Untitled-1

April 13, 2022

[1]:
```python
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
import math
from scipy import stats
import seaborn as sns
import sklearn
```

[2]:
```python
df = pd.read_csv('pd_data_initial_preprocessing.csv',
                 usecols=['loan_amnt', 'term', 'int_rate', 'installment',␣
 ↪'grade', 'sub_grade',
                          'emp_title', 'emp_length', 'home_ownership',␣
 ↪'annual_inc',
                          'verification_status', 'loan_status',␣
 ↪'purpose',"addr_state",
                          'dti',  'open_acc', 'pub_rec',
                          'revol_util', 'total_acc', 'initial_list_status',␣
 ↪'application_type',
                          'mort_acc', 'pub_rec_bankruptcies'])
df.head()
```

[2]:
```
      emp_title  emp_length  revol_util    dti verification_status  \
0           NaN         NaN        55.1  21.61        Not Verified
1       teacher        10.0       105.8  25.61        Not Verified
2  Front Office         7.0        44.9   8.88        Not Verified
3       Manager        10.0        18.7  27.06     Source Verified
4     Paramedic        10.0        88.0   6.79     Source Verified

   annual_inc home_ownership  sub_grade grade  term  …  total_acc  \
0     10000.0            OWN        1.0     C  36.0  …        6.0
1     94000.0       MORTGAGE        1.0     C  60.0  …       26.0
2     46350.0       MORTGAGE        4.0     C  36.0  …       27.0
3     44000.0           RENT        1.0     B  36.0  …       19.0
4     85000.0       MORTGAGE        5.0     B  36.0  …       24.0
```

```
             purpose addr_state  initial_list_status application_type  \
0         credit_card         NY                   w       Individual
1  debt_consolidation         MA                   w       Individual
2    home_improvement         MA                   w       Individual
3                 car         CA                   w       Individual
4  debt_consolidation         MN                   w       Individual

  pub_rec pub_rec_bankruptcies loan_amnt  mort_acc  open_acc
0     0.0                  0.0    2300.0       0.0       4.0
1     0.0                  0.0   16000.0       7.0       9.0
2     0.0                  0.0    6025.0       2.0      11.0
3     0.0                  0.0   20400.0       0.0      15.0
4     0.0                  0.0   13000.0       1.0       5.0

[5 rows x 23 columns]
```

[3]:
```python
df.shape
```

[3]: (884884, 23)

[4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 884884 entries, 0 to 884883
Data columns (total 23 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   emp_title             832183 non-null  object
 1   emp_length            833683 non-null  float64
 2   revol_util            884387 non-null  float64
 3   dti                   884615 non-null  float64
 4   verification_status   884876 non-null  object
 5   annual_inc            884876 non-null  float64
 6   home_ownership        884876 non-null  object
 7   sub_grade             884876 non-null  float64
 8   grade                 884876 non-null  object
 9   term                  884876 non-null  float64
 10  int_rate              884876 non-null  float64
 11  installment           884876 non-null  float64
 12  loan_status           884876 non-null  object
 13  total_acc             884876 non-null  float64
 14  purpose               884876 non-null  object
 15  addr_state            884876 non-null  object
 16  initial_list_status   884876 non-null  object
 17  application_type      884876 non-null  object
 18  pub_rec               884876 non-null  float64
 19  pub_rec_bankruptcies  884876 non-null  float64
 20  loan_amnt             884876 non-null  float64
```

```
 21  mort_acc               884876 non-null  float64
 22  open_acc               884876 non-null  float64
dtypes: float64(14), object(9)
memory usage: 155.3+ MB
```

1 Eliminamos columnas que no aportan informacion útil o que tienen más del 90% de valores NaN:

[5]:
```
df.isnull().sum()
```

[5]:
```
emp_title               52701
emp_length              51201
revol_util                497
dti                       269
verification_status         8
annual_inc                  8
home_ownership              8
sub_grade                   8
grade                       8
term                        8
int_rate                    8
installment                 8
loan_status                 8
total_acc                   8
purpose                     8
addr_state                  8
initial_list_status         8
application_type            8
pub_rec                     8
pub_rec_bankruptcies        8
loan_amnt                   8
mort_acc                    8
open_acc                    8
dtype: int64
```

[6]:
```
df[df['grade'].isna()]
```

[6]:
```
       emp_title  emp_length  revol_util  dti verification_status  annual_inc  \
105451       NaN         NaN         NaN  NaN                 NaN         NaN
105452       NaN         NaN         NaN  NaN                 NaN         NaN
228154       NaN         NaN         NaN  NaN                 NaN         NaN
228155       NaN         NaN         NaN  NaN                 NaN         NaN
463785       NaN         NaN         NaN  NaN                 NaN         NaN
463786       NaN         NaN         NaN  NaN                 NaN         NaN
884882       NaN         NaN         NaN  NaN                 NaN         NaN
884883       NaN         NaN         NaN  NaN                 NaN         NaN
```

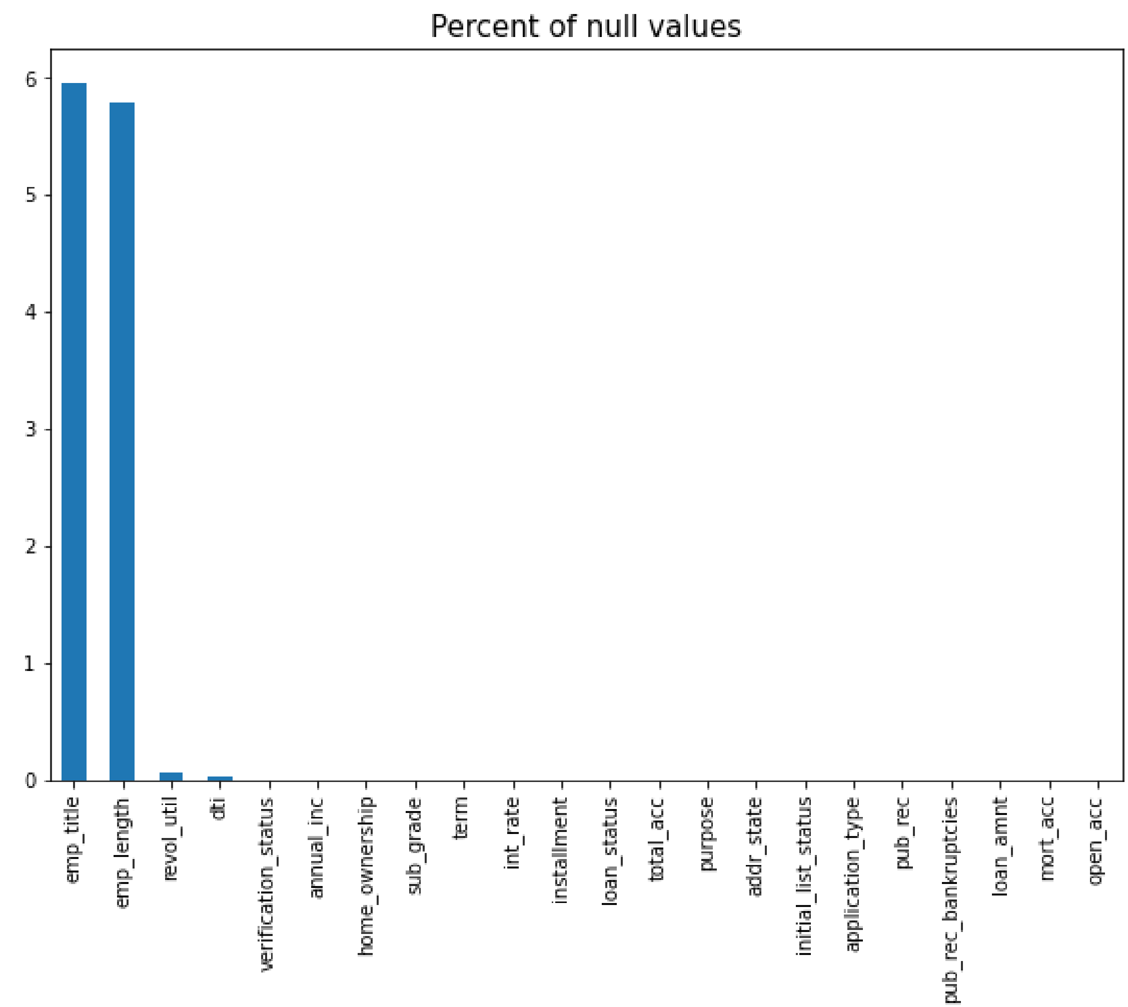
```
       home_ownership  sub_grade grade  term  …  total_acc  purpose  \
105451            NaN        NaN   NaN   NaN  …        NaN      NaN
105452            NaN        NaN   NaN   NaN  …        NaN      NaN
228154            NaN        NaN   NaN   NaN  …        NaN      NaN
228155            NaN        NaN   NaN   NaN  …        NaN      NaN
463785            NaN        NaN   NaN   NaN  …        NaN      NaN
463786            NaN        NaN   NaN   NaN  …        NaN      NaN
884882            NaN        NaN   NaN   NaN  …        NaN      NaN
884883            NaN        NaN   NaN   NaN  …        NaN      NaN

       addr_state  initial_list_status application_type pub_rec  \
105451        NaN                  NaN              NaN     NaN
105452        NaN                  NaN              NaN     NaN
228154        NaN                  NaN              NaN     NaN
228155        NaN                  NaN              NaN     NaN
463785        NaN                  NaN              NaN     NaN
463786        NaN                  NaN              NaN     NaN
884882        NaN                  NaN              NaN     NaN
884883        NaN                  NaN              NaN     NaN

       pub_rec_bankruptcies loan_amnt  mort_acc  open_acc
105451                  NaN       NaN       NaN       NaN
105452                  NaN       NaN       NaN       NaN
228154                  NaN       NaN       NaN       NaN
228155                  NaN       NaN       NaN       NaN
463785                  NaN       NaN       NaN       NaN
463786                  NaN       NaN       NaN       NaN
884882                  NaN       NaN       NaN       NaN
884883                  NaN       NaN       NaN       NaN

[8 rows x 23 columns]
```

Observamos que las 8 observaciones que tienen NaN en alguna de las casillas y en grade son las mismas, procedemos a eliminarlas porque no aportan ninguna información.

[7]:
```
df = df[df['grade'].notna()]
```

[8]:
```
df.drop("grade",axis =1, inplace = True) #la eliminamos porque sub_grade nos␣
 ↪aporta lo mismo
```

[9]:
```
((df.isnull().sum()/len(df))*100).plot(kind = "bar", figsize = (10,7))
plt.title("Percent of null values",fontsize= 15)
plt.show()
```



Las siguientes variables con mayor número de missings son `emp_title`,`emp_lenght`,`revol_util` y `dti`. Vemos que `emp_title` toma muchos valores diferentes (214509) por lo que no interesaría imputarlos ya que para el estudio tampoco aporta mucho.

Vemos que `emp_title` tiene demasiados valores diferentes, a parte de un gran volumen de missings, por lo que procedemos a eliminarla ya que no es útil para nuestr estudio.

[10]:
```
df.describe( include= ["object"]).T
```

[10]:

	count	unique	top	freq
emp_title	832183	214509	Teacher	17113
verification_status	884876	3	Source Verified	363463
home_ownership	884876	5	MORTGAGE	439600
loan_status	884876	7	Current	422685
purpose	884876	14	debt_consolidation	520846
addr_state	884876	51	CA	121479
initial_list_status	884876	2	w	568999

```
application_type     884876       2          Individual   863558
```

[11]:
```
df.drop(['emp_title'], 1, inplace = True)
```

2 Análisis descriptivo de la variable objetivo:

[12]:
```
## Vamos a ver como se distribuyen los diferentes valores de la variable␣
↪objetivo:

df["loan_status"].value_counts(dropna = False)
```

[12]:
```
Current               422685
Fully Paid            345520
Charged Off            97047
Late (31-120 days)     11168
In Grace Period         5507
Late (16-30 days)       2915
Default                   34
Name: loan_status, dtype: int64
```

Vamos a codificar la variable objetivo con los valores *pagado* e *impagado*.

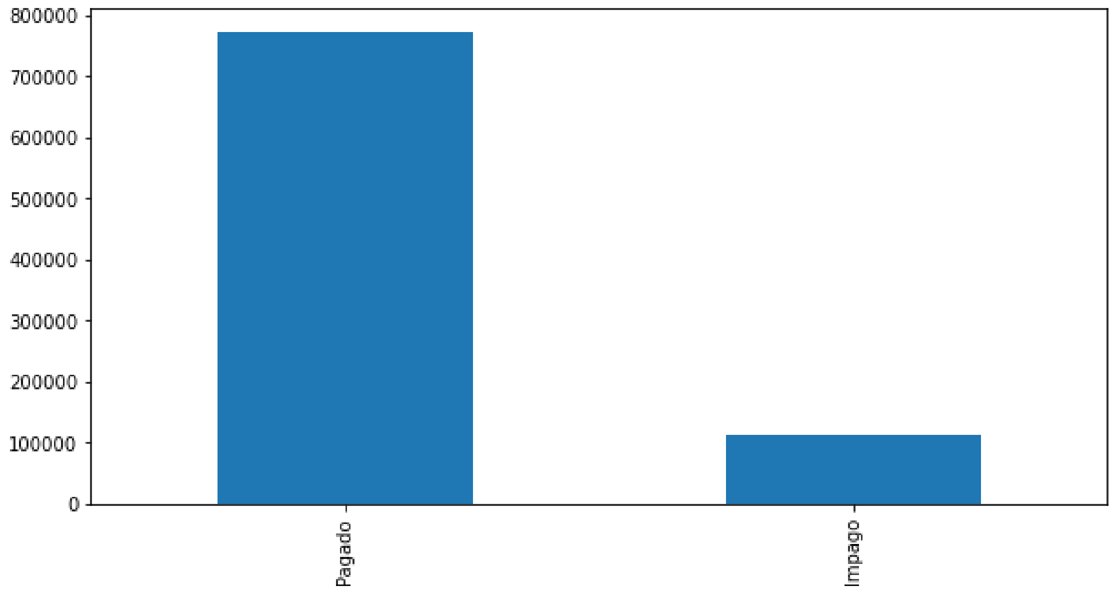
[13]:
```
df['loan_status'] = df.loan_status.map({'Current': 'Pagado'
                                        , 'Fully Paid':'Pagado'
                                        , 'Charged Off': 'Impago'
                                        , 'Late (31-120 days)': 'Impago'
                                        , 'In Grace Period': 'Pagado'
                                        , 'Late (16-30 days)': 'Impago'
                                        , 'Default': 'Impago'})
```

[14]:
```
df["loan_status"].value_counts(dropna = False)
```

[14]:
```
Pagado    773712
Impago    111164
Name: loan_status, dtype: int64
```

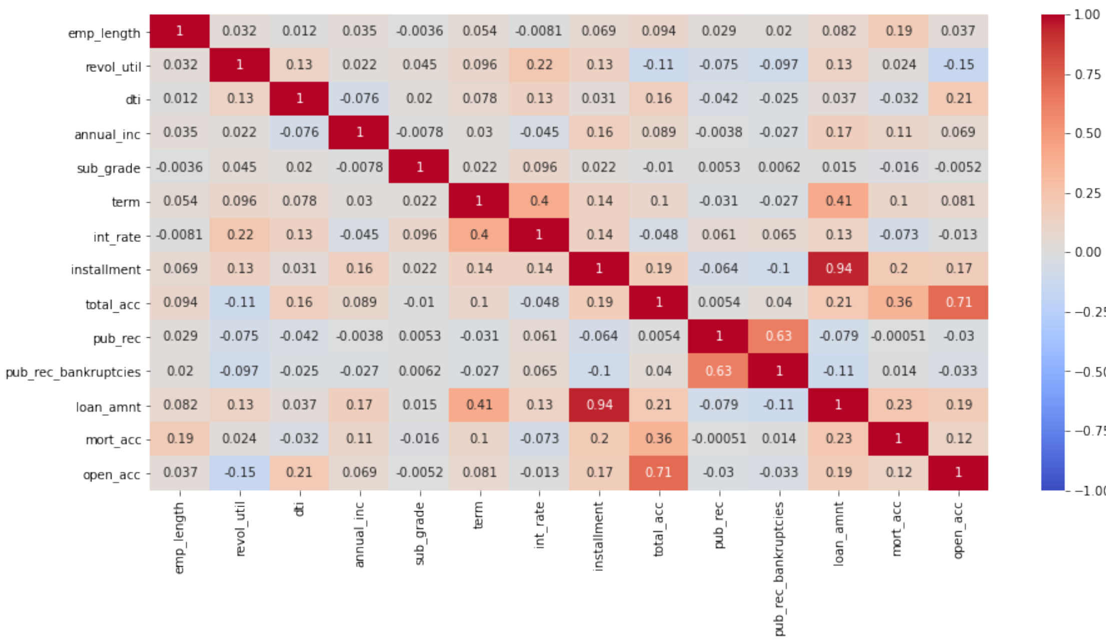
[15]:
```
df["loan_status"].value_counts(dropna = False).plot(kind = "bar",figsize =␣
↪(10,5))
```

[15]: <AxesSubplot:>



[16]:
```
plt.figure(figsize= (15,7))
sns.heatmap(df.corr(), vmin=1, vmax=-1, annot=True, cmap="coolwarm")
```

[16]: <AxesSubplot:>



[17]:
```
df["home_ownership"].value_counts()
```

[17]:
```
MORTGAGE    439600
RENT        350505
OWN          94752
ANY             16
NONE             3
Name: home_ownership, dtype: int64
```

[18]:
```
df["home_ownership"]= df["home_ownership"].replace(["ANY","NONE"], "OTHER")
```

Imputación de missings:

Para variables continuas imputamos media.

Para variables categóricas moda.

[19]:
```
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer
```

[20]:
```
temp_values = df.dti.values.reshape(-1,1)
impute_knn = KNNImputer(n_neighbors=5)
impute_knn.fit_transform(temp_values)
transformed_values = impute_knn.transform(temp_values)
df.dti = transformed_values

temp_values2 = df.revol_util.values.reshape(-1,1)
impute_knn2 = KNNImputer(n_neighbors=5)
impute_knn2.fit_transform(temp_values2)
transformed_values2 = impute_knn2.transform(temp_values2)
df.revol_util = transformed_values2

df["emp_length"].fillna(df["emp_length"].mode()[0], inplace = True)
```

Transformamos las variables categóricas con Dummies:

[21]:
```
df["loan_status"] = df["loan_status"].map({"Pagado":0,"Impago":1})
```

[22]:
```
df['verification_status'] = df.verification_status.map({'Verified': 0, 'Source␣
↪Verified': 1, 'Not Verified': 2})
df['initial_list_status'] = df.initial_list_status.map({'w': 0, 'f': 1})
df['application_type'] = df.application_type.map({'Individual': 0, 'Joint App':␣
↪1})
```

[23]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

df['addr_state'] = le.fit_transform(df['addr_state'].astype(str))
df['sub_grade'] = le.fit_transform(df['sub_grade'].astype(str))
df['purpose'] = le.fit_transform(df['purpose'].astype(str))
df['emp_length'] = le.fit_transform(df['emp_length'].astype(str))
```

```
df['home_ownership'] = le.fit_transform(df['home_ownership'].astype(str))
```

3 KNN

[24]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

[25]:
```
# Dividimos datos:
X = df.drop("loan_status",axis =1 )
y= df["loan_status"]

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.15,␣
↪random_state = 0)
```

[26]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

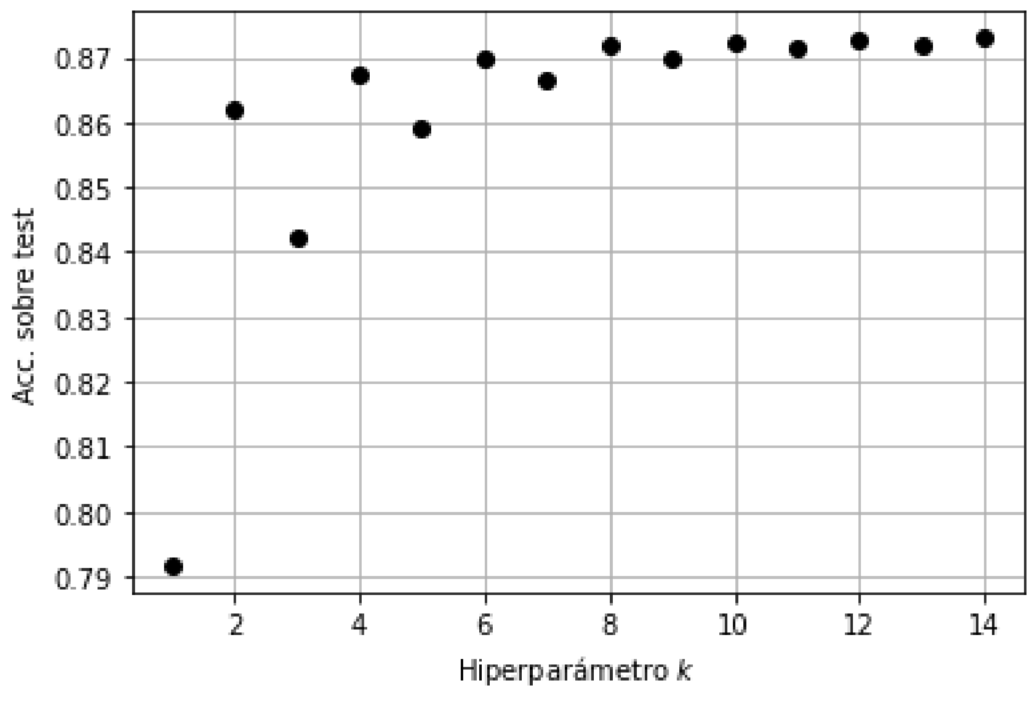
[59]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import auc
import matplotlib.pyplot as plt
```

[35]:
```
from sklearn import metrics
ks = list(range(1, 15))
print(ks)

# En este diccionario iremos guardando las accuracies sobre test asociadas a␣
↪cada valor de $k$
accs = {}
# Vamos recorriendo la rejilla con un bucle for...
for k in ks:

    # Definimos el modelo con el valor de hiperparámetro correspondiente
    knn = KNeighborsClassifier(n_neighbors=k)

    # Ajustamos a los datos de entrenamiento
    knn.fit(X_train, y_train)

    # Hacemos predicciones sobre los datos de test
```

```
    y_pred = knn.predict(X_test)

    # Evaluamos y guardamos la métrica correspondiente (en este caso accuracy)
    acc = metrics.accuracy_score(y_test, y_pred)

    accs[k] = acc
print(accs)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
{1: 0.7916930355905132, 2: 0.8617967031311213, 3: 0.8424419130277552, 4:
0.8672663713347196, 5: 0.8592954223548203, 6: 0.8699484675888256, 7:
0.8665581773799838, 8: 0.8716662146279721, 9: 0.8696320405026671, 10:
0.8723367386914986, 11: 0.8713799234547811, 12: 0.8728264472772203, 13:
0.8720429135400657, 14: 0.8729921947985414}
```

[36]:
```
ks_arr = np.fromiter(accs.keys(), dtype=int)
accs_arr = np.fromiter(accs.values(), dtype=float)

plt.plot(ks_arr, accs_arr, 'ok')
plt.grid(True)
plt.xlabel('Hiperparámetro $k$');
plt.ylabel('Acc. sobre test');
```



[42]:

```
knn = KNeighborsClassifier(n_neighbors=14)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

# En lugar de definir la accuracy como antes, podemos utilizar directamente la␣
↪métrica desde sklearn
from sklearn import metrics
```

```
Accuracy:  0.8729921947985414
```

[49]:

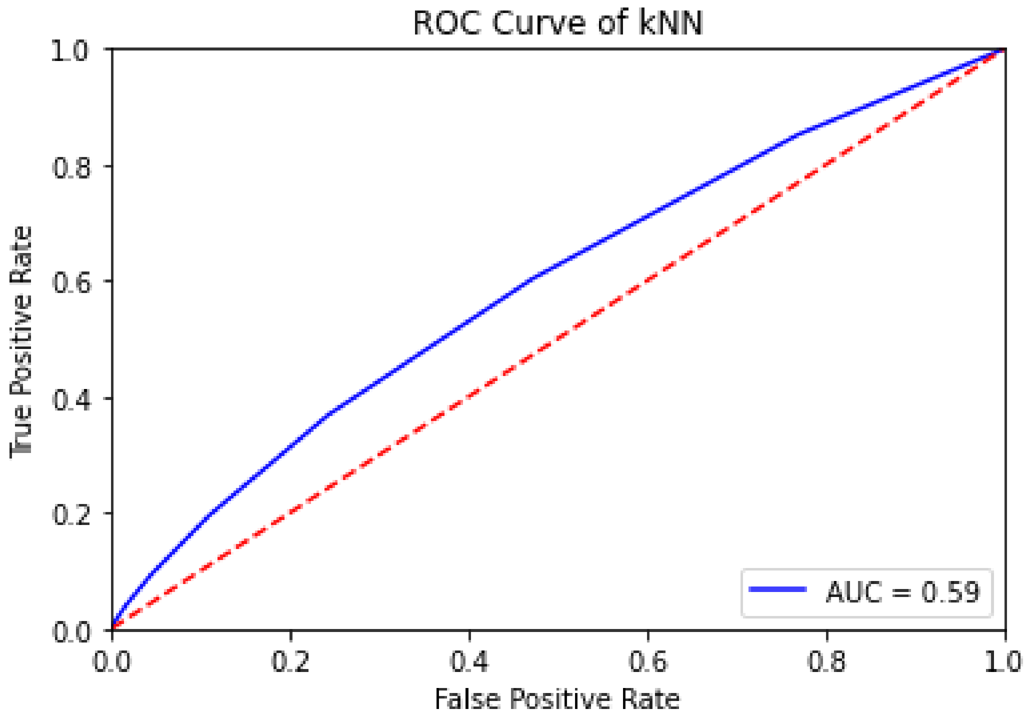
```
acc = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: ', acc)

C = metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix', C)

prec = metrics.precision_score(y_test, y_pred)
print('Precision', prec)

rec = metrics.recall_score(y_test, y_pred)
print('Recall', rec)

f1 = metrics.f1_score(y_test, y_pred)
print('F-score', f1)
```

```
Accuracy:  0.8729921947985414
Confusion Matrix [[115799    177]
 [ 16681     75]]
Precision 0.2976190476190476
Recall 0.004476008593936501
F-score 0.008819379115710256
```

[60]:

```
knn = KNeighborsClassifier(n_neighbors = 14)
knn.fit(X_train,y_train)

y_scores = knn.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test, y_scores[:, 1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```python
plt.title('ROC Curve of kNN')
plt.show()
```



[ ]:

```python

```

[ ]:

```python

```