# 1.Exploratory data analysis

We import the useful libraries

In [116]:
```
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings('ignore', category=DeprecationWarning)

import pandas as pd
pd.options.display.max_columns = 100

from matplotlib import pyplot as plt
import numpy as np

import seaborn as sns

import pylab as plot
params = {
    'axes.labelsize': "large",
    'xtick.labelsize': 'x-large',
    'legend.fontsize': 20,
    'figure.dpi': 150,
    'figure.figsize': [25, 7]
}
plot.rcParams.update(params)
#Importing Data Analysis Libs
import warnings
warnings.filterwarnings('ignore')
```

Now let's start by loading the training & testing datasets.

In [117]:
```
#Getting .csv files
train = pd.read_csv('./Downloads/exercise_02_train.csv')
test = pd.read_csv('./Downloads/exercise_02_test.csv')
```

In [118]:
```
print(train.shape)
print(test.shape)
```

```
(40000, 101)
(10000, 100)
```

In [119]:
```
train.head()
```

Out[119]:

| | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.198560 | 74.425320 | 67.627745 | -3.095111 | -6.822327 | 19.048071 | -0.362378 | -10.699174 | -2 |
| **1** | -29.662621 | 24.320711 | -48.205182 | 1.430339 | -6.552206 | 4.263074 | 6.551412 | 4.265483 | |
| **2** | 15.493759 | -66.160459 | 50.512903 | -2.265792 | 14.428578 | 2.509323 | -6.707536 | 3.820842 | -1 |
| **3** | -19.837651 | 33.210943 | 53.405563 | 1.079462 | 11.364251 | -1.064581 | 9.308857 | 9.266076 | 1 |
| **4** | 11.896655 | -26.717872 | -17.758176 | 1.692017 | 21.553537 | -5.852097 | -0.857435 | -2.186940 | 1 |

5 rows × 101 columns

In [120]:
```
##Statistic Summary

 # Train dataset
train.describe().transpose().head()
```

Out[120]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **x0** | 39989.0 | 3.446069 | 16.247547 | -60.113902 | -7.602474 | 3.448865 | 14.266716 | 75.311659 |
| **x1** | 39990.0 | -7.788884 | 37.014862 | -157.341119 | -32.740989 | -8.019993 | 16.853383 | 153.469221 |
| **x2** | 39992.0 | 1.706058 | 38.385085 | -163.339956 | -24.141605 | 1.963977 | 27.516500 | 154.051060 |
| **x3** | 39991.0 | -0.072972 | 1.503243 | -6.276969 | -1.088182 | -0.062389 | 0.940612 | 5.837559 |
| **x4** | 39992.0 | 0.123077 | 16.289994 | -61.632319 | -10.896241 | 0.104277 | 11.078565 | 65.949709 |

In [121]:
```
#test data set
test.describe().transpose().head()
```

Out[121]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **x0** | 9997.0 | 3.493920 | 16.373366 | -62.382009 | -7.541493 | 3.596409 | 14.430014 | 64.315365 |
| **x1** | 9998.0 | -8.096684 | 37.195123 | -153.367104 | -32.842485 | -8.463437 | 16.340463 | 120.366215 |
| **x2** | 9999.0 | 1.287510 | 38.122644 | -152.919587 | -24.148737 | 1.891110 | 27.417417 | 142.900207 |
| **x3** | 9996.0 | -0.074937 | 1.489704 | -6.467378 | -1.096175 | -0.085627 | 0.950085 | 5.974718 |
| **x4** | 10000.0 | 0.131657 | 16.346463 | -56.532024 | -10.845661 | 0.289724 | 11.217710 | 69.311797 |

In [122]:
```
#Data Types
train.dtypes.head()
```

Out[122]:
```
x0    float64
x1    float64
x2    float64
x3    float64
x4    float64
dtype: object
```

Let check the Data Frame

In [123]:
```
train['x93'].unique()
test['x93'].unique()
```

Out[123]:
```
array(['america', 'asia', 'euorpe', nan], dtype=object)
```

In [124]:
```
train['x34'].unique()
test['x34'].unique()
```

Out[124]:
```
array(['volkswagon', 'bmw', 'Toyota', 'tesla', 'Honda', 'chrystler',
       'ford', 'nissan', nan, 'chevrolet', 'mercades'], dtype=object)
```

In [125]:
```
train['x35'].unique()
test['x35'].unique()
```

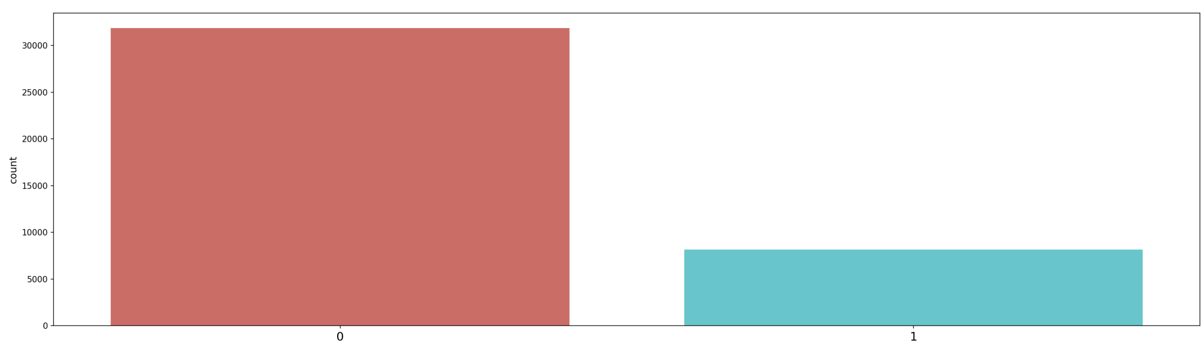
Out[125]:
```
array(['wed', 'thurday', 'wednesday', 'thur', 'tuesday', 'friday',
       'monday', 'fri'], dtype=object)
```

In [126]:
```
train['x35']=train['x35'].replace(to_replace=['wed','thur','fri'],value=
['wednesday','thurday','friday'])
test['x35']=test['x35'].replace(to_replace=['wed','thur','fri'],value=[
'wednesday','thurday','friday'])
```

In [127]:
```
train['x68'].unique()
test['x68'].unique()
```

Out[127]:
```
array(['Aug', 'Jun', 'sept.', 'July', 'Apr', 'May', 'Oct', 'Nov', 'Ma
r',
       'January', 'Dev', 'Feb', nan], dtype=object)
```

In [128]:
```
sns.countplot(x='y',data=train,palette='hls')
plt.show()
plt.savefig('count_plot')
```

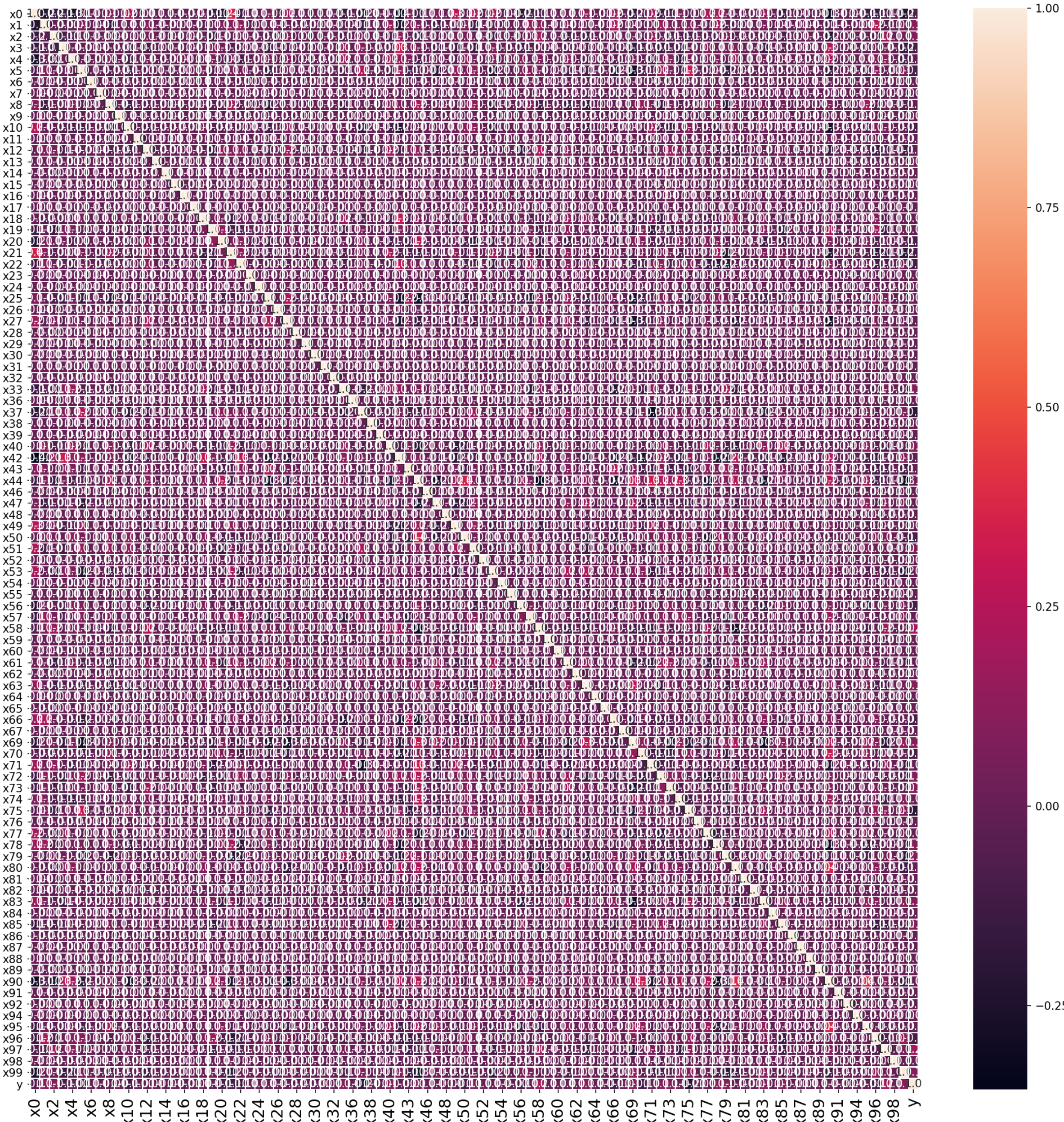



```
<Figure size 3750x1050 with 0 Axes>
```

Checking the correlation between features

In [129]:
```
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(train.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

Out[129]: <matplotlib.axes._subplots.AxesSubplot at 0x1a329725c0>



In [130]:
```
# Identify correlated variables
# Threshold for removing correlated variables
threshold=0.9
#Absolute value correlation matrix
corr_matrix=train.corr().abs()
corr_matrix.head()
```

Out[130]:

| | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **x0** | 1.000000 | 0.219011 | 0.156642 | 0.148241 | 0.260961 | 0.077997 | 0.001033 | 0.003080 | 0.110834 | 0 |
| **x1** | 0.219011 | 1.000000 | 0.025074 | 0.032906 | 0.019089 | 0.082687 | 0.004238 | 0.004183 | 0.111920 | 0 |
| **x2** | 0.156642 | 0.025074 | 1.000000 | 0.088827 | 0.067174 | 0.002726 | 0.005451 | 0.002137 | 0.000051 | 0 |
| **x3** | 0.148241 | 0.032906 | 0.088827 | 1.000000 | 0.018954 | 0.007140 | 0.008659 | 0.004365 | 0.023541 | 0 |
| **x4** | 0.260961 | 0.019089 | 0.067174 | 0.018954 | 1.000000 | 0.004789 | 0.000780 | 0.002464 | 0.072286 | 0 |

In [131]:
```
# Upper triangle of correlations
upper=corr_matrix.where(np.triu(np.ones(corr_matrix.shape),k=1).astype(n
p.bool))
upper.head()
```

Out[131]:

| | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **x0** | NaN | 0.219011 | 0.156642 | 0.148241 | 0.260961 | 0.077997 | 0.001033 | 0.003080 | 0.110834 | 0.0047 |
| **x1** | NaN | NaN | 0.025074 | 0.032906 | 0.019089 | 0.082687 | 0.004238 | 0.004183 | 0.111920 | 0.0059 |
| **x2** | NaN | NaN | NaN | 0.088827 | 0.067174 | 0.002726 | 0.005451 | 0.002137 | 0.000051 | 0.0032 |
| **x3** | NaN | NaN | NaN | NaN | 0.018954 | 0.007140 | 0.008659 | 0.004365 | 0.023541 | 0.0053 |
| **x4** | NaN | NaN | NaN | NaN | NaN | 0.004789 | 0.000780 | 0.002464 | 0.072286 | 0.0022 |

In [132]:
```
# select columns with correlation above threshold
to_drop=[column for column in upper.columns if any(upper[column]>thresho
ld)]
print('There are %d column to remove.'% (len(to_drop)))
```

```
There are 0 column to remove.
```

Data Cleaning Steps

Now let's drop null values using the pandas dropna function.

In [133]:
```
# Train missing values(in Percent)
train_missing=(train.isnull().sum() /len(train)).sort_values(ascending=F
alse)
train_missing.head()
```

Out[133]:
```
x96    0.000375
x63    0.000350
x13    0.000350
x18    0.000350
x85    0.000350
dtype: float64
```

In [134]:
```
#testing missing values(in percent)
test_missing=(test.isnull().sum() /len(test)).sort_values(ascending=Fals
e)
test_missing.head()
```

Out[134]:
```
x55    0.0006
x5     0.0005
x15    0.0005
x87    0.0005
x79    0.0004
dtype: float64
```

In [135]:
```
before_rows = train.shape[0]
print(before_rows)
```

```
40000
```

In [136]:
```
train = train.dropna()
test=test.dropna()
```

In [137]:
```
after_rows = train.shape[0]
print(after_rows)
test=test.dropna()
```

```
39182
```

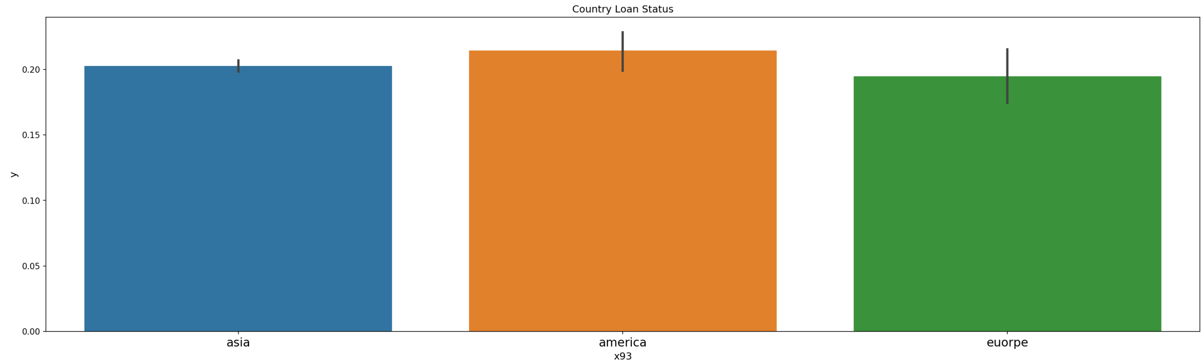
How many rows dropped due to cleaning?

In [138]:
```
before_rows - after_rows
```

Out[138]:
```
818
```

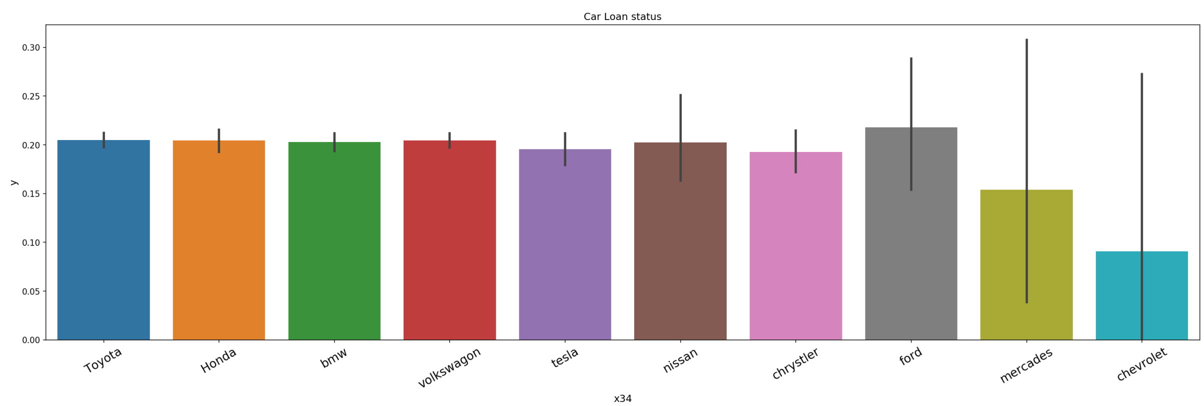
In [139]:
```
# Plot
ax=sns.barplot(train['x93'],train['y'],data=train)
ax.set_title('Country Loan Status')
```

Out[139]: Text(0.5, 1.0, 'Country Loan Status')




In [140]:
```
# Plot
ax=sns.barplot(train['x34'],train['y'],data=train)
ax.set_title('Car Loan status')
ax.set_xticklabels(ax.get_xticklabels(),rotation=30)
```

Out[140]:
```
[Text(0, 0, 'Toyota'),
 Text(0, 0, 'Honda'),
 Text(0, 0, 'bmw'),
 Text(0, 0, 'volkswagon'),
 Text(0, 0, 'tesla'),
 Text(0, 0, 'nissan'),
 Text(0, 0, 'chrystler'),
 Text(0, 0, 'ford'),
 Text(0, 0, 'mercades'),
 Text(0, 0, 'chevrolet')]
```




DataFrame Transformation

In [141]:
```
#Copying Dataframe
dfT = train
```

In [142]:
```
#Label Encoding for x93 Column
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(dfT['x93'])
#list(le.classes_)
dfT['x93'] = le.transform(dfT['x93'])


le.fit(test['x93'])
#list(le.classes_)
test['x93'] = le.transform(test['x93'])
```

In [143]:
```
#Label Encoding for x34 Column
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(dfT['x34'])
#list(le.classes_)
dfT['x34'] = le.transform(dfT['x34'])


le.fit(test['x34'])
#list(le.classes_)
test['x34'] = le.transform(test['x34'])
```

In [144]:
```
#Label Encoding for x35 Column
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(dfT['x35'])
#list(le.classes_)
dfT['x35'] = le.transform(dfT['x35'])


le.fit(test['x35'])
#list(le.classes_)
test['x35'] = le.transform(test['x35'])
```

In [145]:
```
#Label Encoding for x68 Column
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(dfT['x68'])
#list(le.classes_)
dfT['x68'] = le.transform(dfT['x68'])

le.fit(test['x68'])
#list(le.classes_)
test['x68'] = le.transform(test['x68'])
```

In [146]:
```
#Label Encoding for x45 Column
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(dfT['x45'])
#list(le.classes_)
dfT['x45'] = le.transform(dfT['x45'])

le.fit(test['x45'])
#list(le.classes_)
test['x45'] = le.transform(test['x45'])
```

In [147]:
```
#Label Encoding for x41 Column
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(dfT['x41'])
#list(le.classes_)
dfT['x41'] = le.transform(dfT['x41'])

le.fit(test['x41'])
#list(le.classes_)
test['x41'] = le.transform(test['x41'])
```

# 2. Modeling

There is a wide variety of models to use, from logistic regression to decision trees and more sophisticated ones such as random forests and gradient boosted trees.

Let's start by importing the useful libraries.

In [148]:
```
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble.gradient_boosting import GradientBoostingClassifie
r
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression, LogisticRegressionC
V
```

In [149]:
```
x = dfT.drop("y", axis=1) # train dependent variables
y = dfT["y"]  # targets
```

# Standardiazed the scale

In [150]:
```
# Putting Data in the same scale (between 0 and 1)

# Importing libraries
#from pandas import read_csv
from sklearn.preprocessing import MinMaxScaler

colTrain = x.columns
dfMLTrain = x[colTrain]
arrayTrain = dfMLTrain.values

colTest = test.columns
dfMLTest = test[colTest]
arrayTest = dfMLTest.values

# Splitting array in input and output
XTrain = arrayTrain
YTrain = y.values
XTest = arrayTest[:,0:100]

# Creating new scale
scaler = MinMaxScaler(feature_range = (0, 1))
rescaledXTrain = scaler.fit_transform(XTrain)
rescaledXTest = scaler.fit_transform(XTest)

# Data transformed
#print(rescaledXTrain[0:5,:])
```

# 3. Feature Engineering

In [151]:
```
clf = RandomForestClassifier(n_estimators=50, max_features='sqrt')
clf = clf.fit(XTrain,YTrain)
```

In [152]:
```
features = pd.DataFrame()
features['feature'] = x.columns
features['importance'] = clf.feature_importances_
features.sort_values(by=['importance'], ascending=True, inplace=True)
features.set_index('feature', inplace=True)
```

Let's now transform our train set and test set in a more compact datasets.

In [153]:
```
model = SelectFromModel(clf, prefit=True)
train_reduced = model.transform(x)
print (train_reduced.shape)
```

```
(39182, 45)
```

In [154]:
```
test_reduced = model.transform(test)
print (test_reduced.shape)
```

```
(9818, 45)
```

Splitting the data into training and testing datase

In [155]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_reduced,YTrain
,test_size=0.2,random_state=4)
```

# 4. Performance Comparison

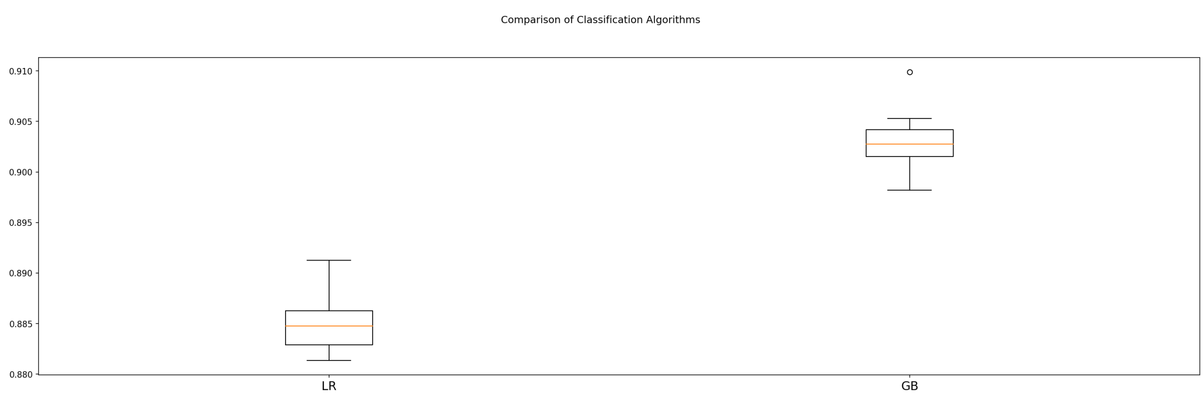
In [156]:
```
# Importing libraries
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
# Defining number of folds
num_folds = 10
num_instances = len(train_reduced)
seed = 7

# Preparing models
modelos = []
modelos.append(('LR', LogisticRegression()))
modelos.append(('GB',GradientBoostingClassifier()))
# Model Evaluation
resultados = []
nomes = []

for nome, modelo in modelos:
    kfold = model_selection.KFold(n_splits = num_folds, random_state = s
eed)
    cv_results = model_selection.cross_val_score(modelo, train_reduced,
YTrain, cv = kfold, scoring = 'accuracy')
    resultados.append(cv_results)
    nomes.append(nome)
    msg = "%s: %f (%f)" % (nome, cv_results.mean(), cv_results.std())
    print(msg)

# Boxplot to compare algorithms
fig = plt.figure()
fig.suptitle('Comparison of Classification Algorithms')
ax = fig.add_subplot(111)
plt.boxplot(resultados)
ax.set_xticklabels(nomes)
plt.show()
```

```
LR: 0.885305 (0.003084)
GB: 0.902991 (0.003056)
```



Both Logistic regression and GradientBoosting trees are used for classification purpose.

Logistic Regression Pros: Logistic regression will efficiently compute a maximum likelihood estimate assuming that all the inputs are independent. Linear regression is straightforward to understand and explain, and can be regularized to avoid overfitting. In addition, linear models can be updated easily with new data using stochastic gradient descent. Cons: Linear regression performs poorly when there are non-linear relationships.

Gradient Boosting Pros: Gradient Boosting method is a method used to solve classification and regression problems. It can learn non-linear relationships, and are fairly robust to outliers. Cons: Unconstrained, individual trees are prone to overfitting because they can keep branching until they memorize the training data.

My understanding is that XGB Models generally fare a little better than Logistic Models for these kind of problems. But, in my case I have improvements with the the boosting model over the logistic model even after tuning it a lot.

Using Logistic Regression

In [157]:
```
# Creating logistic regression model
modelLR = LogisticRegression()

# Training model and checking the score
modelLR.fit(train_reduced, YTrain)
modelLR.score(train_reduced, YTrain)

# Predictions
results2 = modelLR.predict(test_reduced)
```

In [158]:
```
#Checking accuracy
acc_log = round(modelLR.score(train_reduced, YTrain) * 100, 2)
acc_log
```

Out[158]:
```
88.66
```

In [159]:
```
results2 = pd.DataFrame(results2, columns=['results2']).to_csv('results
2.csv')
```

Creating a Gradient Boost Classifier

In [160]:
```
from sklearn.ensemble import GradientBoostingClassifier

ModelCLF = GradientBoostingClassifier(n_estimators = 650, learning_rate
= 1.0, max_depth = 1, random_state = 0)

# Training model and checking the score
ModelCLF.fit(train_reduced, YTrain)
ModelCLF.score(train_reduced,YTrain)

# Predictions
YPredGBC=ModelCLF.predict(test_reduced)
```

In [161]:
```
#Checking accuracy
acc_log = round(ModelCLF.score(train_reduced,YTrain) * 100, 2)
acc_log
```

Out[161]: 90.53

In [162]:
```
results1 = YPredGBC.astype(int)
results1 = pd.DataFrame(results1, columns=['results1']).to_csv('results
1.csv')
```

In [ ]: