In [5]:

```
# Problem - I decided to treat this as a classification problem by creating
# (did the woman have at least one affair?) and trying to predict the classi
# Dataset
# The dataset I chose is the affairs dataset that comes with Statsmodels. It
# by Redbook magazine, in which married women were asked about their partici
# information about the study is available in a 1978 paper from the Journal
# Description of Variables
# The dataset contains 6366 observations of 9 variables:
# rate_marriage: woman's rating of her marriage (1 = very poor, 5 = very goo
# age: woman's age
# yrs_married: number of years married
# children: number of children
# religious: woman's rating of how religious she is (1 = not religious, 4 =
# educ: level of education (9 = grade school, 12 = high school, 14 = some co
# college graduate, 17 = some graduate school, 20 = advanced degree)
# occupation: woman's occupation (1 = student, 2 = farming/semi-skilled/unsk
# "white collar", 4 = teacher/nurse/writer/technician/skilled, 5 = manageria
# professional with advanced degree)
# occupation_husb: husband's occupation (same coding as above)
# affairs: time spent in extra-marital affairs
# Code to loading data and modules
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from patsy import dmatrices
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn.cross_validation import cross_val_score
dta = sm.datasets.fair.load_pandas().data
# add "affair" column: 1 represents having affairs, 0 represents not
dta['affair'] = (dta.affairs > 0).astype(int)
y, X = dmatrices('affair ~ rate_marriage + age + yrs_married + children + \
religious + educ + C(occupation) + C(occupation_husb)',
dta, return_type="dataframe")
X = X.rename(columns = {'C(occupation)[T.2.0]':'occ_2',
'C(occupation)[T.3.0]':'occ_3',
'C(occupation)[T.4.0]':'occ_4',
'C(occupation)[T.5.0]':'occ_5',
'C(occupation)[T.6.0]':'occ_6',
'C(occupation_husb)[T.2.0]':'occ_husb_2',
'C(occupation_husb)[T.3.0]':'occ_husb_3',
'C(occupation_husb)[T.4.0]':'occ_husb_4',
'C(occupation_husb)[T.5.0]':'occ_husb_5',
'C(occupation_husb)[T.6.0]':'occ_husb_6'})
y = np.ravel(y)
```

In [4]:

```
import numpy as np
import pandas as pd
#using pandas.tseries instead of statsmodels.api
import pandas.tseries as pdt
import matplotlib.pyplot as plt
from patsy import dmatrices
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import cross_val_score
#To avoid warnings
import warnings
warnings.filterwarnings('ignore')
dta = sm.datasets.fair.load_pandas().data
df_affair = dta.copy()
```

In [6]:

```
# add "affair" column: 1 represents having affairs, 0 represents not

dta['affair'] = (dta.affairs > 0).astype(int)
y, X = dmatrices('affair ~ rate_marriage + age + yrs_married + children + \
religious + educ + C(occupation) + C(occupation_husb)',
dta, return_type="dataframe")

X = X.rename(columns = {'C(occupation)[T.2.0]':'occ_2',
'C(occupation)[T.3.0]':'occ_3',
'C(occupation)[T.4.0]':'occ_4',
'C(occupation)[T.5.0]':'occ_5',
'C(occupation)[T.6.0]':'occ_6',
'C(occupation_husb)[T.2.0]':'occ_husb_2',
'C(occupation_husb)[T.3.0]':'occ_husb_3',
'C(occupation_husb)[T.4.0]':'occ_husb_4',
'C(occupation_husb)[T.5.0]':'occ_husb_5',
'C(occupation_husb)[T.6.0]':'occ_husb_6'})
y = np.ravel(y)
```

In [7]:

```
dta.head()
```

Out[7]:

	rate_marriage	age	yrs_married	children	religious	educ	occupation	occupation_husb	affa
0	3.0	32.0	9.0	3.0	3.0	17.0	2.0	5.0	0.111
1	3.0	27.0	13.0	3.0	1.0	14.0	3.0	4.0	3.230
2	4.0	22.0	2.5	0.0	1.0	16.0	3.0	5.0	1.400
3	4.0	37.0	16.5	4.0	3.0	16.0	5.0	5.0	0.727
4	5.0	27.0	9.0	1.0	1.0	14.0	3.0	4.0	4.666

In [8]:

```
dta.shape
```

Out[8]: (6366, 10)

In [9]:
```
X.head()
```
Out[9]:

	Intercept	occ_2	occ_3	occ_4	occ_5	occ_6	occ_husb_2	occ_husb_3	occ_husb_4	occ_husl
0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
2	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
4	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	

In [10]:
```
y
```
Out[10]:
```
array([1., 1., 1., ..., 0., 0., 0.])
```

In [11]:
```
print("Lets analyze the data and look at the summary statistics")
dta.describe()
```

```
Lets analyze the data and look at the summary statistics
```

Out[11]:

	rate_marriage	age	yrs_married	children	religious	educ	occupatio
count	6366.000000	6366.000000	6366.000000	6366.000000	6366.000000	6366.000000	6366.00000
mean	4.109645	29.082862	9.009425	1.396874	2.426170	14.209865	3.42412
std	0.961430	6.847882	7.280120	1.433471	0.878369	2.178003	0.94239
min	1.000000	17.500000	0.500000	0.000000	1.000000	9.000000	1.00000
25%	4.000000	22.000000	2.500000	0.000000	2.000000	12.000000	3.00000
50%	4.000000	27.000000	6.000000	1.000000	2.000000	14.000000	3.00000
75%	5.000000	32.000000	16.500000	2.000000	3.000000	16.000000	4.00000
max	5.000000	42.000000	23.000000	5.500000	4.000000	20.000000	6.00000

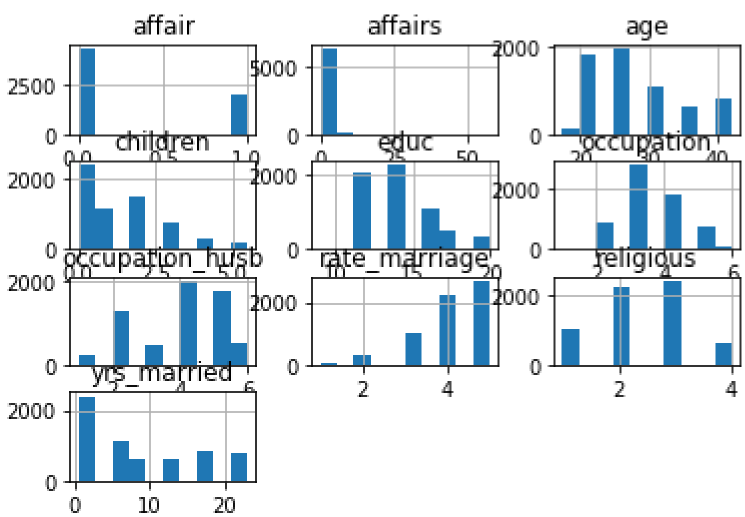
In [12]:

```
# plot all of the columns
%matplotlib inline
plt.figure(figsize=(20,18))
dta.hist()
```

Out[12]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x09FBAE90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09F56D50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09F46C50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x09F3AB50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09F60A50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09F689F0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0A02CB30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0A0465F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0A046B70>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0A05FCF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x039C5DB0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x039E0E70>]],
      dtype=object)
```

```
<Figure size 1440x1296 with 0 Axes>
```



In [13]:

```
print("Split the data into training and test set")
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
Split the data into training and test set
(4456, 17)
(4456,)
(1910, 17)
(1910,)
```

In [14]:

```
#We will use the statsmodels Logit function for logistic regression
logit = sm.Logit(y_train, X_train)

# fit the model
result = logit.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.544479
         Iterations 6
```

In [15]:
```
predictions =  result.predict(X_test)
predictions
```

Out[15]:
```
2764    0.653211
4481    0.087718
5360    0.273074
5802    0.249471
1220    0.249630
5812    0.166215
3719    0.160619
3848    0.202858
1865    0.760648
2535    0.310242
2505    0.104535
6273    0.186280
3710    0.075798
4229    0.300914
1262    0.736723
5321    0.593884
3790    0.296514
994     0.732196
5644    0.296810
2252    0.156072
1804    0.203707
861     0.448163
1601    0.089842
1718    0.471631
2976    0.168971
3603    0.161252
4130    0.396058
5824    0.361373
5901    0.252252
4408    0.087566
          ...
5615    0.242151
1737    0.515613
2701    0.185493
4024    0.419446
1012    0.129569
3888    0.143331
4746    0.207049
5607    0.132057
1946    0.874461
3119    0.133824
156     0.604838
1752    0.653266
624     0.612704
4622    0.556290
1788    0.693977
500     0.264134
726     0.220550
4162    0.087718
48      0.158934
1691    0.572606
5882    0.152058
2244    0.395267
1985    0.841614
```

```
2853    0.223091
18      0.803795
3053    0.144139
1875    0.207506
5851    0.437646
4962    0.190124
1995    0.249630
Length: 1910, dtype: float64
```

In [16]:

```python
from scipy import stats
stats.chisqprob = lambda chisq, df: stats.chi2.sf(chisq, df)
```

In [17]:

```python
result.summary()
```

Out[17]:

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	4456
Model:	Logit	**Df Residuals:**	4439
Method:	MLE	**Df Model:**	16
Date:	Tue, 05 Mar 2019	**Pseudo R-squ.:**	0.1360
Time:	11:51:13	**Log-Likelihood:**	-2426.2
converged:	True	**LL-Null:**	-2808.3
		LLR p-value:	2.844e-152

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	2.4842	0.777	3.198	0.001	0.961	4.007
occ_2	0.9414	0.658	1.432	0.152	-0.347	2.230
occ_3	1.2324	0.652	1.890	0.059	-0.046	2.511
occ_4	0.9731	0.653	1.490	0.136	-0.307	2.254
occ_5	1.6017	0.657	2.436	0.015	0.313	2.890
occ_6	1.8242	0.707	2.581	0.010	0.439	3.209
occ_husb_2	0.0649	0.215	0.302	0.762	-0.356	0.486
occ_husb_3	0.1976	0.235	0.841	0.400	-0.263	0.658
occ_husb_4	0.0304	0.208	0.146	0.884	-0.377	0.438
occ_husb_5	-0.0052	0.210	-0.025	0.980	-0.417	0.406
occ_husb_6	-0.0183	0.236	-0.078	0.938	-0.481	0.445
rate_marriage	-0.7145	0.038	-18.929	0.000	-0.788	-0.640
age	-0.0577	0.012	-4.686	0.000	-0.082	-0.034
yrs_married	0.1081	0.013	8.243	0.000	0.082	0.134
children	-0.0126	0.038	-0.329	0.742	-0.088	0.062
religious	-0.3889	0.042	-9.342	0.000	-0.470	-0.307
educ	0.0046	0.021	0.224	0.823	-0.036	0.045

In [18]:

```
print("Logistic Regression with scikit-learn")
dta.head()
```

```
Logistic Regression with scikit-learn
```

Out[18]:

	rate_marriage	age	yrs_married	children	religious	educ	occupation	occupation_husb	affa
0	3.0	32.0	9.0	3.0	3.0	17.0	2.0	5.0	0.111
1	3.0	27.0	13.0	3.0	1.0	14.0	3.0	4.0	3.2307
2	4.0	22.0	2.5	0.0	1.0	16.0	3.0	5.0	1.4000
3	4.0	37.0	16.5	4.0	3.0	16.0	5.0	5.0	0.7272
4	5.0	27.0	9.0	1.0	1.0	14.0	3.0	4.0	4.6666

In [19]:

```
print('Exploratary data analysis')
# people having affair is represented with 1 and not having affair is repres
dta.affair.value_counts()
```

```
Exploratary data analysis
```

Out[19]:

```
0    4313
1    2053
Name: affair, dtype: int64
```

In [20]:

```
print("We can conclude that women who have affairs, rate their marriage lowe
dta.groupby('affair').mean()
```

```
We can conclude that women who have affairs, rate their marriage lower based on
our findings from below table
```

Out[20]:

	rate_marriage	age	yrs_married	children	religious	educ	occupation	occupatio
affair								
0	4.329701	28.390679	7.989335	1.238813	2.504521	14.322977	3.405286	3
1	3.647345	30.537019	11.152460	1.728933	2.261568	13.972236	3.463712	3

In [21]:

```
print('Checking rate_marriage paramerter')
print('We can say with an increase in age, yrs_married and children correlat
dta.groupby('rate_marriage').mean()
```

```
Checking rate_marriage paramerter
We can say with an increase in age, yrs_married and children correlate with inc
rease in affairs based on findings.
```

Out[21]:

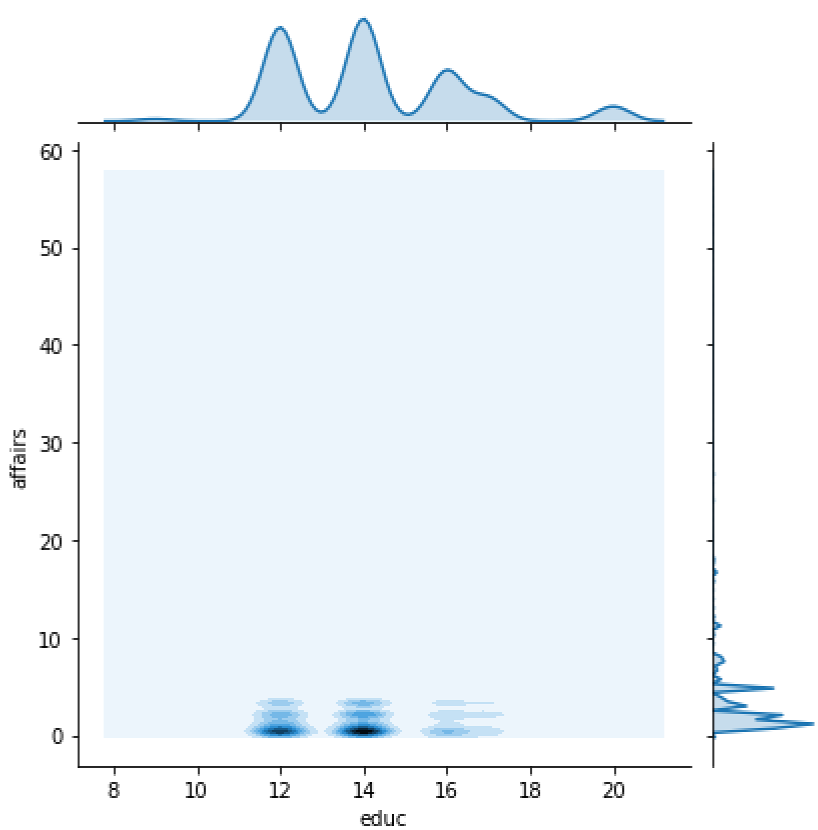
rate_marriage	age	yrs_married	children	religious	educ	occupation	occupation_husb
1.0	33.823232	13.914141	2.308081	2.343434	13.848485	3.232323	3.838384
2.0	30.471264	10.727011	1.735632	2.330460	13.864943	3.327586	3.764368
3.0	30.008056	10.239174	1.638469	2.308157	14.001007	3.402820	3.798590
4.0	28.856601	8.816905	1.369536	2.400981	14.144514	3.420161	3.835861
5.0	28.574702	8.311662	1.252794	2.506334	14.399776	3.454918	3.892697

In [22]:

```
print('Lets visualize our data')
import seaborn as sns
sns.jointplot(x='educ',y='affairs',data=dta,kind='kde')
```

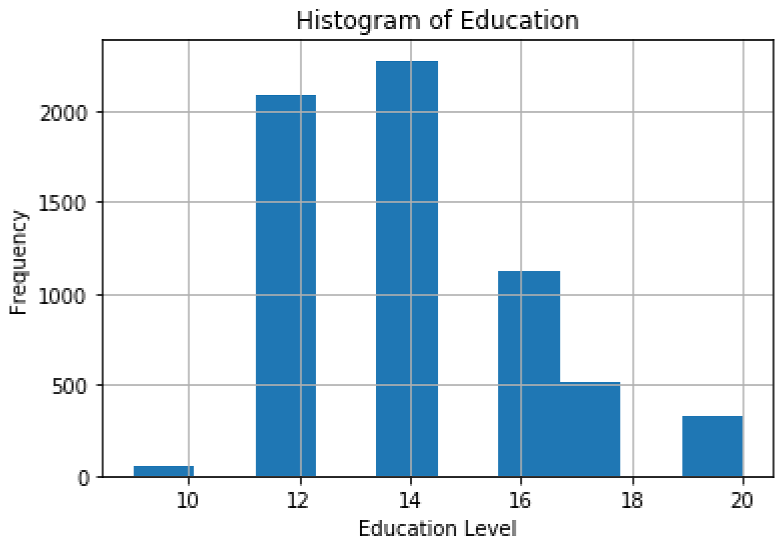
```
Lets visualize our data
```

Out[22]: <seaborn.axisgrid.JointGrid at 0x9eae170>



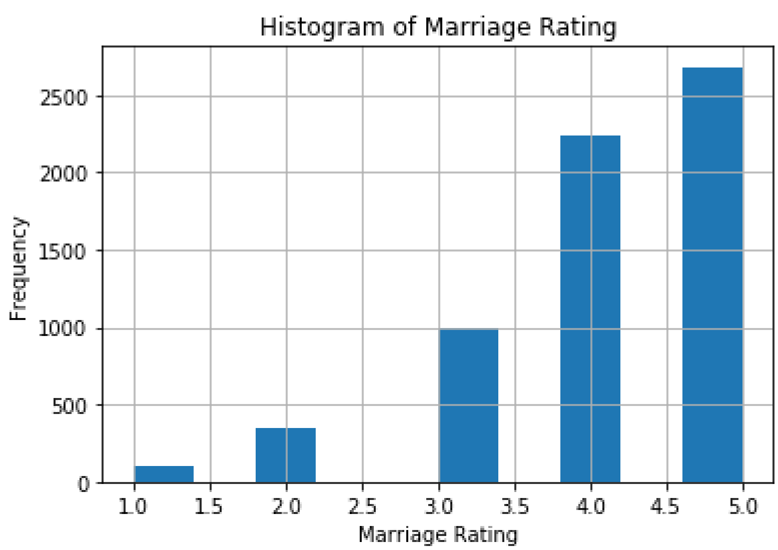
In [23]:
```
# histogram of education
dta.educ.hist()
plt.title('Histogram of Education')
plt.xlabel('Education Level')
plt.ylabel('Frequency')
```

Out[23]: Text(0, 0.5, 'Frequency')



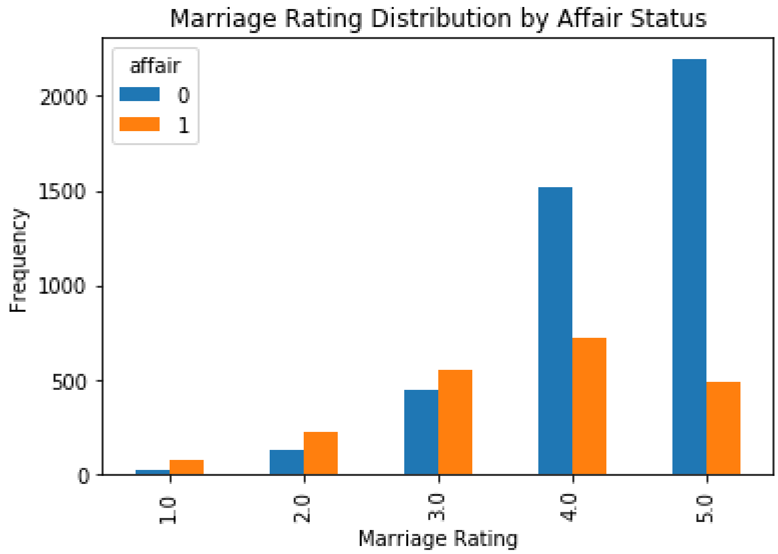
In [24]:
```
# histogram of marriage rating
dta.rate_marriage.hist()
plt.title('Histogram of Marriage Rating')
plt.xlabel('Marriage Rating')
plt.ylabel('Frequency')
```

Out[24]: Text(0, 0.5, 'Frequency')



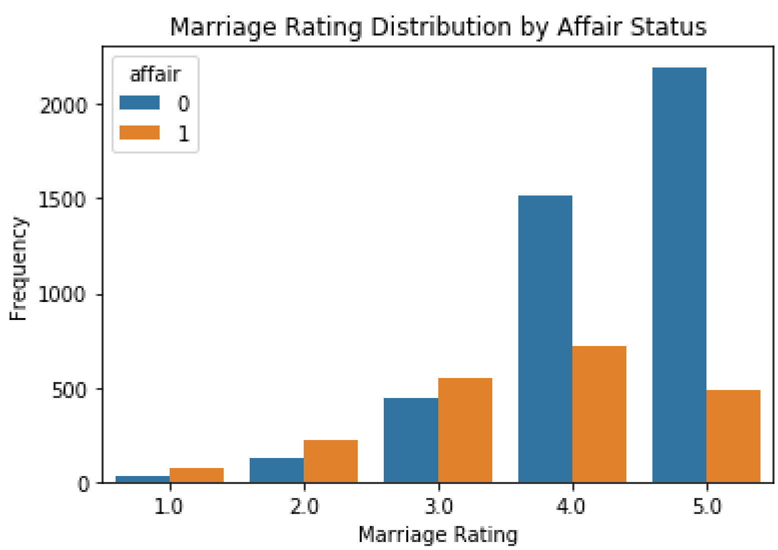
In [25]:
```
# barplot of marriage rating grouped by affair (True or False)
pd.crosstab(dta.rate_marriage, dta.affair).plot(kind='bar')
plt.title('Marriage Rating Distribution by Affair Status')
plt.xlabel('Marriage Rating')
plt.ylabel('Frequency')
```

Out[25]: Text(0, 0.5, 'Frequency')



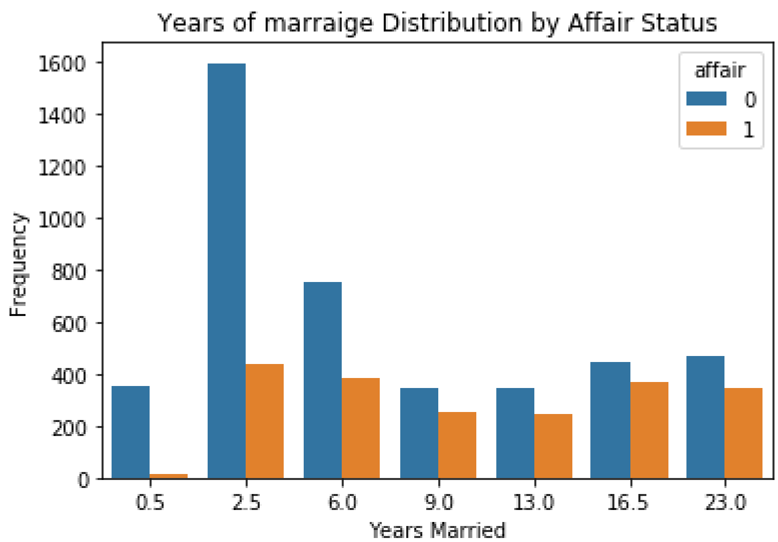
In [26]:
```
sns.countplot(x='rate_marriage',data=dta,hue='affair')
plt.title('Marriage Rating Distribution by Affair Status')
plt.xlabel('Marriage Rating')
plt.ylabel('Frequency')
```

Out[26]: Text(0, 0.5, 'Frequency')



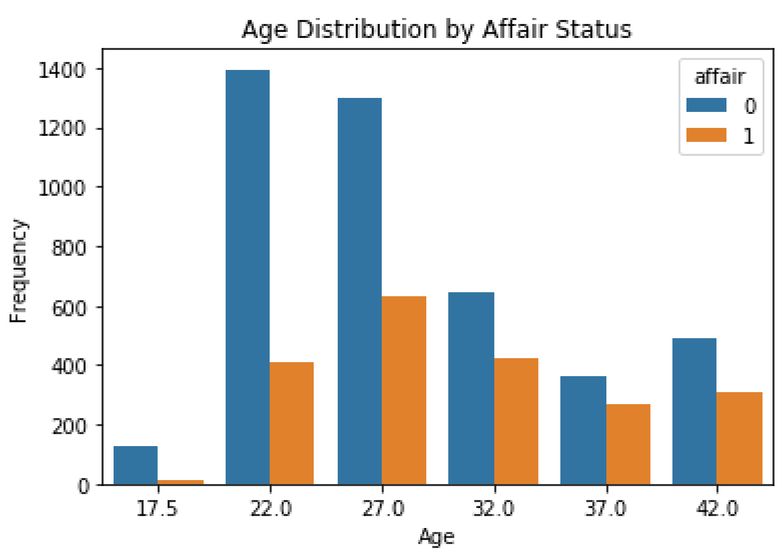
In [27]:
```python
sns.countplot(x='yrs_married',data=dta,hue='affair')
plt.title('Years of marraige Distribution by Affair Status')
plt.xlabel('Years Married')
plt.ylabel('Frequency')
```

Out[27]: Text(0, 0.5, 'Frequency')



In [28]:
```python
import seaborn as sns
sns.countplot(x='age',data=dta,hue='affair')
plt.title('Age Distribution by Affair Status')
plt.xlabel('Age')
plt.ylabel('Frequency')
```

Out[28]: Text(0, 0.5, 'Frequency')



In [29]:
```
print("Model Evaluation Using a Validation Set")
from sklearn.model_selection import train_test_split
# evaluate the model by splitting into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
Model Evaluation Using a Validation Set
(4456, 17)
(4456,)
(1910, 17)
(1910,)
```

In [30]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[30]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [31]:
```
print(model.score(X_train,y_train))
print("Training set has 73% accuracy")
```

```
0.723967684021544
Training set has 73% accuracy
```

In [32]:
```
print("Use the test data set to predict the class / labels")
# predict class labels for the test set
predicted = model.predict(X_test)
predicted
```

```
Use the test data set to predict the class / labels
```

Out[32]:
```
array([1., 0., 0., ..., 0., 0., 0.])
```

In [33]:
```
# generate class probabilities
probs = model.predict_proba(X_test)
probs
```

Out[33]:
```
array([[0.35146338, 0.64853662],
       [0.90955084, 0.09044916],
       [0.72567333, 0.27432667],
       ...,
       [0.55727384, 0.44272616],
       [0.81207046, 0.18792954],
       [0.74734603, 0.25265397]])
```

In [34]:

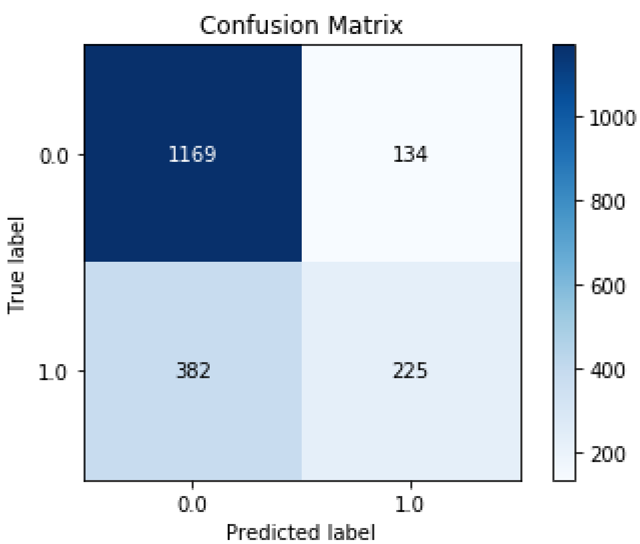
```
print('Evaluating the model')
# generate evaluation metrics
print(metrics.accuracy_score(y_test,predicted))
print(metrics.roc_auc_score(y_test, probs[:, 1]))
print("The accuracy of the model is 73% similar to the training data.")
```

```
Evaluating the model
0.7298429319371728
0.745950606950631
The accuracy of the model is 73% similar to the training data.
```

In [35]:

```
#Using confusion matrix to describe the performance of the classification mo
import scikitplot
scikitplot.metrics.plot_confusion_matrix(y_test,predicted)
```

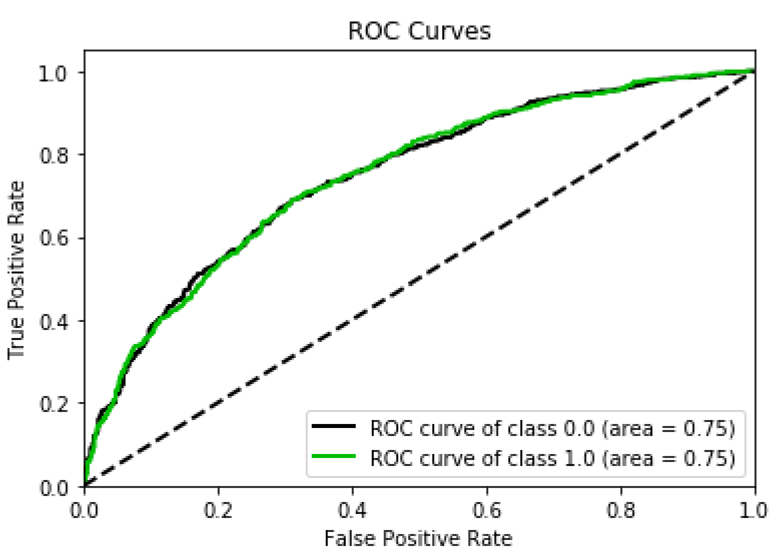
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0xa8acf50>



In [36]:

```
# Plotting the true positive rate (TPR) against the false positive rate (FPR
scikitplot.metrics.plot_roc_curve(y_test, probs,curves=['each_class'])
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0xa9a1dd0>



In [39]:
```
#accuracy report
print(metrics.classification_report(y_test, predicted))
```

```
              precision    recall  f1-score   support

         0.0       0.75      0.90      0.82      1303
         1.0       0.63      0.37      0.47       607

   micro avg       0.73      0.73      0.73      1910
   macro avg       0.69      0.63      0.64      1910
weighted avg       0.71      0.73      0.71      1910
```

In [40]:
```
from sklearn.metrics import  confusion_matrix
cf = confusion_matrix(y_test,predicted)
type(cf)
```

Out[40]:
```
numpy.ndarray
```

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

Out[41]:
```
(2, 2)
```

In [43]:
```
#Calculation of Precision Recall and F1 score
TN = cf[0,0] #True Negative
FP = cf[0,1] #False Positive
FN = cf[1,0] #False Negative
TP = cf[1,1] #True Positive

Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
F1 = (2 *(Precision * Recall)) / (Precision + Recall)
print("Precision : {} , Recall : {}, F1 : {}".format(Precision,Recall,F1))

```

```
Precision : 0.6267409470752089 , Recall : 0.37067545304777594, F1 : 0.465838509
3167702
```

In [44]:
```
#Calculation of True Positive Rate and False Positive Rate
TPR = (TP) / (TP + FN ) #equal to Recall
FPR = FP / (FP + TN )
print("True Positive Rate : {}, False Positive Rate : {}".format(TPR,FPR))

```

```
True Positive Rate : 0.37067545304777594, False Positive Rate : 0.1028396009209
5165
```

In [45]:
```
# evaluate the model using 10-fold cross-validation
scores = cross_val_score(LogisticRegression(), X, y, scoring='accuracy', cv=
scores, scores.mean()
```

Out[45]:
```
(array([0.72100313, 0.70219436, 0.73824451, 0.70597484, 0.70597484,
        0.72955975, 0.7327044 , 0.70440252, 0.75157233, 0.75      ]),
 0.7241630685514876)
```

In [46]:

```
print('Predicting the Probability of an Affair')
print(model.predict_proba(np.array([[1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 3, 25,
print('The predicted probability of an affair is 23%')
```

```
Predicting the Probability of an Affair
[[0.77301481 0.22698519]]
The predicted probability of an affair is 23%
```

In [48]:

```
# Let's predict the probability of an affair for a random woman not present
# She's a 30-year-old teacher who graduated college, has been married for 10
# as strongly religious, rates her marriage as fair, and her husband is a fa
print(model.predict_proba(np.array([[1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 3, 30,
print('The predicted probability of an affair is 31%')
```

```
[[0.68617099 0.31382901]]
The predicted probability of an affair is 31%
```