Universidad Nacional de Córdoba - Diplomatura en Ciencia de Datos, Aprendizaje Automático y sus Aplicaciones

Análisis y Curación de Datos

Practico de Mentoria

Integrantes: Matias Trapaglia, Fernando Fontana, Nazareno Medrano

In [ ]:

In [21]:
```
# https://github.com/diplodatos2020/Introduccion_Mentoria/blob/master/dataset_inf_telec.csv
%load_ext autoreload
%autoreload 2

%matplotlib inline
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
```

In [23]:
```
sns.set_style('whitegrid')
sns.set(rc={'figure.figsize':(15, 5)})
filename = "https://raw.githubusercontent.com/diplodatos2020/Introduccion_Mentoria/master/dataset_inf_telec_ayc.csv"
BLUE = '#35A7FF'
RED = '#FF5964'
GREEN = '#6BF178'
YELLOW = '#FFE74C'
```

In [24]:
```
df = pd.read_csv(filename)
```

1. Importacion de los datos Elija algun PUNTO MEDICION y calcule el rango que existe en la feature FECHA HORA.

Por ejemplo, el PUNTO MEDICION ABA - Abasto Cliente

In [25]:
```
df_abasto = df[df['PUNTO MEDICION'] == "ABA - Abasto Cliente"]
df_abasto.sample(5)
```

Out[25]:

	ID EQUIPO	PUNTO MEDICION	CAPACIDAD MAXIMA [GBS]	FECHA INICIO MEDICION	FECHA HORA	FECHA FIN MEDICION	PASO	LATENCIA [MS]	% PACK LOSS
9890	25	ABA - Abasto Cliente	1.0	2020-06-04 17:00:00.000	2020-06-08 23:00:00.000	2020-06-21 19:00:00.000	7200.0	0.62217	0.0
13	25	ABA - Abasto Cliente	1.0	2020-06-04 17:00:00.000	2020-06-05 21:00:00.000	2020-06-21 19:00:00.000	7200.0	0.75549	0.0
9869	25	ABA - Abasto Cliente	1.0	2020-06-04 17:00:00.000	2020-06-07 05:00:00.000	2020-06-21 19:00:00.000	7200.0	0.72556	0.0
50	25	ABA - Abasto Cliente	1.0	2020-06-04 17:00:00.000	2020-06-08 23:00:00.000	2020-06-21 19:00:00.000	7200.0	0.62217	0.0
9960	25	ABA - Abasto Cliente	1.0	2020-06-04 17:00:00.000	2020-06-14 19:00:00.000	2020-06-21 19:00:00.000	7200.0	0.68769	0.0

In [26]:
```
# Descomentando esta linea aparece el error de querer operar con fechas sin el formato adecuado

#df_abasto['FECHA HORA'].max() - df_abasto['FECHA HORA'].min()
```

In [27]:
```
df.dtypes
```

Out[27]:
```
ID EQUIPO                  int64
PUNTO MEDICION            object
CAPACIDAD MAXIMA [GBS]   float64
FECHA INICIO MEDICION     object
FECHA HORA                object
FECHA FIN MEDICION        object
PASO                     float64
LATENCIA [MS]            float64
% PACK LOSS              float64
INBOUND [BITS]           float64
OUTBOUND [BITS]          float64
MEDIDA                    object
dtype: object
```

Los campos object generalmente son String, entonces parece que no reconoció como fechas en "FECHA_HORA","FECHA_INICIO_MEDICION","FECHA_FIN_MEDICION" :

In [28]:
```
df = pd.read_csv(filename,parse_dates=["FECHA HORA","FECHA INICIO MEDICIO
N","FECHA FIN MEDICION"])
df.describe(include='all')
```

Out[28]:

	ID EQUIPO	PUNTO MEDICION	CAPACIDAD MAXIMA [GBS]	FECHA INICIO MEDICION	FECHA HORA	FECHA FIN MEDICION	PASO	LATENCIA [MS]
count	19680.000000	19680	19680.000000	19680	19680	19680	18505.0	18485.000000
unique	NaN	48	NaN	1	205	1	NaN	NaN
top	NaN	NOC - 6720HI to R4 Silica	NaN	2020-06-04 17:00:00	2020-06-18 21:00:00	2020-06-21 19:00:00	NaN	NaN
freq	NaN	410	NaN	19680	96	19680	NaN	NaN
first	NaN	NaN	NaN	2020-06-04 17:00:00	2020-06-04 19:00:00	2020-06-21 19:00:00	NaN	NaN
last	NaN	NaN	NaN	2020-06-04 17:00:00	2020-06-21 19:00:00	2020-06-21 19:00:00	NaN	NaN
mean	25.250000	NaN	6.211654	NaN	NaN	NaN	7200.0	2.816634
std	17.429466	NaN	8.264031	NaN	NaN	NaN	0.0	2.132946
min	1.000000	NaN	0.027263	NaN	NaN	NaN	7200.0	0.250300
25%	11.000000	NaN	1.000000	NaN	NaN	NaN	7200.0	1.276120
50%	24.000000	NaN	1.000000	NaN	NaN	NaN	7200.0	2.031490
75%	31.000000	NaN	10.000000	NaN	NaN	NaN	7200.0	3.537790
max	62.000000	NaN	40.000000	NaN	NaN	NaN	7200.0	27.051760

In [29]:
```
df.dtypes
```

Out[29]:
```
ID EQUIPO                          int64
PUNTO MEDICION                    object
CAPACIDAD MAXIMA [GBS]           float64
FECHA INICIO MEDICION     datetime64[ns]
FECHA HORA                datetime64[ns]
FECHA FIN MEDICION        datetime64[ns]
PASO                             float64
LATENCIA [MS]                    float64
% PACK LOSS                      float64
INBOUND [BITS]                   float64
OUTBOUND [BITS]                  float64
MEDIDA                            object
dtype: object
```

Ahora podemos ver que las columnas mencionadas son de tipo fecha

In [30]:
```
df_abasto = df[df['PUNTO MEDICION'] == "ABA - Abasto Cliente"]
```

In [31]:
```
 df_abasto['FECHA HORA'].max() - df_abasto['FECHA HORA'].min()
```

Out[31]:
```
Timedelta('17 days 00:00:00')
```

1. **Etiquetas de variables/columnas: no usar caracteres especiales** Chequear que no haya caracteres fuera de a-Z, 0-9 y _ en los nombres de columnas del Dataframe.

In [32]:
```
columns_orig = df.columns
columns_orig
```

Out[32]:
```
Index(['ID EQUIPO', 'PUNTO MEDICION', 'CAPACIDAD MAXIMA [GBS]',
       'FECHA INICIO MEDICION', 'FECHA HORA', 'FECHA FIN MEDICION', 'PASO',
       'LATENCIA [MS]', '% PACK LOSS', 'INBOUND [BITS]', 'OUTBOUND [BITS]',
       'MEDIDA'],
      dtype='object')
```

In [33]:
```
columns_orig.str.match(r'^([\w\d_]+)$')
```

Out[33]:
```
array([False, False, False, False, False, False,  True, False, False,
       False, False,  True])
```

Vemos que muchos no cumplen con la condicion de solo incluir letras, numeros y guion bajo.

In [34]:
```
df.columns = ['ID_EQUIPO', 'PUNTO_MEDICION', 'CAPACIDAD_MAXIMA', 'FECHA_INIC
IO_MEDICION', 'FECHA_HORA', 'FECHA_FIN_MEDICION', 'PASO', 'LATENCIA', 'PACK_
LOSS',
              'INBOUND_BITS', 'OUTBOUND_BITS', 'MEDIDA']
```

In [35]:
```
df.columns
```

Out[35]:
```
Index(['ID_EQUIPO', 'PUNTO_MEDICION', 'CAPACIDAD_MAXIMA',
       'FECHA_INICIO_MEDICION', 'FECHA_HORA', 'FECHA_FIN_MEDICION', 'PASO',
       'LATENCIA', 'PACK_LOSS', 'INBOUND_BITS', 'OUTBOUND_BITS', 'MEDIDA'],
      dtype='object')
```

In [36]:
```
df.columns.str.match(r'^([\w_\d]+)$')
```

Out[36]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True])
```

1. Agregar nuevas caracteristicas

Agregar al Dataframe dos nuevas columnas INBOUND y OUTBOUND que seran las columnas INBOUND_BITS y OUTBOUND_BITS llevadas a la unidad especificada en la columna MEDIDA.

In [37]:
```
set(df['MEDIDA'])
```

Out[37]:
```
{'GB', 'MB'}
```

In [38]:
```
df.loc[:,('INBOUND')]=df['INBOUND_BITS'] * 8 * 1024
df.loc[:,('OUTBOUND')]=df['OUTBOUND_BITS'] * 8 * 1024
```

Primero convierto de Bits a Bytes, luego a MBytes, y finalmente multiplico por 1024 los que tengan GB en el campo MEDIDA

In [39]:
```
df.loc[df['MEDIDA']=='GB', 'INBOUND'] = df.loc[df['MEDIDA']=='GB', 'INBOUND
'] * 1024
df.loc[df['MEDIDA']=='GB', 'OUTBOUND'] = df.loc[df['MEDIDA']=='GB', 'OUTBOUN
D'] * 1024
```

4. Tratar valores faltantes Veamos cuantos valores nulos tenemos:

In [40]:
```
df_missing_values_count = df.isna().sum()
[df_missing_values_count > 0]
```

Out[40]:
```
[ID_EQUIPO                False
 PUNTO_MEDICION           False
 CAPACIDAD_MAXIMA         False
 FECHA_INICIO_MEDICION    False
 FECHA_HORA               False
 FECHA_FIN_MEDICION       False
 PASO                      True
 LATENCIA                  True
 PACK_LOSS                 True
 INBOUND_BITS              True
 OUTBOUND_BITS             True
 MEDIDA                   False
 INBOUND                   True
 OUTBOUND                  True
 dtype: bool]
```

In [41]:
```
def contar_nan(dfaux):
  df_missing_values_count = dfaux.isna().sum()
  print(df_missing_values_count[df_missing_values_count > 0])
contar_nan(df)
```

```
PASO             1175
LATENCIA         1195
PACK_LOSS        1176
INBOUND_BITS     1200
OUTBOUND_BITS    1211
INBOUND          1200
OUTBOUND         1211
dtype: int64
```

In [42]:
```
porcentaje = 100 - len(df.dropna(subset=['LATENCIA']))/len(df) * 100
print(f"Porcentaje de filas con valores nulos en el campo LATENCIA frente al total: %{porcentaje:.2}")
```

```
Porcentaje de filas con valores nulos en el campo LATENCIA frente al total: %6.1
```

Vemos que todas las columnas con valores faltantes carecen de aproximadamente la misma cantidad de valores, que representan un 6% del total. Este porcentaje puede acumularse entre los faltantes de las distintas columnas, entonces evaluamos que no es conveniente eliminar dichas entradas, ya que son considerables en realcion al tamano del dataset.

Inputacion usando Media y Moda

A continuacion enumeramos las tres maneras de imputar valores NaN

In [43]:
```
df_1 = df.copy()
df_1["LATENCIA"].fillna(df_1["LATENCIA"].mean(), inplace = True) # Inputacion con media
contar_nan(df_1)
```

```
PASO             1175
PACK_LOSS        1176
INBOUND_BITS     1200
OUTBOUND_BITS    1211
INBOUND          1200
OUTBOUND         1211
dtype: int64
```

In [44]:
```
df_2 = df.copy()
df_2["LATENCIA"].fillna(df_2["LATENCIA"].mode()[0], inplace = True) # Inputacion con moda
contar_nan(df_2)
```

```
PASO             1175
PACK_LOSS        1176
INBOUND_BITS     1200
OUTBOUND_BITS    1211
INBOUND          1200
OUTBOUND         1211
dtype: int64
```

In [45]:
```
df_3 = df.copy()
df_3["LATENCIA"].fillna(df_3["LATENCIA"].median(), inplace = True) # Inputacion con mediana
contar_nan(df_3)
```

```
PASO             1175
PACK_LOSS        1176
INBOUND_BITS     1200
OUTBOUND_BITS    1211
INBOUND          1200
OUTBOUND         1211
dtype: int64
```

De las tres, perefimos usar la mediana, ya que no se ve afectada por valores muy extremos.

In [46]:
```
df_missing_values_count = df.isna().sum()
columnas_corruptas = df_missing_values_count[df_missing_values_count > 0].keys()
columnas_corruptas
```

Out[46]:
```
Index(['PASO', 'LATENCIA', 'PACK_LOSS', 'INBOUND_BITS', 'OUTBOUND_BITS',
       'INBOUND', 'OUTBOUND'],
      dtype='object')
```

In [47]:
```
df_4 = df.copy()
for columna in columnas_corruptas:
  df_4[columna].fillna(df_4[columna].median(), inplace = True)
contar_nan(df_4)
```

```
Series([], dtype: int64)
```

In [48]:
```
df.size == df_4.size # vemos que el tamano del dataset no se altero
```

Out[48]:
```
True
```

1. Codificar variables

   Las variables categóricas deben ser etiquetadas como variables numéricas, no como cadenas.

Codificar la variable PUNTO MEDICION del Dataframe.

In [49]:
```
from sklearn import preprocessing
label_encoding = preprocessing.LabelEncoder()
label_encoding.fit(df.PUNTO_MEDICION)
label_encoding.classes_
```

Out[49]:
```
array(['ABA - Abasto Cliente', 'ABA - Temple', 'BAZ - Carlos Paz',
       'BAZ - Yocsina', 'Carlos Paz - Cosquin', 'Carlos Paz - La Falda',
       'EDC - Capitalinas', 'EDC - Coral State', 'EDC - ET Oeste',
       'EDC - MOP', 'EDC - NOR', 'EDC - RDB', 'EDC - Telecomunicacioes',
       'EDC - Transporte', 'JM - Totoral Nueva', 'JM - Totoral Vieja',
       'NOC - 6720HI to BAZ', 'NOC - 6720HI to EDC',
       'NOC - 6720HI to ETC', 'NOC - 6720HI to N20-1',
       'NOC - 6720HI to R4 Silica', 'NOC - 6720HI to RPrivado',
       'NOC - ACHALA - Servicios', 'NOC - ACHALA - Solo Dolores',
       'NOC - Almacenes', 'NOC - ET Sur', 'NOC - Interfabricas',
       'NOC - Pilar', 'NOC - S9306 to SS6720HI', 'NOC - SW Clientes 1',
       'NOC - SW Clientes 2', 'NOC - Switch Servers', 'NOC - UTN',
       'RDB - ET Don Bosco - San Roque', 'RDB - ET La Calera',
       'RDB - Escuela de Capacitacion', 'RDB - GZU', 'RDB - JM',
       'RDB - PEA', 'RDB - RIO', 'SF - Freyre', 'SF - La Francia',
       'SF - Las Varillas', 'SF - SF Adm', 'SF - SF Cliente',
       'Yocsina - Alta Gracia', 'Yocsina - Carlos Paz',
       'Yocsina - Mogote'], dtype=object)
```

In [50]:
```
df.sample(10,random_state = 99)
```

Out[50]:

	ID_EQUIPO	PUNTO_MEDICION	CAPACIDAD_MAXIMA	FECHA_INICIO_MEDICION	FECHA_HORA	FE
14106	62	NOC - 6720HI to R4 Silica	1.000000	2020-06-04 17:00:00	2020-06-18 15:00:00	
3318	62	NOC - 6720HI to BAZ	20.000000	2020-06-04 17:00:00	2020-06-07 23:00:00	
11527	24	EDC - ET Oeste	1.000000	2020-06-04 17:00:00	2020-06-08 17:00:00	
859	30	Carlos Paz - Cosquin	1.000000	2020-06-04 17:00:00	2020-06-08 01:00:00	
4574	41	NOC - ACHALA - Servicios	0.027263	2020-06-04 17:00:00	2020-06-10 03:00:00	
13077	28	JM - Totoral Vieja	10.000000	2020-06-04 17:00:00	2020-06-18 07:00:00	
12956	28	JM - Totoral Vieja	10.000000	2020-06-04 17:00:00	2020-06-08 05:00:00	
7635	31	RDB - JM	10.000000	2020-06-04 17:00:00	2020-06-08 23:00:00	
8536	11	SF - La Francia	10.000000	2020-06-04 17:00:00	2020-06-15 17:00:00	
16257	4	NOC - Switch Servers	1.000000	2020-06-04 17:00:00	2020-06-09 23:00:00	

In [51]:
```
df.PUNTO_MEDICION = label_encoding.transform(df.PUNTO_MEDICION)
```

In [52]:
```
df.sample(10,random_state = 99)
```

Out[52]:

	ID_EQUIPO	PUNTO_MEDICION	CAPACIDAD_MAXIMA	FECHA_INICIO_MEDICION	FECHA_HORA	FE
14106	62	20	1.000000	2020-06-04 17:00:00	2020-06-18 15:00:00	
3318	62	16	20.000000	2020-06-04 17:00:00	2020-06-07 23:00:00	
11527	24	8	1.000000	2020-06-04 17:00:00	2020-06-08 17:00:00	
859	30	4	1.000000	2020-06-04 17:00:00	2020-06-08 01:00:00	
4574	41	22	0.027263	2020-06-04 17:00:00	2020-06-10 03:00:00	
13077	28	15	10.000000	2020-06-04 17:00:00	2020-06-18 07:00:00	
12956	28	15	10.000000	2020-06-04 17:00:00	2020-06-08 05:00:00	
7635	31	37	10.000000	2020-06-04 17:00:00	2020-06-08 23:00:00	
8536	11	41	10.000000	2020-06-04 17:00:00	2020-06-15 17:00:00	
16257	4	31	1.000000	2020-06-04 17:00:00	2020-06-09 23:00:00	

Vemos por ejemplo en la fila 13077 y 12956, que correspondian a JM - Totoral Vieja, recibieron el mismo encoding, por lo que verificamos una codificacion exitosa.