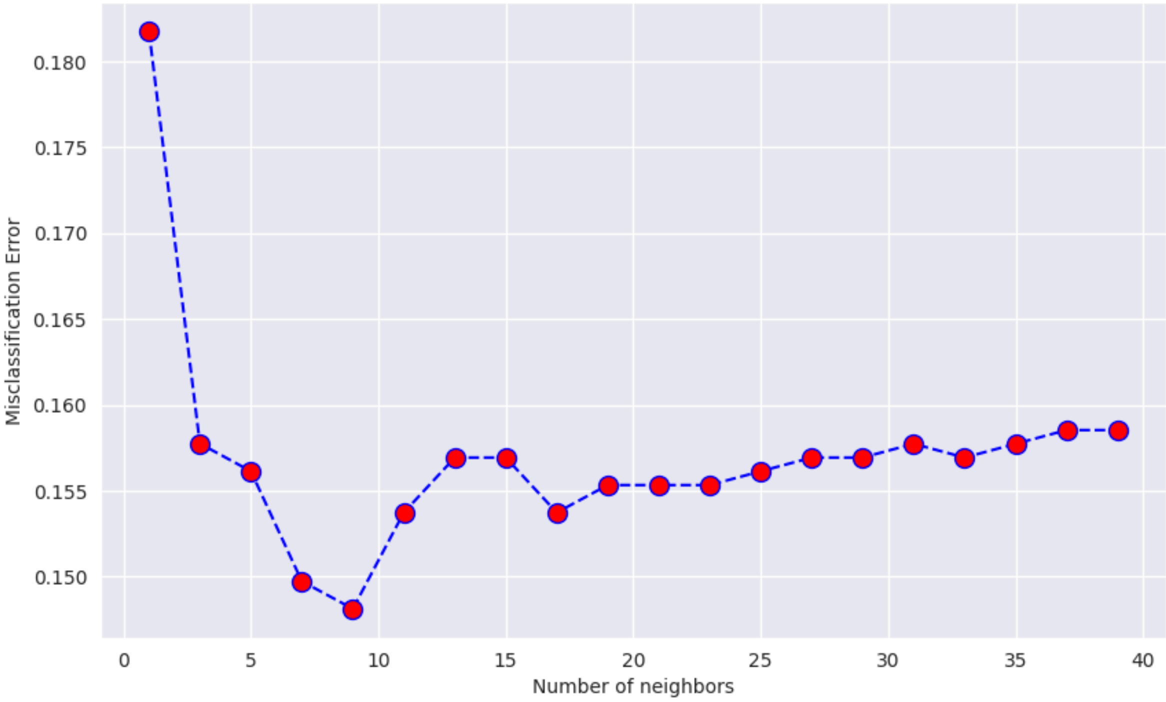
In [54]:
```
### here using transform only to avoid data leakage
### (training mean and training std will be used for standardisation when we use transform)
X_test=scaler.transform(X_test)
X_test
```

Out[54]:
```
array([[ 0.1074914 , -0.5031381 ,  1.1083967 , ..., -1.1736939 ,
        -0.38987004, -1.16761884],
       [-0.65533107, -0.5031381 , -0.01115468, ...,  1.01469003,
        -0.69345337,  1.08119524],
       [-0.4373818 , -0.5031381 , -0.50873307, ...,  0.46759404,
         1.73521333,  1.08119524],
       ...,
       [-0.54635643,  1.40695153, -0.50873307, ...,  1.01469003,
         0.82446332, -0.32431356],
       [ 0.65236459, -0.5031381 ,  2.35234268, ..., -0.62659792,
        -0.38987004, -0.60541532],
       [ 0.54338995, -0.5031381 , -0.88191686, ...,  0.46759404,
         0.52087998,  2.7678058 ]])
```

In [55]:
```
from sklearn.neighbors import KNeighborsClassifier
neighbors = []
cv_scores = []

from sklearn.model_selection import cross_val_score
# perform 10 fold cross validation
for k in range(1, 40, 2):
        neighbors.append(k)
        knn = KNeighborsClassifier(n_neighbors = k)
        scores = cross_val_score(
                knn, X_train, y_train, cv = 10, scoring = 'accuracy')
        cv_scores.append(scores.mean())
error_rate = [1-x for x in cv_scores]

# determining the best k
optimal_k = neighbors[error_rate.index(min(error_rate))]
print('The optimal number of neighbors is % d ' % optimal_k)
```

```
print('The optimal number of neighbors is %d' % optimal_k)

# plot misclassification error versus k
plt.figure(figsize = (10, 6))
plt.plot(range(1, 40, 2), error_rate, color ='blue', linestyle ='dashed', marker ='o',
                markerfacecolor ='red', markersize = 10)
plt.xlabel('Number of neighbors')
plt.ylabel('Misclassification Error')
plt.show()
```

```
The optimal number of neighbors is  9
```



In [56]:

```
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix

def print_score(clf, X_train, y_train, X_test, y_test, train = True):
        if train:
                print("Train Result:")
                print("------------")
                print("Classification Report: \n {}\n".format(classification_report(
                                y_train, clf.predict(X_train))))
                print("Confusion Matrix: \n {}\n".format(confusion_matrix(
                                y_train, clf.predict(X_train))))

                res = cross_val_score(clf, X_train, y_train,
                                                cv = 10, scoring ='accuracy')
                print("Average Accuracy: \t {0:.4f}".format(np.mean(res)))
                print("Accuracy SD: \t\t {0:.4f}".format(np.std(res)))
                print("accuracy score: {0:.4f}\n".format(accuracy_score(
                                y_train, clf.predict(X_train))))
                print("-----------------------------------------------------------")

        elif train == False:
                print("Test Result:")
                print("-----------")
                print("Classification Report: \n {}\n".format(
                                classification_report(y_test, clf.predict(X_test))))
                print("Confusion Matrix: \n {}\n".format(
                                confusion_matrix(y_test, clf.predict(X_test))))
                print("accuracy score: {0:.4f}\n".format(
                                accuracy_score(y_test, clf.predict(X_test))))
                print("-----------------------------------------------------------")

knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(X_train, y_train)
```

```
knn.fit(X_train, y_train)
print_score(knn, X_train, y_train, X_test, y_test, train = True)
print_score(knn, X_train, y_train, X_test, y_test, train = False)
```

```
Train Result:
------------
Classification Report:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1047
           1       0.90      0.22      0.35       202

    accuracy                           0.87      1249
   macro avg       0.88      0.61      0.64      1249
weighted avg       0.87      0.87      0.83      1249


Confusion Matrix:
 [[1042    5]
 [ 158   44]]

Average Accuracy:        0.8503
Accuracy SD:             0.0103
accuracy score: 0.8695

-----------------------------------------------------------
Test Result:
-----------
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.99      0.92       186
           1       0.60      0.09      0.15        35

    accuracy                           0.85       221
   macro avg       0.73      0.54      0.53       221
weighted avg       0.81      0.85      0.79       221


Confusion Matrix:
 [[184   2]
 [ 32   3]]

accuracy score: 0.8462

------------------------------------------------------------
```

In [57]:

```
# Validation scores of all base models
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error, r2_score,
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import KFold

models = []
models.append(('Log',LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state = 12345)))
models.append(('RF', RandomForestClassifier(random_state = 12345)))
models.append(('SVM', SVC(gamma='auto', random_state = 12345)))
models.append(('XGB', GradientBoostingClassifier(random_state = 12345)))
models.append(("LightGBM", LGBMClassifier(random_state = 12345)))

# evaluate each model in turn
results = []
names = []
```

In [58]:

```
for name, model in models:

        kfold = KFold(n_splits = 10, random_state = 12345, shuffle=True)
        cv_results = cross_val_score(model, X, y, cv = 10, scoring= "accuracy")
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

# boxplot algorithm comparison
fig = plt.figure(figsize=(15,10))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
Log: 0.844218 (0.013758)
KNN: 0.821769 (0.019432)
CART: 0.774830 (0.028287)
RF: 0.853741 (0.017007)
SVM: 0.838776 (0.003117)
XGB: 0.858503 (0.021252)
[LightGBM] [Info] Number of positive: 213, number of negative: 1110
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000433 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 531
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.160998 -> initscore=-1.650823
[LightGBM] [Info] Start training from score -1.650823
[LightGBM] [Info] Number of positive: 213, number of negative: 1110
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000546 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 533
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.160998 -> initscore=-1.650823
[LightGBM] [Info] Start training from score -1.650823
[LightGBM] [Info] Number of positive: 213, number of negative: 1110
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000219 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 532
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.160998 -> initscore=-1.650823
[LightGBM] [Info] Start training from score -1.650823
[LightGBM] [Info] Number of positive: 213, number of negative: 1110
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000264 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 532
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.160998 -> initscore=-1.650823
[LightGBM] [Info] Start training from score -1.650823
[LightGBM] [Info] Number of positive: 213, number of negative: 1110
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000259 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 531
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.160998 -> initscore=-1.650823
[LightGBM] [Info] Start training from score -1.650823
[LightGBM] [Info] Number of positive: 213, number of negative: 1110
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000241 seconds.
You can set `force_row_wise=true` to remove the overhead.
```

```
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 532
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.160998 -> initscore=-1.650823
[LightGBM] [Info] Start training from score -1.650823
[LightGBM] [Info] Number of positive: 213, number of negative: 1110
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000243
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 533
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.160998 -> initscore=-1.650823
[LightGBM] [Info] Start training from score -1.650823
[LightGBM] [Info] Number of positive: 214, number of negative: 1109
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000244
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 531
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.161754 -> initscore=-1.645238
[LightGBM] [Info] Start training from score -1.645238
[LightGBM] [Info] Number of positive: 214, number of negative: 1109
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000223
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 533
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.161754 -> initscore=-1.645238
[LightGBM] [Info] Start training from score -1.645238
[LightGBM] [Info] Number of positive: 214, number of negative: 1109
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000262
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 532
[LightGBM] [Info] Number of data points in the train set: 1323, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.161754 -> initscore=-1.645238
[LightGBM] [Info] Start training from score -1.645238
LightGBM: 0.863265 (0.013758)
```