rhyme

Predict Employee Churn with Decision Trees and Random Forests

Task 1: Import Libraries

In [1]:

```
from __future__ import print_function
%matplotlib inline
import os
import warnings
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as image
import pandas as pd
import pandas_profiling
plt.style.use("ggplot")
warnings.simplefilter("ignore")
```

In [2]:

```
plt.rcParams['figure.figsize'] = (12,8)
```

Task 2: Exploratory Data Analysis

In [3]:

```
hr = pd.read_csv('data/employee_data.csv')
hr_orig = hr
hr.head()
```

Out[3]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_
0	0.38	0.53	2	157	3	
1	0.80	0.86	5	262	6	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	

In [4]:

```
hr.profile_report(title="Data")
```

Out[4]:

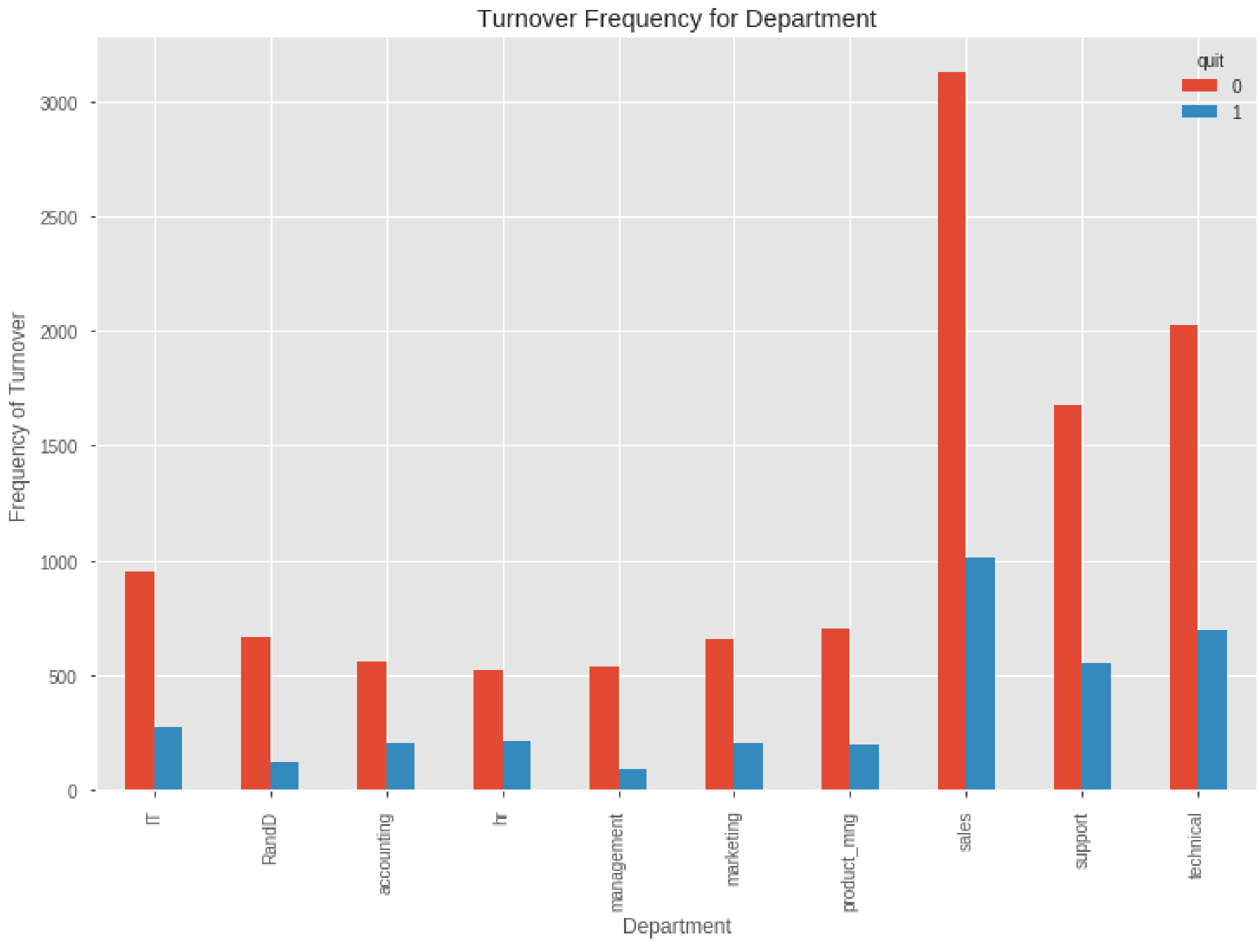
Task 3: Encode Categorical Features

In [5]:

```
pd.crosstab(hr.salary,hr.quit).plot(kind='bar')
plt.title('Turnover Frequency on Salary Bracket')
plt.xlabel('Salary')
plt.ylabel('Frequency of Turnover')
plt.show()
```

In [6]:

```
pd.crosstab(hr.department,hr.quit).plot(kind='bar')
plt.title('Turnover Frequency for Department')
plt.xlabel('Department')
plt.ylabel('Frequency of Turnover')
plt.show()
```



In [7]:

```
cat_vars=['department','salary']
for var in cat_vars:
```

```
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(hr[var], prefix=var)
    hr1=hr.join(cat_list)
    hr=hr1
```

In [8]:

```
hr.columns
```

Out[8]:

```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'quit',
       'promotion_last_5years', 'department', 'salary', 'department_IT',
       'department_RandD', 'department_accounting', 'department_hr',
       'department_management', 'department_marketing',
       'department_product_mng', 'department_sales', 'department_support',
       'department_technical', 'salary_high', 'salary_low', 'salary_medium'],
      dtype='object')
```

In [9]:

```
hr.drop(columns=['department','salary'], axis=1, inplace=True)
#hr.drop(hr.columns[[8,9,10,11,12,13,14,15,16,17,18,19,20,21,22]], axis=1, inplace=True)
```

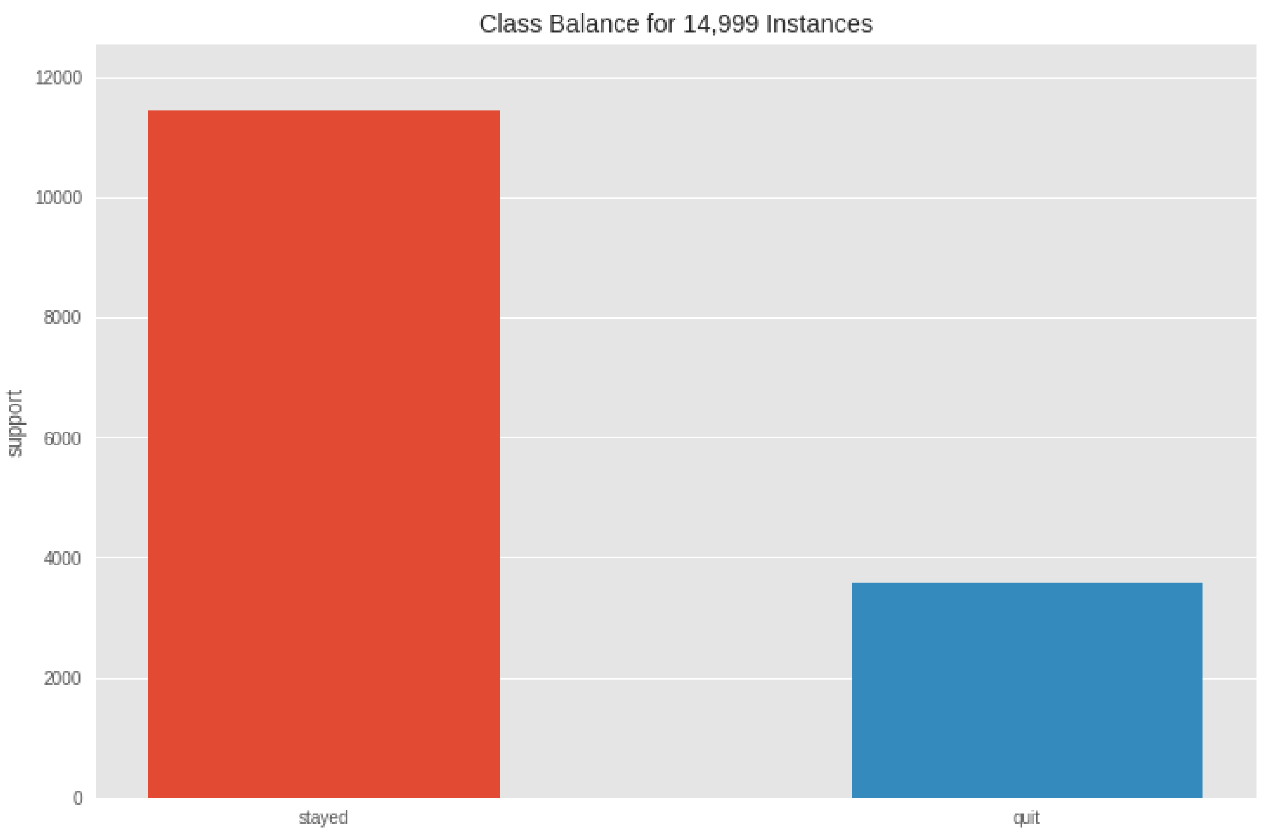
Task 4: Visualize Class Imbalance

In [20]:

```
from yellowbrick.target import ClassBalance
plt.style.use("ggplot")
plt.rcParams['figure.figsize'] = (12,8)
```

In [10]:

```
visualizer = ClassBalance(labels=["stayed", "quit"])

visualizer.fit(hr.quit)
visualizer.show();
```



Task 5: Create Training and Test Sets

In [11]:

```
X = hr.loc[:, hr.columns != 'quit']
```

```
y = hr.quit
```

In [12]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0,
                                                    stratify=y)
```

Task 6 & 7: Build an Interactive Decision Tree Classifier

Supervised learning:

- The inputs are random variables $X = X_1, \dots, X_p$;
- The output is a random variable Y.

- Data is a finite set

$$\mathbb{L} = \{(x_i, y_i) | i = 0, \dots, N-1\}$$

  where $x_i \in X = X_1 \times \dots \times X_p$ and $y_i \in y$ are randomly drawn from $P_{X,Y}$.

E.g., $(x_i, y_i) = ((\text{salary} = \text{low}, \text{department} = \text{sales}, \dots), \text{quit} = 1)$

- The goal is to find a model $\varphi_{\mathbb{L}} : X \mapsto y$ minimizing

$$\text{Err}(\varphi_{\mathbb{L}}) = \mathbb{E}_{X,Y}\{L(Y, \varphi_{\mathbb{L}}(X))\}.$$

About:

- Decision trees are non-parametric models which can model arbitrarily complex relations between inputs and outputs, without any a priori assumption

- Decision trees handle numeric and categorical variables

- They implement feature selection, making them robust to noisy features (to an extent)

- Robust to outliers or errors in labels

- Easily interpretable by even non-ML practioners.

Decision trees: partitioning the feature space:

partition

- Decision trees generally have low bias but have high variance.
- We will solve the high variance problem in Task 8.

In [13]:

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import export_graphviz # display the tree within a Jupyter notebook
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
from ipywidgets import interactive, IntSlider, FloatSlider, interact
import ipywidgets
from IPython.display import Image
from subprocess import call
import matplotlib.image as mpimg
```

In [14]:

```
@interact
def plot_tree(crit=["gini", "entropy"],
              split=["best", "random"],
              depth=IntSlider(min=1,max=30,value=2, continuous_update=False),
              min_split=IntSlider(min=2,max=5,value=2, continuous_update=False),
              min_leaf=IntSlider(min=1,max=5,value=1, continuous_update=False)):

    estimator = DecisionTreeClassifier(random_state=0,
                                       criterion=crit,
                                       splitter = split,
                                       max_depth = depth,
                                       min_samples_split=min_split,
                                       min_samples_leaf=min_leaf)
    estimator.fit(X_train, y_train)
    print('Decision Tree Training Accuracy: {:.3f}'.format(accuracy_score(y_train, estim
    print('Decision Tree Test Accuracy: {:.3f}'.format(accuracy_score(y_test, estimator.

    graph = Source(tree.export_graphviz(estimator,
                                        out_file=None,
                                        feature_names=X_train.columns,
                                        class_names=['0', '1'],
                                        filled = True))

    display(Image(data=graph.pipe(format='png')))

    return estimator
```

```
interactive(children=(Dropdown(description='crit', options=('gini', 'entropy'), value='g
ini'), Dropdown(descri…
```

Task 8: Build an Interactive Random Forest Classifier

Although randomization increases bias, it is possible to get a reduction in variance of the ensemble. Random forests are one of the most robust machine learning algorithms for a variety of problems.

- Randomization and averaging lead to a reduction in variance and improve accuracy
- The implementations are parallelizable
- Memory consumption and training time can be reduced by bootstrapping
- Sampling features and not solely sampling examples is crucial to improving accuracy

In [15]:

```
@interact
def plot_tree_rf(crit=["gini", "entropy"],
                 bootstrap=["True", "False"],
                 depth=IntSlider(min=1,max=30,value=3, continuous_update=False),
                 forests=IntSlider(min=1,max=200,value=100,continuous_update=False),
                 min_split=IntSlider(min=2,max=5,value=2, continuous_update=False),
                 min_leaf=IntSlider(min=1,max=5,value=1, continuous_update=False)):

    estimator = RandomForestClassifier(random_state=1,
                                       criterion=crit,
                                       bootstrap=bootstrap,
                                       n_estimators=forests,
                                       max_depth=depth,
                                       min_samples_split=min_split,
                                       min_samples_leaf=min_leaf,
                                       n_jobs=-1,
                                      verbose=False).fit(X_train, y_train)

    print('Random Forest Training Accuracy: {:.3f}'.format(accuracy_score(y_train, estim
    print('Random Forest Test Accuracy: {:.3f}'.format(accuracy_score(y_test, estimator.
    num_tree = estimator.estimators_[0]
    print('\nVisualizing Decision Tree:', 0)
```

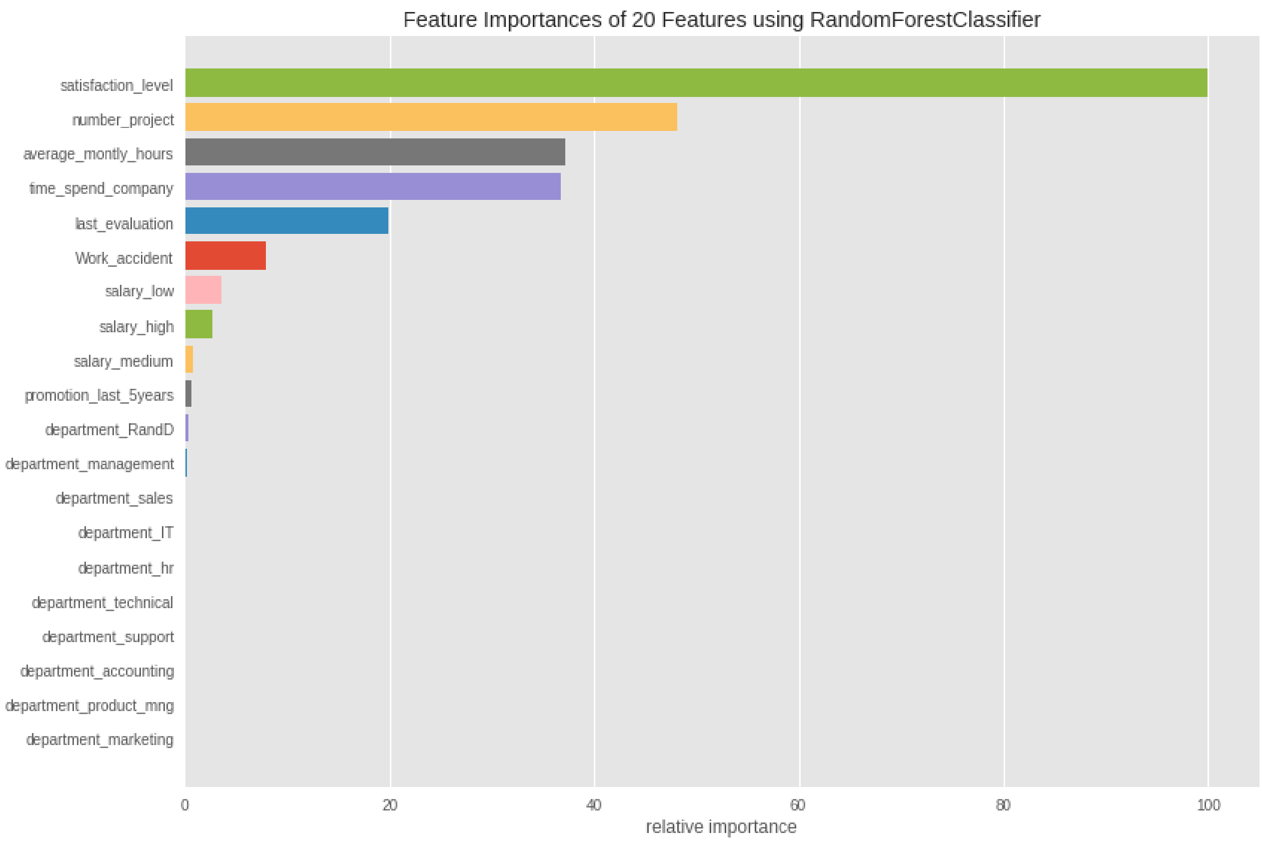
```
    graph = Source(tree.export_graphviz(num_tree,
                                        out_file=None,
                                        feature_names=X_train.columns,
                                        class_names=['0', '1'],
                                        filled = True))

    display(Image(data=graph.pipe(format='png')))

    return estimator
```

```
interactive(children=(Dropdown(description='crit', options=('gini', 'entropy'), value='g
ini'), Dropdown(descri…
```

Task 9: Feature Importance and Evaluation Metrics

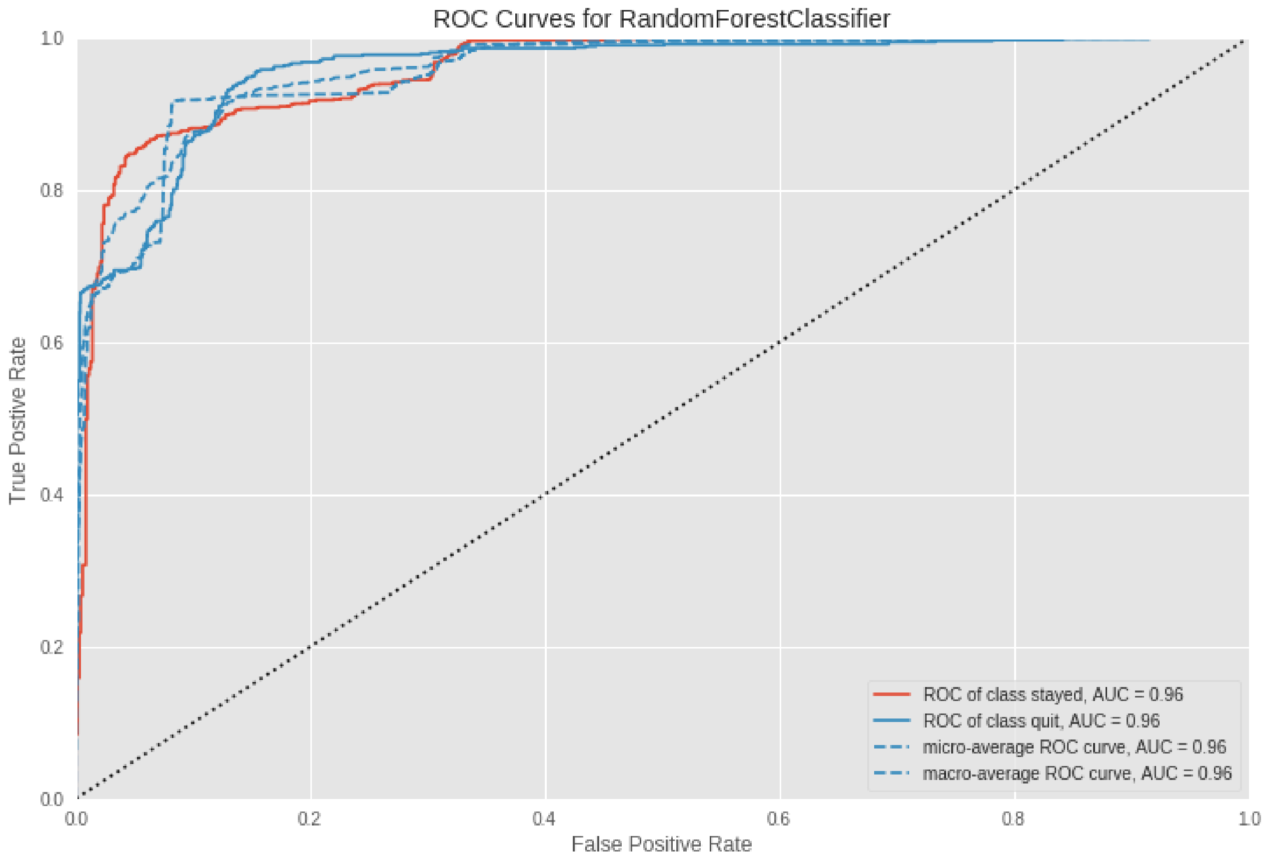
In [16]:

```
from yellowbrick.model_selection import FeatureImportances
plt.rcParams['figure.figsize'] = (12,8)
plt.style.use("ggplot")

rf = RandomForestClassifier(bootstrap='True', class_weight=None, criterion='gini',
             max_depth=3, max_features='auto', max_leaf_nodes=None,
             min_impurity_decrease=0.0, min_impurity_split=None,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
             oob_score=False, random_state=1, verbose=False,
             warm_start=False)

viz = FeatureImportances(rf)
viz.fit(X_train, y_train)
viz.show();
```



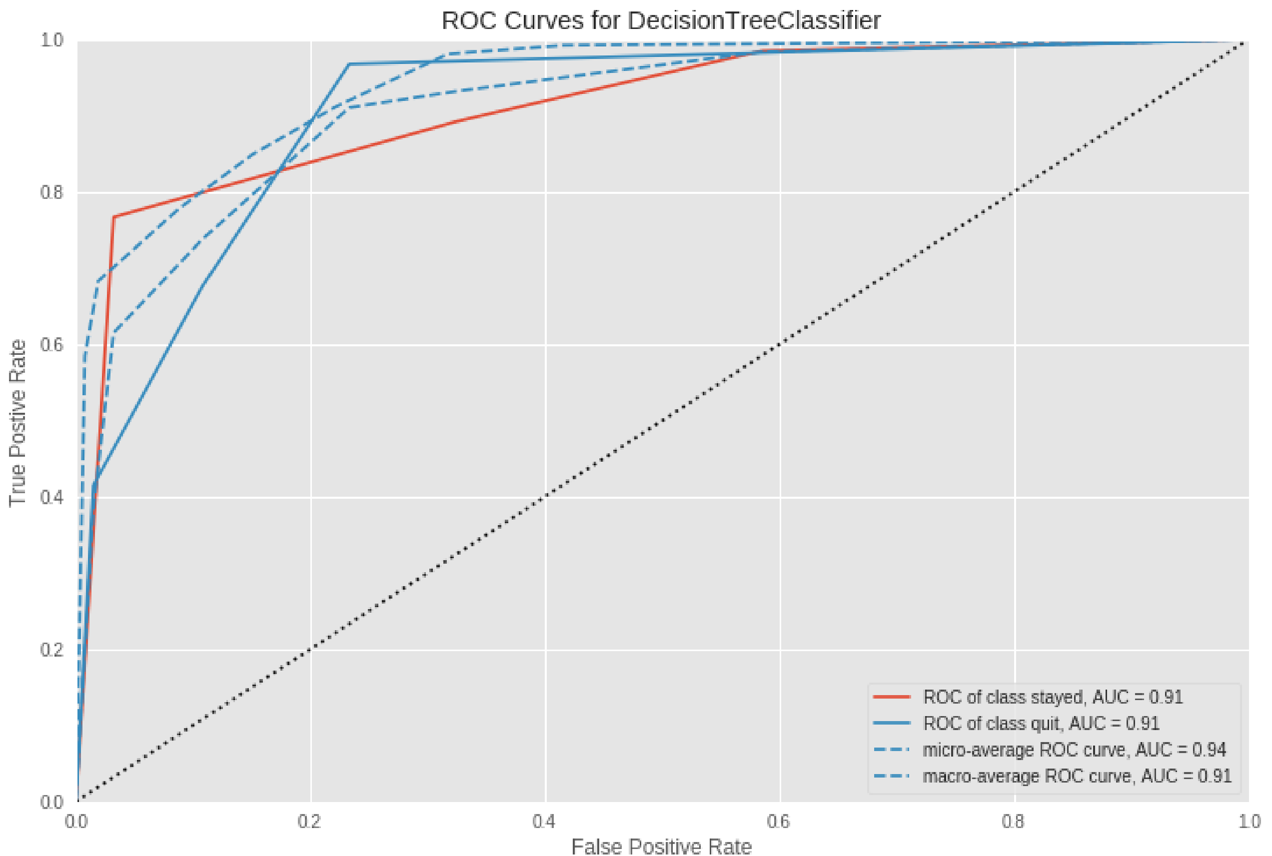
In [17]:

```
from yellowbrick.classifier import ROCAUC

visualizer = ROCAUC(rf, classes=["stayed", "quit"])

visualizer.fit(X_train, y_train)        # Fit the training data to the visualizer
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.poof();
```



In [18]:

```
dt = DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')

visualizer = ROCAUC(dt, classes=["stayed", "quit"])

visualizer.fit(X_train, y_train)        # Fit the training data to the visualizer
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.poof();
```

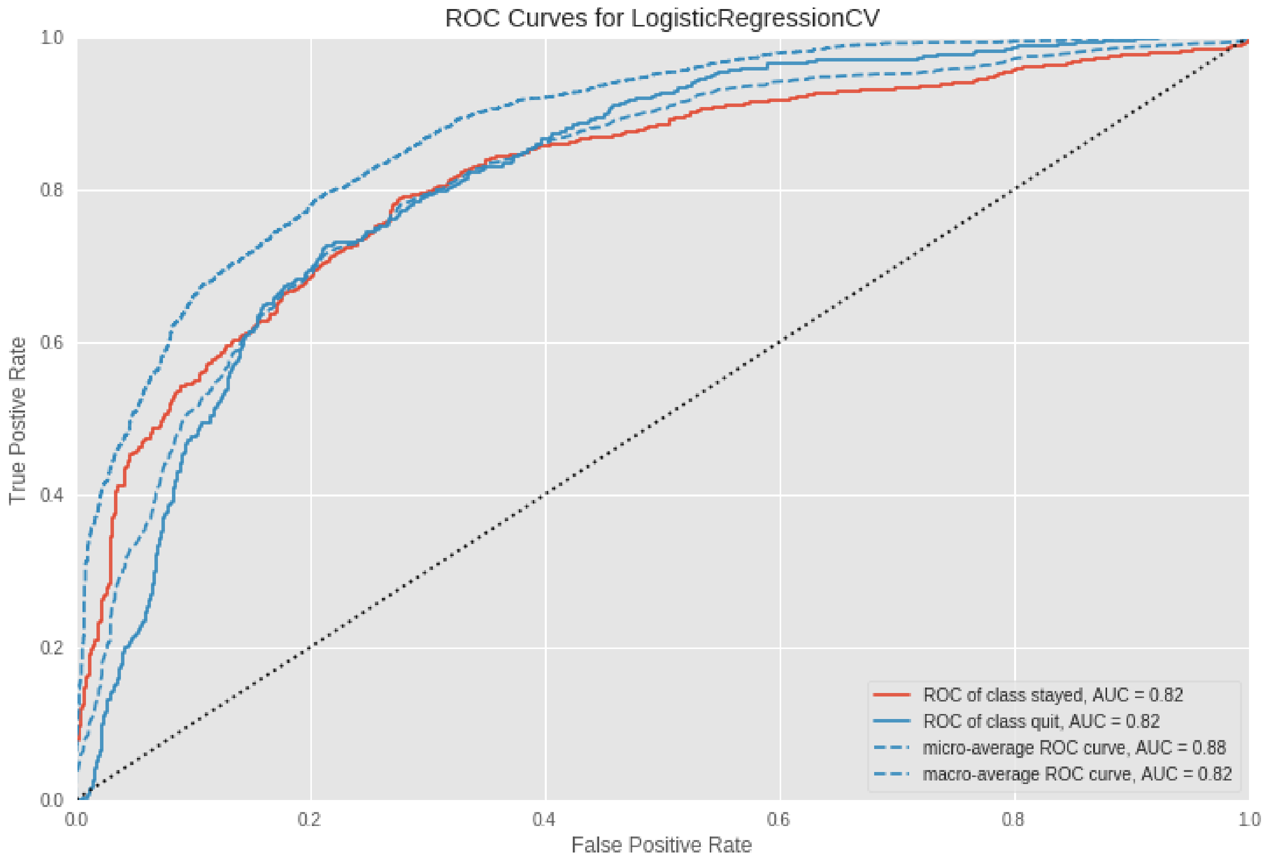


Optional: Comparison with Logistic Regression Classifier

In [19]:

```
from sklearn.linear_model import LogisticRegressionCV

logit = LogisticRegressionCV(random_state=1, n_jobs=-1,max_iter=500,
                             cv=10)

lr = logit.fit(X_train, y_train)

print('Logistic Regression Accuracy: {:.3f}'.format(accuracy_score(y_test, lr.predict(X_

visualizer = ROCAUC(lr, classes=["stayed", "quit"])

visualizer.fit(X_train, y_train)        # Fit the training data to the visualizer
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.poof();
```

```
Logistic Regression Accuracy: 0.790
```



In [ ]: