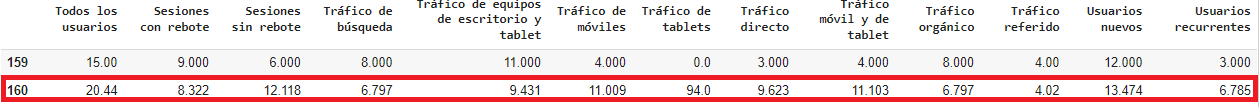
# Avances de la consulta anterior

# Pregunta enviada al email

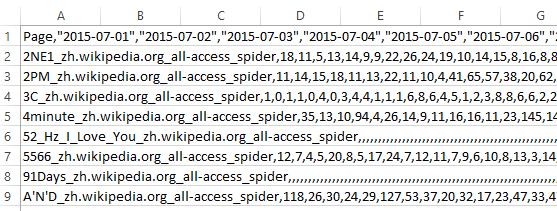
# Respecto a avanzar con la limpieza de los datos de la tesis, pude sacar las filas y columnas que tienen solo valores ceros del dataset, ¿ahora como continuo? Sin embargo todavía hay algunas columnas raras como "Tráfico de tablets" la cual tiende a tener ceros y la columna "Usuarios Recurrentes" que tiene ceros los primeros 5 valores, también está la última fila la cual es la única que tiene valores decimales.

# Usted había mencionado algo referente a un rellenado con datos estadísticos de los ceros o similar, quisiera saber su opinión al respecto.

La última fila es la única que tiene valores en decimal

# Relevamiento del pre-procesado de datos de wikipedia

**Tiene los datos así**



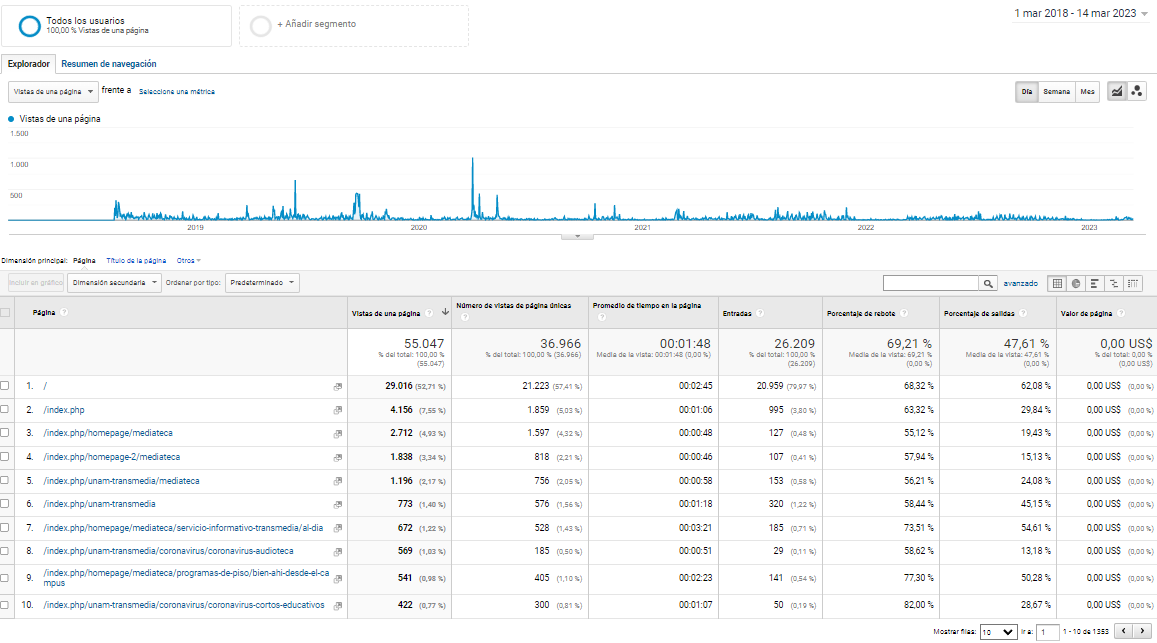
Tiene por cada ruta los datos /”ruta” tipo de acceso (spyder,mobile,desktop) y site, country

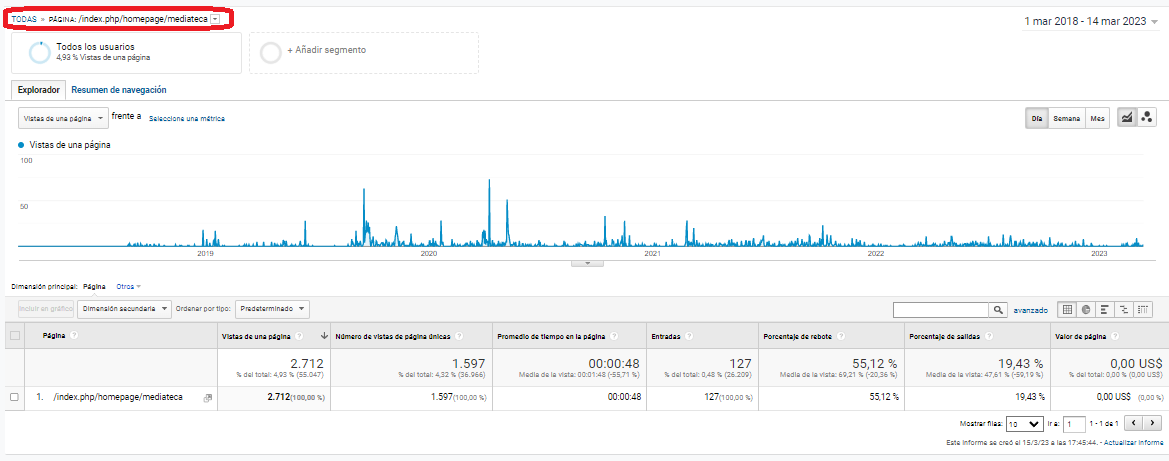
Básicamente reemplaza valores  
nan ,,, en el CSV por 0 para cálculos

Luego vuelve a poner los nan devuelta al final

# **Mis datos son así en General de todas las páginas**

De todos los usuarios,(12 distintos tipos de usuarios), pero **NO** por ruta /”ruta”



Pero también se pueden descargar todos los usuarios por /”ruta” + 12 categorías

Descubrí que si se puede descargar, debería descargarlas una por una?

Si, No

Volviendo al pre-procesado de datos en 2017

**Después básicamente 4 columnas 1 valor para cada página  
hacen por cada url de wikipedia**

Finds agent types (spider, desktop, mobile, all) for each unique url, i.e. groups pages by agents

**Después calcula auto correlación(lo tengo que hacer?)**

Calcula autocorrelación para  
# Yearly(annual) autocorrelation  
batch de 365  
# Quarterly autocorrelation  
int(round(365.25/4)  
"Percent of undefined autocorr = yearly:%.3f, quarterly:%.3f" % (year\_unknown\_pct, quarter\_unknown\_pct)

**Luego calcula las características de las páginas site, country, agent Las cuales son strings**

Calculates page features (site, country, agent, etc) from urls

pd.DataFrame({  
 **'agent'**: agents,  
 **'site'**: sites,  
 **'country'**: countries,  
 **'term'**: terms,  
 **'marker'**: markers,  
 **'page'**: source  
})

Dropea [**'term'**, **'marker'**]

Aplica one-hot encoding a las características de la página y normaliza  
ya que sus caracteristicas site, country, agent, etc son string

**Calcula Popularidad por página por mediana**

page\_popularity = df.median(axis=1)

Normalizacion:  
page\_popularity = (page\_popularity - page\_popularity.mean()) /page\_popularity.std()

Normaliza tambien las autocorrelaciones yearly y quaterly

Agrega los nans devuelta al dataframe

Queda al final

*# Put NaNs back*df[nans] = np.NaN

*# Assemble final output*tensors = dict(  
 hits=df,#dataframe original  
 lagged\_ix=lagged\_ix,#revisar  
 page\_map=page\_map,# *agent types (spider, desktop, mobile, all) for each unique url*  
 page\_ix=df.index.values,  
 pf\_agent=encoded\_page\_features[**'agent'**],#agente del one hot encoding  
 pf\_country=encoded\_page\_features[**'country'**], #pais del one hot encoding  
 pf\_site=encoded\_page\_features[**'site'**], #sitio del one hot encoding  
 page\_popularity=page\_popularity, #mediana  
 year\_autocorr=year\_autocorr, #autocorrelacion anual  
 quarter\_autocorr=quarter\_autocorr, #autocorrelacion 1 cuarto del año  
 dow=dow,#nose#revisar  
)

plain = dict(  
 features\_days=len(features\_days),  
 data\_days=len(df.columns),  
 n\_pages=len(df),  
 data\_start=data\_start,  
 data\_end=data\_end,  
 features\_end=features\_end  
  
)

Sobre lag\_indexes: “My best public score was  
achieved using only lagged datapoints, without attention.

<https://github.com/Arturus/kaggle-web-traffic/blob/master/images/lagged_data.png?raw=true>

Additional important benefit of lagged datapoints: model can use much shorter  
encoder without fear of losing information from the past, because this  
information now explicitly contained in features”

Hiperparámetros:

Se menciona en alguna parte el uso de “SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization”

Lista de hiperparametros en hparams.py

*# Manually selected params*

|  |  |  |
| --- | --- | --- |
| Nombre | Valor por defecto | Descripcion |
| batch\_size | 256 | Tamaño del lote |
| train\_window | 283 |  |
| train\_skip\_first | 0 |  |
| rnn\_depth | 267 | Unidades de la red neuronal GRU por layer |
| use\_attn | False |  |
| attention\_depth | 64 |  |
| attention\_heads | 1 |  |
| encoder\_readout\_dropout | 0.4768781146510798 |  |
| encoder\_rnn\_layers | 1 | Numero Capa de encoder de red neuronal |
| decoder\_rnn\_layers | 1 | Numero Capa de decoder de red neuronal |
| decoder\_input\_dropout | [1.0, 1.0, 1.0] |  |
| decoder\_output\_dropout | [0.975, 1.0, 1.0] |  |
| decoder\_state\_dropout | [0.99, 0.995, 0.995] |  |
| decoder\_variational\_dropout | [**False**, **False**, **False**] |  |
| decoder\_candidate\_l2 | 0.0 |  |
| decoder\_gates\_l2 | 0.0 |  |
| fingerprint\_fc\_dropout | 0.8232342370695286 |  |
| gate\_dropout | 0.9967589439360334 |  |
| gate\_activation | **'none'** |  |
| encoder\_dropout | 0.030490422531402273 |  |
| encoder\_stability\_loss | 0.0 | No aplicar esta funcion |
| encoder\_activation\_loss | 1e-06 |  |
| decoder\_stability\_loss | 0.0 |  |
| decoder\_activation\_loss | 5e-06 |  |
|  |  |  |

# Adaptación a Keras

Fuente: <https://www.analyticsvidhya.com/blog/2020/10/multivariate-multi-step-time-series-forecasting-using-stacked-lstm-sequence-to-sequence-autoencoder-in-tensorflow-2-0-keras/>

En el artículo de arriba se tiene el código para Multivariate Multistep con LSTM y predice todas las columnas(features), nó sola 1, el cual usa la arquitectura encoder-decoder con Keras y un tensorflow más actual.

En mi caso reemplacé:

LSTM con GRU (teóricamente solo cambiar el nombre de uno por el otro funciona).

De Multi-Step a futuro para 1 solo paso a futuro

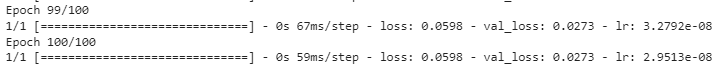
Usé mis datos CSV a Dataframe

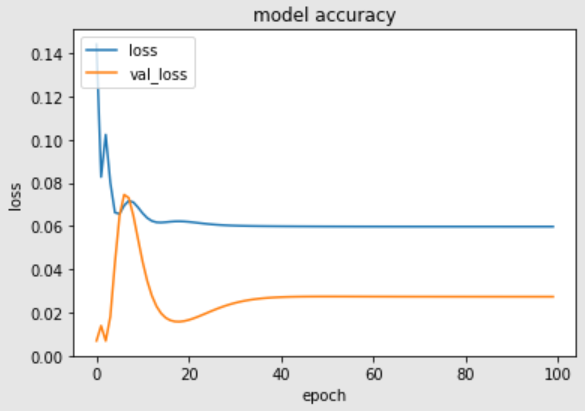
Configuré para el tamaño de mis datos, mis características (features), total de datos, etc

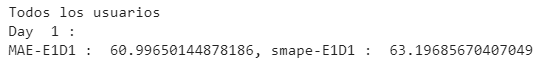
Modifiqué el código para pronosticar en vez de las 13 features 1 sola feature, en mi caso el de “Todos los Usuarios”

Para pasar de pronosticar todas las features a 1 sola use este video: <https://www.youtube.com/watch?v=tepxdcepTbY&t=867s>

Algunas capturas de los Resultados:







Preguntas:

Dado que la arquitectura es un encoder-decoder el mismo que se usó en el paper wikipedia 2018 ¿Se puede utilizar este modelo?

¿Es confiable la página de donde se sacó el código (<https://www.analyticsvidhya.com>)?

En caso de utilizar este modelo, ¿cómo sé que el código que modifiqué está bien?¿tiene alguien que verifique el código que hacen los alumnos de tesis?

En caso de utilizar este modelo, ¿cómo mejoro los hiperparámetros?, dado que no es solo copiar los mismos de wikipedia