```
In [197]: import pandas as pd
import numpy as np
import statistics

# Visualizations
import seaborn as sns
import matplotlib.pyplot as plt

# Pre-Processing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import RidgeClassifierCV
from sklearn.neural_network import MLPClassifier
from sklearn import ensemble

# Linear Regression
from sklearn import  linear_model
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from scipy import stats

# Evaluation
from sklearn.metrics import confusion_matrix
```

1. Data Pre-Processing/Cleaning

Data Reading and Overview

```
In [112]: df=pd.read_csv("census_income_data.csv")
df.head(10)
```

Out[112]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_p
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	
6	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-family	Black	Female	0	0	
7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	
8	31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-family	White	Female	14084	0	
9	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	5178	0	

```
In [113]: df.shape
```

Out[113]: (48842, 15)

Data Type Overview

In [114]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
age               48842 non-null int64
workclass         48842 non-null object
fnlwgt            48842 non-null int64
education         48842 non-null object
education_num     48842 non-null int64
marital_status    48842 non-null object
occupation        48842 non-null object
relationship      48842 non-null object
race              48842 non-null object
sex               48842 non-null object
capital_gain      48842 non-null int64
capital_loss      48842 non-null int64
hours_per_week    48842 non-null int64
native_country    48842 non-null object
income            48842 non-null object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

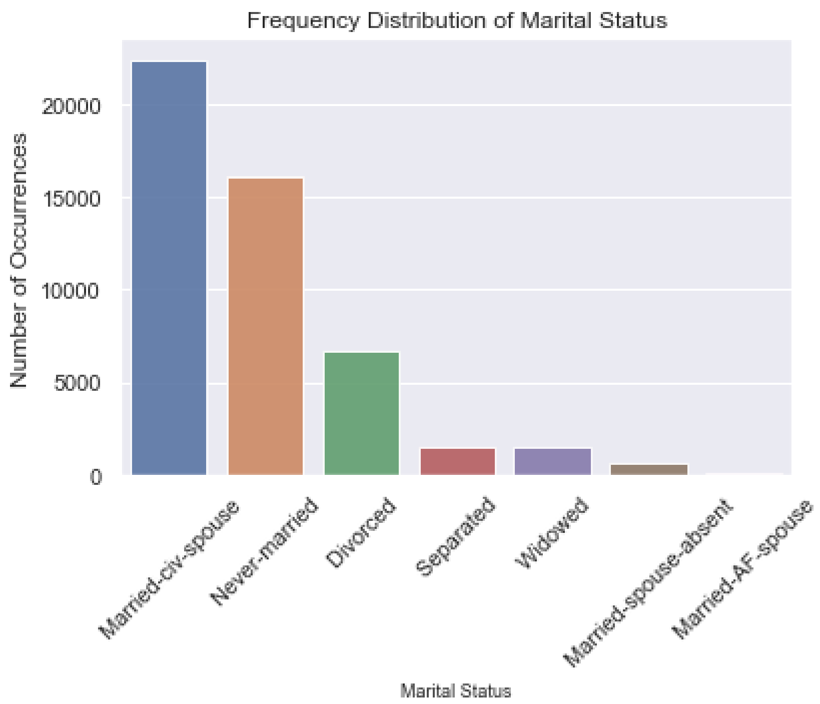
Check for Missing Values

In [115]:
```
df.isnull().values.sum()
```

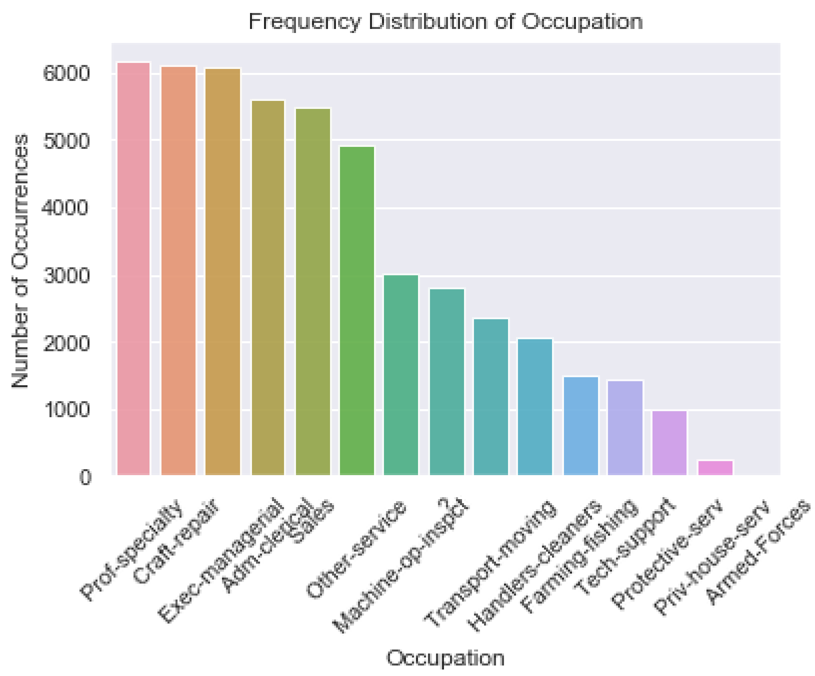
Out[115]:
```
0
```

Visualizations on some features

In [116]:
```
%matplotlib inline
marital_status = df['marital_status'].value_counts()
sns.set(style="darkgrid")
sns.barplot(marital_status.index, marital_status.values, alpha=0.9)
plt.xticks(rotation=45)
plt.title('Frequency Distribution of Marital Status')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Marital Status', fontsize=9)
plt.show()
```



```
In [117]: #occupation
%matplotlib inline
occupation = df['occupation'].value_counts()
sns.set(style="darkgrid")
sns.barplot(occupation.index, occupation.values, alpha=0.9)
plt.xticks(rotation=45)
plt.title('Frequency Distribution of Occupation')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Occupation', fontsize=12)
plt.show()
```



Data Cleaning

Feature Selection

```
In [120]: df=df.drop(["fnlwgt","education"], axis=1)
```

Target Variable Overview

Problem: Same class is coded differently

```
In [121]: df.groupby("income").count()
```

Out[121]:

	age	workclass	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	nat
income												
<=50K	24720	24720	24720	24720	24720	24720	24720	24720	24720	24720	24720	
<=50K.	12435	12435	12435	12435	12435	12435	12435	12435	12435	12435	12435	
>50K	7841	7841	7841	7841	7841	7841	7841	7841	7841	7841	7841	
>50K.	3846	3846	3846	3846	3846	3846	3846	3846	3846	3846	3846	

Cleaning Target Variable

The Problem

```
In [122]: # The Problem
df.groupby("income").mean()
```

Out[122]:

	age	education_num	capital_gain	capital_loss	hours_per_week
income					
<=50K	36.783738	9.595065	148.752468	53.142921	38.840210
<=50K.	37.048010	9.605308	143.547004	56.157780	38.839727
>50K	44.249841	11.611657	4006.142456	195.001530	45.473026
>50K.	44.326833	11.584763	4115.832033	190.526781	45.411856

The Solution

In [123]:
```
#The solution
df["income"] = df["income"].astype('category')
df["income"] = df["income"].cat.codes
df.loc[df.income==1, 'income'] = 0
df.loc[df.income==2, 'income'] = 1
df.loc[df.income==3, 'income'] = 1
df["income"] = df["income"].astype('category')
df.groupby("income").mean()
```

Out[123]:

	age	education_num	capital_gain	capital_loss	hours_per_week
income					
0	36.872184	9.598493	147.010308	54.151931	38.840048
1	44.275178	11.602807	4042.239497	193.528964	45.452896

Transforming categorical features

In [124]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 13 columns):
age               48842 non-null int64
workclass         48842 non-null object
education_num     48842 non-null int64
marital_status    48842 non-null object
occupation        48842 non-null object
relationship      48842 non-null object
race              48842 non-null object
sex               48842 non-null object
capital_gain      48842 non-null int64
capital_loss      48842 non-null int64
hours_per_week    48842 non-null int64
native_country    48842 non-null object
income            48842 non-null category
dtypes: category(1), int64(5), object(7)
memory usage: 4.5+ MB
```

In [125]:
```
categoricals=df.select_dtypes(include="object").columns
df=pd.get_dummies(df,columns=categoricals,drop_first=True)
```

In [126]:
```
columns = df.columns.tolist()
columns.remove("income")
columns.append("income")
```

In [127]:
```
df=df[columns]
df.shape
```

Out[127]: (48842, 85)

Standardize Numerical Variable

In [ ]:
```
numericals=df.select_dtypes(include="int64").columns
df[numericals] = StandardScaler().fit_transform(df[numericals])
```

In [129]:
```
df.head(3)
```

Out[129]:

	age	education_num	capital_gain	capital_loss	hours_per_week	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc	...	native_Pu
0	0.025996	1.136512	0.146932	-0.217127	-0.034087	0	0	0	0	0	...	
1	0.828308	1.136512	-0.144804	-0.217127	-2.213032	0	0	0	0	0	...	
2	-0.046942	-0.419335	-0.144804	-0.217127	-0.034087	0	0	0	1	0	...	

3 rows × 85 columns

Summary Data Processing

- 2 variables are dropped because they are (1) irrelevant: "fnlwgt" or (2) redundant: "education"
- the Target variable is unified and transformed to a dummy
- all numerical variables are standarized
- all categorical variables are transformed into dummy variables with m-1 columns for m manifestations

Linear Regression for Variable Effect Overview

In [99]:

```
X=df.loc[:,df.columns!="income"]
y=df["income"]
X2 = sm.add_constant(X)
est = sm.OLS(y, X2).fit()
print(est.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 income   R-squared:                       0.362
Model:                            OLS   Adj. R-squared:                  0.361
Method:                 Least Squares   F-statistic:                     333.9
Date:                Tue, 15 Oct 2019   Prob (F-statistic):               0.00
Time:                        15:21:55   Log-Likelihood:                -16708.
No. Observations:               48842   AIC:                         3.358e+04
Df Residuals:                   48758   BIC:                         3.432e+04
Df Model:                          83
Covariance Type:            nonrobust
==============================================================================================================
                                                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------------
const                                         -0.4532      0.031    -14.643      0.000      -0.514      -0.393
age                                            0.0026      0.000     17.935      0.000       0.002       0.003
education_num                                  0.0313      0.001     41.091      0.000       0.030       0.033
capital_gain                                8.065e-06   2.11e-07     38.164      0.000    7.65e-06    8.48e-06
capital_loss                                9.285e-05   3.87e-06     24.012      0.000    8.53e-05       0.000
hours_per_week                                 0.0028      0.000     20.261      0.000       0.003       0.003
workclass_ Federal-gov                         0.1119      0.011     10.144      0.000       0.090       0.134
workclass_ Local-gov                           0.0164      0.009      1.747      0.081      -0.002       0.035
workclass_ Never-worked                        0.0739      0.108      0.684      0.494      -0.138       0.286
workclass_ Private                             0.0386      0.007      5.167      0.000       0.024       0.053
workclass_ Self-emp-inc                        0.0958      0.011      8.730      0.000       0.074       0.117
workclass_ Self-emp-not-inc                   -0.0313      0.009     -3.445      0.001      -0.049      -0.013
workclass_ State-gov                          -0.0008      0.010     -0.075      0.940      -0.021       0.020
workclass_ Without-pay                        -0.0601      0.071     -0.844      0.399      -0.200       0.080
marital_status_ Married-AF-spouse              0.0833      0.059      1.407      0.159      -0.033       0.199
marital_status_ Married-civ-spouse             0.1257      0.019      6.557      0.000       0.088       0.163
marital_status_ Married-spouse-absent          0.0432      0.014      3.003      0.003       0.015       0.071
marital_status_ Never-married                 -0.0027      0.006     -0.470      0.638      -0.014       0.009
marital_status_ Separated                      0.0198      0.010      2.033      0.042       0.001       0.039
marital_status_ Widowed                        0.0190      0.010      1.888      0.059      -0.001       0.039
occupation_ Adm-clerical                      -0.0023      0.008     -0.303      0.762      -0.017       0.013
occupation_ Armed-Forces                       0.0336      0.084      0.398      0.691      -0.132       0.199
occupation_ Craft-repair                      -0.0186      0.008     -2.441      0.015      -0.034      -0.004
occupation_ Exec-managerial                    0.1368      0.008     17.926      0.000       0.122       0.152
occupation_ Farming-fishing                   -0.0973      0.011     -9.133      0.000      -0.118      -0.076
occupation_ Handlers-cleaners                 -0.0536      0.010     -5.584      0.000      -0.072      -0.035
occupation_ Machine-op-inspct                 -0.0476      0.009     -5.463      0.000      -0.065      -0.031
occupation_ Other-service                     -0.0218      0.008     -2.786      0.005      -0.037      -0.006
occupation_ Priv-house-serv                    0.0158      0.022      0.712      0.476      -0.028       0.059
occupation_ Prof-specialty                     0.1057      0.008     13.461      0.000       0.090       0.121
occupation_ Protective-serv                    0.0558      0.012      4.507      0.000       0.032       0.080
occupation_ Sales                              0.0432      0.008      5.621      0.000       0.028       0.058
occupation_ Tech-support                       0.0620      0.011      5.846      0.000       0.041       0.083
occupation_ Transport-moving                  -0.0412      0.009     -4.439      0.000      -0.059      -0.023
relationship_ Not-in-family                   -0.1613      0.019     -8.447      0.000      -0.199      -0.124
relationship_ Other-relative                  -0.1373      0.019     -7.315      0.000      -0.174      -0.100
relationship_ Own-child                       -0.1326      0.019     -6.952      0.000      -0.170      -0.095
relationship_ Unmarried                       -0.1514      0.020     -7.645      0.000      -0.190      -0.113
relationship_ Wife                             0.0956      0.009     10.972      0.000       0.079       0.113
race_ Asian-Pac-Islander                       0.0478      0.021      2.320      0.020       0.007       0.088
race_ Black                                    0.0299      0.017      1.803      0.071      -0.003       0.062
race_ Other                                    0.0354      0.023      1.509      0.131      -0.011       0.081
race_ White                                    0.0476      0.016      2.998      0.003       0.016       0.079
sex_ Male                                      0.0553      0.005     11.963      0.000       0.046       0.064
native_country_ Cambodia                       0.1231      0.066      1.859      0.063      -0.007       0.253
native_country_ Canada                         0.0685      0.028      2.453      0.014       0.014       0.123
native_country_ China                         -0.0509      0.035     -1.463      0.144      -0.119       0.017
native_country_ Columbia                      -0.0858      0.039     -2.207      0.027      -0.162      -0.010
native_country_ Cuba                           0.0172      0.031      0.549      0.583      -0.044       0.079
native_country_ Dominican-Republic            -0.0056      0.036     -0.157      0.875      -0.076       0.065
native_country_ Ecuador                       -0.0084      0.052     -0.161      0.872      -0.111       0.094
native_country_ El-Salvador                    0.0597      0.030      1.987      0.047       0.001       0.118
native_country_ England                        0.0900      0.033      2.770      0.006       0.026       0.154
native_country_ France                         0.1148      0.057      2.029      0.042       0.004       0.226
native_country_ Germany                        0.0310      0.027      1.169      0.243      -0.021       0.083
native_country_ Greece                         0.0109      0.050      0.217      0.829      -0.087       0.109
native_country_ Guatemala                      0.0615      0.039      1.597      0.110      -0.014       0.137
native_country_ Haiti                          0.0353      0.041      0.854      0.393      -0.046       0.116
native_country_ Holand-Netherlands            -0.1589      0.341     -0.465      0.642      -0.828       0.510
native_country_ Honduras                       0.0444      0.077      0.575      0.566      -0.107       0.196
native_country_ Hong                          -0.0328      0.064     -0.512      0.609      -0.158       0.093
native_country_ Hungary                        0.0446      0.079      0.564      0.573      -0.111       0.200
native_country_ India                          0.0157      0.031      0.499      0.618      -0.046       0.077
native_country_ Iran                           0.0207      0.046      0.451      0.652      -0.069       0.111
native_country_ Ireland                        0.1228      0.057      2.142      0.032       0.010       0.235
native_country_ Italy                          0.0802      0.035      2.267      0.023       0.011       0.150
native_country_ Jamaica                        0.0247      0.035      0.698      0.485      -0.045       0.094
native_country_ Japan                          0.0413      0.038      1.092      0.275      -0.033       0.116
native_country_ Laos                          -0.0794      0.073     -1.087      0.277      -0.222       0.064
native_country_ Mexico                         0.0234      0.017      1.413      0.158      -0.009       0.056
native_country_ Nicaragua                     -0.0301      0.050     -0.599      0.549      -0.128       0.068
native_country_ Outlying-US(Guam-USVI-etc)    -0.0792      0.072     -1.099      0.272      -0.220       0.062
native_country_ Peru                          -0.0574      0.052     -1.110      0.267      -0.159       0.044
native_country_ Philippines                    0.0408      0.025      1.607      0.108      -0.009       0.091
```

```
native_country_ Poland                 -0.0167      0.038     -0.434      0.664      -0.092       0.059
native_country_ Portugal                0.0570      0.043      1.313      0.189      -0.028       0.142
native_country_ Puerto-Rico            -0.0030      0.028     -0.107      0.915      -0.058       0.052
native_country_ Scotland               -0.1063      0.075     -1.411      0.158      -0.254       0.041
native_country_ South                  -0.0836      0.036     -2.351      0.019      -0.153      -0.014
native_country_ Taiwan                  0.0064      0.045      0.142      0.887      -0.082       0.095
native_country_ Thailand               -0.0673      0.064     -1.052      0.293      -0.193       0.058
native_country_ Trinadad&Tobago        -0.0870      0.067     -1.303      0.193      -0.218       0.044
native_country_ United-States           0.0287      0.012      2.404      0.016       0.005       0.052
native_country_ Vietnam                -0.0592      0.040     -1.473      0.141      -0.138       0.020
native_country_ Yugoslavia              0.0989      0.072      1.372      0.170      -0.042       0.240
==============================================================================
Omnibus:                     2236.226   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2489.501
Skew:                           0.541   Prob(JB):                         0.00
Kurtosis:                       2.770   Cond. No.                     1.49e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.25e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

2. Prediction

This is a supervised 2-class classification problem.

2.1 Comparison of Default Models

Five different models in their default setting are trained on a train set (80%) and evaluated on the test set (20%) with their prediction accuracy.

In [135]:
```python
def prediction(df, model="logit", seed=1):
    '''This function returns the accuracy of a
    prediction dependent on the model used.'''

    # Create Train and Test Sets
    X=df.loc[:,df.columns!="income"]
    y=df["income"].values.ravel()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = seed)

    # Select Model
    if model=="Logit":
        model= LogisticRegression(random_state=1, solver='liblinear')#regularization is applied by default
    elif model=="Naive Bayes":
        model= GaussianNB()
    elif model=="Multi-layer Perceptron":
        model= MLPClassifier()
    elif model=="Ridge Classifier":
        model= RidgeClassifierCV()
    elif model=="Gradient Boosting":
        model= ensemble.GradientBoostingClassifier()

    # Fit Model and Return Accuracy
    model.fit(X_train, y_train)
    accuracy=model.score(X_test,y_test)
    return(accuracy)
```

The results are illustrated in a table

In [131]:
```python
def create_results(result_table,n,model):
    '''This functions shows the mean accuracy (and its variance)
    over n different train-test-splits.'''

    acc=[]
    for seed in range(n):
        acc_n=prediction(df,seed=seed,model=model)
        acc.append(acc_n)
    mean = sum(acc)/len(acc)
    sd = statistics.stdev(acc)
    result_table.loc[model,"Mean"] = mean
    result_table.loc[model,"SD"] = sd
    return(result_table)
```

In [ ]:
```
columns=["Mean", "SD"]
models=["Naive Bayes",
        "Logit",
        "Ridge Classifier",
        "Gradient Boosting",
        "Multi-layer Perceptron"]
result_df=pd.DataFrame(columns=columns, index=models)
for model in models:
    results=create_results(result_df,10,model)
```

Results with default parameters

In [133]:
```
result_df
```

Out[133]:

	Mean	SD
Naive Bayes	0.58007	0.026521
Logit	0.851377	0.00390552
Ridge Classifier	0.840874	0.00274896
XGB	0.867407	0.00288284
MLP	0.850537	0.00539972

2.2 Manual Optimization of Parameters for Gradient Boost Model

Since the Gradient Boosting Classifier has 2 percentage points higher accuracy than the other classifiers in its default settings, this classifier is further optimized. (It is possible that another classifier is even superior when optimized in its parameters.)

The following parameters are chosen for a simultaneous optimization

- subsample (default 1.0) "Choosing subsample < 1.0 leads to a reduction of variance and an increase in bias"
- max_depth (default 3) "The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables."

from its [scikit-learn documentation (https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)](https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)

In [166]:
```
subsamples=["0.3", "0.6", "1"]
max_depths=["1","2","3","6","9"]
parameter_selection_1=pd.DataFrame(columns=subsamples, index=max_depths)
parameter_selection_1
```

Out[166]:

	0.3	0.6	1
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
6	NaN	NaN	NaN
9	NaN	NaN	NaN

In [ ]:
```
def gradient_boost(dataframe,n,subsample=subsample, max_depth = max_depth):
    '''This function returns the accuracy of a
    gradient boost prediction dependent on the
    parameters subsample and max_depth used.'''
    X   = df.loc[:,df.columns!="income"]
    y   = df["income"].values.ravel()
    acc = []
    for seed in range(n):
        # Create Train and Test Sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = seed)

        # Set Parameters
        model = ensemble.GradientBoostingClassifier(subsample=subsample,max_depth=max_depth)

        # Fit Model and Return Accuracy
        model.fit(X_train, y_train)
        accuracy = model.score(X_test,y_test)
        acc.append(accuracy)
    mean_accuracy = sum(acc)/len(acc)
    return(mean_accuracy)
```

In [ ]:
```
for subsample in subsamples:
    print(subsample)
        for max_depth in max_depths:
            subsample_num = float(subsample)
            max_depth_num = float(max_depth)
            result=gradient_boost(dataframe=df, n=3, subsample=subsample_num, max_depth=max_depth_num)
            print(result)
            parameter_selection_1.loc[max_depth,subsample]=result
```

In [187]:
```
parameter_selection_1
```

Out[187]:

	0.3	0.6	1
1	0.85471	0.855222	0.855666
2	0.861876	0.861774	0.861228
3	0.866073	0.867677	0.867506
6	0.870782	0.873	0.873921
9	0.868496	0.870202	0.871259

Zooming closer to the optimal area

In [ ]:
```
subsamples_2 = ["0.5","0.75","0.9","1"]
max_depths_2 = ["5","6","7","8"]
parameter_selection_2=pd.DataFrame(columns=subsamples_2, index=max_depths_2)
for subsample in subsamples_2:
    print(subsample)
    for max_depth in max_depths_2:
        subsample_num = float(subsample)
        max_depth_num = int(max_depth)
        result=gradient_boost(dataframe=df, n=3, subsample=subsample_num, max_depth=max_depth_num)
        print(result)
        parameter_selection_2.loc[max_depth,subsample]=result
```

In [191]:
```
parameter_selection_2
```

Out[191]:

	0.5	0.75	0.9	1
5	0.872215	0.872863	0.873341	0.872761
6	0.872283	0.873307	0.873546	0.873443
7	0.873068	0.873136	0.873716	0.873068
8	0.870782	0.871737	0.872693	0.872658

Result

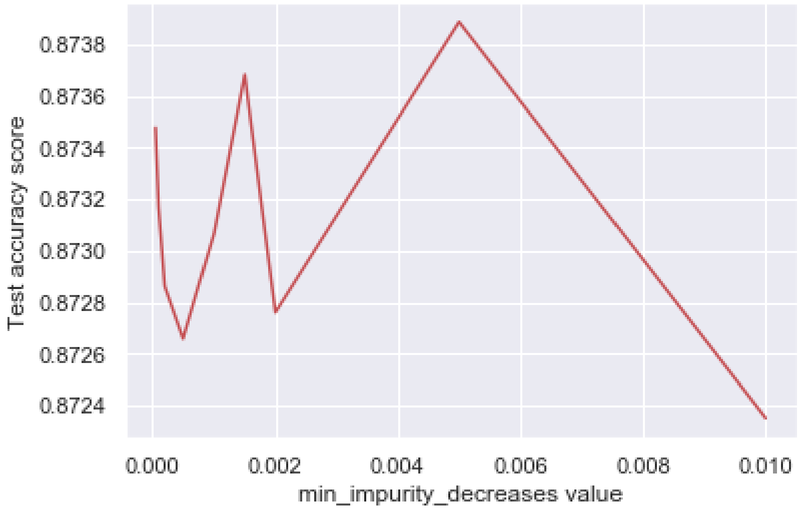
The results do not change much in these areas of the two considered parameters. We chose 0.9 for subsample and and 7 for the maximal depth as this results in the highest accuracy of **0.8737**.

Iterative Optimization Approach

After the simultaneous optimization in a two-dimensional space, another, third, parameter (*min_impurity_decrease*) is optimized at the optimum of the first two parameters. (It is possible that the overall optimum of these three parameters is different from this result. The computational costs are significantly lower for the iterative optimization though.)

In [196]:
```
min_impurity_decreases = [0.00005,0.0001,0.0002,0.0005,0.001,0.0015,0.002,0.005,0.01]
test_acc = []
for min_impurity_decrease in min_impurity_decreases:
    X  = df.loc[:,df.columns!="income"]
    y  = df["income"].values.ravel()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
    model = ensemble.GradientBoostingClassifier(subsample=0.9, max_depth=7, min_impurity_decrease=min_impurity_decreas
e)
    model.fit(X_train, y_train)
    accuracy = model.score(X_test,y_test)
    test_acc.append(accuracy)
```

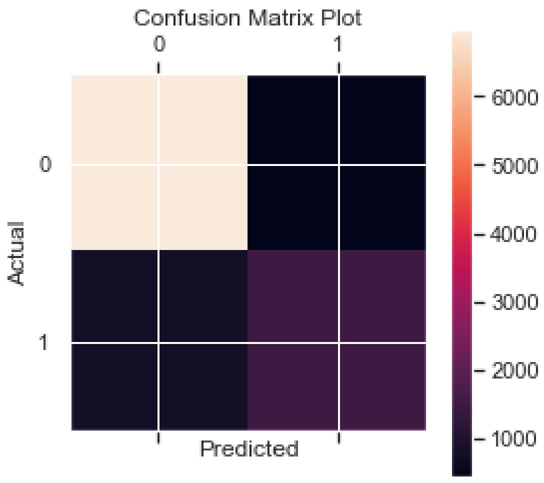
In [203]:
```
plt.plot(min_impurity_decreases, test_acc, "r")
plt.ylabel("Test accuracy score")
plt.xlabel("min_impurity_decreases value")
plt.show()
```



There does not seem to be a significant change corresponding to the min_impurity_decrease variable. The fluctuations indicate a lack of significance. Nevertheless, we chose 0.005 as it leads to the maximal accuracy.

Final Model and Result

In [209]:
```
model = ensemble.GradientBoostingClassifier(subsample=0.9, max_depth=7, min_impurity_decrease=0.005)
y_pred = model.fit(X_train, y_train).predict(X_test)
cm = confusion_matrix(y_test, y_pred)
plt.matshow(cm)
plt.title('Confusion Matrix Plot',y=1.1)
plt.colorbar()
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show();
```



In [207]:
```
accuracy = model.score(X_test,y_test)
accuracy
```

Out[207]: 0.8725560446309756