In [39]:
```
## Import Library ##
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

## For Data Encoding ##
from sklearn.preprocessing import LabelEncoder

## For Model Evaluation ##
from sklearn.model_selection import KFold

## Machine Learning Model ##
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression

## For Model Performance ##
from sklearn.metrics import accuracy_score, confusion_matrix, precision_s
```

In [62]:
```
## Model Performance
def evaluation(gt, pred):
    acc = accuracy_score(gt, pred)
    precision = precision_score(gt, pred)
    recall = recall_score(gt, pred)
    f1 = f1_score(gt, pred)
    matrix = confusion_matrix(gt, pred)

    return acc, precision, recall, f1, matrix
```

In [41]:
```
raw_data = pd.read_csv("./EmployeeAttrition.csv", index_col=0)
```

In [77]:
```
1 raw_data
```

Out[77]:

	Age	Attrition	BusinessTravel	Department	DistanceFromHome	Education	EducationField	En
0	41	1	Travel_Rarely	Sales	1	2	Life Sciences	
1	49	0	Travel_Frequently	Research & Development	8	1	Life Sciences	
2	37	1	Travel_Rarely	Research & Development	2	2	Other	
3	33	0	Travel_Frequently	Research & Development	3	4	Life Sciences	
4	27	0	Travel_Rarely	Research & Development	2	1	Medical	
...	...	...	...	...	...	...	...	
1465	36	0	Travel_Frequently	Research & Development	23	2	Medical	
1466	39	0	Travel_Rarely	Research & Development	6	1	Medical	
1467	27	0	Travel_Rarely	Research & Development	4	3	Life Sciences	
1468	49	0	Travel_Frequently	Sales	2	3	Medical	
1469	34	0	Travel_Rarely	Research & Development	8	3	Medical	

1470 rows × 27 columns

In [43]:
```
1 raw_data.columns
```

Out[43]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'Department', 'DistanceFromHom
e',
       'Education', 'EducationField', 'EmployeeNumber',
       'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
       'JobRole', 'JobSatisfaction', 'MaritalStatus', 'MonthlyIncome',
       'NumCompaniesWorked', 'PerformanceRating', 'RelationshipSatisfactio
n',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager'],
      dtype='object')
```

In [44]:
```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1470 entries, 0 to 1469
Data columns (total 27 columns):
Age                         1470 non-null int64
Attrition                   1470 non-null object
BusinessTravel              1470 non-null object
Department                  1470 non-null object
DistanceFromHome            1470 non-null int64
Education                   1470 non-null int64
EducationField              1470 non-null object
EmployeeNumber              1470 non-null int64
EnvironmentSatisfaction     1470 non-null int64
Gender                      1470 non-null object
JobInvolvement              1470 non-null int64
JobLevel                    1470 non-null int64
JobRole                     1470 non-null object
JobSatisfaction             1470 non-null int64
MaritalStatus               1470 non-null object
MonthlyIncome               1470 non-null int64
NumCompaniesWorked          1470 non-null int64
PerformanceRating           1470 non-null int64
RelationshipSatisfaction    1470 non-null int64
StockOptionLevel            1470 non-null int64
TotalWorkingYears           1470 non-null int64
TrainingTimesLastYear       1470 non-null int64
WorkLifeBalance             1470 non-null int64
YearsAtCompany              1470 non-null int64
YearsInCurrentRole          1470 non-null int64
YearsSinceLastPromotion     1470 non-null int64
YearsWithCurrManager        1470 non-null int64
dtypes: int64(20), object(7)
memory usage: 321.6+ KB
```

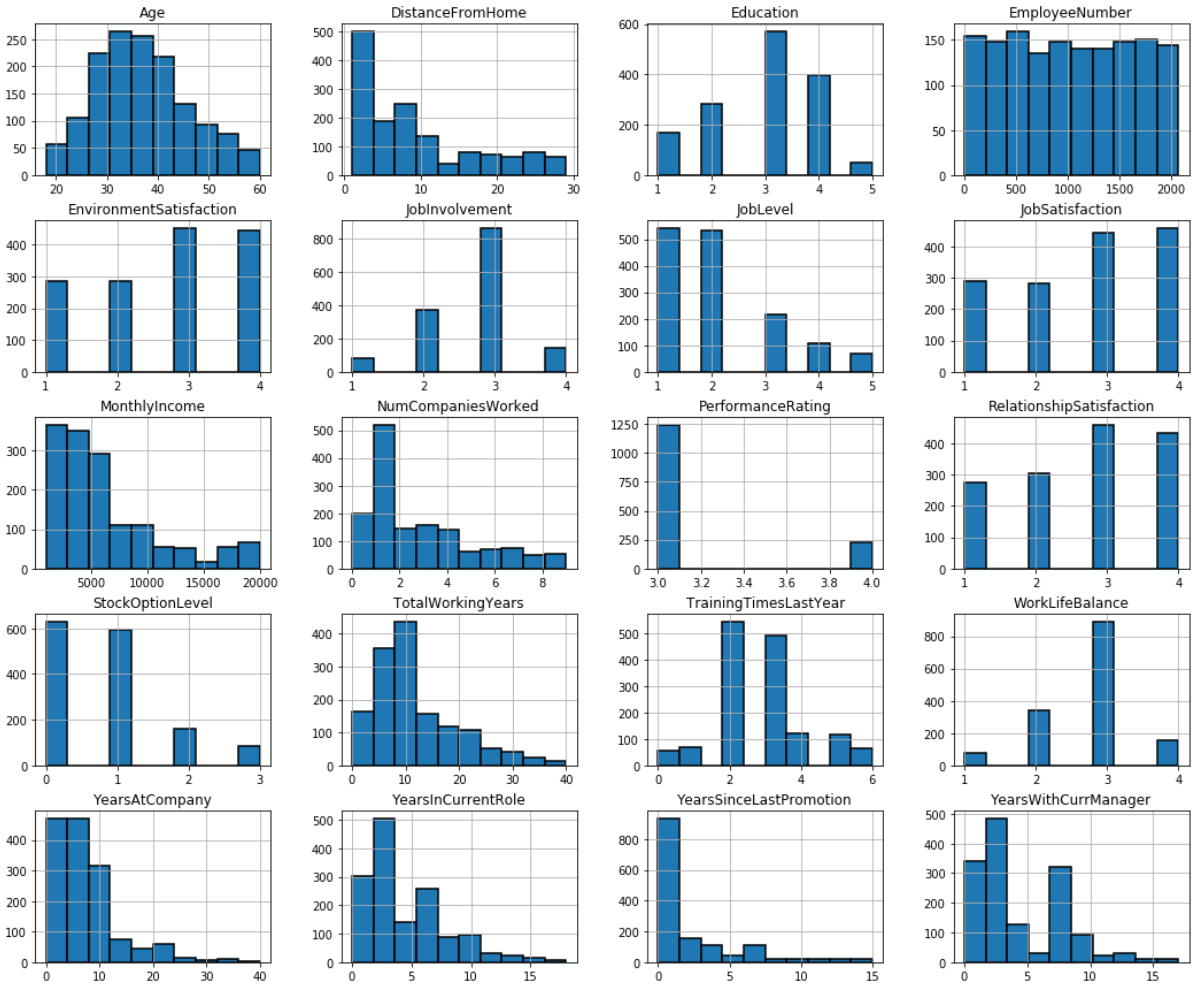
In [45]:
```
categorical_col = []
numeric_col = []
for col in raw_data.columns:
    if raw_data[col].dtype == object and col != "Attrition":
        categorical_col.append(col)
        print(col, raw_data[col].unique())
        print("=====================================================================
    elif raw_data[col].dtype == int and col != "Attrition":
        numeric_col.append(col)
```

```
BusinessTravel ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
======================================================================
Department ['Sales' 'Research & Development' 'Human Resources']
======================================================================
EducationField ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical De
gree'
 'Human Resources']
======================================================================
Gender ['Female' 'Male']
======================================================================
JobRole ['Sales Executive' 'Research Scientist' 'Laboratory Technician'
 'Manufacturing Director' 'Healthcare Representative' 'Manager'
 'Sales Representative' 'Research Director' 'Human Resources']
======================================================================
MaritalStatus ['Single' 'Married' 'Divorced']
======================================================================
```

In [46]:
```
## Encode Label
raw_data["Attrition"] = raw_data["Attrition"].astype("category").cat.code
raw_data["Attrition"].value_counts()
```

Out[46]:
```
0    1233
1     237
Name: Attrition, dtype: int64
```

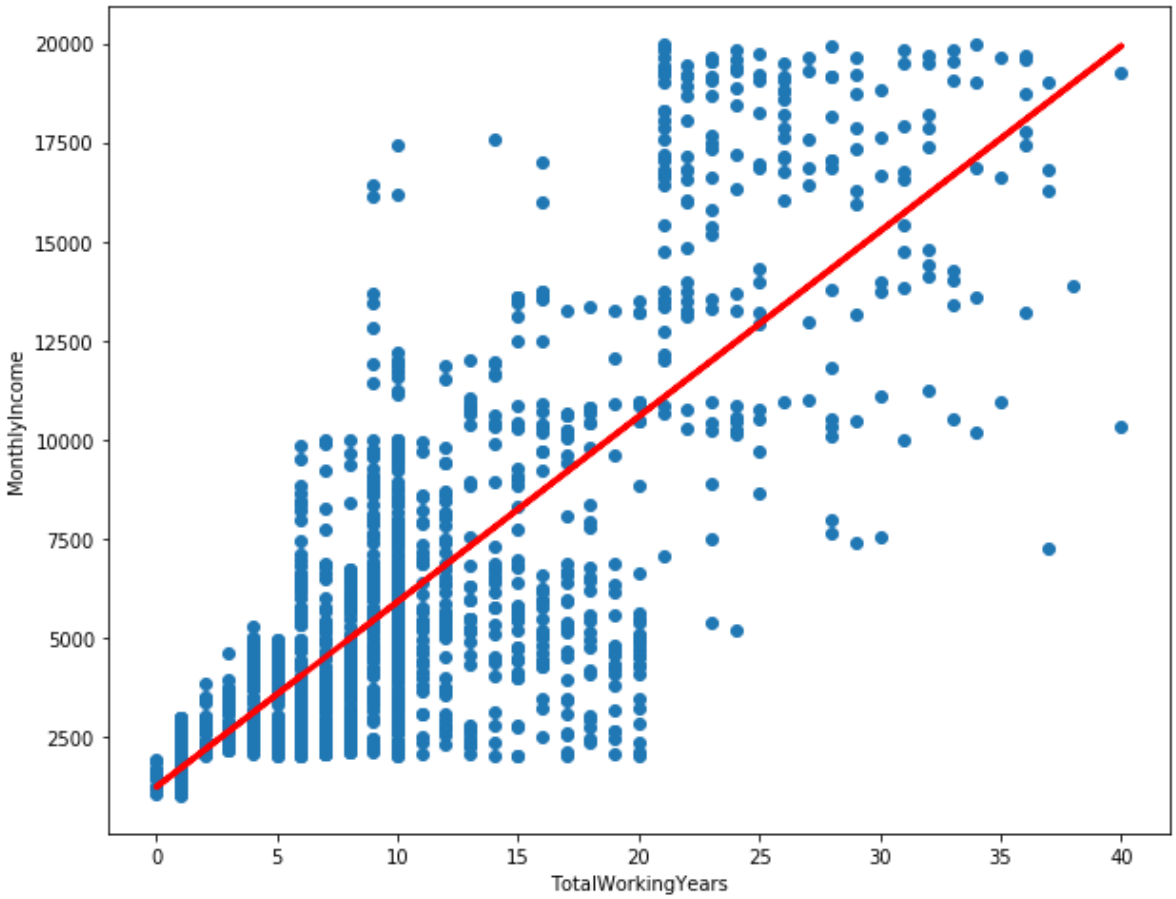
In [52]:
```
## Data Visualization for numerical data
raw_data[numeric_col].hist(edgecolor='black', linewidth=1.5, figsize=(18,
plt.show()
```



In [55]:
```
X = np.array(raw_data["TotalWorkingYears"]).reshape(len(raw_data), 1)
y = np.array(raw_data["MonthlyIncome"])
model = LinearRegression()
model = model.fit(X, y)
pred = model.predict(X)
print("y = %f * X + %f" % (model.coef_[0], model.intercept_))
```

```
y = 467.658412 * X + 1227.935288
```

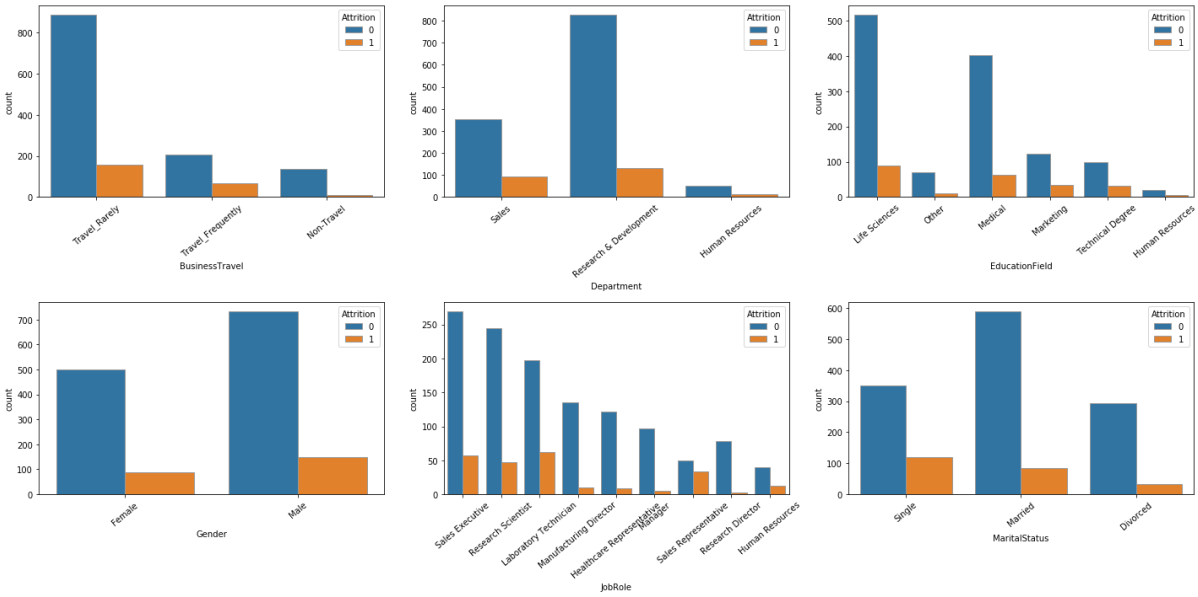
In [56]:
```
plt.figure(figsize=(10, 8))
plt.scatter(raw_data["TotalWorkingYears"], raw_data["MonthlyIncome"])
plt.plot(raw_data["TotalWorkingYears"], pred, color="red", linewidth = 3)
plt.xlabel("TotalWorkingYears")
plt.ylabel("MonthlyIncome")
plt.show()
```



In [50]:
```
categorical_col
```

Out[50]:
```
['BusinessTravel',
 'Department',
 'EducationField',
 'Gender',
 'JobRole',
 'MaritalStatus']
```

In [57]:
```
## Data Visualization for categorical data
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for index, col in enumerate(categorical_col):
    row_num = int(index / 3)
    col_num = (index % 3)
    label = list(raw_data[col].unique())
    sns.countplot(x=str(col), hue="Attrition", edgecolor=".6", data=raw_d
    axes[row_num, col_num].set_xticklabels(label, rotation=40)
plt.tight_layout()
plt.show()
```



In [58]:
```
## Data Encoding (one-hot encoding)
one_hot_encoding_df = pd.get_dummies(raw_data, columns=categorical_col)
```

In [16]:
```
categorical_col
```

Out[16]:
```
['BusinessTravel',
 'Department',
 'EducationField',
 'Gender',
 'JobRole',
 'MaritalStatus']
```

In [60]:
```
1 set(one_hot_encoding_df.columns) - set(numeric_col)
```

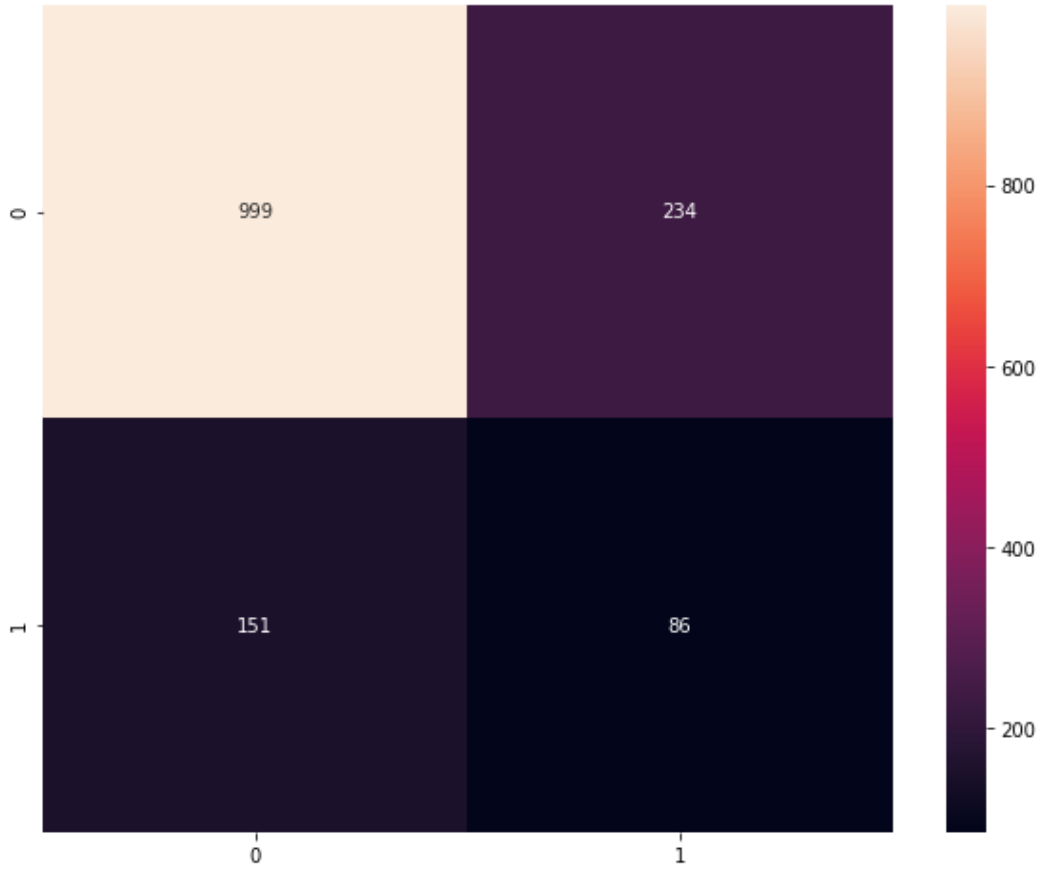
Out[60]:
```
{'Attrition',
 'BusinessTravel_Non-Travel',
 'BusinessTravel_Travel_Frequently',
 'BusinessTravel_Travel_Rarely',
 'Department_Human Resources',
 'Department_Research & Development',
 'Department_Sales',
 'EducationField_Human Resources',
 'EducationField_Life Sciences',
 'EducationField_Marketing',
 'EducationField_Medical',
 'EducationField_Other',
 'EducationField_Technical Degree',
 'Gender_Female',
 'Gender_Male',
 'JobRole_Healthcare Representative',
 'JobRole_Human Resources',
 'JobRole_Laboratory Technician',
 'JobRole_Manager',
 'JobRole_Manufacturing Director',
 'JobRole_Research Director',
 'JobRole_Research Scientist',
 'JobRole_Sales Executive',
 'JobRole_Sales Representative',
 'MaritalStatus_Divorced',
 'MaritalStatus_Married',
 'MaritalStatus_Single'}
```

In [66]:

```
## Data Splitting and Model Learning (Decision Tree)
avg_acc = 0
avg_precision = 0
avg_recall = 0
avg_f1 = 0
avg_confusion_matrix = []
avg_feature_importance = []

kf = KFold(n_splits=5)
fold_count = 0
for train_index, test_index in kf.split(one_hot_encoding_df):
    print("Training Data: %d, Testing Data: %d" % (len(train_index), len(
    train_X = one_hot_encoding_df.loc[train_index, one_hot_encoding_df.co
    train_y = one_hot_encoding_df.loc[train_index]["Attrition"]
    test_X = one_hot_encoding_df.loc[test_index, one_hot_encoding_df.colu
    test_y = one_hot_encoding_df.loc[test_index]["Attrition"]

    model = DecisionTreeClassifier(random_state=200)
    model = model.fit(train_X, train_y)
    test_predict = model.predict(test_X)
    avg_feature_importance.append(model.feature_importances_)

    acc, precision, recall, f1, matrix = evaluation(test_y, test_predict)

    print("Fold: %d, Accuracy: %f, Precision: %f, Recall: %f, F1: %f" % (
    avg_acc += acc
    avg_precision += precision
    avg_recall += recall
    avg_f1 += f1
    avg_confusion_matrix.append(matrix)
    fold_count += 1

print("================================================================
print("Avg Accuracy: %f, Avg Precision: %f, Avg Recall: %f, Avg F1: %f" %


```

```
Training Data: 1176, Testing Data: 294
Fold: 1, Accuracy: 0.752000, Precision: 0.268000, Recall: 0.319000, F1: 0.2
91000
Training Data: 1176, Testing Data: 294
Fold: 2, Accuracy: 0.789000, Precision: 0.333000, Recall: 0.442000, F1: 0.3
80000
Training Data: 1176, Testing Data: 294
Fold: 3, Accuracy: 0.741000, Precision: 0.311000, Recall: 0.237000, F1: 0.2
69000
Training Data: 1176, Testing Data: 294
Fold: 4, Accuracy: 0.738000, Precision: 0.233000, Recall: 0.311000, F1: 0.2
67000
Training Data: 1176, Testing Data: 294
Fold: 5, Accuracy: 0.670000, Precision: 0.235000, Recall: 0.558000, F1: 0.3
31000
================================================================================
======
Avg Accuracy: 0.738000, Avg Precision: 0.276000, Avg Recall: 0.374000, Avg
F1: 0.308000
```

In [67]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(np.sum(np.array(avg_confusion_matrix), axis=0), annot=True, f
plt.show()
```



In [68]:
```
importance_dict = {}
for col, importance in zip(train_X.columns, np.mean(np.array(avg_feature_
    importance_dict[col] = importance

sorted(importance_dict.items(), key=lambda x: -x[1])[:10]
```

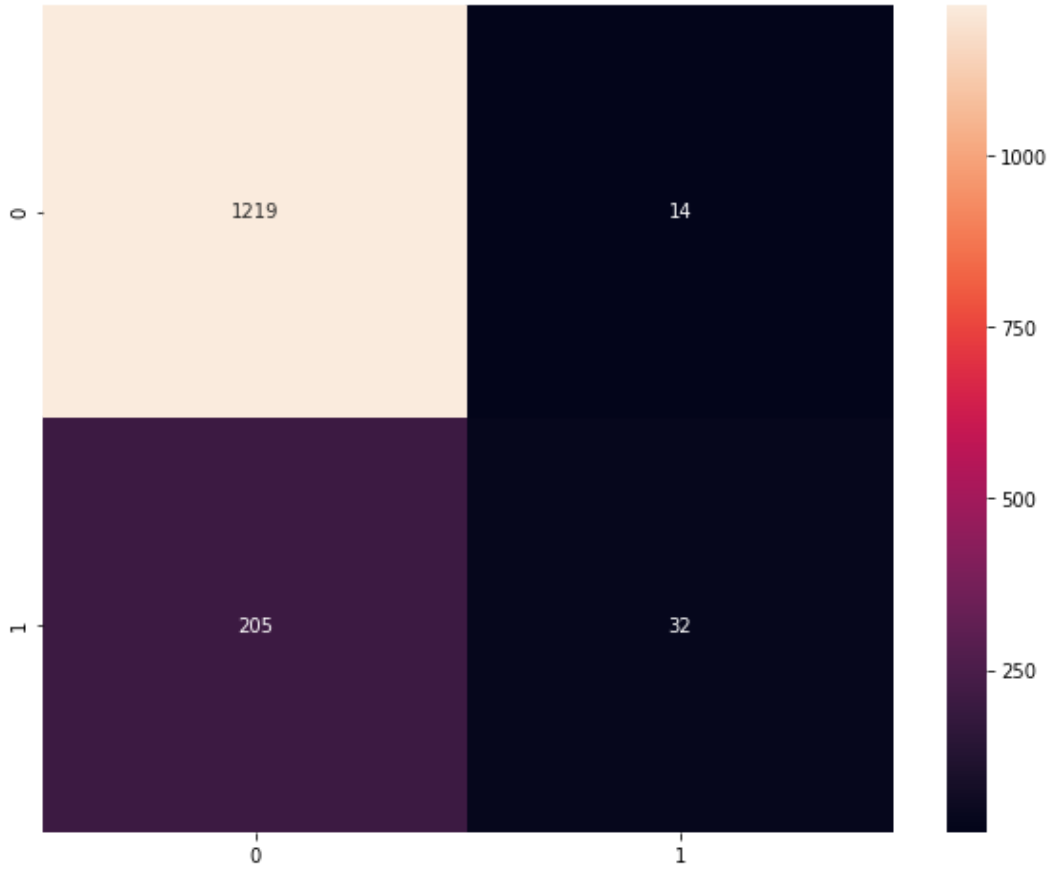
Out[68]:
```
[('MonthlyIncome', 0.10020746536225093),
 ('TotalWorkingYears', 0.08286263996666518),
 ('Age', 0.07866270282338719),
 ('DistanceFromHome', 0.07651699500507207),
 ('EmployeeNumber', 0.06251880620094238),
 ('NumCompaniesWorked', 0.054148502062084915),
 ('TrainingTimesLastYear', 0.04497546911268388),
 ('JobInvolvement', 0.03828324416180727),
 ('StockOptionLevel', 0.035453500524868876),
 ('JobSatisfaction', 0.03481479445074119)]
```

In [69]:

```
## Data Splitting and Model Learning (Random Forest)
avg_acc = 0
avg_precision = 0
avg_recall = 0
avg_f1 = 0
avg_confusion_matrix = []
avg_feature_importance = []

kf = KFold(n_splits=5)
fold_count = 0
for train_index, test_index in kf.split(one_hot_encoding_df):
    print("Training Data: %d, Testing Data: %d" % (len(train_index), len(
    train_X = one_hot_encoding_df.loc[train_index, one_hot_encoding_df.co
    train_y = one_hot_encoding_df.loc[train_index]["Attrition"]
    test_X = one_hot_encoding_df.loc[test_index, one_hot_encoding_df.colu
    test_y = one_hot_encoding_df.loc[test_index]["Attrition"]

    model = RandomForestClassifier(n_estimators=300)
    model = model.fit(train_X, train_y)
    test_predict = model.predict(test_X)

    avg_feature_importance.append(model.feature_importances_)

    acc, precision, recall, f1, matrix = evaluation(test_y, test_predict)
    print("Fold: %d, Accuracy: %f, Precision: %f, Recall: %f, F1: %f" % (
    avg_acc += acc
    avg_precision += precision
    avg_recall += recall
    avg_f1 += f1
    avg_confusion_matrix.append(matrix)
    fold_count += 1

print("================================================================
print("Avg Accuracy: %f, Avg Precision: %f, Avg Recall: %f, Avg F1: %f" %



```

```
Training Data: 1176, Testing Data: 294
Fold: 1, Accuracy: 0.847000, Precision: 0.625000, Recall: 0.106000, F1: 0.1
82000
Training Data: 1176, Testing Data: 294
Fold: 2, Accuracy: 0.871000, Precision: 0.727000, Recall: 0.186000, F1: 0.2
96000
Training Data: 1176, Testing Data: 294
Fold: 3, Accuracy: 0.806000, Precision: 0.667000, Recall: 0.068000, F1: 0.1
23000
Training Data: 1176, Testing Data: 294
Fold: 4, Accuracy: 0.861000, Precision: 0.667000, Recall: 0.178000, F1: 0.2
81000
Training Data: 1176, Testing Data: 294
Fold: 5, Accuracy: 0.871000, Precision: 0.778000, Recall: 0.163000, F1: 0.2
69000
================================================================
======
Avg Accuracy: 0.851000, Avg Precision: 0.693000, Avg Recall: 0.140000, Avg
F1: 0.230000
```

In [72]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(np.sum(np.array(avg_confusion_matrix), axis=0), annot=True, f
plt.show()
```



In [73]:
```
importance_dict = {}
for col, importance in zip(train_X.columns, np.mean(np.array(avg_feature_
    importance_dict[col] = importance

sorted(importance_dict.items(), key=lambda x: -x[1])[:10]
```

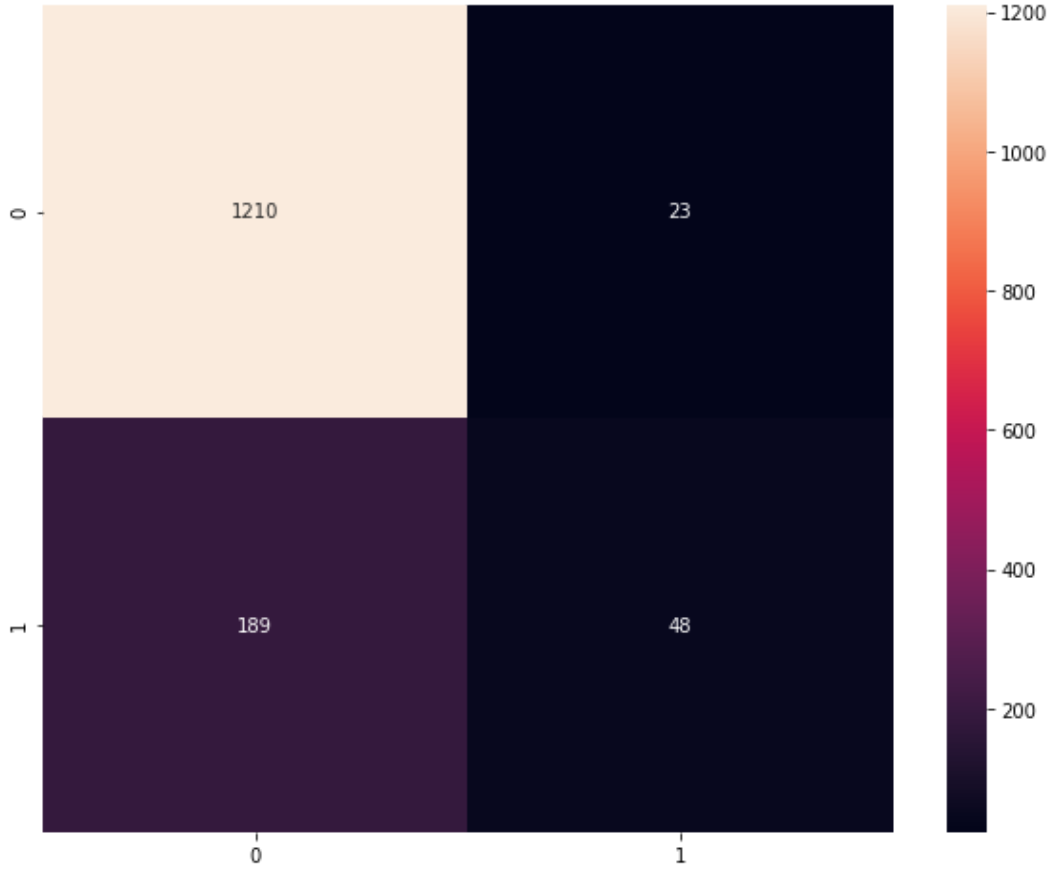
Out[73]:
```
[('MonthlyIncome', 0.08101629049886984),
 ('Age', 0.0716518888272812),
 ('EmployeeNumber', 0.06303118631947577),
 ('TotalWorkingYears', 0.057268848061669896),
 ('DistanceFromHome', 0.0561689074827289),
 ('YearsAtCompany', 0.0471668933929604),
 ('NumCompaniesWorked', 0.03970843810345892),
 ('YearsWithCurrManager', 0.03646828928980059),
 ('EnvironmentSatisfaction', 0.03451136743068421),
 ('YearsInCurrentRole', 0.03261817566506732)]
```

In [74]:

```
## Data Splitting and Model Learning (Logistic Regression)
avg_acc = 0
avg_precision = 0
avg_recall = 0
avg_f1 = 0
avg_feature_importance = []
avg_confusion_matrix = []

kf = KFold(n_splits=5)
fold_count = 0
for train_index, test_index in kf.split(one_hot_encoding_df):
    print("Training Data: %d, Testing Data: %d" % (len(train_index), len(
    train_X = one_hot_encoding_df.loc[train_index, one_hot_encoding_df.co
    train_y = one_hot_encoding_df.loc[train_index]["Attrition"]
    test_X = one_hot_encoding_df.loc[test_index, one_hot_encoding_df.colu
    test_y = one_hot_encoding_df.loc[test_index]["Attrition"]

    model = LogisticRegression(solver='liblinear')
    model = model.fit(train_X, train_y)
    test_predict = model.predict(test_X)

    acc, precision, recall, f1, matrix = evaluation(test_y, test_predict)
    print("Fold: %d, Accuracy: %f, Precision: %f, Recall: %f, F1: %f" % (
    avg_acc += acc
    avg_precision += precision
    avg_recall += recall
    avg_f1 += f1
    avg_confusion_matrix.append(matrix)
    fold_count += 1

print("================================================================
print("Avg Accuracy: %f, Avg Precision: %f, Avg Recall: %f, Avg F1: %f" %



```

```
Training Data: 1176, Testing Data: 294
Fold: 1, Accuracy: 0.854000, Precision: 0.833000, Recall: 0.106000, F1: 0.1
89000
Training Data: 1176, Testing Data: 294
Fold: 2, Accuracy: 0.861000, Precision: 0.562000, Recall: 0.209000, F1: 0.3
05000
Training Data: 1176, Testing Data: 294
Fold: 3, Accuracy: 0.813000, Precision: 0.611000, Recall: 0.186000, F1: 0.2
86000
Training Data: 1176, Testing Data: 294
Fold: 4, Accuracy: 0.871000, Precision: 0.733000, Recall: 0.244000, F1: 0.3
67000
Training Data: 1176, Testing Data: 294
Fold: 5, Accuracy: 0.881000, Precision: 0.750000, Recall: 0.279000, F1: 0.4
07000
================================================================================
======
Avg Accuracy: 0.856000, Avg Precision: 0.698000, Avg Recall: 0.205000, Avg
F1: 0.311000
```

In [76]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(np.sum(np.array(avg_confusion_matrix), axis=0), annot=True, f
plt.show()
```



In [ ]:
```
```