In [2]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\samil\anaconda3\lib\site-packages (2.0.3)
Requirement already satisfied: numpy>=1.20.3; python_version < "3.10" in c:\users\samil\anaconda3\lib\site-packages (from pandas) (1.24.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\samil\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\samil\anaconda3\lib\site-packages (from pandas) (2020.1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\samil\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\samil\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
pip install plotly
```

```
Requirement already satisfied: plotly in c:\users\samil\anaconda3\lib\site-packages (5.18.0)
Requirement already satisfied: packaging in c:\users\samil\anaconda3\lib\site-packages (from plotly) (20.4)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\samil\anaconda3\lib\site-packages (from plotly) (8.2.3)
Requirement already satisfied: six in c:\users\samil\anaconda3\lib\site-packages (from packaging->plotly) (1.15.0)
Note: you may need to restart the kernel to use updated packages.Requirement already satisfied: pyparsing>=2.0.2 in c:\users\samil\anaconda3\lib\site-packages (from pack
aging->plotly) (2.4.7)
```

In [4]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
C:\Users\Samil\anaconda3\lib\site-packages\pandas\core\computation\expressions.py:20: UserWarning: Pandas requires version '2.7.3' or newer of 'numexpr' (version '2.7.1'
currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
```

In [5]:
```
raw_dataset=pd.read_csv("Doublet_EAF_35F.csv",sep=",")
```

In [6]:
```
 Doublet_EAF_35F= raw_dataset.copy()
Doublet_EAF_35F.head()
```

Out[6]:

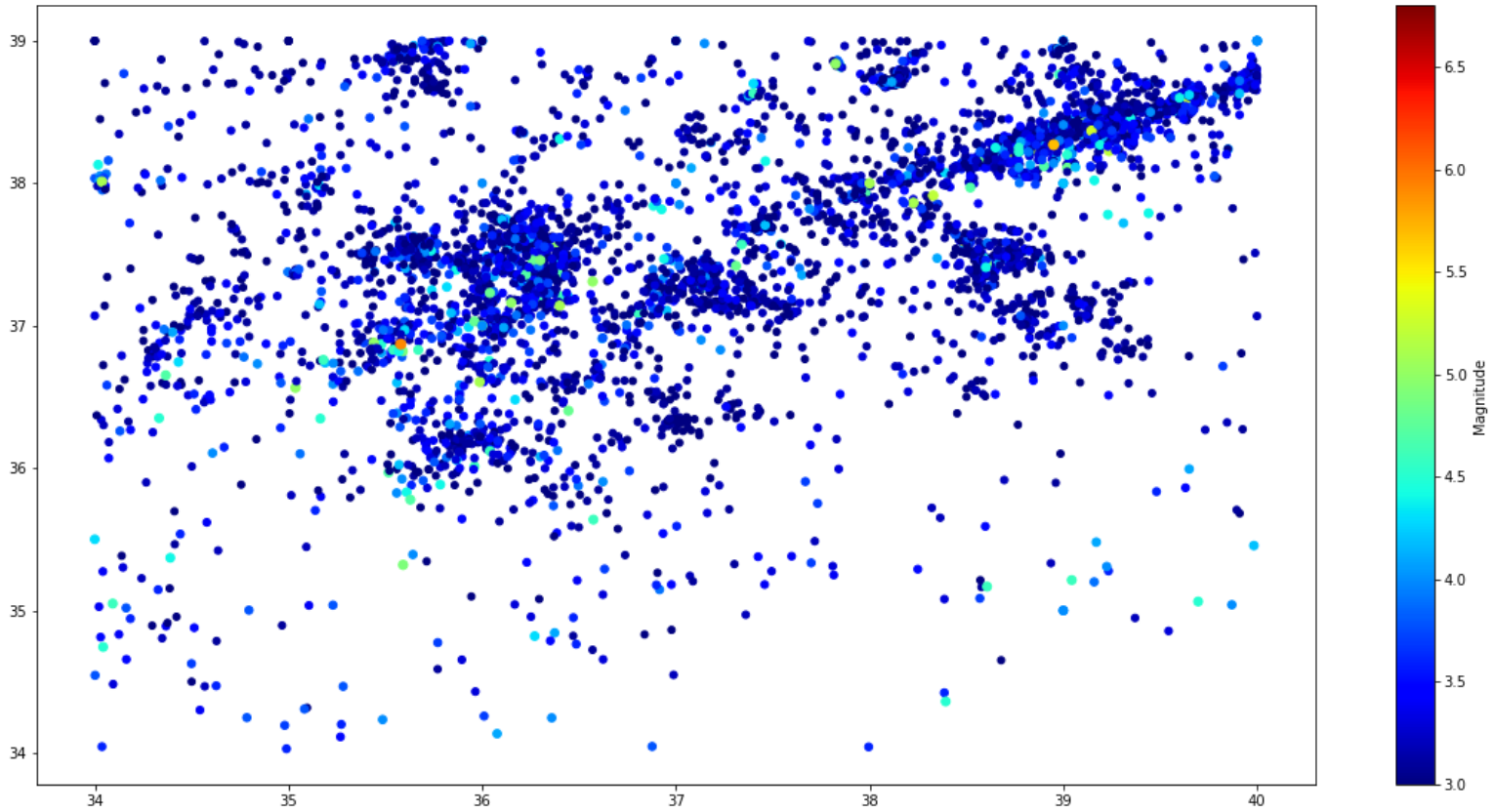
	Date	Longitude	Latitude	Depth	Magnitude
0	05/02/2023 06:06:55	34.162	35.018	30.50	3.8
1	03/02/2023 11:05:08	36.403	37.208	7.00	4.6
2	29/01/2023 16:12:39	35.784	35.884	18.35	4.4
3	22/01/2023 02:28:39	36.609	38.454	6.85	3.1
4	21/01/2023 14:27:54	36.374	34.843	18.05	4.1

In [26]:
```
Doublet_EAF_35F.shape
```

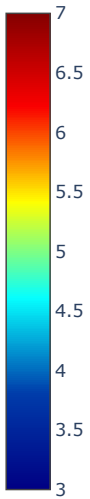
Out[26]:
```
(5414, 5)
```

In [27]:
```
x = Doublet_EAF_35F.iloc[:,1].values
y = Doublet_EAF_35F.iloc[:,2].values
z = Doublet_EAF_35F.iloc[:,3].values
colors = Doublet_EAF_35F.iloc[:,4].values
sizes = Doublet_EAF_35F.iloc[:,4].values*8
```

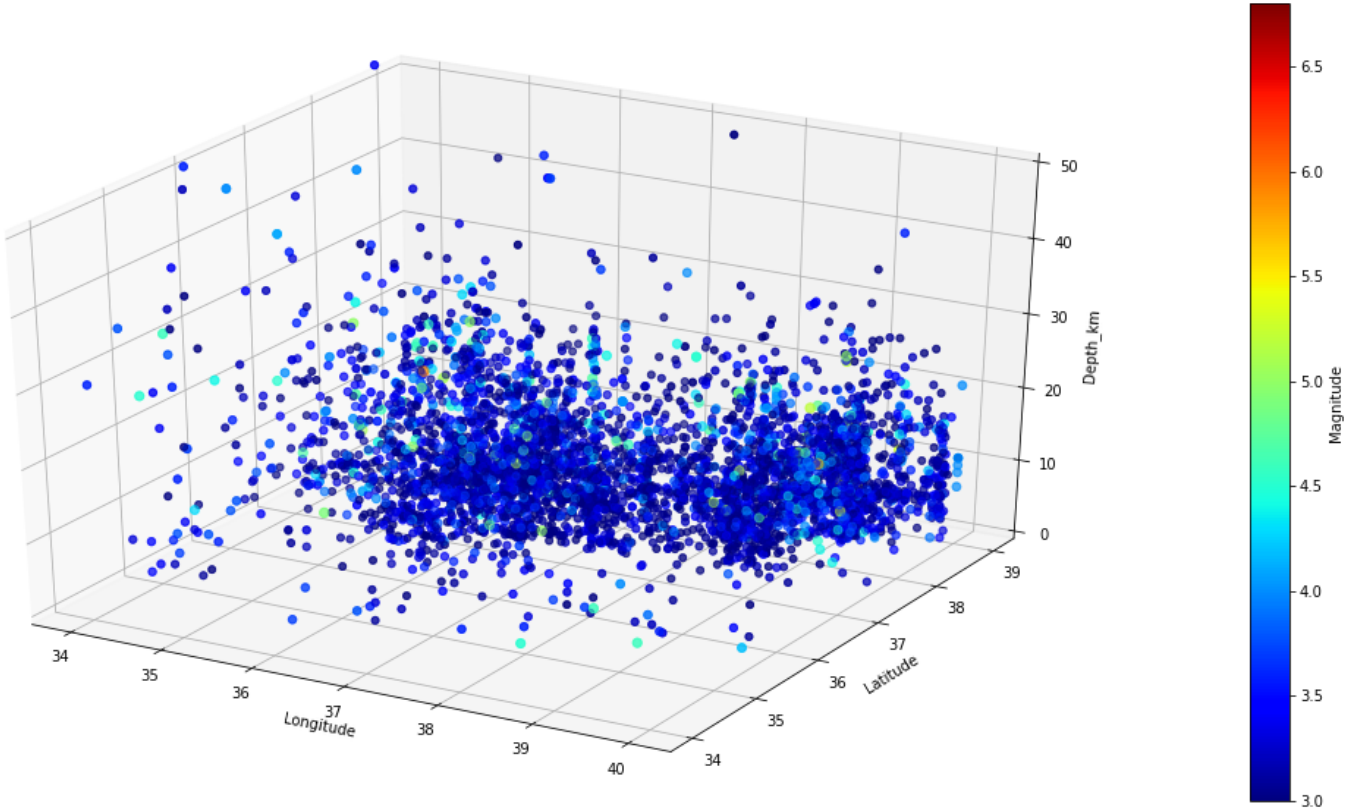
In [28]:
```
fig = plt.figure(figsize=(20, 10))
my_cmap = plt.get_cmap('jet')

plt.scatter(x, y, c=colors, s=sizes, cmap= 'jet')
ax.scatter(x, y, z, c=colors, s=sizes, cmap='jet')
cbar = plt.colorbar(ax.scatter(x, y, z, c=colors, s=sizes, cmap='jet'))

ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
cbar.set_label('Magnitude')
ax.set_zlabel('Depth_km')

font_size = 700
dpi = (5000)

ax.set_xlim(34, 39)

plt.show()
```



In [29]:
```
import plotly.graph_objects as go
# Yüksek çözünürlüklü dünya haritası verilerini çevrimiçi olarak alın
fig = go.Figure(go.Choroplethmapbox(
 geojson="https://raw.githubusercontent.com/johan/world.geo.json/master/countries.geo.json",
 locations=["USA", "CAN", "MEX", "RUS", "CHN"], # Örnek ülke kodları (ABD, Kanada, Meksika, Rusya, Çin)
 z=[1, 1, 1, 1, 1], # Ülkelere atanacak değerler (hepsi 1 olarak ayarlanmıştır)
 colorscale='Jet', # Renk skalası adı (Viridis, YlGnBu, Jet vb.)
 zmin=3,
 zmax=7,
 marker_opacity=0.9, # Ülke sınırlarının opaklığı
 marker_line_width=1, # Ülke sınırlarının kenarlık kalınlığı
))
# Örnek deprem verilerini oluşturun
deprem_verileri = {
 'Longitude': x,
 'Latitude': y,
 'Magnitude': colors,
 }
# Scatter plot ile deprem verilerini ekleyin
fig.add_trace(go.Scattermapbox(
 lat=deprem_verileri['Latitude'],
 lon=deprem_verileri['Longitude'],
 mode='markers',
 marker=dict(
 size=deprem_verileri['Magnitude'] * 2, # Magnitude değerine göre nokta boyutlarını belirleme
 color=deprem_verileri['Magnitude'], # Magnitude değerine göre renk skalasını belirleme
 colorscale='Jet', # Renk skalası adı (Viridis, YlGnBu, Jet vb.)
 ),
 ))
# Harita düzenini ve stilini belirleyin
fig.update_layout(
 mapbox_style="carto-positron", # Harita stilini belirleme (diğer stiller için: "open-street-map", "stamen-terrain" vb.)
 mapbox_zoom=6, # Harita yakınlaştırma düzeyini belirleme
 mapbox_center={"lat": 30.000, "lon": 30.0000}, # Harita merkezini belirleme (ABD'nin merkezi)
)
dpi = (9000)
font_size = 1000
# Grafiği görüntüleyin
fig.show()
```


7
6.5
6
5.5
5
4.5
4
3.5
3

In [12]:
```
fig = plt.figure(figsize=(20, 10))

ax = fig.add_subplot(111, projection='3d')

ax.scatter(x, y, z, c=colors, s=sizes, cmap='jet')


cbar = plt.colorbar(ax.scatter(x, y, z, c=colors, s=sizes, cmap='jet'))

cbar.set_label('Magnitude')


ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_zlabel('Depth_km')




dpi = (5000)
font_size = 1000

ax.set_zlim(0, 50)


plt.show()
```



In [13]:
```
x = Doublet_EAF_35F.iloc[:,1].values
y = Doublet_EAF_35F.iloc[:,2].values
z = Doublet_EAF_35F.iloc[:,3].values
colors = Doublet_EAF_35F.iloc[:,4].values
sizes = Doublet_EAF_35F.iloc[:,4].values*1
```

In [14]:
```
import plotly.graph_objs as go
import numpy as np

# Veri oluşturma (x, y, z, colors, sizes tanımlanmış olarak varsayıldı)

trace = go.Scatter3d(
    x=x,
    y=y,
    z=z,
    mode='markers',
    marker=dict(
        size=sizes,
        color=colors,
        colorscale='Jet',
        opacity=0.5,
        colorbar=dict(title='Magnitude')
    )
)

layout = go.Layout(
    scene=dict(
        xaxis=dict(title='Longitude'),
        yaxis=dict(title='Latitude'),
        zaxis=dict(title='Depth_km'),
        aspectmode='manual',
        aspectratio=dict(x=1, y=1, z=1),
        camera=dict(eye=dict(x=2, y=1, z=1))
    ),
    coloraxis=dict(colorbar=dict(len=0.75))
)

fig = go.Figure(data=[trace], layout=layout)

# Grafik döndürme
frames = []
for angle in np.linspace(0, 360, 36):
    frame = go.Frame(layout=dict(scene=dict(camera=dict(eye=dict(x=2*np.cos(np.radians(angle)), y=2*np.sin(np.radians(angle)), z=2)))))
    frames.append(frame)

fig.frames = frames
fig.update_layout(updatemenus=[dict(type='buttons', showactive=False, buttons=[dict(label='Play', method='animate', args=[None, dict(frame=dict(duration=200, redraw=True)

fig.show()
```

In [15]:
```
x = Doublet_EAF_35F.iloc[:,1].values
y = Doublet_EAF_35F.iloc[:,2].values
z = Doublet_EAF_35F.iloc[:,0].values
colors = Doublet_EAF_35F.iloc[:,4].values
sizes = Doublet_EAF_35F.iloc[:,4].values*8
```

In [16]:
```
z
```

Out[16]:
```
array(['05/02/2023 06:06:55', '03/02/2023 11:05:08',
       '29/01/2023 16:12:39', ..., '07/06/1990 21:02:16',
       '04/06/1990 09:14:20', '12/05/1990 21:50:39'], dtype=object)
```

In [17]:
```
from datetime import datetime

old_dates = z

old_format = '%d/%m/%Y %H:%M:%S'
new_format = '%Y-%m-%d %H:%M:%S'

new_dates = []

for date_str in old_dates:
    new_date = datetime.strptime(date_str, old_format).strftime(new_format)
    new_dates.append(new_date)

print(new_dates)
```

```
['2023-02-05 06:06:55', '2023-02-03 11:05:08', '2023-01-29 16:12:39', '2023-01-22 02:28:39', '2023-01-21 14:27:54', '2023-01-21 13:31:59', '2023-01-18 13:30:47', '2023-01-17 18:22:40', '2023-01-15 03:36:26', '2023-01-12 20:40:49', '2023-01-09 05:57:42', '2023-01-02 01:51:44', '2022-12-27 18:15:06', '2022-12-23 22:14:59', '2022-12-23 17:23:31', '2022-12-22 22:19:00', '2022-12-22 22:18:43', '2022-12-18 18:13:09', '2022-12-14 20:53:18', '2022-12-13 02:31:08', '2022-12-07 18:13:16', '2022-11-28 23:04:31', '2022-11-17 01:53:12', '2022-11-02 11:16:27', '2022-10-31 18:34:25', '2022-10-29 04:13:40', '2022-10-28 19:16:10', '2022-10-27 01:02:51', '2022-10-23 09:00:43', '2022-10-21 07:49:37', '2022-10-20 17:14:02', '2022-10-20 12:46:01', '2022-10-20 12:05:33', '2022-10-20 11:34:59', '2022-10-18 17:17:03', '2022-10-18 05:32:48', '2022-10-18 03:31:21', '2022-10-17 23:16:23', '2022-10-16 09:24:24', '2022-10-12 23:04:25', '2022-10-11 15:48:46', '2022-10-10 13:07:30', '2022-10-05 04:02:56', '2022-09-30 16:18:31', '2022-09-30 03:46:32', '2022-09-24 08:22:40', '2022-09-21 22:53:20', '2022-09-15 07:30:02', '2022-09-12 16:54:28', '2022-09-01 22:33:22', '2022-08-24 08:33:23', '2022-08-20 03:17:24', '2022-08-16 20:53:31', '2022-08-13 03:50:44', '2022-08-08 18:58:43', '2022-08-05 21:12:03', '2022-08-02 18:09:57', '2022-08-02 00:21:30', '2022-08-01 07:03:19', '2022-08-01 01:56:31', '2022-07-30 19:14:47', '2022-07-27 15:38:42', '2022-07-26 06:10:59', '2022-07-22 20:38:01', '2022-07-22 01:21:06', '2022-07-22 00:14:52', '2022-07-19 18:01:42', '2022-07-19 06:52:19', '2022-07-19 06:41:53', '2022-07-19 02:18:54', '2022-07-15 12:13:46', '2022-07-13 18:15:51', '2022-07-13 14:56:41', '2022-07-11 09:23:47', '2022-07-06 01:37:06', '2022-07-04 15:22:13', '2022-06-23 11:10:24', '2022-06-11 18:53:35', '2022-06-10 01:40:21', '2022-06-09 12:14:33', '2022-06-09 07:22:18', '2022-06-05 17:22:00', '2022-05-30 18:07:18', '2022-05-30 14:32:59', '2022-05-23 18:43:22', '2022-05-15 03:32:48', '2022-05-15 03:12:08', '2022-05-13 16:23:02', '2022-05-01 09:02:12', '2022-04-22 16:04:19', '2022-04-19 21:42:42', '2022-04-19 07:17:34', '2022-04-15 18:54:50', '2022-04-15 05:22:25', '2022-04-14 10:55:05', '2022-04-10 15:49:43', '2022-04-10 02:08:10', '2022-04-09 20:08:04', '2022-04-09 20:06:00', '2022-04-09 15:39:00', '2022-04-09 15:37:46', '2022-04-09 15:37:07', '2022-04-09 14:22:10', '2022-04-09 14:02:14', '2022-04-08 16:17:23', '2022-04-07 22:09:28', '2022-04-02 06:46:34', '2022-04-01 05:34:02', '2022-03-27 07:26:56', '2022-03-26 21:48:27', '2022-03-14 12:38:18', '2022-03-11 00:04:36', '2022-03-10 23:53:20', '2022-03-10 10:12:58', '2022-03-05 06:33:32', '2022-03-04 07:58:57', '2022-03-02 09:42:11', '2022-02-28 15:30:08', '2022-02-20 06:26:23', '2022-02-14 13:47:37', '2022-02-08 15:38:05', '2022-02-04 21:28:41', '2022-01-31 11:44:47', '2022-01-31 01:00:05', '2022-01-27 13:39:36', '2022-01-24 19:46:43', '2022-01-21 16:39:35', '2022-01-19 05:59:06', '2022-01-19 05:58:25', '2022-01-16 18:17:49', '2022-01-12 13:39:59', '2022-01-11 21:34:43', '2021-12-23 23:21:21', '2021-12-14 13:08:14', '2021-12-13 17:10:33', '2021-12-13 16:58:51', '2021-12-12 21:23:5
```

In [18]:
```
from datetime import datetime

date_times = new_dates

single_numbers = []

for date_time_str in date_times:
    date_time_obj = datetime.strptime(date_time_str, '%Y-%m-%d %H:%M:%S')
    single_number = int(date_time_obj.strftime('%Y%m%d%H%M%S'))
    single_numbers.append(single_number)

print(single_numbers)
```

```
[20230205060655, 20230203110508, 20230129161239, 20230122022839, 20230121142754, 20230121133159, 20230118133047, 20230117182240, 20230115033626, 20230112204049, 20230
109055742, 20230102015144, 20221227181506, 20221223221459, 20221223172331, 20221222221900, 20221222221843, 20221218181309, 20221214205318, 20221213023108, 20221207181
316, 20221128230431, 20221117015312, 20221102111627, 20221031183425, 20221029041340, 20221028191610, 20221027010251, 20221023090043, 20221021074937, 20221020171402, 2
0221020124601, 20221020120533, 20221020113459, 20221018171703, 20221018053248, 20221018033121, 20221017231623, 20221016092424, 20221012230425, 20221011154846, 2022101
0130730, 20221005040256, 20220930161831, 20220930034632, 20220924082240, 20220921225320, 20220915073002, 20220912165428, 20220901223322, 20220824083323, 2022082003172
4, 20220816205331, 20220813035044, 20220808185843, 20220805211203, 20220802180957, 20220802002130, 20220801070319, 20220801015631, 20220730191447, 20220727153842, 202
20726061059, 20220722203801, 20220722012106, 20220722001452, 20220719180142, 20220719065219, 20220719064153, 20220719021854, 20220715121346, 20220713181551, 202207131
45641, 20220711092347, 20220706013706, 20220704152213, 20220623111024, 20220611185335, 20220610014021, 20220609121433, 20220609072218, 20220605172200, 20220530180718,
20220530143259, 20220523184322, 20220515033248, 20220515031208, 20220513162302, 20220501090212, 20220422160419, 20220419214242, 20220419071734, 20220415185450, 202204
15052225, 20220414105505, 20220410154943, 20220410020810, 20220409200804, 20220409200600, 20220409153900, 20220409153746, 20220409153707, 20220409142210, 202204091402
14, 20220408161723, 20220407220928, 20220402064634, 20220401053402, 20220327072656, 20220326214827, 20220314123818, 20220311000436, 20220310235320, 20220310101258, 20
220305063332, 20220304075857, 20220302094211, 20220228153008, 20220220062623, 20220214134737, 20220208153805, 20220204212841, 20220131114447, 20220131010005, 20220127
133936, 20220124194643, 20220121163935, 20220119055906, 20220119055825, 20220116181749, 20220112133959, 20220111213443, 20211223232121, 20211214130814, 2021121317103
3, 20211213165851, 20211212212356, 20211209032809, 20211130025419, 20211129200413, 20211128230433, 20211121113711, 20211112091913, 20211111071749, 20211109061323, 202
11109060634, 20211105221246, 20211101191231, 20211028161223, 20211017182251, 20211013231106, 20211011093931, 20211009205350, 20211009165846, 20211004221602, 202109212
15333, 20210921152432, 20210921073152, 20210916150742, 20210909235814, 20210909014240, 20210906145041, 20210826205255, 20210806075705, 20210804123815, 20210804065557,
20210725205505, 20210725175729, 20210719074542, 20210719073941, 20210719073250, 20210718220926, 20210717020315, 20210716084405, 20210715080104, 20210709084112, 202107
08211011, 20210704064341, 20210629034155, 20210621065450, 20210620135634, 20210603042644, 20210602061427, 20210530161333, 20210527085545, 20210524134947, 202105231654
23, 20210523120054, 20210521160657, 20210520210938, 20210515080330, 20210514101409, 20210511023743, 20210508003045, 20210507201837, 20210430045811, 20210423022845, 20
```

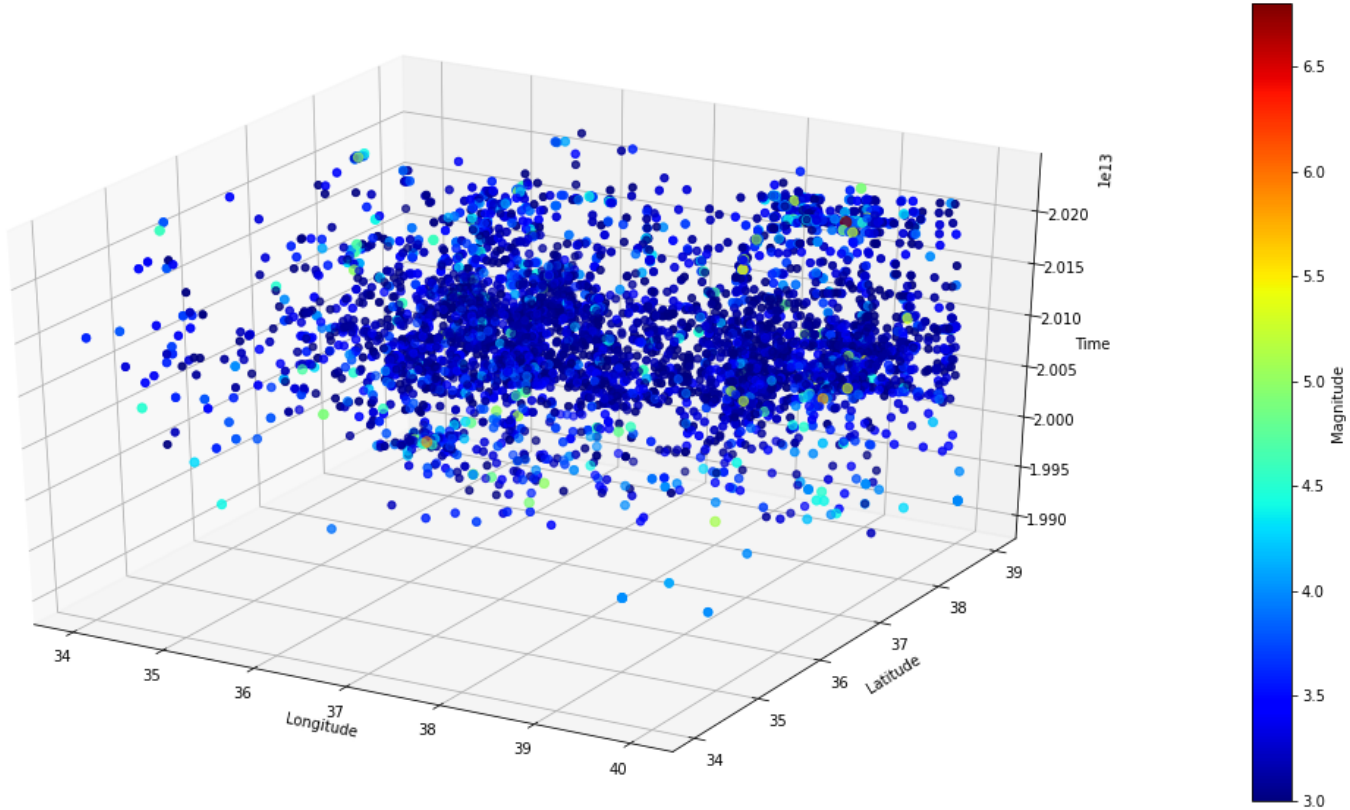
In [19]:
```
timestamps=single_numbers
```

In [20]:
```
timestamps
```

Out[20]:
```
[20230205060655,
 20230203110508,
 20230129161239,
 20230122022839,
 20230121142754,
 20230121133159,
 20230118133047,
 20230117182240,
 20230115033626,
 20230112204049,
 20230109055742,
 20230102015144,
 20221227181506,
 20221223221459,
 20221223172331,
 20221222221900,
 20221222221843,
 20221218181309,
 20221214205318,
```

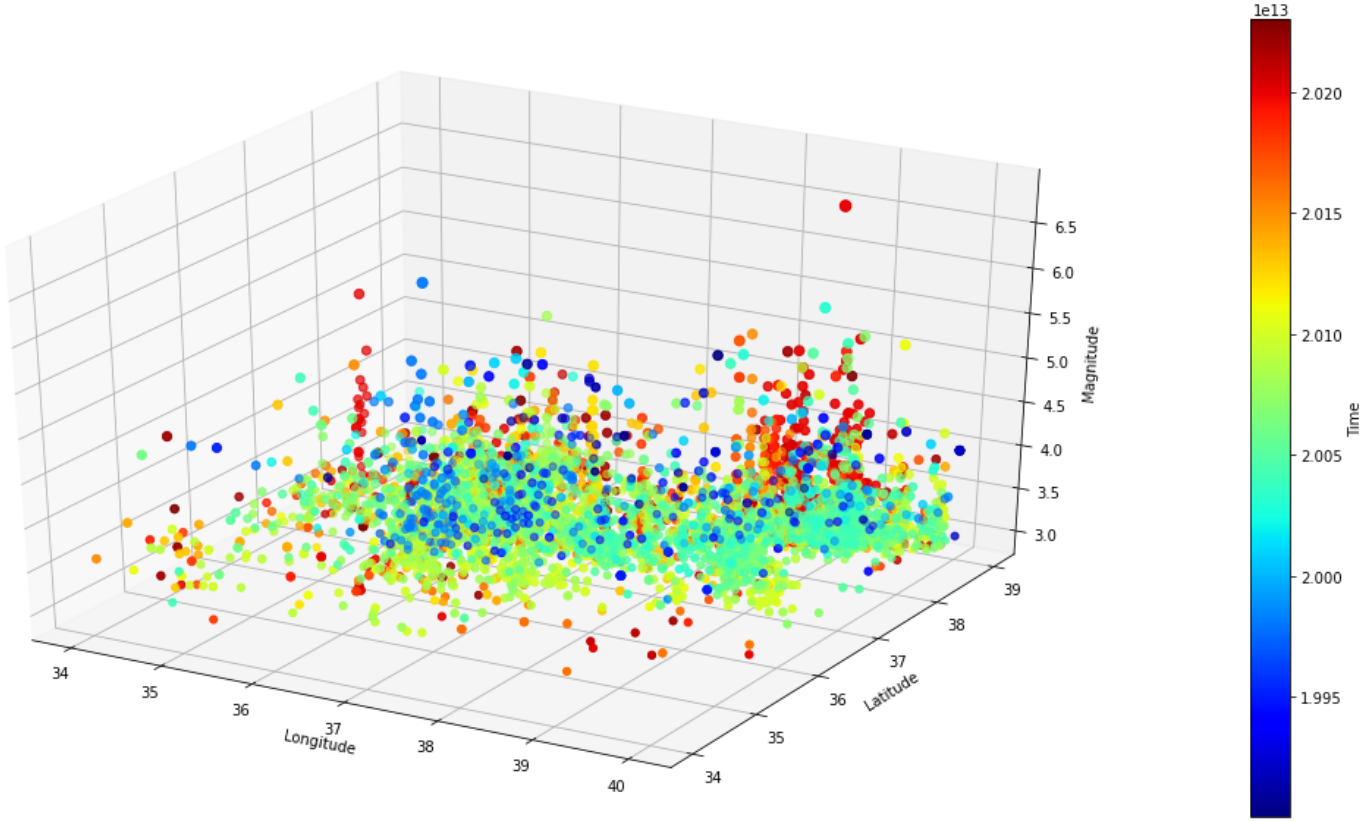
In [21]:
```
x = Doublet_EAF_35F.iloc[:,1].values
y = Doublet_EAF_35F.iloc[:,2].values
z = timestamps
colors = Doublet_EAF_35F.iloc[:,4].values
sizes = Doublet_EAF_35F.iloc[:,4].values*8
```

In [22]:
```
fig = plt.figure(figsize=(20, 10))

ax = fig.add_subplot(111, projection='3d')

ax.scatter(x, y, z, c=colors, s=sizes, cmap='jet')


cbar = plt.colorbar(ax.scatter(x, y, z, c=colors, s=sizes, cmap='jet'))

cbar.set_label('Magnitude')


ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_zlabel('Time')

font_size = 700

dpi = (5000)
font_size = 1000




plt.show()
```



In [23]:
```
x = Doublet_EAF_35F.iloc[:,1].values
y = Doublet_EAF_35F.iloc[:,2].values
z = Doublet_EAF_35F.iloc[:,4].values
colors = timestamps
sizes = Doublet_EAF_35F.iloc[:,4].values*8
```

In [24]:
```
fig = plt.figure(figsize=(20, 10))

ax = fig.add_subplot(111, projection='3d')

ax.scatter(x, y, z, c=colors, s=sizes, cmap='jet')


cbar = plt.colorbar(ax.scatter(x, y, z, c=colors, s=sizes, cmap='jet'))

cbar.set_label('Time')


ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_zlabel('Magnitude')

font_size = 700

dpi = (5000)
font_size = 1000




plt.show()
```


Longitude
34
35
36
37
38
39
40
Latitude
34
35
36
37
38
39
Magnitude
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
1e13
2.020
2.015
2.010
2.005
2.000
1.995
Time