Author: Perikleous Michalis (985936)

What can we learn from the visualization?

This chart shows the count of locations of **top seven** `period` that occur much more frequently in the dataset. It also shows the count of locations of **top seven** `type` of locations the occur much more frequently in the dataset, for each of these `period` .

What is the name for the type of visualization(s) used?

For these visualizations we used **bar charts**(histograms)

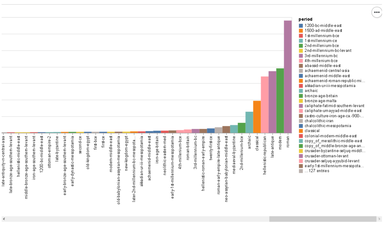
Data Preparation

```
import altair as alt
import vega_datasets as data
import pandas as pd
import numpy as np

locations = pd.read_csv('pleiades-locations.csv')

alt.data_transformers.disable_max_rows()
#Data preparation: Match each period with the type based on order they are inserted

#Remove locations that with missing information, either from timePeriodsKeys or featureType
locations['timePeriodsKeys'] = locations['timePeriodsKeys'].str.replace(' ','')
locations['featureType'] = locations['featureType'].str.replace('-2','')
locations.dropna(subset=['timePeriodsKeys'],inplace=True)
locations = locations[locations.featureType != 'unknown']
locations = locations[locations.featureType != 'unknown,']

#Split timePeriodsKeys column to several columns separating the periods,
#and keep the first 4 periods of the location
periodIndex = [locations.id,locations.featureType]
period_df = pd.DataFrame(pd.DataFrame(locations.timePeriodsKeys.str.split(',').tolist(),
                                      index=periodIndex)).dropna(thresh=800,axis='columns')
#Reformat column indexes
period_df.reset_index(inplace=True)
period_df.columns = ['id', 'featureType','period1','period2','period3','period4']

#Further removing of missing information
period_df = period_df[period_df.featureType != '']
period_df = period_df.dropna(subset=['featureType'])

#Split featureType column to several columns separating type,
#and keep first 4 types of the location
period_feature_index = [period_df.id,period_df.period1,period_df.period2,
                        period_df.period3,period_df.period4]
period_feature_df = pd.DataFrame(pd.DataFrame(period_df.featureType.str.split(',').tolist(),
                                              index=period_feature_index).dropna(thresh=10,axis='columns'))

#Reformat column indexes
period_feature_df.reset_index(inplace=True)
period_feature_df.columns = ['id','period1','period2','period3','period4',
                             'type1','type2','type3','type4']


#Create a numpy array from the dataframe(general bad practise but here is usefvl)
arr = np.array(period_feature_df,dtype=str)
for i in range(arr.shape[0]):
    for j in range(5,9):
        if arr[i][j] in ["","None","nan"]:
            arr[i][j] = "nan"
            for k in range(j,9):
                if(arr[i][k-4] != "nan"):
                    arr[i][k] = arr[i][j-1]

#Create array list to create combination of period and type columns
arr_list = []
for i in range(arr.shape[0]):
    for j in range(1,5):
        if arr[i][j] != "nan":
            arr_list.append([arr[i][0],arr[i][j],arr[i][j+4]])

#Create new dataframe from our array with correct split of period and type for each period
#(based on comma seperate ordering)
final_df = pd.DataFrame(arr_list)
final_df.columns = ['id','period','type']

#Find the 7 periods that occur more often in the dataset
top7Periods = final_df['period'].value_counts().keys()[0:7]

#Find the 7 types that occur more often in the dataset
top7Types = final_df['type'].value_counts().keys()[0:7]

#Filter the dataset based on the top 7 periods and types
top7_df = final_df[final_df['period'].isin(top7Periods)]
top7_df = top7_df[top7_df['type'].isin(top7Types)]
```

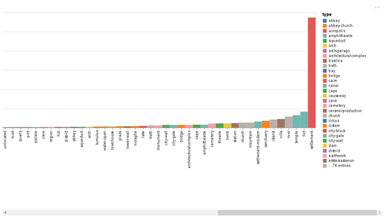
Use of all periods and types

In the following bar charts we created two histograms. The first one using all distinct `period` values we have in the dataset, and the second one using all distinct `type` values from the dataset. To produce our visualizations we used `final_df` dataframe that is the dataset after data preprocessing phase.

```
#All periods histogram
period_hist = alt.Chart(final_df).mark_bar(size=20).encode(
    x = alt.X("period:N",title='Period',sort ='y'),
    y = alt.Y('count(period):Q'),
    color='period:N',
    tooltip=['count()']
).properties(
    title="Period Histogram")
period_hist
```



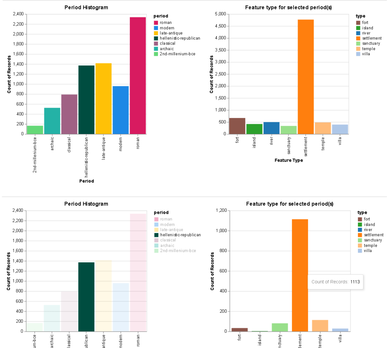

```
#All types histogram
type_hist = alt.Chart(final_df).mark_bar(size=20).encode(
    x = alt.X('type:N',title='Feature Type',sort ='y'),
    y = alt.Y('count(type):Q'),
    color=alt.Color('type:N'),
    tooltip=['count()']
).properties(
    title="Feature type for selected period(s)"
)
type_hist
```

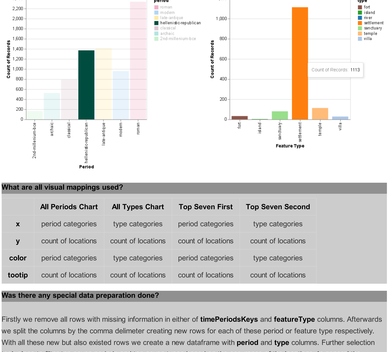


Use of top seven categories

To produce our visualizations we used `top7_df` dataframe that is the dataset that contains only the top seven `period` and `type` .

```
selection = alt.selection_multi(fields=['period'], bind='legend')

#period histogram
period_hist = alt.Chart(top7_df).mark_bar(size=40).encode(
    x = alt.X("period:N",title='Period'),
    y = alt.Y('count(period):Q'),
    color=alt.Color('period:N',scale=alt.Scale(
        domain=['roman', 'modern', 'late-antique', 'hellenistic-republican','classical', 'archaic', '2nd-millenium-bc
        range=['#DB1B0D', '#11EBD5', '#FFC107', '#0004D4D', '#4AD61B4', '#24B2A6', '#65D877'])),
    opacity=alt.condition(selection, alt.OpacityValue(1), alt.OpacityValue(0.1)),
    tooltip=['count()']
).properties(
    title="Period Histogram",
    width=300).add_selection(
selection)

#type histogram
type_hist = alt.Chart(top7_df).mark_bar(size=40).encode(
    x = alt.X('type:N',title='Feature Type'),
    y = alt.Y('count(type):Q'),
    color=alt.Color('type:N',scale=alt.Scale(
        domain=['fort', 'island', 'river', 'settlement','sanctuary', 'temple', 'villa'],
        range=['#8c564b' , '#2ca02c' ,'#1f77b4', '#ff7f0e', '#9bbf8a' , '#aec7e8']),
    tooltip=['count()']
).properties(
    title="Feature type for selected period(s)",
     width=300,
).transform_filter(
selection).interactive())

alt.hconcat(period_hist, type_hist).resolve_scale(
    color='independent'
)
```





What are all visual mappings used?

	All Periods Chart	All Types Chart	Top Seven First	Top Seven Second
x	period categories	type categories	period categories	type categories
y	count of locations	count of locations	count of locations	count of locations
color	period categories	type categories	period categories	type categories
tootip	count of locations	count of locations	count of locations	count of locations

Was there any special data preparation done?

Firstly we remove all rows with missing information in either of **timePeriodsKeys** and **featureType** columns. Afterwards we split the columns by the comma delimeter creating new rows for each of these period or feature type respectively. With all these new but also existed rows we create a new dataframe with **period** and **type** columns. Further selection took place to filter top seven periods and top seven types based on the occurance of the locations. In general the preprocessing was done to deal with the above two columns that have comma separated values. For the final visualizations we aggregate periods and types by category to count locations.

What are the limitations of your design?

One location can map to many types and periods. With the above charts we are not showing this relation between a location, periods and types. Someone can assume that the sum of count of locations of each period results to a number of distinct locations which is not the case. Additional bar charts to show how many locations occurs more that once would minimize this confusion.

What can we learn from the visualization?

This visualization shows the **connections** between pleiades' places around Mediterranean Sea. It can also focus on a specific structure and find its relationships

What is the name for the type of visualization(s) used?

For these visualizations we used **maps**

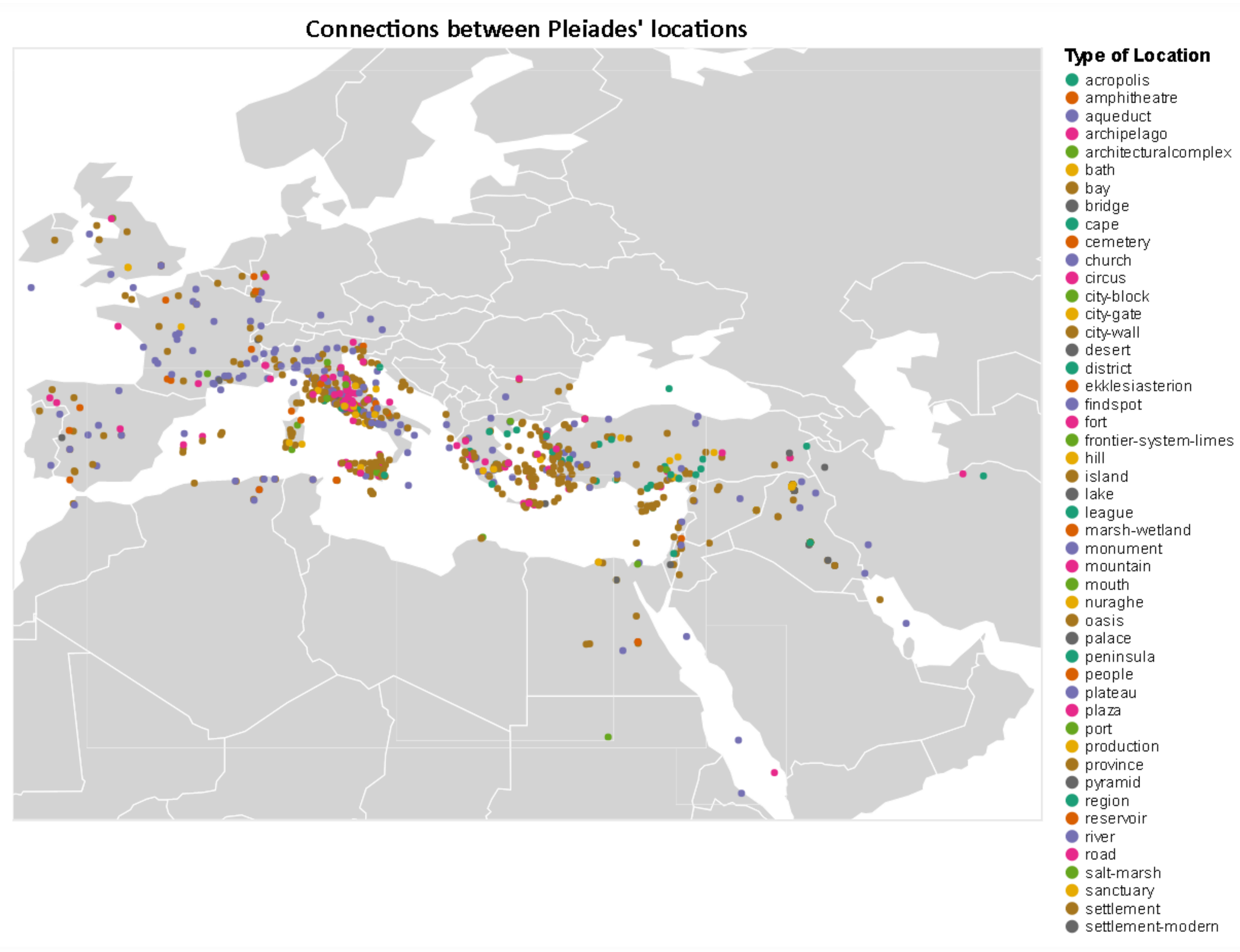
Data Preparation

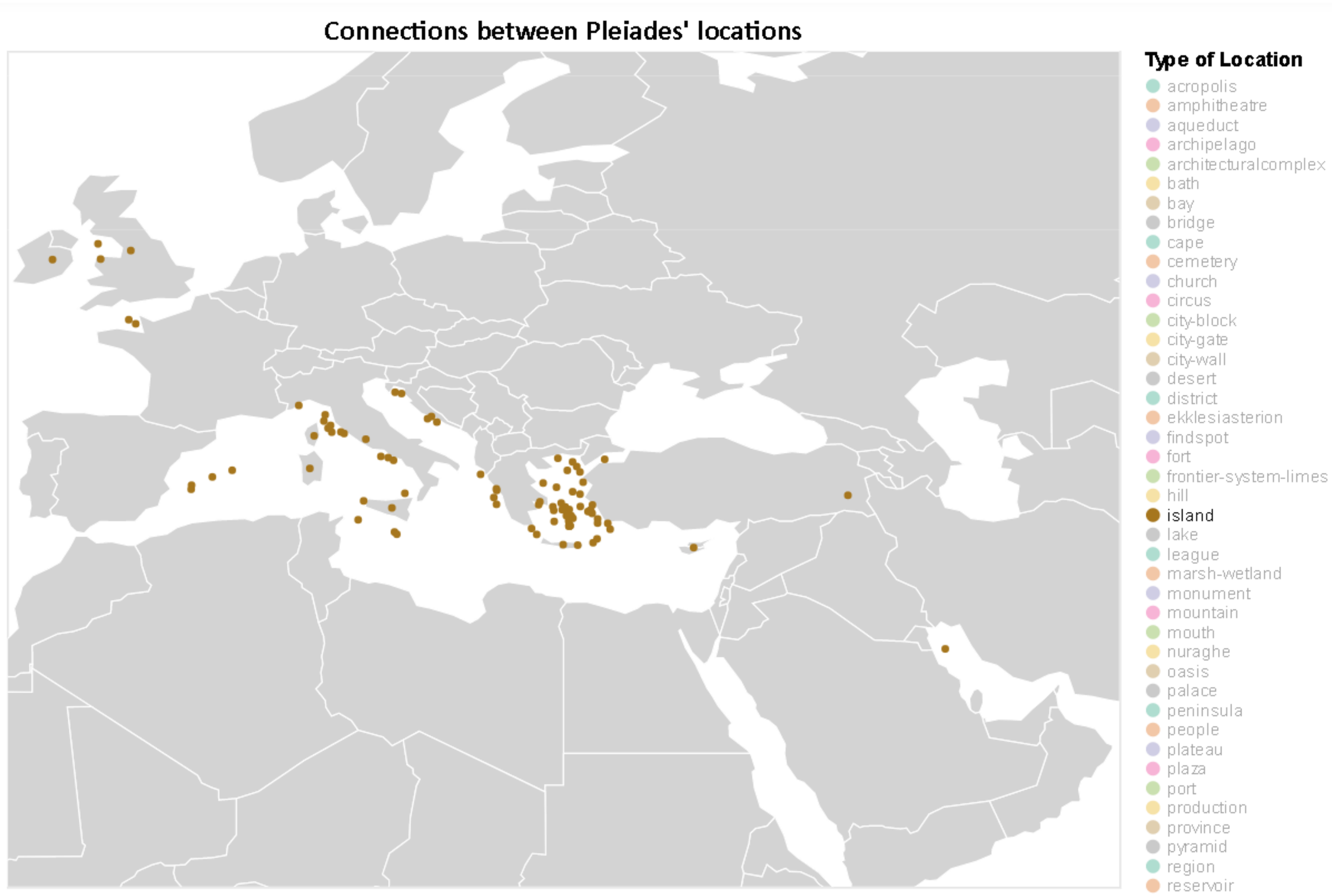
```
import altair as alt
from vega_datasets import data
import pandas as pd
import numpy as np

places = pd.read_csv('pleiades-places.csv')

alt.data_transformers.disable_max_rows()

#Choose only the columns we need and remove missing or wrong inputs
places = places[['reprLong','reprLat','connectsWith','hasConnectionsWith','featureTypes','path']]
places.dropna(subset=['reprLong','reprLat','connectsWith','hasConnectionsWith','featureTypes','path'],inplace=True)
places['path'] = places['path'].str.replace('/places/','')
places['initialType'] = places['featureTypes'].str.split(',').str[0]
places['initialType'] = places['initialType'].str.replace('-2','')
places = places[places.featureTypes != 'unknown']
places = places[places.featureTypes != 'unknown,']
places.drop(columns=['featureTypes'],inplace=True)
places['connectsWith'] = places['connectsWith'].str.replace(' ','')
places['hasConnectionsWith'] = places['hasConnectionsWith'].str.replace(' ','')
places.reset_index(inplace=True)
places.drop(columns=['index'],inplace=True)
places.drop_duplicates(subset=['reprLong', 'reprLat'], inplace=True)

#Create a numpy array from the dataframe(general bad practise but here is uselful)
#From the numpy array we create a list of connections between the places
arr = np.array(places,dtype=str)
con_list = []
for i in range(arr.shape[0]):
    connections = arr[i][2].split(',')
    x = arr[i][3].split(',')
    for con in x:
        connections.append(con)
    for con in connections:
        for k in range(arr.shape[0]):
            if con == arr[k][4]:
                con_list.append([arr[i][4],arr[k][4]])
                break

#convert the connection list into dataframe
connections = pd.DataFrame(con_list)
connections.columns=['origin','destination']
connections.sort_values(by=['origin'],inplace=True)
connections.reset_index(inplace=True)
connections.drop(columns=['index'],inplace=True)
```

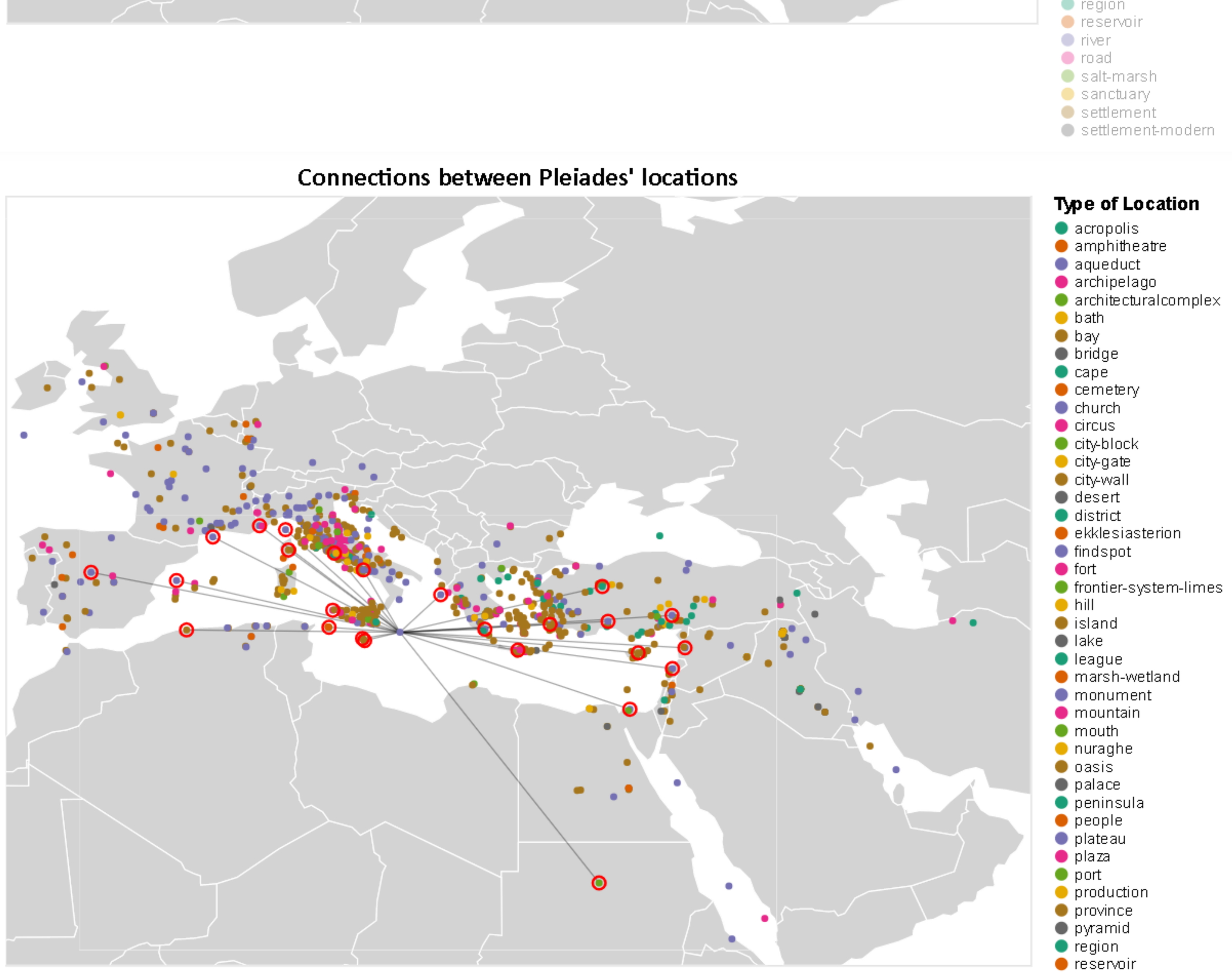
Connections of Pleiades's places

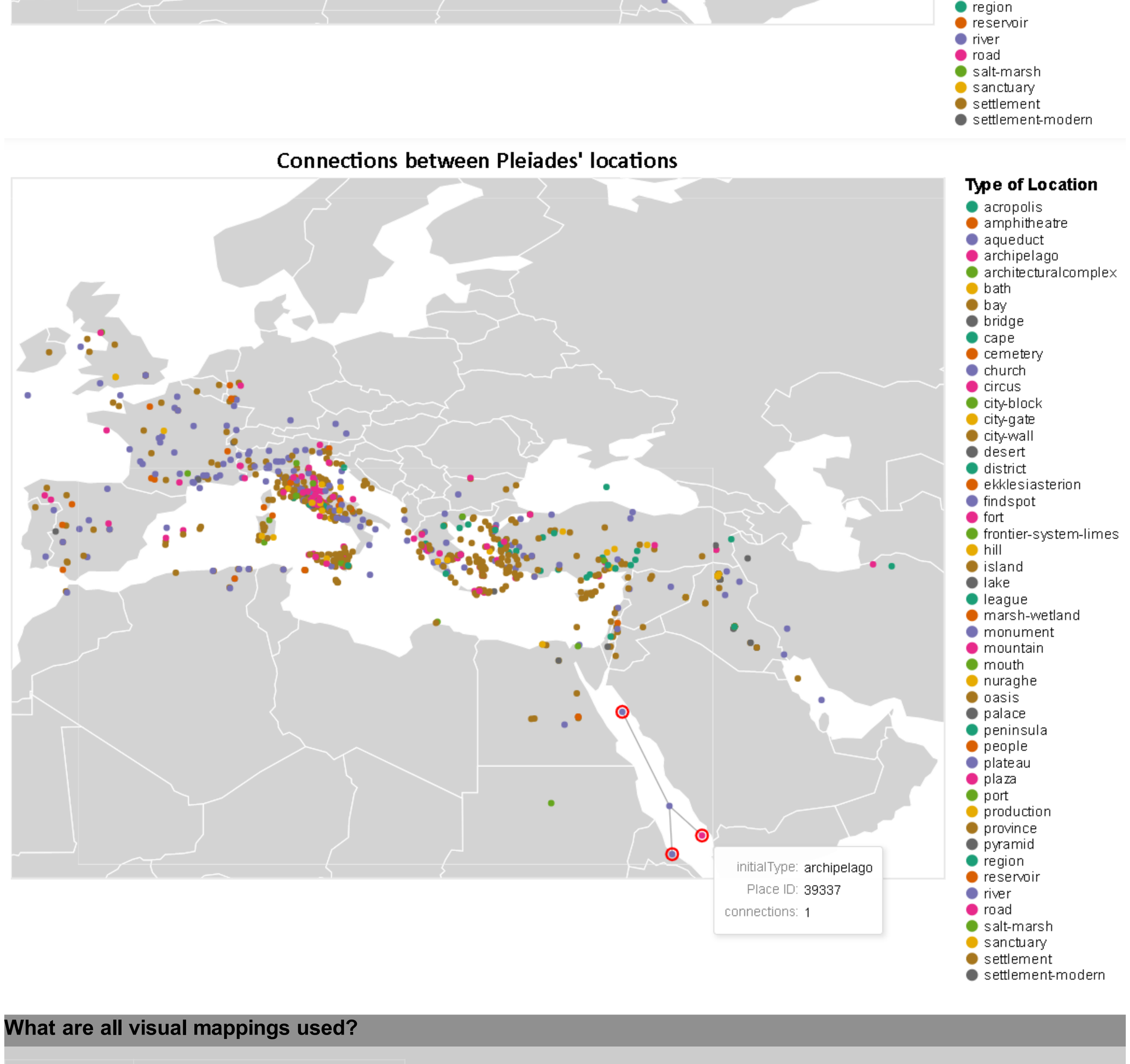
In the following visualization we can select specific type of locations to appear on map. By **clicking** on a location we can see the connections that a specific place has with other places. We can also see how many connections a place has.

```
# Create on click selection
select_city = alt.selection_single(
    on="click", fields=["origin"], empty="none"
)

#legend selection for types
selection = alt.selection_single(fields=['initialType'], bind="legend")

#world map from vega_datasets
world_map = alt.topo_feature(data.world_110m.url, 'countries')

# Define which attributes to lookup from places
lookup_data = alt.LookupData(
    places, key="path", fields=["path", "reprLat", "reprLong","initialType"]
)

#world map background from topojson data
background = alt.Chart(world_map).mark_geoshape(
    fill="lightgray",
    stroke="white"
).properties(
    width=800,
    height=600
)

#create the connections between places
edges = alt.Chart(connections).mark_rule(opacity=0.35).encode(
    latitude="reprLat:Q",
    longitude="reprLong:Q",
    latitude2="lat2:Q",
    longitude2="long2:Q"
).transform_lookup(
    lookup="origin",
    from_=lookup_data
).transform_lookup(
    lookup="destination",
    from_=lookup_data,
    as_=["path", "lat2", "long2", "type"]
).transform_filter(
    select_city
)


#point for each location of the dataset colored with its type
points = alt.Chart(connections).mark_circle().encode(
    latitude="reprLat:Q",
    longitude="reprLong:Q",
    color=alt.Color("initialType:N",scale=alt.Scale(scheme="dark2"),
                    legend=alt.Legend(symbolLimit=0),title="Type of Location"),
    opacity=alt.condition(selection, alt.OpacityValue(1), alt.OpacityValue(0)),
    tooltip=["initialType:N",alt.Tooltip("origin:N",title="Place ID"), "connections:Q"]
).transform_aggregate(
    connections="count()",
    groupby=["origin"]
).transform_lookup(
    lookup="origin",
    from_=lookup_data,
    as_=["path", "reprLat", "reprLong", "initialType"]
).add_selection(
    select_city,
    selection
)

#highlight all connections for specific place
selected_points = alt.Chart(connections).mark_point(size=100).encode(
    latitude="lat2:Q",
    longitude="long2:Q",
    color=alt.value("red"),
    opacity=alt.value(1)
).transform_lookup(
    lookup="origin",
    from_=lookup_data
).transform_lookup(
    lookup="destination",
    from_=lookup_data,
    as_=["path", "lat2", "long2", "type"]
).transform_filter(
    select_city
)

final_map = alt.layer(background + edges + points + selected_points).properties(
    title="Connections between Pleiades' locations"
).configure_legend(
    titleFont='Arial',
    titleFontSize=14,
    labelFont='Arial',
    labelFontSize = 12
).configure_title(
    fontSize=20,
    font='Calibri',
    anchor='middle',
    color='black'
).project(
    type= 'equirectangular',
    scale= 650,
    center= [25,40],
    clipExtent= [[0, 0], [800, 600]])

final_map
```









What are all visual mappings used?

	Mediterranean Sea Map
longitude	longitude of a place
latitude	latitude of a place
color	initial type of a place
tootip1	initial type of a place
tootip2	Place ID
tootip3	number of connections

Was there any special data preparation done?

I firstly extract from the dataset only the columns that I will use. Using a `connectsWith` and `hasConnectionsWith` I created a new dataframe containing all those connections between distinct places in separate rows. Grouping by `origin` (Place ID) in `connections` dataframe we created the **number of connections** that each place has.

What are the limitations of your design?

One limitation of this visualization is the fact that the map is not scalable, therefore cannot zoom or navigate through the map. Also the high density of places around Italy and Greece does not allow us to clearly distinguish the connections of some places. A scalable map could solve both of this problems.

What can we learn from the visualization?

This visualization shows the distribution of pleiades' locations based on their initial structure type around **Mediterranean Sea**, through the years. In this visualization you can also see the middle year, lifetime but also the last known structure type of the location.

What is the name for the type of visualization(s) used?

For these visualizations we used **maps**

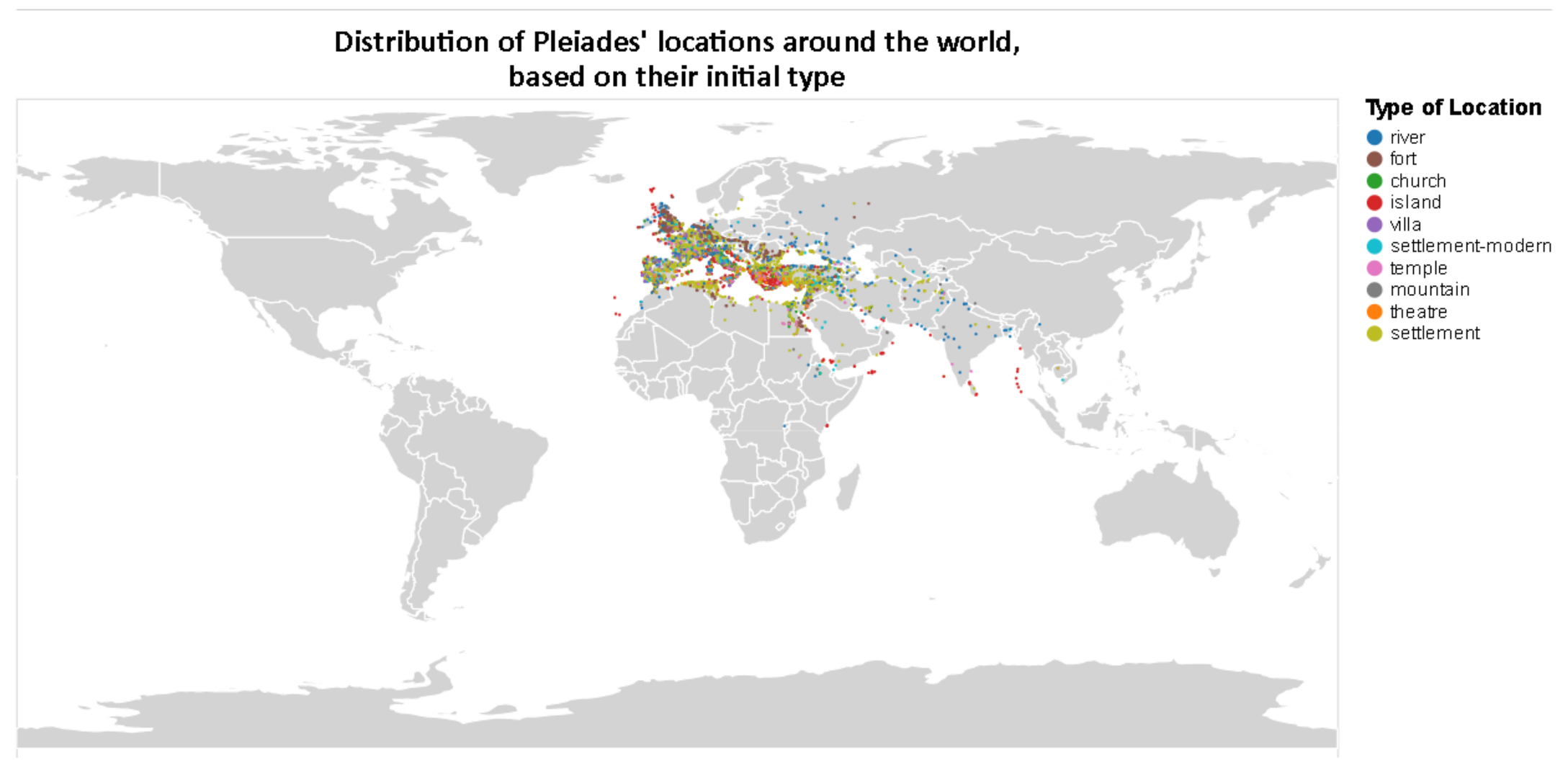
Data Preparation

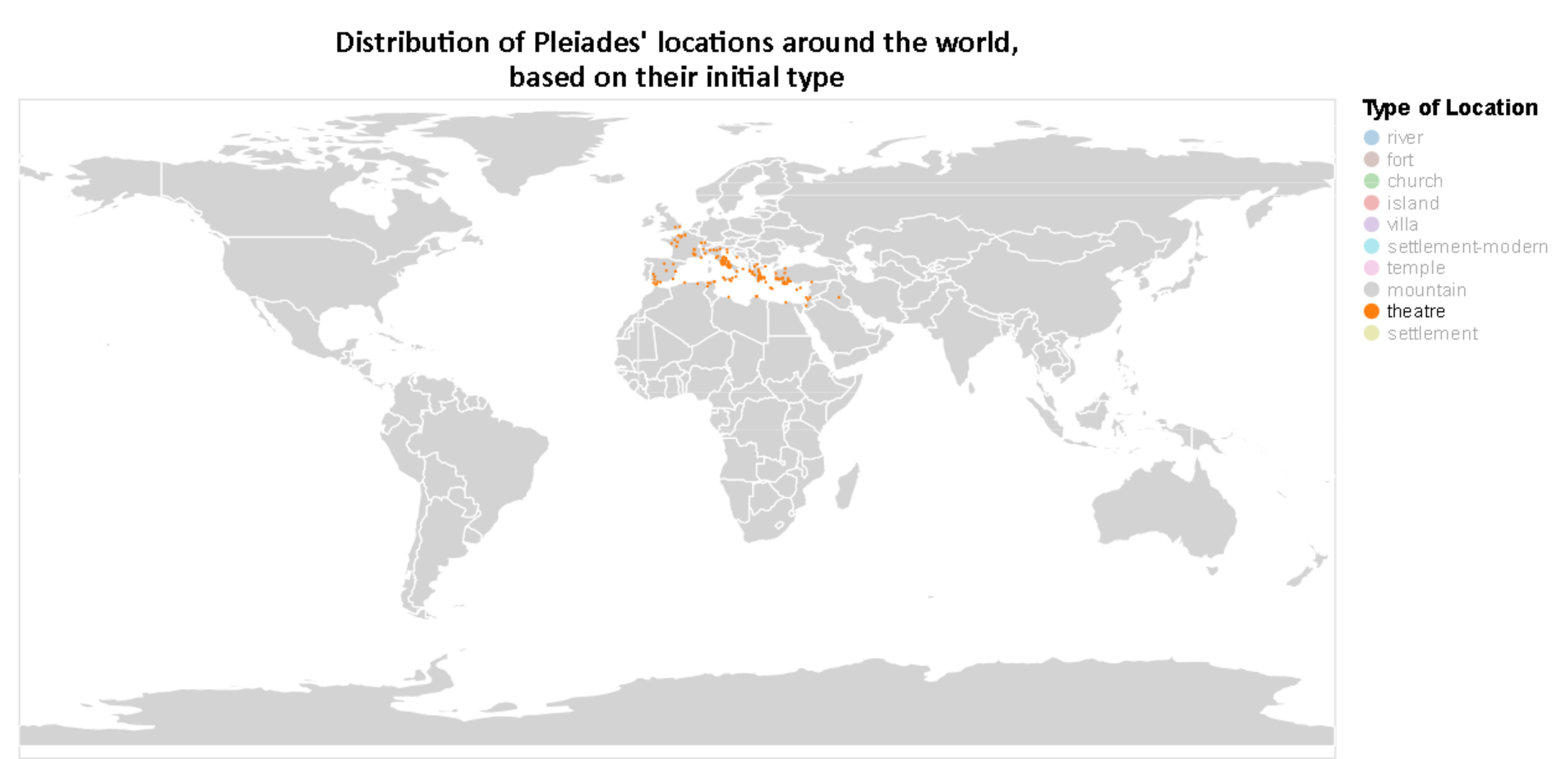
```python
import altair as alt
from vega_datasets import data
import pandas as pd
import numpy as np
from altair import datum

#Load the dataset
locations = pd.read_csv('pleiades-locations.csv')

alt.data_transformers.disable_max_rows()
pd.options.mode.chained_assignment = None

#Extract only the columns that we will use
locations = locations[['maxDate','minDate','reprLat','reprLong','featureType']]

#Remove rows with missing data and fix them properly
locations['featureType'] = locations['featureType'].str.replace('-2','')
locations = locations[locations.featureType != 'unknown']
locations = locations[locations.featureType != 'unknown,']
locations = locations[locations.featureType != '']
locations.dropna(subset=['maxDate','minDate','reprLat','reprLong','featureType'],inplace=True)

#Remove locations before archaic period based on the dataset description
locations= locations[~(locations['minDate'] < -1000)]

#Sequence of actions to split featureType column to two columns,
#the initial and last type of a location

type_index = [locations.maxDate,locations.minDate,locations.reprLat,locations.reprLong]
type_df = pd.DataFrame(locations.featureType.str.split(',').tolist(),index=type_index)

#Reformat column indexes
type_df.reset_index(inplace=True)
type_df.columns = ['maxDate','minDate','reprLat','reprLong','type0','type1','type2','type3','type4']

#Create a numpy array from the dataframe(general bad practise but here is uselful)
arr = np.array(type_df,dtype=str)
for i in range(arr.shape[0]):
    if arr[i][5] in ["","None","nan"]:
        arr[i][5] = arr[i][4]
        continue
    for j in range(6,9):
        if arr[i][j] not in ["","None","nan"]:
            arr[i][5] = arr[i][j]

#Create a new dataframe and keep only the first two types that now
#have the first and last type of each location
final_df = pd.DataFrame(arr)
final_df.columns = ['maxDate','minDate','reprLat','reprLong','initialType','lastType','','','']
final_df = final_df[['maxDate','minDate','reprLat','reprLong','initialType','lastType']]
final_df = final_df.apply(pd.to_numeric, errors='ignore')
final_df = final_df[~(final_df['reprLat'] < -90)]
final_df = final_df[~(final_df['reprLat'] > 90)]

#Find the 10 types that occur more often in the dataset
top_10 = final_df['initialType'].value_counts().keys()[0:10]

#Filter the dataset based on the top 10 types
top10_df = final_df[final_df['initialType'].isin(top_10)]
top10_df = top10_df[top10_df['lastType'].isin(top_10)]
top10_df['averageYear'] = (top10_df['minDate'] + top10_df['maxDate']) /2
```

Distribution all over the world

In the following map, we can see the distribution of pleiades' locations around the world. It is obvious that the **majority** of locations are around **Mediterranean Sea**, but with this visualization we cannot distinguish any of the locations.

```python
#world map from vega_datasets
world_map = alt.topo_feature(data.world_110m.url, 'countries')

#legend selection for types
selection = alt.selection_multi(fields=['initialType'], bind='legend')

#color based on vega color schema 'category10'
dmn=['river', 'fort', 'church', 'island', 'villa', 'settlement-modern','temple', 'mountain', 'theatre', 'settlement']
rng=['#1F77B4','#8C564B','#2CA02C','#D62728','#9467BD','#17BECF','#E377C2','#7F7F7F','#FF7F0E','#BCBD22']
color = alt.Color('initialType:N',scale=alt.Scale(domain=dmn,range=rng),title="Type of Location")

#world map background from topojson data
background = alt.Chart(world_map).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=850, height=425
)

#point for each location of the dataset colored with its type
points = alt.Chart(top10_df).mark_circle(size=3).encode(
    longitude='reprLong:Q',
    latitude='reprLat:Q',
    color=color,
    opacity=alt.condition(selection, alt.OpacityValue(1), alt.OpacityValue(0))
).add_selection(
    selection)

#Create a layer chart from the topojson world map and points of locations
final_map=alt.layer(background,points).properties(
    title=["Distribution of Pleiades' locations around the world,","based on their initial type"]
).configure_legend(
    titleFont='Arial',
    titleFontSize=14,
    labelFont='Arial',
    labelFontSize = 12
).configure_title(
    fontSize=20,
    font='Calibri',
    anchor='middle',
    color='black').project(
    type= 'equirectangular'
)

final_map
```

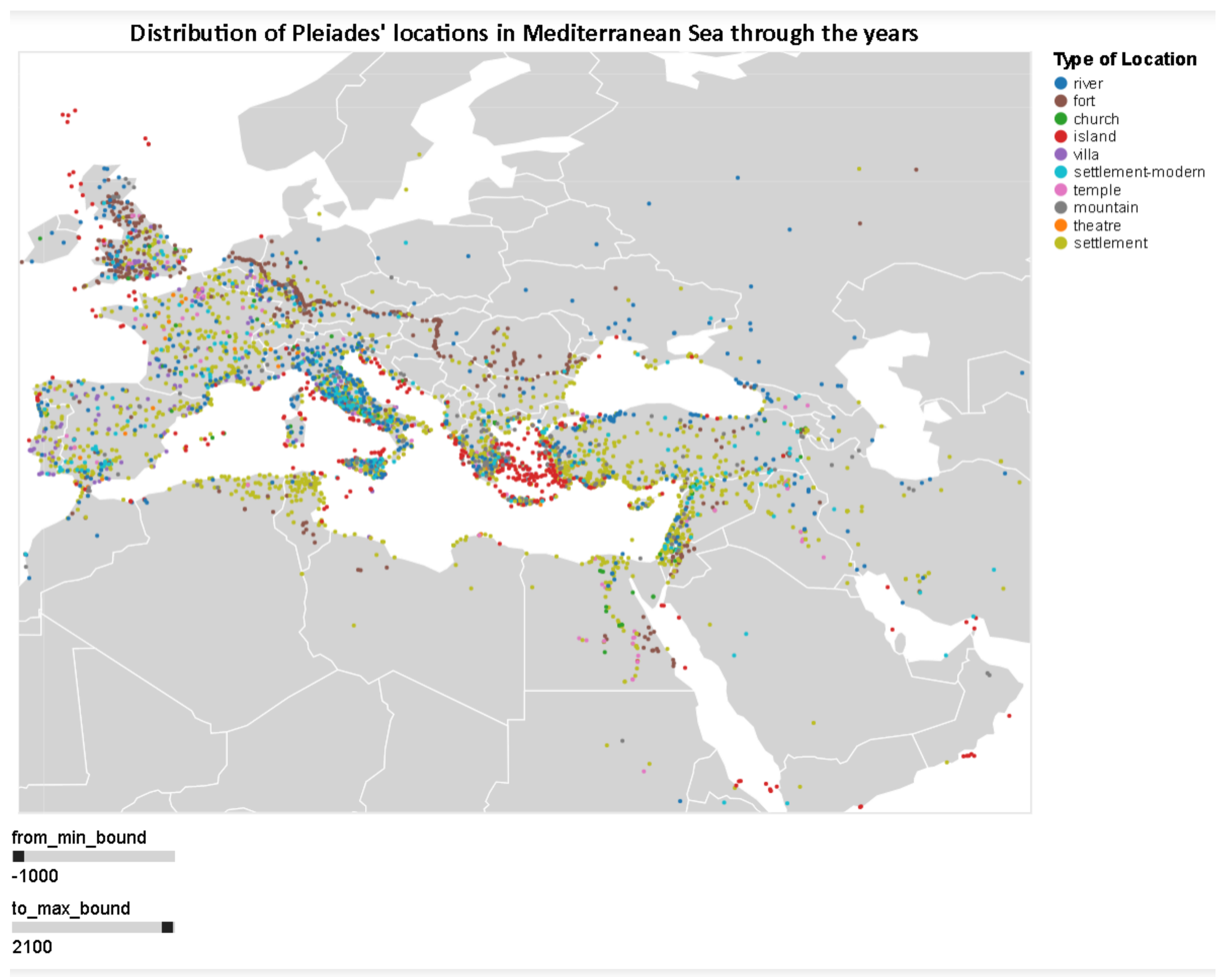


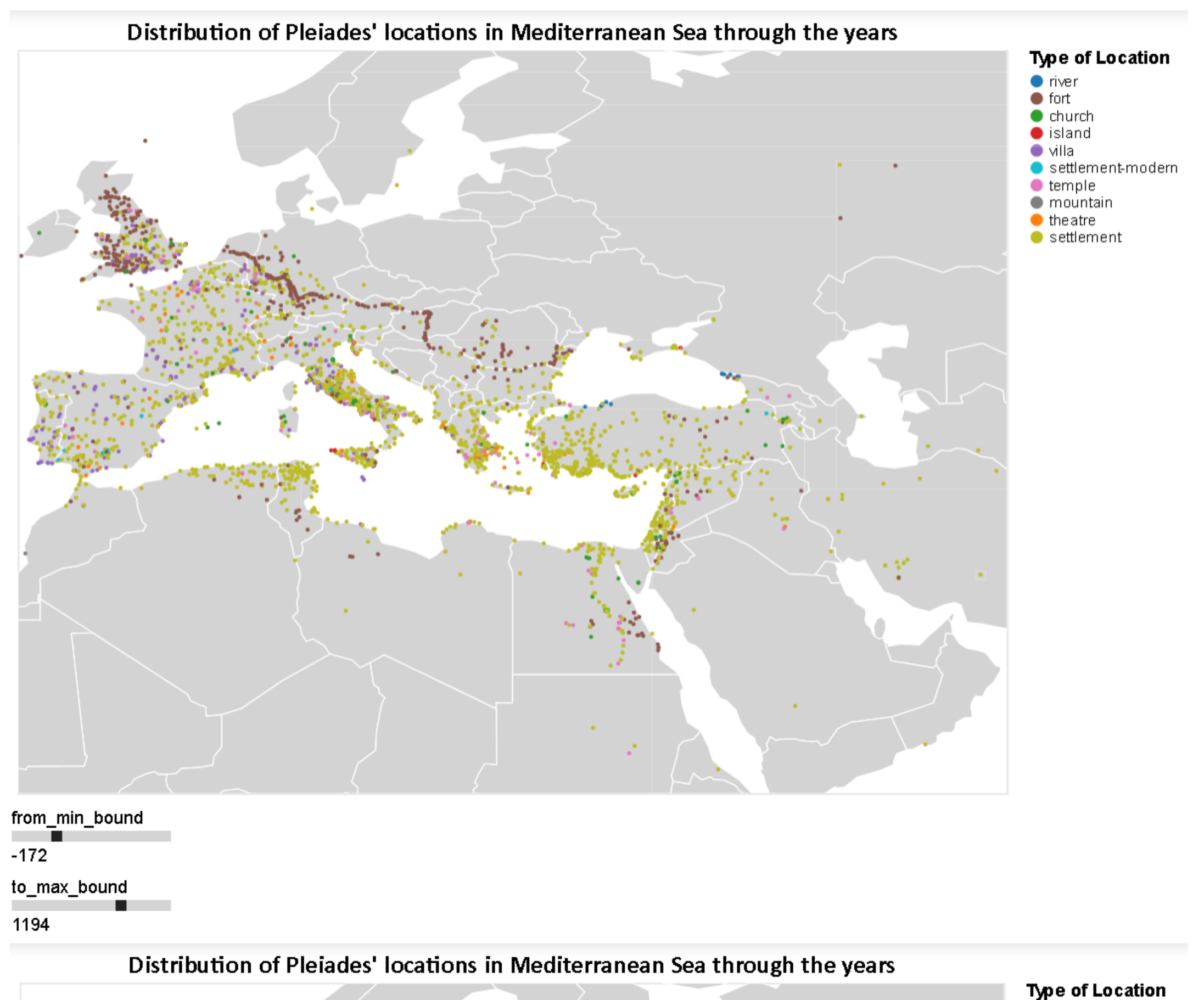


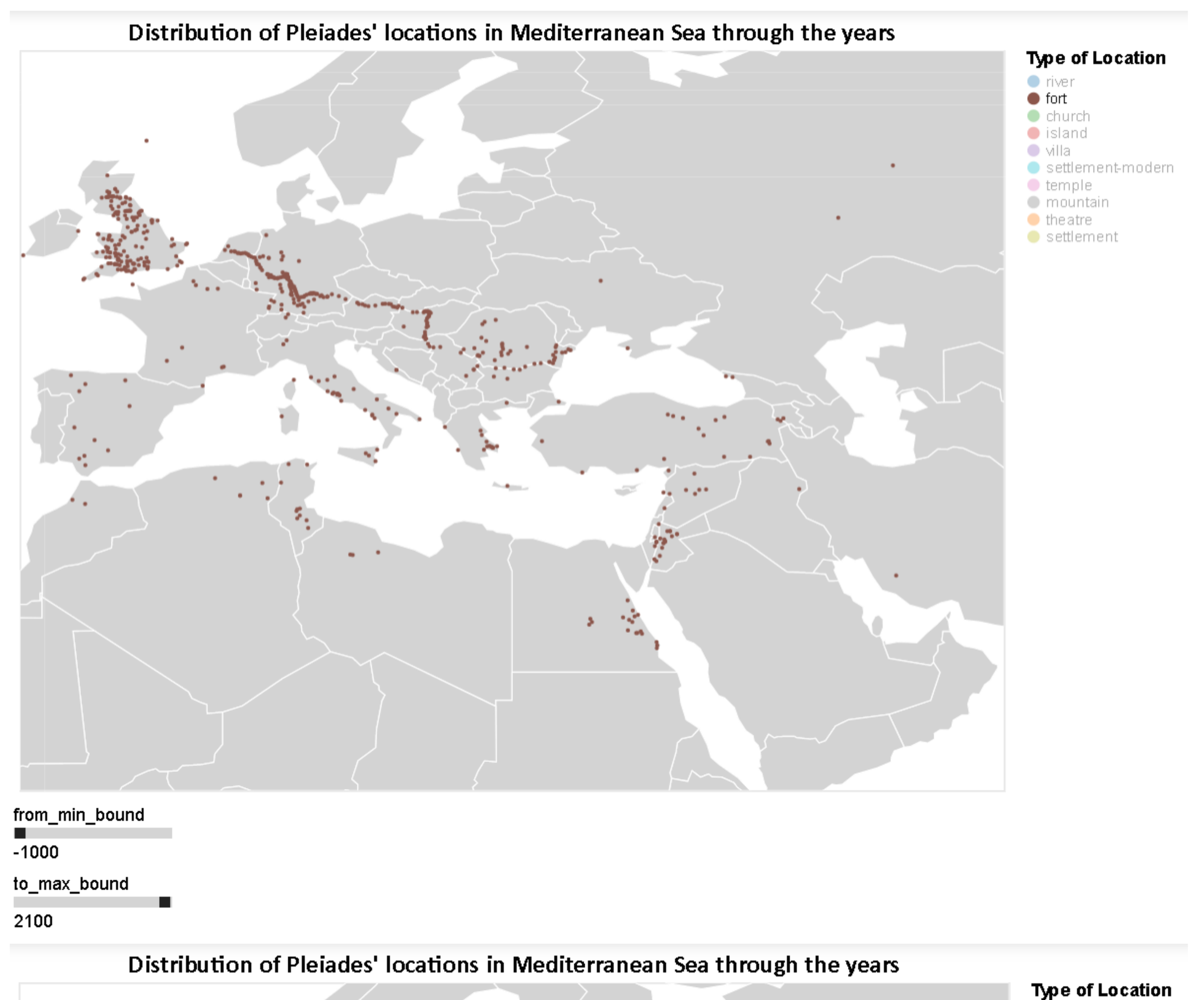
Distribution around Mediterranean Sea

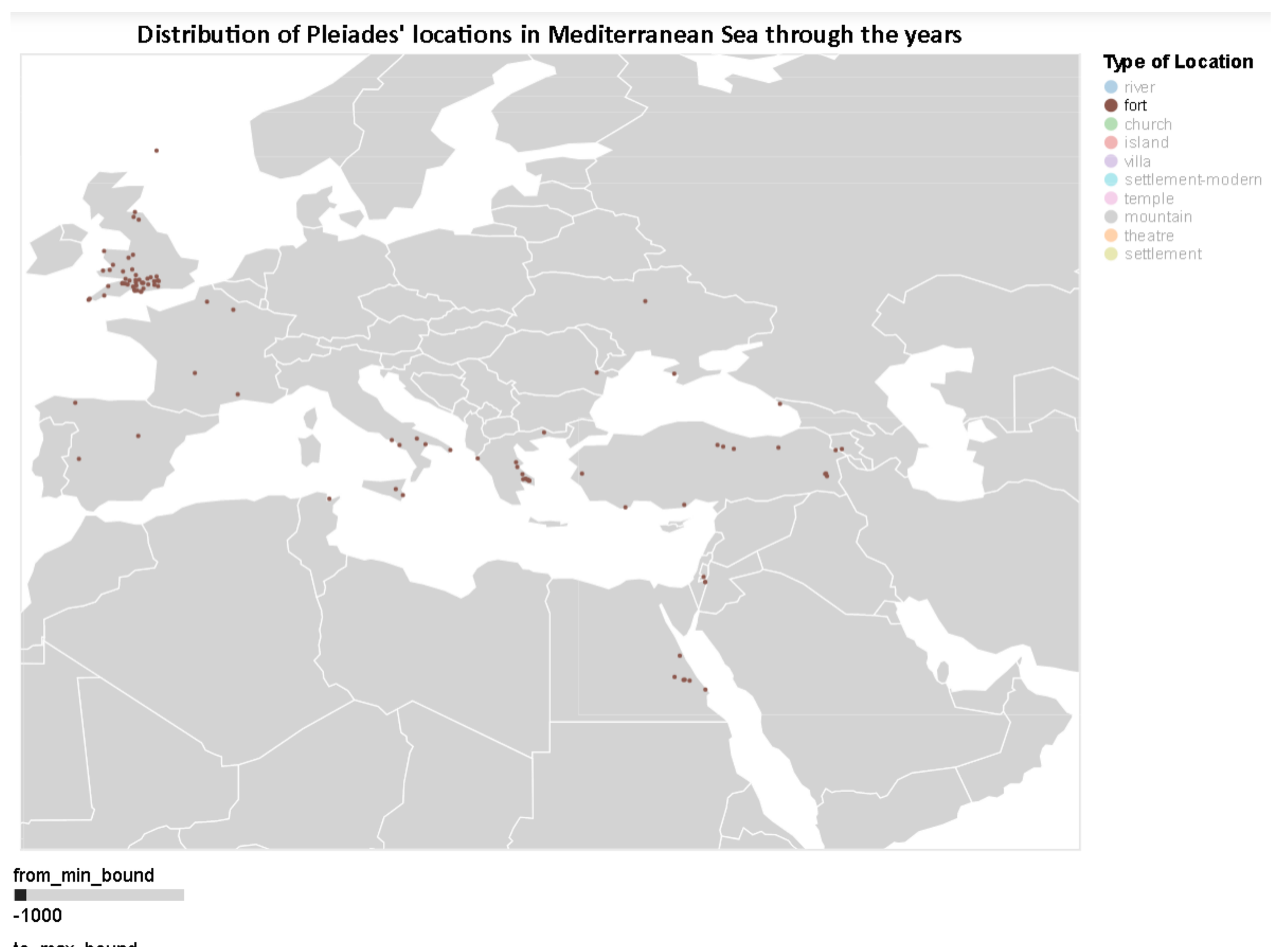
In the visualization below we can see the distribution of locations through the years but also we can select a specific structure to see its distribution. The most interesting aspects of this visualization is that most of the locations are around Italy and Greece showing a high activity through roman and hellinistic years. A line of **fonts** in the middle of Europe is another interesting aspect of this visualization.

```python
#world map from vega_datasets
world_map = alt.topo_feature(data.world_110m.url, 'countries')

#legend selection for types
selection = alt.selection_multi(fields=['initialType'], bind='legend')

#color based on vega color schema 'category10'
dmn=['river', 'fort', 'church', 'island', 'villa', 'settlement-modern','temple', 'mountain', 'theatre', 'settlement']
rng=['#1F77B4','#8C564B','#2CA02C','#D62728','#9467BD','#17BECF','#E377C2','#7F7F7F','#FF7F0E','#BCBD22']
color = alt.Color('initialType:N',scale=alt.Scale(domain=dmn,range=rng),title="Type of Location")

#calculate min and max to be used as bound in sliders
minDate = top10_df['minDate'].min()
maxDate = top10_df['maxDate'].max()

#create minimum and maximum sliders for the chart
min_slider = alt.binding_range(min=minDate, max=maxDate, step = 1)
max_slider = alt.binding_range(min=minDate, max=maxDate, step = 1)

min_select = alt.selection_single(bind=min_slider, fields=['min_bound'], name="from",init={'min_bound':minDate})
max_select = alt.selection_single(bind=max_slider, fields=['max_bound'], name="to",init={'max_bound':maxDate})

#world map background from topojson data
background = alt.Chart(world_map).mark_geoshape(
    fill='lightgrey',
    stroke='white'
).properties(
    width=800, height=600
)
points = alt.Chart(top10_df).mark_circle(size=10).encode(
    longitude='reprLong:Q',
    latitude='reprLat:Q',
    color=color,
    opacity=alt.condition(selection, alt.OpacityValue(1), alt.OpacityValue(0)),
    tooltip=[alt.Tooltip('averageYear',title="Middle year"),
             alt.Tooltip('lastType',title="Last known type"),
             alt.Tooltip('lifetime:Q',title="Lifetime(years)")]
).add_selection(
    selection,
    max_select,
    min_select
).transform_calculate(
    lifetime= 'datum.maxDate - datum.minDate'
).transform_filter(
    #filter to display locations that was known inside year range specified by sliders
    ((alt.datum.minDate >= min_select.min_bound) & (alt.datum.minDate <= max_select.max_bound)) |
    ((alt.datum.maxDate >= min_select.min_bound) & (alt.datum.maxDate <= max_select.max_bound)))

#Create a layer chart from the topojson world map and points of locations
final_map=alt.layer(background,points).properties(
    title="Distribution of Pleiades' locations in Mediterranean Sea through the years"
).configure_legend(
    titleFont='Arial',
    titleFontSize=14,
    labelFont='Arial',
    labelFontSize = 12
).configure_title(
    fontSize=20,
    font='Calibri',
    anchor='middle',
    color='black'
).project(
    type= 'equirectangular',
    scale= 650,
    center= [25,40],
    clipExtent= [[0, 0], [800, 600]],)

final_map
```









What are all visual mappings used?

	World Map	Mediterranean Sea Map
longitude	longitude of location	longitude of location
latitude	latitude of location	latitude of location
color	initial location type	initial location type
tootip1	N/A	middle year((minDate+maxDate)/2)
tootip2	N/A	last known location type
tootip3	N/A	lifetime of location(maxDate-minDate)

Was there any special data preparation done?

I firstly extract from the dataset only the columns that I will use. Using `minDate` and `maxDate` I create another column with the `averageYear` of the location. The extracted data are also filtered with only the top 10 `featureType`. Another column is created inside `transform_calculate` again with the use of `minDate` and `maxDate` to find the `lifetime` of a location.

What are the limitations of your design?

One limitation of this visualization is the fact that the map is not scalable, therefore cannot zoom or navigate through the map. Adding such a feature would lead to more interesting results. Also the fact that the last known type is used as a tooltip is not optimal. Changing the type of the location through the years would lead to more accurate visualizations.

What can we learn from the visualization?

This visualization shows the count of top 10 languages that they spoke in top 10 periods of the dataset colored with the lifetime of the places groub by these periods and languages.

What is the name for the type of visualization(s) used?

For these visualizations we used **scatter plot**

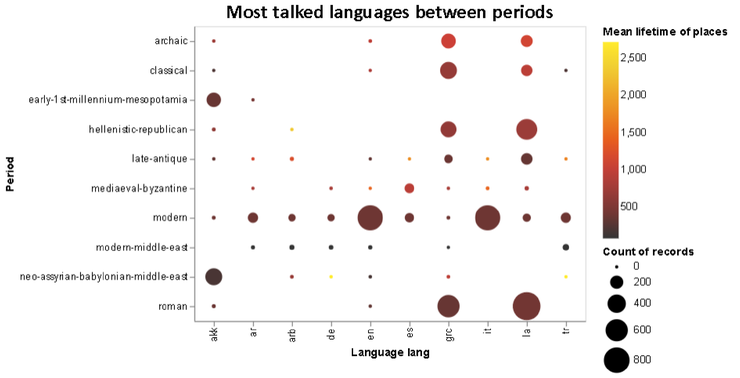
Data Preparation

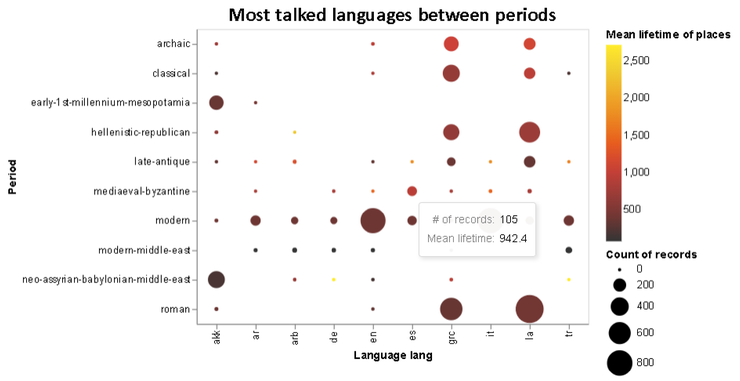
```
import altair as alt
import vega_datasets as data
import pandas as pd
import numpy as np

names = pd.read_csv('pleiades-names.csv')

alt.data_transformers.disable_max_rows()

#Data preparation: Match each period with the type based on order they are inserted
names = names[['nameLanguage','minDate','maxDate','timePeriodsKeys']]

#Remove locations that with missing information, either from timePeriodsKeys or featureType
names['timePeriodsKeys'] = names['timePeriodsKeys'].str.replace(' ','')
names.dropna(subset=['nameLanguage','minDate','maxDate','timePeriodsKeys'],inplace=True)
names['timePeriodsKeys'] = names['timePeriodsKeys'].str.split(',').str[0]
names.reset_index(inplace=True)
names.drop(columns=['index'],inplace=True)

#Find the 10 periods that occur more often in the dataset
top10Periods = names['timePeriodsKeys'].value_counts().keys()[0:10]

#Find the 10 names that occur more often in the dataset
top10Lang = names['nameLanguage'].value_counts().keys()[0:10]

#Filter the dataset based on the top 7 periods and types
names = names[names['timePeriodsKeys'].isin(top10Periods)]
names = names[names['nameLanguage'].isin(top10Lang)]
names['lifetime'] = names['maxDate'] - names['minDate']
```

Top languages among the top periods

In the following visualization we can see what language was the most spoken in every period based on the language were spoken in each place but also the lifetime of those places.

```
alt.Chart(names).mark_circle().encode(
    x=alt.X('nameLanguage:N',title="Language lang"),
    y=alt.Y('timePeriodsKeys:N',title="Period"),
    size=alt.Size('count():Q',scale=alt.Scale(range=[10,800]),title="Count of records"),
    color=alt.Color('mean_lifetime:Q',scale=alt.Scale(scheme="darkred"),title="Mean lifetime of places"),
    tooltip=[alt.Tooltip('count()',title="# of records"),alt.Tooltip('mean_lifetime:Q',title="Mean lifetime")]
).transform_joinaggregate(
    mean_lifetime= 'mean(lifetime)',
    groupby=['nameLanguage','timePeriodsKeys']
).properties(
    title="Most talked languages between periods",
    width=400,
    height=300).configure_title(
    fontSize=20,
    font='Calibri',
    anchor='middle',
    color='black'
)
```





What are all visual mappings used?

	Languages between Periods
x	lang of the language
y	period categories
color	mean lifetime of places
size	count of places
tootip1	count of places
tootip2	mean lifetime of places

Was there any special data preparation done?

I firstly extract only the columns of the dataset that I will use. Using `minDate` and `maxDate` we calculate the `lifetime` of each place. Then I aggregate the data on `nameLanguage` and `timePeriodsKeys` and found the mean of the lifetimes of places in each categoty.

What are the limitations of your design?

From this scatter plot we cannot distinguish the exact correlation between languages and periods.

What can we learn from the visualization?

This visualization shows the relation between top 10 types of places based on their connectivity.

What is the name for the type of visualization(s) used?

For these visualizations we used **heatmap**

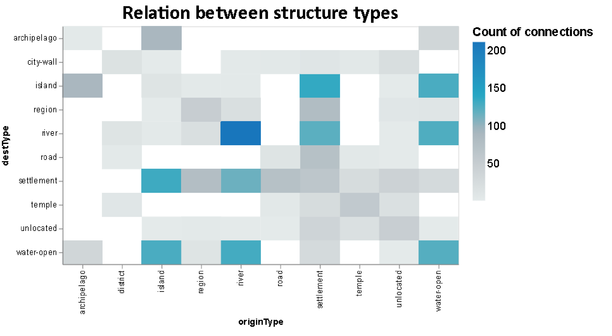
Data Preparation

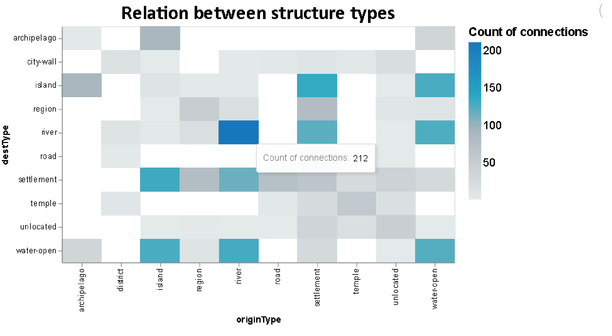
```
import altair as alt
from vega_datasets import data
import pandas as pd
import numpy as np

places = pd.read_csv('pleiades-places.csv')

alt.data_transformers.disable_max_rows()

#Choose only the columns we need and remove missing or wrong inputs
places = places[['reprLong','reprLat','connectsWith','hasConnectionsWith','featureTypes','path']]
places.dropna(subset=['reprLong','reprLat','connectsWith','hasConnectionsWith','featureTypes','path'],inplace=True)
places['path'] = places['path'].str.replace('/places/','')
places['initialType'] = places['featureTypes'].str.split(',').str[0]
places['initialType'] = places['initialType'].str.replace('-2','')
places = places[places.featureTypes != 'unknown']
places = places[places.featureTypes != 'unknown,']
places.drop(columns=['featureTypes'],inplace=True)
places['connectsWith'] = places['connectsWith'].str.replace(' ','')
places['hasConnectionsWith'] = places['hasConnectionsWith'].str.replace(' ','')
places.reset_index(inplace=True)
places.drop(columns=['index'],inplace=True)
places.drop_duplicates(subset=['reprLong', 'reprLat'], inplace=True)

#Create a numpy array from the dataframe(general bad practise but here is uselful)
#From the numpy array we create a list of connections between the places
arr = np.array(places,dtype=str)
con_list = []
for i in range(arr.shape[0]):
    connections = arr[i][2].split(',')
    x = arr[i][3].split(',')
    for con in x:
        connections.append(con)
    for con in connections:
        for k in range(arr.shape[0]):
            if con == arr[k][4]:
                con_list.append([arr[i][4],arr[k][4],arr[i][5],arr[k][5]])
                break

#convert the connection list into dataframe
connections = pd.DataFrame(con_list)
connections.columns=['origin','destination','originType','destType']
connections.sort_values(by=['origin'],inplace=True)
connections.reset_index(inplace=True)
connections.drop(columns=['index'],inplace=True)

#Find the 10 types that occur more often in the dataset
top10_org = connections['originType'].value_counts().keys()[0:10]

top10_des = connections['destType'].value_counts().keys()[0:10]

#Filter the dataset based on the top 10 types
final = connections[connections['originType'].isin(top10_org)]
final = final[final['destType'].isin(top10_des)]
```

Relation between connections of structure types

In the below visualization we can see which of structure types are strongly related to each other based on the connections of places. Except the "obvious" relation between `river` we can see the places that are strongly connected are `open-water`, `river`, `settelment` and `island`.

```
alt.Chart(final).mark_rect().encode(
    x='originType:N',
    y='destType:N',
    color=alt.Color('sum_all:Q', scale=alt.Scale(scheme="lightgreyteal"),title="Count of connections"),
    tooltip=[alt.Tooltip('sum_all:Q',title="Count of connections")]
).transform_joinaggregate(
    suma = 'count()',
    groupby=['originType','destType']
).transform_joinaggregate(
    suma1 = 'count()',
    groupby=['destType','originType']
).transform_calculate(sum_all='datum.suma+datum.suma1').properties(
    title="Relation between structure types",
    width=500,
    height=300
).configure_title(
    fontSize=25,
    font='Calibri',
    anchor='middle',
    color='black'
).configure_legend(
    titleFont='Arial',
    titleFontSize=15,
    labelFont='Arial',
    labelFontSize = 14
)
```





What are all visual mappings used?

	Relation between types
x	origin **type** of connection
y	destination **type** of connection
color	aggregate number of connections
tootip	aggregate number of connections

Was there any special data preparation done?

I firstly extract from the dataset only the columns that I will use. Using a `connectsWith` and `hasConnectionsWith` I created a new dataframe containing all those connections between distinct places in separate rows with its `originType` and `destType` corresponding to type of each place respectively. Grouping by `destType` and `originType` in `connections` dataframe we count the **number of connections** that each combination of connection has.

What are the limitations of your design?

Beside the fact that this visualization shows the relations of structure types based on the connections of places, a hirearchical approach would lead to more clear results because these connections indicate some inheritance. (eg. `river` to `water-open`)