In [1]:
```
import os
import warnings
from datetime import datetime
from collections import defaultdict

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from imblearn.over_sampling import SMOTE

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
from sklearn.exceptions import ConvergenceWarning

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFECV
```

In [2]:
```
sns.set()
pd.set_option('display.max_columns', None)
```

1. The HR Analytics Dataset

This dataset came from Kaggle: https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset (https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset)

Load the dataset from a csv file.

In [3]:
```
# Load the csv dataset into pandas dataframes
data_file = os.path.join(os.getcwd(), 'dataset', 'hr_employee_attrition.csv')
hr_df = pd.read_csv(data_file)
hr_df.head(3)
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Emp
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	

In [4]:
```
hr_df.describe(include='all')
```

Out[4]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Edu
count	1470.000000	1470	1470	1470.000000	1470	1470.000000	1470.000000	
unique	NaN	2	3	NaN	3	NaN	NaN	
top	NaN	No	Travel_Rarely	NaN	Research & Development	NaN	NaN	L
freq	NaN	1233	1043	NaN	961	NaN	NaN	
mean	36.923810	NaN	NaN	802.485714	NaN	9.192517	2.912925	
std	9.135373	NaN	NaN	403.509100	NaN	8.106864	1.024165	
min	18.000000	NaN	NaN	102.000000	NaN	1.000000	1.000000	
25%	30.000000	NaN	NaN	465.000000	NaN	2.000000	2.000000	
50%	36.000000	NaN	NaN	802.000000	NaN	7.000000	3.000000	
75%	43.000000	NaN	NaN	1157.000000	NaN	14.000000	4.000000	
max	60.000000	NaN	NaN	1499.000000	NaN	29.000000	5.000000	

All rows have values for each column. Therefore, there's no need to drop any rows.

In [5]:
```
hr_df.columns[hr_df.isna().any()].tolist()
```

Out[5]: []

Upon inspection, column `Over18` has values that are all 'Y'. A good assumption would be that a person has to be 18 years old at the minimum to be employed. Initially, this can be considered uselss. However, I retained this for the following reasons:

1. Country - There are some countries where the legal minimum working age is lower than 18 (e.g. 15 for Japan). This dataset did not indicate which countries the employees are from.
2. Live Production Data - This dataset is a snippet in its current form. However, if production data is always extracted, there will always be a possibility that the opposite value, 'N', would appear.
3. Interns - While not present in the dataset, interns can be below 18 years old. In addition, every employer keeps records of their interns as they're still considered employees.

In [6]:
```
hr_df['Over18'].value_counts()
```

Out[6]:
```
Y    1470
Name: Over18, dtype: int64
```

Column `EmployeeCount` contains a single value as well. However, this column does not make sense at all so I removed this one.

In [7]:
```
hr_df['EmployeeCount'].value_counts()
```

Out[7]:
```
1    1470
Name: EmployeeCount, dtype: int64
```

In [8]:
```
hr_df.drop(['EmployeeCount'], axis=1, inplace=True)
```

Column `StandardHours` contains only one value which is 80. There is a possibility that in the future, there will be a new value. However, at present, we cannot invent rows with say, 88, without considering other variables. There, this column needs to be dropped.

In [9]:
```
hr_df['StandardHours'].value_counts()
```

Out[9]:
```
80    1470
Name: StandardHours, dtype: int64
```

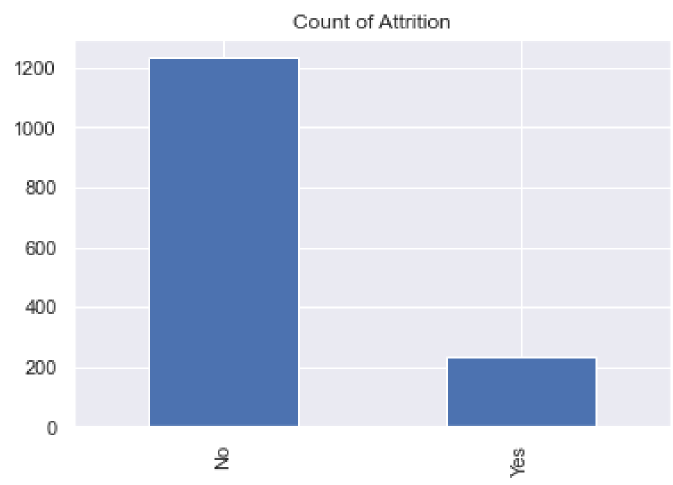
In [10]:
```
hr_df.drop(['StandardHours'], axis=1, inplace=True)
```

2. Exploring the Data

2.1. Exploring the Whole Data

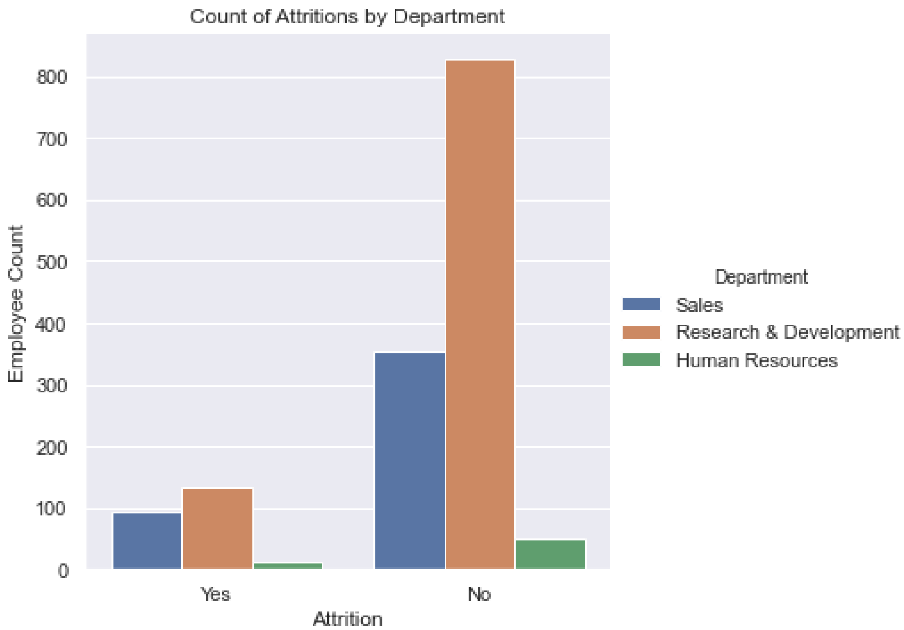
In the attrition, those who stayed (No) greatly outnumber those who left (Yes).

In [11]:
```
ax = hr_df['Attrition'].value_counts().plot(kind='bar')
ax.set_title('Count of Attrition');
```



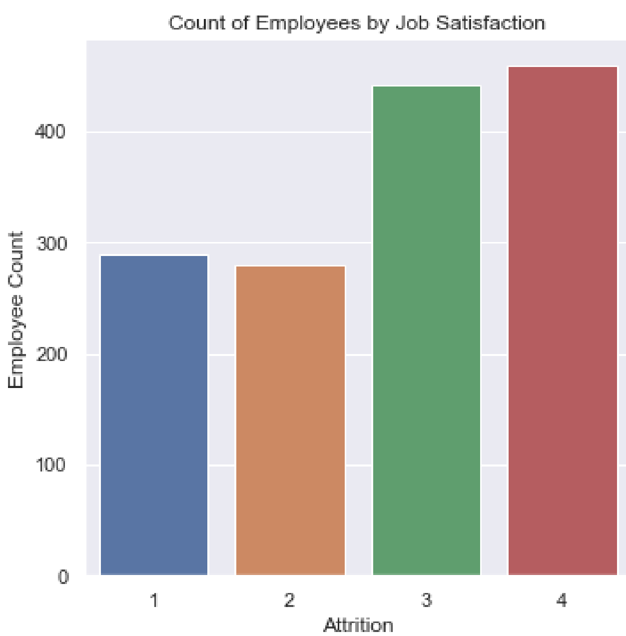
Research & Development department has the highest number of employees, followed by Sales and Human Resources. R & D also has the highest number of employees who left.

In [12]:
```
fig = sns.catplot(x='Attrition', kind='count', hue='Department' ,data=hr_df )
fig.set(xlabel='Attrition', ylabel="Employee Count", title="Count of Attritions by Departm
ent");
```



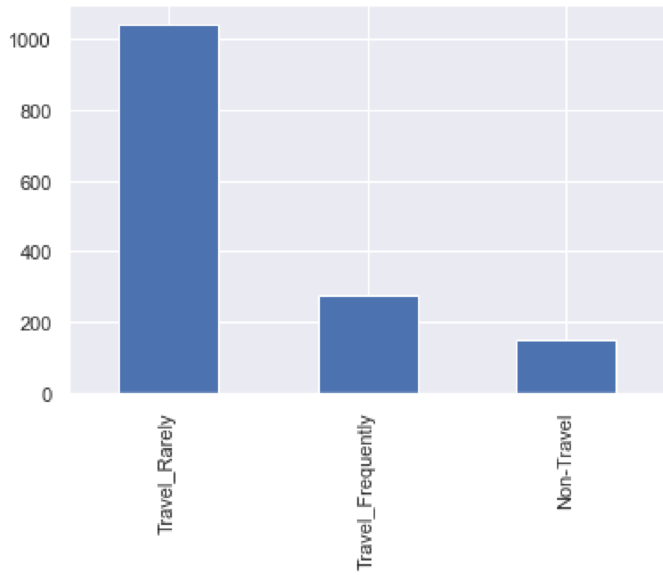
If combined, those with Job Satisfaction of 1 and 2 are greater than those with 3 and 4.

In [13]:
```
fig = sns.catplot(x='JobSatisfaction', kind='count', data=hr_df )
fig.set(xlabel='Attrition', ylabel="Employee Count", title="Count of Employees by Job Sati
sfaction");
```



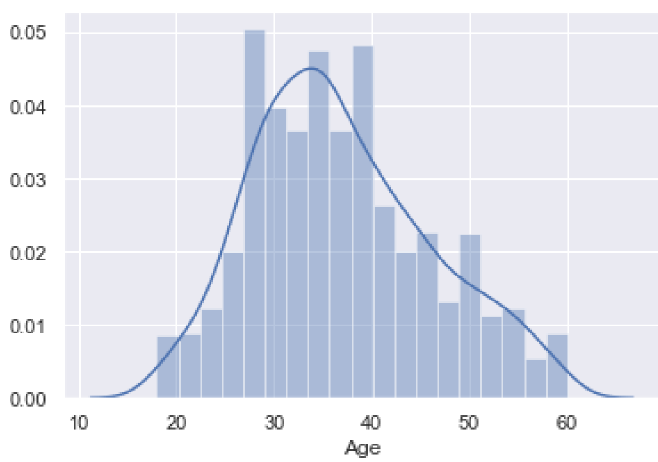
Majority of employees traver rarely.

In [14]:
```
hr_df['BusinessTravel'].value_counts().plot(kind='bar');
```



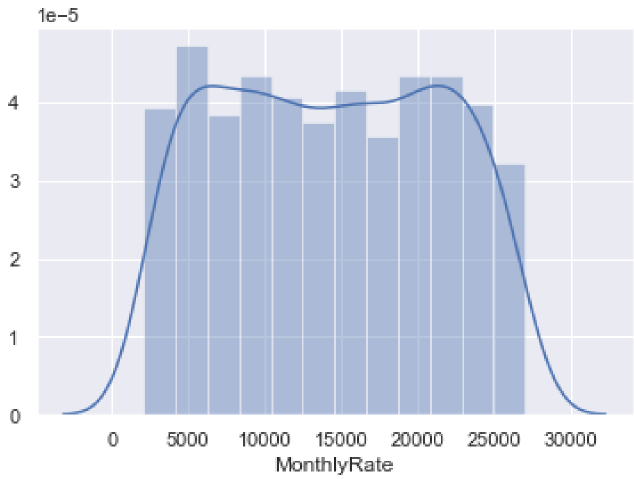
In [15]:
```
print("Mean Age: {}".format(hr_df['Age'].mean()))
sns.distplot(hr_df['Age']);
```

```
Mean Age: 36.923809523809524
```



In [16]:
```
print("Mean Monthly Rate: {}".format(hr_df['MonthlyRate'].mean()))
sns.distplot(hr_df['MonthlyRate']);
```

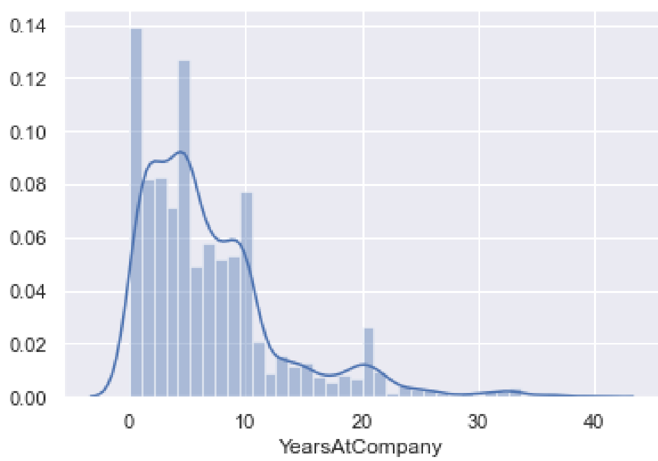
```
Mean Monthly Rate: 14313.103401360544
```



While the average tenure is 7 years, there are some outliers. These are employees who have stayed for longer than 10 years.

In [17]:
```
print("Mean Tenure (YearsAtCompany): {}".format(hr_df['YearsAtCompany'].mean()))
sns.distplot(hr_df['YearsAtCompany']);
```

```
Mean Tenure (YearsAtCompany): 7.0081632653061225
```



This chart shows the distribution and average monthly rates for each department. Human Resources has the lowest average monthly salary.

In [18]:
```python
unique_depts = hr_df['Department'].unique()

for dept in unique_depts:
    subset = hr_df[hr_df['Department'] == dept]
    dept_mean = subset['MonthlyRate'].mean()
    dept_label = '{} (Ave: {:.2f})'.format(dept, dept_mean)
    sns.distplot(subset['MonthlyRate'], hist=False, label=dept_label)

plt.legend(title='Departments');
```

2.2. Exploring those who Left

In [19]:
```python
leaver_df = hr_df[hr_df['Attrition'] == 'Yes']
```

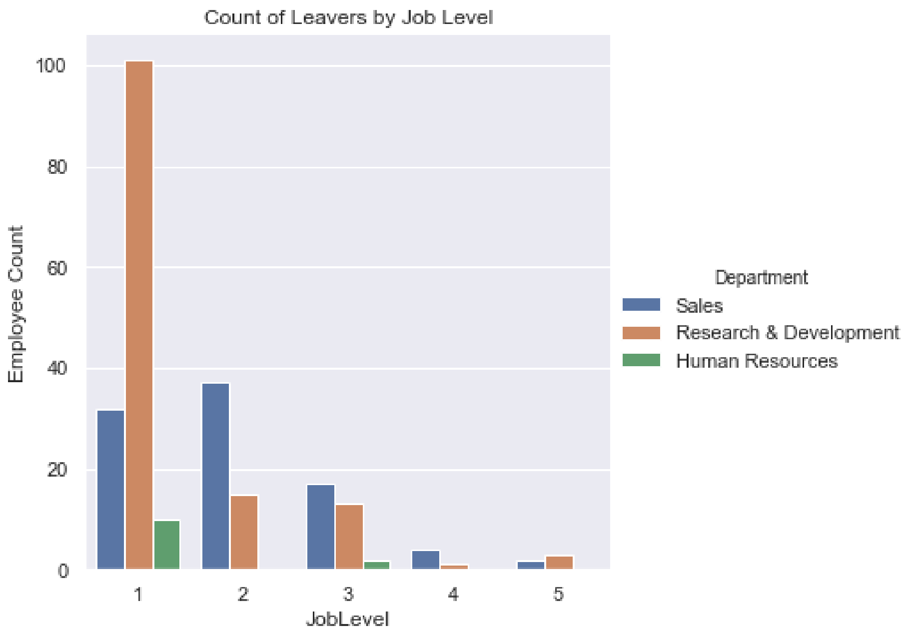
In [20]:
```python
leaver_df.describe(include='all')
```

Out[20]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educa
count	237.000000	237	237	237.000000	237	237.000000	237.000000	
unique	NaN	1	3	NaN	3	NaN	NaN	
top	NaN	Yes	Travel_Rarely	NaN	Research & Development	NaN	NaN	Life
freq	NaN	237	156	NaN	133	NaN	NaN	
mean	33.607595	NaN	NaN	750.362869	NaN	10.632911	2.839662	
std	9.689350	NaN	NaN	401.899519	NaN	8.452525	1.008244	
min	18.000000	NaN	NaN	103.000000	NaN	1.000000	1.000000	
25%	28.000000	NaN	NaN	408.000000	NaN	3.000000	2.000000	
50%	32.000000	NaN	NaN	699.000000	NaN	9.000000	3.000000	
75%	39.000000	NaN	NaN	1092.000000	NaN	17.000000	4.000000	
max	58.000000	NaN	NaN	1496.000000	NaN	29.000000	5.000000	

Many R&D employees of level 1 have left the company, followed by the Sales members.

In [21]:
```
fig = sns.catplot(x='JobLevel', kind='count', hue='Department', data=leaver_df )
fig.set(xlabel='JobLevel', ylabel="Employee Count", title="Count of Leavers by Job Level"
);
```

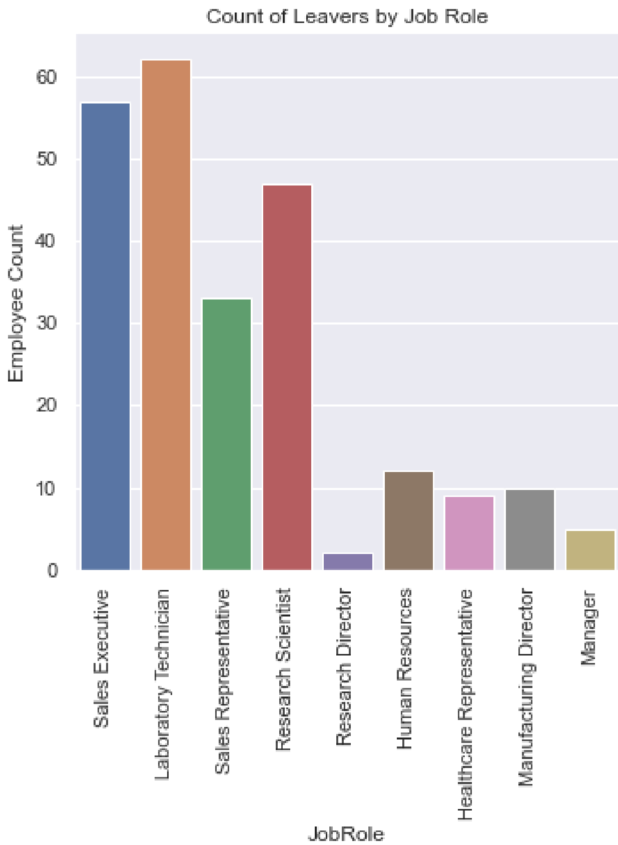


In [22]:
```
leaver_df[leaver_df['Department'] == 'Research & Development']['JobRole'].value_counts()
```

Out[22]:
```
Laboratory Technician        62
Research Scientist           47
Manufacturing Director       10
Healthcare Representative     9
Manager                       3
Research Director             2
Name: JobRole, dtype: int64
```

Laboratory Technicians, Sales Executives and Research Scientists are the roles with the most number of leavers.

In [23]:
```
fig = sns.catplot(x='JobRole', kind='count', data=leaver_df )
fig.set(xlabel='JobRole', ylabel="Employee Count", title="Count of Leavers by Job Role")
fig.set_xticklabels(rotation=90);
```



3. Reusable Functions

Below cells contain functions that will be reused during the logistic regressions.

In [24]:
```python
def get_features_and_target(dataframe, target_column):

    if isinstance(target_column, str):
        target = dataframe[target_column]
        features = dataframe.drop([target_column], axis=1)
    elif isinstance(target_column, pd.core.series.Series):
        target = target_column
        features = dataframe

    return features, target
```

In [25]:
```
def statsmodel_logit(features, target):

    if 'const' not in features.columns:
        features = sm.add_constant(features)

    # Create a Logit object
    sm_obj = sm.Logit(target, features)

    # Create a fitted logit model
    sm_model = sm_obj.fit()

    return sm_model
```

In [26]:
```
def sklearn_logit(features, target):

    with warnings.catch_warnings():
        warnings.filterwarnings('error')
        try:
            sk_model = LogisticRegression(fit_intercept=True)
            sk_model.fit(features, target)
        except ConvergenceWarning:
            # The default max_iter is 1000. There are cases when during fitting, max_iter
 has been reached
            # If the fitting has encountered this warning, increase max_iter to 3000
            # Set C to a high number as well to make sure that statsmodels' coefs match wi
th sklearn's
            sk_model = LogisticRegression(fit_intercept=True, C=1e9, max_iter=3000)
            sk_model.fit(features, target)

    score = sk_model.score(features, target)
    print('Model Accuracy Score: {}%'.format(score * 100))

    return sk_model
```

In [27]:
```
def sklearn_predict(sklearn_model, features, actual_target, title):
    predicted_target = sklearn_model.predict(features)

    labels = actual_target.unique()
    cm = plot_confusion_matrix(sklearn_model, X=features, y_true=actual_target, display_la
bels=labels,
                               values_format='.2f', cmap=plt.cm.Blues)
    cm.ax_.set_title(title)
    plt.show()
```

4. Logistic Regression on Initial Data State

This section attempts to train a model to predict which employee will be a part of the attrition. The dependent variable is **Attrition**. Since it has two values, Yes and No, I used binary logistic regression.

<u>For clarity, the Attrition Values are:</u>
Yes (1) = Employee left the company
No (2) = Employee stayed in the company

4.1. Initial Data Preparation

4.1.1. Train Test Split

Before proceeding with the `train_test_split` , I added a constant column of 1s. This will be useful later on for the Varianc Inflation Factor and for running statmodels' `Logit` .

In [28]:
```
# Add constants column for the whole dataset
# https://stackoverflow.com/questions/42658379/variance-inflation-factor-in-python
hr_df = sm.add_constant(hr_df)
```

In [29]:
```
# Split the dataset into train and test
target = hr_df['Attrition']
features = hr_df.drop(['Attrition'], axis=1)
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, rando
m_state=432)
```

I turned `x_train` and `y_train` into `train_df` . All preprocessing will be fitted on the `train_df` and applied to the `test_df` after. I also reinserted the `Attrition` at its original column index 0. This will come in handy later on during One Hot Encoding.

In [30]:
```
train_df = x_train.copy()
train_df.insert(0, 'Attrition', y_train)
train_df.reset_index(drop=True, inplace=True)
train_df.head(2)
```

Out[30]:

	Attrition	const	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationFielc
0	No	1.0	31	Travel_Rarely	1274	Research & Development	9	1	Life Sciences
1	No	1.0	28	Travel_Rarely	640	Research & Development	1	3	Technica Degree

Quick sanity check if the re-inserted `Attrition` column still has the same values with the original `hr_df` . Join the train_df and hr_df on `EmployeeNumber` resulting to a new dataframe, `check_train_df` .

In [31]:
```
check_train_df = pd.merge(left=train_df, right=hr_df, on=['EmployeeNumber'])
```

If `check_train_df.Attrition` is not equal with `hr_df.Attrition` it should show some rows. Luckily, it didn't, so the reinserted Attrition column in `train_df` still retained the original values.

In [32]:
```
check_train_df[(check_train_df['Attrition_x'] != check_train_df['Attrition_y'])]
```

Out[32]:

Attrition_x	const_x	Age_x	BusinessTravel_x	DailyRate_x	Department_x	DistanceFromHome_x	Education_x

For the `test_df` , do the same method as that for `train_df` . A quick check ensured that the reinserted Attrition column still has the original values.

In [33]:
```
test_df = x_test.copy()
test_df.insert(0, 'Attrition', y_test)
test_df.reset_index(drop=True, inplace=True)
test_df.head(2)
```

Out[33]:

	Attrition	const	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField
0	Yes	1.0	50	Travel_Rarely	869	Sales	3	2	Marketing
1	No	1.0	27	Travel_Rarely	1103	Research & Development	14	3	Life Sciences

In [34]:
```
check_test_df = pd.merge(left=test_df, right=hr_df, on=['EmployeeNumber'])
check_test_df[(check_test_df['Attrition_x'] != check_test_df['Attrition_y'])]
```

Out[34]:

Attrition_x	const_x	Age_x	BusinessTravel_x	DailyRate_x	Department_x	DistanceFromHome_x	Education_x

4.1.2. One-Hot Encoding the Training Dataset

These columns have categorical values. To be able to proceed with preprocessing and model fitting, I used One-Hot Encoding.

In [35]:
```
nominal_categories = ['Attrition', 'BusinessTravel', 'Department', 'EducationField',
                      'Gender', 'JobRole', 'MaritalStatus', 'Over18', 'OverTime']
```

Multicollinearity is the problem where 2 or more variables are highly correlated. To avoid this, the argument `drop='first'` is specified.

In [36]:
```
ohe = OneHotEncoder(categories='auto', drop='first')
```

Fit OHE on to the `train_df` , then use it to transform the same dataframe.

In [37]:
```
ohe.fit(train_df[nominal_categories])
train_arr = ohe.transform(train_df[nominal_categories]).toarray()
```

These are the new feature names generated from OHE.

In [38]:
```
train_cols = ohe.get_feature_names()
train_cols_list = train_cols.tolist()
print(train_cols_list)
```

```
['x0_Yes', 'x1_Travel_Frequently', 'x1_Travel_Rarely', 'x2_Research & Development', 'x2_S
ales', 'x3_Life Sciences', 'x3_Marketing', 'x3_Medical', 'x3_Other', 'x3_Technical Degre
e', 'x4_Male', 'x5_Human Resources', 'x5_Laboratory Technician', 'x5_Manager', 'x5_Manufa
cturing Director', 'x5_Research Director', 'x5_Research Scientist', 'x5_Sales Executive',
'x5_Sales Representative', 'x6_Married', 'x6_Single', 'x8_Yes']
```

For each new feature name, append the original column name with the new value name. Put them all in a list.

In [39]:
```
new_cols_list = []
for index, value in enumerate(train_cols_list):
    new_cols_list.append(nominal_categories[int(value[1])] + value[2:])
    continue
print(new_cols_list)
```

```
['Attrition_Yes', 'BusinessTravel_Travel_Frequently', 'BusinessTravel_Travel_Rarely', 'De
partment_Research & Development', 'Department_Sales', 'EducationField_Life Sciences', 'Ed
ucationField_Marketing', 'EducationField_Medical', 'EducationField_Other', 'EducationFiel
d_Technical Degree', 'Gender_Male', 'JobRole_Human Resources', 'JobRole_Laboratory Techni
cian', 'JobRole_Manager', 'JobRole_Manufacturing Director', 'JobRole_Research Director',
'JobRole_Research Scientist', 'JobRole_Sales Executive', 'JobRole_Sales Representative',
'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_Yes']
```

Create `train_ohe_df` with the transformed `train_arr`. Label the columns with the `new_cols_list`.

In [40]:
```
train_ohe_df = pd.DataFrame(train_arr, columns=new_cols_list)
train_ohe_df.head(2)
```

Out[40]:

	Attrition_Yes	BusinessTravel_Travel_Frequently	BusinessTravel_Travel_Rarely	Department_Research & Development	Departm
0	0.0	0.0	1.0	1.0	
1	0.0	0.0	1.0	1.0	

Merge the original `train_df` and `train_ohe_df` by their indexes. The result will be the new value of `train_df`.

In [41]:
```
train_df = pd.merge(left=train_df, right=train_ohe_df, left_index=True, right_index=True)
```

Drop the original nominal category columns so that only the one-hot encoded columns remain

In [42]:
```
train_df.drop(nominal_categories, axis=1, inplace=True)
```

Drop `EmployeeNumber` column since they will affect the model even if they are meaningless.

In [43]:
```
train_df.drop(['EmployeeNumber'], axis=1, inplace=True)
```

In [44]:
```
train_df.head(2)
```

Out[44]:

	const	Age	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement
0	1.0	31	1274	9	1	3	33	3
1	1.0	28	640	1	3	4	84	3

4.1.3. Applying OHE to the Test Dataset

With the OHE fitted on the training dataset, use it again to transform the `test_df` 's categorical columns. Do the same processing for `test_df` after.

In [45]:
```
test_arr = ohe.transform(test_df[nominal_categories]).toarray()
```

In [46]:
```
test_cols = ohe.get_feature_names()
test_cols_list = test_cols.tolist()
```

In [47]:
```
new_cols_list = []
for index, value in enumerate(test_cols_list):
    new_cols_list.append(nominal_categories[int(value[1])] + value[2:])
    continue
```

In [48]:
```
test_ohe_df = pd.DataFrame(test_arr, columns=new_cols_list)
test_ohe_df.head(2)
```

Out[48]:

	Attrition_Yes	BusinessTravel_Travel_Frequently	BusinessTravel_Travel_Rarely	Department_Research & Development	Departm
0	1.0	0.0	1.0	0.0	
1	0.0	0.0	1.0	1.0	

In [49]:
```
test_df = pd.merge(left=test_df, right=test_ohe_df, left_index=True, right_index=True)
```

In [50]:
```
test_df.drop(nominal_categories, axis=1, inplace=True)
```

In [51]:
```
test_df.drop(['EmployeeNumber'], axis=1, inplace=True)
```

4.1.4. Variance Inflation Factor

To detect multicollinearity, I used Variance Inflation Factor (VIF).

In [52]:

```
vif_df = pd.DataFrame()
variables = train_df.drop(['Attrition_Yes'], axis=1)
vif_df['Independent Variables'] = variables.columns
vif_df['VIF'] = [vif(variables.values, i) for i in range(variables.shape[1])]
vif_df.sort_values(by='VIF')
```

Out[52]:

	Independent Variables	VIF
3	DistanceFromHome	1.019908
6	HourlyRate	1.030179
11	MonthlyRate	1.032481
33	Gender_Male	1.032588
9	JobSatisfaction	1.033529
15	RelationshipSatisfaction	1.035164
19	WorkLifeBalance	1.039642
7	JobInvolvement	1.041250
18	TrainingTimesLastYear	1.041352
5	EnvironmentSatisfaction	1.041774
2	DailyRate	1.043193
44	OverTime_Yes	1.045045
4	Education	1.080111
12	NumCompaniesWorked	1.263437
22	YearsSinceLastPromotion	1.771375
42	MaritalStatus_Married	1.808821
16	StockOptionLevel	1.913115
37	JobRole_Manufacturing Director	1.944353
1	Age	2.035180
24	BusinessTravel_Travel_Frequently	2.417022
25	BusinessTravel_Travel_Rarely	2.418450
14	PerformanceRating	2.555080
13	PercentSalaryHike	2.576426
38	JobRole_Research Director	2.723024
23	YearsWithCurrManager	2.773584
21	YearsInCurrentRole	2.794820
43	MaritalStatus_Single	2.967017
35	JobRole_Laboratory Technician	3.221695
39	JobRole_Research Scientist	3.431305
36	JobRole_Manager	4.473090
20	YearsAtCompany	4.740765
41	JobRole_Sales Representative	5.025629
17	TotalWorkingYears	5.123743
31	EducationField_Other	5.734186
34	JobRole_Human Resources	7.486205
32	EducationField_Technical Degree	8.747694
29	EducationField_Marketing	11.075825
40	JobRole_Sales Executive	13.746882
8	JobLevel	14.570042
10	MonthlyIncome	19.216663

	Independent Variables	VIF
30	EducationField_Medical	21.445197
28	EducationField_Life Sciences	24.054523
27	Department_Sales	46.152920
26	Department_Research & Development	46.584758
0	const	429.360556

Drop the columns whose VIF are greater than 10 from both `train_df` and `test_df` .

In [53]:
```
high_vif_columns = vif_df[(vif_df['VIF'] > 10) & (vif_df['Independent Variables'] != 'cons
t')]['Independent Variables']
high_vif_columns = high_vif_columns.values
print(high_vif_columns)
```

```
['JobLevel' 'MonthlyIncome' 'Department_Research & Development'
 'Department_Sales' 'EducationField_Life Sciences'
 'EducationField_Marketing' 'EducationField_Medical'
 'JobRole_Sales Executive']
```

In [54]:
```
train_df.drop(high_vif_columns, axis=1, inplace=True)
test_df.drop(high_vif_columns, axis=1, inplace=True)
```

4.2. Logistic Regression on Data after OHE and VIF

4.2.1. Training the Model

In [55]:
```
train_features, train_target = get_features_and_target(train_df, 'Attrition_Yes')
```

In [56]:
```
m1_sm_model = statsmodel_logit(train_features, train_target)
```

```
Optimization terminated successfully.
         Current function value: 0.284189
         Iterations 8
```

In [57]:

```
m1_sm_model.summary()
```

Out[57]:

Logit Regression Results

Dep. Variable:	Attrition_Yes	**No. Observations:**	1176
Model:	Logit	**Df Residuals:**	1139
Method:	MLE	**Df Model:**	36
Date:	Tue, 21 Apr 2020	**Pseudo R-squ.:**	0.3448
Time:	00:08:48	**Log-Likelihood:**	-334.21
converged:	True	**LL-Null:**	-510.10
Covariance Type:	nonrobust	**LLR p-value:**	1.876e-53

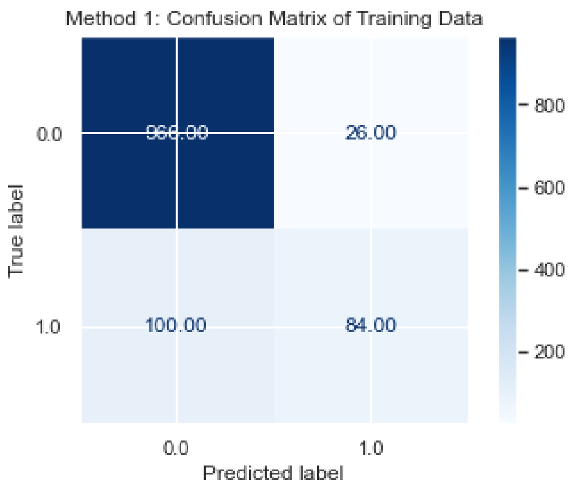
	coef	std err	z	P>|z|	[0.025	0.975]
const	1.6982	1.492	1.138	0.255	-1.226	4.623
Age	-0.0342	0.015	-2.232	0.026	-0.064	-0.004
DailyRate	-0.0004	0.000	-1.441	0.150	-0.001	0.000
DistanceFromHome	0.0522	0.012	4.287	0.000	0.028	0.076
Education	0.0486	0.099	0.490	0.624	-0.146	0.243
EnvironmentSatisfaction	-0.4767	0.095	-5.026	0.000	-0.663	-0.291
HourlyRate	0.0040	0.005	0.789	0.430	-0.006	0.014
JobInvolvement	-0.5154	0.138	-3.730	0.000	-0.786	-0.245
JobSatisfaction	-0.3393	0.094	-3.604	0.000	-0.524	-0.155
MonthlyRate	1.398e-05	1.43e-05	0.978	0.328	-1.4e-05	4.2e-05
NumCompaniesWorked	0.2226	0.042	5.247	0.000	0.139	0.306
PercentSalaryHike	-0.0340	0.045	-0.753	0.451	-0.122	0.054
PerformanceRating	0.2377	0.454	0.524	0.600	-0.651	1.127
RelationshipSatisfaction	-0.2209	0.094	-2.348	0.019	-0.405	-0.037
StockOptionLevel	-0.2917	0.176	-1.659	0.097	-0.636	0.053
TotalWorkingYears	-0.0684	0.030	-2.286	0.022	-0.127	-0.010
TrainingTimesLastYear	-0.2558	0.083	-3.080	0.002	-0.419	-0.093
WorkLifeBalance	-0.4144	0.143	-2.888	0.004	-0.696	-0.133
YearsAtCompany	0.1050	0.043	2.462	0.014	0.021	0.189
YearsInCurrentRole	-0.1519	0.051	-2.958	0.003	-0.253	-0.051
YearsSinceLastPromotion	0.1603	0.048	3.363	0.001	0.067	0.254
YearsWithCurrManager	-0.1455	0.051	-2.826	0.005	-0.246	-0.045
BusinessTravel_Travel_Frequently	1.7595	0.475	3.706	0.000	0.829	2.690
BusinessTravel_Travel_Rarely	1.0772	0.437	2.465	0.014	0.221	1.934
EducationField_Other	0.1825	0.451	0.405	0.686	-0.701	1.066
EducationField_Technical Degree	0.9444	0.318	2.971	0.003	0.321	1.567
Gender_Male	0.4026	0.210	1.919	0.055	-0.009	0.814
JobRole_Human Resources	0.8306	0.486	1.707	0.088	-0.123	1.784
JobRole_Laboratory Technician	0.5852	0.291	2.010	0.044	0.014	1.156
JobRole_Manager	-0.7580	0.635	-1.194	0.232	-2.002	0.486
JobRole_Manufacturing Director	-1.0030	0.504	-1.991	0.046	-1.990	-0.016
JobRole_Research Director	-1.3319	0.831	-1.604	0.109	-2.960	0.296

JobRole_Research Scientist	-0.1178	0.298	-0.396	0.692	-0.701	0.466
JobRole_Sales Representative	1.1972	0.403	2.969	0.003	0.407	1.987
MaritalStatus_Married	0.3926	0.305	1.288	0.198	-0.205	0.990
MaritalStatus_Single	1.0671	0.395	2.698	0.007	0.292	1.842
OverTime_Yes	1.9744	0.222	8.910	0.000	1.540	2.409

In [58]:
```
sk_model = sklearn_logit(train_features, train_target)
```

```
Model Accuracy Score: 89.28571428571429%
```

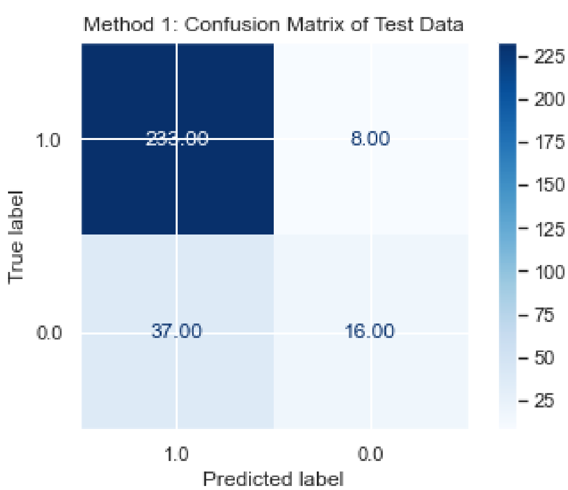
In [59]:
```
sklearn_predict(sk_model, train_features, train_target, title='Method 1: Confusion Matrix of Training Data')
```



4.2.2 Testing the Model

In [60]:
```
test_features, test_target = get_features_and_target(test_df, 'Attrition_Yes')
```

In [61]:
```
sklearn_predict(sk_model, test_features, test_target, title='Method 1: Confusion Matrix of Test Data')
```



4.2.3. Conclusion

The model has an accuracy score of **89.29%**. The pseudo R-squared is **0.3448**. However, from statsmodels' summary, there are many variables whose p-values are greater than 0.05 .

There is a possibility that accuracy can be increased if scaled data is used. In the next section, the data will be scaled.

5. Logistic Regression on a Scaled Dataset

5.1. Fitting the Scaler on the Training Dataset

The `scaled_train_df` is a copy of `train_df` . Sklearn's `StandardScaler` is fitted on the training data. The scaled results are now in the `scaled_train_df` .

During fitting and transforming, columns `Attrition_Yes` and `const` shouldn't be scaled so I left them out.

In [62]:
```
scaled_train_df = train_df.copy()
```

In [63]:
```
scaler = StandardScaler()
scaler.fit(scaled_train_df.drop(['Attrition_Yes', 'const'], axis=1))
```

Out[63]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [64]:
```
scaled_train_arr = scaler.transform(train_df.drop(['Attrition_Yes', 'const'], axis=1))
```

Put the results `scaled_train_arr` in a dataframe, then merge them back with `train_df` 's columns `Attrition_Yes` and `const` .

In [65]:
```
scaled_train_df = pd.DataFrame(scaled_train_arr, columns=train_df.drop(['Attrition_Yes',
'const'], axis=1).columns)
scaled_train_df = pd.merge(left=scaled_train_df, right=train_df[['Attrition_Yes', 'const'
]], left_index=True, right_index=True)
scaled_train_df.head(3)
```

Out[65]:

	Age	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	J
0	-0.689868	1.143634	-0.016933	-1.917270	0.236912	-1.614977	0.392794	
1	-1.021999	-0.409053	-1.006426	0.071887	1.156410	0.906036	0.392794	
2	1.967178	0.695461	-0.882739	-1.917270	1.156410	-0.033165	0.392794	

5.2 Scaling the Test Dataset

Apply the fitted StandardScaler to transform the test dataset. After that, do the same steps to get `scaled_test_df` .

In [66]:
```
scaled_test_arr = scaler.transform(test_df.drop(['Attrition_Yes', 'const'], axis=1))
```

In [67]:
```
scaled_test_df = pd.DataFrame(scaled_test_arr, columns=test_df.drop(['Attrition_Yes', 'con
st'], axis=1).columns)
scaled_test_df = pd.merge(left=scaled_test_df, right=test_df[['Attrition_Yes', 'const']],
left_index=True, right_index=True)
scaled_test_df.head(3)
```

Out[67]:

	Age	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	J
0	1.413627	0.151775	-0.759053	-0.922691	-1.602085	1.004900	-1.002752	
1	-1.132709	0.724849	0.601500	0.071887	-1.602085	-1.170092	0.392794	
2	-0.136317	-0.607425	0.354126	1.066465	0.236912	0.510583	0.392794	

5.3. Logistic Regression on Scaled Data

5.3.1. Training the Model

In [68]:
```
train_features, train_target = get_features_and_target(scaled_train_df, 'Attrition_Yes')
```

In [69]:
```
m2_sm_model = statsmodel_logit(train_features, train_target)
```

```
Optimization terminated successfully.
         Current function value: 0.284189
         Iterations 8
```

In [70]:
```
m2_sm_model.summary()
```

Out[70]:

Logit Regression Results

Dep. Variable:	Attrition_Yes	**No. Observations:**	1176
Model:	Logit	**Df Residuals:**	1139
Method:	MLE	**Df Model:**	36
Date:	Tue, 21 Apr 2020	**Pseudo R-squ.:**	0.3448
Time:	00:08:55	**Log-Likelihood:**	-334.21
converged:	True	**LL-Null:**	-510.10
Covariance Type:	nonrobust	**LLR p-value:**	1.876e-53

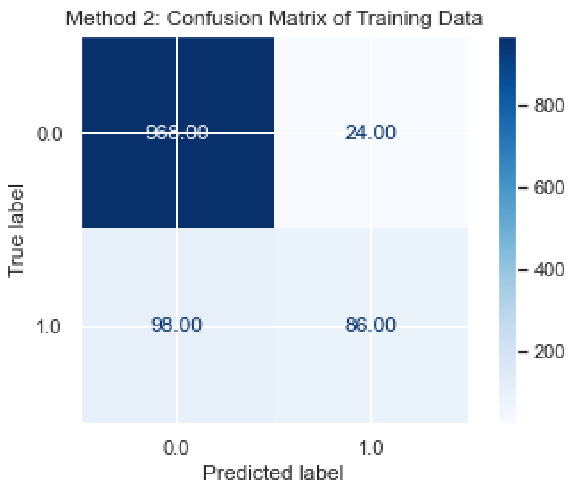
	coef	std err	z	P>|z|	[0.025	0.975]
Age	-0.3088	0.138	-2.232	0.026	-0.580	-0.038
DailyRate	-0.1458	0.101	-1.441	0.150	-0.344	0.053
DistanceFromHome	0.4218	0.098	4.287	0.000	0.229	0.615
Education	0.0488	0.100	0.490	0.624	-0.146	0.244
EnvironmentSatisfaction	-0.5185	0.103	-5.026	0.000	-0.721	-0.316
HourlyRate	0.0810	0.103	0.789	0.430	-0.120	0.282
JobInvolvement	-0.3693	0.099	-3.730	0.000	-0.563	-0.175
JobSatisfaction	-0.3717	0.103	-3.604	0.000	-0.574	-0.170
MonthlyRate	0.0994	0.102	0.978	0.328	-0.100	0.299
NumCompaniesWorked	0.5558	0.106	5.247	0.000	0.348	0.763
PercentSalaryHike	-0.1238	0.164	-0.753	0.451	-0.446	0.198
PerformanceRating	0.0852	0.163	0.524	0.600	-0.233	0.404
RelationshipSatisfaction	-0.2389	0.102	-2.348	0.019	-0.438	-0.039
StockOptionLevel	-0.2523	0.152	-1.659	0.097	-0.550	0.046
TotalWorkingYears	-0.5345	0.234	-2.286	0.022	-0.993	-0.076
TrainingTimesLastYear	-0.3292	0.107	-3.080	0.002	-0.539	-0.120
WorkLifeBalance	-0.2906	0.101	-2.888	0.004	-0.488	-0.093
YearsAtCompany	0.6495	0.264	2.462	0.014	0.133	1.166
YearsInCurrentRole	-0.5574	0.188	-2.958	0.003	-0.927	-0.188
YearsSinceLastPromotion	0.5255	0.156	3.363	0.001	0.219	0.832
YearsWithCurrManager	-0.5220	0.185	-2.826	0.005	-0.884	-0.160
BusinessTravel_Travel_Frequently	0.6897	0.186	3.706	0.000	0.325	1.055
BusinessTravel_Travel_Rarely	0.4892	0.198	2.465	0.014	0.100	0.878
EducationField_Other	0.0408	0.101	0.405	0.686	-0.157	0.238
EducationField_Technical Degree	0.2681	0.090	2.971	0.003	0.091	0.445
Gender_Male	0.1974	0.103	1.919	0.055	-0.004	0.399
JobRole_Human Resources	0.1559	0.091	1.707	0.088	-0.023	0.335
JobRole_Laboratory Technician	0.2207	0.110	2.010	0.044	0.005	0.436
JobRole_Manager	-0.2015	0.169	-1.194	0.232	-0.532	0.129
JobRole_Manufacturing Director	-0.3002	0.151	-1.991	0.046	-0.596	-0.005
JobRole_Research Director	-0.3066	0.191	-1.604	0.109	-0.681	0.068
JobRole_Research Scientist	-0.0461	0.117	-0.396	0.692	-0.274	0.182

JobRole_Sales Representative	0.2716	0.091	2.969	0.003	0.092	0.451
MaritalStatus_Married	0.1953	0.152	1.288	0.198	-0.102	0.493
MaritalStatus_Single	0.4948	0.183	2.698	0.007	0.135	0.854
OverTime_Yes	0.8920	0.100	8.910	0.000	0.696	1.088
const	-2.7582	0.161	-17.181	0.000	-3.073	-2.444

In [71]:
```
m2_sk_model = sklearn_logit(train_features, train_target)
```

```
Model Accuracy Score: 89.62585034013605%
```

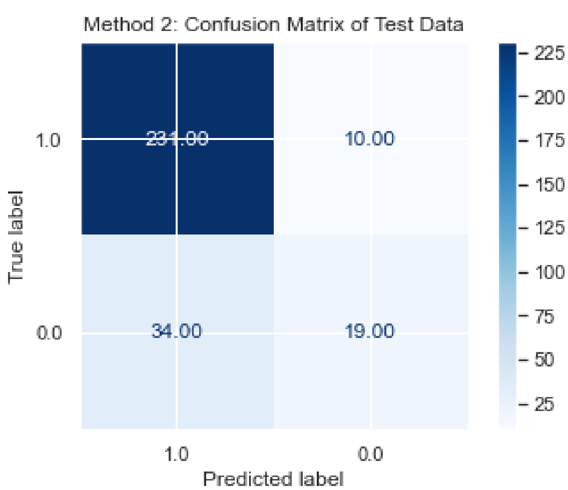
In [72]:
```
sklearn_predict(m2_sk_model, train_features, train_target, title='Method 2: Confusion Matrix of Training Data')
```



5.3.2. Testing the Model

In [73]:
```
test_features, test_target = get_features_and_target(scaled_test_df, 'Attrition_Yes')
```

In [74]:
```
sklearn_predict(m2_sk_model, test_features, test_target, title='Method 2: Confusion Matrix of Test Data')
```



5.3.2. Conclusion

The model has an accuracy score of **89.62%**. It's an increase of 0.33% from the previous model. However, the pseudo R-squared is still **0.3448** and there are still a lot of variables whose p-values are greater than 0.05 .

6. Logistic Regression of Selected Features

6.1. Recursive Feature Elimination with Cross Validation (RFECV)

Recursive Feature Elimination (RFE) repeatedly fits a model and chooses the best features based on their coefficients. RFE with Cross Validation (RFECV) cross-validates the number of features selected.

Create a LogisticRegression object.

In [75]:
```
log_reg = LogisticRegression()
```

In the RFECV instance, the `estimator` is the LogisticRegression object. `StratifiedKFold()` returns approximately the same percentage samples of the targets. `scoring='accuracy'` uses `sklearn.metrics.accuracy_score` to compute multilable classification accuracy.

In [76]:
```
rfecv = RFECV(estimator=log_reg, cv=StratifiedKFold(), scoring='accuracy')
```

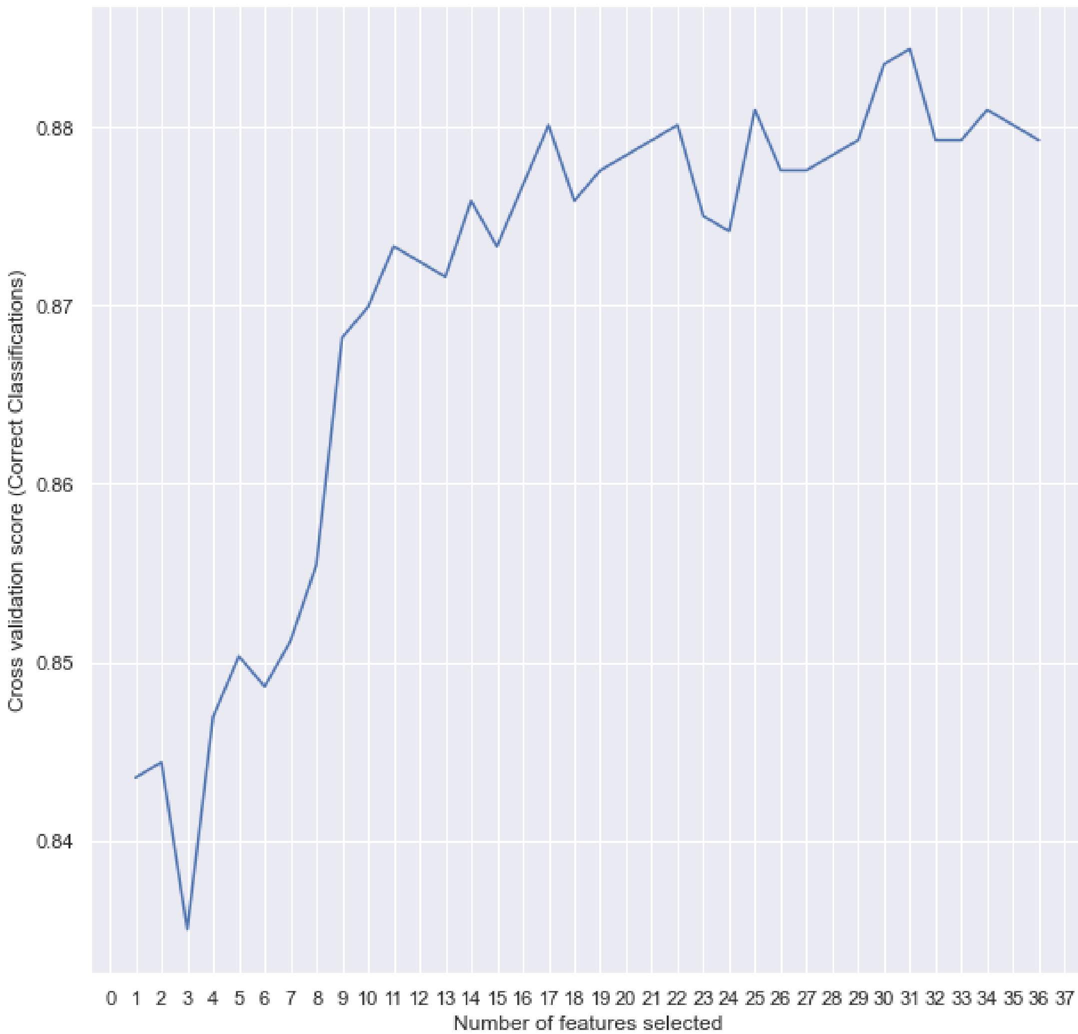
Fit the `rfecv` on the training features (except `const`) and on the training target.

In [77]:
```
rfecv.fit(scaled_train_df.drop(['Attrition_Yes', 'const'], axis=1), scaled_train_df['Attri
tion_Yes'])
```

Out[77]:
```
RFECV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
      estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                   fit_intercept=True, intercept_scaling=1,
                                   l1_ratio=None, max_iter=100,
                                   multi_class='auto', n_jobs=None,
                                   penalty='l2', random_state=None,
                                   solver='lbfgs', tol=0.0001, verbose=0,
                                   warm_start=False),
      min_features_to_select=1, n_jobs=None, scoring='accuracy', step=1,
      verbose=0)
```

Plot the

In [78]:
```
plt.figure(figsize=(10, 10))
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (Correct Classifications)")
plt.xticks(np.arange(0,50,1));
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_);
```



Based on this plot, the most ideal number of features is **31**.

Get the indices selected by the RFECV, then extract them from the `scaled_train_df` and `scaled_test_df` .

In [79]:
```
rfecv.get_support(indices=True)
```

Out[79]:
```
array([ 0,  1,  2,  4,  6,  7,  8,  9, 10, 12, 13, 14, 15, 16, 17, 18, 19,
       20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35],
      dtype=int32)
```

In [80]:
```
rfecv_idx = rfecv.get_support(indices=True).tolist()
```

Reattach the target `Attrition_Yes` and the `const` column to the datasets.m

In [81]:
```
cols_to_retain = scaled_train_df.iloc[:, rfecv_idx].columns.tolist()
cols_to_retain.append('Attrition_Yes')
cols_to_retain.append('const')
print(len(cols_to_retain))
```

```
33
```

In [82]:
```
scaled_train_df = scaled_train_df[cols_to_retain]
```

In [83]:
```
scaled_test_df = scaled_test_df[cols_to_retain]
```

6.2. Logistic Regression on the RFECV-Selected Features

6.2.1. Train the Model

In [84]:
```
train_features, train_target = get_features_and_target(scaled_train_df, 'Attrition_Yes')
```

In [85]:
```
m3_sm_model = statsmodel_logit(train_features, train_target)
```

```
Optimization terminated successfully.
         Current function value: 0.284805
         Iterations 8
```

In [86]:
```
m3_sm_model.summary()
```

Out[86]:

Logit Regression Results

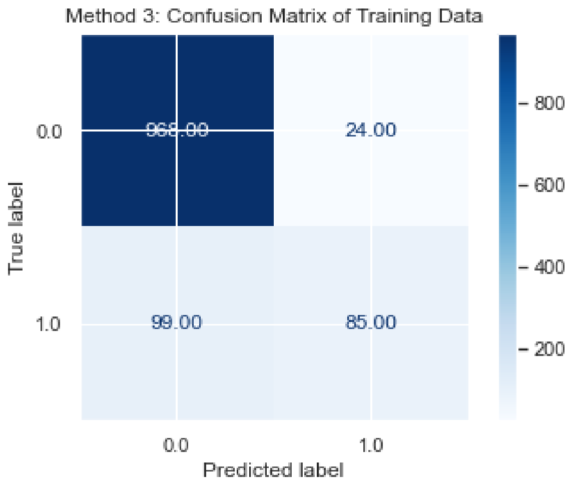
Dep. Variable:	Attrition_Yes	**No. Observations:**	1176
Model:	Logit	**Df Residuals:**	1144
Method:	MLE	**Df Model:**	31
Date:	Tue, 21 Apr 2020	**Pseudo R-squ.:**	0.3434
Time:	00:09:07	**Log-Likelihood:**	-334.93
converged:	True	**LL-Null:**	-510.10
Covariance Type:	nonrobust	**LLR p-value:**	9.291e-56

	coef	std err	z	P>|z|	[0.025	0.975]
Age	-0.2975	0.137	-2.174	0.030	-0.566	-0.029
DailyRate	-0.1410	0.101	-1.400	0.162	-0.338	0.056
DistanceFromHome	0.4199	0.098	4.283	0.000	0.228	0.612
EnvironmentSatisfaction	-0.5156	0.102	-5.048	0.000	-0.716	-0.315
JobInvolvement	-0.3673	0.098	-3.729	0.000	-0.560	-0.174
JobSatisfaction	-0.3806	0.102	-3.713	0.000	-0.581	-0.180
MonthlyRate	0.0909	0.101	0.904	0.366	-0.106	0.288
NumCompaniesWorked	0.5521	0.106	5.227	0.000	0.345	0.759
PercentSalaryHike	-0.0569	0.103	-0.553	0.580	-0.258	0.144
RelationshipSatisfaction	-0.2383	0.101	-2.356	0.018	-0.436	-0.040
StockOptionLevel	-0.2596	0.152	-1.710	0.087	-0.557	0.038
TotalWorkingYears	-0.5119	0.228	-2.242	0.025	-0.959	-0.064
TrainingTimesLastYear	-0.3240	0.107	-3.042	0.002	-0.533	-0.115
WorkLifeBalance	-0.2858	0.100	-2.864	0.004	-0.481	-0.090
YearsAtCompany	0.6353	0.262	2.424	0.015	0.122	1.149
YearsInCurrentRole	-0.5461	0.187	-2.916	0.004	-0.913	-0.179
YearsSinceLastPromotion	0.5185	0.155	3.342	0.001	0.214	0.823
YearsWithCurrManager	-0.5086	0.183	-2.776	0.006	-0.868	-0.149
BusinessTravel_Travel_Frequently	0.6959	0.185	3.767	0.000	0.334	1.058
BusinessTravel_Travel_Rarely	0.4929	0.197	2.503	0.012	0.107	0.879
EducationField_Technical Degree	0.2644	0.090	2.941	0.003	0.088	0.441
Gender_Male	0.1929	0.103	1.881	0.060	-0.008	0.394
JobRole_Human Resources	0.1635	0.087	1.869	0.062	-0.008	0.335
JobRole_Laboratory Technician	0.2414	0.096	2.519	0.012	0.054	0.429
JobRole_Manager	-0.1864	0.167	-1.117	0.264	-0.513	0.141
JobRole_Manufacturing Director	-0.2926	0.147	-1.993	0.046	-0.580	-0.005
JobRole_Research Director	-0.3126	0.191	-1.639	0.101	-0.686	0.061
JobRole_Sales Representative	0.2761	0.085	3.260	0.001	0.110	0.442
MaritalStatus_Married	0.2026	0.151	1.339	0.181	-0.094	0.499
MaritalStatus_Single	0.4929	0.183	2.699	0.007	0.135	0.851
OverTime_Yes	0.8904	0.099	9.009	0.000	0.697	1.084
const	-2.7485	0.160	-17.222	0.000	-3.061	-2.436

In [87]:
```
m3_sk_model = sklearn_logit(train_features, train_target)
```

```
Model Accuracy Score: 89.54081632653062%
```

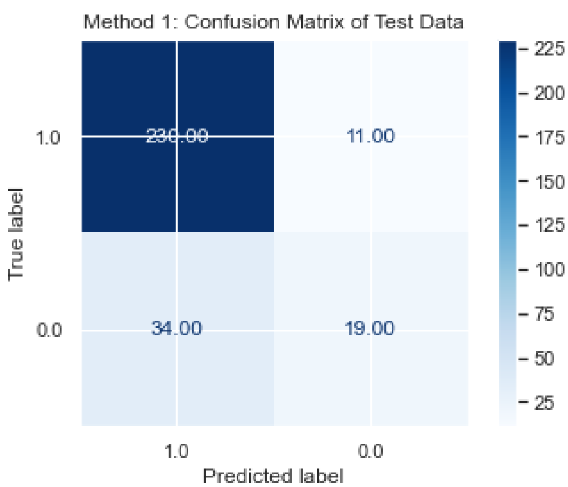
In [88]:
```
sklearn_predict(m3_sk_model, train_features, train_target, title='Method 3: Confusion Matr
ix of Training Data')
```



6.2.2. Test the Model

In [89]:
```
test_features, test_target = get_features_and_target(scaled_test_df, 'Attrition_Yes')
```

In [90]:
```
sklearn_predict(m3_sk_model, test_features, test_target, title='Method 1: Confusion Matrix
of Test Data')
```



6.2.3. Conclusion

There is a slight decrease from the previous results. This model has is **89.54% accurate** and the pseudo R-squared is **0.3434**.

However, nearly all variables are statistically significant, save for a few. These need to be dropped from the training and testing datasets.

6.3. Logistic Regression on Significant Features

6.3.1. Dropping Features with P-values > 0.05

From the last model's summary, there are still a few features left whose p-values are greater than 0.05, meaning they are statistically insignificant.

In [91]:
```
insignificant_cols = ['DailyRate', 'MonthlyRate', 'PercentSalaryHike', 'StockOptionLevel',
                      'Gender_Male', 'JobRole_Human Resources', 'JobRole_Manager',
                      'JobRole_Research Director', 'MaritalStatus_Married']
```

Drop these columns from the scaled training and testing datasets.

In [92]:
```
scaled_train_df.drop(insignificant_cols, axis=1, inplace=True)
```

In [93]:
```
scaled_test_df.drop(insignificant_cols, axis=1, inplace=True)
```

6.3.2. Train the Model

In [94]:
```
train_features, train_target = get_features_and_target(scaled_train_df, 'Attrition_Yes')
```

In [95]:
```
m4_sm_model = statsmodel_logit(train_features, train_target)
```

```
Optimization terminated successfully.
         Current function value: 0.293821
         Iterations 8
```

In [96]:
```
m4_sm_model.prsquared
```

Out[96]:
```
0.32261357128047796
```

In [97]:
```
m4_sm_model.summary()
```

Out[97]:

Logit Regression Results

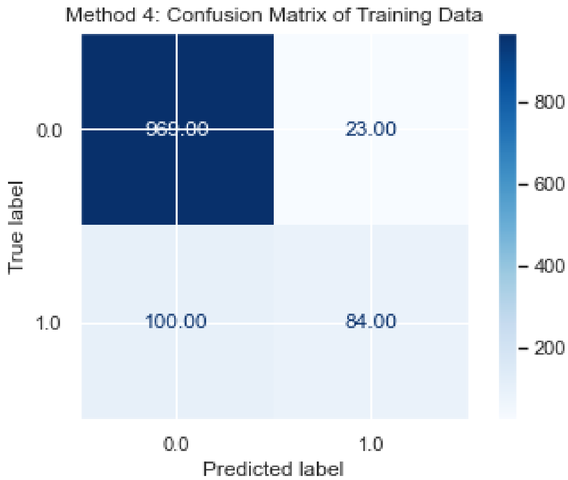
Dep. Variable:	Attrition_Yes	**No. Observations:**	1176
Model:	Logit	**Df Residuals:**	1153
Method:	MLE	**Df Model:**	22
Date:	Tue, 21 Apr 2020	**Pseudo R-squ.:**	0.3226
Time:	00:09:10	**Log-Likelihood:**	-345.53
converged:	True	**LL-Null:**	-510.10
Covariance Type:	nonrobust	**LLR p-value:**	1.449e-56

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Age	-0.2960	0.136	-2.173	0.030	-0.563	-0.029
DistanceFromHome	0.4012	0.095	4.233	0.000	0.215	0.587
EnvironmentSatisfaction	-0.5164	0.101	-5.132	0.000	-0.714	-0.319
JobInvolvement	-0.3778	0.096	-3.929	0.000	-0.566	-0.189
JobSatisfaction	-0.3786	0.100	-3.792	0.000	-0.574	-0.183
NumCompaniesWorked	0.5261	0.103	5.128	0.000	0.325	0.727
RelationshipSatisfaction	-0.2126	0.098	-2.163	0.031	-0.405	-0.020
TotalWorkingYears	-0.6428	0.210	-3.065	0.002	-1.054	-0.232
TrainingTimesLastYear	-0.3526	0.105	-3.355	0.001	-0.559	-0.147
WorkLifeBalance	-0.2740	0.097	-2.821	0.005	-0.464	-0.084
YearsAtCompany	0.6010	0.256	2.346	0.019	0.099	1.103
YearsInCurrentRole	-0.4914	0.184	-2.676	0.007	-0.851	-0.132
YearsSinceLastPromotion	0.5037	0.149	3.383	0.001	0.212	0.795
YearsWithCurrManager	-0.5145	0.183	-2.807	0.005	-0.874	-0.155
BusinessTravel_Travel_Frequently	0.7196	0.180	4.005	0.000	0.367	1.072
BusinessTravel_Travel_Rarely	0.5055	0.192	2.627	0.009	0.128	0.883
EducationField_Technical Degree	0.2639	0.089	2.978	0.003	0.090	0.438
JobRole_Laboratory Technician	0.2204	0.093	2.373	0.018	0.038	0.402
JobRole_Manufacturing Director	-0.2760	0.141	-1.952	0.051	-0.553	0.001
JobRole_Sales Representative	0.2448	0.083	2.962	0.003	0.083	0.407
MaritalStatus_Single	0.5097	0.096	5.313	0.000	0.322	0.698
OverTime_Yes	0.8408	0.095	8.808	0.000	0.654	1.028
const	-2.6231	0.147	-17.813	0.000	-2.912	-2.334

In [98]:
```
m4_sk_model = sklearn_logit(train_features, train_target)
```

```
Model Accuracy Score: 89.54081632653062%
```

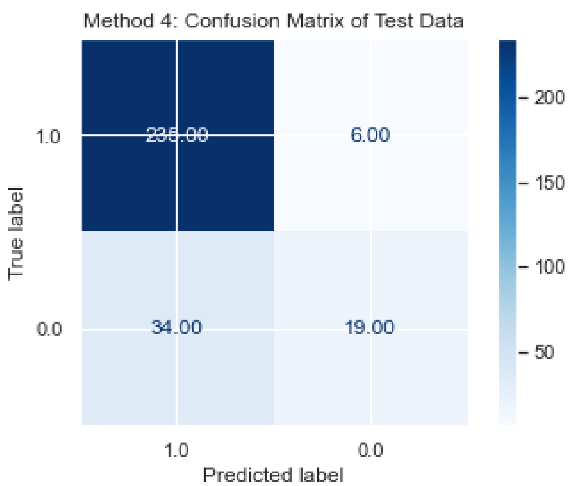
```
In [99]: sklearn_predict(m4_sk_model, train_features, train_target, title='Method 4: Confusion Matr
ix of Training Data')
```



6.3.3. Testing the Model

```
In [100]: test_features, test_target = get_features_and_target(scaled_test_df, 'Attrition_Yes')
```

```
In [101]: sklearn_predict(m4_sk_model, test_features, test_target, title='Method 4: Confusion Matrix
of Test Data')
```



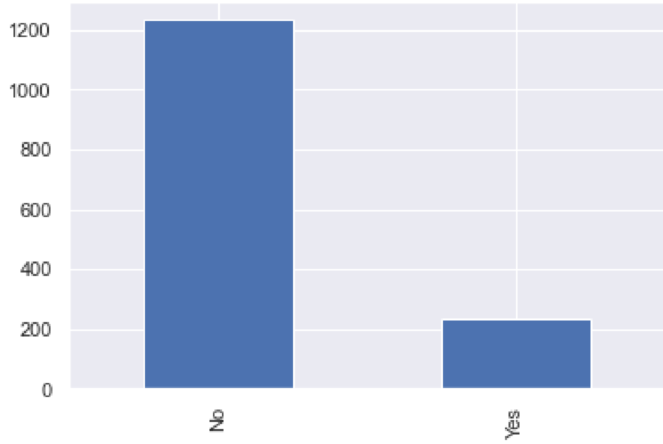
6.3.4. Conclusion

This model has the same accuracy of **89.54%** like the previous one. However, the Pseudo R-squared decreased to **0.3226**.

7. Logistic Regression on Oversampled Data

In the original dataset `hr_df` , the count of rows with `Attrition` value of *No* greatly outnumbers those with *Yes*. Those with *No* values are then the minority class.

In [102]:
```
ax = hr_df['Attrition'].value_counts().plot(kind='bar');
```



There is a great imbalance between the two values of the target variable. To solve this, I used SMOTE.

7.1. Synthetic Minority Oversampling Technique (SMOTE)

SMOTE selects a random sample from the minority class and finds that sample's k-nearest neighbors. It then chooses a random neighbor then generates synthetic data based on the combined random sample and the random neighbor.

Fit `SMOTE` on the `scaled_train_df` .

In [103]:
```
smote = SMOTE(random_state=18)
```

In [104]:
```
train_target = scaled_train_df['Attrition_Yes']
train_scaled_features = scaled_train_df.drop(['Attrition_Yes'], axis=1)
```

In [105]:
```
smote_train_feat, smote_train_target = smote.fit_sample(train_scaled_features, train_targe
t)
```

In [106]:
```
smote_train_feat.shape
```

Out[106]:
```
(1984, 23)
```

Both 1 and 0 values of `Attrition_Yes` now have the same number of rows/instances.

In [107]:
```
smote_train_target.value_counts()
```

Out[107]:
```
1.0    992
0.0    992
Name: Attrition_Yes, dtype: int64
```

7.2. Logistic Regression on the Balanced Model Created from SMOTE

7.2.1 Train the Model

In [108]:
```
train_features, train_target = get_features_and_target(smote_train_feat, smote_train_targe
t)
```

In [109]:
```
m5_sm_model = statsmodel_logit(train_features, train_target)
```

```
Optimization terminated successfully.
         Current function value: 0.447414
         Iterations 7
```

In [110]:
```
m5_sm_model.summary()
```

Out[110]:

Logit Regression Results

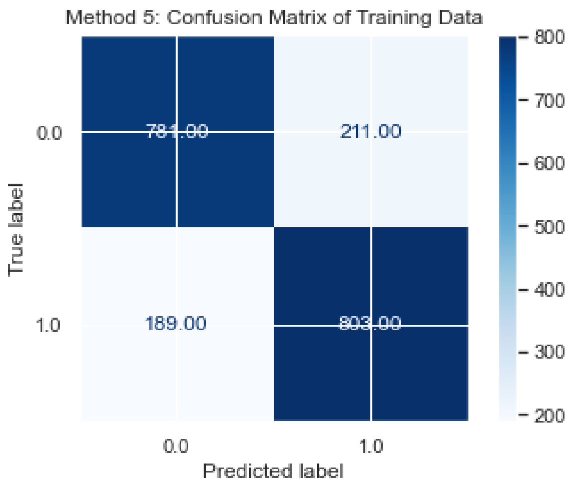
Dep. Variable:	Attrition_Yes	**No. Observations:**	1984
Model:	Logit	**Df Residuals:**	1961
Method:	MLE	**Df Model:**	22
Date:	Tue, 21 Apr 2020	**Pseudo R-squ.:**	0.3545
Time:	00:09:13	**Log-Likelihood:**	-887.67
converged:	True	**LL-Null:**	-1375.2
Covariance Type:	nonrobust	**LLR p-value:**	3.941e-192

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Age	-0.1643	0.085	-1.924	0.054	-0.332	0.003
DistanceFromHome	0.4250	0.062	6.815	0.000	0.303	0.547
EnvironmentSatisfaction	-0.5234	0.063	-8.364	0.000	-0.646	-0.401
JobInvolvement	-0.3436	0.063	-5.435	0.000	-0.468	-0.220
JobSatisfaction	-0.3701	0.064	-5.825	0.000	-0.495	-0.246
NumCompaniesWorked	0.5161	0.069	7.518	0.000	0.382	0.651
RelationshipSatisfaction	-0.1254	0.062	-2.035	0.042	-0.246	-0.005
TotalWorkingYears	-0.7765	0.133	-5.821	0.000	-1.038	-0.515
TrainingTimesLastYear	-0.4546	0.067	-6.760	0.000	-0.586	-0.323
WorkLifeBalance	-0.1919	0.062	-3.112	0.002	-0.313	-0.071
YearsAtCompany	0.6416	0.152	4.223	0.000	0.344	0.939
YearsInCurrentRole	-0.4597	0.117	-3.915	0.000	-0.690	-0.230
YearsSinceLastPromotion	0.4671	0.090	5.216	0.000	0.292	0.643
YearsWithCurrManager	-0.4277	0.109	-3.939	0.000	-0.640	-0.215
BusinessTravel_Travel_Frequently	0.8406	0.120	6.999	0.000	0.605	1.076
BusinessTravel_Travel_Rarely	0.7964	0.129	6.171	0.000	0.543	1.049
EducationField_Technical Degree	0.1889	0.060	3.173	0.002	0.072	0.306
JobRole_Laboratory Technician	0.1346	0.061	2.207	0.027	0.015	0.254
JobRole_Manufacturing Director	-0.3799	0.088	-4.315	0.000	-0.552	-0.207
JobRole_Sales Representative	0.2244	0.054	4.121	0.000	0.118	0.331
MaritalStatus_Single	0.4867	0.061	7.969	0.000	0.367	0.606
OverTime_Yes	0.8315	0.060	13.922	0.000	0.714	0.949
const	-0.9714	0.076	-12.795	0.000	-1.120	-0.823

In [111]:
```
m5_sk_model = sklearn_logit(train_features, train_target)
```

```
Model Accuracy Score: 79.83870967741935%
```

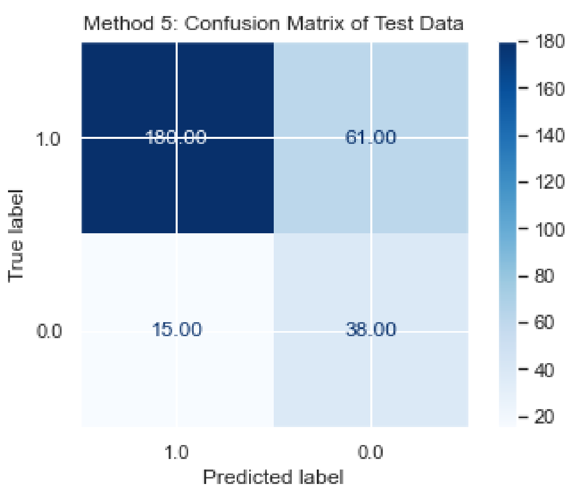
In [112]:
```
sklearn_predict(m5_sk_model, train_features, train_target, title='Method 5: Confusion Matr
ix of Training Data')
```



7.2.2. Testing the Model

In [113]:
```
test_features, test_target = get_features_and_target(scaled_test_df, 'Attrition_Yes')
```

In [114]:
```
sklearn_predict(m5_sk_model, test_features, test_target, title='Method 5: Confusion Matrix
of Test Data')
```



7.2.3. Conclusion

The accuracy model definitely decreased to **79.84%**. However, this model has increased its pseudo r-squared to **0.3545**.

8. Takeaways

The last model which was trained on feature-selected and oversampled data is the one I'm going to use. While it is less accurate than its previous versions, it is still accurate at almost 80%. In addition, this model is **more relevant because of its higher pseudo r-squared**.

While it might sound good to use the previous models (especially the one with 89.62% accuracy), there is a strong possibility that they are **overfitted**. Those models have barely undergone feature selection, and having too many features without regards to the sample size runs a risk of overfitting. If those models are used on unseen data, it will perform poorly.

For my next action, I will look at other ways to improve the last model or find another method of binary classification.