Predicting Employee Attrition

In [1]:
```
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```
# Set working directory
import io
%cd "E:\Simplilearn\Electives\Hackathon\Predicting Employee Attrition"
```

```
E:\Simplilearn\Electives\Hackathon\Predicting Employee Attrition
```

In [3]:
```
# import the train and test data

employee_train = pd.read_csv('train_MpHjUjU.csv')
employee_test = pd.read_csv('test_hXY9mYw.csv')

# get number of rows and columns in the dataset

print(employee_train.shape)
print(employee_test.shape)
```

```
(19104, 13)
(741, 1)
```

In [4]:
```
employee_train.head() #first 5 records
```

Out[4]:

	MMM-YY	Emp_ID	Age	Gender	City	Education_Level	Salary	Dateofjoining	LastWorkingDate	Joining Designation	Designation	Total Business Value	Quarterly Rating
0	2016-01-01	1	28	Male	C23	Master	57387	2015-12-24	NaN	1	1	2381060	2
1	2016-02-01	1	28	Male	C23	Master	57387	2015-12-24	NaN	1	1	-665480	2
2	2016-03-01	1	28	Male	C23	Master	57387	2015-12-24	2016-03-11	1	1	0	2
3	2017-11-01	2	31	Male	C7	Master	67016	2017-11-06	NaN	2	2	0	1
4	2017-12-01	2	31	Male	C7	Master	67016	2017-11-06	NaN	2	2	0	1

In [5]:
```
employee_test.head() #first 5 records
```

Out[5]:

	Emp_ID
0	394
1	173
2	1090
3	840
4	308

In [6]:
```
employee_train.dtypes # check datatypes of the variables
```

Out[6]:
```
MMM-YY                 object
Emp_ID                  int64
Age                     int64
Gender                 object
City                   object
Education_Level        object
Salary                  int64
Dateofjoining          object
LastWorkingDate        object
Joining Designation     int64
Designation             int64
Total Business Value    int64
Quarterly Rating        int64
dtype: object
```

In [7]:
```
employee_train.info() # overview of data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   MMM-YY                 19104 non-null  object
 1   Emp_ID                 19104 non-null  int64
 2   Age                    19104 non-null  int64
 3   Gender                 19104 non-null  object
 4   City                   19104 non-null  object
 5   Education_Level        19104 non-null  object
 6   Salary                 19104 non-null  int64
 7   Dateofjoining          19104 non-null  object
 8   LastWorkingDate        1616 non-null   object
 9   Joining Designation    19104 non-null  int64
 10  Designation            19104 non-null  int64
 11  Total Business Value   19104 non-null  int64
 12  Quarterly Rating       19104 non-null  int64
dtypes: int64(7), object(6)
memory usage: 1.9+ MB
```

In [8]:
```
employee_train.describe() # basic statistics about data
```

Out[8]:

	Emp_ID	Age	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating
count	19104.000000	19104.000000	19104.000000	19104.000000	19104.000000	1.910400e+04	19104.000000
mean	1415.591133	34.650283	65652.025126	1.690536	2.252670	5.716621e+05	2.008899
std	810.705321	6.264471	30914.515344	0.836984	1.026512	1.128312e+06	1.009832
min	1.000000	21.000000	10747.000000	1.000000	1.000000	-6.000000e+06	1.000000
25%	710.000000	30.000000	42383.000000	1.000000	1.000000	0.000000e+00	1.000000
50%	1417.000000	34.000000	60087.000000	1.000000	2.000000	2.500000e+05	2.000000
75%	2137.000000	39.000000	83969.000000	2.000000	3.000000	6.997000e+05	3.000000
max	2788.000000	58.000000	188418.000000	5.000000	5.000000	3.374772e+07	4.000000

In [9]:
```
employee_train.isnull().sum() #checking for null values
```

Out[9]:
```
MMM-YY                      0
Emp_ID                      0
Age                         0
Gender                      0
City                        0
Education_Level             0
Salary                      0
Dateofjoining               0
LastWorkingDate         17488
Joining Designation         0
Designation                 0
Total Business Value        0
Quarterly Rating            0
dtype: int64
```

As we can see that there are 19401 rows and 13 columns with object datatype - 6 and rest 7 - integer datatype. Also there is no target column given in the data. So, we will use LastWorkingDate column and first we will fill null values by 0 and then we will create a new variable "Attrition" and use a for loop to fill this variable by 0 & 1 indicating employee left or not respectively. Finally we will concat this with our data. The steps are given below:

In [10]:
```
# Replacing null values in LastWorkingDate column
employee_train['LastWorkingDate'] = employee_train['LastWorkingDate'].fillna(0)
```

In [11]:
```
# Creating target variable
Attrition = []
for i in employee_train['LastWorkingDate']:
    if i == 0:
        Attrition.append(0)
    else:
        Attrition.append(1)
```

In [12]:
```
# converting target variable into a dataframe
Attrition = pd.DataFrame(Attrition)
```

In [13]:
```
# merge target variable with train data
employee_train = pd.concat([employee_train, Attrition], axis=1)
```

In [14]:
```
# Renaming the target variable as 'Attrition'
employee_train.rename(columns={0:'Attrition'},inplace=True)
```

In [15]:
```
employee_train.head(10) # first 10 records after performing above changes
```

Out[15]:

	MMM-YY	Emp_ID	Age	Gender	City	Education_Level	Salary	Dateofjoining	LastWorkingDate	Joining Designation	Designation	Total Business Value	Quarterly Rating	Attrition
0	2016-01-01	1	28	Male	C23	Master	57387	2015-12-24	0	1	1	2381060	2	0
1	2016-02-01	1	28	Male	C23	Master	57387	2015-12-24	0	1	1	-665480	2	0
2	2016-03-01	1	28	Male	C23	Master	57387	2015-12-24	2016-03-11	1	1	0	2	1
3	2017-11-01	2	31	Male	C7	Master	67016	2017-11-06	0	2	2	0	1	0
4	2017-12-01	2	31	Male	C7	Master	67016	2017-11-06	0	2	2	0	1	0
5	2016-12-01	4	43	Male	C13	Master	65603	2016-12-07	0	2	2	0	1	0
6	2017-01-01	4	43	Male	C13	Master	65603	2016-12-07	0	2	2	0	1	0
7	2017-02-01	4	43	Male	C13	Master	65603	2016-12-07	0	2	2	0	1	0
8	2017-03-01	4	43	Male	C13	Master	65603	2016-12-07	0	2	2	350000	1	0
9	2017-04-01	4	43	Male	C13	Master	65603	2016-12-07	2017-04-27	2	2	0	1	1

Now it will be very good if we have a column which shows the how experience every employee has in years. For this first we need to convert the datatype of Dateofjoining column into a datetime and then we will replace the '0' in LastWorkingDate column by today's date '2021-11-20' and also convert it to datetime datatype. After that we will create a column with name 'Experience' which will be result of substraction between LastWorkingDate and Dateofjoining. Now you will see that the output will be in days and datatype will be timedelta. So to convert it into years we will first convert it into integer using pd.to_numeric and then divide the values in the column by 365 then rounding it upto two places to get Experience in years. The steps are given below:

In [16]:
```
# Dateofjoining to datetime

employee_train['Dateofjoining'] = pd.to_datetime(employee_train['Dateofjoining'])
```

In [17]:
```
# replacing '0' in LastWorkingDate column by todays date - '2021-11-20'

employee_train['LastWorkingDate'] = employee_train['LastWorkingDate'].replace(0,'2021-11-20')
```

In [18]:
```
# LastWorkingDate to datetime

employee_train['LastWorkingDate'] = pd.to_datetime(employee_train['LastWorkingDate'])
```

In [19]:
```
# creating 'Experience' column

employee_train['Experience'] = employee_train['LastWorkingDate'] - employee_train['Dateofjoining']
```

In [20]:
```
# converting its datatype to integer

employee_train['Experience'] = pd.to_numeric(employee_train['Experience'].dt.days, downcast='integer')
```

In [21]:
```
# converting the values to years

employee_train['Experience'] = round(employee_train['Experience']/365,2)
```

In [22]:
```
# Now we dont need 'Dateofjoining' and 'LastWorkingDate' column so we will drop it

employee_train = employee_train.drop(['Dateofjoining', 'LastWorkingDate'],axis=1)
```

In [23]:
```
employee_train.head()
```

Out[23]:

	MMM-YY	Emp_ID	Age	Gender	City	Education_Level	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating	Attrition	Experience
0	2016-01-01	1	28	Male	C23	Master	57387	1	1	2381060	2	0	5.91
1	2016-02-01	1	28	Male	C23	Master	57387	1	1	-665480	2	0	5.91
2	2016-03-01	1	28	Male	C23	Master	57387	1	1	0	2	1	0.21
3	2017-11-01	2	31	Male	C7	Master	67016	2	2	0	1	0	4.04
4	2017-12-01	2	31	Male	C7	Master	67016	2	2	0	1	0	4.04

Now we can see that there are duplicate EMP_ID in our data so we will use drop_duplicates and keep the last of the records for further analysis.

In [24]:
```
employee = employee_train.drop_duplicates(subset='Emp_ID', keep='last')
```

In [25]:
```
employee.reset_index(inplace=True, drop=True) # resetting the index
```

In [26]:
```
employee.head()
```

Out[26]:

	MMM-YY	Emp_ID	Age	Gender	City	Education_Level	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating	Attrition	Experience
0	2016-03-01	1	28	Male	C23	Master	57387	1	1	0	2	1	0.21
1	2017-12-01	2	31	Male	C7	Master	67016	2	2	0	1	0	4.04
2	2017-04-01	4	43	Male	C13	Master	65603	2	2	0	1	1	0.39
3	2016-03-01	5	29	Male	C9	College	46368	1	1	0	1	1	0.16
4	2017-12-01	6	31	Female	C11	Bachelor	78728	3	3	0	2	0	4.31

In [27]:
```
employee.shape
```

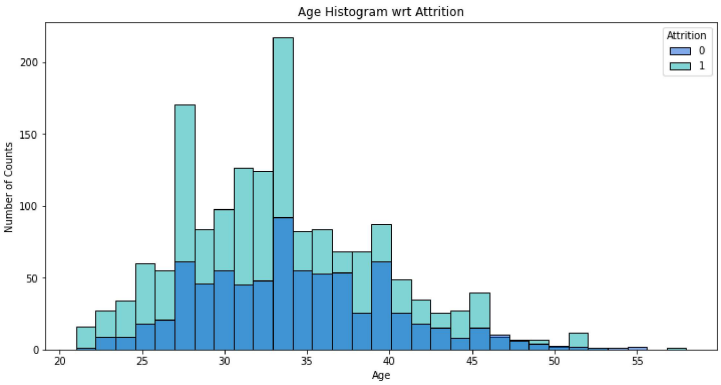
Out[27]:
```
(2381, 13)
```

Exploratory Data Analysis (EDA)

Now we will perform some EDA on our data using some basic visualization techniques.

In [28]:
```
# Lets create a histogram of Age column and derive insights from it

plt.figure(figsize=(12,6))
sns.histplot(data=employee, x='Age', hue='Attrition', palette='winter')
plt.xlabel('Age')
plt.ylabel('Number of Counts')
plt.title('Age Histogram wrt Attrition')
plt.show()
```



From above analysis we can see that persons with Age group between 28 - 35 has more Attrition.

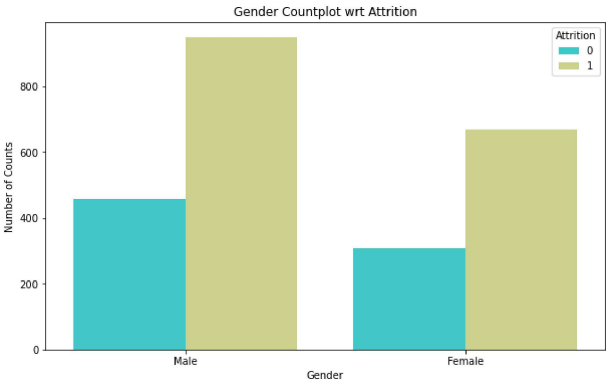
In [29]:
```
# Lets create a countplot of Gender column and derive insights from it

print(employee['Gender'].value_counts())

# Creating countplot

plt.figure(figsize=(10,6))
sns.countplot(data=employee, x='Gender', hue='Attrition', palette='rainbow')
plt.xlabel('Gender')
plt.ylabel('Number of Counts')
plt.title('Gender Countplot wrt Attrition')
plt.show()
```

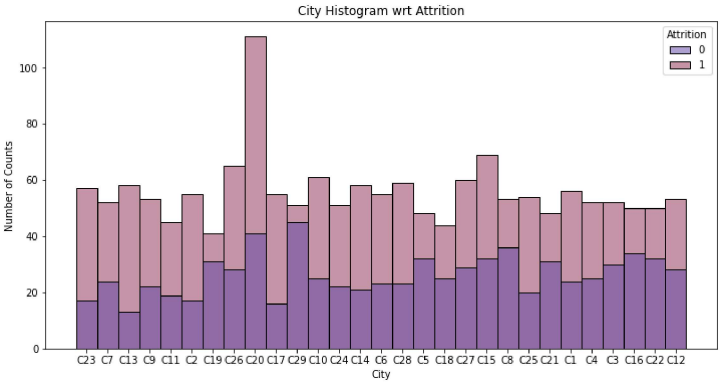
```
Male      1404
Female     977
Name: Gender, dtype: int64
```



From above analysis we can see that the percentage of Male and Female employees leaving is nearly equal accoring to their individual total counts.

In [30]:
```
# Lets create a histogram of City column and derive insights from it

plt.figure(figsize=(12,6))
sns.histplot(data=employee, x='City', hue='Attrition', palette='twilight')
plt.xlabel('City')
plt.ylabel('Number of Counts')
plt.title('City Histogram wrt Attrition')
plt.show()
```



From above analysis we can see that person from city 'C20' has maximum Attrition

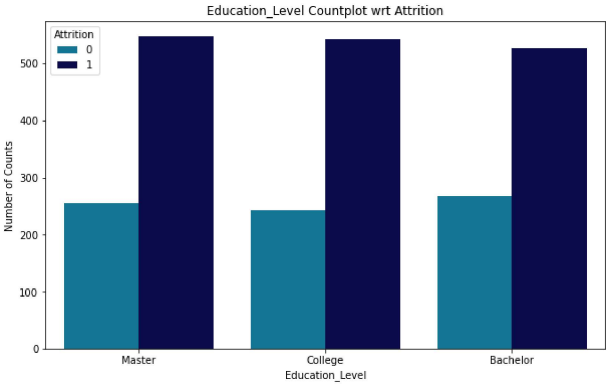
In [31]:
```
# Lets create a countplot of Education_Level column and derive insights from it

print(employee['Education_Level'].value_counts())

# Creating countplot

plt.figure(figsize=(10,6))
sns.countplot(data=employee, x='Education_Level', hue='Attrition', palette='ocean_r')
plt.xlabel('Education_Level')
plt.ylabel('Number of Counts')
plt.title('Education_Level Countplot wrt Attrition')
plt.show()
```

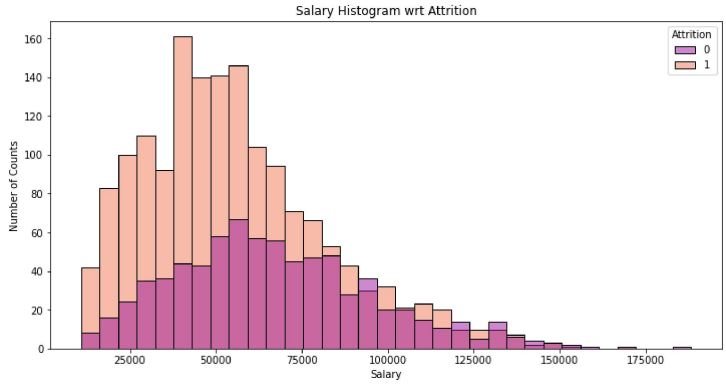
```
Master      802
Bachelor    795
College     784
Name: Education_Level, dtype: int64
```



From above analysis we can see that the count of people leaving corresponding Education_Level is nearly the same

In [32]:
```
# Lets create a histogram of Salary column and derive insights from it

plt.figure(figsize=(12,6))
sns.histplot(data=employee, x='Salary', hue='Attrition', palette='plasma')
plt.xlabel('Salary')
plt.ylabel('Number of Counts')
plt.title('Salary Histogram wrt Attrition')
plt.show()
```



From above analysis we can say that the persons with a salary ranging between 45000 - 60000 has more Attrition

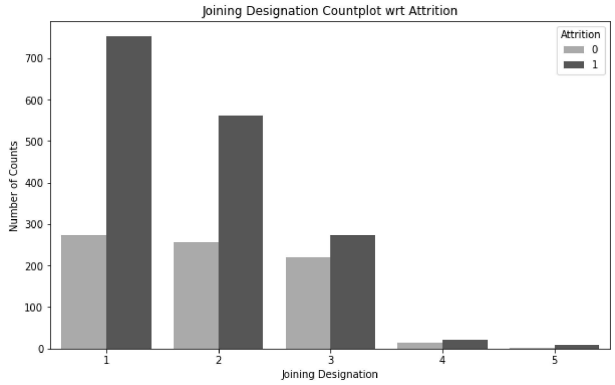
In [33]:
```
# Lets create a countplot of Joining Designation column and derive insights from it

print(employee['Joining Designation'].value_counts())

# Creating countplot

plt.figure(figsize=(10,6))
sns.countplot(data=employee, x='Joining Designation', hue='Attrition', palette='binary')
plt.xlabel('Joining Designation')
plt.ylabel('Number of Counts')
plt.title('Joining Designation Countplot wrt Attrition')
plt.show()
```

```
1    1026
2     815
3     493
4      36
5      11
Name: Joining Designation, dtype: int64
```



So by above analysis we can see that Joining Designation 1 - 3 has more attrition

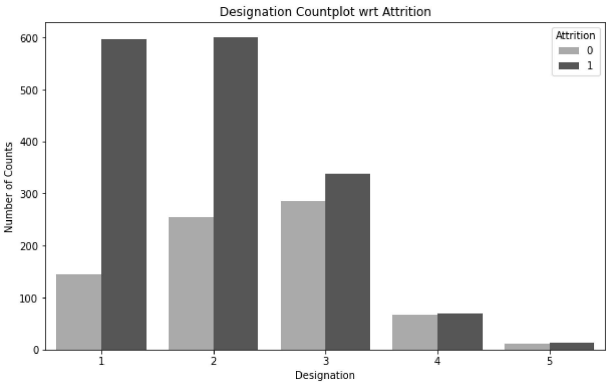
In [34]:
```
# Lets create a countplot of Designation column and derive insights from it

print(employee['Designation'].value_counts())

# Creating countplot

plt.figure(figsize=(10,6))
sns.countplot(data=employee, x='Designation', hue='Attrition', palette='binary')
plt.xlabel('Designation')
plt.ylabel('Number of Counts')
plt.title('Designation Countplot wrt Attrition')
plt.show()
```

```
2    855
1    741
3    623
4    138
5     24
Name: Designation, dtype: int64
```



So by above analysis we can see that Designation 1 - 3 has more attrition

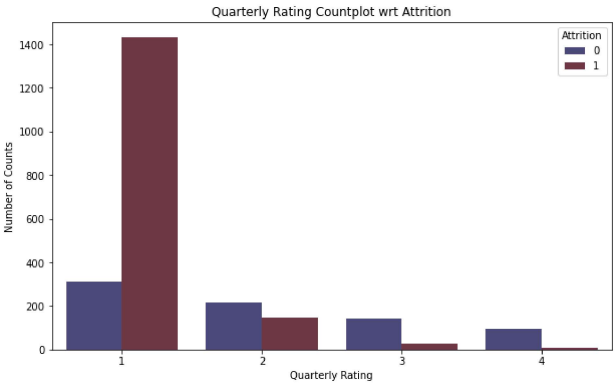
In [35]:
```
# Lets create a countplot of Quarterly Rating column and derive insights from it

print(employee['Quarterly Rating'].value_counts())

# Creating countplot

plt.figure(figsize=(10,6))
sns.countplot(data=employee, x='Quarterly Rating', hue='Attrition', palette='icefire')
plt.xlabel('Quarterly Rating')
plt.ylabel('Number of Counts')
plt.title('Quarterly Rating Countplot wrt Attrition')
plt.show()
```

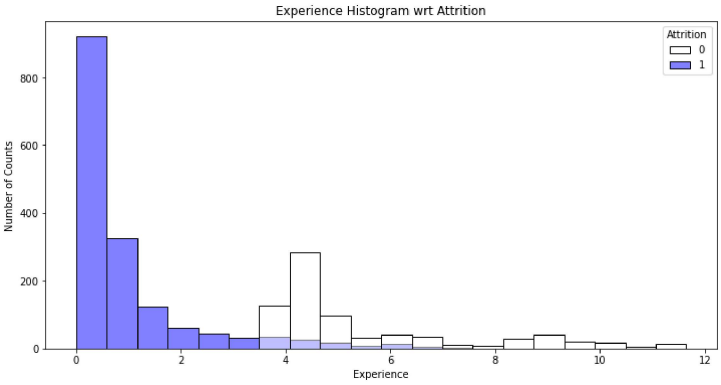
```
1    1744
2     362
3     168
4     107
Name: Quarterly Rating, dtype: int64
```



So by above analysis the Quarterly Rating 1 has maximum attrition

In [36]:
```
# Lets create a histogram of Experience column and derive insights from it

plt.figure(figsize=(12,6))
sns.histplot(data=employee, x='Experience', hue='Attrition', palette='flag')
plt.xlabel('Experience')
plt.ylabel('Number of Counts')
plt.title('Experience Histogram wrt Attrition')
plt.show()
```



From above analysis we can say that mostly employees having experience in the range 0 - 4 years leaves the organisation in which maximum count is for the experience range 0 - 3.5.

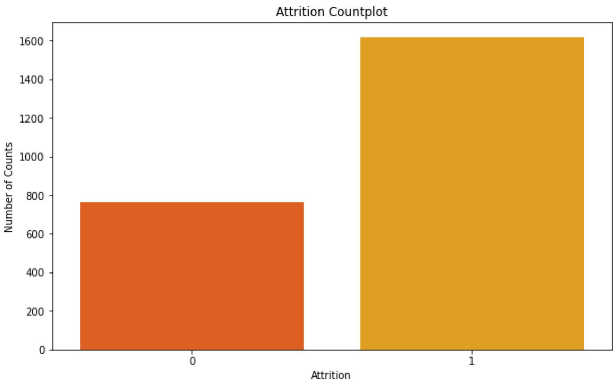
In [37]:
```
# Lets create a countplot of Attrition column

print(employee['Attrition'].value_counts())

# Creating countplot

plt.figure(figsize=(10,6))
sns.countplot(data=employee, x='Attrition', palette='autumn')
plt.xlabel('Attrition')
plt.ylabel('Number of Counts')
plt.title('Attrition Countplot')
plt.show()
```

```
1    1616
0     765
Name: Attrition, dtype: int64
```

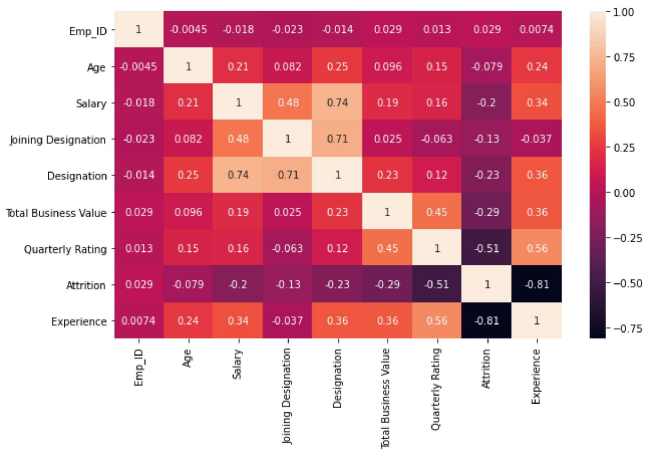


So we can see that our data is imbalanced as there are twice the count of Attrition as '1' compared to Attrition '0'

Now lets create a heatmap to see if there is any multicollinearity exists within the variables

In [38]:
```
# Heatmap
plt.figure(figsize=(10,6))
sns.heatmap(data=employee.corr(), annot=True)
```

Out[38]: <AxesSubplot:>



Here we see that the our target variable 'Attrition' has negative correlation with other feature variables

Now we will make 2 copy of employee one X_train and X_test and as we are given with the test data with only Emp_ID we will divide our data accordingly using below steps

In [39]:
```
X_train = employee.copy(deep=True)
```

In [40]:
```
X_train.head()
```

Out[40]:

	MMM-YY	Emp_ID	Age	Gender	City	Education_Level	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating	Attrition	Experience
0	2016-03-01	1	28	Male	C23	Master	57387	1	1	0	2	1	0.21
1	2017-12-01	2	31	Male	C7	Master	67016	2	2	0	1	0	4.04
2	2017-04-01	4	43	Male	C13	Master	65603	2	2	0	1	1	0.39
3	2016-03-01	5	29	Male	C9	College	46368	1	1	0	1	1	0.16
4	2017-12-01	6	31	Female	C11	Bachelor	78728	3	3	0	2	0	4.31

In [41]:
```
X_train.shape
```

Out[41]:
```
(2381, 13)
```

In [42]:
```
for i in range(0,741):
    X_train.drop(X_train.index[X_train['Emp_ID']==employee_test['Emp_ID'][i]], inplace=True)
```

In [43]:
```
X_train.shape
```

Out[43]:
```
(1640, 13)
```

In [44]:
```
X_test = employee.copy(deep=True)
```

In [45]:
```
X_test.shape
```

Out[45]:
```
(2381, 13)
```

In [46]:
```
X_test.head()
```

Out[46]:

	MMM-YY	Emp_ID	Age	Gender	City	Education_Level	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating	Attrition	Experience
0	2016-03-01	1	28	Male	C23	Master	57387	1	1	0	2	1	0.21
1	2017-12-01	2	31	Male	C7	Master	67016	2	2	0	1	0	4.04
2	2017-04-01	4	43	Male	C13	Master	65603	2	2	0	1	1	0.39
3	2016-03-01	5	29	Male	C9	College	46368	1	1	0	1	1	0.16
4	2017-12-01	6	31	Female	C11	Bachelor	78728	3	3	0	2	0	4.31

In [47]:
```
ID = []
for i in employee_test['Emp_ID']:
    ID.append(i)

len(ID)
```

Out[47]:
```
741
```

In [48]:
```
X_test.loc[X_test['Emp_ID'].isin(ID)]
```

Out[48]:

	MMM-YY	Emp_ID	Age	Gender	City	Education_Level	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating	Attrition	Experience
1	2017-12-01	2	31	Male	C7	Master	67016	2	2	0	1	0	4.04
4	2017-12-01	6	31	Female	C11	Bachelor	78728	3	3	0	2	0	4.31
6	2017-12-01	11	28	Female	C19	Master	42172	1	1	0	1	0	3.96
9	2017-12-01	14	39	Female	C26	College	19734	3	3	0	1	0	4.10
17	2017-12-01	25	31	Male	C24	Bachelor	102077	1	3	2013180	4	0	7.06
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2370	2017-12-01	2775	27	Male	C9	College	85112	3	3	0	1	0	4.14
2372	2017-12-01	2778	35	Male	C13	Master	50180	2	2	0	1	0	3.98
2374	2017-12-01	2781	25	Male	C23	Master	46952	2	2	2366500	4	0	4.76
2376	2017-12-01	2784	34	Male	C24	College	82815	2	3	505480	4	0	9.10
2380	2017-12-01	2788	30	Male	C27	Master	70254	2	2	411480	2	0	4.45

741 rows × 13 columns

In [49]:
```
X_test = X_test.set_index('Emp_ID')
X_test = X_test.reindex(index=employee_test['Emp_ID'])
X_test = X_test.reset_index()
```

In [50]:
```
X_test.head()
```

Out[50]:

	Emp_ID	MMM-YY	Age	Gender	City	Education_Level	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating	Attrition	Experience
0	394	2017-12-01	34	Female	C20	Master	97722	2	4	2701750	3	0	10.47
1	173	2017-12-01	39	Male	C28	College	56174	1	3	706010	3	0	10.45
2	1090	2017-12-01	39	Male	C13	College	96750	2	4	1518520	2	0	10.41
3	840	2017-12-01	40	Female	C8	College	88813	1	4	1151820	2	0	10.37
4	308	2017-12-01	32	Male	C5	Master	188418	2	5	3772910	2	0	10.27

Now we will create y_train and y_test from X_train and X_test respectively. Also we will remove columns which are not necessary.

In [51]:
```
y_train = X_train['Attrition']
y_test = X_test['Attrition']
```

In [52]:
```
X_train = X_train.drop(['MMM-YY','Attrition','Emp_ID'],axis=1)
X_test = X_test.drop(['MMM-YY','Attrition','Emp_ID'],axis=1)
```

In [53]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1640, 10)
(741, 10)
(1640,)
(741,)
```

Now we will first treat the column with object datatype using Label Encoder and convert them to numeric

In [54]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [55]:
```
X_train['Gender'] = le.fit_transform(X_train['Gender'])
X_train['City'] = le.fit_transform(X_train['City'])
X_train['Education_Level'] = le.fit_transform(X_train['Education_Level'])
X_test['Gender'] = le.fit_transform(X_test['Gender'])
X_test['City'] = le.fit_transform(X_test['City'])
X_test['Education_Level'] = le.fit_transform(X_test['Education_Level'])
```

In [56]:
```
X_train.head()
```

Out[56]:

	Age	Gender	City	Education_Level	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating	Experience
0	28	1	15	2	57387	1	1	0	2	0.21
2	43	1	4	2	65603	2	2	0	1	0.39
3	29	1	28	1	46368	1	1	0	1	0.16
5	34	1	11	1	70656	3	3	0	1	0.16
7	35	1	15	2	28116	1	1	0	1	0.48

In [57]:
```
X_test.head()
```

Out[57]:

	Age	Gender	City	Education_Level	Salary	Joining Designation	Designation	Total Business Value	Quarterly Rating	Experience
0	34	0	12	2	97722	2	4	2701750	3	10.47
1	39	1	20	1	56174	1	3	706010	3	10.45
2	39	1	4	1	96750	2	4	1518520	2	10.41
3	40	0	27	1	88813	1	4	1151820	2	10.37
4	32	1	24	2	188418	2	5	3772910	2	10.27

In [58]:
```
# We will scale the data using StandardScalar so that we can improve our ML model performance
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Logistic Regression

In [126]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=1000, penalty='elasticnet', solver='saga',l1_ratio=1)
```

In [127]:
```
lr.fit(X_train,y_train)
```

Out[127]:
```
LogisticRegression(l1_ratio=1, max_iter=1000, penalty='elasticnet',
                   solver='saga')
```

In [128]:
```
lr.score(X_train,y_train)
```

Out[128]:
```
0.9963414634146341
```

In [129]:
```
lr_pred = lr.predict(X_test)
```

In [130]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

In [131]:
```
confusion_matrix(y_test, lr_pred)
```

Out[131]:
```
array([[ 99, 642],
       [  0,   0]], dtype=int64)
```

In [132]:
```
accuracy_score(y_test, lr_pred)
```

Out[132]:
```
0.13360323886639677
```

In [133]:
```
print(classification_report(y_test, lr_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.13      0.24       741
           1       0.00      0.00      0.00         0

    accuracy                           0.13       741
   macro avg       0.50      0.07      0.12       741
weighted avg       1.00      0.13      0.24       741

C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Here we can say that Logistic regression is over-fitting and as we can say that it could not classify the class '1' as precision and recall is 0 for the class '1'.

Random Forest Classifier

First we will find out the best parameter for Random Forest Classifier. the steps are given below:

In [134]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
```

In [135]:
```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, N
one], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

In [136]:
```
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()
```

In [137]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
RF = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
RF_random = RandomizedSearchCV(estimator = RF, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2,
                               random_state=42, n_jobs = -1)
# Fit the random search model
RF_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[137]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

In [138]:
```
RF_random.best_params_
```

Out[138]:
```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 50,
 'bootstrap': False}
```

In [139]:
```
# Lets use this parametrs for our model

RF = RandomForestClassifier(n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features='auto',
                           max_depth=50, bootstrap=False)
```

In [140]:
```
# fit the train data
RF.fit(X_train, y_train)
```

Out[140]:
```
RandomForestClassifier(bootstrap=False, max_depth=50, n_estimators=1000)
```

In [141]:
```
RF.score(X_train, y_train)
```

Out[141]:
```
1.0
```

In [142]:
```
RF_pred = RF.predict(X_test)
```

In [143]:
```
confusion_matrix(y_test, RF_pred)
```

Out[143]:
```
array([[ 96, 645],
       [  0,   0]], dtype=int64)
```

In [144]:
```
accuracy_score(y_test, RF_pred)
```

Out[144]:
```
0.12955465587044535
```

In [145]:
```
print(classification_report(y_test, RF_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.13      0.23       741
           1       0.00      0.00      0.00         0

    accuracy                           0.13       741
   macro avg       0.50      0.06      0.11       741
weighted avg       1.00      0.13      0.23       741

C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Here we can see that even after hyper parameter tuning still the model is over-fitting and it could not classify the class '1' as precision and recall is 0 for the class '1'.

Naive Bayes Classifier

In [156]:
```
# Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
```

In [157]:
```
# fit the train data
NB.fit(X_train, y_train)
```

Out[157]:
```
GaussianNB()
```

In [158]:
```
NB.score(X_train, y_train)
```

Out[158]:
```
0.9689024390243902
```

In [159]:
```
NB_pred = NB.predict(X_test)
```

In [160]:
```
confusion_matrix(y_test, NB_pred)
```

Out[160]:
```
array([[163, 578],
       [  0,   0]], dtype=int64)
```

In [161]:
```
accuracy_score(y_test, NB_pred)
```

Out[161]:
```
0.21997300944669365
```

In [162]:
```
print(classification_report(y_test, NB_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.22      0.36       741
           1       0.00      0.00      0.00         0

    accuracy                           0.22       741
   macro avg       0.50      0.11      0.18       741
weighted avg       1.00      0.22      0.36       741


C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Again we see that the this model is not also able to classify the class '1' as precision and recall is 0 for the class '1'.

K-Nearest Neighbor Classifier

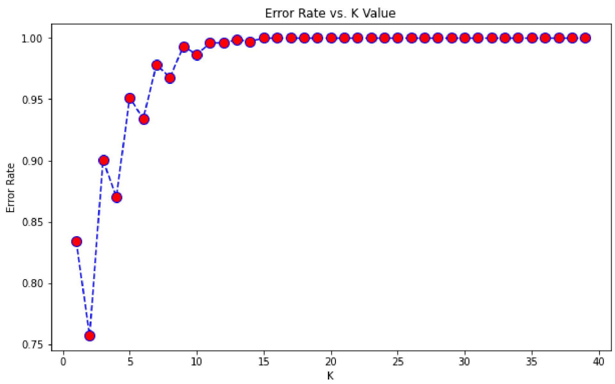
In [163]:
```
# K-Nearest Neighbor Classifier

# First we will find an optimum value k for n_neighbours

from sklearn.neighbors import KNeighborsClassifier
error_rate = []
for i in range(1,40):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[163]:
```
Text(0, 0.5, 'Error Rate')
```



In [169]:
```
KNN = KNeighborsClassifier(n_neighbors = 2, metric='minkowski', p=2)
```

In [170]:
```
# fit the train data
KNN.fit(X_train, y_train)
```

Out[170]:
```
KNeighborsClassifier(n_neighbors=2)
```

In [171]:
```
KNN.score(X_train, y_train)
```

Out[171]:
```
0.9975609756097561
```

In [172]:
```
KNN_pred = KNN.predict(X_test)
```

In [173]:
```
confusion_matrix(y_test, KNN_pred)
```

Out[173]:
```
array([[180, 561],
       [  0,   0]], dtype=int64)
```

In [174]:
```
accuracy_score(y_test, KNN_pred)
```

Out[174]:
```
0.242914979757085
```

In [175]:
```
print(classification_report(y_test, KNN_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.24      0.39       741
           1       0.00      0.00      0.00         0

    accuracy                           0.24       741
   macro avg       0.50      0.12      0.20       741
weighted avg       1.00      0.24      0.39       741

C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Again we see that the this model is not also able to classify the class '1' as precision and recall is 0 for the class '1'.

XGBoost Classifier

In [183]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier()
```

In [184]:
```
xgb.fit(X_train, y_train)
```

```
[22:36:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with t
he objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

C:\Users\LENOVO\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future
release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integer
s starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

Out[184]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=4,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [185]:
```
xgb.score(X_train, y_train)
```

Out[185]:
```
1.0
```

In [186]:
```
xgb_pred = xgb.predict(X_test)
```

In [187]:
```
confusion_matrix(y_test, xgb_pred)
```

Out[187]:
```
array([[184, 557],
       [  0,   0]], dtype=int64)
```

In [188]:
```
accuracy_score(y_test, xgb_pred)
```

Out[188]:
```
0.2483130904183536
```

In [189]:
```
print(classification_report(y_test, xgb_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.25      0.40       741
           1       0.00      0.00      0.00         0

    accuracy                           0.25       741
   macro avg       0.50      0.12      0.20       741
weighted avg       1.00      0.25      0.40       741

C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Again we see that the this model is not also able to classify the class '1' as precision and recall is 0 for the class '1'.

Lets choose XGB predictions for our our submission to see the results

In [190]:
```
pd.DataFrame(xgb_pred).to_csv('XGB_DATA.csv')
```