Notebook_for_clustering_final

April 30, 2023

```
In [1]: # Loading the libraries

        import sys
        from pyspark.sql import SparkSession
        import pyspark.sql.functions as F
        import matplotlib.pyplot as plt
        from pyspark.sql.functions import corr
        import matplotlib.pyplot as plt
        import seaborn as sns
        from pyspark.ml.feature import VectorAssembler
        import pandas as pd
        from pyspark.ml.stat import Correlation
        from pyspark.ml.feature import MinMaxScaler
        from pyspark.sql.functions import col

        spark = SparkSession.builder \
        .master("local") \
        .appName("Project") \
        .getOrCreate()
```

1 Loading the dataset before preprocessing

```
In [2]: df = spark.read.csv("query.csv", header=True, inferSchema=True)
        column_names = df.columns
        print(column_names)
```

```
['time', 'latitude', 'longitude', 'depth', 'mag', 'magType', 'nst', 'gap', 'dmin', 'rms', 'net
```

```
In [3]: # Convert the PySpark DataFrame to a Pandas DataFrame
        pd_df = df.select(['time', 'latitude', 'longitude', 'depth', 'mag', 'magType', 'place'

In [4]: pd_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30336 entries, 0 to 30335
Data columns (total 8 columns):
```

```
time              30336 non-null datetime64[ns]
latitude          30336 non-null float64
longitude         30336 non-null float64
depth             30336 non-null float64
mag               30336 non-null float64
magType           30331 non-null object
place             30298 non-null object
locationSource    30336 non-null object
dtypes: datetime64[ns](1), float64(4), object(3)
memory usage: 1.9+ MB
```

In [5]:
```python
# Extracting useful Variables
pd_df = pd_df.dropna(subset=['time', 'latitude', 'longitude', 'depth', 'mag', 'magType
```

In [6]:
```python
# Creating new variable region
pd_df['region'] = pd_df['place'].str.split(',').str[-1].str.strip()
```

In [7]:
```python
# Creating new variable year
pd_df['year'] = pd_df['time'].dt.year
```

In [8]:
```python
# Creating new variable decade_year
pd_df['decade_year'] = (pd_df['year'] // 10) * 10
```

In [9]:
```python
pd_df.head(10)
```

Out[9]:
```
                      time  latitude  longitude   depth  mag magType  \
0  2023-02-28 15:49:17.330   37.9073    36.7306  10.000  3.9     mwr
2  2023-02-19 15:26:21.752   34.6990    32.9989   6.139  3.6      ml
3  2022-08-19 23:16:06.606   35.0510    25.2338  12.618  3.7      mb
4  2022-08-01 02:04:44.377   45.4685    16.2190   9.826  3.7      mb
5  2022-05-12 23:12:04.312   43.6720    11.2558   7.930  3.5      ml
6  2022-05-03 17:50:49.842   43.6656    11.2840   7.420  3.5      ml
7  2022-01-23 11:28:07.151   32.6567    35.4637  10.000  3.7      ml
8  2021-11-03 00:25:46.418   43.9764    20.8848  10.000  3.9      mb
9  2021-07-22 01:03:46.760   38.9222    26.0911  14.730  3.9      mb
10 2021-06-19 14:07:52.133   40.9881    29.1424   9.810  3.5      ml

                                  place locationSource          region  year  \
0                        Central Turkey             us  Central Turkey  2023
2     0 km SE of Páno Polemídia, Cyprus             us          Cyprus  2023
3            8 km ENE of Pýrgos, Greece             us          Greece  2022
4         6 km WNW of Petrinja, Croatia             us         Croatia  2022
5             1 km S of Impruneta, Italy             us           Italy  2022
6   1 km NW of Strada in Chianti, Italy             us           Italy  2022
7         4 km ENE of Kafr Mi?r, Israel             us          Israel  2022
8       5 km SSW of Kragujevac, Serbia             us          Serbia  2021
9      19 km SSW of Polichnítos, Greece             us          Greece  2021
10          2 km ENE of Ata?ehir, Turkey             us          Turkey  2021
```

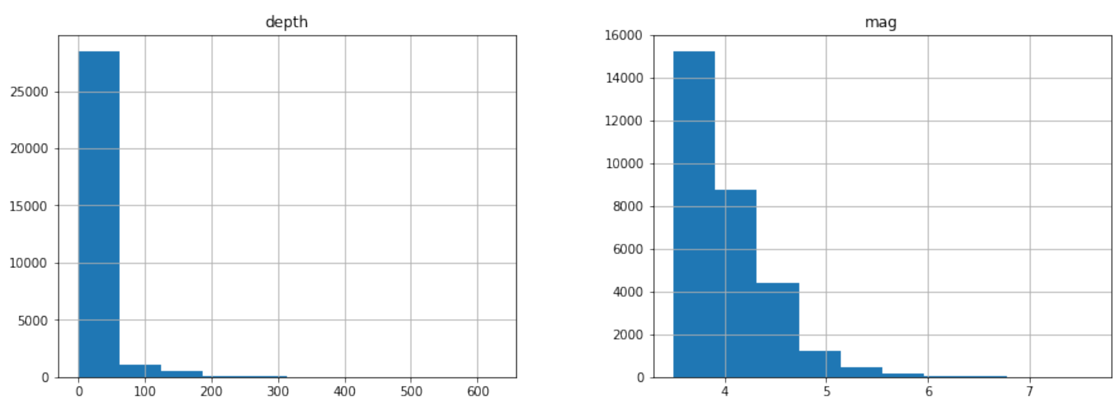
```
    decade_year
0          2020
2          2020
3          2020
4          2020
5          2020
6          2020
7          2020
8          2020
9          2020
10         2020
```

In [10]:
```
# Descriptive Statistics
pd_df[['latitude', 'longitude', 'depth', 'mag']].describe()
```

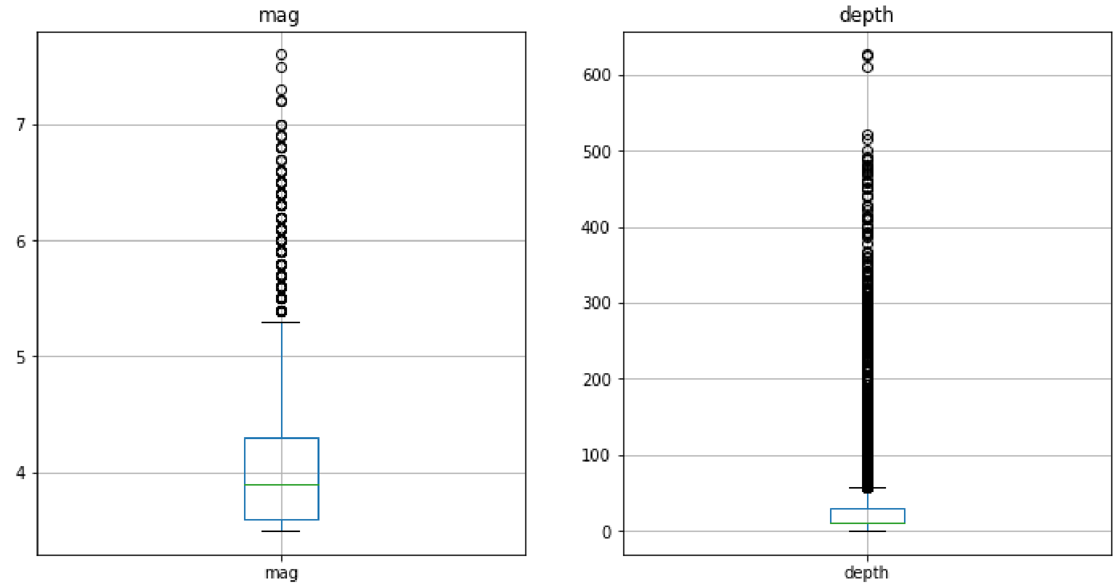
Out[10]:

	latitude	longitude	depth	mag
count	30293.00000	30293.000000	30293.000000	30293.000000
mean	38.32942	21.310123	23.410525	4.018592
std	2.90449	8.200119	36.356840	0.458081
min	30.07900	-5.727000	0.000000	3.500000
25%	36.11000	20.050000	10.000000	3.600000
50%	38.04800	22.314000	10.000000	3.900000
75%	39.94400	26.485900	28.900000	4.300000
max	46.01200	36.730600	626.200000	7.600000

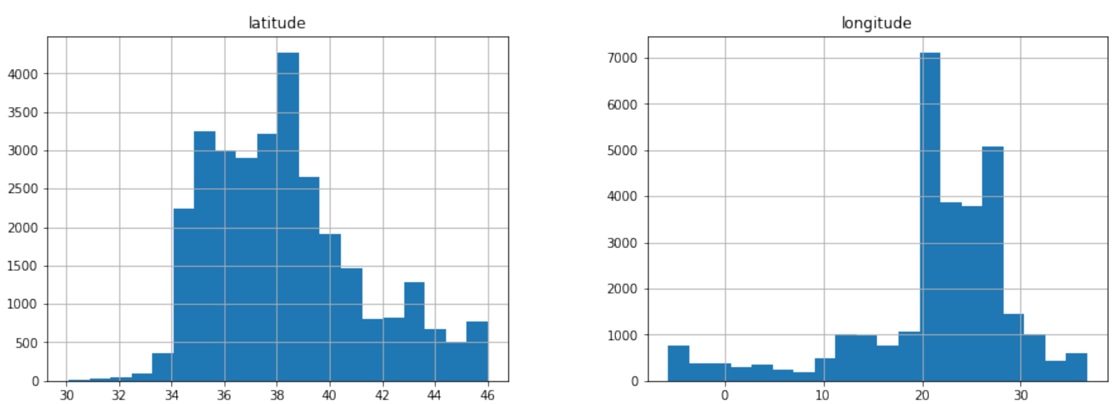
In [11]:
```
# Extract the "magnitude" and "depth" columns to plot
columns = ["mag", "depth"]

# Plot the histograms side-by-side
pd_df.hist(column=columns, bins=10, figsize=(15, 5))

plt.show()
```



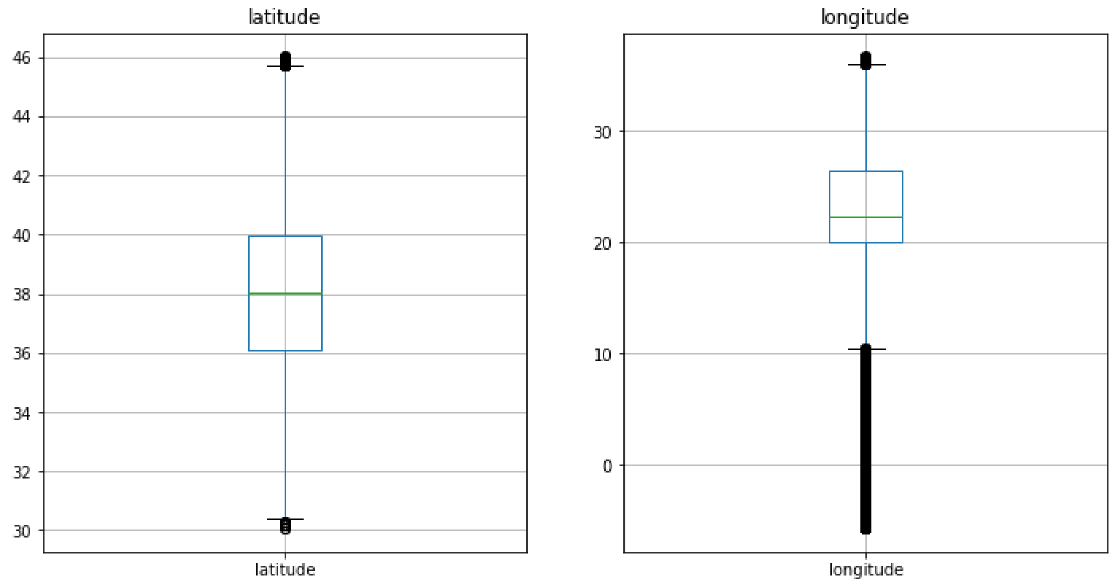
In [12]:
```
# Plot the boxplot
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
for i, col in enumerate(columns):
    pd_df.boxplot(column=col, ax=axes[i])
    axes[i].set_title(col)
plt.show()
```



In [13]:
```
# Extract the "latitude" and "longitude" columns to plot
columns = ["latitude", "longitude"]

# Plot the histograms side-by-side
pd_df.hist(column=columns, bins=20, figsize=(15, 5))

plt.show()
```



In [15]:
```python
# Plot the histograms side-by-side
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
for i, col in enumerate(columns):
    pd_df.boxplot(column=col, ax=axes[i])
    axes[i].set_title(col)
plt.show()
```

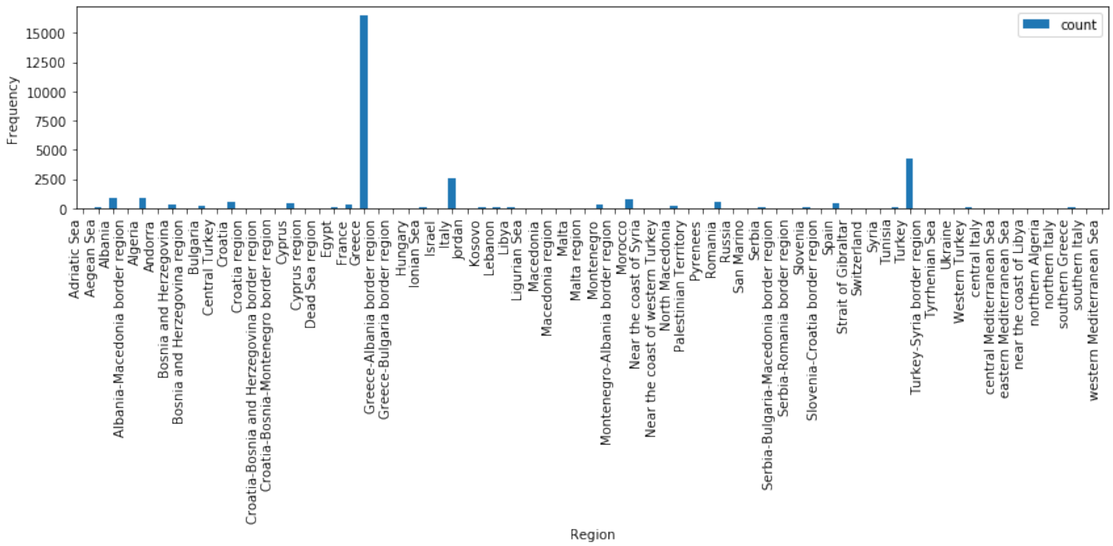


In [16]:
```python
# Descriptive statistics for each decade
decade_mag_stats = pd_df.groupby('decade_year')['mag'].describe()
print(decade_mag_stats)
```

```
               count      mean       std  min  25%  50%  75%  max
decade_year
1970          1737.0  4.226079  0.507529  3.5  3.8  4.2  4.5  7.5
1980          5447.0  4.052084  0.460869  3.5  3.7  4.0  4.3  7.3
1990          7386.0  3.936055  0.417730  3.5  3.6  3.8  4.1  7.6
2000         10554.0  3.846077  0.391493  3.5  3.6  3.7  4.0  6.9
2010          3508.0  4.375285  0.373245  3.5  4.1  4.3  4.5  6.9
2020          1661.0  4.401626  0.366131  3.5  4.2  4.3  4.5  7.0
```

In [17]:
```python
# Creating a barplot to depict the number of earthquake in each region

region_counts = pd_df.groupby('region').size().reset_index(name='count')
fig, ax = plt.subplots(figsize=(12, 6))
region_counts.plot(x='region', y='count', kind='bar', ax=ax)
ax.set_xlabel('Region')
ax.set_ylabel('Frequency')
```

```
ax.set_xticklabels(region_counts['region'], rotation=90, ha='right')
plt.tight_layout()
plt.show()
```



In [19]:
```
# Descriptive statistics for each region
Country_mag_stats = pd_df.groupby('region')['mag'].describe()
print(Country_mag_stats)
```

```
                                              count      mean       std  \
region
Adriatic Sea                                   10.0  3.910000  0.401248
Aegean Sea                                     45.0  3.820000  0.344172
Albania                                       924.0  4.029870  0.467046
Albania-Macedonia border region                 7.0  3.885714  0.353217
Algeria                                       839.0  4.135042  0.522215
Andorra                                         1.0  3.700000       NaN
Bosnia and Herzegovina                        316.0  4.018671  0.448913
Bosnia and Herzegovina region                   2.0  3.800000  0.282843
Bulgaria                                      150.0  4.012667  0.432147
Central Turkey                                 33.0  4.345455  0.439525
Croatia                                       496.0  3.992540  0.441262
Croatia region                                  7.0  3.957143  0.320713
Croatia-Bosnia and Herzegovina border region    1.0  3.500000       NaN
Croatia-Bosnia-Montenegro border region         3.0  4.033333  0.611010
Cyprus                                        481.0  4.046778  0.484977
Cyprus region                                  13.0  4.261538  0.550058
Dead Sea region                                 1.0  3.900000       NaN
Egypt                                          46.0  4.236957  0.449868
France                                        318.0  3.888994  0.388037
```

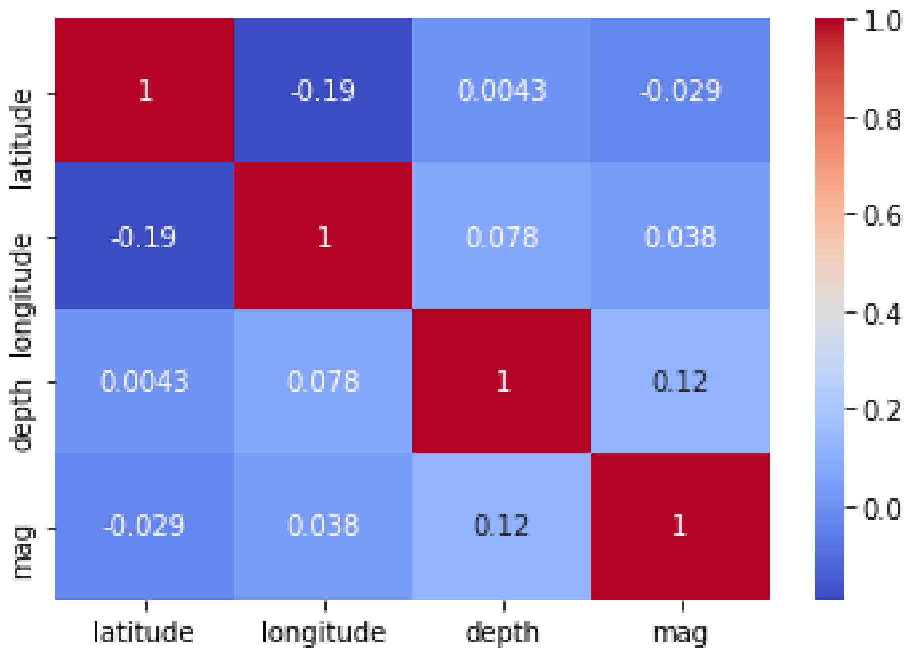
Greece	16480.0	4.004345	0.447618
Greece-Albania border region	16.0	4.062500	0.538981
Greece-Bulgaria border region	1.0	3.800000	NaN
Hungary	1.0	3.800000	NaN
Ionian Sea	38.0	4.039474	0.457691
Israel	26.0	4.153846	0.508512
Italy	2609.0	4.053354	0.471021
Jordan	11.0	4.127273	0.293567
Kosovo	89.0	4.108989	0.495122
Lebanon	36.0	4.027778	0.396132
Libya	80.0	4.255000	0.514326
...	...	...	...
North Macedonia	162.0	3.948765	0.457815
Palestinian Territory	9.0	4.122222	0.338296
Pyrenees	7.0	3.700000	0.258199
Romania	536.0	4.208582	0.521143
Russia	1.0	4.000000	NaN
San Marino	1.0	3.600000	NaN
Serbia	114.0	4.076316	0.509086
Serbia-Bulgaria-Macedonia border region	1.0	5.600000	NaN
Serbia-Romania border region	1.0	3.800000	NaN
Slovenia	44.0	3.977273	0.443487
Slovenia-Croatia border region	1.0	3.600000	NaN
Spain	473.0	3.889852	0.408623
Strait of Gibraltar	24.0	3.937500	0.503736
Switzerland	1.0	3.600000	NaN
Syria	30.0	4.273333	0.301643
Tunisia	87.0	4.352874	0.469745
Turkey	4305.0	4.018908	0.458927
Turkey-Syria border region	9.0	4.433333	0.469042
Tyrrhenian Sea	3.0	4.366667	0.321455
Ukraine	24.0	4.054167	0.329663
Western Turkey	71.0	3.976056	0.335714
central Italy	19.0	4.063158	0.577553
central Mediterranean Sea	27.0	4.066667	0.543493
eastern Mediterranean Sea	21.0	4.128571	0.324257
near the coast of Libya	2.0	4.050000	0.070711
northern Algeria	22.0	4.159091	0.523413
northern Italy	19.0	3.884211	0.341993
southern Greece	48.0	4.027083	0.455108
southern Italy	12.0	4.125000	0.510125
western Mediterranean Sea	2.0	3.850000	0.353553

	min	25%	50%	75%	max
region					
Adriatic Sea	3.5	3.525	3.85	4.175	4.6
Aegean Sea	3.5	3.600	3.70	3.900	4.7
Albania	3.5	3.600	4.00	4.300	6.4

Albania-Macedonia border region	3.5	3.550	4.00	4.150	4.3
Algeria	3.5	3.700	4.00	4.400	7.3
Andorra	3.7	3.700	3.70	3.700	3.7
Bosnia and Herzegovina	3.5	3.700	3.90	4.200	5.7
Bosnia and Herzegovina region	3.6	3.700	3.80	3.900	4.0
Bulgaria	3.5	3.600	3.95	4.300	5.6
Central Turkey	3.5	4.100	4.30	4.400	5.8
Croatia	3.5	3.600	3.90	4.200	6.4
Croatia region	3.5	3.750	3.90	4.200	4.4
Croatia-Bosnia and Herzegovina border region	3.5	3.500	3.50	3.500	3.5
Croatia-Bosnia-Montenegro border region	3.5	3.700	3.90	4.300	4.7
Cyprus	3.5	3.700	4.00	4.300	6.8
Cyprus region	3.5	4.100	4.20	4.300	5.9
Dead Sea region	3.9	3.900	3.90	3.900	3.9
Egypt	3.5	3.900	4.20	4.500	5.5
France	3.5	3.600	3.80	4.100	5.3
Greece	3.5	3.600	3.90	4.300	7.2
Greece-Albania border region	3.5	3.575	4.05	4.325	5.3
Greece-Bulgaria border region	3.8	3.800	3.80	3.800	3.8
Hungary	3.8	3.800	3.80	3.800	3.8
Ionian Sea	3.5	3.700	4.05	4.300	5.5
Israel	3.5	3.700	4.05	4.375	5.3
Italy	3.5	3.700	4.00	4.300	6.9
Jordan	3.6	4.000	4.10	4.300	4.5
Kosovo	3.5	3.700	4.00	4.300	5.7
Lebanon	3.5	3.775	4.00	4.200	5.1
Libya	3.5	3.900	4.20	4.400	6.4
...	...	...	...	...	...
North Macedonia	3.5	3.600	3.80	4.200	5.8
Palestinian Territory	3.5	4.000	4.10	4.400	4.6
Pyrenees	3.5	3.500	3.50	3.900	4.1
Romania	3.5	3.800	4.10	4.500	7.5
Russia	4.0	4.000	4.00	4.000	4.0
San Marino	3.6	3.600	3.60	3.600	3.6
Serbia	3.5	3.700	4.00	4.300	5.8
Serbia-Bulgaria-Macedonia border region	5.6	5.600	5.60	5.600	5.6
Serbia-Romania border region	3.8	3.800	3.80	3.800	3.8
Slovenia	3.5	3.675	3.90	4.300	5.2
Slovenia-Croatia border region	3.6	3.600	3.60	3.600	3.6
Spain	3.5	3.600	3.80	4.100	6.3
Strait of Gibraltar	3.5	3.500	3.75	4.200	5.5
Switzerland	3.6	3.600	3.60	3.600	3.6
Syria	3.5	4.100	4.30	4.475	5.0
Tunisia	3.5	4.000	4.40	4.650	5.2
Turkey	3.5	3.700	3.90	4.300	7.6
Turkey-Syria border region	4.0	4.200	4.30	4.500	5.6
Tyrrhenian Sea	4.0	4.250	4.50	4.550	4.6
Ukraine	3.5	3.800	4.10	4.200	4.9

```
Western Turkey                               3.5  3.700  3.90  4.200  5.0
central Italy                                3.5  3.600  4.10  4.250  5.6
central Mediterranean Sea                    3.5  3.650  4.00  4.200  5.4
eastern Mediterranean Sea                    3.7  3.900  4.10  4.400  4.8
near the coast of Libya                      4.0  4.025  4.05  4.075  4.1
northern Algeria                             3.6  3.750  4.10  4.275  5.8
northern Italy                               3.5  3.650  3.80  4.100  4.6
southern Greece                              3.5  3.600  4.00  4.400  5.5
southern Italy                               3.6  3.775  3.90  4.350  5.3
western Mediterranean Sea                    3.6  3.725  3.85  3.975  4.1

[70 rows x 8 columns]
```

In [20]:
```python
# Calculate correlation coefficients
corr = pd_df[['latitude', 'longitude', 'depth', 'mag']].corr()

# Create a heatmap of the correlation matrix
sns.heatmap(corr, cmap='coolwarm', annot=True)

# Show the plot
plt.show()
```



In [144]:
```python
s_df = spark.createDataFrame(pd_df)
```

```
In [145]: # Dataset after extraction of columns
          new_df = s_df.select(['year', 'decade_year','region','locationSource','magType','lat
          new_df.show(10)
```

```
+----+-----------+--------------+--------------+-------+--------+---------+------+---+
|year|decade_year|        region|locationSource|magType|latitude|longitude| depth|mag|
+----+-----------+--------------+--------------+-------+--------+---------+------+---+
|2023|       2020|Central Turkey|            us|    mwr| 37.9073|  36.7306|  10.0|3.9|
|2023|       2020|        Cyprus|            us|     ml|  34.699|  32.9989| 6.139|3.6|
|2022|       2020|        Greece|            us|     mb|  35.051|  25.2338|12.618|3.7|
|2022|       2020|       Croatia|            us|     mb| 45.4685|   16.219| 9.826|3.7|
|2022|       2020|         Italy|            us|     ml|  43.672|  11.2558|  7.93|3.5|
|2022|       2020|         Italy|            us|     ml| 43.6656|   11.284|  7.42|3.5|
|2022|       2020|        Israel|            us|     ml| 32.6567|  35.4637|  10.0|3.7|
|2021|       2020|        Serbia|            us|     mb| 43.9764|  20.8848|  10.0|3.9|
|2021|       2020|        Greece|            us|     mb| 38.9222|  26.0911| 14.73|3.9|
|2021|       2020|        Turkey|            us|     ml| 40.9881|  29.1424|  9.81|3.5|
+----+-----------+--------------+--------------+-------+--------+---------+------+---+
only showing top 10 rows
```

```
In [146]: # Checking the datatype and possibility for null or nan value
          new_df.printSchema()
```

```
root
 |-- year: long (nullable = true)
 |-- decade_year: long (nullable = true)
 |-- region: string (nullable = true)
 |-- locationSource: string (nullable = true)
 |-- magType: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- depth: double (nullable = true)
 |-- mag: double (nullable = true)
```

```
In [147]: # Check for null values
          null_counts = [(column, new_df.where(new_df[column].isNull()).count()) for column in
          print(null_counts)
```

```
[('year', 0), ('decade_year', 0), ('region', 0), ('locationSource', 0), ('magType', 0), ('lati
```

```
In [148]: # Get the number of rows and columns
          num_rows = new_df.count()
          num_cols = len(new_df.columns)
```

```
# Print the dimensions
print("Number of rows: ", num_rows)
print("Number of columns: ", num_cols)
```

```
Number of rows:  30293
Number of columns:  9
```

Scaling the Dataset using Standarization method

In [152]:
```
from pyspark.ml.feature import StandardScaler
from pyspark.sql.functions import log

# Define the input columns to the scaler
input_cols = ['latitude', 'longitude', 'decade_year','depth','mag']

# Assemble the input columns into a single vector column
assembler = VectorAssembler(inputCols=input_cols, outputCol='features')
new_df1 = assembler.transform(new_df)

# Standardize the data
standardScaler = StandardScaler(inputCol="features", outputCol="scaled_features")
scaled_data = standardScaler.fit(new_df1).transform(new_df1)

# Preview of the scaled_feature
scaled_data.show(10)

scaled_data.select("scaled_features").show(1, truncate=False)
```

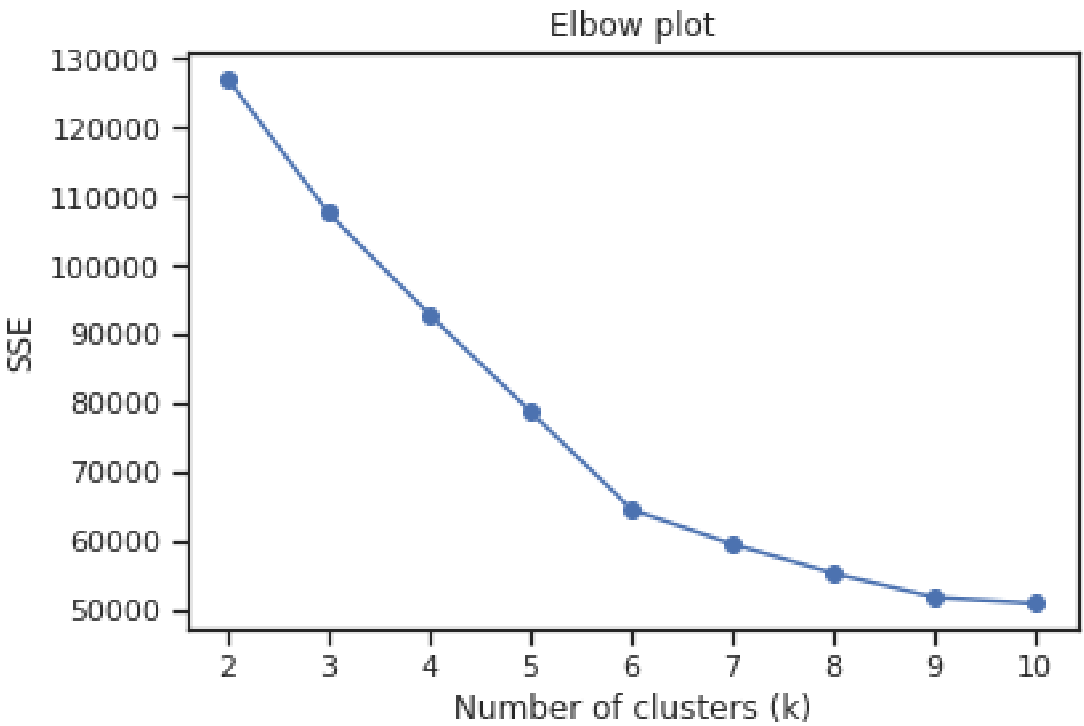
```
+----+-----------+--------------+--------------+-------+--------+---------+------+---+---------
|year|decade_year|        region|locationSource|magType|latitude|longitude| depth|mag|
+----+-----------+--------------+--------------+-------+--------+---------+------+---+---------
|2023|       2020|Central Turkey|            us|    mwr| 37.9073|  36.7306|  10.0|3.9|1.360976
|2023|       2020|        Cyprus|            us|     ml|  34.699|  32.9989| 6.139|3.6|1.280933
|2022|       2020|        Greece|            us|     mb|  35.051|  25.2338|12.618|3.7| 1.308833
|2022|       2020|       Croatia|            us|     mb| 45.4685|   16.219| 9.826|3.7| 1.308833
|2022|       2020|         Italy|            us|     ml|  43.672|  11.2558|  7.93|3.5| 1.25276
|2022|       2020|         Italy|            us|     ml| 43.6656|   11.284|  7.42|3.5| 1.25276
|2022|       2020|        Israel|            us|     ml| 32.6567|  35.4637|  10.0|3.7| 1.308833
|2021|       2020|        Serbia|            us|     mb| 43.9764|  20.8848|  10.0|3.9|1.360976
|2021|       2020|        Greece|            us|     mb| 38.9222|  26.0911| 14.73|3.9|1.360976
|2021|       2020|        Turkey|            us|     ml| 40.9881|  29.1424|  9.81|3.5| 1.25276
+----+-----------+--------------+--------------+-------+--------+---------+------+---+---------
only showing top 10 rows

+-----------------------------------------------------------------------------------------------
|scaled_features
+-----------------------------------------------------------------------------------------------
|[13.05127609493899,4.479276476379836,164.28722233459348,0.27505140956220075,8.513784632227685]
```

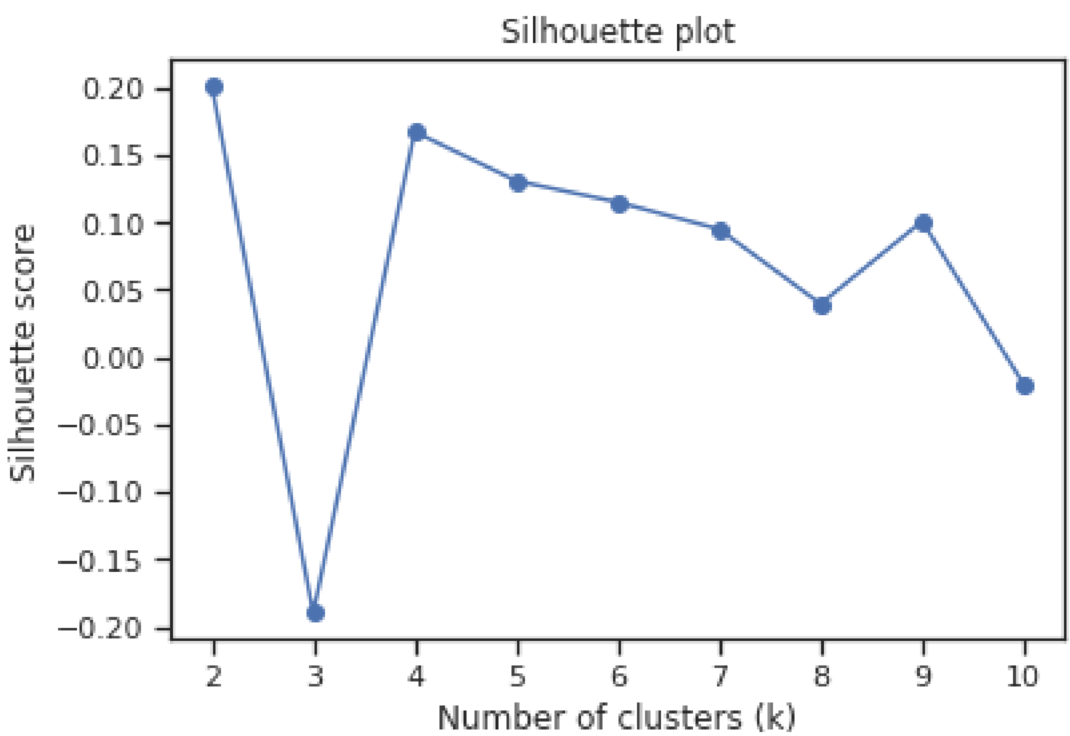
```
+------------------------------------------------------------------------------------------------
only showing top 1 row
```

Using Elbow and Silhouette method for finding optimal cluster

```
In [153]: from pyspark.ml.clustering import KMeans
          from pyspark.ml.evaluation import ClusteringEvaluator
          import numpy as np
          import matplotlib.pyplot as plt

          # Define the range of k values to test
          k_values = range(2, 11)

          # Initialize lists to store SSE values and silhouette scores
          sse_values = []
          silhouette_scores = []

          # Loop over each k value and fit the k-means model
          for k in k_values:
              kmeans = KMeans(k=k, seed=1).setFeaturesCol("scaled_features")
              model = kmeans.fit(scaled_data)

              # Make predictions and calculate SSE
              predictions = model.transform(scaled_data)
              sse = model.computeCost(scaled_data)

              # Calculate silhouette score
              evaluator = ClusteringEvaluator()
              silhouette_score = evaluator.evaluate(predictions)


              # Append SSE and silhouette score to lists
              sse_values.append(sse)
              silhouette_scores.append(silhouette_score)

          # Plot the SSE values as a function of k
          plt.plot(k_values, sse_values, '-o')
          plt.xlabel('Number of clusters (k)')
          plt.ylabel('SSE')
          plt.title('Elbow plot')
          plt.show()

          # Plot the silhouette scores as a function of k
          plt.plot(k_values, silhouette_scores, '-o')
          plt.xlabel('Number of clusters (k)')
          plt.ylabel('Silhouette score')
```

```
plt.title('Silhouette plot')
plt.show()
```





Clustering Using K means with 4 cluster. You can also change the cluster i.e., k value to see the change in analysis

In [154]:
```
from pyspark.ml.clustering import KMeans

# Set the number of clusters : Here elbow method suggest 4 clusters where slihouette
k = 4
# Create the KMeans object and fit to the data
kmeans = KMeans(k=k, seed=1).setFeaturesCol("scaled_features")
model = kmeans.fit(scaled_data)

# Make predictions and evaluate the model
predictions = model.transform(scaled_data)

# Extract the predicted cluster values from predictions
predicted_clusters = predictions.select('prediction')

predicted_clusters.show(10)
```

```
+----------+
|prediction|
+----------+
|         3|
|         3|
|         3|
|         1|
|         1|
|         1|
|         3|
|         3|
|         3|
|         0|
+----------+
only showing top 10 rows
```

In [155]:
```
new_df.show(10)
```

```
+----+-----------+--------------+--------------+-------+--------+---------+------+---+--------
|year|decade_year|        region|locationSource|magType|latitude|longitude| depth|mag|
+----+-----------+--------------+--------------+-------+--------+---------+------+---+--------
|2023|       2020|Central Turkey|            us|    mwr| 37.9073|  36.7306|  10.0|3.9|1.360976
|2023|       2020|        Cyprus|            us|     ml|  34.699|  32.9989| 6.139|3.6|1.280933
|2022|       2020|        Greece|            us|     mb|  35.051|  25.2338|12.618|3.7| 1.30833
|2022|       2020|       Croatia|            us|     mb| 45.4685|   16.219| 9.826|3.7| 1.30833
|2022|       2020|         Italy|            us|     ml|  43.672|  11.2558|  7.93|3.5| 1.25276
```

```
|2022|       2020|         Italy|           us|     ml| 43.6656|   11.284|  7.42|3.5| 1.25276
|2022|       2020|        Israel|           us|     ml| 32.6567|  35.4637|  10.0|3.7| 1.30833
|2021|       2020|        Serbia|           us|     mb| 43.9764|  20.8848|  10.0|3.9|1.360976
|2021|       2020|        Greece|           us|     mb| 38.9222|  26.0911| 14.73|3.9|1.360976
|2021|       2020|        Turkey|           us|     ml| 40.9881|  29.1424|  9.81|3.5| 1.25276
+----+-----------+-------------+-------------+-------+--------+---------+------+---+--------
only showing top 10 rows
```

In [191]:
```python
import numpy as np
# Creating a Pandas dataframe for analysing clusters

data = np.array(new_df.select('decade_year','latitude', 'longitude','mag','depth').co

clusters = np.array(predictions.select("prediction").collect()).reshape(-1,1)

data1 = np.hstack((data, clusters))

df1 = pd.DataFrame(data = data1, columns=('decade_year','latitude', 'longitude', 'mag
df1.head(10)
```

Out[191]:

	decade_year	latitude	longitude	mag	depth	clusters
0	2020.0	37.9073	36.7306	3.9	10.000	3.0
1	2020.0	34.6990	32.9989	3.6	6.139	3.0
2	2020.0	35.0510	25.2338	3.7	12.618	3.0
3	2020.0	45.4685	16.2190	3.7	9.826	1.0
4	2020.0	43.6720	11.2558	3.5	7.930	1.0
5	2020.0	43.6656	11.2840	3.5	7.420	1.0
6	2020.0	32.6567	35.4637	3.7	10.000	3.0
7	2020.0	43.9764	20.8848	3.9	10.000	3.0
8	2020.0	38.9222	26.0911	3.9	14.730	3.0
9	2020.0	40.9881	29.1424	3.5	9.810	0.0

Cluster analysis

In [179]:
```python
# Descriptive of clusters
summary_mean = df1.groupby('clusters').mean()
print(summary_mean)
```

```
                year   latitude  longitude       mag      depth
clusters
0.0       1996.793299  37.722833  24.450266  3.768903  17.497919
1.0       1997.519381  40.753160   8.267179  3.943706  10.924427
2.0       1987.466782  38.704592  23.393156  4.445039  80.260745
3.0       2013.217297  37.376092  24.377763  4.473754  17.140235
```

In [180]:
```python
summary_sd = df1.groupby('clusters').std()
print(summary_sd)
```

```
             year  latitude  longitude       mag      depth
clusters
0.0      8.856128  2.020070   3.802415  0.240784  13.489890
1.0     10.704326  3.686642   8.152568  0.380088   9.324065
2.0     11.320844  3.309735   4.864333  0.482056  80.413636
3.0      7.703772  2.397469   5.525668  0.408550  15.433507
```

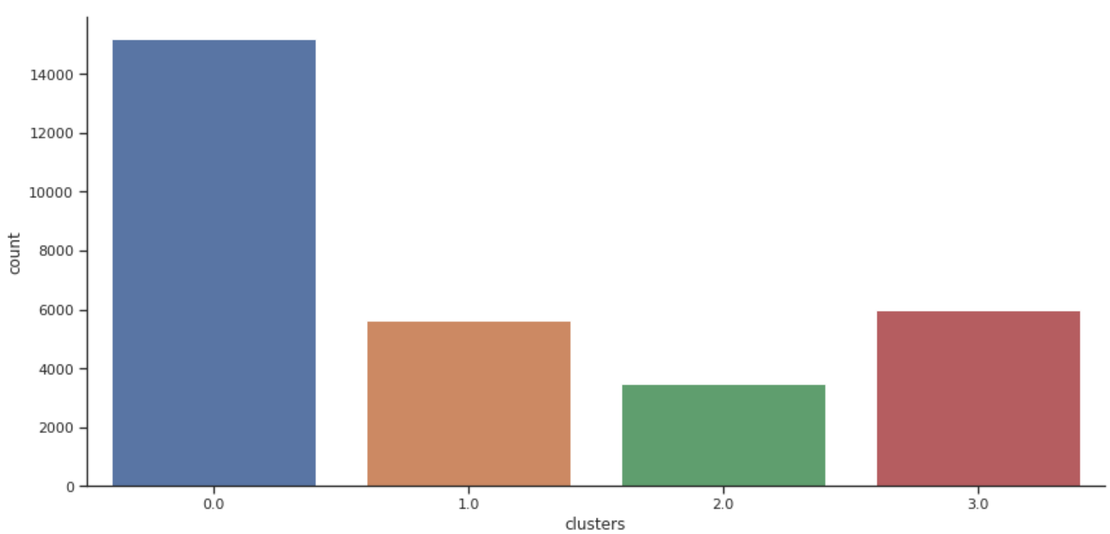
In [181]:
```python
summary_min = df1.groupby('clusters').min()
print(summary_min)
```

```
            year  latitude  longitude  mag  depth
clusters
0.0       1973.0    30.088      9.280  3.5    0.0
1.0       1973.0    31.678     -5.727  3.5    0.0
2.0       1973.0    30.480     -4.634  3.5    2.0
3.0       1988.0    30.079     -4.372  3.6    0.0
```

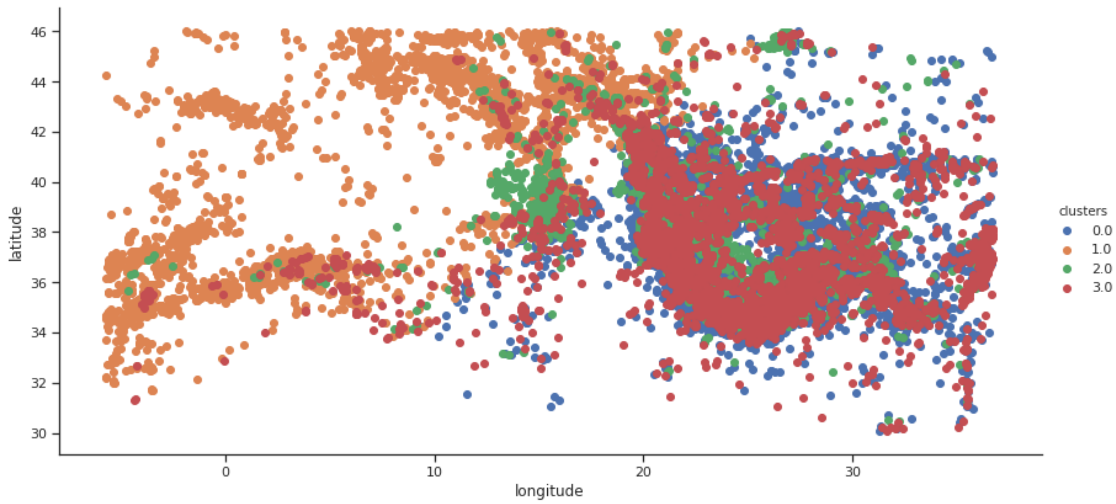
In [182]:
```python
summary_max = df1.groupby('clusters').max()
print(summary_max)
```

```
            year  latitude  longitude  mag   depth
clusters
0.0       2021.0   46.0000    36.7160  4.7  107.00
1.0       2023.0   46.0120    25.2730  6.0  116.50
2.0       2023.0   45.9820    36.4660  7.5  626.20
3.0       2023.0   45.9104    36.7306  7.6  141.14
```

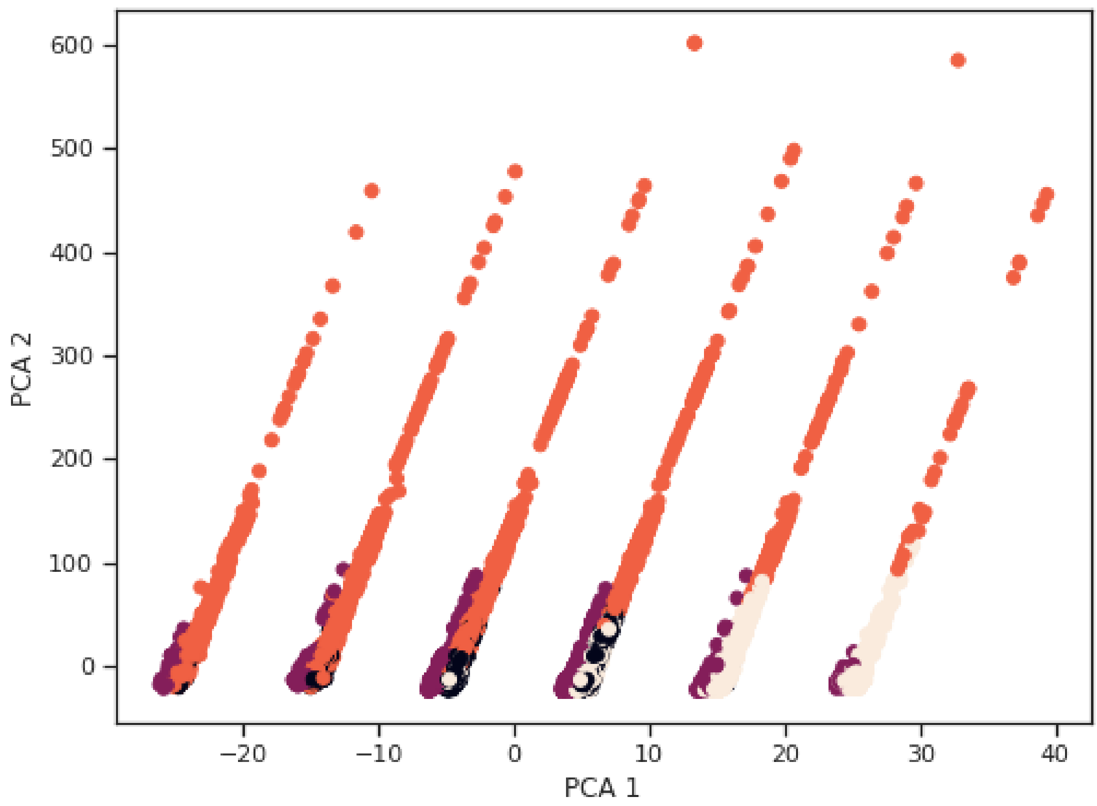
In [183]:
```python
sns.catplot(data = df1, x='clusters', kind='count', height=6, aspect=2)
plt.show()
```



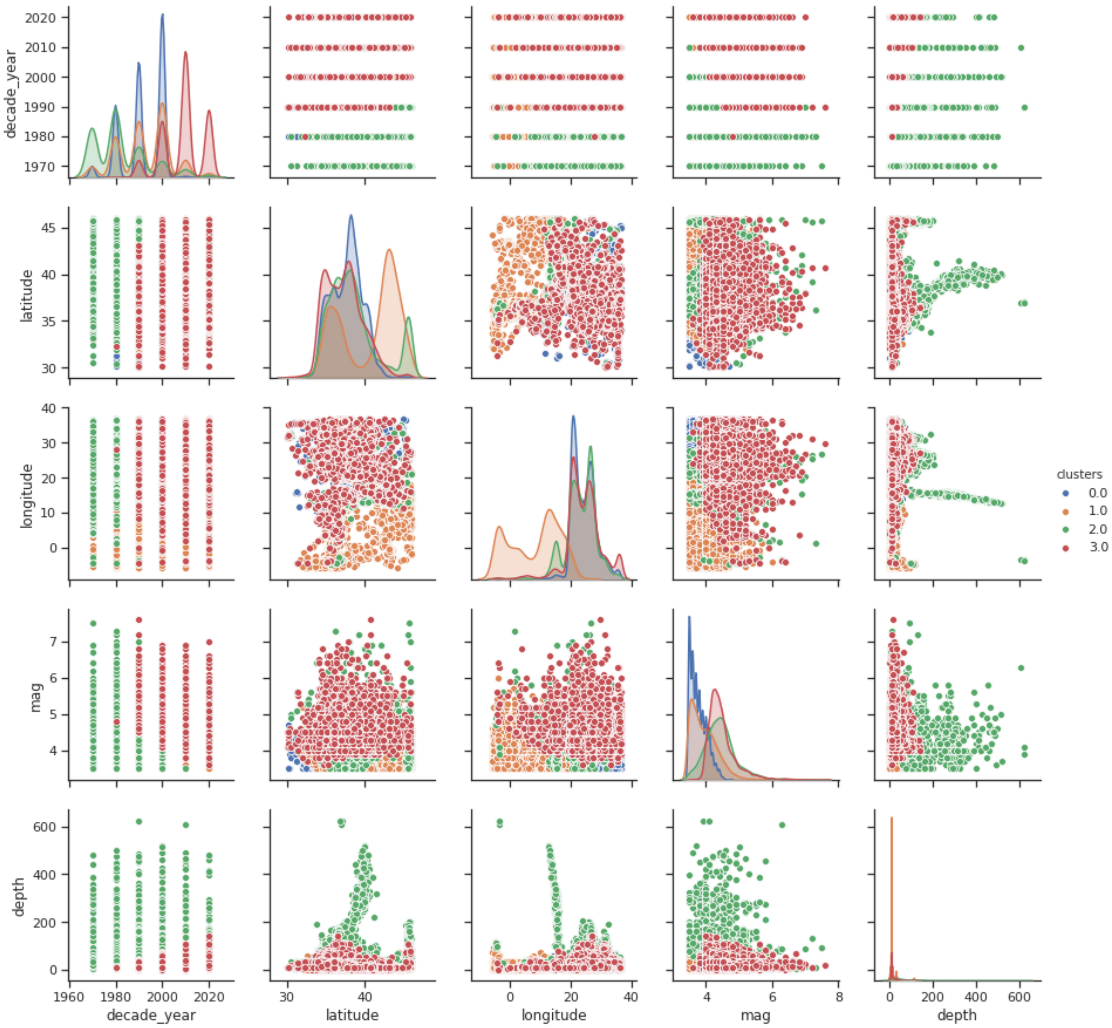
```
In [184]: sns.FacetGrid(df1,hue="clusters", height=6, aspect = 2).map(plt.scatter, 'longitude'

Out[184]: <seaborn.axisgrid.FacetGrid at 0x7fa897716390>
```




```
In [192]: from sklearn.decomposition import PCA
          # create PCA plot
          pca = PCA(n_components=2).fit_transform(df1[['latitude', 'longitude', 'decade_year',

          # Convert the Spark DataFrame to a Pandas DataFrame
          pandas_df = predictions.toPandas()

          # Extract the predicted cluster values as a NumPy array
          predicted_clusters = pandas_df['prediction'].to_numpy()

          plt.figure(figsize=(8,6))
          plt.scatter(pca[:,1], pca[:,0], c = predicted_clusters)
          plt.xlabel('PCA 1')
          plt.ylabel('PCA 2')
          plt.show()
```




```
In [193]: sns.set(style="ticks")
          sns.pairplot(df1, vars=['decade_year','latitude', 'longitude', 'mag','depth'], hue='d
          plt.show()
```




```
In [ ]: # group the dataframe by cluster and depth and count the number of occurrences
        cluster_mag_counts = df1.groupby(['clusters', 'mag']).size().reset_index(name='counts'

        # loop through each cluster and create a bar graph
        for cluster in cluster_mag_counts['clusters'].unique():
            # get the data for the current cluster
            cluster_data = cluster_mag_counts[cluster_mag_counts['clusters'] == cluster]
            # create the bar graph
            plt.figure(figsize=(10,6))
            plt.bar(cluster_data['mag'], cluster_data['counts'], color='blue')
            plt.title(f'Cluster {cluster} Magnitude Counts')
            plt.xlabel('Magnitude')
            plt.ylabel('Counts')
            plt.show()

In [ ]:
```

```
In [ ]:
```