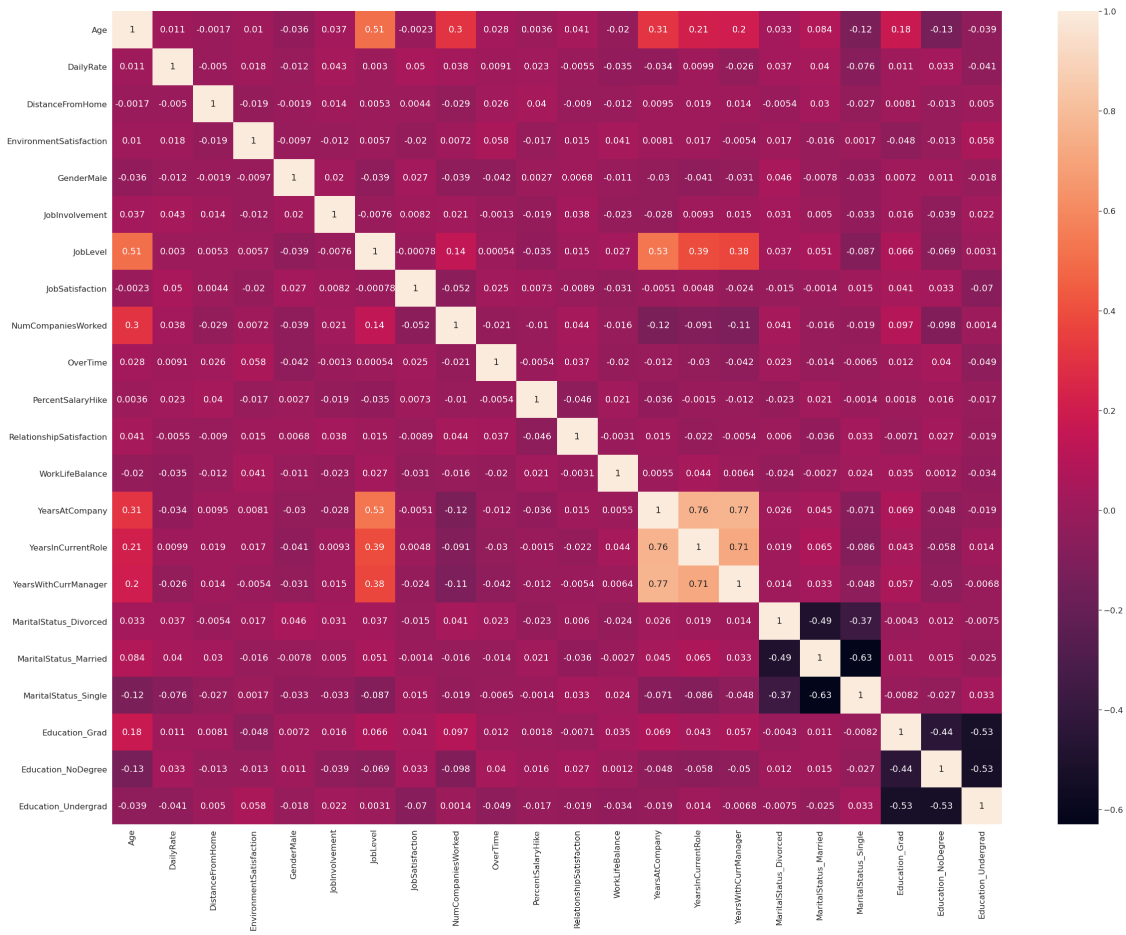
```
[146]: data_corr = data.drop(['Attrition'],axis=1)
       corrMatrix = data_corr.corr()
       sn.set(rc={'figure.figsize':(40,30)})
       sn.set(font_scale=1.5)
       sn.heatmap(corrMatrix, annot=True)
       plt.show()
```



The cut-off for 'strongly correlated' is usually a score of 0.70 or above. We notice that there are a few moderately correlated features (0.50 or above), but we will not remove them from our predictor set now. Let's remove our strongly correlated features from our predictor set.

```
[147]: data = data.
         ↪drop(columns=['YearsAtCompany','YearsInCurrentRole','YearsWithCurrManager'])
```

Another approach to reducing collinearity and unwanted variance is to use a variance inflation factor calculation, or VIF score, to evaluate factors that contribute the most to model variance.

```
[148]: data_corr = data.drop(['Attrition'],axis=1)
       vif = pd.DataFrame()
```

```
vif["features"] = data_corr.columns
vif["vif_Factor"] = [variance_inflation_factor(data_corr.values, i) for i in␣
 ↪range(data_corr.shape[1])]
vif
```

[148]:

	features	vif_Factor
0	Age	1.511601
1	DailyRate	1.015848
2	DistanceFromHome	1.005338
3	EnvironmentSatisfaction	1.011806
4	GenderMale	1.009569
5	JobInvolvement	1.009735
6	JobLevel	1.361918
7	JobSatisfaction	1.013570
8	NumCompaniesWorked	1.115367
9	OverTime	1.013752
10	PercentSalaryHike	1.008354
11	RelationshipSatisfaction	1.011094
12	WorkLifeBalance	1.011037
13	MaritalStatus_Divorced	inf
14	MaritalStatus_Married	inf
15	MaritalStatus_Single	inf
16	Education_Grad	inf
17	Education_NoDegree	inf
18	Education_Undergrad	inf

We will eliminate variables with a VIF score greater than 10.

VIF scores of INF indicate perfect collinearity - we will need to eliminate further variables in order to resolve this issue.

[149]:
```
data = data.
 ↪drop(columns=['Education_Grad','Education_NoDegree','Education_Undergrad','MaritalStatus_Di
```

[150]:
```
data_corr = data.drop(['Attrition'],axis=1)
vif = pd.DataFrame()
vif["features"] = data_corr.columns
vif["vif_Factor"] = [variance_inflation_factor(data_corr.values, i) for i in␣
 ↪range(data_corr.shape[1])]
vif
```

[150]:

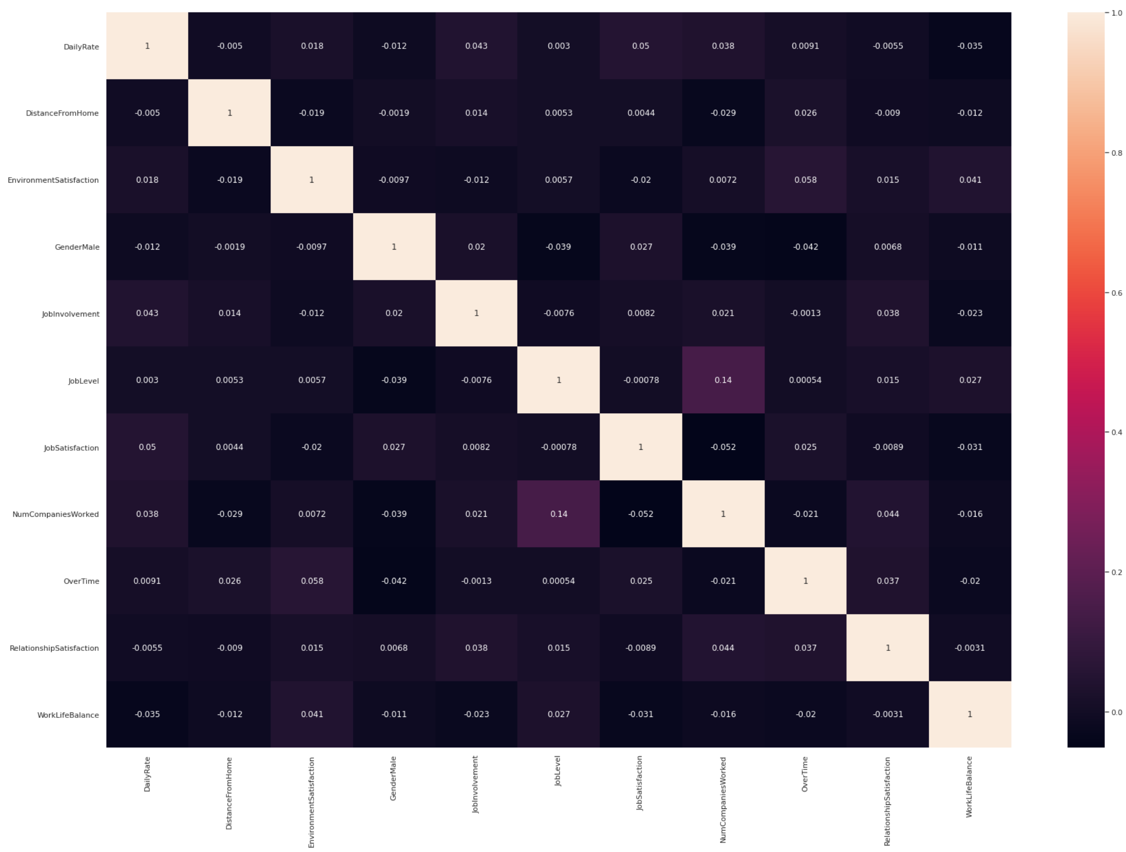
	features	vif_Factor
0	Age	19.783245
1	DailyRate	4.676489
2	DistanceFromHome	2.249682
3	EnvironmentSatisfaction	2.517029
4	GenderMale	2.422625
5	JobInvolvement	3.124185

```
6                     JobLevel    6.085303
7              JobSatisfaction    2.534030
8           NumCompaniesWorked    2.397062
9                     OverTime    1.403532
10          PercentSalaryHike    12.221724
11    RelationshipSatisfaction    2.484607
12             WorkLifeBalance    3.297188
```

As we've removed variables, we can see that there's additional variables that have poor VIF scores now. Let's eliminate those.

[151]:
```python
data = data.drop(columns=['Age','PercentSalaryHike'])
```

[152]:
```python
data_corr = data.drop(['Attrition'],axis=1)
corrMatrix = data_corr.corr()
sn.set(rc={'figure.figsize':(30,20)})
sn.set(font_scale=1)
sn.heatmap(corrMatrix, annot=True)
plt.show()
```



We can see clearly now that we have reduced collinearity significantly within our predictors.

[153]:
```
data_corr = data.drop(['Attrition'],axis=1)
vif = pd.DataFrame()
vif["features"] = data_corr.columns
vif["vif_Factor"] = [variance_inflation_factor(data_corr.values, i) for i in␣
↪range(data_corr.shape[1])]
vif
```

[153]:

	features	vif_Factor
0	DailyRate	4.161205
1	DistanceFromHome	2.143422
2	EnvironmentSatisfaction	2.427644
3	GenderMale	2.305405
4	JobInvolvement	2.947651
5	JobLevel	3.956010
6	JobSatisfaction	2.420726
7	NumCompaniesWorked	2.153264
8	OverTime	1.387235
9	RelationshipSatisfaction	2.403458
10	WorkLifeBalance	3.033991

Now that we've eliminated variables that contribute to collinearity, we are ready to start selecting variables and fitting a model.

Feature Selection & Modeling

Logistic Regression Model

Principle Component Analysis is an approach that can help us reduce dimensionality, and help us understand what proportion of the variance we are capturing.

Let's start by creating a logistic regression model, to see whether we're able to create a working model given the predictors we curently have, or if there are only a few significant factors we should include in our final model.

[154]:
```
y = data['Attrition']
x = data.drop('Attrition',axis=1)
print('Available Features',x.columns)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.20,␣
↪shuffle=True, random_state=2)
```

```
Available Features Index(['DailyRate', 'DistanceFromHome',
'EnvironmentSatisfaction',
       'GenderMale', 'JobInvolvement', 'JobLevel', 'JobSatisfaction',
       'NumCompaniesWorked', 'OverTime', 'RelationshipSatisfaction',
       'WorkLifeBalance'],
      dtype='object')
```

[155]:
```python
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(max_iter = 2500)
clf.fit(x_train,y_train)
clf.fit(x_train,y_train)
print('Coefficients and Intercept Value',clf.coef_,clf.intercept_)
y_pred = clf.predict(x_test)
y_true = y_test
```

```
Coefficients and Intercept Value [[-4.86883193e-04  3.12415667e-02
-6.59671371e-01  3.47610215e-01
  -6.84516221e-01 -5.21127560e-01 -5.15016658e-01  8.49435641e-02
   1.50266044e+00 -1.86046123e-01 -4.18659682e-01]] [-0.11419202]
```

These are the coefficients and intercept for our model. Unfortunately, we are not able to retrieve any information from our model about the p-values or statistical significance of any of our coefficients or the intercept. However, we can use this information to discuss the relevance of some of our features to the outcome. We might use the coefficients to understand how a one unit change in the predictor impacts the log-likelihood. For example, our first predictor is DailyRate. We can say that in this model, a one unit change in DailyRate corresponds to a decrease in the log likelihood of attrition by 0.0004. Is this enough for us to come to our business leaders and share the relationship between these variables and our outcome? Unfortunately no, as we do not know the statistical significance of any of these features, and whether there are additional interaction effects that we'd like to consider or other contraints.

In order to get some information about our coefficients and features, let's use a statsmodels implementation of the logistic regression model, to see if we can get any additional information.

[156]:
```python
from statsmodels.discrete.discrete_model import Logit
from statsmodels.tools import add_constant
x_train2 = add_constant(x_train)
print(Logit(y_train, x_train2).fit().summary())
```

```
Optimization terminated successfully.
         Current function value: 0.368265
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:              Attrition   No. Observations:                 1176
Model:                          Logit   Df Residuals:                     1164
Method:                           MLE   Df Model:                           11
Date:                Mon, 02 May 2022   Pseudo R-squ.:                  0.1647
Time:                        21:05:32   Log-Likelihood:                -433.08
converged:                       True   LL-Null:                       -518.44
Covariance Type:            nonrobust   LLR p-value:                 8.362e-31
================================================================================
============
                               coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
```

```
------------
const                          0.0219      0.387      0.057      0.955      -0.737
0.781
DailyRate                     -0.0005      0.000     -2.404      0.016      -0.001
-9.5e-05
DistanceFromHome               0.0309      0.010      2.963      0.003       0.010
0.051
EnvironmentSatisfaction       -0.6919      0.175     -3.958      0.000      -1.035
-0.349
GenderMale                     0.3414      0.184      1.854      0.064      -0.020
0.702
JobInvolvement                -0.7220      0.180     -4.008      0.000      -1.075
-0.369
JobLevel                      -0.5404      0.099     -5.469      0.000      -0.734
-0.347
JobSatisfaction               -0.5476      0.175     -3.128      0.002      -0.891
-0.205
NumCompaniesWorked             0.0845      0.034      2.471      0.013       0.017
0.152
OverTime                       1.5539      0.178      8.719      0.000       1.205
1.903
RelationshipSatisfaction      -0.1984      0.178     -1.113      0.266      -0.548
0.151
WorkLifeBalance               -0.4576      0.183     -2.503      0.012      -0.816
-0.099
================================================================================
============
```

We can see here that our model does not take into account interaction effects, and does not match the model created by our other method. This is to be expected, as we're using different methods. If we investigate the p-values, we notice that there are a few significant factors. DailyRate,DistanceFromHome,EnvironmentSatisfaction,JobInvolvement,JobLevel,JobSatisfaction,NumCompaniesWo and OverTime are significant. Let's rebuild the model, only including these factors.

[157]:
```python
x_train2 = add_constant(x_train)
x_train2 = x_train2.
 ↪drop(columns=['GenderMale','RelationshipSatisfaction','WorkLifeBalance'])
model = Logit(y_train, x_train2).fit()
print(model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.372889
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:              Attrition   No. Observations:                 1176
Model:                          Logit   Df Residuals:                     1167
Method:                           MLE   Df Model:                            8
```

```
Date:                Mon, 02 May 2022   Pseudo R-squ.:                  0.1542
Time:                        21:05:35   Log-Likelihood:                -438.52
converged:                       True   LL-Null:                       -518.44
Covariance Type:            nonrobust   LLR p-value:                 1.719e-30
================================================================================
===========
                               coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
const                       -0.2038      0.327     -0.623      0.533      -0.845
0.437
DailyRate                   -0.0005      0.000     -2.209      0.027      -0.001
-5.27e-05
DistanceFromHome             0.0316      0.010      3.054      0.002       0.011
0.052
EnvironmentSatisfaction     -0.7141      0.174     -4.114      0.000      -1.054
-0.374
JobInvolvement              -0.6997      0.179     -3.919      0.000      -1.050
-0.350
JobLevel                    -0.5554      0.099     -5.619      0.000      -0.749
-0.362
JobSatisfaction             -0.5137      0.174     -2.959      0.003      -0.854
-0.173
NumCompaniesWorked           0.0811      0.034      2.389      0.017       0.015
0.148
OverTime                     1.5295      0.176      8.684      0.000       1.184
1.875
================================================================================
===========
```

We see that our coefficients remained significant, but no improvement in our intercept. The confidence interval contains 0, which tells us that when x = 0, the log odds of having attrition as an outcome are likely 0. The two models are performing similarly, which means that perhaps we can drop the other variables from our analysis.

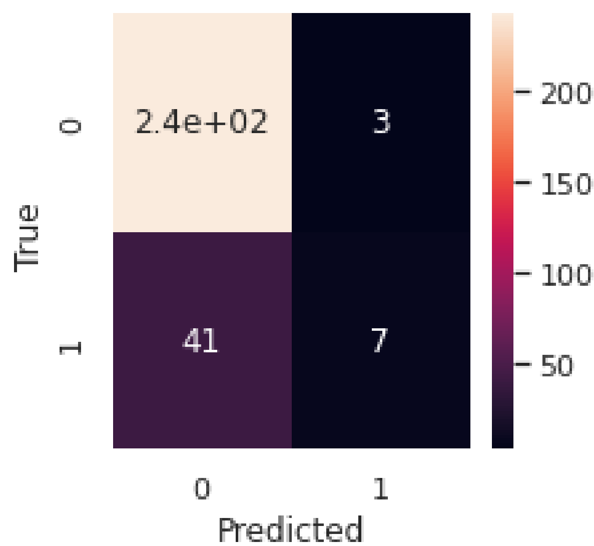
However, it could also mean that there are interaction effects we are missing. Our sklearn model does more to incorporate polynomial and interaction effects. Let's see how our sklearn model performed, so we know if we're able to build a logistic regression model that performs sufficiently well. If we are not able to build a logistic regression model that performs sufficiently well, then perhaps we are using the wrong model, which might suggest that the decision boundary is very non-linear and flexible.

[158]:
```
import numpy as np
from sklearn.metrics import accuracy_score
print("Training accuracy:")
print(np.round(accuracy_score(y_train,clf.predict(x_train)),2))
print("Test accuracy:")
print(np.round(accuracy_score(y_true,y_pred),2))
```

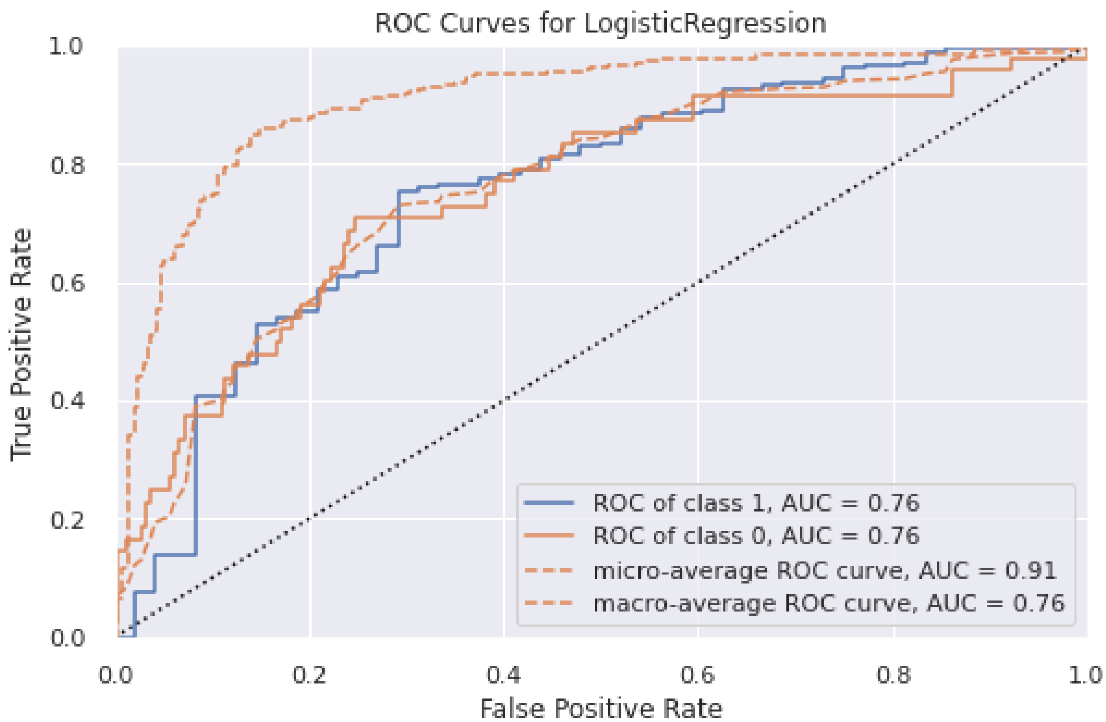
```
from sklearn.metrics import confusion_matrix
sn.set(rc={'figure.figsize':(3,3)})
sn.set(font_scale=1)
matrix = confusion_matrix(y_true,y_pred)
sn.heatmap(matrix,annot=True)
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
Training accuracy:
0.86
Test accuracy:
0.85
```

[158]: Text(3.5, 0.5, 'True')



[159]:
```
from yellowbrick.classifier import ROCAUC
sn.set(rc={'figure.figsize':(8,5)})
visualizer = ROCAUC(clf,classes=[1,0])
visualizer.fit(x_train.values, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
```



[159]:
```
<AxesSubplot:title={'center':'ROC Curves for LogisticRegression'}, xlabel='False
Positive Rate', ylabel='True Positive Rate'>
```

When looking at our ROC curve, we want to investigate our micro-average ROC value, since we have unbalanced class sizes. This gives us a great ROC score for our model.

So we're clearly able to create a (relatively) accurate model using a logistic regression approach that incorporates all of our current predictors. We can see that we have a relatively low false positive rate, but a relatively higher false negative rate. Let's see if principle component analysis can help us reduce the dimensionality, and potentially improve our model's performance.

Principal Component Analysis

[160]:
```
from sklearn.preprocessing import StandardScaler

y = data['Attrition']
x = data.drop('Attrition',axis=1)

scaler = StandardScaler()
x = scaler.fit_transform(x)
```

[161]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = None)
pca.fit(x)
```

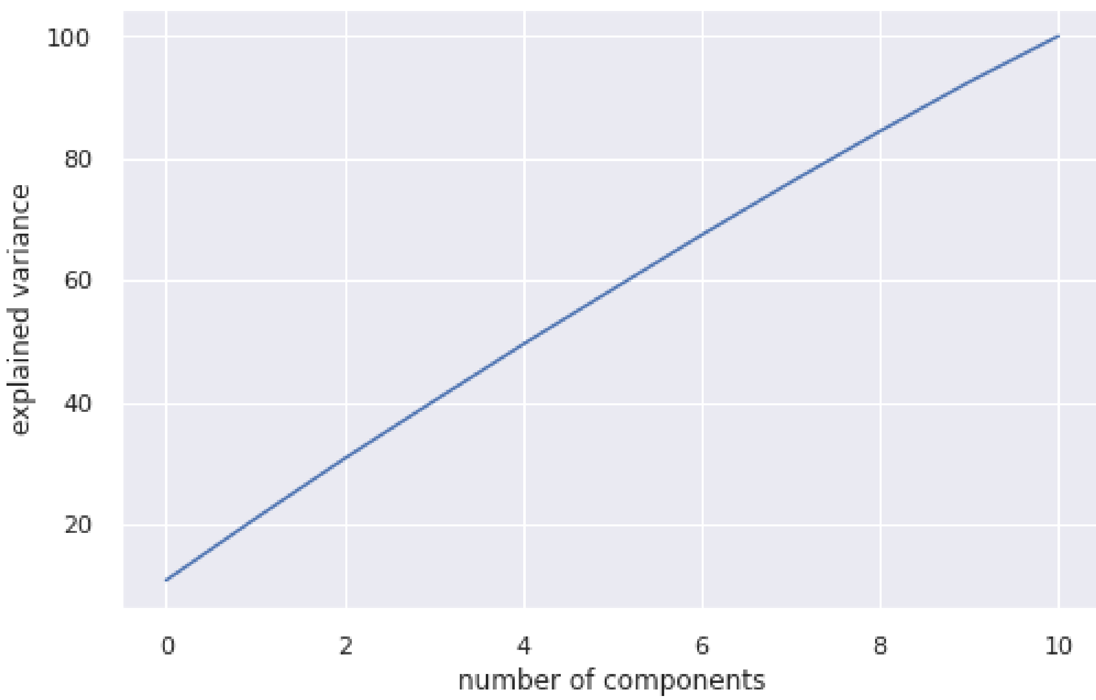
[161]: PCA()

[162]:
```
print('Variance Explained - %')
print(pca.explained_variance_ratio_ * 100)
```

```
Variance Explained - %
[10.81905112 10.11529244  9.84157375  9.41441669  9.2943261   8.97426177
  8.93572742  8.66026532  8.35614929  8.0661466   7.52278951]
```

[163]:
```
plt.plot(np.cumsum(pca.explained_variance_ratio_*100))
plt.xlabel('number of components')
plt.ylabel('explained variance')
```

[163]: Text(0, 0.5, 'explained variance')



We can see that we likely need all of our included predictors in order to explain enough of the variance. Unfortunately, with this approach, we can't retrieve p-values or other statistical measures to identify significance of any of these features. What this approach can tell us is whether dimension reduction should be explored (meaning that there are unneeded variables that we can drop from our analysis), as well as whether a logistic regression model (or other linear model) can be used to model our data. Because we are getting relatively good accuracy with our model, we have sufficient evidence that we can build a relatively good classifier just with a logistic regression model. Additionally, because each of our features seems to explain a relatively equal amount of variance (which is visualized by the linear plot above), we would not want to explore dimension reduction. This makes any simple logistic regression model difficult to fit manually, particularly with interaction effects, because we have so many features. It also rules out some non-parametric methods like K-Nearest Neighbors, which suffer from the curse of dimensionality. This is not necessarily surprising, given that we're working with a simulated dataset. Unfortunately, this means we are constrained to fitting models with most (if not all) of our current predictors, with none of them particularly more important than another.

We can see that we need all of our predictors that we curently have in order to explain a large proportion of the variance. For example, we could look at our original dataset, and see the difference in the shape of the curve.

[164]:
```
y = data2['Attrition']
x = data2.drop('Attrition',axis=1)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.20,␣
↪shuffle=True, random_state=2)
```

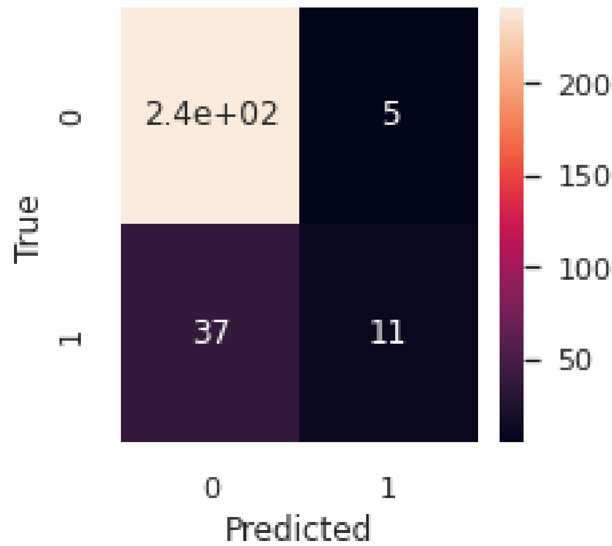
[165]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(max_iter = 2500)
clf.fit(x_train,y_train)
print(clf.coef_,clf.intercept_)
y_pred = clf.predict(x_test)
y_true = y_test
```

```
[[-4.01931431e-02 -3.87113694e-04  3.69115417e-02 -6.91870722e-01
   3.90152196e-01 -6.12167694e-01 -3.46976445e-01 -6.28506388e-01
   1.19136262e-01  1.62643921e+00 -1.59687368e-02 -2.83262485e-01
  -5.17870791e-01  7.94042672e-02 -1.00768865e-01 -9.67570081e-02
  -3.60854674e-01  1.03640760e-02  9.91690466e-01  2.83111536e-01
   1.27603698e-01  2.30484634e-01]] [0.74751367]
```

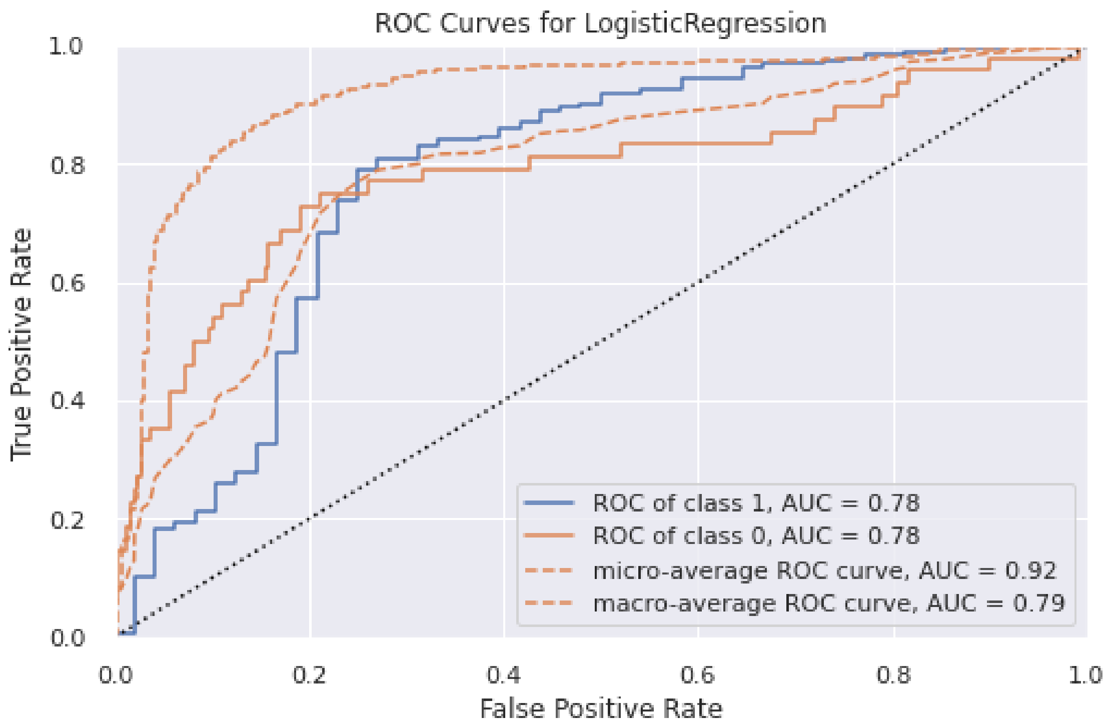
[166]:
```
import numpy as np
from sklearn.metrics import accuracy_score
print("Training accuracy:")
print(np.round(accuracy_score(y_train,clf.predict(x_train)),2))
print("Test accuracy:")
print(np.round(accuracy_score(y_true,y_pred),2))
from sklearn.metrics import confusion_matrix
sn.set(rc={'figure.figsize':(3,3)})
sn.set(font_scale=1)
matrix = confusion_matrix(y_true,y_pred)
sn.heatmap(matrix,annot=True)
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
Training accuracy:
0.86
Test accuracy:
0.86
```

[166]: Text(0.0, 0.5, 'True')



[167]:
```
from yellowbrick.classifier import ROCAUC
visualizer = ROCAUC(clf,classes=[1,0])
sn.set(rc={'figure.figsize':(8,5)})
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
```



[167]: <AxesSubplot:title={'center':'ROC Curves for LogisticRegression'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>

We can see that we're getting a relatively similar result when using all of our original variables. So we were clearly able to reduce dimensionality successfully, reducing from over 30 predictors to 11.

[168]:
```
from sklearn.preprocessing import StandardScaler
y = data_full['Attrition']
x = data_full.drop('Attrition',axis=1)
scaler = StandardScaler()
x = scaler.fit_transform(x)
```

[169]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = None)
pca.fit(x)
```

[169]: PCA()

[170]:
```
import numpy as np
import statsmodels.formula.api as smf
import statsmodels.api as sm
train = np.random.choice(data.index,200)
train_data = data.loc[pd.Index(train)]
test = np.random.choice(data.index,200)
```

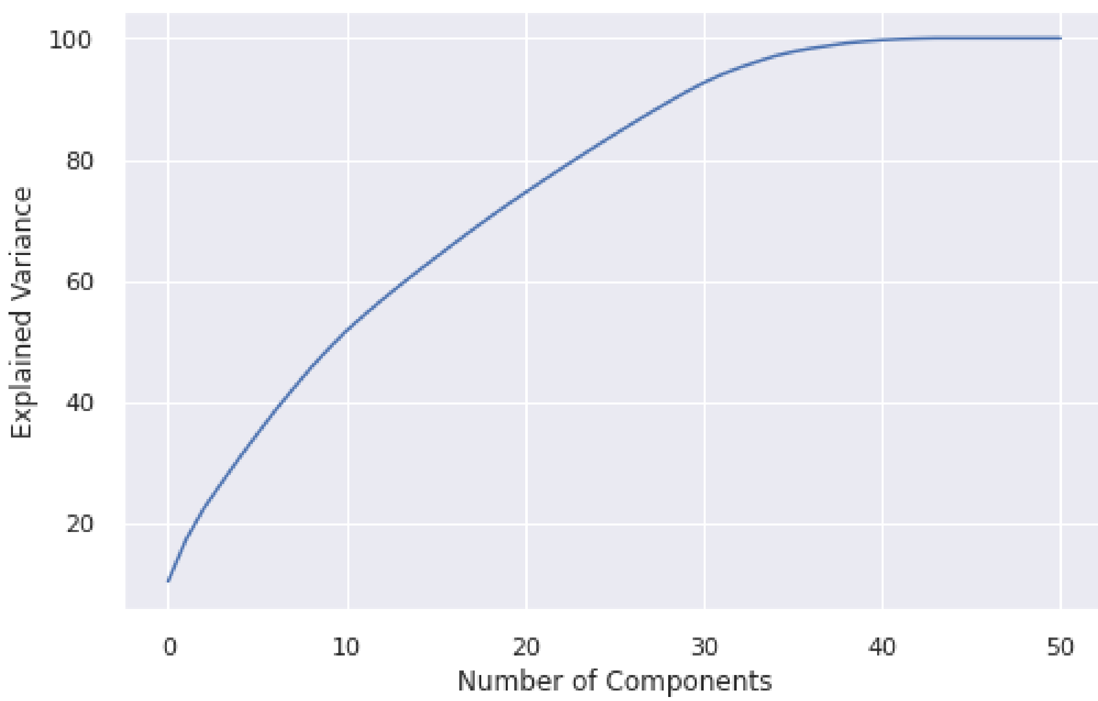
```
test_data = data.loc[pd.Index(train)]
```

[171]:
```
print('Variance Explained - %')
print(pca.explained_variance_ratio_ * 100)
```

```
Variance Explained - %
[1.04948948e+01 6.87358414e+00 5.17043810e+00 4.22547271e+00
 4.12216131e+00 3.92193761e+00 3.83255414e+00 3.54533801e+00
 3.48951017e+00 3.16112386e+00 2.95863529e+00 2.60466287e+00
 2.49698104e+00 2.39279674e+00 2.30537900e+00 2.25217367e+00
 2.24221930e+00 2.19956459e+00 2.14417305e+00 2.07928353e+00
 1.99293944e+00 1.98635845e+00 1.96447115e+00 1.90451280e+00
 1.88158548e+00 1.84621729e+00 1.81914228e+00 1.78006107e+00
 1.71610976e+00 1.68043870e+00 1.54641017e+00 1.34643969e+00
 1.13435417e+00 9.94155254e-01 9.41399841e-01 6.99525497e-01
 5.44454514e-01 4.43135267e-01 4.29131066e-01 2.91281317e-01
 2.17835738e-01 1.58099378e-01 1.01184067e-01 6.78737353e-02
 2.06975304e-30 6.43525977e-31 3.32774468e-31 3.10047628e-31
 2.26411319e-31 1.88038381e-31 6.84064085e-32]
```

[172]:
```
plt.plot(np.cumsum(pca.explained_variance_ratio_*100))
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
```

[172]: Text(0, 0.5, 'Explained Variance')



We can see that we're getting relatively similar accuracy, but that the amount of variance we're able to explain tapers off past around 30 variables. So it makes sense why our current predictor set of 11 variables is capturing a good amount of the variance. This is in part because we reduced collinearity manually above, so we would expect to have less variables explaining the overall variance.

We could try to use methods to reduce the dimensionality by combining variables. This will unfortunately make it difficult for us to use the model to infer something about the relationship between our variables and the outcome of attrition. Therefore, we will use another approach to building our model below.

`Recursive Feature Elimination - RFE`

We can use a different method altogether for feature selection - recursive feature elimination. Let's use our original cleaned dataset to see whether this algorithm selects the same features that we did during our EDA.

[173]:
```python
from sklearn.feature_selection import RFE
x = data_full.drop(columns=['Attrition'])
y = data_full['Attrition']
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size = 0.2)
model = LogisticRegression()
rfe = RFE(model)
fit = rfe.fit(train_x,train_y)
```

[174]:
```python
col = x.columns
RFE_sup = rfe.support_
RFE_rank = rfe.ranking_
dataset = pd.DataFrame({'Columns': col, 'RFE_support': RFE_sup, 'RFE_ranking':␣
↪RFE_rank}, columns=['Columns', 'RFE_support', 'RFE_ranking'])
df = dataset[(dataset["RFE_support"] == True) & (dataset["RFE_ranking"] == 1)]
filtered_features = df['Columns']
filtered_features
```

[174]:
```
4               EnvironmentSatisfaction
6                        JobInvolvement
8                       JobSatisfaction
17                TrainingTimesLastYear
18                       WorkLifeBalance
23             BusinessTravel_Non-Travel
24      BusinessTravel_Travel_Frequently
28                      Department_Sales
29          EducationField_Human Resources
30            EducationField_Life Sciences
32                  EducationField_Medical
33                    EducationField_Other
34       EducationField_Technical Degree
36                           Gender_Male
```