Proyecto Integrador

Maestría de Ciencias de Datos e Información

Integrantes:

Karen Itzel Velasco Chincoya,
Martha Karina Terán Botello,
Yuridiana de Jesús Reyes Delgado,

Modelos Avanzados en Ciencia de Datos

Octubre 10, 2022

Contenido

Introducción

Datos generales del problema:

</br> **Proyecto Elegido** </br>

- **Nombre:** American Express - Default Prediction
- **Enlace:** https://www.kaggle.com/competitions/amex-default-prediction

Descripción del problema

El proyecto consiste en la predicción de incumplimiento de pago en las tarjetas de crédito basándonos en un histórico de información de 18 meses, condicionada a un plazo de 120 días para poder realizar el pago del saldo pendiente, en donde se incluyen variables categóricas como:

- Delincuencia
- Pagos
- Gastos
- Equilibrio
- Riesgo

Los cuales se encuentran segmentados en entrenamiento, validación y conjuntos de prueba que incluyen comportamientos de series de tiempo y la anonimidad de la información. Estas características están compartidas con el proyecto de investigación gracias a las series de tiempo y los datos existentes, además de que los datos corresponden a personas.

Justificación

Desarrollo

Estado del arte

Historial: trabajos previos que sugieren soluciones

Predicting Credit Card Defaults with Machine Learning

En este estudio de caso se menciona de nuevo el impacto de este tema en la vida real debido al aumento de estafas a los bancos a través de los créditos. Los datos que utilizaron fueron del repositorio de UC Irvine. Las visualizaciones fueron obtenidas de kaggle, por su parte, el modelo para la solución fue escrito utilizando python, considerando la optimización y el performance en todo el proceso. Al final, ellos obtuvieron un porcentaje de confianza del 95%, y en sus palabras, este porcentaje puede ahorrar grandes cantidades de dinero.[1]

Credit Default Risk Prediction

La compañía alemana Record Evolution especializada en ciencia de datos e internet de las cosas, en su estudio Credit Default Risk Prediction explican a detalle el procedimiento para predecir el riesgo de incumplimiento de pago usando machine learning. Abordan los diferentes factores a considerar para construir un modelo de machine learning que tenga éxito, ya que no depende solamente de la selección del método, sino que hay factores clave que contribuyen: no

se puede obtener un modelo confiable de alto rendimiento del modelo de predicción sin una cantidad suficiente de datos enriquecidos, el feature engineering que se debe aplicar, ya que consume la mayor parte del tiempo de un problema de machine learning, la elección del modelo de machine learning y las métricas de desempeño. Usaron como ejemplo el conjunto de datos Home Credit Default Risk de Kaggle y compararon tres principales algoritmos: regresión logística, árbol de decisión y árbol de decisión con boost, mencionando los pros y contras de cada uno.

Optimize Cash Collection: Use Machine learning to Predicting Invoice Payment

En el paper llamado "Optimize Cash Collection: Use Machine learning to Predicting Invoice Payment" se menciona como objetivo identificar el pago de facturas para respaldar la toma de decisiones, presentando un prototipo que apoyaba a los encargados de cobrar, alcanzando un 77% de precisión usando cinco clasificadores como: Naive Bayes, Logistic Regression, "K-Nearest Neighbors, Random Forest" y "Gardient Boosted Decision Trees". Esto para poder atender a los clientes que tengan dificultad de pago y poder concentrarse en ellos y no descartarlos como clientes; podría segmentarse para atenderlos de diferente manera sin perder rentabilidad como negocio.[3]

Solución

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime

from imblearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing, decomposition, impute
from sklearn.experimental import enable_iterative_imputer
```

In [2]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Extracción de los datos

En este trabajo se uso una muestra de 10,000 registros para los siguientes algoritmos, ya que el volumen de datos dificulta su procesamiento de manera local y dificulta la colaboración.

Columnas: Se eliminaron las columnas que no tuvieran al menos 75% de valores, eliminando 23 Columnas.

PCA: Para trabajar con el número de columnas del dataset, se aplica Principal Componen Analysis para realizar una reducción.

In [38]:
```
train_df_sample = pd.read_csv('content/train_df_sample.csv')
train_label_df = pd.read_csv('content/train_label_df.csv')
train_df = pd.merge(train_df_sample, train_label_df, how="inner", on=["customer_
test_df = pd.read_csv('content/test_df.csv')
sample_submission_df = pd.read_csv('content/sample_submission_df-2.csv')
```

Exploración de los datos

In [39]:
```
print("-------- Archivo de Entrenamiento ----------")
print("Filas:",train_df.shape[0]," Columnas:",train_df.shape[1])
train_df.info()
display(train_df.head())
print("\n")


print("-------- Archivo de Pruebas ----------")
print("Filas:",test_df.shape[0]," Columnas:",test_df.shape[1])
test_df.info()
display(test_df.head())
```

```
-------- Archivo de Entrenamiento ----------
Filas: 10000  Columnas: 193
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 9999
Columns: 193 entries, Unnamed: 0_x to target
dtypes: float64(185), int64(4), object(4)
memory usage: 14.8+ MB
```

	Unnamed: 0_x	customer_ID	S_2	P_2	D_39	
0	0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-03-09	0.938469	0.001733	0.
1	1	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-04-07	0.936665	0.005775	0.
2	2	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-05-28	0.954180	0.091505	0
3	3	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-06-13	0.960384	0.002455	0.
4	4	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-07-16	0.947248	0.002483	0

5 rows × 193 columns

```
-------- Archivo de Pruebas ----------
Filas: 10000  Columnas: 191
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 191 entries, Unnamed: 0 to D_145
dtypes: float64(185), int64(2), object(4)
memory usage: 14.6+ MB
```

	Unnamed: 0	customer_ID	S_2	P_2	D_39
0	0	00000469ba478561f23a92a868bd366de6f6527a684c9a...	2019-02-19	0.631315	0.001912
1	1	00000469ba478561f23a92a868bd366de6f6527a684c9a...	2019-03-25	0.587042	0.005275
2	2	00000469ba478561f23a92a868bd366de6f6527a684c9a...	2019-04-25	0.609056	0.003326
3	3	00000469ba478561f23a92a868bd366de6f6527a684c9a...	2019-05-20	0.614911	0.009065
4	4	00000469ba478561f23a92a868bd366de6f6527a684c9a...	2019-06-15	0.591673	0.238794

5 rows × 191 columns

In [12]:

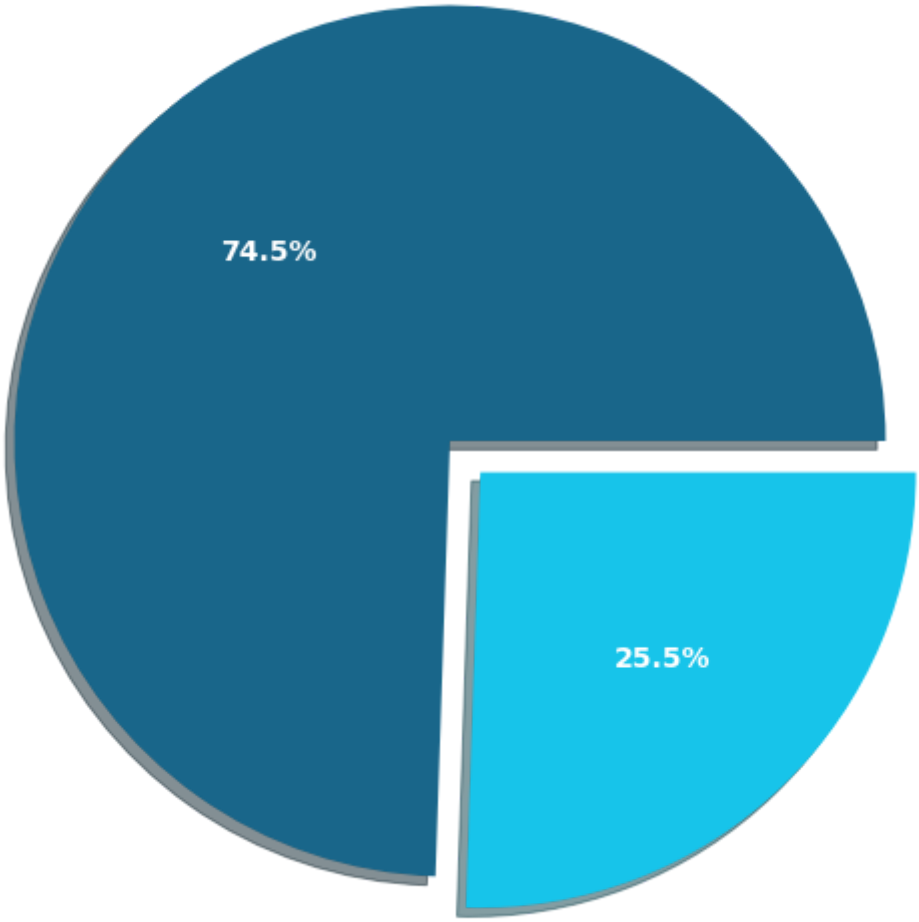
```
clientes = train_df.groupby('target').count().reset_index()
clientes['Conteo'] = clientes['customer_ID']
clientes = clientes.filter(['target','Conteo'])
clientes
```

Out[12]:

	target	Conteo
0	0	7447
1	1	2553

In [13]:

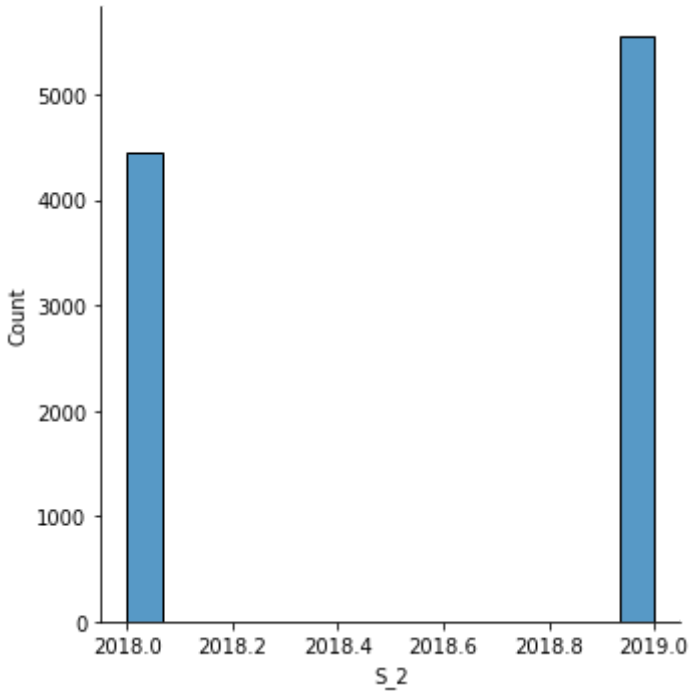
```
plt.figure(figsize=(20,10))
plt.pie(clientes['Conteo'],
        labels=clientes['target'],
        colors=['#19668A','#17C4EA'],
        shadow=True,
        explode=[0, 0.1],
        autopct='%1.1f%%',
        textprops=dict(color="w", weight="bold", size=14)
       )
plt.show()
```



In [14]:
```
anios = pd.to_datetime(test_df['S_2']).dt.year
anios
```

Out[14]:
```
0       2019
1       2019
2       2019
3       2019
4       2019
        ...
9995    2018
9996    2018
9997    2018
9998    2019
9999    2019
Name: S_2, Length: 10000, dtype: int64
```

In [15]:
```
grafica_barras = sns.displot(anios)
```



Relación de la variable objetivo con las demás variables presentes

La relación es que las diferentes variables indicadas de los rubros de **Delincuencia (D), Gasto (S), Pago (P), Saldo (B) y Riesgo (R)** nos permiten calificar la capacidad de pago de un cliente o no.

Las registros que tienen un target=1(Pago) representa el 25.4% mientras que los que tienen un target=0(No pago) son el 74.6%

Variables mayormente correlacionadas con el objetivo

In [41]:
```
columnas_categoricas = train_df[['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_1
columnas_categoricas
```

Out[41]:

	B_30	B_38	D_114	D_116	D_117	D_120	D_126	D_63	D_64	D_68
0	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
1	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
2	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
3	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
4	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
...	...	...	...	...	...	...	...	...	...	...
9995	0.0	3.0	0.0	0.0	-1.0	0.0	1.0	CR	O	6.0
9996	0.0	3.0	0.0	0.0	-1.0	0.0	1.0	CR	O	6.0
9997	0.0	2.0	1.0	0.0	-1.0	0.0	1.0	CO	O	6.0

	B_30	B_38	D_114	D_116	D_117	D_120	D_126	D_63	D_64	D_68
9998	0.0	3.0	1.0	0.0	-1.0	0.0	1.0	CO	O	6.0
9999	0.0	3.0	1.0	0.0	-1.0	0.0	1.0	CO	O	6.0

10000 rows × 10 columns

In [42]:
```
columnas_categoricas.isnull().sum()
columnas_categoricas
```

Out[42]:

	B_30	B_38	D_114	D_116	D_117	D_120	D_126	D_63	D_64	D_68
0	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
1	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
2	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
3	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
4	0.0	2.0	1.0	0.0	4.0	0.0	1.0	CR	O	6.0
...	...	...	...	...	...	...	...	...	...	...
9995	0.0	3.0	0.0	0.0	-1.0	0.0	1.0	CR	O	6.0
9996	0.0	3.0	0.0	0.0	-1.0	0.0	1.0	CR	O	6.0
9997	0.0	2.0	1.0	0.0	-1.0	0.0	1.0	CO	O	6.0
9998	0.0	3.0	1.0	0.0	-1.0	0.0	1.0	CO	O	6.0
9999	0.0	3.0	1.0	0.0	-1.0	0.0	1.0	CO	O	6.0

10000 rows × 10 columns

In [43]:
```
LabelEncoder = preprocessing.LabelEncoder()
columnas_categoricas = columnas_categoricas.apply(LabelEncoder.fit_transform)
columnas_categoricas
```

Out[43]:

	B_30	B_38	D_114	D_116	D_117	D_120	D_126	D_63	D_64	D_68
0	0	1	1	0	4	0	2	2	1	6
1	0	1	1	0	4	0	2	2	1	6
2	0	1	1	0	4	0	2	2	1	6
3	0	1	1	0	4	0	2	2	1	6
4	0	1	1	0	4	0	2	2	1	6
...	...	...	...	...	...	...	...	...	...	...
9995	0	2	0	0	0	0	2	2	1	6
9996	0	2	0	0	0	0	2	2	1	6
9997	0	1	1	0	0	0	2	1	1	6
9998	0	2	1	0	0	0	2	1	1	6

	B_30	B_38	D_114	D_116	D_117	D_120	D_126	D_63	D_64	D_68
9999	0	2	1	0	0	0	2	1	1	6

10000 rows × 10 columns

In [44]:
```
Encoder = preprocessing.OneHotEncoder()
Encoder.fit(columnas_categoricas)
ohl = Encoder.transform(columnas_categoricas).toarray()
ohl
```

Out[44]:
```
array([[1., 0., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 1., 0.],
       ...,
       [1., 0., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 1., 0.]])
```

In [45]:
```
nombres_columnas_categoricas = Encoder.get_feature_names_out(list(columnas_categ
nombres_columnas_categoricas
```

Out[45]:
```
array(['B_30_0', 'B_30_1', 'B_30_2', 'B_38_0', 'B_38_1', 'B_38_2',
       'B_38_3', 'B_38_4', 'B_38_5', 'B_38_6', 'D_114_0', 'D_114_1',
       'D_114_2', 'D_116_0', 'D_116_1', 'D_116_2', 'D_117_0', 'D_117_1',
       'D_117_2', 'D_117_3', 'D_117_4', 'D_117_5', 'D_117_6', 'D_117_7',
       'D_120_0', 'D_120_1', 'D_120_2', 'D_126_0', 'D_126_1', 'D_126_2',
       'D_126_3', 'D_63_0', 'D_63_1', 'D_63_2', 'D_63_3', 'D_63_4',
       'D_63_5', 'D_64_0', 'D_64_1', 'D_64_2', 'D_64_3', 'D_64_4',
       'D_68_0', 'D_68_1', 'D_68_2', 'D_68_3', 'D_68_4', 'D_68_5',
       'D_68_6', 'D_68_7'], dtype=object)
```

In [46]:
```
df_cc = pd.DataFrame(ohl,columns=nombres_columnas_categoricas)
df_cc
```

Out[46]:

	B_30_0	B_30_1	B_30_2	B_38_0	B_38_1	B_38_2	B_38_3	B_38_4	B_38_5	B_38_6	..
0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	..
1	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	..
2	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	..
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	..
4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	..
...	...	...	...	...	...	...	...	...	...	...	..
9995	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	..
9996	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	..
9997	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	..
9998	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	..
9999	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	..

10000 rows × 50 columns

In [47]:
```
train_df = train_df.drop(columns = columnas_categoricas.columns)
train_df = train_df.join(df_cc)
train_df
```

Out[47]:

	Unnamed: 0_x	customer_ID	S_2	P_2	D_3
0	0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-03-09	0.938469	0.00173
1	1	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-04-07	0.936665	0.00577
2	2	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-05-28	0.954180	0.09150
3	3	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-06-13	0.960384	0.00245
4	4	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-07-16	0.947248	0.00248
...	...	...	...	...	.
9995	9995	007793144e0eeef1e29a7aa93244815328beb0d46ccbe3...	2018-02-26	0.331150	1.47202
9996	9996	007793144e0eeef1e29a7aa93244815328beb0d46ccbe3...	2018-03-19	0.361060	2.06556
9997	9997	0077b7efafef4cfa81e42538b87e39799e9928e47c6826...	2017-03-28	0.708199	0.35734
9998	9998	0077b7efafef4cfa81e42538b87e39799e9928e47c6826...	2017-04-15	0.692401	0.00938
9999	9999	0077b7efafef4cfa81e42538b87e39799e9928e47c6826...	2017-05-31	0.779090	0.44505

10000 rows × 233 columns

In [48]:
```
train_df_x = train_df.drop(columns=['target'])
train_df_y = train_df[['target']]
```

In [49]:
```
train_df_PCA = train_df_x.drop(columns=['customer_ID','S_2'])
```

In [50]:
```
mv = impute.IterativeImputer()
```

In [51]:
```
train_df_x_impute = mv.fit_transform(train_df_PCA)
train_df_x_impute
```

Out[51]:
```
array([[0.00000000e+00, 9.38468719e-01, 1.73333900e-03, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 9.36664605e-01, 5.77544307e-03, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [2.00000000e+00, 9.54180277e-01, 9.15053968e-02, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       ...,
       [9.99700000e+03, 7.08198594e-01, 3.57349217e-01, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [9.99800000e+03, 6.92401424e-01, 9.38038431e-03, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [9.99900000e+03, 7.79089561e-01, 4.45051691e-01, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00]])
```

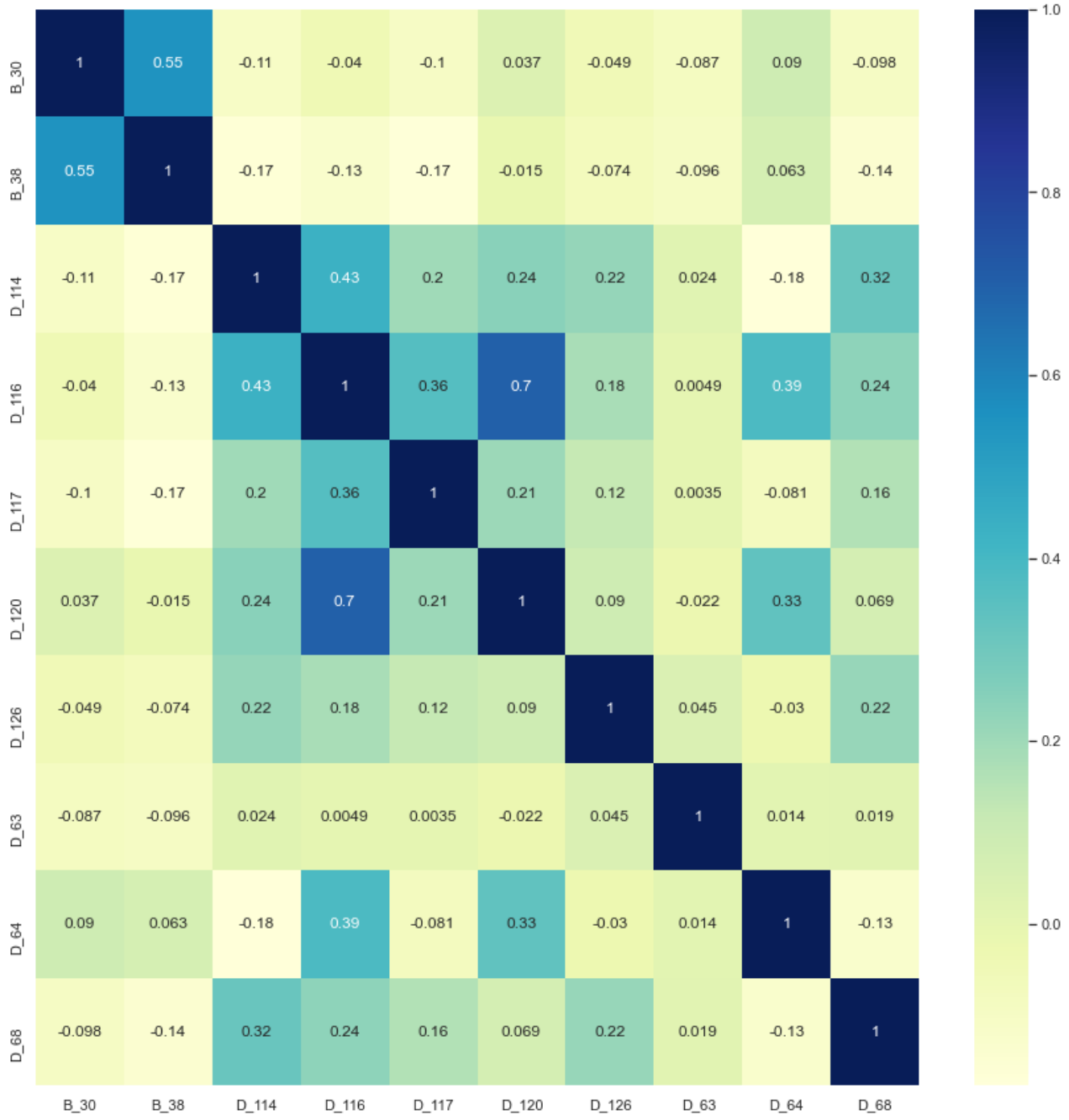
In [52]:
```
train_df_x_impute_df = pd.DataFrame(train_df_x_impute,columns=list(train_df_PCA.
```

In [53]:
```
correlation = columnas_categoricas.corr()
correlation
```

Out[53]:

	B_30	B_38	D_114	D_116	D_117	D_120	D_126	D_63
B_30	1.000000	0.552650	-0.107343	-0.039710	-0.101645	0.036912	-0.048972	-0.087249
B_38	0.552650	1.000000	-0.165780	-0.127780	-0.168237	-0.015177	-0.074441	-0.096441
D_114	-0.107343	-0.165780	1.000000	0.433167	0.199741	0.244399	0.221030	0.023940
D_116	-0.039710	-0.127780	0.433167	1.000000	0.363936	0.697978	0.181348	0.004882
D_117	-0.101645	-0.168237	0.199741	0.363936	1.000000	0.207880	0.118046	0.003531
D_120	0.036912	-0.015177	0.244399	0.697978	0.207880	1.000000	0.089669	-0.022383
D_126	-0.048972	-0.074441	0.221030	0.181348	0.118046	0.089669	1.000000	0.044564
D_63	-0.087249	-0.096441	0.023940	0.004882	0.003531	-0.022383	0.044564	1.000000
D_64	0.089850	0.062872	-0.177099	0.386644	-0.080536	0.334884	-0.029564	0.013637
D_68	-0.098154	-0.140624	0.316220	0.235788	0.158986	0.068663	0.216136	0.019347

In [54]:
```
sns.set(rc = {'figure.figsize':(15,15)})
sns.heatmap(correlation, cmap="YlGnBu", annot=True)
plt.show()
```



Aplicación de ingeniería de características para diseñar nuevas características en el conjunto de datos

Principal Components Analysis

In [55]:
```
pca = decomposition.PCA(n_components=20)
train_df_x_impute_pca = pca.fit_transform(train_df_x_impute_df)
```

In [56]:
```
train_df_x_impute_pca = pd.DataFrame(train_df_x_impute_pca,columns=[f'col_{n}' f
train_df_x_impute_pca
```

Out[56]:

	col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_
0	-5016.429389	5.029293	-0.566677	-1.759818	-0.169616	0.063979	0.728013	-1.75277
1	-5015.432777	5.042492	-0.564935	-1.770769	-0.171855	0.060605	0.722809	-1.75531
2	-5014.436167	5.065250	-0.565415	-1.789775	-0.166162	0.064811	0.679344	-1.82867
3	-5013.439551	5.071428	-0.563937	-1.761054	-0.171618	0.071222	0.751990	-1.79773
4	-5012.442939	5.088302	-0.569223	-1.804569	-0.176044	0.059975	0.733709	-1.79680
...	...	...	...	...	...	...	...	.
9995	5012.479367	5.673911	-0.593855	-1.220974	6.725236	0.194878	-0.316639	-0.26552
9996	5013.475869	5.586829	-0.565340	-1.201878	1.565576	0.071175	-0.541059	-0.03306
9997	5014.555429	-4.035355	0.057812	-0.807884	-0.157564	0.173885	0.471653	-1.55212
9998	5015.552075	-3.949420	0.072385	-0.645219	-0.108784	0.096649	0.076239	-1.12654
9999	5016.548697	-3.967084	0.073735	-0.536466	-0.127232	0.064166	-0.018809	-1.03953

10000 rows × 20 columns

In [57]:

```
train_df_pca = train_df_y.join(train_df_x[['customer_ID', 'S_2']]).join(train_df
train_df_pca
```

Out[57]:

	target	customer_ID	S_2	col_0	co
0	0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-03-09	-5016.429389	5.0292
1	0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-04-07	-5015.432777	5.0424
2	0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-05-28	-5014.436167	5.0652
3	0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-06-13	-5013.439551	5.0714
4	0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-07-16	-5012.442939	5.0883
...	...	...	...	...	
9995	1	007793144e0eeef1e29a7aa93244815328beb0d46ccbe3...	2018-02-26	5012.479367	5.673
9996	1	007793144e0eeef1e29a7aa93244815328beb0d46ccbe3...	2018-03-19	5013.475869	5.5868
9997	0	0077b7efafef4cfa81e42538b87e39799e9928e47c6826...	2017-03-28	5014.555429	-4.0353
9998	0	0077b7efafef4cfa81e42538b87e39799e9928e47c6826...	2017-04-15	5015.552075	-3.9494

	target	customer_ID	S_2	col_0	co
9999	0	0077b7efafef4cfa81e42538b87e39799e9928e47c6826...	2017-05-31	5016.548697	-3.9670

10000 rows × 23 columns

Modelado de datos

In [58]:
```
x = train_df_pca.drop(columns=['target','customer_ID','S_2'])
y = train_df_pca['target']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25, strati
```

In [59]:
```
print("X_train  Filas:",X_train.shape[0]," Columnas:",X_train.shape[1])
print("X_test Filas:",X_test.shape[0]," Columnas:",X_test.shape[1])
print("y_train Filas:",y_train.shape)
print("y_test Filas:",y_test.shape)
```

```
X_train  Filas: 7500  Columnas: 20
X_test Filas: 2500  Columnas: 20
y_train Filas: (7500,)
y_test Filas: (2500,)
```

Random Forest Classifier

Gran parte de los problemas en ciencia de datos son las clasificaciones, interesa saber a que clase o grupo pertenecen ciertas variables. La habilidad para determinar el grupo al que una variable forma parte es extremadamente valiosa. Uno de los algoritmos más exactos es el clasificador aleatorio de bosque. Árboles de decisión La base para comprender el clasificador de bosque son los árboles de decisión, lo que sucede en un árbol de decisión es evaluar el conjunto de valores iniciales por ronda, cada valor es evaluado con una pregunta, si la respuesta es sí, generalmente se va hacia la izquierda, si es no, hacia la derecha.

In [60]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators = 100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
from sklearn import metrics
print()

# using metrics module for accuracy calculation
print("Accuracy del modelo: ", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy del modelo:  0.9664
```

Logistic Regression

Este tipo de modelo generalmente se utiliza para la clasificación y el análisis predictivo. La regresión logística estima la probabilidad de que ocurra un evento, tal como votar o no votar

dado un conjunto de datos independientes. Dado que la salida es la probabilidad, la variable dependiente está contenida entre 0 y 1.

In [61]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Accuracy del modelo: ", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy del modelo:  0.8552
/Users/kvelasco/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
```

K Nearest Neighbors

In [66]:
```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)
print("Accuracy del modelo: ", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy del modelo:  0.9936
```

Conclusiones

Este trabajo permitió poner en práctica lo revisado en los cursos de la plataforma Kaggle: exploración de datos, ingeniería de características y Modelos de ML principalmente, así como la visualización de los datos. Se selección un tema donde el conjunto de datos era de un tamaño muy grande que resultaba complejo de manejar al ejecutar este notebook, por lo que se hizo un subconjunto para efectos de la codificación.

Entre las técnicas que se utilizaron se encuentra la regresión logística, la cual es útil para realizar proyecciones y pronósticos, en este caso el resultado obtenido utilizando la métrica de accuracy fue de 86%, con el fin de mejorar el resultado se considero necesario aplicar otros métodos como PCA y la correlación. El uso de PCA permitió analizar la relación entre variables, con el fin de apoyar el proceso de selección de atributos. De igual manera, se aplico la técnica de la correlación con la cual también fue posible medir la dependencia entre las variables para conocer el grado de correlación entre éstas.

Con el fin de comparar este estudio con otro modelo de ML, se utilizó el modelo de random forest, el cual según estudios de la literatura [4,5] destaca por su rendimiento en procesos de regresión y clasificación. El desempeño obtenido con random forest destacó del obtenido por el modelo de regresión lineal, logrando un 96.08%, no obstante, reconocemos que la selección de

atributos obtenida anteriormente con PCA y el modelo de correlación apoyo para el resultado de este modelo.

Pero el análisis no acaba, veamos que utilizando el modelo de K vecinos cercanos, con 3 vecinos, obtenemos una métrica de 99.36%, aúm mejor que el 96.64% del modelo de Random Forest.

A través de la realización de este ejercicio, pudimos darnos cuenta, que el mismo conjunto de datos tendrá diferentes medidas de exactitud dependiendo del método, el único punto a tener en mente ahora, es que se deben considerar todas las métricas al momento de elegir un algoritmo, incluyendo: f-score, precision, accuracy, recall.

Referencias

[1] Dominguez, M. (2021). Predicting credit cards Default with Machine Learning. Disponible en: https://medium.com/swlh/predicting-credit-card-defaults-with-machine-learning-fcc8da2fdafb

[2] Record Evolution Company. (2020) Credit Default Risk Prediction. Disponible en: https://www.record-evolution.de/en/blog/credit-default-risk-prediction/

[3] Appel, A. P., Oliveira, V., Lima, B., Louzada Malfatti, G., Figueredo de Santana, V., & De Paula, R. (2019). Optimize Cash Collection: Use Machine learning to Predicting Invoice Payment. Recuperado de https://arxiv.org/pdf/1912.10828.pdf

[4] Strobl, Carolin, et al. "Bias in random forest variable importance measures: Illustrations, sources and a solution." BMC bioinformatics 8.1 (2007): 1-21.

[5] Liu, Yanli, Yourong Wang, and Jian Zhang. "New machine learning algorithm: Random forest." International Conference on Information Computing and Applications. Springer, Berlin, Heidelberg, 2012.

[6] https://www.kaggle.com/code/girishkumarsahu/american-express-default-prediction-eda/notebook

[7] https://www.kaggle.com/code/girishkumarsahu/american-express-default-prediction-ml-model

[8] Jamieson Bolder, David. Credit-Risk Modelling: Theoretical Foundations, Diagnostic Tools, Practical Examples, and Numerical Recipes in Python. Springer 2018.

[9] IBM. (2022). What is logistic regression? Disponible en: https://www.ibm.com/topics/logistic-regression

[10] Yiu, T. (2019). Understanding random forest. Disponible en: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[11] Condusef. (2022). 3.3 millones de reclamaciones por fraude se registran en el primer semestre del año. Disponible en: https://www.condusef.gob.mx/?p=contenido&idc=448&idcat=1

- https://www.kaggle.com/code/girishkumarsahu/american-express-default-prediction-eda/notebook
- https://www.kaggle.com/code/girishkumarsahu/american-express-default-prediction-ml-model
- Jamieson Bolder, David. **Credit-Risk Modelling: Theoretical Foundations, Diagnostic Tools, Practical Examples, and Numerical Recipes in Python**. Springer 2018.

In [ ]: