Práctica 2: Limpieza y análisis de datos

Autores

Hemos realizado esta práctica:

- Ignacio Such Ballester
- Andrés Isidro Fonts Santana

1. Descripción del *dataset*

1.1 Contexto

Se pretende sacar al mercado un nuevo juego de mesa lo más existoso posible y convertirlo en un bestseller.

Para ello, hemos escogido el *dataset* Board Game Data, disponible en la plataforma Kaggle.

Este conjunto de datos se ha extraído mediante la API del portal Board Games Geek. El *dataset* se generó en enero de 2018 y contiene datos sobre los primeros 5000 juegos de mesa del *ranking* de Board Games Geek.

A través de este set de datos, podemos realizar un análisis profundo del mismo, obteniendo correlaciones, clasificaciones en incluso predicciones para averigurar cómo diseñar nuestro juego de mesa.

Además, se podrá proceder a crear modelos de regresión que permitan predecir si un juego será un bestseller o no en función de sus características y contrastes de hipótesis que ayuden a identificar propiedades interesantes en las muestras.

1.2 Descripción de los atributos

Cada uno de los 5000 registros con que cuenta al *dataset* viene determinado por 20 attributos:

Nombre	Tipo	Descripción	Ejemplo
rank	int	Posición en el *ranking* de BGG	21
bgg_url	string	Link a url de la reseña en BGG	https://boardgamegeek.com/boardgame/167791/terraforming-mars
game_id	string	Identificador del juego en BGG	25613
names	string	Nombre del juego	Terraforming Mars
min_players	int	Número mínimo de jugadores	2
max_players	int	Número máximo de jugadores	4
avg_time	int	Tiempo medio de partida (minutos)	60
min_time	int	Tiempo mínimo de partida (minutos)	30
max_time	int	Tiempo máx de partida (minutos)	120
year	int	Año de publicación	2014
avg_rating	float	Puntuación media del juego según usuarios de BGG (sobre 10)	8.0096

Nombre	Tipo	Descripción	Ejemplo
geek_rating	float	Puntuació de BGG, obtenida a través de un algoritmo de ponderación bayesiana (algoritmo no público)	8.49837
num_votes	int	Número de usuarios que han dado puntuación al juego	1779
image_url	string	Enlace a la imagen del juego	https://cf.geekdo-images.com/images/pic361592.jpg
age	int	Edad mínimia recomendada	12
mechanic	string	Tipo de Mecánicas del juego, separadas por comas	Area Enclosure, Card Drafting, Hand Management, Variable Player Powers, Worker Placement
owned	int	Número de usuarios de BGG que han notificado que poseen el juego	18217
category	string	Categorías a las que pertenece el juego, separadas por comas	Ancient, Card Game, City Building, Civilization
designer	string	Diseñador/a del juego. Si más de uno, separados por comas	Jamey Stegmaier
weight	float	Grado de complejidad del juego (escala de 1 a 5)	2.394

2. Carga de datos y selección

Utilizaremos la librería `pandas` para trabajar con los datos así como `plotly` para realizar gráficos.

In [1]:
```
import pandas as pd
from IPython.display import display, HTML
import numpy as np

# Importamos plotly y también lo asignamos como backend de pandas plot.
import plotly.express as px
pd.options.plotting.backend = "plotly"

# Llamamos "bgg" al dataframe creado a partir del dataset
bgg = pd.read_csv('../csv/bgg_db_2018_01.csv',sep=',',encoding='latin-1')
```

Veamos qué aspecto tienen los datos, utilizando `.head()` .

In [2]:
```
bgg.head()
```

Out[2]:

	rank	bgg_url	game_id	names	min_players	max_players	avg
0	1	https://boardgamegeek.com/boardgame/174430/glo...	174430	Gloomhaven	1	4	
1	2	https://boardgamegeek.com/boardgame/161936/pan...	161936	Pandemic Legacy: Season 1	2	4	
2	3	https://boardgamegeek.com/boardgame/182028/thr...	182028	Through the Ages: A New Story of Civilization	2	4	

	rank	bgg_url	game_id	names	min_players	max_players	avg
3	4	https://boardgamegeek.com/boardgame/12333/twil...	12333	Twilight Struggle	2	2	
4	5	https://boardgamegeek.com/boardgame/167791/ter...	167791	Terraforming Mars	1	5	

Constatamos que los atributos `bgg_url` y `image_url` no nos van a ninguna información relevante al estudio que queremos realizar sobre los datos.

Eliminamos estos datos usando el método `.drop()` .

In [3]:
```
# Eliminamos las columnas que no utilizaremos para el análisis
bgg.drop('bgg_url'  , inplace=True, axis=1)
bgg.drop('image_url', inplace=True, axis=1)
```

Por otro lado, según la documentación de BoardGameGeek, para evaluar el *ranking* global del juego se utiliza el atributo `geek_rating` . Este atributo realiza una ponderación Bayesiana utilizando un algoritmo que BGG no publica.

Sí sabemos que en este algoritmo se compensan aquellos juegos con pocas valoraciones, pudiendo añadir hasta 100 votos *virtuales* (ver este enlace).

Para nuestro estudio, partiremos de aquellos datos que tengan como mínimo 500 valoracion de usuarios.

In [4]:
```
# Filtrar por mínimo de votos
bgg = bgg[bgg.num_votes > 500]
```

3. Limpieza de los datos

3.1 Valores nulos, ceros y elementos vacíos

En primer lugar, revisamos si existens valores nulos o elementos vacíos. Utilizaremos el método `isnull()` para determinar si un dato es nulo, y posteriormente `.sum()` para contar cuantos elementos nulos entontramos en cada atributo.

In [5]:
```
# Muestra elementos "null" de nuestro conjunto de datos bgg
bgg.isnull().sum()
```

Out[5]:
```
rank           0
game_id        0
names          0
min_players    0
max_players    0
avg_time       0
min_time       0
max_time       0
year           0
avg_rating     0
geek_rating    0
num_votes      0
age            0
mechanic       0
owned          0
category       0
```

```
designer        0
weight          0
dtype: int64
```

En el conjunto de datos `bgg` no observamos valores nulos.

Los valores nulos podrían estar representados también como valores "cero". Con `.describe()` obtendremos un resumen de nuestro *dataset* y podremos identificar artributos con valores "cero".

In [6]:
```
# Muestra valores con .describe(), redondeando a dos decimales
bgg.describe().round(2)
```

Out[6]:

	rank	game_id	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	geek
count	2721.00	2721.00	2721.00	2721.00	2721.00	2721.00	2721.00	2721.00	2721.00	2
mean	1541.47	76880.33	2.07	5.21	86.48	67.53	86.33	1996.08	6.92	
std	1078.85	71001.19	0.69	6.16	294.46	142.69	294.49	161.60	0.55	
min	1.00	1.00	0.00	0.00	0.00	0.00	0.00	-3000.00	5.77	
25%	681.00	8668.00	2.00	4.00	30.00	30.00	30.00	2003.00	6.50	
50%	1366.00	45986.00	2.00	4.00	60.00	45.00	60.00	2009.00	6.88	
75%	2205.00	144270.00	2.00	6.00	90.00	75.00	90.00	2014.00	7.31	
max	4987.00	236191.00	8.00	99.00	12000.00	6000.00	12000.00	2017.00	9.07	

Aquí se puede ver que para algunas observaciones encontramos juegos que tienen valores "cero", como son en las variables `min_players`, `max_players`, `avg_time`, `min_time`, `max_time`, `age` y `weight`.

3.1.1 Variable `min_players`

Empezemos analizando qué datos corresponden a `min_players = 0`.

In [7]:
```
# Mostramos aquellos registros en que la variable min_players = 0
bgg[bgg['min_players']==0]
```

Out[7]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
2164	2165	18291	Unpublished Prototype	0	0	0	0	0	0	6.97370
2475	2476	23953	Outside the Scope of BGG	0	0	0	0	0	0	6.73655
2523	2524	21804	Traditional Card Games	0	0	0	0	0	0	6.52588

Por el nombre del juego entendemos que estos registros corresponden a reseñas "genéricas" publicadas en el portal Board Game Geek, no se trata de juegos de mesa reales. De hecho los registros `min_players`, `max_players`, `avg_time`, `min_time`, `max_time`, `year` y `age` son todos `0`.

Decidimos retirar estos registros de nuestro conjunto de datos.

In [8]:
```
# Eliminamos los registros que corresponden a min_players = 0
bgg = bgg[bgg['min_players']!=0]
```

Volvamos a comprobar nuestros datos:

In [9]:
```
# Muestra valores con .describe(), redondeando a dos decimales
bgg.describe().round(2)
```

Out[9]:

	rank	game_id	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	geek
count	2718.00	2718.00	2718.00	2718.00	2718.00	2718.00	2718.00	2718.00	2718.00	2
mean	1540.53	76941.62	2.07	5.21	86.58	67.60	86.42	1998.29	6.92	
std	1079.07	71016.34	0.69	6.16	294.61	142.75	294.64	147.44	0.55	
min	1.00	1.00	1.00	1.00	1.00	0.00	0.00	-3000.00	5.77	
25%	680.25	8581.00	2.00	4.00	30.00	30.00	30.00	2003.00	6.50	
50%	1364.50	46110.00	2.00	4.00	60.00	45.00	60.00	2009.00	6.88	
75%	2203.75	144311.25	2.00	6.00	90.00	75.00	90.00	2014.00	7.31	
max	4987.00	236191.00	8.00	99.00	12000.00	6000.00	12000.00	2017.00	9.07	

Quedan por analizar los valores cero de los atributos `min_time`, `max_time`, `age` y `weight`.

3.1.2 Variable `min_time`

Veamos qué registros presentan `min_time = 0`.

In [10]:
```
# Mostramos aquellos registros en que la variable min_time = 0
bgg[bgg['min_time']==0]
```

Out[10]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
567	568	153016	Telestrations: 12 Player Party Pack	4	12	30	0	30	2011	7.81886
893	894	181810	Kodama: The Tree Spirits	2	5	40	0	40	2016	6.93798
1056	1056	175307	Adventure Time Love Letter	2	4	20	0	20	2015	7.29491
1252	1252	198487	Smash Up: Cease and Desist	2	2	45	0	45	2016	7.62728
3092	3093	2375	Sequence	2	12	30	0	30	1982	5.93718

En estos registros no consta el valor `min_time`. En este caso asumiremos que `min_time = max_time`. Corregimos estos registros.

In [11]:
```
# Sustituimos los valores '0' por el valor max_time
bgg['min_time'].where(bgg['min_time']!=0, bgg['max_time'], inplace=True)
```

3.1.3 Variable `max_time`

Comprobemos los registros `max_time = 0` .

In [12]:
```
# Mostramos aquellos registros en que la variable max_time = 0
bgg[bgg['max_time']==0]
```

Out[12]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
790	791	177524	Ice Cool	2	4	20	20	0	2016	6.93164
799	800	160902	Dungeons & Dragons Dice Masters: Battle for Fa...	2	2	60	60	0	2015	7.42356
1359	1359	205046	Capital Lux	2	4	30	30	0	2016	7.12118
1478	1478	193308	Spyfall 2	3	12	10	10	0	2017	7.27587
1802	1802	162559	Smash Up: Munchkin	2	4	45	45	0	2015	6.81948
1834	1834	206931	Noch mal!	1	6	20	20	0	2016	6.97665
1959	1959	174614	Apotheca	1	4	30	30	0	2016	6.93168
2457	2458	180602	Game of Trains	2	4	30	30	0	2015	6.39184
2520	2521	172971	Crossing	3	6	15	15	0	2013	6.63155
3127	3128	191982	Knit Wit	2	8	15	15	0	2016	6.28238

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
3140	3141	194232	Rick and Morty: Total Rickall Card Game	2	5	30	30	0	2016	6.44826
3436	3437	23010	Risk: 40th Anniversary Collector's Edition	2	6	120	120	0	1999	6.33854

Como hemos realizado con `min_time`, asumiremos que `max_time = min_time` para todo `max_time = 0`.

In [13]:
```
# Sustituimos los valores '0' por el valor min_time
bgg['max_time'].where(bgg['max_time']!=0, bgg['min_time'], inplace=True)
```

3.1.4 Variable age

Revisemos ahora la variable age en el caso `age = 0`.

In [14]:
```
# Mostramos aquellos registros en que la variable age = 0
bgg[bgg['age']==0].head(10)
```

Out[14]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
388	389	144189	Fire in the Lake	1	4	180	180	180	2014	8.05631
414	415	38996	Washington's War	2	2	90	90	90	2010	7.66716
477	478	111799	Cuba Libre	1	4	180	180	180	2013	7.77699
499	500	132018	Churchill	1	3	300	60	300	2015	7.75969
527	528	165722	KLASK	2	2	10	10	10	2014	7.67097
571	572	26990	Container	3	5	90	90	90	2007	7.12172
622	623	173064	Leaving Earth	1	5	180	60	180	2015	7.90326

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
665	666	30618	Eat Poop You Cat	3	99	20	20	20	1984	7.49211
763	764	163154	Falling Sky: The Gallic Revolt Against Caesar	1	4	360	180	360	2016	8.07103
885	886	146221	Forge War	1	4	180	60	180	2015	7.40848

Un total de 54 registros presentan `age = 0`. En este caso consideramos que el valor se desconoce, así que le asignaremos el valor vacío , reemplazando el valor `0`.

In [15]:
```
# Escribimos el valor None para los valores desconocidos del atributo age
bgg['age'].where(bgg['age']!=0, None, inplace=True)
```

3.1.5 Variable `weight`

Finalmente revisemos `weight = 0`.

In [16]:
```
bgg[bgg['weight']==0]
```

Out[16]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	geek
1252	1252	198487	Smash Up: Cease and Desist	2	2	45	45	45	2016	7.62728	6

Asumiremos pues que el valor de weight se desconoce. Por tanto retiraremos este registro del *dataset*. Consideramos que el grado de complejidad es una variable a tener en cuenta en nuestro estudio. No disponer de este dato impide realizar el estudio completo.

In [17]:
```
# Eliminamos los registros que corresponden a weight = 0
bgg = bgg[bgg['weight']!=0]
```

3.1.6 Valores nulos en Mecánicas y Categorías

Durante el análisis previo del *dataset* observamos que la la cadena de caracteres `none` describe los valores nulos en los atribuots `mechanic` y `owned`.

Veamos un ejemplo:

In [18]:
```
mchnc_none = bgg['mechanic'] == "none"
ctgry_none = bgg['category'] == "none"

bgg[mchnc_none | ctgry_none].head()
```

Out[18]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	gee

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	gee
171	172	163068	Trickerion: Legends of Illusion	2	4	180	60	180	2015	7.81778	
510	511	106217	Hawaii	2	5	90	90	90	2011	7.21125	
527	528	165722	KLASK	2	2	10	10	10	2014	7.67097	
532	533	200147	Kanagawa	2	4	45	45	45	2016	7.25164	
566	567	160851	Lanterns: The Harvest Festival	2	4	30	30	30	2015	7.02865	

Reemplazamos el *string* `"none"` por el valor nulo `None` .

In [19]:
```
from numpy import nan

bgg[ctgry_none] = bgg[ctgry_none].replace("none", nan)
bgg[mchnc_none] = bgg[mchnc_none].replace("none", nan)

# Mostramos registros en que ambos category y mechanic son None
bgg[bgg['category'].isnull() & bgg['mechanic'].isnull()].count()
```

Out[19]:
```
rank           0
game_id        0
names          0
min_players    0
max_players    0
avg_time       0
min_time       0
max_time       0
year           0
avg_rating     0
geek_rating    0
num_votes      0
age            0
mechanic       0
owned          0
category       0
designer       0
weight         0
dtype: int64
```

Observamos que no hay ningún registro com ambos valores `mechanic` y `caegory` nulos.

3.2 Análisis de valores extremos

Tras haber tratado los valores nulos, volvamos a revisar las estadísticas sobre nuestro *dataset*.

In [20]:
```
bgg.describe().round(2)
```

Out[20]:

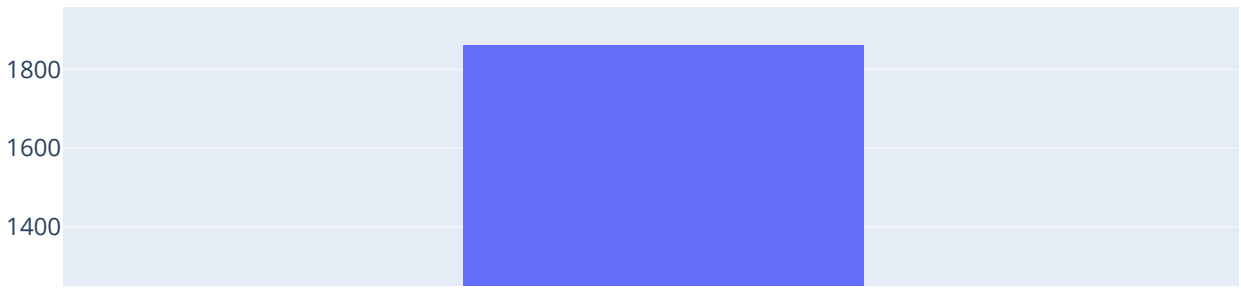
	rank	game_id	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	geek
count	2717.00	2717.00	2717.00	2717.00	2717.00	2717.00	2717.00	2717.00	2717.00	2
mean	1540.64	76896.88	2.07	5.21	86.59	67.67	86.59	1998.28	6.92	
std	1079.25	70991.10	0.69	6.16	294.66	142.75	294.66	147.46	0.55	
min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	-3000.00	5.77	
25%	680.00	8552.00	2.00	4.00	30.00	30.00	30.00	2003.00	6.50	
50%	1365.00	46007.00	2.00	4.00	60.00	45.00	60.00	2009.00	6.88	
75%	2204.00	144270.00	2.00	6.00	90.00	75.00	90.00	2014.00	7.31	
max	4987.00	236191.00	8.00	99.00	12000.00	6000.00	12000.00	2017.00	9.07	

Si comparamos los valores máximos y mínimos con la media aritmética (`mean`) y la mediana (percentil 50) observamos que los atributos `min_players` , `max_players` , `avg_time` , `min_time` , `max_time` y `year` presentan valores extremos.

3.2.1 Valores extremos de `min_players` y `max_players`

Empecemos analizando los valores extremos de las variables `min_players` y `max_players` .

In [21]:
```
px.histogram(bgg, x="min_players")
```



In [22]:
```
px.histogram(bgg, x="max_players")
```

In [23]:
```
bgg[bgg.min_players > 4].head(10)
```

Out[23]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
77	78	128882	The Resistance: Avalon	5	10	30	30	30	2012	7.68066
159	160	41114	The Resistance	5	10	30	30	30	2009	7.38861
204	205	188834	Secret Hitler	5	10	45	45	45	2016	7.61138
643	644	38159	Ultimate Werewolf: Ultimate Edition	5	68	90	30	90	2008	7.10933
719	720	25821	The Werewolves of Miller's Hollow	8	18	30	30	30	2001	6.83518

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
728	729	134352	Two Rooms and a Boom	6	30	15	15	15	2013	7.07414
822	823	184424	Mega Civilization	5	18	720	360	720	2015	8.31974
873	874	135779	A Fake Artist Goes to New York	5	10	20	20	20	2012	7.15159
1110	1110	925	Werewolf	8	24	60	60	60	1986	6.63330
1243	1243	176558	Mafia de Cuba	6	12	20	10	20	2015	6.67716

In [24]:
```
bgg[bgg.max_players > 12].head(10)
```

Out[24]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	g
277	278	36553	Time's Up! Title Recall!	4	18	60	60	60	2008	7.78343	
370	371	1353	Time's Up!	4	18	90	90	90	1999	7.34153	
494	495	156546	Monikers	4	20	60	60	60	2015	7.77531	
564	565	51	Ricochet Robots	1	99	30	30	30	1999	7.00243	
643	644	38159	Ultimate Werewolf: Ultimate Edition	5	68	90	30	90	2008	7.10933	
665	666	30618	Eat Poop You Cat	3	99	20	20	20	1984	7.49211	

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	g
719	720	25821	The Werewolves of Miller's Hollow	8	18	30	30	30	2001	6.83518	
728	729	134352	Two Rooms and a Boom	6	30	15	15	15	2013	7.07414	
822	823	184424	Mega Civilization	5	18	720	360	720	2015	8.31974	
830	831	1540	BattleTech	2	20	90	90	90	1985	6.97916	

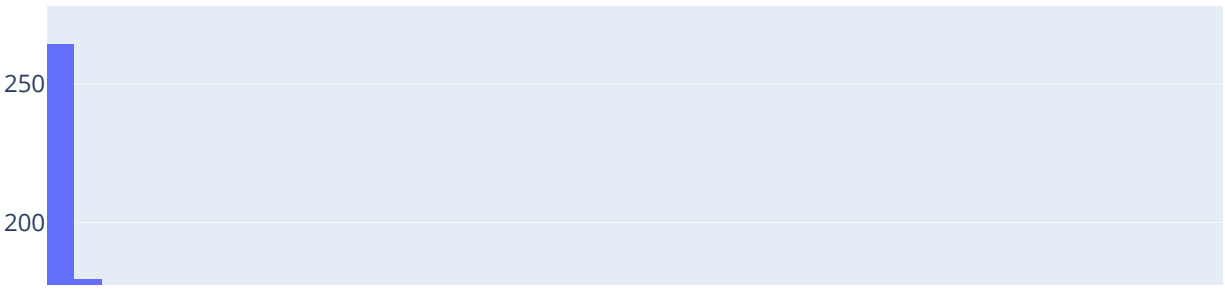
Observamos que estos juegos con valores extremos en `min_players` o `max_players` corresponden en su mayoría a juegos de tipo colaborativo, grupales o participativos, que requieren de un mínimo necesario de jugadores y pueden no especificar un número máximo.

Estos valores extremos no son pues erróneaos, por lo que no modificaremos estos valores, pero tendremos en cuenta este análisis en caso necesario.

3.2.2 Valores extremos de `min_time`, `max_time` y `avg_time`

A continuación evaluemos los valores extremos de los atributos que hacen referencia a la duración de las partidas. Empezemos observando la distribución de `min_time` para juegos que tengan una duración superior al percentil 75.

In [25]:
```
perc75 = bgg.min_time.quantile(.75)
px.histogram(bgg[bgg.min_time > perc75], x="min_time")
```



Son pocos los juegos que presentan una duración mínima superior a 500 minutos. Revisemos estos valores extremos.

In [26]:
```
bgg[bgg.min_time > 500]
```

Out[26]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	ge
848	849	6205	Europe Engulfed	2	3	720	720	720	2003	7.51743	
880	881	2081	The Civil War	2	2	1200	1200	1200	1983	7.65883	
930	930	254	Empires in Arms	2	7	12000	6000	12000	1983	7.53768	
1385	1385	1563	Rise and Decline of the Third Reich	2	6	1440	1440	1440	1974	6.81558	
1641	1641	17393	Pax Romana	2	4	600	600	600	2006	7.34684	
1974	1974	329	Russian Front	2	2	1200	1200	1200	1985	7.15687	
2313	2314	283	Advanced Third Reich	2	6	2480	2480	2480	1992	6.75945	

Hemos complementado el análisis de nuevo realizando una búsqueda en BoardGameGeek de estos juegos para comprender mejor los datos.

Estos son juegos de tipo estratégico, de campaña o *legacy*. Son juegos cuyas partidas son muy largas (llegando a durar decenas de partidas). Estos valores extremos pues son correctos, no se trata de errores en el database.

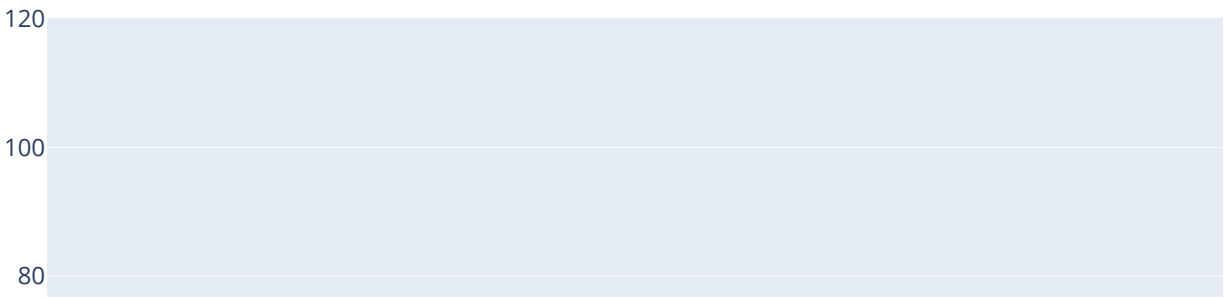
Tendremos en cuenta este tipo de datos al realizar el análisis.

Analizando `min_time` observamos que también son juegos con un `max_time` muy elevado.

3.2.3 Valores extremos de `year`

Finalmente revisemos el atributo `year`, representando la distribución del primer percentil 5%.

In [27]:
```
px.histogram(bgg[bgg.year < bgg.year.quantile(0.05)], x='year')
```



Revisemos los datos de los juegos publicados antes de 1800.

In [28]:
```
bgg[bgg.year < 1800]
```

Out[28]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
112	113	188	Go	2	2	180	30	180	-2200	7.66016
412	413	171	Chess	2	2	60	60	60	1475	7.08996

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating
551	552	2398	Cribbage	2	4	30	30	30	1630	7.02070
988	988	2393	Xiangqi	2	2	60	60	60	762	7.15497
1079	1079	2065	Shogi	2	2	60	60	60	1000	7.27231
1082	1082	2397	Backgammon	2	2	30	30	30	-3000	6.52509
1170	1170	5072	Carrom	2	4	60	60	60	0	7.00855
2350	2351	2932	Hnefatafl	2	2	20	20	20	400	6.51044
2566	2567	5451	Hanafuda	2	7	60	60	60	1701	6.70123
2591	2592	15889	Scopa	2	6	30	30	30	1600	6.55147
2637	2638	125048	Dobble Free Demo Version	2	8	5	5	5	0	6.62907
2706	2707	4505	Tarot	3	5	30	30	30	1430	6.70783
3444	3445	2448	Kalah	2	2	10	10	10	700	5.85695

Comprobamos que se trata de juegos clásicos o históricos, como puede ser el ajedrez o el backgammon. La fecha es pues una aproximación según el conocimiento que se tenga del juego.

Los valores `year = 0` sí corresponden a un valor desconocido del año de publicación (ver Carrom, por ejemplo). Asignaremos el varlos `None` como hemos realizado en el apartado de gestión de valores vacíos.

In [29]:
```
# Sustituimos los valores '0' por None
bgg['year'].where(bgg['year']!=0, None, inplace=True)
```

3.2.5 Cálculo correcto de `avg_time`

Realizando el análisis de los datos hemos observado que la variable `avg_time` es en todos los casos igual a `max_time` . Vamos a recalcular esta variable de manera que

$$time_{avg} = \frac{time_{max} + time_{min}}{2}$$

In [30]:
```
bgg.avg_time = (bgg.max_time + bgg.min_time) / 2
bgg.head(5)
```

Out[30]:

	rank	game_id	names	min_players	max_players	avg_time	min_time	max_time	year	avg_rating	g
0	1	174430	Gloomhaven	1	4	120.0	90	150	2017.0	9.01310	
1	2	161936	Pandemic Legacy: Season 1	2	4	60.0	60	60	2015.0	8.66575	
2	3	182028	Through the Ages: A New Story of Civilization	2	4	210.0	180	240	2015.0	8.65702	
3	4	12333	Twilight Struggle	2	2	150.0	120	180	2005.0	8.35188	
4	5	167791	Terraforming Mars	1	5	120.0	120	120	2016.0	8.38331	

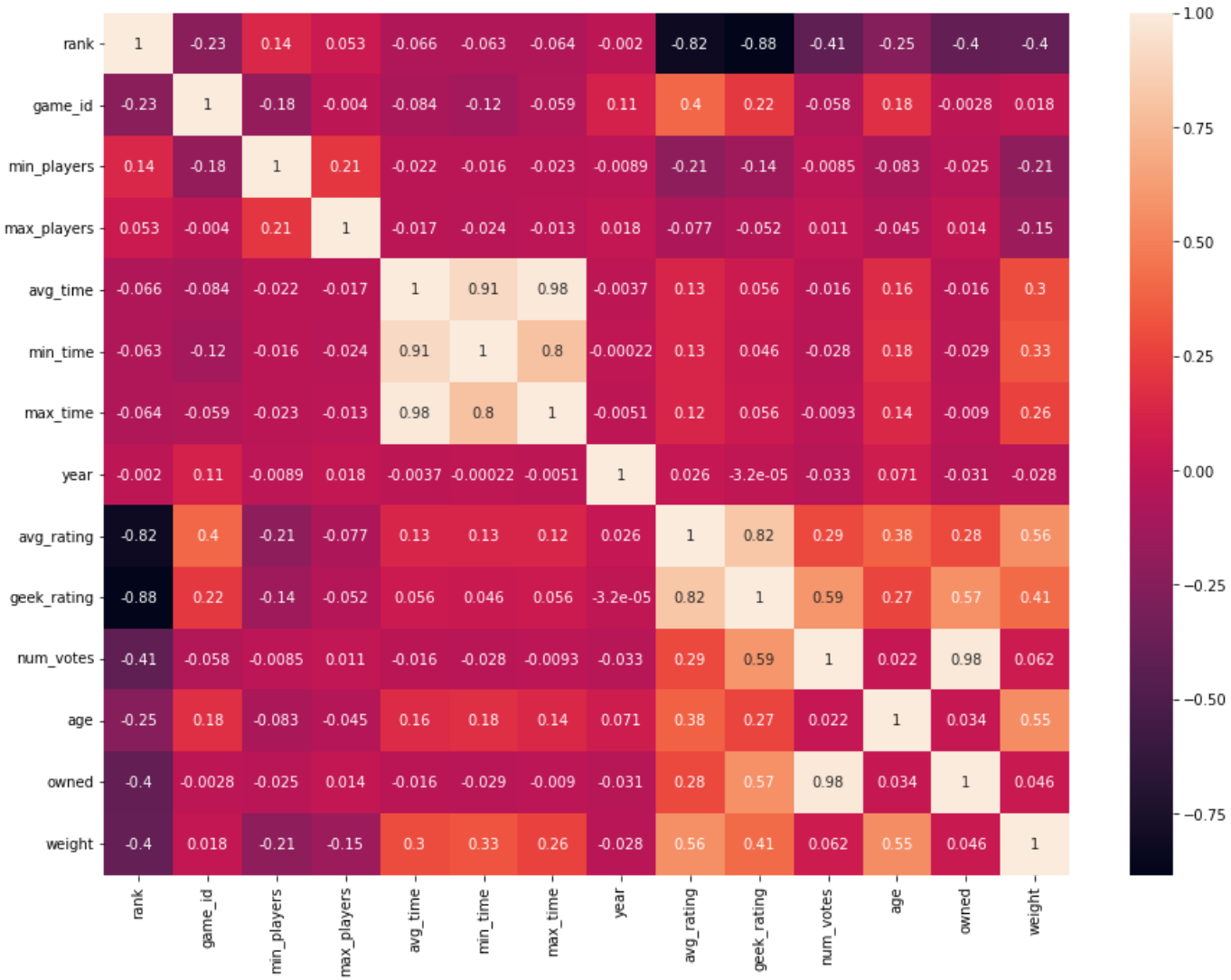
4. Análisis de los datos

4.1 Matriz de Correlación

Tras haber limpiado los datos revisemo la matriz de correlación. Utilizaremos la librería `seaborn` y `matplotlib` .

In [31]:
```
import seaborn as sns
from matplotlib import pyplot as plt
fig, ax = plt.subplots(figsize=(15,11))

# Calculamos la matriz de correlación
corr = bgg.corr()

# Representamos la matriz de correlación
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True)
```

Out[31]:
```
<AxesSubplot:>
```



En primer lugar observamos una fuerte correlación entre `avg_time`, `min_time` y `max_time`. Esto es de esperar dada la forma en que hemos calculado `avg_time`. También es lógica la correlación entre `min_time` y `max_time`, relación que ya hemos notado en el apartado de limpieza de los datos.

Por otro lado es notable la relación entre `geek_rating` y `avg_rating`. Dado que `geek_rating` se calcula con un algoritmo y considera `avg_rating` como uno de los *inputs* esta correlación también es esperable.

También observamos correlación entre la complejidad (`weight`) el el tiempo de duración de la partida. Posiblemente a mayor complejidad mayor tiempo requerido para jugar al juego.

Comentar también que hay una correlación entre `weight` y `age`. De nuevo, parece lógico pensar que a mayor complejidad también la edad mínima requerida pueda ser más elevada.

A continuación analizaremos qué parámetro es más adecuado para determinar la **popularidad** de un juego. Vemos que el atributo `owned` y `num_votes` tienen una fuerte correlación. Analicemos ambas variables con detalle.

4.2 Popularidad de un juego

Queremos comprender qué métrica podemos usar para considerar que un juego es **popular**. Disponemos de la variable `owned`, que indica el número de usuarios que han notificado que poseen el juego.

A partir de la matriz de correlación es comprensible asumir que los juegos que tengan más votos de los usuarios (`num_votes`) puedan ser también juegos más populares.

Por otro lado, el hecho de que un juego tenga un `avg_rating` menor o mayor no necesariamente indica que sea popular. Podemos encontrar juegos de muchísima calidad y con una puntuación elevada, pero que no sea asequible (por precio, complejidad, mecánica, etc) a la mayoría del público.

Veamos la relación entre `owned` y `num_votes`.

In [32]:
```
# Realizo un scatter plot de num_votes vs owned
fig_1 = px.scatter(bgg, x='owned', y='num_votes',
                   title="Relación entre Número de votos y Usuarios que tienen el juego",
                   labels = {
                       "owned": "Usuarios que tienen el juego (owned)",
                       "num_votes": "Usuarios que han valorado el juego (num_votes)"
                   })
fig_1.show()
```

Exite pues una relación lineal entre ambas variables `owned` y `num_votes`. Dada la concentración de *data points* en el extremo inferior izquierdo del gráfico, realizamos un *scatter plot* con ejes logarítmicos.

Analizemos la **homocedasticidad** entre ambos atributos. Utilizaremos la Prueba de Levene, disponible en la librería `scipy`.

In [33]:
```
from scipy.stats import levene

df = bgg['owned']
norm_rating = (df - df.min()) / (df.max() - df.min())

df2 = bgg['num_votes']
norm_votes = (df2 - df2.min()) / (df2.max() - df2.min())
```

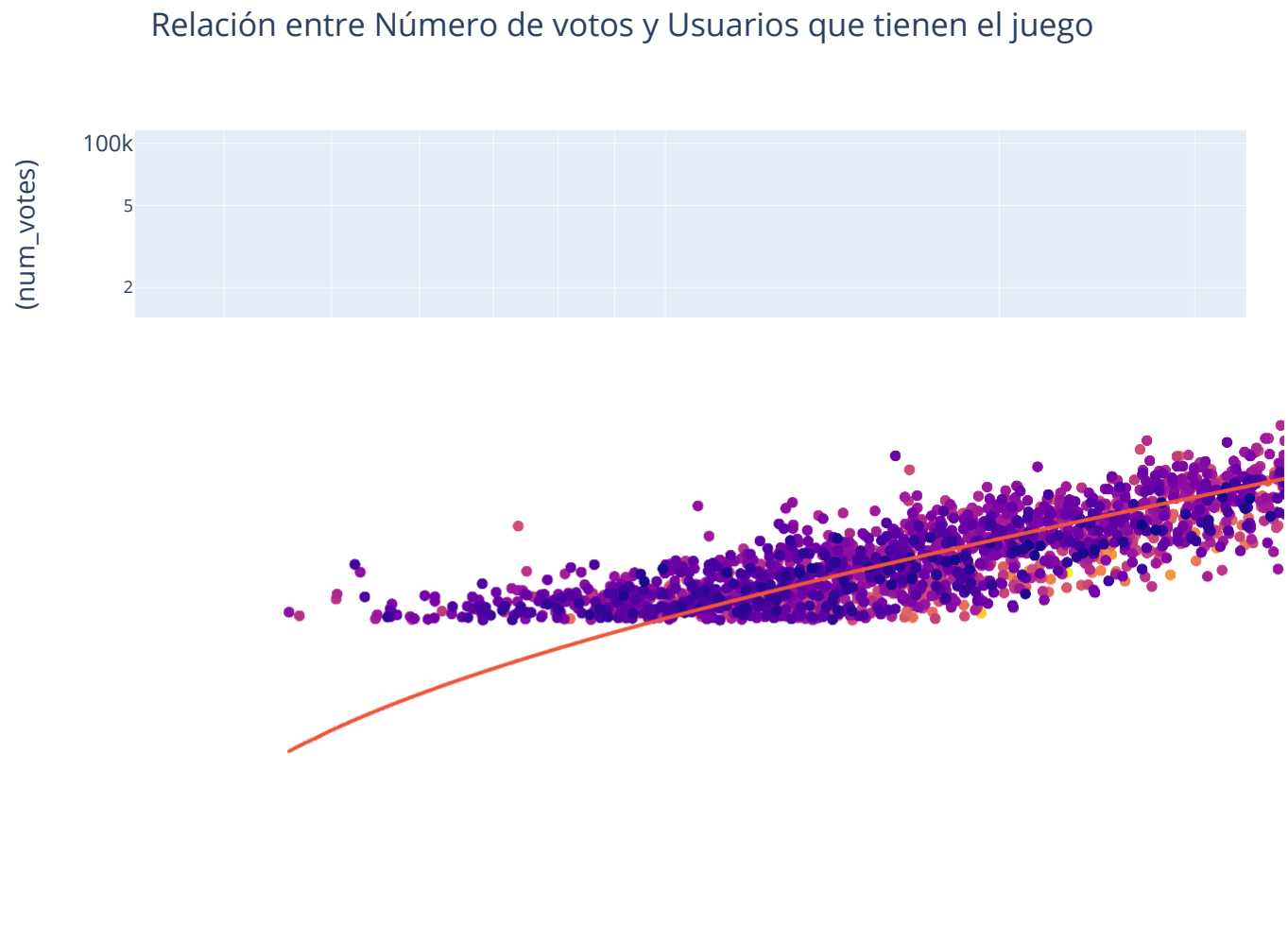
```
stat, p = levene(norm_rating, norm_votes)

print("p-value = ", round(p,5))
```

```
p-value =  0.44065
```

El *p-value* es mayor que el nivel de significancia $\alpha = 0.05$, luego ambas variables son aptas para elaborar un modelo de regresión.

Añadiremos pues una recta de regresión sobre los datos, coloreando cada *data point* según su `avg_rating` .

In [34]:
```
fig_2 = px.scatter(bgg, x='owned', y='num_votes', trendline='ols', log_x=True, log_y=True,
                   color='avg_rating', title="Relación entre Número de votos y Usuarios que tie
                   labels = {
                     "owned": "Usuarios que tienen el juego (owned)",
                     "num_votes": "Usuarios que han valorado el juego (num_votes)"
                  })
fig_2.show()

# Obtenemos el factor R2 que indica la bondad de la regresión:
r_squared = px.get_trendline_results(fig_2).px_fit_results.iloc[0].rsquared
print("El valor R squared es: ", round(r_squared,3))
```




```
El valor R squared es:  0.967
```

La bondad del ajuste según el parámetro R^2 es $R^2 = 0.967$, luego la recta de regresión se ajusta muy bien a los datos de la curva `num_votes` vs `owned` .

Consideraremos entonces que `num_votes` es un indicador de la **popularidad** que tiene un juego. Podremos también comparar si es necesario con `owned` , pese a que no esperaremos obtener resultados

muy diferentes.

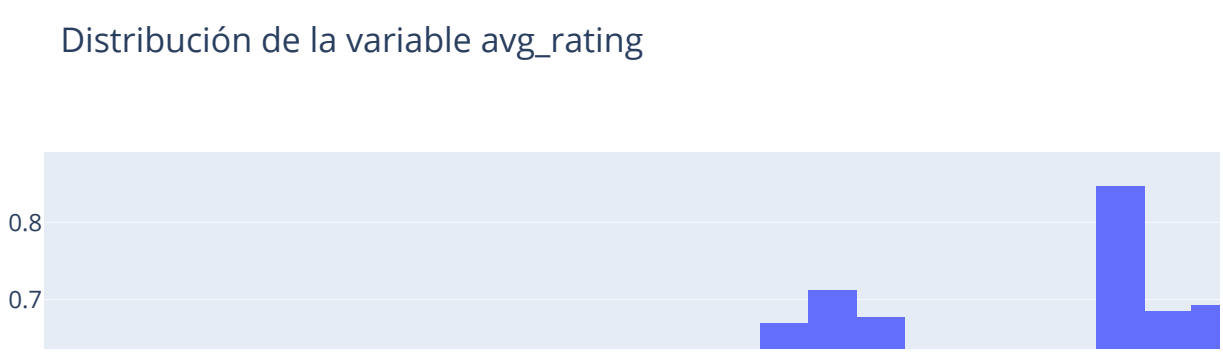
4.3 Análisis de la Valoración Media `avg_rating`

La valoración media de los usuarios tiene cierto grado de correlación con `num_votes`. También `geek_rating` tiene una correlación elevada, pero siendo esta una variable calculada y desconociendo el método de cálculo nos centramos en la valoración media dada por los usuarios.

El análisis de `num_votes` nos ayudará a acotar el espacio de diseño de nuestro juego de mesa.

Representamos en primer lugar la distribución de esta variable.

In [35]:
```
fig_rating = px.histogram(bgg, x="avg_rating", histnorm="probability density", title="Distribuc
                  labels = {
                      "avg_rating": "Valoración media de los usuarios (avg_rating)",
                  })
fig_rating.show()
```



Comprobemos con el test de *Shapiro-Wilk* la normalidad de esta distribución. Hemos consultado en la documentación de `plotly` (ver este enlace) cómo utilizar la librería `scipy` para realizar el análisis.

In [36]:
```
# Importamos la librería scipy.stats
from scipy.stats import shapiro

# Realizamos el test de shapiro, guardando el resultado en las variables stat y p
stat, p = shapiro(bgg['avg_rating'])

# Interpretamos el resultado
alpha = 0.05
if p > alpha:
```

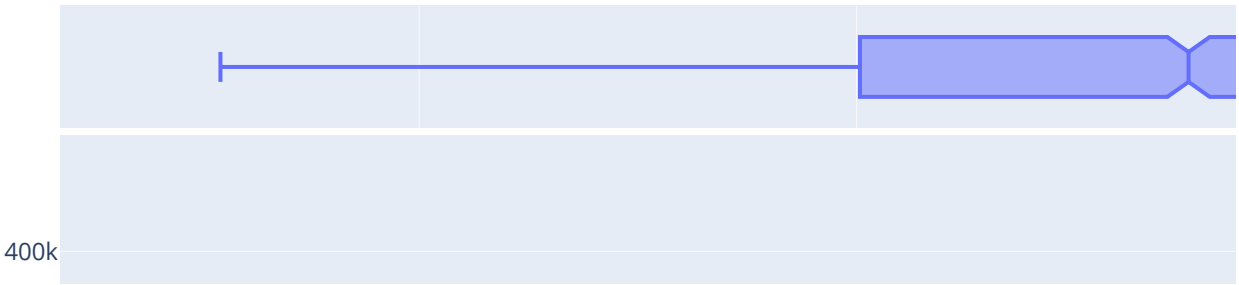
```
    msg = 'Sample looks Gaussian (fail to reject H0)'
else:
    msg = 'Sample does not look Gaussian (reject H0)'

print("p-value = ", p)
print(msg)
```

```
p-value =  1.086880596543841e-15
Sample does not look Gaussian (reject H0)
```

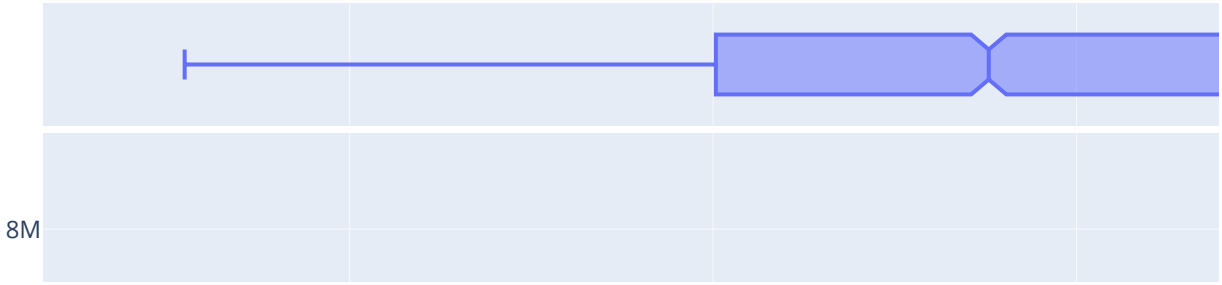
Del análisis de normalidad obtenemos que la variable `avg_rating` **no sigue una distribución normal**.

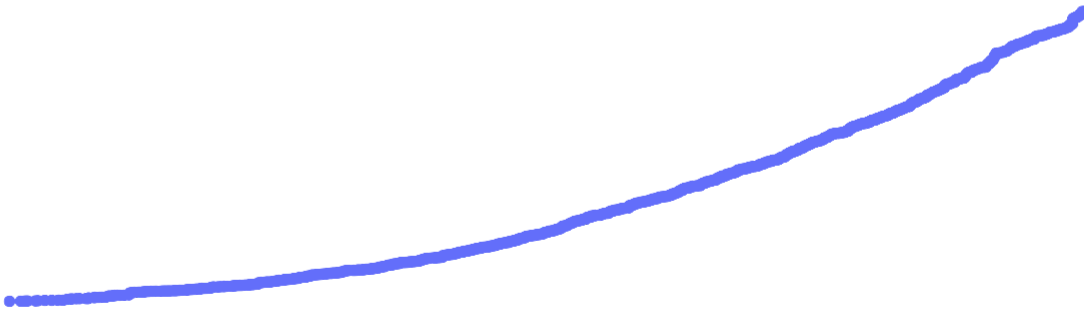
A continuación obtengamos la distribución de `avg_rating` con respecto a `num_votes` . Realizamos un hisograma así como un gráfico cumulativo.

In [37]:
```
fig = px.histogram(bgg, x="avg_rating", y="num_votes", marginal="box", histfunc="sum", hover_da

fig.show()
```



In [38]:
```
# Obtengo variables avg_rating y num_votes
df = bgg[['avg_rating','num_votes']]

# Ordeno por avg_rating
df = df.sort_values(by='avg_rating')

# Calculo suma cumulativa
df['votes_cumsum'] = df['num_votes'].cumsum()

px.scatter(df, x='avg_rating', y='votes_cumsum', marginal_x='box',marginal_y='box')
```





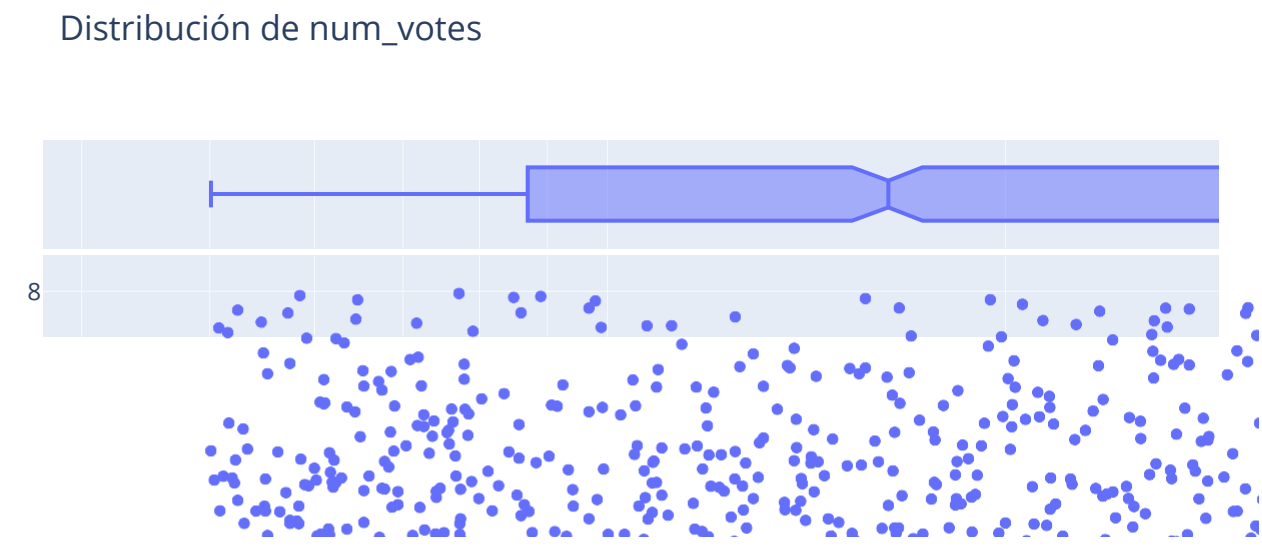
A partir de los gráficos anteriores centraremos nuestra atención en aquellos juegos que han obtenido un `avg_rating` mínimo de 6.5 (corresponde al percentil 25) y como máximo un `avg_rating` de 8 (valor a partir del cual apenas se incrementa la suma cumulativa de `num_votes`).

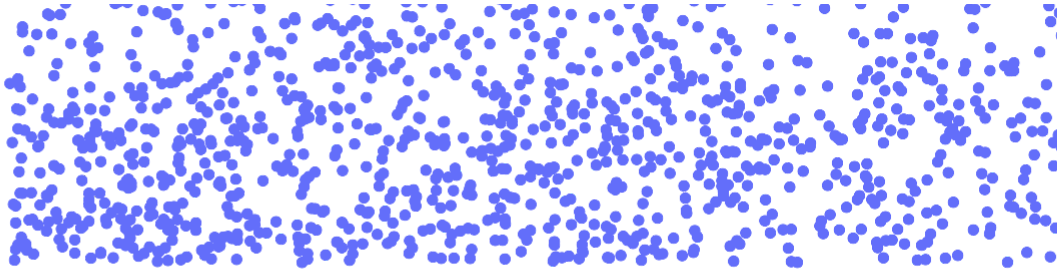
In [39]:
```
min_rating = 6.5
max_rating = 8.0

df_game = bgg[(bgg['avg_rating'] > min_rating) & (bgg['avg_rating'] < max_rating)]
```

De esta manera conseguimos acotar nuestro conjunto de datos para estudiar aquellos juegos más populares.

Si sobre estos datos representamos la distribución de `num_votes`:

In [40]:
```
px.scatter(df_game, x="num_votes", y="avg_rating", log_x=True, marginal_x="box", title="Distrib
```





Acotaremos finalmente nuestro juego de datos para aquellos juegos que pertenecen al percentil 25 superior. Estos corresponden a los juegos que han tenido más popularidad, tras haber estudiado la relación con `avg_rating` .

In [41]:
```
perc75 = df_game['num_votes'].quantile(0.75)

df_game = df_game[df_game['num_votes'] >= perc75]
```

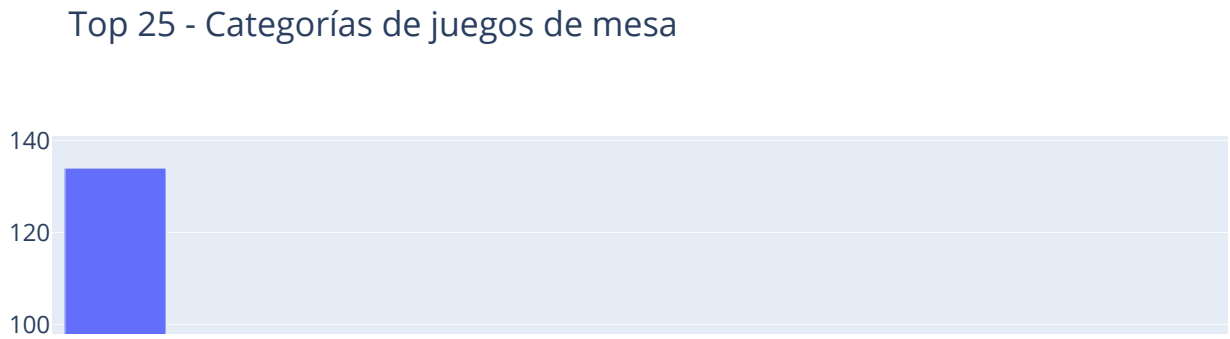
4.4 Categorías de Juegos de Mesa

Queremos ver cúales son las temáticas o categorías que más juegos hay en el mercado, para de este modo saber las tendencias tanto actuales como pasadas. De este modo, podemos ver cuáles podrían ser las posibles oportunidades de negocio a la hora de crear nuestro juego de mesa.

El análisis de las categorías de los juegos de mesa con un alto número de votos, nos dará una mejor visión sobre los juegos de mesa más comprados y valorados.

In [42]:
```
# Separamos las categorías de cada observación y se cuentan el total de cada una de ellas.
cat = df_game.category.str.get_dummies(', ').stack().sum(level=1)

# Ordenamos dichas categorías y las mostramos por pantalla.
px.bar(cat.sort_values(ascending = False).head(25), title="Top 25 - Categorías de juegos de mes
      labels = {
          "index" : "Categorías de Juegos de mesa (category)",
          "value" : "Frecuencia (count)"
      })
```

```
C:\Miniconda3\envs\boardgame\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:

Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed
in a future version. Use groupby instead. df.sum(level=1) should use df.groupby(level=1).sum().
```



Vemos que de las 5 categorías más populares son `Card Game`, `Fantasy`, `Fighting`, `Economic` y `Science Fiction`.

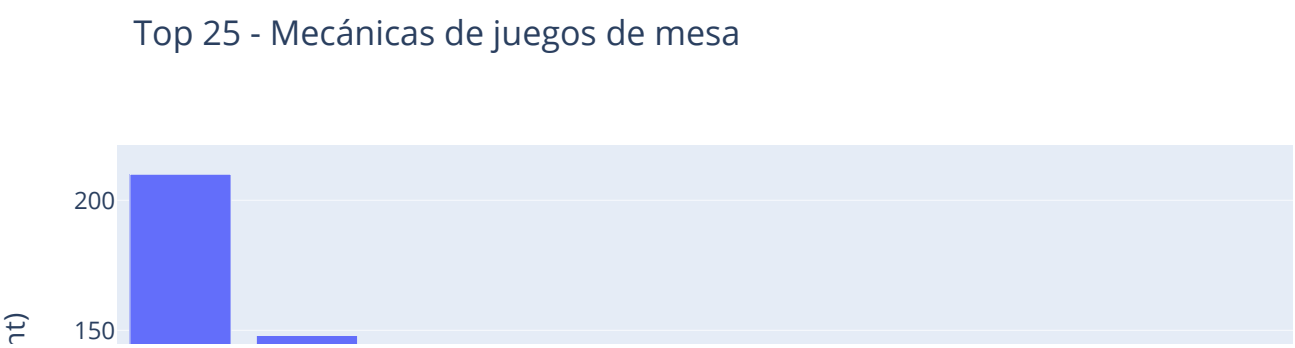
La categoría más frecuentre entre los juegos más populares es `Card Game`, por lo que podría ser un comienzo sobre el tipo de juego que queremos crear o el enfoque a dar.

4.5 Mecánicas de Juegos de Mesa

Por otro lado analizamos las mecánicas más empleadas. Así conoceremos los estilos de juegos favoritos de la comunidad.

In [43]:
```
# Separamos las mecánicas de cada observación y se cuentan el total de cada una de ellas.
mech = df_game.mechanic.str.get_dummies(', ').stack().sum(level=1)

# Ordenamos dichas categorías y las mostramos por pantalla.
px.bar(mech.sort_values(ascending = False).head(25), title="Top 25 - Mecánicas de juegos de mes
      labels = {
          "index" : "Mecánicas de Juegos de mesa (category)",
          "value" : "Frecuencia (count)"
      })
```

```
C:\Miniconda3\envs\boardgame\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:

Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed
in a future version. Use groupby instead. df.sum(level=1) should use df.groupby(level=1).sum().
```

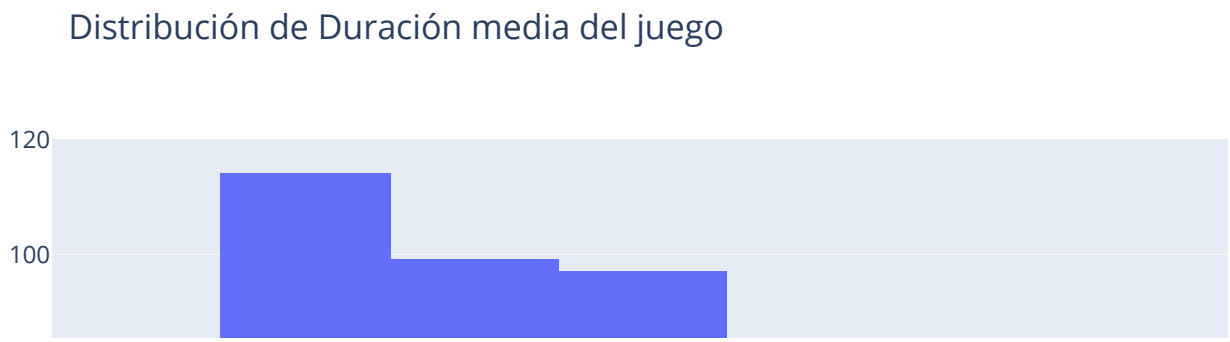


Vemos que los más jugados son `Hand Management`, `Dice Rolling`, `Variable Player Powers`, `Set Collection` y `Card Drafting`.

4.6 Duración media del juego

Podemos evaluar también qué duración de partida podríamos esperar de nuestro juego de mesa. Analizamos `avg_time`.

In [44]:
```
px.histogram(df_game, x="avg_time", nbins=35, title="Distribución de Duración media del juego",
             labels={
                 "avg_time":"Duración Media del Juego",
                 "count":"Frecuencia (count)"
             })
```

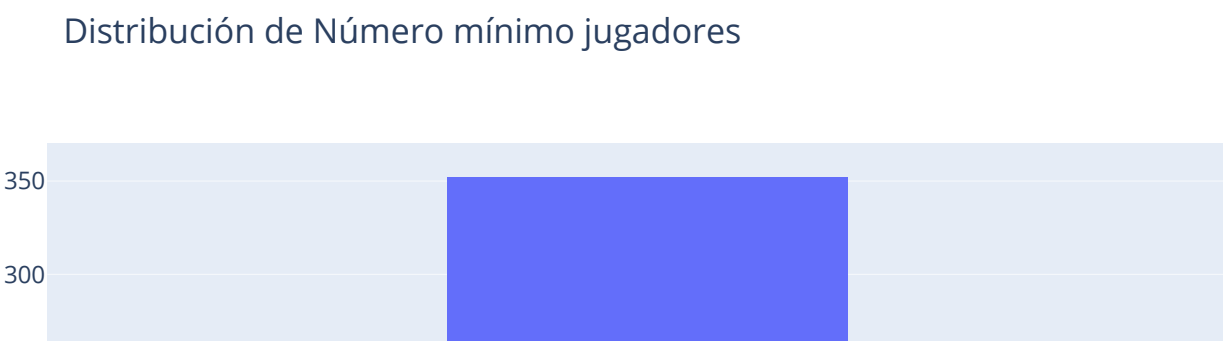


Considermaos que el juego debería tener una duración entre 30min y 100min según los datos analizados.

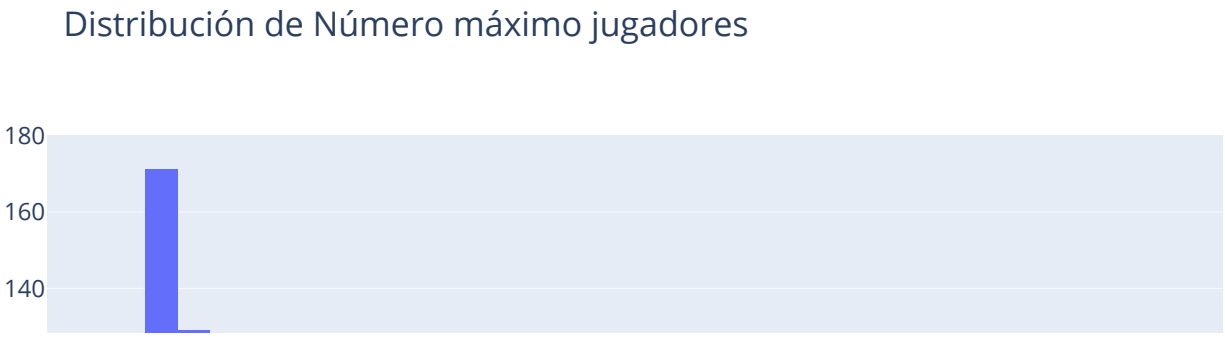
4.7 Número de jugadores

Evaluamos también el número de jugadores recomendado. Analizamos `min_players` y `max_players`.

In [45]:
```
px.histogram(df_game, x="min_players", nbins=10, title="Distribución de Número mínimo jugadores
             labels={
                 "min_players":"Número mínimo de jugadores",
                 "count":"Frecuencia (count)"
             })
```



In [46]:
```
px.histogram(df_game, x="max_players", nbins=100, title="Distribución de Número máximo jugadore
             labels={
                 "max_players":"Número máximo de jugadores",
                 "count":"Frecuencia (count)"
             })
```

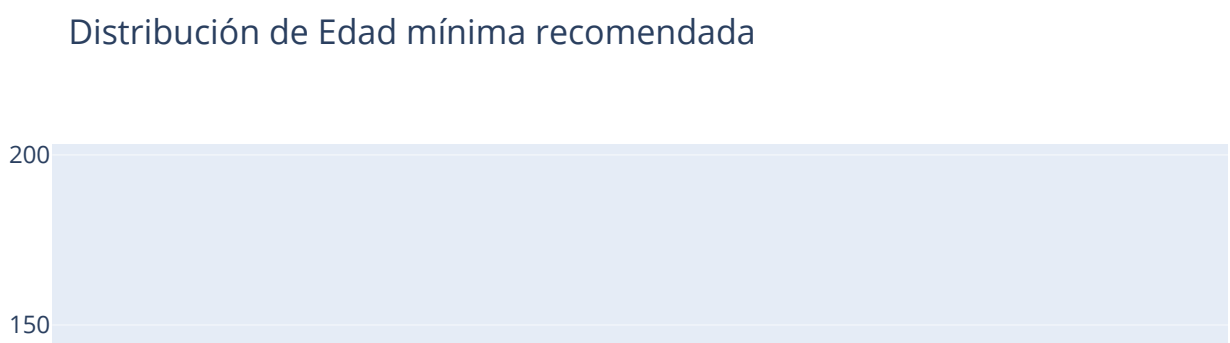


Nuestro juego de mesa deberá poderse jugar entre 2 y hasta 6 jugadores.

4.8 Edad mínimia recomendada y complejidad del juego

Finalmente veamos la edad mínima recomendada así como el grado de complejidad con que diseñar el juego.

In [47]:
```
px.histogram(df_game, x="age", nbins=10, title="Distribución de Edad mínima recomendada",
             labels={
                 "age":"Edad mínima recomendada",
                 "count":"Frecuencia (count)"
             })
```

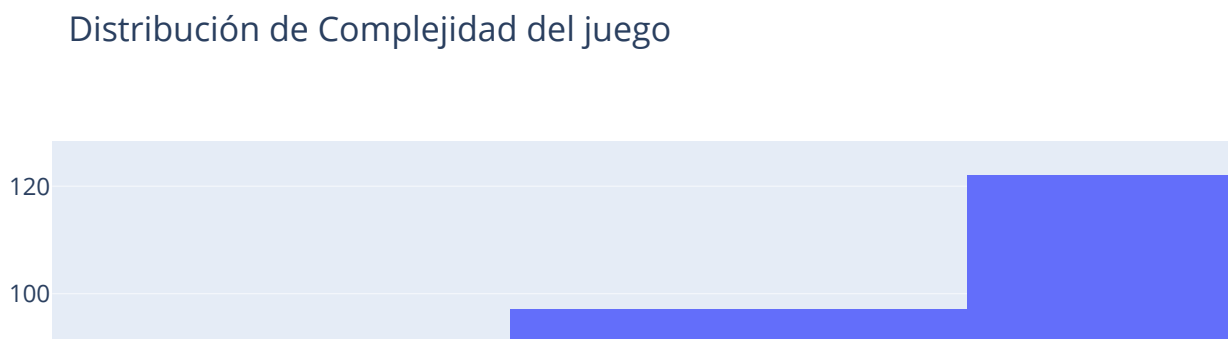


Según BGG se utilizan 5 niveles de complejidad:

- Light (1)
- Medium Light (2)

- Medium (3)
- Medium Heavy (4)
- Heavy (5)

In [48]:
```
px.histogram(df_game, x="weight", nbins=10, title="Distribución de Complejidad del juego",
             labels={
                 "weight":"Nivel de Complejidad",
                 "count":"Frecuencia (count)"
             })
```



Según estos datos el nivel de complejidad debe encontrarse entre ligeramente superior a Light (1) y no superior a Medium (3).

5. Resolución del problema y conclusiones

Hemos querido analizar este conjunto de datos para determinar el espacio de diseño de un nuevo juego de mesa para maximizar su popularidad. Para lograrlo, hemos preparado el conjunto de datos adecuadamente y analizado las variables más representativas.

Hemos decidido usar `num_votes` como índica de popularidad y tras acotar el *dataset* hemos analizado los diferentes atributos para determinar el espacio de diseño de nuestro juego de mesa:

Atributo	Espacio de solución
Duración de la partida	Entre 30 y 100 minutos
Número de jugadores	Entre 2 y 6 jugadores

Atributo	Espacio de solución
Duración de la partida	Entre 30 y 100 minutos
Edad mínima recomendada	10 años
Categorías	Card Game, Fantasy, Fighting, Economic y Science Fiction
Mecánicas	Hand Management, Dice Rolling, Variable Player Powers, Set Collection y Card Drafting

5.1 Ejemplo de espacio de diseño del juego de mesa

A modo de ejemplo, podríamos ambientar un **juego de cartas** (*Card Game*) en un futuro distópico (*Science Fiction*) en que diferentes razas (clásico en los géneros de *Fantasy*, por ejemplo, humanos terrícolas, humanos marcianos, alienígenas, mutantes, cyborgs y la IA) compiten por los recursos del Sistema Solar (*Economy*) por lo que tendrán que forjar alianzas, comerciar a la par que luchar por el domino del Sistema (*Fighting*).

Las razas poseerán habilidades completamente diferentes (*Variable Player Powers*), que podrán ser potenciadas mediante la colección de su recurso más preciado (*Set Collection*), como puede ser el agua para los terrícolas y los marcianos, la materia oscura para los alienígenas, el uranio para los mutantes, el oxígeno para los cybors y el silicio para la IA (Inteligencia Artificial).

Las acciones a realizar (por ejemplo: comerciar, espiar o batallar) en cada turno dependerán del uso de la manos de cartas de cada jugador (*Hand Management*), que irá alimentándose (*Card Drafting*) cada inicio de turno según su desarrollo como raza y su dominio sobre el Sistema.

Cada turno ser verá afectado por una catastrofe natural o cataclismo cósmico de consecuencias imprevisibles. El azar (*Dice Rolling*) determinará si eventos como una explosión solar, el impacto de un meteorito el cambio abrupto del campo magnético de la Tierra devastan los recursos de cada raza.

6. Tabla de Contribuciones

Las contribuciones de Ignacio Such constan como IS, las de Andrés Fonts como AF.

Contribuciones	Firma
Investigación previa	IS, AF
Redacción de las respuestas	IS, AF
Desarrollo del código	IS, AF

Referencias consultadas

Board Game Rank [en línea] [fecha de consulta: 30 de mayo de 2022]. Disponible en: https://boardgamegeek.com/browse/boardgame?sort=rank&sortdir=desc

BoardGameGeek FAQ [en línea] [fecha de consulta: 31 de mayo de 2022]. Disponible en: https://boardgamegeek.com/wiki/page/BoardGameGeek_FAQ#toc13

BoardGameWiki. Weight [en línea] [fecha de consulta: 31 de mayo de 2022]. Disponible en: https://boardgamegeek.com/wiki/page/Weight

How to delete a column in pandas [en línea] [fecha de consulta: 01 de junio de 2022]. Disponible en: https://www.educative.io/edpresso/how-to-delete-a-column-in-pandas

Pandas Plotting Backend in Python [en línea] [fecha de consulta: 02 de junio de 2022]. Disponible en: https://plotly.com/python/pandas-backend/

Working with Markdown tables in GitHub [en línea] [fecha de consulta: 03 de junio de 2022]. Disponible en: https://www.pluralsight.com/guides/working-tables-github-markdown

Practical Business Python. Overview of Pandas Data Types [en línea] [fecha de consulta: 3 de junio de 2022]. Disponible en: https://pbpython.com/pandas_dtypes.html

Setting the Font, Title, Legend Entries, and Axis Titles in Python [en línea] [fecha de consulta: 3 de junio de 2022]. Disponible en: https://plotly.com/python/figure-labels/

Plotly Documentation. plotly.express.histogram [en línea] [fecha de consulta: 3 de junio de 2022]. Disponible en: https://plotly.github.io/plotly.py-docs/generated/plotly.express.histogram.html

Plotly Documentation. Normality Tests in Python/v3 [en línea] [fecha de consulta: 3 de junio de 2022]. Disponible en: https://plotly.com/python/v3/normality-test/

Stackoverflow. Correlation heatmap [en línea] [fecha de consulta: 3 de junio de 2022]. Disponible en: https://stackoverflow.com/questions/39409866/correlation-heatmap

Seaborn Documentation. seaborn.heatmap [en línea] [fecha de consulta: 3 de junio de 2022]. Disponible en: https://seaborn.pydata.org/generated/seaborn.heatmap.html

Box Plots in Python [en línea] [fecha de consulta: 6 de junio de 2022]. Disponible en: https://plotly.com/python/box-plots/

Subplots in Python [en línea] [fecha de consulta: 6 de junio de 2022]. Disponible en: https://plotly.com/python/subplots/

scipy.stats.levene [en línea] [fecha de consulta: 6 de junio de 2022]. Disponible en: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.levene.html