CREDIT CARD FRAUD DETECTION

PHASE 2

PROBLEM DEFINITION

The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be fraud. This model is then used to identify whether a new transaction is fraudulent or not. Our aim here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications. This model is then used to identify whether a new transaction is fraudulent or not.

DATASETS

The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numeric input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise. There are no "Null" values, so we don't have to work on ways to replace values.

DATA PREPROCESSING

In [3]:
```
#Use parameter 'n' to display instances other than 5.
creditcard_data.head(n=20)
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.1285
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.1671
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.3276
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.6473
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.2060
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.2327
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.7501
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.4152
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.3732
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.0697
10	10.0	1.449044	-1.176339	0.913860	-1.375667	-1.971383	-0.629152	-1.423236	0.048456	-1.720408	...	-0.009302	0.313894	0.027740	0.500512	0.2513
11	10.0	0.384978	0.616109	-0.874300	-0.094019	2.924584	3.317027	0.470455	0.538247	-0.558895	...	0.049924	0.238422	0.009130	0.996710	-0.7673
12	10.0	1.249999	-1.221637	0.383930	-1.234899	-1.485419	-0.753230	-0.689405	-0.227487	-2.094011	...	-0.231809	-0.483285	0.084668	0.392831	0.1611
13	11.0	1.069374	0.287722	0.828613	2.712520	-0.178398	0.337544	-0.096717	0.115982	-0.221083	...	-0.036876	0.074412	-0.071407	0.104744	0.5482
14	12.0	-2.791855	-0.327771	1.641750	1.767473	-0.136588	0.807596	-0.422911	-1.907107	0.755713	...	1.151663	0.222182	1.020586	0.028317	-0.2327
15	12.0	-0.752417	0.345485	2.057323	-1.468643	-1.158394	-0.077850	-0.608581	0.003603	-0.436167	...	0.499625	1.353650	-0.256573	-0.065084	-0.0391
16	12.0	1.103215	-0.040296	1.267332	1.289091	-0.735997	0.288069	-0.586057	0.189380	0.782333	...	-0.024612	0.196002	0.013802	0.103758	0.3642
17	13.0	-0.436905	0.918966	0.924591	-0.727219	0.915679	-0.127867	0.707642	0.087962	-0.665271	...	-0.194796	-0.672638	-0.156858	-0.888386	-0.3424
18	14.0	-5.401258	-5.450148	1.186305	1.736239	3.049106	-1.763406	-1.559738	0.160842	1.233090	...	-0.503600	0.984460	2.458589	0.042119	-0.4816
19	15.0	1.492936	-1.029346	0.454795	-1.438026	-1.555434	-0.720961	-1.080664	-0.053127	-1.978682	...	-0.177650	-0.175074	0.040002	0.295814	0.3329

20 rows × 31 columns

In [4]:
```
#Number of instances and attributes,i.e., Dimensionality of the dataset
creditcard_data.shape
```

Out[4]:
```
(284807, 31)
```

Thus there are 284807 rows and 31 columns.

In [5]:
```
#observe the different feature type present in the data
creditcard_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

This shows that there are 284807 instances and 31 attributes including the class attribute.

In [6]:
```
#Sum of missing cells for each attribute
creditcard_data.isnull().sum()
```

Out[6]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

The 0 sum for all attributes shows that there are no missing values.

In [7]:
```
#Number of distinct categories or classes i.e., Fraudulent and Genuine
creditcard_data['Class'].nunique()
```

Out[7]:
```
2
```

As expected, there are only 2 classes.

In [8]:
```
#number of instances per class
creditcard_data.Class.value_counts()
```

Out[8]:
```
0    284315
1       492
Name: Class, dtype: int64
```

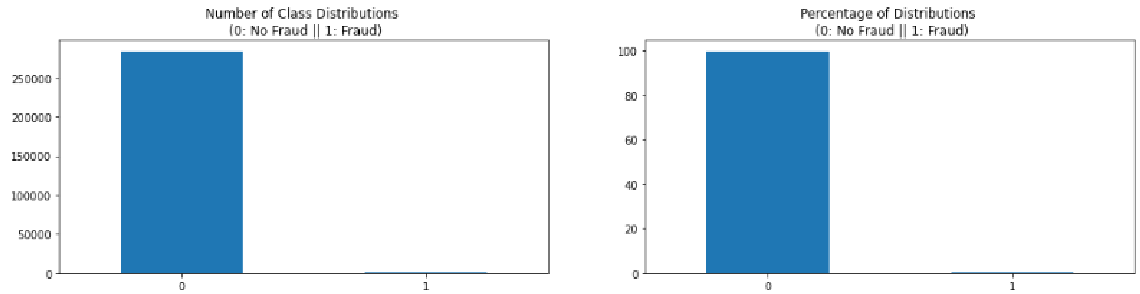
This shows a complete imbalance of classes. There are 284315 'Genuine' (0) instances and only 492 'Fraudulent' (1) instances.

In [9]:
```
#Here we will observe the distribution of our classes
classes=creditcard_data['Class'].value_counts()
normal_share=classes[0]/creditcard_data['Class'].count()*100
fraud_share=classes[1]/creditcard_data['Class'].count()*100

print("normal_share=",normal_share,"            ","fraud_share=",fraud_share)

imbalance= (fraud_share/normal_share)*100
print('Imbalance Percentage = ' + str(imbalance))
```

```
normal_share= 99.82725143693798              fraud_share= 0.1727485630620034
Imbalance Percentage = 0.173047500131896
```

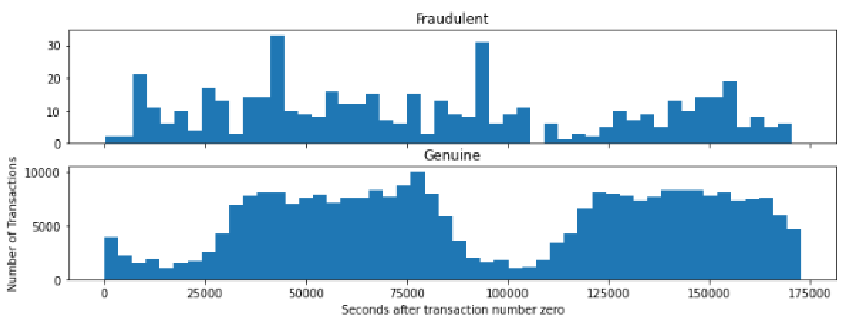
In [11]:

```python
# Create a bar plot for the number and percentage of fraudulent vs non-fraudulent transcations
fig, ax = plt.subplots(1, 2, figsize=(18,4))

classes.plot(kind='bar', rot=0, ax=ax[0])
ax[0].set_title('Number of Class Distributions \n (0: No Fraud || 1: Fraud)')

(classes/creditcard_data['Class'].count()*100).plot(kind='bar', rot=0, ax=ax[1])
ax[1].set_title('Percentage of Distributions \n (0: No Fraud || 1: Fraud)')

plt.show()
```



In [12]:

```python
#Histrogram for feature Time
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(12,4))

ax1.hist(creditcard_data["Time"][creditcard_data["Class"] == 1], bins = 50)
ax1.set_title('Fraudulent')

ax2.hist(creditcard_data["Time"][creditcard_data["Class"] == 0], bins = 50)
ax2.set_title('Genuine')

plt.xlabel('Seconds after transaction number zero')
plt.ylabel('Number of Transactions')
plt.show()
```



The transactions occur in a cyclic way. But the time feature does not provide any useful information as the time when the first transaction was initiated is not given. Thus, we'll drop this feature.

In [13]:
```
# Drop unnecessary columns
creditcard_data = creditcard_data.drop(['Time'],axis=1)
creditcard_data.head()
```

Out[13]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.1
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.1
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.3
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.6
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267	-0.2

5 rows × 30 columns

Now there are 30 features in the dataset.

DESCRIPTIVE STATISTICS

In [14]:
```
creditcard_data.describe().T.head()
```

Out[14]:

	count	mean	std	min	25%	50%	75%	max
V1	284807.0	3.918649e-15	1.958696	-56.407510	-0.920373	0.018109	1.315642	2.454930
V2	284807.0	5.682686e-16	1.651309	-72.715728	-0.598550	0.065486	0.803724	22.057729
V3	284807.0	-8.761736e-15	1.516255	-48.325589	-0.890365	0.179846	1.027196	9.382558
V4	284807.0	2.811118e-15	1.415869	-5.683171	-0.848640	-0.019847	0.743341	16.875344
V5	284807.0	-1.552103e-15	1.380247	-113.743307	-0.691597	-0.054336	0.611926	34.801666

In [15]:
```
creditcard_data.shape
```

Out[15]: (284807, 30)

Thus there are 284807 rows and 31 columns.

FRAUD CASES AND GENUINE CASES

In [16]:
```
fraud_cases=len(creditcard_data[creditcard_data['Class']==1])
print('Fraudulent Transactions:',fraud_cases)
```

```
Fraudulent Transactions: 492
```

In [17]:
```
non_fraud_cases=len(creditcard_data[creditcard_data['Class']==0])
print('Genuine Transactions:',non_fraud_cases)
```

```
Genuine Transactions: 284315
```

```
In [18]: #Descriptive statistics for Fraudulent Transactions
         print("Fraudulent Transactions")
         creditcard_data['Amount'][creditcard_data['Class']==1]. describe()
```

```
Fraudulent Transactions

Out[18]: count     492.000000
         mean      122.211321
         std       256.683288
         min         0.000000
         25%         1.000000
         50%         9.250000
         75%       105.890000
         max      2125.870000
         Name: Amount, dtype: float64
```

```
In [19]: #Descriptive statistics for Genuine Transactions
         print("Genuine Transactions")
         creditcard_data['Amount'][creditcard_data['Class']==0]. describe()
```

```
Genuine Transactions

Out[19]: count    284315.000000
         mean         88.291022
         std         250.105092
         min           0.000000
         25%           5.650000
         50%          22.000000
         75%          77.050000
         max       25691.160000
         Name: Amount, dtype: float64
```

Nothing much can be determined from the Amount, as most of the transactions are around 100 in both cases..

Python Packages:

```
In [ ]:
import pandas as pd
import numpy as np
from pandas import read_csv
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn import metrics
from sklearn import preprocessing
from sklearn.preprocessing import PowerTransformer

import warnings
warnings.filterwarnings('ignore')

from sklearn import linear_model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import roc_curve
from sklearn.model_selection import KFold
```

Learning Algorithms:

A.Logistic Regression

Logistic Regression is a supervised classification method that returns the probability of binary dependent variable that is predicted from the independent variable of dataset i.e. logistic regression predicts the probability of an outcome which has two values, either zero or one, no or yes and false or true. Logistic regression has similarities to linear regression, but, in linear regression a straight line is obtained, logistic regression shows a curve. The use of one or several predictors or independent variable is on what prediction is based, logistic regression produces logistic curves which plots the values between zero and one.

Logistic Regression is a regression model where the dependent variable is categorical and analyzes the relationship between multiple independent variables. There are many types of logistic regression model such as binary logistic model, multiple logistic model, binomial logistic models. Binary Logistic Regression model is used to estimate the probability of a binary response based on one or more predictors.

$$p = \frac{e^{\alpha+\beta_n X}}{1 + e^{\alpha+\beta_n X}}$$

Above equation represents the logistic regression in mathematical form.

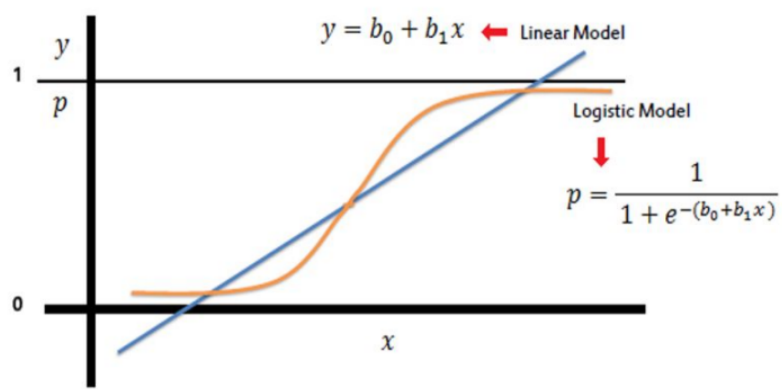


Figure 6: Logistic Curve

This graph shows the difference between linear regression and logistic regression where logistic regression shows a curve but linear regression represents a straight line.

B.. K-Nearest Neighbour Classifier

The k-nearest neighbour is an instance based learning which carries out its classification based on a similarity measure, like Euclidean, Manhattan or Minkowski distance functions. The first two distance measures work well with continuous variables while the third suits categorical variables. The Euclidean distance measure is used in this study for the kNN classifier. The Euclidean distance (Dij) between two input vectors (Xi, Xj) is given by:

$$D_{ij} = \sqrt{\sum_{k=1}^{n} \left(X_{ik} - X_{jk}\right)^2} \quad k=1,2,\ldots,n$$

For every data point in the dataset, the Euclidean distance between an input data point and current point is calculated. These distances are sorted in increasing order and k items with lowest distances to the input data point are selected. The majority class among these items is found and the classifier returns the majority class as the classification for the input point. Parameter tuning for k is carried out for k = 1, 3, 5, 7, 9, 11 and k = 3 showed optimal performance. Thus, value of k = 3 is used in the classifier.

C. SVM Model (Support Vector Machine)

SVM is a one of the popular machine learning algorithm for regression, classification. It is a supervised learning algorithm that analyses data used for classification and regression. SVM modeling involves two steps, firstly to train a data set and to obtain a model & then, to use this model to predict information of a testing data set. A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane where SVM model represents the training data points as points in space and then mapping is done so that the points which are of different classes are divided by a gap that is as wide as possible. Mapping is done in to the same space for new data points and then predicted on which side of the gap they fall.

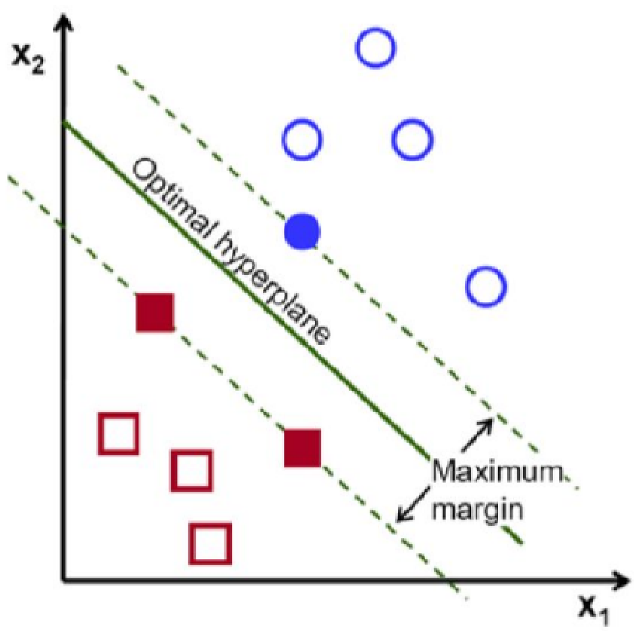


Figure 7: SVM Model Graph

In SVM algorithm, plotting is done as each data item is taken as a point in n-dimensional space where n is number of features, with the value of each feature being the value of a particular coordinate. Then, classification is performed by locating the hyperplane that separates the two classes very well.

Splitting Data set into Training and Testing

```
In [56]: from sklearn.model_selection import train_test_split
         df.drop('Time',axis=1,inplace=True)

         x=df.iloc[:,:-1]
         y=df.iloc[:,-1]
         X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
         print("Training Sample Size",X_train.shape)
         print("Testing Sample Size",X_test.shape)

         Training Sample Size (227845, 29)
         Testing Sample Size (56962, 29)
```

Model Evaluation

A.Logistic Regression

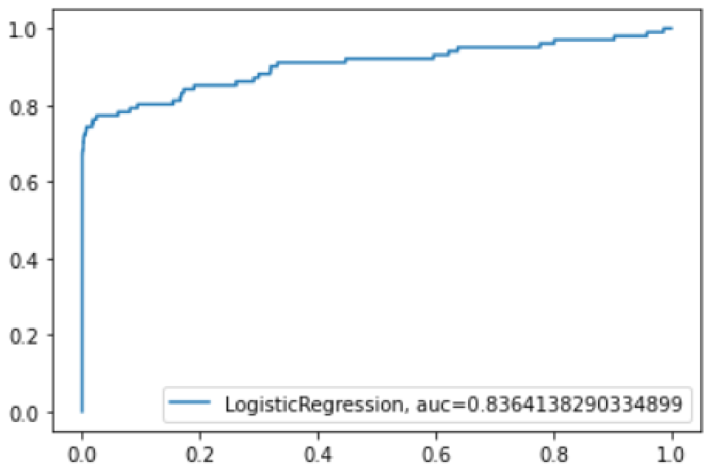
```
In [14]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import roc_auc_score
         for c in [0.01, 0.1, 1, 10]:
             print("C=",c, "Penalty= l2")
             logreg_classifier = linear_model.LogisticRegression(penalty='l2',C=c)
             logreg_classifier.fit(X_train,y_train)
             y_test_pred= logreg_classifier.predict_proba(X_test)
             cv_score= roc_auc_score(y_true=y_test,y_score=y_test_pred[:,1])
             print("ROC-AUC Score=", cv_score)
```

```
C= 0.01 Penalty= l2
ROC-AUC Score= 0.9233102923735683
C= 0.1 Penalty= l2
ROC-AUC Score= 0.9250668775218917
C= 1 Penalty= l2
ROC-AUC Score= 0.9031830444260374
C= 10 Penalty= l2
ROC-AUC Score= 0.90359389520493
```

We are considering value of C as 0.01 from the above results, because it has highest score

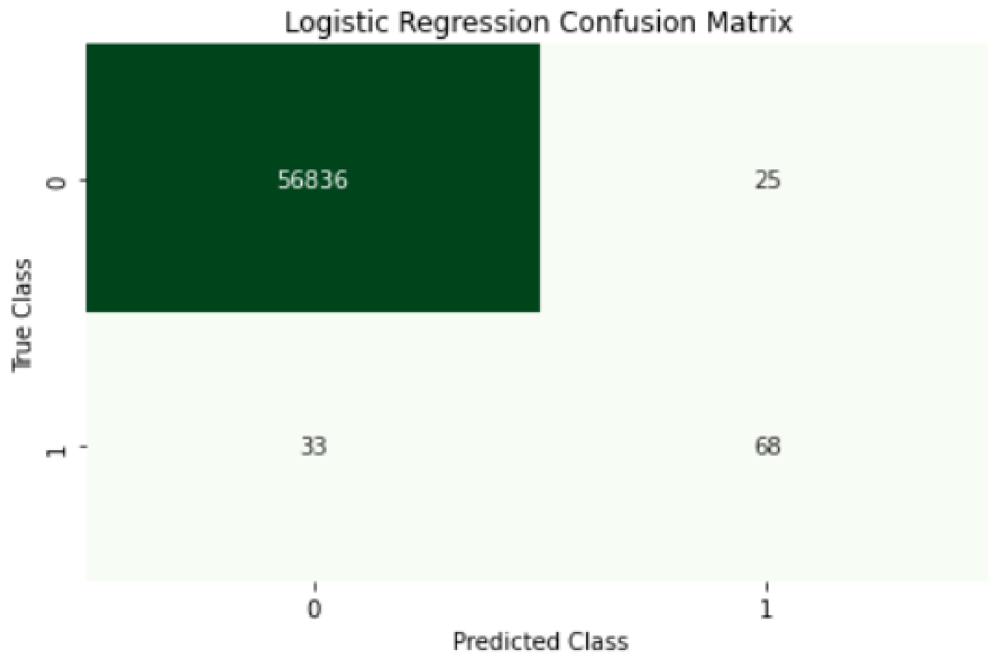
```
In [5]: clf = linear_model.LogisticRegression(penalty='l2',C=0.01)
        clf.fit(X_train, y_train)
        #predict on test to give probability
        y_pred= clf.predict_proba(X_test)
        #calculate the ROC-AUC
        score= roc_auc_score(y_true=y_test,y_score=y_pred[:,1])
        print("LogisticRegression ROC-AUC Score =", score)



        #diff bw predict and predict_prob

        LogisticRegression ROC-AUC Score = 0.9233102923735683
```

In [18]:
```python
from sklearn import metrics

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

y_pred = clf.predict(X_test)
y_pred_probability = clf.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_probability)
auc = metrics.roc_auc_score(y_test, y_pred)
plt.plot(fpr,tpr,label="LogisticRegression, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Confusion Matrix

In [65]:
```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score, accuracy_score
from sklearn.model_selection import cross_val_score

cm=confusion_matrix(y_test, y_pred)
#print(cm)
LR_ConfusionMatrix = pd.DataFrame(cm, index=[0,1], columns=[0,1])
sns.heatmap(LR_ConfusionMatrix , annot=True, cbar=None, cmap='Greens', fmt = 'g')

plt.title("Logistic Regression Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```



Evaluation Metrics

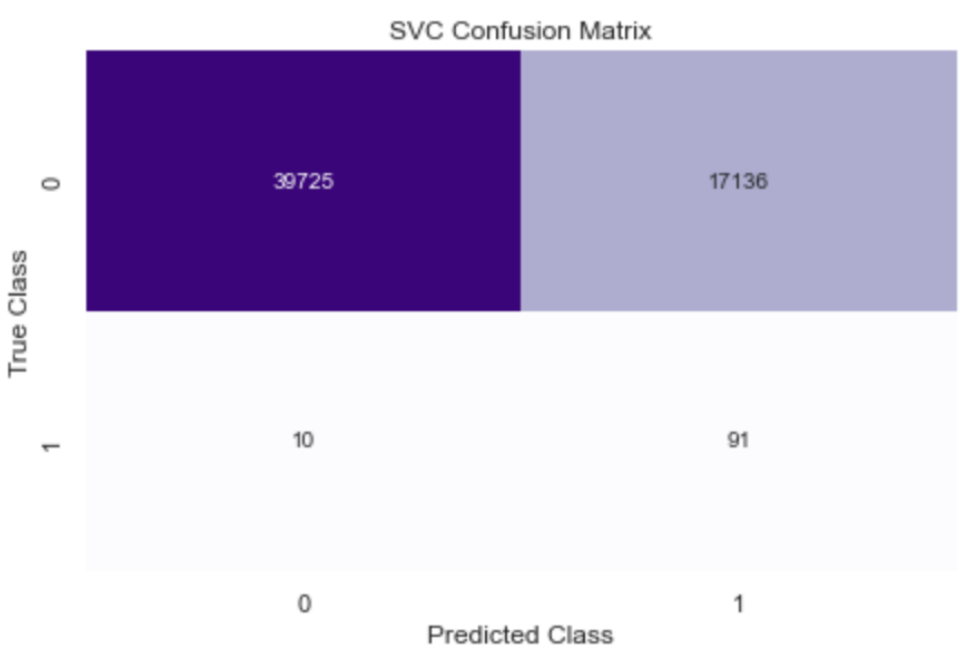
In [67]:
```
f1=f1_score(y_test, y_pred)
acc=accuracy_score(y_test, y_pred)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
pr=precision_score(y_test, y_pred)
t=tp+tn+fn+fp
print("specificity: "+str(tn/(tn+fp)))
print("sensitivity: "+str(tp/(tp+fn)))
print("accuracy_score:",acc)
print("precision_score:",pr)
print("f1_score:",f1)
```

```
specificity: 0.9998768927736058
sensitivity: 0.6138613861386139
accuracy_score: 0.9991924440855307
precision_score: 0.8985507246376812
f1_score: 0.7294117647058823
```

B. Support Vector Classifier

```
In [40]: #SVC Model
         clf = SVC(kernel= 'rbf', max_iter=100, C=1.0, gamma='auto')
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         #Score
         TrainScore =  round(clf.score(X_train, y_train) * 100, 2)
         TestScore = round(clf.score(X_test, y_test) * 100, 2)
         print('SVC Train Score: ', TrainScore)
         print('SVC Test Score: ',TestScore)

         #Confusion Matrix
         cm = confusion_matrix(y_test, y_pred)
         SVC_ConfusionMatrix = pd.DataFrame(cm, index=[0,1], columns=[0,1])
         sns.heatmap(SVC_ConfusionMatrix, annot=True, cbar=None, cmap="Purples", fmt = 'g')

         plt.title("SVC Confusion Matrix"), plt.tight_layout()
         plt.ylabel("True Class"), plt.xlabel("Predicted Class")
         plt.show()

         SVC Train Score:  69.82
         SVC Test Score:  69.9
```

Confusion Matrix



Evaluation Metrics

```
In [41]: f1=f1_score(y_test, y_pred)
         acc=accuracy_score(y_test, SVCModel_y_pred)
         tn, fp, fn, tp = confusion_matrix(y_test,SVCModel_y_pred).ravel()
         pr=precision_score(y_test, SVCModel_y_pred)
         t=tp+tn+fn+fp
         print("specificity: "+str(tn/(tn+fp)))
         print("sensitivity: "+str(tp/(tp+fn)))
         print("accuracy_score:",acc)
         print("precision_score:",pr)
         print("f1_score:",f1)
```

```
specificity: 0.6986335097870245
sensitivity: 0.9009900990099901
accuracy_score: 0.6989923106632492
precision_score: 0.005282405526208858
f1_score: 0.7771428571428572
```

C.**K-Nearest Neighbour Classifier**

In [69]:
```python
from sklearn import linear_model
from sklearn.neighbors import KNeighborsClassifier
a=[3,5,7,9]
test_accuracy=[]
train_accuracy=[]
for n_neighbor in [3,5,7,9]:
    print("n_neighbors=",n_neighbor)
    cv_score_mean=0
    k_score=0
    knn_classifier= KNeighborsClassifier(n_neighbors=n_neighbor)
    knn_classifier.fit(X_train,y_train)
    y_test_pred= knn_classifier.predict_proba(X_test)
    cv_score= roc_auc_score(y_true=y_test,y_score=y_test_pred[:,1])

    k_score=knn_classifier.score(X_test,y_test)
    k_score1=knn_classifier.score(X_train,y_train)
    print("ROC-AUC Score=", cv_score)
    test_accuracy.append(k_score)
    train_accuracy.append(k_score1)
plt.plot(a,test_accuracy,label='Testing Accuracy')
plt.plot(a,train_accuracy,label='Training Accuracy')
plt.legend(loc=2)

plt.show()
```

```
n_neighbors= 3
ROC-AUC Score= 0.8860284790372074
n_neighbors= 5
ROC-AUC Score= 0.890937183797696
n_neighbors= 7
ROC-AUC Score= 0.8908863389460594
n_neighbors= 9
ROC-AUC Score= 0.8908660532432661
```

Testing and Training Accuracy for different values of K

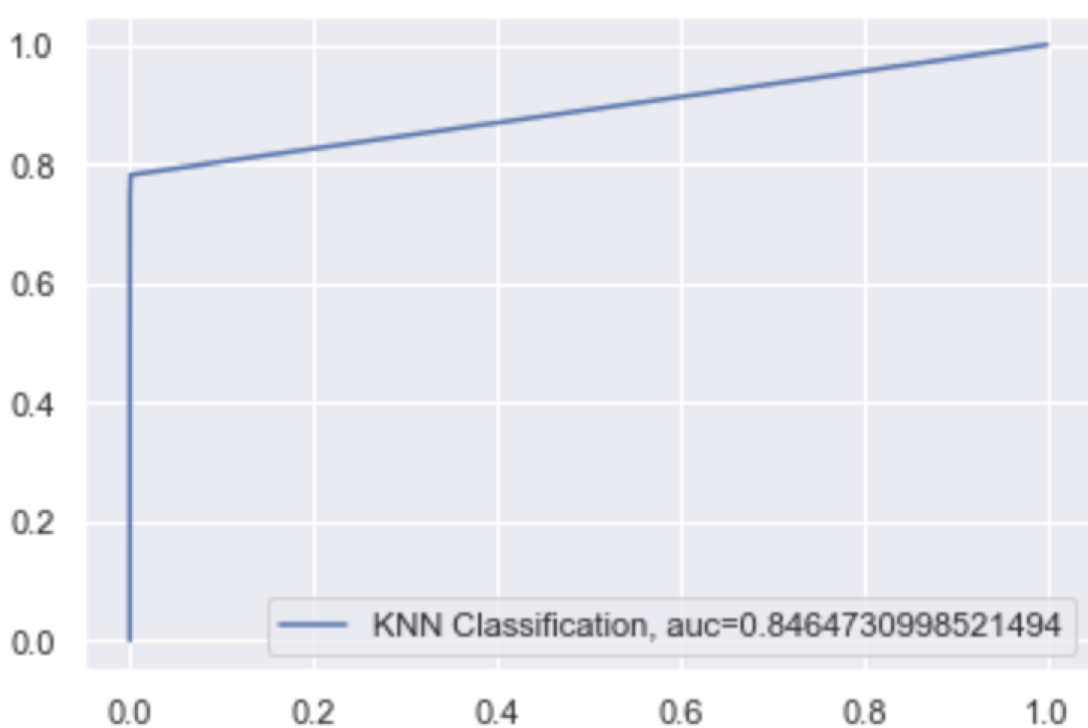
We are considering n_neigbors as 5 because it has the highest R0C-AUC Score

In [72]:
```
clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
y_pred= clf.predict_proba(X_test)
score= roc_auc_score(y_true=y_test,y_score=y_pred[:,1])
print("KNeighbors Classifier ROC-AUC Score =", score)
```

```
KNeighbors Classifier ROC-AUC Score = 0.890937183797696
```

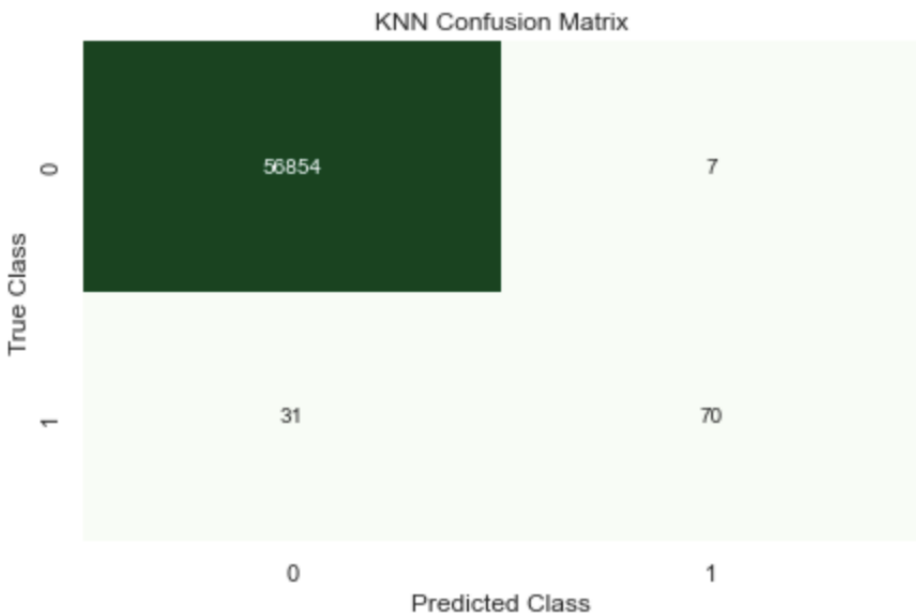
Plotting ROC Curve

In [73]:
```
y_pred = clf.predict(X_test)
y_pred_probability = clf.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_probability)
auc = metrics.roc_auc_score(y_test, y_pred)
plt.plot(fpr,tpr,label="KNN Classification, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Confusion Matrix

In [74]:
```
cm=confusion_matrix(y_test, y_pred)
#print(cm)
KNN_ConfusionMatrix = pd.DataFrame(cm, index=[0,1], columns=[0,1])
sns.heatmap(KNN_ConfusionMatrix , annot=True, cbar=None, cmap='Greens', fmt = 'g')
plt.title("KNN Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```



Evaluation Metrics

In [75]:
```
f1=f1_score(y_test, y_pred)
acc=accuracy_score(y_test, y_pred)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
pr=precision_score(y_test, y_pred)
t=tp+tn+fn+fp
print("specificity: "+str(tn/(tn+fp)))
print("sensitivity: "+str(tp/(tp+fn)))
print("accuracy_score:",acc)
print("precision_score:",pr)
print("f1_score:",f1)
```

```
specificity: 0.9998768927736058
sensitivity: 0.693069306930693
accuracy_score: 0.9993328885923949
precision_score: 0.9090909090909091
f1_score: 0.7865168539325842
```

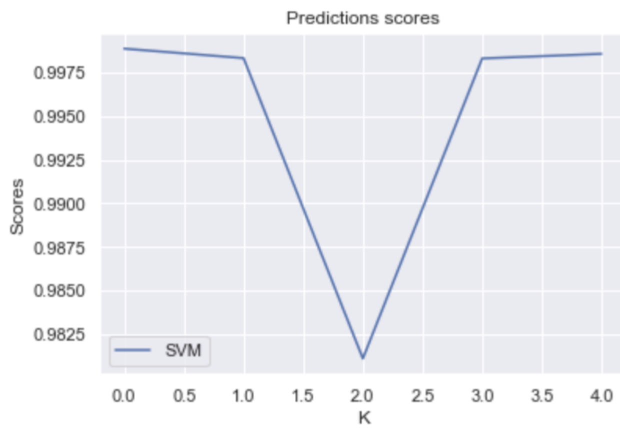
K-Fold Cross Validation (k=5)

Logistic Regression

In [79]:
```
plt.title('Predictions scores')
plt.plot(df1['I'],df1['Logistic'], label = 'Logistic')
plt.legend()
plt.xlabel('K')
plt.ylabel('Scores')
plt.show()
```

SVM

In [80]:
```
plt.title('Predictions scores')
plt.plot(df1['I'],df1['SVM'], label = 'SVM')
plt.legend()
plt.xlabel('K')
plt.ylabel('Scores')
plt.show()
```



K-Nearest Neighbour Classifier

In [81]:
```
plt.title('Predictions scores')
plt.plot(df1['I'],df1['KNN'], label = 'KNN')
plt.legend()
plt.xlabel('K')
plt.ylabel('Scores')
plt.show()
```

Plotting Accuracy curves for all the three models

```
In [76]: r =5
b = np.array([])
for i in range(r):
    b = np.append(b,[i,Logistic_scores[i],knn_scores[i],svc_scores[i]])
b = b.reshape(5,4)
df1 = pd.DataFrame(b, columns=["I",'Logistic','KNN','SVM'])
plt.title('Predictions scores')
plt.plot(df1['I'],df1['Logistic'], label = 'Logistic')
plt.plot(df1['I'],df1['KNN'], label = 'KNN')
plt.plot(df1['I'],df1['SVM'], label = 'SVM')
plt.legend()
plt.xlabel('K')
plt.ylabel('Scores')
plt.show()
```

From the above graphs accuracy of logistic and KNN models is same .