Data Scientist Technical Challenge

Debido al incremento de fraude en los sistemas bancarios y plataformas de pago, se debe buscar un modelo que me permita detectar de forma fácil y eficiente que permita prevenir los casos de fraude.

Desarrollo:

Para este propósito se plantearán modelos de clasificación, esto se decide debido a que el set de datos cuenta con variables discretas (binarias, si/no, etc), este tipo de modelos es la manera mas optima de buscar un proceso que permita detectar y prevenir un fraude.

Los pasos a seguir son:

1. Importar librerías requeridas para el desarrollo del ejercicio:
    a. Instalación desde PIP.

```
In [1]: pip install xgboost

Requirement already satisfied: xgboost in c:\anaconda\lib\site-packages (1.4.2)
Requirement already satisfied: scipy in c:\anaconda\lib\site-packages (from xgboost) (1.3.1)
Requirement already satisfied: numpy in c:\anaconda\lib\site-packages (from xgboost) (1.16.5)
Note: you may need to restart the kernel to use updated packages.
```

```
In [2]: pip install seaborn

Requirement already satisfied: seaborn in c:\anaconda\lib\site-packages (0.9.0)
Requirement already satisfied: matplotlib>=1.4.3 in c:\anaconda\lib\site-packages (from seaborn) (3.1.1)
Requirement already satisfied: pandas>=0.15.2 in c:\anaconda\lib\site-packages (from seaborn) (0.25.1)
Requirement already satisfied: scipy>=0.14.0 in c:\anaconda\lib\site-packages (from seaborn) (1.3.1)
Requirement already satisfied: numpy>=1.9.3 in c:\anaconda\lib\site-packages (from seaborn) (1.16.5)
Requirement already satisfied: cycler>=0.10 in c:\anaconda\lib\site-packages (from matplotlib>=1.4.3->seaborn) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\anaconda\lib\site-packages (from matplotlib>=1.4.3->seaborn) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\anaconda\lib\site-packages (from matplotlib>=1.4.
3->seaborn) (2.4.2)
Requirement already satisfied: python-dateutil>=2.1 in c:\anaconda\lib\site-packages (from matplotlib>=1.4.3->seaborn) (2.8.0)
Requirement already satisfied: pytz>=2017.2 in c:\anaconda\lib\site-packages (from pandas>=0.15.2->seaborn) (2019.3)
Requirement already satisfied: six in c:\users\wsepulveda\appdata\roaming\python\python37\site-packages (from cycler>=0.10->mat
plotlib>=1.4.3->seaborn) (1.13.0)
Requirement already satisfied: setuptools in c:\anaconda\lib\site-packages (from kiwisolver>=1.0.1->matplotlib>=1.4.3->seaborn)
(41.4.0)
Note: you may need to restart the kernel to use updated packages.
```

```
In [3]: pip install scikit-learn==0.20.4

Requirement already satisfied: scikit-learn==0.20.4 in c:\anaconda\lib\site-packages (0.20.4)
Requirement already satisfied: scipy>=0.13.3 in c:\anaconda\lib\site-packages (from scikit-learn==0.20.4) (1.3.1)
Requirement already satisfied: numpy>=1.8.2 in c:\anaconda\lib\site-packages (from scikit-learn==0.20.4) (1.16.5)
Note: you may need to restart the kernel to use updated packages.
```

```
In [4]: #Importar Paquetes

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from termcolor import colored as cl
import itertools

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

from numpy import mean
from numpy import std
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

2. Importar datos y realizar transformaciones iniciales: después de importar los datos, se hace una conversión a numérico de las variables Q, R, S y Monto.

```
In [5]: #Importar Datos y transformaciones inciales

df = pd.read_csv('Data_Scientist_Technical_Challenge.csv')

df['Monto'] = pd.to_numeric(df['Monto'],errors = 'coerce')
df['Q'] = pd.to_numeric(df['Q'],errors = 'coerce')
df['R'] = pd.to_numeric(df['R'],errors = 'coerce')
df['S'] = pd.to_numeric(df['S'],errors = 'coerce')

print(df.head())
```

```
   A   B        C  D  E    F    G  H  I   J  ...  L  M  N  O  P    Q    R  \
0  0  10  50257.0  0  0  0.0  0.0  0  0  UY  ...  0  3  1  0  5  0.0  0.0
1  0  10  29014.0  0  0  0.0  0.0  0  0  UY  ...  0  1  1  0  3  0.0  0.0
2  0   7     92.0  0  1  0.0  0.0  0  1  UY  ...  0  3  1  0  2  0.0  0.0
3  9  16  50269.0  0  0  0.0  0.0  0  0  UY  ...  0  3  1  0  5  0.0  0.0
4  0   8   8180.0  0  0  0.0  0.0  0  0  UY  ...  0  1  1  0  1  0.0  0.0

       S   Monto  Fraude
0   7.25   37.51       1
1  11.66    8.18       1
2  86.97   13.96       1
3   2.51   93.67       1
4  25.96  135.40       1

[5 rows x 21 columns]
```

Se crea la variable Country, variable sintética donde se codifica los valores originales de J

```
In [6]: import pandas as pd
        import numpy as np
        from sklearn.preprocessing import LabelEncoder
        labelencoder = LabelEncoder()

        df['Country'] = labelencoder.fit_transform(df['J'])
        print(df.head())
```

```
   A   B        C  D  E    F    G  H  I   J  ...  M  N  O  P    Q    R      S  \
0  0  10  50257.0  0  0  0.0  0.0  0  0  UY  ...  3  1  0  5  0.0  0.0   7.25
1  0  10  29014.0  0  0  0.0  0.0  0  0  UY  ...  1  1  0  3  0.0  0.0  11.66
2  0   7     92.0  0  1  0.0  0.0  0  1  UY  ...  3  1  0  2  0.0  0.0  86.97
3  9  16  50269.0  0  0  0.0  0.0  0  0  UY  ...  3  1  0  5  0.0  0.0   2.51
4  0   8   8180.0  0  0  0.0  0.0  0  0  UY  ...  1  1  0  1  0.0  0.0  25.96

    Monto  Fraude  Country
0   37.51       1       18
1    8.18       1       18
2   13.96       1       18
3   93.67       1       18
4  135.40       1       18

[5 rows x 22 columns]
```

3. Exploración de Datos: se observa un porcentaje elevado de nulos en la variable K.

```
In [8]: # Cantidad de NA total
        (df.isna().sum()/len(df)*100).round(1)

Out[8]: A           0.0
        B           0.0
        C          18.9
        D           0.0
        E           0.0
        F           0.0
        G           0.0
        H           0.0
        I           0.0
        J           0.0
        K          76.2
        L           0.0
        M           0.0
        N           0.0
        O           0.0
        P           0.0
        Q           0.1
        R           0.0
        S           0.0
        Monto       1.4
        Fraude      0.0
        Country     0.0
        dtype: float64
```

In [9]:
```python
# División variables cuantitativas y cualitativas

cuantitativas = df.select_dtypes(include = ['int64', 'float64','int32'])
cuantitativas.drop('Fraude', axis = 1, inplace = True)
cuantitativas.drop('Country', axis = 1, inplace = True)

print()
print("Estadísticos descriptivos variables cuantitativas:")
print('=================================================')
print()

cuantitativas.describe().round(2)
```

```
Estadísticos descriptivos variables cuantitativas:
=================================================
```

Out[9]:

D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	Monto
16880.00	16880.00	16880.00	16880.00	16880.00	16880.00	4016.00	16880.00	16880.00	16880.00	16880.00	16880.00	16856.00	16874.00	16880.00	16642.00
0.20	0.43	0.02	0.01	0.05	0.14	0.68	0.43	1.54	1.09	0.01	1.63	6.34	1.48	29.13	140.50
2.04	1.54	0.10	0.06	0.53	0.82	0.15	0.66	1.02	0.41	0.12	1.09	46.75	25.55	26.51	160.95
0.00	0.00	0.00	0.00	0.00	0.00	0.12	0.00	1.00	1.00	0.00	1.00	0.00	0.00	-1.00	0.05
0.00	0.00	0.00	0.00	0.00	0.00	0.58	0.00	1.00	1.00	0.00	1.00	0.00	0.00	9.56	33.35
0.00	0.00	0.00	0.00	0.00	0.00	0.68	0.00	1.00	1.00	0.00	1.00	0.00	0.00	20.64	79.64
0.00	0.00	0.00	0.00	0.00	0.00	0.80	1.00	2.00	1.00	0.00	2.00	0.00	0.00	39.21	190.74
180.00	45.00	1.00	1.00	21.00	24.00	0.99	7.00	13.00	10.00	3.00	41.00	984.42	984.44	99.97	998.11

In [10]:
```python
print()
print("Estadísticos descriptivos variables cualitativas:")
print('=====================================================')
print()

cualitativas = df.select_dtypes(include = ['object'])

cualitativas.describe().round(2)
```

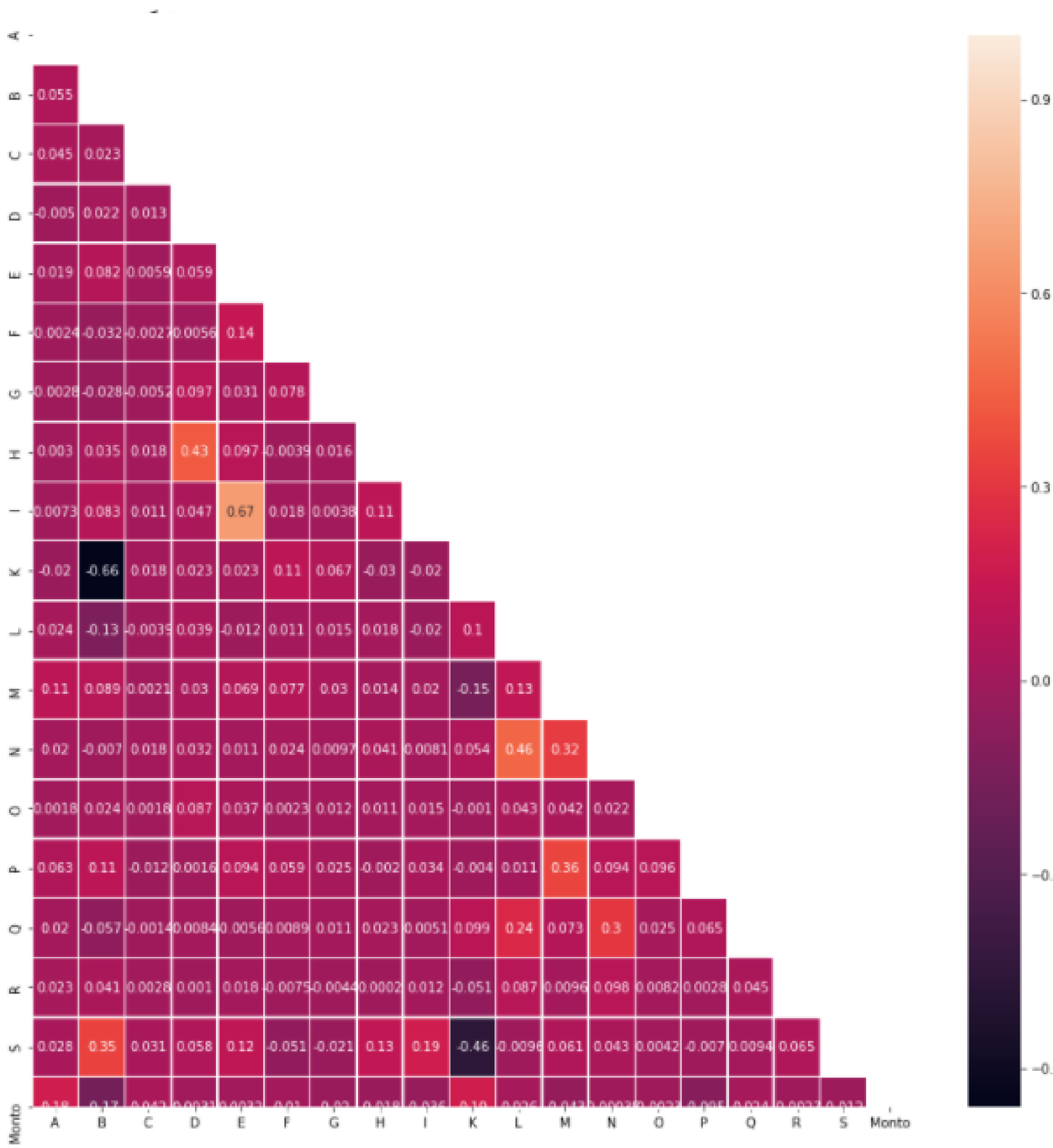
```
Estadísticos descriptivos variables cualitativas:
=====================================================
```

Out[10]:

	J
count	16880
unique	19
top	AR
freq	9329

Matriz de correlación: se observa una relación moderada entre algunas variables de estudio, pero en su mayoría son correlaciones muy bajas entre ellas, se debe determinar a que hacen referencia estas variables par dar un diagnostico mas preciso.

In [31]:
```
# Matriz de Correlación
import seaborn as sn
corrMatrix = cuantitativas.corr()
plt.figure(figsize = (15,15))
mask = np.triu(np.ones_like(corrMatrix, dtype = bool))
sn.heatmap(corrMatrix, annot=True, linewidths=.5,mask=mask)
plt.savefig('Matriz de Correlación.png')
plt.show()
```



In [12]:
```python
#Exploración de datos

# Validación Numerica

Registros = len(df)
Total_NoFraudes = len(df[df.Fraude == 0])
Total_Fraudes = len(df[df.Fraude == 1])
Porcentaje_Fraude = round(Total_Fraudes/Total_NoFraudes*100, 2)

print(cl('----'))
print(cl('Total de Registros {}'.format(Registros), attrs = ['bold']))
print(cl('Total de Registros no Fraudelentos {}'.format(Total_NoFraudes), attrs = ['bold']))
print(cl('Total de Registros Fraudelentos {}'.format(Total_Fraudes), attrs = ['bold']))
print(cl('Porcentaje de Casos Fraudulentos {}'.format(Porcentaje_Fraude), attrs = ['bold']))
print(cl('----'))

# Descripción

Registros_NoFraudulentos = df[df.Fraude == 0]
Registros_Fraudulentos = df[df.Fraude == 1]

print(cl('Stats Registros No-Fraudulentos', attrs = ['bold']))
print(Registros_NoFraudulentos.Monto.describe())
print(cl('----'))
print(cl('Stats Registros Fraudulentos', attrs = ['bold']))
print(Registros_Fraudulentos.Monto.describe())
print(cl('----'))
```

```
----
Total de Registros 16880
Total de Registros no Fraudelentos 12269
Total de Registros Fraudelentos 4611
Porcentaje de Casos Fraudulentos 37.58
----
Stats Registros No-Fraudulentos
count    12078.000000
mean       147.775621
std        167.903794
min          0.050000
25%         32.905000
50%         85.035000
75%        196.980000
max        998.110000
Name: Monto, dtype: float64
----
Stats Registros Fraudulentos
count    4564.000000
mean      121.249154
std       139.101256
min         1.580000
25%        34.305000
50%        69.060000
75%       158.090000
max       977.040000
Name: Monto, dtype: float64
----
```

4. Normalización de Datos:

```
In [15]: #Normalización de Datos
         df.drop('J', axis = 1, inplace = True)

         sc = StandardScaler()
         df_SS = sc.fit_transform(df[df.columns[0:19]])
         df_SS
```

```
Out[15]: array([[-0.2826625 ,  0.50244186,  0.12106137, ..., -0.05796642,
                 -0.82527032, -0.63990639],
                [-0.2826625 ,  0.50244186, -0.11227069, ..., -0.05796642,
                 -0.65891651, -0.82214058],
                [-0.2826625 , -0.13771453, -0.42994851, ..., -0.05796642,
                  2.1819237 , -0.78622808],
                ...,
                [ 0.63138789, -0.99125638,  0.45947688, ..., -0.05796642,
                 -0.18739211,  0.34253041],
                [-0.2826625 ,  0.2890564 ,  3.94473927, ..., -0.05796642,
                 -0.00481786, -0.64326153],
                [-0.2826625 ,  0.92921279, -0.26479383, ..., -0.05796642,
                  2.18607311, -0.73888323]])
```

5. Preparación de Datos:

```
In [23]: # Preparación de datos para modelado

         X = df3.drop('Fraude', axis = 1).values
         y = df3['Fraude'].values

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 10)

         print(cl('X_train samples : ', attrs = ['bold']), X_train[:1])
         print(cl('X_test samples : ', attrs = ['bold']), X_test[0:1])
         print(cl('y_train samples : ', attrs = ['bold']), y_train[0:10])
         print(cl('y_test samples : ', attrs = ['bold']), y_test[0:10])
```

```
X_train samples :  [[-0.2826625   1.78275464 -0.12700017 -0.09726214 -0.28213252 -0.16083165
  -0.08752131 -0.09471186 -0.1746932  -0.65128035 -0.53223723 -0.22487747
  -0.08078252  1.25810281 -0.13569334 -0.05796642 -0.21681523 -0.72490345
   0.        ]]
X_test samples :  [[-2.82662503e-01 -5.64485454e-01 -9.42022167e-18 -9.72621408e-02
  -2.82132520e-01 -1.60831654e-01 -8.75213126e-02 -9.47118620e-02
  -1.74693200e-01 -6.51280349e-01 -5.32237234e-01 -2.24877469e-01
  -8.07825154e-02 -5.79703900e-01 -1.35693337e-01 -5.79664189e-02
   1.44561068e-01  1.55380068e+00  0.00000000e+00]]
y_train samples :  [1 0 0 1 0 0 1 0 1 1]
y_test samples :  [0 0 0 0 1 0 0 0 1 0]
```

6. Creación de Modelos (Decision Tree, K-NN, Logistic Regression, XGBoost)

In [24]:
```
# Modelado

#Decision Tree

tree_model = DecisionTreeClassifier(max_depth = 5)
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)

#K-NN

n = 5

knn = KNeighborsClassifier(n_neighbors = n)
knn.fit(X_train, y_train)
knn_yhat = knn.predict(X_test)

#Logistic Regression

lr = LogisticRegression(max_iter=1000,
                        solver='lbfgs')
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)

#XGBoost

xgb = XGBClassifier(eval_metric='mlogloss',
                    use_label_encoder=False)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)
```

7. Validación F1 Score:

In [25]:
```
# Validación F1 score

print(cl('----'))
print(cl('Validación F1 Score', attrs = ['bold']))
print(cl('----'))
print(cl('F1 score Decision Tree: {}'.format(f1_score(y_test, tree_yhat)), attrs = ['bold']))
print(cl('----'))
print(cl('F1 score KNN: {}'.format(f1_score(y_test, knn_yhat)), attrs = ['bold']))
print(cl('----'))
print(cl('F1 score Logistic Regression: {}'.format(f1_score(y_test, lr_yhat)), attrs = ['bold']))
print(cl('----'))
print(cl('F1 score XGBoost: {}'.format(f1_score(y_test, xgb_yhat)), attrs = ['bold']))
print(cl('----'))
```

```
----
Validación F1 Score
----
F1 score Decision Tree: 0.46423562412342223
----
F1 score KNN: 0.4963981663392272
----
F1 score Logistic Regression: 0.3691588785046729
----
F1 score XGBoost: 0.5650708024275118
----
```

8. Validación Cruzada por modelo

In [26]:
```
# Validación Cruzada por modelo

kfold = StratifiedKFold(n_splits=10, random_state=5)

scores = cross_val_score(tree_model, X_train, y_train, scoring='accuracy', cv=kfold, n_jobs=-1)
print("Presición Decision Tree: mean:(%.1f%%) std:(%.1f%%)" % (scores.mean()*100, scores.std()*100))
print(cl('----'))
scores = cross_val_score(knn, X_train, y_train, scoring='accuracy', cv=kfold, n_jobs=-1)
print("Presición knn: mean:(%.1f%%) std:(%.1f%%)" % (scores.mean()*100, scores.std()*100))
print(cl('----'))
scores = cross_val_score(lr, X_train, y_train, scoring='accuracy', cv=kfold, n_jobs=-1)
print("Presición Logistic Regression: mean:(%.1f%%) std:(%.1f%%)" % (scores.mean()*100, scores.std()*100))
print(cl('----'))
scores = cross_val_score(xgb, X_train, y_train, scoring='accuracy', cv=kfold, n_jobs=-1)
print("Presición XGBoost: mean:(%.1f%%) std:(%.1f%%)" % (scores.mean()*100, scores.std()*100))
print(cl('----'))
```

```
Presición Decision Tree: mean:(76.3%) std:(1.0%)
----
Presición knn: mean:(76.9%) std:(1.0%)
----
Presición Logistic Regression: mean:(74.9%) std:(0.8%)
----
Presición XGBoost: mean:(80.1%) std:(0.6%)
----
```

9. Conclusión: después de hacer una valoración de los modelos y validar las variables mas importantes proporcionadas, se determina que:
    a. El modelo que me permite tener mejor desempeño y que me ayudara mejorar en la búsqueda de prevenir fraudes es el XGBoost, seguido de el árbol de decisión.
    b. Las variables mas relevantes son las variables Country (variable original J) y Monto, lo que tiene mucho sentido ya que dependiendo del país y sus condiciones sociodemográficas puede tener una gran influencia en los resultados del modelo.
    c. Se necesita hacer un trabajo de balanceo en la data y validar con el negocio cual es la mejor manera de imputar los valores N/A o vacíos ya que esto puede tener y tiene una gran influencia en los resultados.
    d. Se desconoce si las variables proporcionadas en el set de datos son fruto del calculo de otras variables por lo que para este ejercicio se toman todas las variables cuantitativas proporcionadas, se recomienda hacer un estudio de las variables origen para determinar si estas pueden ser o no usadas de manera independiente y de esta manera enriquecer el modelo.