

CODERHOUSE DATA SCIENCE

PROYECTO FINAL

Aplicando Ciencia de Datos en el Fútbol

Equipo

| Karen Justo | Juan Lacasa | Marcos Unamuno |

Tutor

Franco

Índice

Índice	1
Tema de Análisis	1
Objetivos	2
Stack del proyecto	2
Visualización	2
Data Acquisition	3
nwsIR	3
ASA	3
Data Wrangling	5
Conectando la data de nwsIR con la de “players” de ASA	5
1. Obteniendo la fecha de nwsIR	6
2. Eliminar futbolistas sin minutos de juego de nwsIR	6
3. Limpiar los nombres de nwsIR	6
4. Limpiar la posición en nwsIR	6
5. Filtrar futbolistas que no participaron en las temporadas 2016-2019 de la data de ASA	7
Finalmente: Conectando la data por medio del nombre	7
Conectando la data de nwsIR y con la de “games” de ASA	7
1. Limpiando la fecha de “games” de ASA	7
2. Merge entre “xgoals” y “games” de ASA	8
3. Merge entre el dataframe resultante anterior y el de nwsIR	8
Análisis	8
Algoritmo de Clasificación	9
Árbol de Decisión	9
SVM	9
Fuentes	9
Datasets	9
Información General	9
Gráficos con Plotly	11

Tema de Análisis

La data que se va a analizar corresponde a la NWSL (National Women's Soccer League), que es la máxima división del fútbol profesional femenina en los Estados Unidos.

El análisis se enfoca en revisar si es posible clasificar a cada persona en una posición definida sabiendo cuáles fueron sus estadísticas para un partido. Algunas suposiciones antes de empezar el análisis son:

- Las personas en defensa central y portería tendrán “mejores” estadísticas en lo que se refiere al % de pases completados en comparación a otras posiciones en el campo.
- En ataque, se tendrán números más altos de tiros a gol, intentos de asistencia, xG, xA, entre otros valores que corresponden más a instancias de ataque en el juego.

Si se lograra identificar la posición en base a algunos parámetros, un siguiente paso sería revisar si su posición general concuerda con su clasificación, y ver si no se está "empleado" mal a alguien en el campo.

Objetivos

Esta primera instancia de este análisis se enfoca más en un público que tenga algún conocimiento general sobre fútbol y las métricas que lo caracterizan.

No es la idea que sea utilizado por un equipo técnico o personas dedicadas profesionalmente al deporte, al menos no en esta primera iteración. En su lugar, la idea es que lo utilicen fanáticos o personas que quieran indagar un poco más sobre el deporte.

La pregunta principal a contestar es:

¿Se puede clasificar a los futbolistas según sus estadísticas de forma clara y precisa?

De la cual surgen las siguientes preguntas:

¿Qué datos permiten esta clasificación?

¿Hay datos que se puedan usar de forma global para todas las posiciones?

¿Si se tiene la posibilidad de clasificar, se tiene también la posibilidad de hacer comparaciones para evaluar el desempeño con estas características?

¿Se tiene que considerar a cualquiera en el análisis, o hay casos que rompen el algoritmo, como puede ser alguien que juega menos de X minutos en un partido?

Estas últimas preguntas son un extra de lo que se quiere lograr, por lo que esta iteración puede o no dar respuesta a ellas.

Stack del proyecto

Para el manejo de los datos:

- Python

- Jupyter

Visualización

Seleccionando la librería:

> [Matplotlib vs. seaborn vs. Plotly vs. MATLAB vs. ggplot2 vs. pandas - Ritza Articles](#)

“Consider Plotly if you don't mind some closed-source packages or if you want to display your interactive data visualizations on the web”

Data Acquisition

El trabajo se encuentra disponible en: [Final Project - Notebook](#) En curso ▾

La presentación se puede encontrar en: [Data Science - NWSL](#) En curso ▾

nwsIR

Parte de la data principal se obtuvo del repositorio nwsIR, mantenido por: Arielle Dror y Sophia Tannir.

nwsIR es un repositorio con información de partidos desde el 2016 al 2019. Se escoge esta fuente porque ofrece datos de un montón de acciones para cada atleta en cada partido.

Se pueden encontrar datos que también se encuentran en la otra fuente (ASA), pero incluye otra información que no se encuentra (de forma gratuita) en otras partes. Esto es especialmente útil con respecto a acciones defensivas, que no se suelen conseguir fácilmente (barridas, duelos, despejes, entre otras).

Se usa sólo la tabla de Advanced Stats porque la otra información disponible está de forma más limpia y ordenada en ASA (equipos, juegos, jugadores).

Esta data se obtiene por medio del siguiente código:

```
url = 'https://raw.githubusercontent.com/adror1/nwsIR/master/data-raw/adv_player_stats.csv'
df_adv_stats = pd.read_csv(url)
```

La información está en github, así que es simplemente traer la data de ahí. Para más detalles sobre toda la información disponible, se recomienda revisar el [repositorio](#).

ASA

Otra parte se obtuvo de la API de ASA (American Soccer Analysis) llamada “[itscalledfootball](#)”.

Esta fuente es una API habilitada recientemente por el equipo de American Soccer Analysis. En esta se tienen diferentes endpoints para obtener información sobre las ligas americanas NWSL (liga femenina) y la MLS (liga masculina).

Se utiliza la información de la NWSL para complementar la data que se obtuvo de la nwsIR, sumando información más estadística como lo es:

- xG (probabilidad de gol de un disparo)
- xA (probabilidad de asistencia de un pase), entre otros.

Esta data está bastante limpia y completa, por lo que la data general sobre equipos, jugadores y partidos se utiliza como base del proyecto.

Para poder trabajar con esta API, se tienen los siguientes pasos (disponibles en la documentación):

```
!pip install itscalledsoccer
from itscalledsoccer.client import AmericanSoccerAnalysis
asa_client = AmericanSoccerAnalysis()
```

Luego, se debe llamar a los endpoints de la siguiente form:

```
asa_client.get_player_xgoals
```

Los endpoints que se utilizaron para este trabajo fueron:

get_player_xgoals

Trae la data de los "expected values" (valores esperados, como xgoals y xassists) para cada futbolista, por cada juego en el que jugaron (lo cual se logra pasando true para el parámetro split_by_games).

Se filtró la data desde el principio con los parámetros leagues con el valor "nwsI" (porque es la liga a analizar) y season_name con el valor "2016, 2017, 2018, 2019", porque son las temporadas disponibles en nwsIR.

get_games

Trae la data básica de cada juego. Se filtró desde el llamado de la API usando los mismos parámetros mencionados antes, para obtener los datos de la NWSL para las temporadas 2016-2019.

🕒 En algún punto se pensó en filtrar los partidos "excepcionales" (los de campeonatos, que pueden durar más de 90 minutos e ir a penales) usando el parámetro stage porque son outliers si se comparan con otros partidos. Pero luego se notó que primero sería mejor hacer el merge entre dataframes primero y luego filtrar esos partidos usando la columna knockout_game. Esto para facilitar el análisis de si el merge se hizo correctamente o no.

get_players

Trae la información sobre cada futbolista en la NWSL. Acá sólo se filtra con leagues también, y luego se harán ciertas transformaciones y cambios para limitar la lista y obtener sólo a quienes jugaron entre 2016-2019.

Otros endpoints que no se utilizaron al final, pero podrían ser útiles a futuro:

get_teams

Trae la data de los equipos en la NWSL, y se filtra con el parámetro leagues para conseguir esto.

get_team_xgoals

Trae la información relacionada a los "expected values" para cada equipo, separado por partido (según se pide con el parámetro `split_by_games`).

Como el resto de los endpoints que tienen la opción, se filtra con `leagues` y `season_name`.

Data Wrangling

Antes de entrar en detalle sobre qué se hizo, primero se tiene que explicar a dónde se quiere llegar: se quiere poder juntar ambas fuentes de datos para utilizar información de ambas en la clasificación y el análisis posterior.

Ambas fuentes tienen campos de "id", pero luego de revisar con una jugadora, se puede ver que no hay relación entre estos ids.

Para Michelle Betos, se tiene el id

- en nwsIR `9ifjrfggioh9cmphk85gykzx1`
- en ASA `NWMWVjJjQl`

Lo mismo se ve en cuanto al id para los partidos.

Como se tienen dos fuentes de datos que no comparten ninguna key entre sí, la fusión entre ambas fuentes se debe hacer por medio de nombres, fechas, entre otros campos.

Conectando la data de nwsIR con la de "players" de ASA

La primera de las transformaciones a hacer es revisar los nombres que se tienen en ambas fuentes y lograr conectarlas con esta información.

Además, el objetivo final es trabajar sobre las posiciones (defensa, mediocampo, etc). Es por ello que también se hace una limpieza sobre este campo en la info de nwsIR.

También, para poder matchear los partidos de una fuente con otra, se utilizan las fechas. Un problema que surge acá es que las fechas en nwsIR están en ET y las de ASA en GMT, por lo que se tuvo que trabajar sobre eso.

Al principio, se tiene que:

🔍 hay más futbolistas en la fuente de ASA que en la de nwsIR.

💡 La diferencia es probablemente causada porque desde nwsIR se tiene data de 2016 a 2019 nada más. Y como se mencionó antes, la de ASA incluye datos de futbolistas que no participaron en las temporadas de 2016-2019.

Parte de la limpieza se enfoca en llegar a corregir esto, pero hay otros pasos:

En la data de nwsIR:

1. Se obtiene la fecha a partir de la columna donde está el id, porque es el único lugar donde se menciona.
2. Borrar futbolistas que realmente no jugaron en el partido, es decir, cuyos minutos de juego son cero.

3. Limpiar los nombres, agregando una columna llamada `player_name` que sea igual a `first_name + ' ' + last_name`.
4. Completar la data sobre de la posición para los registros que no la tienen.

En la data de players desde ASA:

1. Borrar futbolistas que no participaron en las temporadas de 2016-2019, para reducir el total de registros y pasar de 654 a un número más cercano a 440 (como en `nwsIR`).

1. Obteniendo la fecha de `nwsIR`

Para la limpieza de la fecha en la información de `nwsIR`, se utiliza primero el siguiente código para poder usar una librería que permite extraer la fecha de un texto.

```
!pip install datefinder
import datefinder
datefinder.find_dates
```

Esa fecha se guarda en la columna `"played_on"`. Un detalle sobre esta fecha: está en la hora ET, que es la franja horaria que normalmente se utiliza en la liga. Este detalle es importante para una de las transformaciones futuras.

2. Eliminar futbolistas sin minutos de juego de `nwsIR`

Lo único que se hace en este punto es guardar una nueva versión de la información, eliminando los registros que tienen el campo `mins_played <= 0`.

3. Limpiar los nombres de `nwsIR`

El primer paso es reemplazar los valores que tienen `"first_name"` nulo, para tener un string vacío en su lugar (`" "`).

Luego de eso, se genera una columna llamada `"player_name"` uniendo el primer nombre junto con el segundo nombre, si es que tiene primer nombre. En caso de que no lo tenga, se utiliza el último nombre.

4. Limpiar la posición en `nwsIR`

Como el análisis va a utilizar la posición como target, se tiene que limpiar un poco.

Primero, se corrigen las que tienen `"substitute"` en el campo de `"position"`. Cuando tienen ese valor en la mayoría de los casos se tiene la posición `"real"` en la columna de `"sub_position"`. Esto es una suposición, y luego de revisarlo se ve que hay casos donde se tiene `sub_position = '0'`.

Por ende, el reemplazo se hace guardando en `"position"` el valor de `"the sub_position"` sólo para los casos donde `"sub_position"` es diferente de `'0'`. Como dato adicional, cabe aclarar que acá no se pierde data, porque a fin de cuentas el dato de si alguien entró como reemplazo se puede encontrar en la columna `"total_sub_on"`.

Luego de hacer esto, aún se tienen unos pocos casos donde no se tiene valor para la `"position"`. Estos datos no se van a borrar porque la persona Sí jugó en el partido.

La lógica utilizada para “corregir” estos datos, va a ser copiar el valor de la posición que se encuentre para un partido futuro para el mismo equipo del registro al que le falta esa información. Esta “lógica” se basa en que para un equipo, es muy poco lo que cambia la posición de cada persona, además de que son sólo 137 registros de más de 10 mil, por lo que no debería causar grandes cambios.

Si no hay un partido posterior del que copiar la posición, se toma el de un partido anterior, pero siempre considerando los partidos para el mismo equipo.

5. Filtrar futbolistas que no participaron en las temporadas 2016-2019 de la data de ASA

Lo único que se hace en este punto es guardar una nueva versión de la información, eliminando los registros de futbolistas que no participaron en esas temporadas.


Finalmente: Conectando la data por medio del nombre

No son muchos los casos donde no se logra conectar el dataframe de nwsIR con el de ASA, con 47 futbolistas, a pesar de los cambios que se habían hecho anteriormente. La única forma de corregir estos casos es manualmente.

La mayoría de estos casos se tienen porque en nwsIR se tienen:

- Nombres que no tienen "caracteres especiales".
- Mantiene el apellido de soltera de algunas jugadoras.
- Algunos otros casos especiales.


Los matches corregidos manualmente se encuentran en este listado:

 [Matching names manually](#)

Con esto, se puede asociar el player_id de ASA con la data de nwsIR.

Conectando la data de nwsIR y con la de “games” de ASA

El siguiente paso es poder conectar lo que tenemos de nwsIR con las estadísticas por futbolista por juego.

 Mirando los dataframes que queremos conectar, parecía que lo útil iba a ser conectar “games” con “xgoals” primero, para tener la fecha del partido y luego poder usar esa fecha para poder conectar con el dataframe de nwsIR usando el “player_id” y “played_on”

1. No hay una fecha “limpia” en games de ASA, y se va a crear usando la columna date_time_utc, para obtener sólo la fecha porque la hora no es necesaria.
2. Luego, games de ASA se mergeará con “xgoals” también de ASA, usando la columna de game_id del dataframe con los datos de “games” de ASA.

1. Limpiando la fecha de “games” de ASA

Primero, se importa una librería que será de ayuda para pasar las fechas de UTC a ET (eastern time).

```
from datetime import datetime
import pytz
UTC = pytz.utc
```



```
eastern = pytz.timezone("US/Eastern")
```

Este código se aplica sobre cada uno de los registros, transformando los valores de la columna “date_time_utc” a ET:

```
datetime.fromisoformat(str(datetime.fromisoformat(x.replace(' UTC',''))  
                        .astimezone(eastern)).split(' ')[0])
```

2. Merge entre “xgoals” y “games” de ASA

Este match se logra por medio del game_id, que al ser ambos dataframes información obtenida de ASA, sí comparten ese dato.

3. Merge entre el dataframe resultante anterior y el de nwsIR

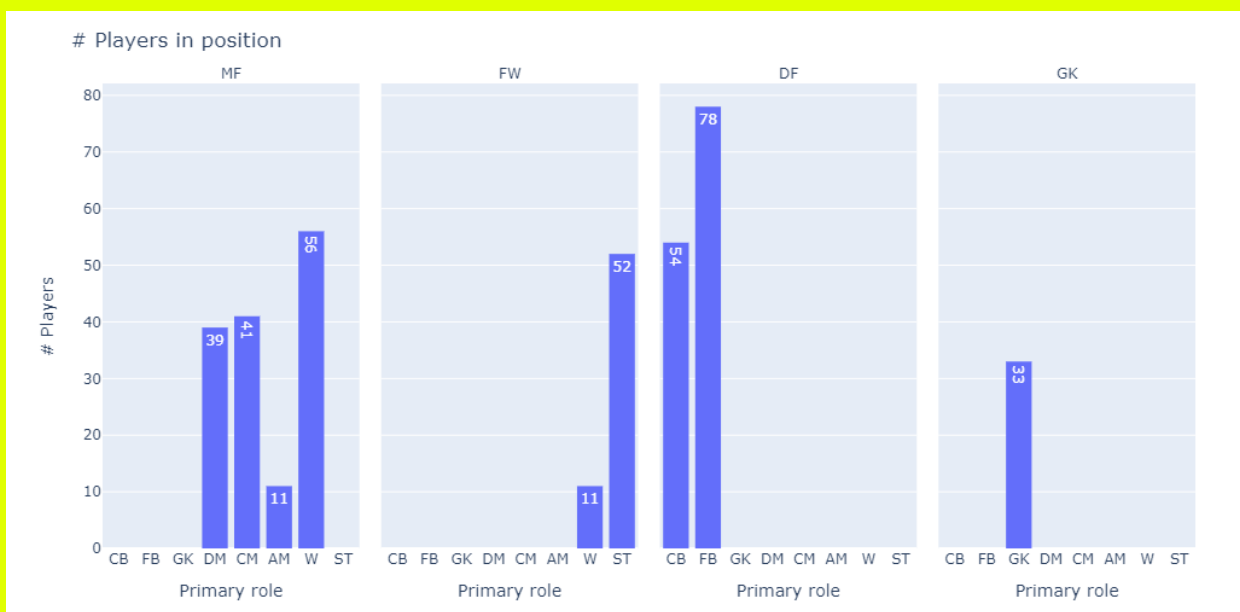
El merge entre ambos se hace por medio de las columnas de fechas y de player_id disponible en ambos dataframes. Al hacer el merge (left merge), se obtiene que hay cuatro registros para los que no se encontró un valor en el dataframe de “xgoals + games”.

Probablemente son registros para los que no se tiene data porque los minutos jugados fueron mínimos, y al final no tenía sentido que ASA tuviese datos para esos casos. Después de una exploración de los datos, se confirma esta hipótesis, con futbolistas que jugaron 1 minuto, si acaso.

Por ende, se borran esos registros del dataframe final “df_final2”.

Análisis

La primera gráfica que se realiza es una para ver la cantidad de registros que se tienen para cada uno de los valores “target” en la tabla de “players” de ASA, para entender si se tiene una base de datos balanceada.



La mayor cantidad de datos se acumula para posiciones en el medio campo, aunque sólo 11 son “Attacking Midfielders”. Para el ataque (forwards) también se tiene que “Wingers” tiene poca

representación, probablemente porque la mayoría de esos registros están marcados como mediocampo (MF).

Algoritmo de Clasificación

Se abordó este problema de clasificación mediante modelos clásicos.

Resultados preliminares

	train accuracy score	test accuracy score	f1 score
Árbol de decisión	0.6579	0.5984	0.6069
Support Vector Machine	0.7014	0.7014	0.7023
Random Forest	0.9977	0.6653	0.6653
xGBoost	0.8442	0.7203	0.7244
Ensamble	0.8572	0.64096	0.6398

Mejora de resultados

Se intentó mejorar los resultados de los primeros tres modelos variando los hiperparámetros con la ayuda de GridSearchCV y RandomizedSearchCV, pero las mejoras no alcanzan al resultado obtenido con xGBoost.

	train GridSearch	test GridSearch	train RandomizedS earch	test RandomizedS earch
Árbol de decisión	0.6713	0.6008	0.6063	0.5965
Support Vector Machine	0.7220	0.6936	0.6858	0.6801
Random Forest	0.6571	0.6531	0.6594	0.6417

Modelo seleccionado

Por el método de selección ilustrado más arriba, modelo que se terminó seleccionando es el xgBoost.

Futuras líneas

Con ayuda del heatmap, se podría mejorar el modelo eliminando columnas que están altamente correlacionados.

Haciendo un análisis con la varianza, descartar algunas columnas del dataset.

Conclusiones

Después del análisis desarrollado se pudo determinar con un accuracy score de 0.7203 y F1 score de 0.7244 que el mejor modelo para predecir la posición del jugador en base a datos de performance es el xgBoost. 2 puntos por encima de los otros modelos probados.

Estas métricas no están aún a la altura de lo que se esperaría de un modelo productivo. Pero mediante mejoras en el modelo se podría intentar incrementar este score.

Fuentes

Datasets

> American Soccer Analysis

- API: [American-Soccer-Analysis/itscaldedsoccer: R and Python packages that wrap the ASA API \(github.com\)](https://github.com/itscaldedsoccer/itscaldedsoccer)
- Definition of the API: [itscaldedsoccer/client.py at main · American-Soccer-Analysis/itscaldedsoccer \(github.com\)](https://github.com/itscaldedsoccer/client.py)

> nwsIR: [adorr1/nwsIR: Datasets and Analytics for the National Women's Soccer League \(NWSL\) \(github.com\)](https://github.com/adorr1/nwsIR)

> Algunas anotaciones sobre la información en adv_stats de nwsIR, hecho por mí para el curso de Data Analysis en Coderhouse: [Entrega del Proyecto Final - Documentos de Google](#)

Información General

> Clases y ejemplos:

- [24465-data-science - Google Drive](#)
- [Ejemplo 1 - Análisis Univariado de Datos \(CoderHouse\).ipynb - Colaboratory \(google.com\)](#)
- [sightes/COFFETALKS \(github.com\)](https://github.com/sightes/COFFETALKS)

Recordatorios de cómo:

> Filtrar

- [How to select rows and columns in Pandas using \[\], .loc, .iloc, .at and .iat - KDnuggets](#)
- [python - How do I select rows from a DataFrame based on column values? - Stack Overflow](#)
- [How to Select Rows by Index in a Pandas DataFrame - Statology](#)
- Filter: [python - Efficient way to apply multiple filters to pandas DataFrame or Series - Stack Overflow](#)
- Use loc: [pandas.DataFrame.loc — pandas 1.4.1 documentation \(pydata.org\)](#)
- Use np.where with multiple conditions: [python - Numpy where function multiple conditions - Stack Overflow](#)
- [Select DataFrame rows between two index values in Python Pandas \(tutorialspoint.com\)](#)

> Obtener valores únicos: [Pandas : Get unique values in columns of a Dataframe in Python – thisPointer](#)

> Iterar por índices:

- [python - How do I convert a pandas Series or index to a Numpy array? - Stack Overflow](#)
PENDING: replace .values with other alternatives.
- [Python Loop Through an Array \(w3schools.com\)](https://www.w3schools.com/python/python_arrays.asp)

> Devolver columnas específicas: [How to select multiple columns in a pandas dataframe - GeeksforGeeks](#)

- Add column: [Adding new column to existing DataFrame in Pandas - GeeksforGeeks](#)
- Sort: [Pandas Sort: Your Guide to Sorting Data in Python – Real Python](#)
- LaTeX: [Line breaks and blank spaces - Overleaf. Éditeur LaTeX en ligne](#)
- Work with groupby: [python - How to print a groupby object - Stack Overflow](#)

> Limpiar data

- [Delete Rows & Columns in DataFrames using Pandas Drop \(shanelynn.ie\)](#)
- [pandas.DataFrame.drop — pandas 1.4.1 documentation \(pydata.org\)](#)
- [How to drop rows in Pandas DataFrame by index labels? - GeeksforGeeks](#)
- [python - Drop all data in a pandas dataframe - Stack Overflow](#)

> Reemplazar valores

- [How to Replace Values in Column Based on Condition in Pandas? - GeeksforGeeks](#)
- [How to use loc and iloc for selecting data in Pandas | by B. Chen | Towards Data Science](#)

> Copiar dataframe: [python - why should I make a copy of a data frame in pandas - Stack Overflow](#)

> Lambda

- [Applying Lambda functions to Pandas Dataframe - GeeksforGeeks](#)
- [pandas.DataFrame.apply — pandas 1.4.2 documentation \(pydata.org\)](#)

> Conectar dos dataframes:

- [Merge, join, concatenate and compare — pandas 1.4.2 documentation \(pydata.org\)](#)
- [python - Difference\(s\) between merge\(\) and concat\(\) in pandas - Stack Overflow](#)

> Extraer fechas de un string

- [parsing - Best way to identify and extract dates from text Python? - Stack Overflow](#)
- La solución usada de la fuente anterior devuelve un "generator":
<https://wiki.python.org/moin/Generators>

> Cuando sufrí porque season_name tenía valores de tipo dict, str y list:

- [python - How to filter a pandas dataframe by dict column? - Stack Overflow](#)
- [python - Drop rows in a dataframe based on the data type of columns - Stack Overflow](#)
- [python - Check if type is dictionary - Stack Overflow](#)
- [How to properly apply a lambda function into a pandas data frame column - Stack Overflow](#)
- [Check if element exists in list in Python - GeeksforGeeks](#)
- [python - Lambda with nested if else is not working - Stack Overflow](#)
- [Python List count\(\) Method \(w3schools.com\)](#)
- [python - Can I check if a list contains any item from another list? - Stack Overflow](#)
- [Python any\(\) Function \(w3schools.com\)](#)

A veces aparecía este error

- [python - How to deal with SettingWithCopyWarning in Pandas - Stack Overflow](#)
- [Indexing and selecting data — pandas 1.4.1 documentation \(pydata.org\)](#)

O este error:

- [python - ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool\(\), a.item\(\), a.any\(\) or a.all\(\). df\[condition\] - Stack Overflow](#)

Gráficos con Plotly

- [Plotly express with Python](#)
- [Histograms with Python \(plotly.com\)](#)

- [Axes with Python \(plotly.com\)](#)
- [Hover text and formatting with Python \(plotly.com\)](#)
- [Plotly - Format Axis and Ticks \(tutorialspoint.com\)](#)Plotly - Format Axis and Ticks (tutorialspoint.com)

> Histogramas

[Histograms with Python \(plotly.com\)](#)

[plotly.express.histogram — 5.6.0 documentation](#)

> Subplots

- [Subplots with Python \(plotly.com\)](#)
- [python - how can i create subplots with plotly express? - Stack Overflow](#)
- [plotly.subplots: helper function for laying out multi-plot figures — 5.6.0 documentation](#)

> Información sobre ticklabels

[python - Plotly: How to set xticks for all subplots? - Stack Overflow](#)

[python - How can I edit the axes tick labels in plotly graph? - Stack Overflow](#)

[How to Create Subplots in Plotly Python - Life With Data](#)

> A futuro, un dashboard:

- [Part 2. Layout | Dash for Python Documentation | Plotly](#)

> El trabajo con las fechas:

- [datetime — Basic date and time types — Python 3.10.4 documentation](#)
- [How to convert date and time with different timezones in Python? - GeeksforGeeks](#)
- [python - How do you convert a datetime/timestamp from one timezone to another timezone? - Stack Overflow](#)
- [python - How to convert local time string to UTC? - Stack Overflow](#)
- [python - How do I convert a datetime to date? - Stack Overflow](#)
- [python - How do I remove a substring from the end of a string? - Stack Overflow](#)
- [Converting object to datetime format in python - Stack Overflow](#)
- [python - Pandas: Convert Timestamp to datetime.date - Stack Overflow](#)
- [python - How to convert datetime to date only? \(Odo13\) - Stack Overflow](#)
- [Isoformat\(\) Method Of Datetime Class In Python - GeeksforGeeks](#)
- [How to make a timezone aware datetime object in Python? - Stack Overflow](#)

Otras búsquedas:

- [pandas.concat — pandas 1.4.2 documentation \(pydata.org\)](#)
- [python - Fastest way to check if a value exists in a list - Stack Overflow](#)
- [python - pandas: merge \(join\) two data frames on multiple columns - Stack Overflow](#)
- [pandas.DataFrame.replace — pandas 1.4.2 documentation \(pydata.org\)](#)
- [Python: TypeError: unhashable type: 'list' \(net-informations.com\)](#)
- [Immutable vs. Hashable – Real Python](#)
- [How to find Length of Set in Python? \(tutorialkart.com\)](#)