```
#Credit Card fraud detection
!pip install pendulum
!pip install category_encoders
```

```
Collecting pendulum
  Downloading pendulum-2.1.2-cp37-cp37m-manylinux1_x86_64.whl (155 kB)
     |████████████████████████████████| 155 kB 4.9 MB/s
Collecting pytzdata>=2020.1
  Downloading pytzdata-2020.1-py2.py3-none-any.whl (489 kB)
     |████████████████████████████████| 489 kB 45.1 MB/s
Requirement already satisfied: python-dateutil<3.0,>=2.6 in /usr/local/lib/pyt
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packa
Installing collected packages: pytzdata, pendulum
Successfully installed pendulum-2.1.2 pytzdata-2020.1
Collecting category_encoders
  Downloading category_encoders-2.4.0-py2.py3-none-any.whl (86 kB)
     |████████████████████████████████| 86 kB 3.0 MB/s
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.7/
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Installing collected packages: category-encoders
Successfully installed category-encoders-2.4.0
```

```
import pandas as pd
import numpy as np
from numpy import argmax

from datetime import date, time, timedelta
import pendulum # for time formatting

import matplotlib.pyplot as plt
import seaborn as sns

import category_encoders as ce # for categorical encoding
from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, reca
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: Futur
  import pandas.util.testing as tm
```

```
!wget "https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmc
```

```
--2022-05-04 15:50:09--  https://drive.google.com/uc?export=download&id=1HC7Mf
Resolving drive.google.com (drive.google.com)... 173.194.216.102, 173.194.216.
Connecting to drive.google.com (drive.google.com)|173.194.216.102|:443... conn
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0k-14-docs.googleusercontent.com/docs/securesc/ha0ro937g
Warning: wildcards not supported in HTTP.
--2022-05-04 15:50:11--  https://doc-0k-14-docs.googleusercontent.com/docs/sec
Resolving doc-0k-14-docs.googleusercontent.com (doc-0k-14-docs.googleuserconte
Connecting to doc-0k-14-docs.googleusercontent.com (doc-0k-14-docs.googleuserc
HTTP request sent, awaiting response... 200 OK
Length: 37517511 (36M) [text/csv]
Saving to: ‘orders.csv’

orders.csv          100%[===================>]  35.78M  98.3MB/s    in 0.4s

2022-05-04 15:50:12 (98.3 MB/s) - ‘orders.csv’ saved [37517511/37517511]
```

```
df = pd.read_csv('orders.csv')
```

```
df.head()
```

Double-click (or enter) to edit

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76829 entries, 0 to 76828
Data columns (total 34 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   order_id                        76829 non-null  int64  
 1   city                            76829 non-null  object 
 2   category_name                   76743 non-null  object 
 3   product_id                      76829 non-null  int64  
 4   product_name                    76829 non-null  object 
 5   amount                          76829 non-null  float64
 6   device                          76829 non-null  object 
 7   payment_id                      76829 non-null  object 
 8   customer_ip                     76829 non-null  object 
 9   customer_id                     76829 non-null  object 
 10  payment_method                  76829 non-null  object 
 11  payment_method_provider         76829 non-null  object 
 12  payment_method_bin              76778 non-null  float64
 13  payment_method_type             76741 non-null  object 
 14  payment_method_product          74731 non-null  object 
 15  payment_method_card_category    74228 non-null  object 
 16  payment_method_issuer_bank      73690 non-null  object 
 17  payment_method_issuer_country   76730 non-null  object 
 18  is_fraudulent                   76829 non-null  bool   
 19  time_diff                       76829 non-null  int64  
 20  created_at_hod_sin              76829 non-null  float64
 21  created_at_hod_cos              76829 non-null  float64
 22  created_dom_sin                 76829 non-null  float64
 23  created_dom_cos                 76829 non-null  float64
 24  created_dow_sin                 76829 non-null  float64
 25  created_dow_cos                 76829 non-null  float64
 26  created_wom_sin                 76829 non-null  float64
 27  created_wom_cos                 76829 non-null  float64
 28  experience_dom_sin              76829 non-null  float64
 29  experience_dom_cos              76829 non-null  float64
 30  experience_dow_sin              76829 non-null  float64
 31  experience_dow_cos              76829 non-null  float64
 32  experience_wom_sin              76829 non-null  float64
 33  experience_wom_cos              76829 non-null  float64
dtypes: bool(1), float64(16), int64(3), object(14)
memory usage: 19.4+ MB
```

```
df.columns
```

```
Index(['order_id', 'city', 'category_name', 'product_id', 'product_name',
       'amount', 'device', 'payment_id', 'customer_ip', 'customer_id',
       'payment_method', 'payment_method_provider', 'payment_method_bin',
       'payment_method_type', 'payment_method_product',
       'payment_method_card_category', 'payment_method_issuer_bank',
       'payment_method_issuer_country', 'is_fraudulent', 'time_diff',
       'created_at_hod_sin', 'created_at_hod_cos', 'created_dom_sin',
       'created_dom_cos', 'created_dow_sin', 'created_dow_cos',
       'created_wom_sin', 'created_wom_cos', 'experience_dom_sin',
       'experience_dom_cos', 'experience_dow_sin', 'experience_dow_cos',
       'experience_wom_sin', 'experience_wom_cos'],
      dtype='object')
```

```
df["is_fraudulent"].value_counts()
```

```
False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

```
df.nunique() # unique values per feature
```

```
order_id                          76829
city                                  2
category_name                        96
product_id                          353
product_name                        340
amount                             8627
device                                5
payment_id                        76829
customer_ip                       44208
customer_id                       51109
payment_method                        1
payment_method_provider               5
payment_method_bin                 6991
payment_method_type                   8
payment_method_product              144
payment_method_card_category          2
payment_method_issuer_bank         2046
payment_method_issuer_country       151
is_fraudulent                         2
time_diff                           117
created_at_hod_sin                   24
created_at_hod_cos                   13
created_dom_sin                      31
created_dom_cos                      18
created_dow_sin                       6
created_dow_cos                       4
created_wom_sin                       5
created_wom_cos                       4
experience_dom_sin                   31
experience_dom_cos                   18
experience_dow_sin                    6
experience_dow_cos                    4
experience_wom_sin                    5
experience_wom_cos                    4
dtype: int64
```

```
# dropping product_name as we have product_id
df = df.drop(columns=["order_id", "payment_method", "payment_id", "product_name"])
df.head()
```

```
df.shape
```

```
(76829, 30)
```

```
# drop duplicates
if df.shape[0] == df.drop_duplicates().shape[0] :
    print('No duplicates Found')
else:
    duplicates = df.shape[0] - df.drop_duplicates().shape[0]
    print('{} duplicates found'.format(duplicates))
```

```
6821 duplicates found
```

```
df = df.drop_duplicates()
```

```
#NAN values
df.isna().sum()
```

```
city                                0
category_name                      83
product_id                          0
amount                              0
device                              0
customer_ip                         0
customer_id                         0
payment_method_provider             0
payment_method_bin                 46
payment_method_type                61
payment_method_product           1886
payment_method_card_category     2413
payment_method_issuer_bank       2838
payment_method_issuer_country      66
is_fraudulent                       0
time_diff                           0
created_at_hod_sin                  0
created_at_hod_cos                  0
created_dom_sin                     0
created_dom_cos                     0
created_dow_sin                     0
created_dow_cos                     0
created_wom_sin                     0
created_wom_cos                     0
experience_dom_sin                  0
```

```
experience_dom_cos                 0
experience_dow_sin                 0
experience_dow_cos                 0
experience_wom_sin                 0
experience_wom_cos                 0
dtype: int64
```

```
df = df.dropna(axis = 0, how= 'any',
               subset = ['payment_method_issuer_bank','payment_method_product', 'pa
```

```
df.isna().sum()
```

```
city                              0
category_name                    75
product_id                        0
amount                            0
device                            0
customer_ip                       0
customer_id                       0
payment_method_provider           0
payment_method_bin                0
payment_method_type               0
payment_method_product            0
payment_method_card_category      0
payment_method_issuer_bank        0
payment_method_issuer_country    11
is_fraudulent                     0
time_diff                         0
created_at_hod_sin                0
created_at_hod_cos                0
created_dom_sin                   0
created_dom_cos                   0
created_dow_sin                   0
created_dow_cos                   0
created_wom_sin                   0
created_wom_cos                   0
experience_dom_sin                0
experience_dom_cos                0
experience_dow_sin                0
experience_dow_cos                0
experience_wom_sin                0
experience_wom_cos                0
dtype: int64
```

```
df['category_name']
```

```
0                    Singapore Zoo
2                      Dubai Frame
3               Singapore Cable Car
4       Universal Studios Singapore
6                  Trickeye Museum
                      ...
76822             At The Top Tickets
76823           Dubai Dinner Cruises
76824                 Dubai Aquarium
76825                 Dubai Aquarium
```

```
76828      Lifestyle & Entertainment
Name: category_name, Length: 63344, dtype: object
```

```
# fill other for catgeory_name=null
df['category_name'] = df['category_name'].fillna('other')
```

```
df['payment_method_issuer_country']
```

```
0                   Singapore
1                   Singapore
2                      Brazil
3                       India
4                      Brunei
                 ...
76824                 Hungary
76825                  France
76826           United States
76827    United Arab Emirates
76828           United States
Name: payment_method_issuer_country, Length: 69547, dtype: object
```

```
df['payment_method_issuer_country'] = df['payment_method_issuer_country'].fillna('o
```

```
df.isna().sum()
```

```
city                             0
category_name                    0
product_id                       0
amount                           0
device                           0
customer_ip                      0
customer_id                      0
payment_method_provider          0
payment_method_bin               0
payment_method_type              0
payment_method_product           0
payment_method_card_category     0
payment_method_issuer_bank       0
payment_method_issuer_country    0
is_fraudulent                    0
time_diff                        0
created_at_hod_sin               0
created_at_hod_cos               0
created_dom_sin                  0
created_dom_cos                  0
created_dow_sin                  0
created_dow_cos                  0
created_wom_sin                  0
created_wom_cos                  0
experience_dom_sin               0
experience_dom_cos               0
experience_dow_sin               0
experience_dow_cos               0
```

```
experience_wom_sin              0
experience_wom_cos              0
dtype: int64
```

```
# Dataset
X = df.drop(['is_fraudulent'], axis = 1)
y = df['is_fraudulent']
```

```
# Train, CV, test split
from sklearn.model_selection import train_test_split
#0.6, 0.2, 0.2 split

X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X, y, test_size=0.2, random_sta

X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.25,
```

```
X_train.head()
```

```
set(X.columns) - set(X.select_dtypes(['number']).columns)
```

```
{'category_name',
 'city',
 'customer_id',
 'customer_ip',
 'device',
 'payment_method_card_category',
 'payment_method_issuer_bank',
 'payment_method_issuer_country',
 'payment_method_product',
 'payment_method_provider',
 'payment_method_type'}
```

```
# Target encoding using mean
#
```

```
ce_target = ce.TargetEncoder(cols = ['category_name','city','customer_id','customer
X_train = ce_target.fit_transform(X_train, y_train)
X_val = ce_target.transform(X_val)
X_test = ce_target.transform(X_test)


# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
l=0.01
h= 1000.0
d=50.0

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
  scaled_lr.fit(X_train, y_train)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_train, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)


plt.figure()
plt.plot(list(np.arange(l,h,d)), train_scores, label="train")
plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```

```
# minority class needs more re-weighting
```

```
# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
l=0.01
h= 1000.0
d=50.0

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la, class_weight={ 0:0.
  scaled_lr.fit(X_train, y_train)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_train, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)

#plotting
plt.figure()
plt.plot(list(np.arange(l,h,d)), train_scores, label="train")
plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```

```
# minority class needs more weighting

# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score
```

```
train_scores = []
val_scores = []
scaler = StandardScaler()
l=0.01
h= 1000.0
d=50.0

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la, class_weight={ 0:0.
  scaled_lr.fit(X_train, y_train)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_train, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)

#plotting
plt.figure()
plt.plot(list(np.arange(l,h,d)), train_scores, label="train")
plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```

```
# minority class needs more weighting

# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
l=0.01
h= 10000.0 # change to 10000.0
```

```
d=500.0 # change to 500.0

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la, class_weight={ 0:0.
  scaled_lr.fit(X_train, y_train)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_train, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)

#plotting
plt.figure()
plt.plot(list(np.arange(l,h,d)), train_scores, label="train")
plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```

```
best_idx = np.argmax(val_scores)
print(val_scores[best_idx])
```

```
0.6283662477558348
```

```
# Model with lambda_best
best_idx = np.argmax(val_scores)
l_best = l+d*best_idx
scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:
scaled_lr.fit(X_train, y_train)

y_pred_test = scaled_lr.predict(X_test)
test_score = f1_score(y_test, y_pred_test)

print(test_score)
```

```
0.6007751937984496
```

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[13549,   175],
       [   31,   155]])
```

```
# minority class needs more weighting

# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
l=0.01
h= 10000.0 # change to 10000.0
d=500.0 # change to 500.0

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la, class_weight={ 0:0.
  scaled_lr.fit(X_train, y_train)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_train, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)

#plotting
plt.figure()
plt.plot(list(np.arange(l,h,d)), train_scores, label="train")
plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```

```
# Alternatives metrics for imbalanced data:
# Brier-Score: 1/n*SUM[Y_i - Y_i_hat]^2 = AVG Squared Error
# G-mean: SQRT(Sensitivity * Specificity)=SQRT(TPR*TNR) ----Seldom used
```

✓ 0s completed at 22:24