A marketing agency has many customers that use their service to produce ads for the client/customer websites. They've noticed that they have quite a bit of churn in clients. They basically randomly assign account managers right now, but want you to create a machine learning model that will help predict which customers will churn (stop buying their service) so that they can correctly assign the customers most at risk to churn an account manager. Luckily they have some historical data, can you help them out? Create a classification algorithm that will help classify whether or not a customer churned. Then the company can test this against incoming data for future customers to predict which customers will churn and assign them an account manager.

In [1]:

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

In [2]:

```
df=pd.read_csv(r'C:\Users\chumj\Downloads\customer.csv')
```

In [3]:

```
df.head(3)
```

Out[3]:

	Names	Age	Total_Purchase	Account_Manager	Years	Num_Sites	Onboard_date	Location	Company	Churn
0	Cameron Williams	42.0	11066.80	0	7.22	8.0	2013-08-30 07:00:40	10265 Elizabeth Mission Barkerburgh, AK 89518	Harvey LLC	1
1	Kevin Mueller	41.0	11916.22	0	6.50	11.0	2013-08-13 00:38:46	6157 Frank Gardens Suite 019 Carloshaven, RI 1...	Wilson PLC	1
2	Eric Lozano	38.0	12884.75	0	6.67	12.0	2016-06-29 06:20:07	1331 Keith Court Alyssahaven, DE 90114	Miller, Johnson and Wallace	1

Exploratory Data Analysis

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Names            900 non-null    object 
 1   Age              900 non-null    float64
 2   Total_Purchase   900 non-null    float64
 3   Account_Manager  900 non-null    int64  
 4   Years            900 non-null    float64
 5   Num_Sites        900 non-null    float64
 6   Onboard_date     900 non-null    object 
 7   Location         900 non-null    object 
 8   Company          900 non-null    object 
 9   Churn            900 non-null    int64  
dtypes: float64(4), int64(2), object(4)
memory usage: 70.4+ KB
```

In [5]:

```
len(df)
```

Out[5]:

```
900
```

In [ ]:

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
Names              0
Age                0
Total_Purchase     0
Account_Manager    0
Years              0
Num_Sites          0
Onboard_date       0
Location           0
Company            0
Churn              0
dtype: int64
```

In [7]:

```
df.describe().transpose()
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Age	900.0	41.816667	6.127560	22.0	38.0000	42.000	46.000	65.00
Total_Purchase	900.0	10062.824033	2408.644532	100.0	8497.1225	10045.870	11760.105	18026.01
Account_Manager	900.0	0.481111	0.499921	0.0	0.0000	0.000	1.000	1.00
Years	900.0	5.273156	1.274449	1.0	4.4500	5.215	6.110	9.15
Num_Sites	900.0	8.587778	1.764836	3.0	7.0000	8.000	10.000	14.00
Churn	900.0	0.166667	0.372885	0.0	0.0000	0.000	0.000	1.00

In [8]:

```
df['Account_Manager'].nunique
```

Out[8]:

```
<bound method IndexOpsMixin.nunique of 0      0
1      0
2      0
3      0
4      0
      ..
895    1
896    0
897    0
898    1
899    1
Name: Account_Manager, Length: 900, dtype: int64>
```

In [9]:

```
# This is our soul purpose in this project
df['Churn'].nunique
```

Out[9]:

```
<bound method IndexOpsMixin.nunique of 0      1
```

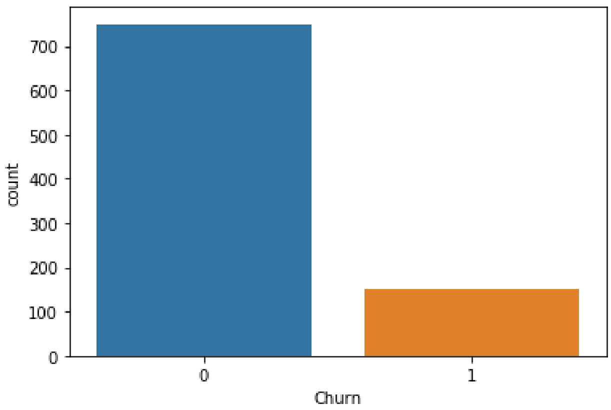
```
<bound method IndexOpsMixin.nunique of 0      1
1      1
2      1
3      1
4      1
      ..
895    0
896    0
897    0
898    0
899    0
Name: Churn, Length: 900, dtype: int64>
```

In [10]:

```python
sns.countplot(df['Churn'])
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dfc966a908>
```



In [11]:

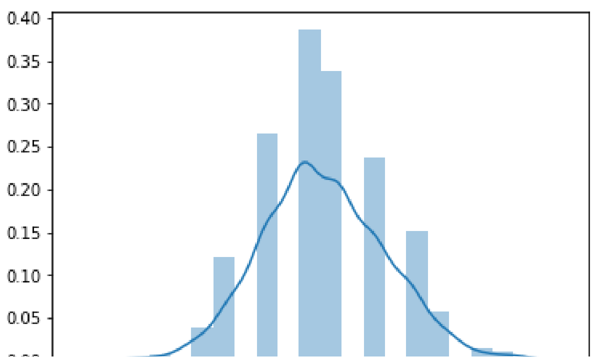
```python
df.corr()
```

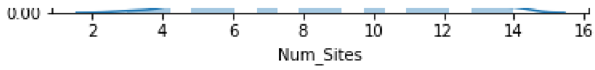
Out[11]:

	Age	Total_Purchase	Account_Manager	Years	Num_Sites	Churn
Age	1.000000	-0.037208	-0.014749	0.005625	-0.006070	0.085926
Total_Purchase	-0.037208	1.000000	0.015856	-0.005623	-0.003390	0.024031
Account_Manager	-0.014749	0.015856	1.000000	0.022930	0.033401	0.070611
Years	0.005625	-0.005623	0.022930	1.000000	0.051642	0.214329
Num_Sites	-0.006070	-0.003390	0.033401	0.051642	1.000000	0.525398
Churn	0.085926	0.024031	0.070611	0.214329	0.525398	1.000000

In [12]:

```python
sns.distplot(df['Num_Sites']);
```





In [13]:

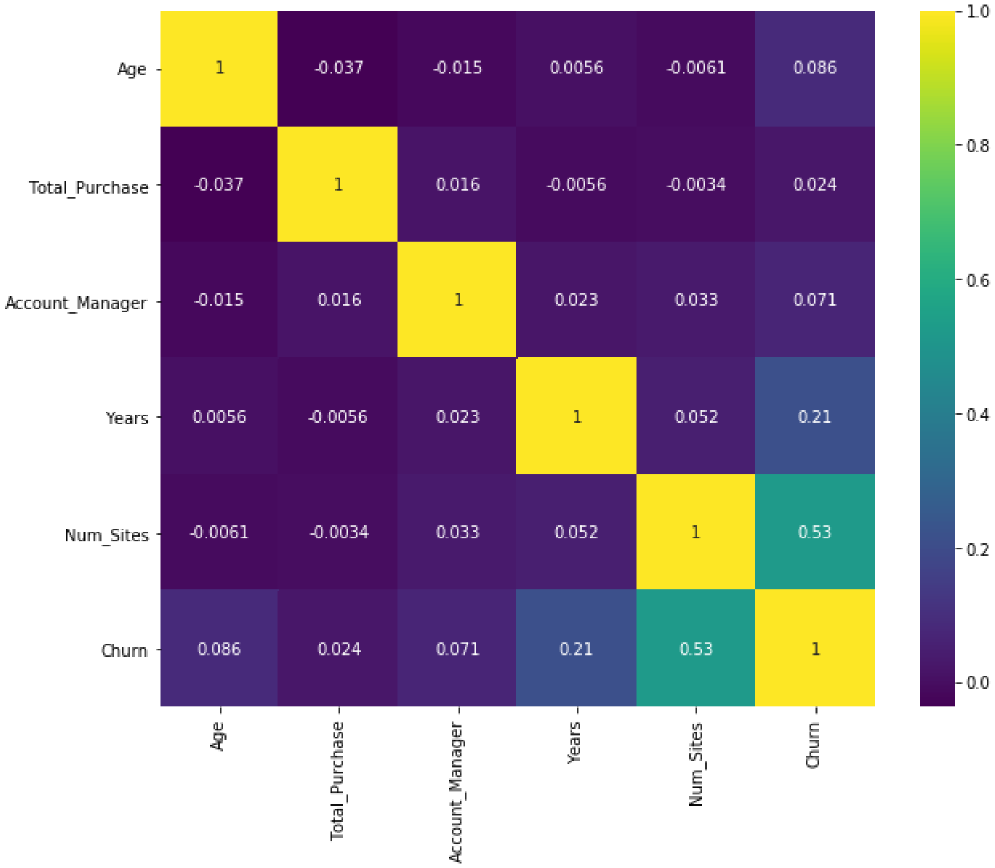
```
df.corr()['Num_Sites'].sort_values(ascending=False)
```

Out[13]:

```
Num_Sites          1.000000
Churn              0.525398
Years              0.051642
Account_Manager    0.033401
Total_Purchase    -0.003390
Age               -0.006070
Name: Num_Sites, dtype: float64
```

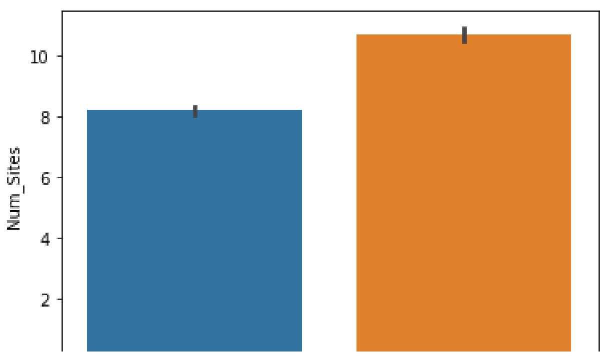
In [14]:

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),annot=True,cmap='viridis');
```



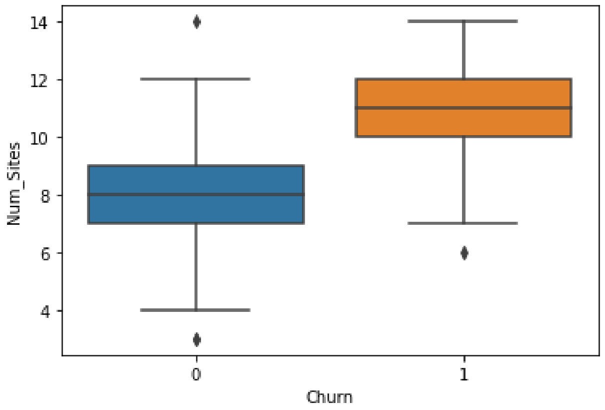
In [15]:

```
sns.barplot(x='Churn',y='Num_Sites',data=df);
```



In [16]:

```
sns.boxplot(x='Churn',y='Num_Sites',data=df);
```



In [17]:

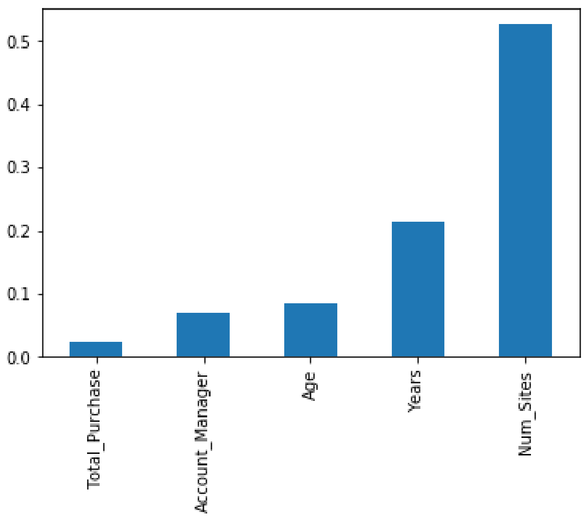
```
df.groupby('Churn')['Num_Sites'].describe()
```

Out[17]:

	count	mean	std	min	25%	50%	75%	max
Churn								
0	750.0	8.173333	1.50720	3.0	7.0	8.0	9.0	14.0
1	150.0	10.660000	1.47839	6.0	10.0	11.0	12.0	14.0

In [18]:

```
df.corr()['Churn'].sort_values().drop('Churn').plot(kind='bar');
```



In [19]:

```
df['Years']
```

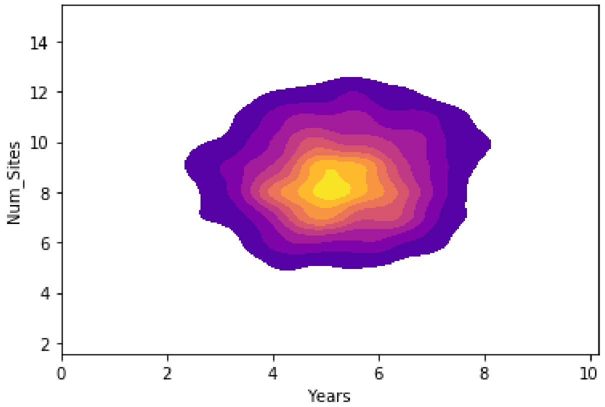
Out[19]:

```
0       7.22
1       6.50
2       6.67
3       6.71
```

```
3       6.71
4       5.56
        ...
895     3.62
896     6.91
897     5.46
898     5.47
899     5.02
Name: Years, Length: 900, dtype: float64
```

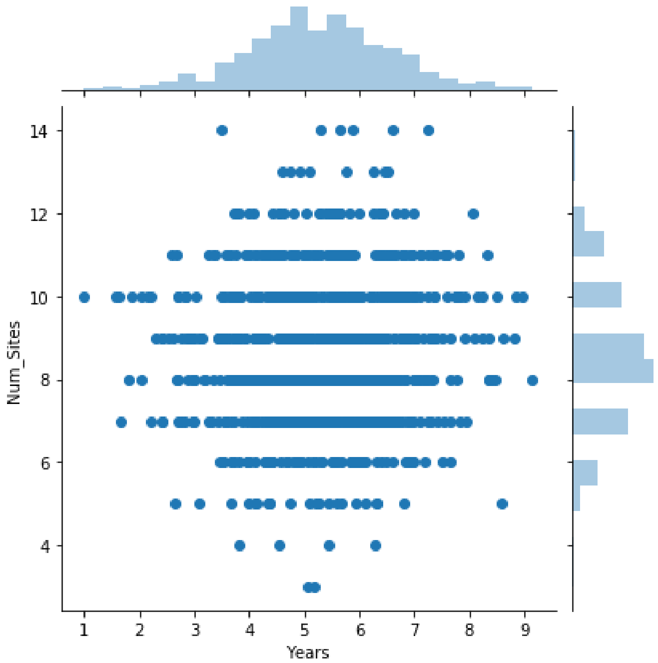
In [20]:

```python
sns.kdeplot(df['Years'],df['Num_Sites'],cmap="plasma", shade=True, shade_lowest=False);
```



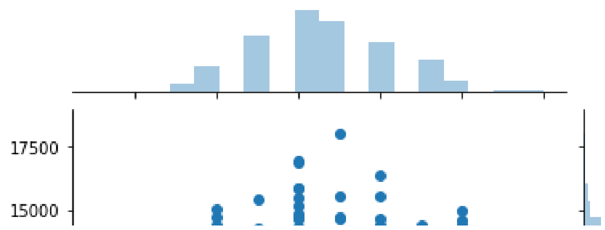
In [21]:

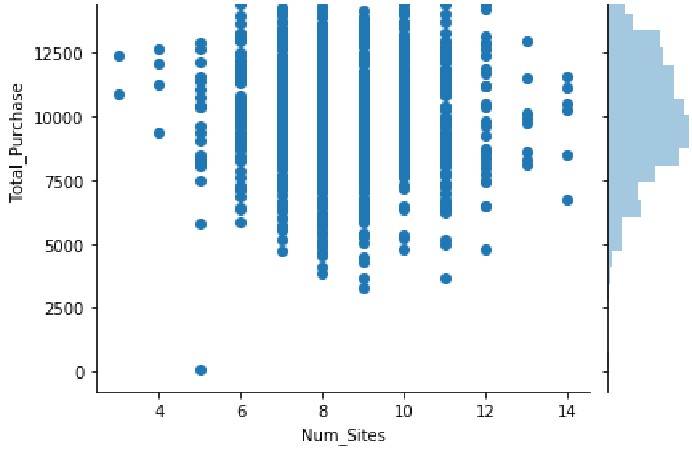
```python
sns.jointplot(x='Years',y='Num_Sites',data=df);
```



In [22]:

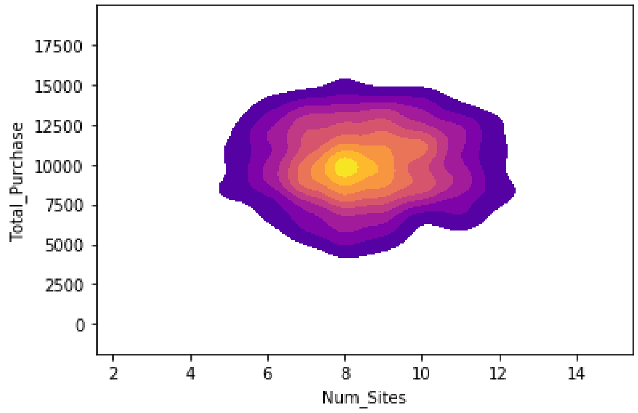
```python
sns.jointplot(x='Num_Sites',y='Total_Purchase',data=df);
```





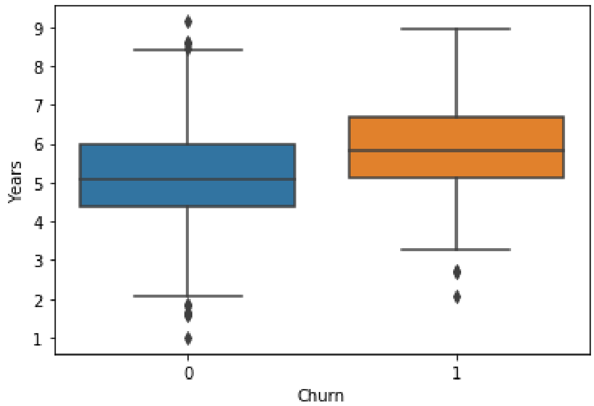
In [23]:

```
sns.kdeplot(df['Num_Sites'],df['Total_Purchase'],cmap="plasma", shade=True, shade_lowest=False);
```



In [24]:

```
sns.boxplot(x='Churn',y='Years',data=df);
```



In [25]:

```
df.groupby('Churn')['Years'].describe()
```

Out[25]:

	count	mean	std	min	25%	50%	75%	max
Churn								
0	750.0	5.151067	1.254465	1.00	4.3625	5.08	5.9900	9.15
1	150.0	5.883600	1.199583	2.05	5.1300	5.80	6.6775	8.97

Data preprocessing

In [26]:

```
df.head(2)
```

Out[26]:

	Names	Age	Total_Purchase	Account_Manager	Years	Num_Sites	Onboard_date	Location	Company	Churn
0	Cameron Williams	42.0	11066.80	0	7.22	8.0	2013-08-30 07:00:40	10265 Elizabeth Mission Barkerburgh, AK 89518	Harvey LLC	1
1	Kevin Mueller	41.0	11916.22	0	6.50	11.0	2013-08-13 00:38:46	6157 Frank Gardens Suite 019 Carloshaven, RI 1...	Wilson PLC	1

In [27]:

```
# Names will not really determine outcome
# Account Manager were randomly assigned,does not determine outcome
#Time is not real necessary,it seems random
#loaction and address not necessary,people can use VPA
df=df.drop(['Names','Account_Manager','Onboard_date','Location','Company'],axis=1)
```

In [28]:

```
df
```

Out[28]:

	Age	Total_Purchase	Years	Num_Sites	Churn
0	42.0	11066.80	7.22	8.0	1
1	41.0	11916.22	6.50	11.0	1
2	38.0	12884.75	6.67	12.0	1
3	42.0	8010.76	6.71	10.0	1
4	37.0	9191.58	5.56	9.0	1
...	...	...	...	...	...
895	42.0	12800.82	3.62	8.0	0
896	52.0	9893.92	6.91	7.0	0
897	45.0	12056.18	5.46	4.0	0
898	51.0	6517.93	5.47	10.0	0
899	39.0	9315.60	5.02	10.0	0

900 rows × 5 columns

In [29]:

```
#This preprocessed data will be used for other Meachine learning models like logestic reg,Decision
tree and random forest,SVM etc.
#ANN and Pyspark in Databrick
df.to_csv(r'C:\Users\chumj\Downloads\churn3.csv',index=False)
```

USING MACHINE LEARNING MODELS

1)LOGISTIC REG

In [30]:

```
#Setting X and y variables for our different models,that is the features(X),and label(y)
X=df.drop('Churn',axis=1)
y=df['Churn']
```

```
y=df['Churn']
```

In [31]:

```
import statsmodels.api as sm
logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.388394
         Iterations 7
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.138
Dependent Variable: Churn            AIC:              707.1094
Date:               2020-08-25 18:35 BIC:              726.3190
No. Observations:   900              Log-Likelihood:   -349.55
Df Model:           3                LL-Null:          -405.51
Df Residuals:       896              LLR p-value:      4.2780e-24
Converged:          1.0000           Scale:            1.0000
No. Iterations:     7.0000
-----------------------------------------------------------------
                   Coef.  Std.Err.    z    P>|z|   [0.025  0.975]
-----------------------------------------------------------------
Age               -0.1124   0.0132 -8.5383 0.0000 -0.1382 -0.0866
Total_Purchase    -0.0002   0.0000 -5.4722 0.0000 -0.0003 -0.0001
Years              0.0479   0.0690  0.6944 0.4874 -0.0873  0.1831
Num_Sites          0.5356   0.0558  9.6066 0.0000  0.4264  0.6449
=================================================================
```

In [32]:

```
from sklearn.model_selection import train_test_split
```

In [33]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
```

In [34]:

```
from sklearn.linear_model import LogisticRegression
```

In [35]:

```
lr=LogisticRegression()
```

In [36]:

```
lr.fit(X_train,y_train)
```

Out[36]:

```
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)
```

In [ ]:

In [37]:

```
predictions = lr.predict(X_test)
```

In [38]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [39]:

```
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
print(accuracy_score(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.95      0.90       223
           1       0.48      0.21      0.29        47

    accuracy                           0.82       270
   macro avg       0.66      0.58      0.60       270
weighted avg       0.79      0.82      0.79       270

[[212  11]
 [ 37  10]]
0.8222222222222222
```

DECISION TREE AND RANDOM FOREST

In [40]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [41]:

```
DT=DecisionTreeClassifier()
```

In [42]:

```
DT.fit(X_train,y_train)
```

Out[42]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, 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='deprecated',
                       random_state=None, splitter='best')
```

In [43]:

```
prediction1=DT.predict(X_test)
```

In [44]:

```
print(classification_report(y_test,prediction1))
print(confusion_matrix(y_test,prediction1))
print(accuracy_score(y_test,prediction1))
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       223
           1       0.66      0.53      0.59        47

    accuracy                           0.87       270
   macro avg       0.78      0.74      0.76       270
weighted avg       0.86      0.87      0.86       270

[[210  13]
 [ 22  25]]
0.8703703703703703
```

0.8703703703703703

In [45]:

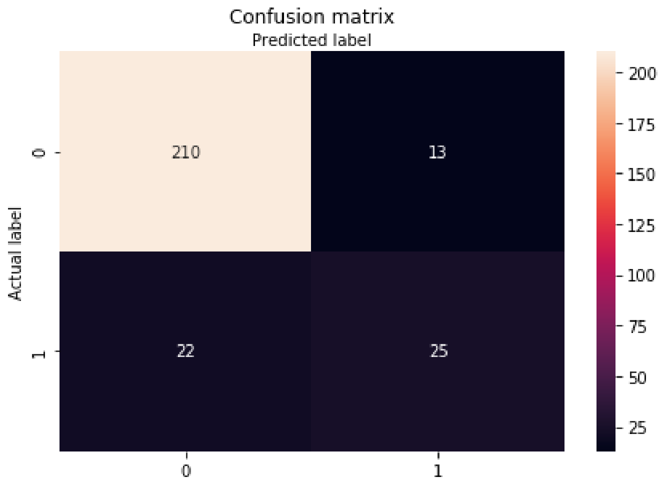
```
cm1=confusion_matrix(y_test,prediction1)
```

In [46]:

```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cm1), annot=True ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[46]:

```
Text(0.5, 257.44, 'Predicted label')
```



RANDOM FOREST

In [47]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [48]:

```
RF=RandomForestClassifier(n_estimators=150)
```

In [49]:

```
RF.fit(X_train,y_train)
```

Out[49]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=150,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [50]:

```
prediction2=RF.predict(X_test)
```

In [51]:

```
print(classification_report(y_test,prediction2))
print(confusion_matrix(y_test,prediction2))
print(accuracy_score(y_test,prediction2))
```

```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92       223
           1       0.65      0.43      0.51        47

    accuracy                           0.86       270
   macro avg       0.77      0.69      0.72       270
weighted avg       0.84      0.86      0.85       270

[[212  11]
 [ 27  20]]
0.8592592592592593
```

In [52]:

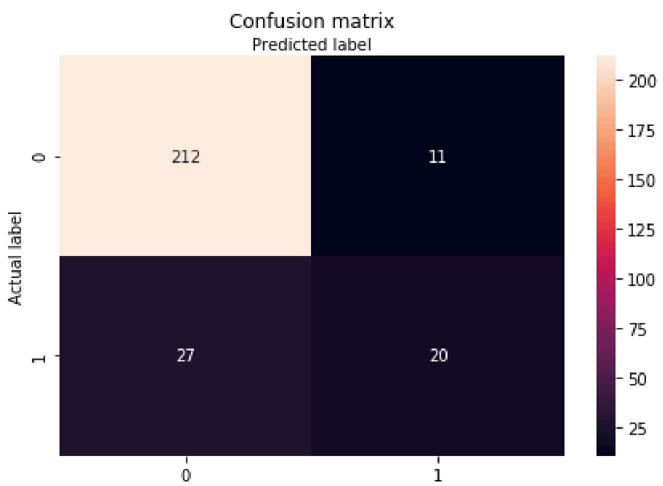
```
cm=confusion_matrix(y_test,prediction2)
```

In [53]:

```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cm), annot=True ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[53]:

```
Text(0.5, 257.44, 'Predicted label')
```



In [61]:

```
#Given a new customer below,will it be churn or not
import random
random_ind=random.randint(0,len(df))
```

In [62]:

```
new_person6=df.drop('Churn',axis=1).iloc[random_ind]
```

In [63]:

```
new_person6
```

Out[63]:

```
Age                   41.00
Total_Purchase     11699.26
Years                  6.99
Num_Sites             12.00
Name: 40, dtype: float64
```

In [64]:

```
RF.predict(new_person6.values.reshape(1,4))
```

Out[64]:

```
array([1], dtype=int64)
```

In [65]:

```
#check if this is a churn or not
df.iloc[random_ind]['Churn']
```

Out[65]:

```
1.0
```