Exploratory Component

In the explorartory part we will attempt to answer a few questions about cities using the same previous cities dataset. We will be investigating the following:

- Predicting gasoline pump price
- Identify country based on transport infrastructure related variables
- Is the CO2 emmission of cities generally dependent on the cities themselves or are they dependent on the country where the city is located in.
- Based on various social/ economic parameters, is it possible to predict the location (coordinates) of the city.

To start we will look at modesharing in a city.

Loading the dataset

In [17]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.formula.api import ols
from sklearn.cluster import KMeans
from geopy.geocoders import Nominatim
from sklearn import decomposition
import seaborn
from  matplotlib import pyplot
from matplotlib.pyplot import figure
%matplotlib inline
import plotly.offline as py
import plotly.graph_objs as go
# Set notebook mode to work in offline
py.init_notebook_mode()
from ast import literal_eval

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)


df_new = pd.read_excel('Cities.xls', index_col=0, skipinitialspace=True) # Read with excel index.
# Skip all white-spaces.
```

In [2]:
```
df_new.reset_index(drop=True, inplace=True)
df_new.head(3)
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Su L
0	Baltimore(MD)	285	7	Auto Sprawl	United States	85.0	6.1	0.3	2.6	0.66	8.5	24.
1	Melbourne	10	8	Auto Innovative	Australia	80.0	14.0	2	4.0	1.11	5.4	0.0
2	Niamey	186	1	Congested Emerging	Niger	NaN	9.0	2	60.0	1.02	26.4	0.0

Importing transport modal shares

First we will perform some modelling and predicting relationships between transportation infrastructure related variables. Modeshares are mostly filled in European countries and USA. With 30 cities in the dataset Chinese cities creates 9% of all observations. The modeshares for Chinese cities will be imported as country's mean values and afterwards scaled, so they sum up to 100%. Same approach is applied on Indian cities. Other big countries like Russia,Mexico, Brazil etc. will have their modeshares imported from external data sources.

Russia

First, lets have a look on Russian cities,

In [3]:
```
df_new.loc[(df_new['Country'] == 'Russia')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	S
9	Yekaterinburg	209	2	BusTransit Sprawl	Russia	NaN	NaN	NaN	NaN	0.73	18.9	1
41	Samara	211	2	BusTransit Sprawl	Russia	NaN	NaN	NaN	NaN	0.71	18.9	1
49	Nizhny Novgorod	210	2	BusTransit Sprawl	Russia	NaN	NaN	NaN	NaN	0.71	18.9	1
230	St. Petersburg	207	2	BusTransit Sprawl	Russia	NaN	NaN	NaN	NaN	0.72	18.9	1
270	Novosibirsk	208	2	BusTransit Sprawl	Russia	NaN	NaN	NaN	NaN	0.70	18.9	1
291	Moscow	206	2	BusTransit Sprawl	Russia	26.0	49.0	NaN	24.0	0.73	7.0	3
316	Kazan	212	2	BusTransit Sprawl	Russia	33.0	67.0	NaN	NaN	0.73	18.9	1

All Russian cities are in same typology category with predominant public transpor Based on statistics of Saint Petersburg modeshares [https://cyberleninka.ru/article/n/analiz-transportnoy-sistemy-sankt-peterburga-i-vozmozhnosti-povysheniya-v-ney-roli-prigorodnyh-zheleznyh-dorog/viewer (https://cyberleninka.ru/article/n/analiz-transportnoy-sistemy-sankt-peterburga-i-vozmozhnosti-povysheniya-v-ney-roli-prigorodnyh-zheleznyh-dorog/viewer)](https://cyberleninka.ru/article/n/analiz-transportnoy-sistemy-sankt-peterburga-i-vozmozhnosti-povysheniya-v-ney-roli-prigorodnyh-zheleznyh-dorog/viewer) And Moscow Modeshares [https://megaobuchalka.ru/12/6677.html (https://megaobuchalka.ru/12/6677.html)](https://megaobuchalka.ru/12/6677.html) Modeshares for other Russian cities will be imputed as average of available observations.

In [4]:
```
#First lets impute Saint Petersburg modeshares to other Russian cities
indices = [9,41,49,230,270,316]
df_new.at[291,['Bicycle Modeshare (%)']]=np.array([0])
df_new.at[indices,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'W
alking Modeshare (%)']]= np.array([34, 64, 1,1])
```

As a former Soviet Union part, Ukraine Cities has same typology as Russian cities. It is reasonable to impute Odessa and Kharkiv modeshares with Russian cities values. For capital city, Kiev modehsares data are imported from: [https://www.slideshare.net/EMBARQNetwork/revision-of-kievs-ground-transport-network-through-data-collection-transforming-transportation-2016 (https://www.slideshare.net/EMBARQNetwork/revision-of-kievs-ground-transport-network-through-data-collection-transforming-transportation-2016)](https://www.slideshare.net/EMBARQNetwork/revision-of-kievs-ground-transport-network-through-data-collection-transforming-transportation-2016)

In [5]:
```
df_new.loc[(df_new['Country'] == 'Ukraine')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Subwa Leng (kr
21	Kharkiv	256	6	Hybrid Giant	Ukraine	NaN	NaN	NaN	NaN	1.11	13.5	38.1
283	Odessa	257	1	Congested Emerging	Ukraine	NaN	NaN	NaN	NaN	1.12	13.5	0.0
305	Kiev	255	6	Hybrid Giant	Ukraine	NaN	NaN	NaN	NaN	1.14	13.5	67.6

In [6]:
```
df_new.at[283,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([34, 64, 1,1])
df_new.at[21,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([34, 64, 1,1])
df_new.at[305,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([28, 37, 0,35])
```

China

According to arcticle https://www.intechopen.com/online-first/the-rise-and-decline-of-car-use-in-beijing-and-shanghai (https://www.intechopen.com/online-first/the-rise-and-decline-of-car-use-in-beijing-and-shanghai) car traffic growth in two major Chinese cities , Beijing and Shangai, has already reached its peak values. Because majority of available modalshares are in similar ranges (for cars around 20%), it seems reasonable to impute modalshares on the basis of mean value of Chinese cities with subjective rounding so the final sum is 100%

In [7]:
```
df_china=df_new.loc[(df_new['Country'] == 'China')]
df_china
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	S L
4	Urumqi	67	12	MetroBike Emerging	China	21.70	54.70	NaN	NaN	1.16	18.8	0.(
16	Hefei	68	12	MetroBike Emerging	China	42.00	24.60	2.7	NaN	1.16	18.8	24
19	Dalian	63	12	MetroBike Emerging	China	NaN	43.00	NaN	NaN	1.16	18.8	14
32	Chongqing	47	11	MetroBike Giant	China	20.60	32.60	NaN	46.30	1.18	18.8	20
54	Chengdu	51	12	MetroBike Emerging	China	11.00	15.00	NaN	NaN	1.16	18.8	10
77	Wuhan	48	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	12
78	Changsha	66	12	MetroBike Emerging	China	NaN	NaN	3.2	NaN	1.16	18.8	50
83	Jinan	60	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	0.(
85	Shenzhen	46	11	MetroBike Giant	China	19.30	16.70	6.2	50.00	1.19	18.8	23
89	Guangzhou	45	11	MetroBike Giant	China	21.00	32.00	9	38.00	1.19	18.8	24
124	Harbin	53	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	17
129	Shenyang	54	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	55
147	Xiamen	71	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	0.(
149	Beijing	44	11	MetroBike Giant	China	21.00	26.00	32	21.00	1.17	4.4	55
151	Zhengzhou	57	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	45
154	Taiyuan	61	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	0.(
166	Shijiazhuang	70	12	MetroBike Emerging	China	5.00	7.16	47.28	26.82	1.16	18.8	0.(
182	Hangzhou	55	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	81
186	Fuzhou	69	12	MetroBike Emerging	China	NaN	NaN	37	26.00	1.16	18.8	24
193	Qingdao	58	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	12
212	Kunming	62	12	MetroBike Emerging	China	22.10	25.20	55.3	27.50	1.16	18.8	40
221	Tianjin	49	12	MetroBike Emerging	China	27.80	42.00	13.85	14.27	1.16	18.8	13
249	Changchun	59	12	MetroBike Emerging	China	20.00	33.00	10	27.00	1.16	18.8	50
271	Nanjing	52	12	MetroBike Emerging	China	NaN	NaN	39	20.00	1.16	18.8	22
272	Suzhou	64	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	66
292	Xi,Äôan	56	12	MetroBike Emerging	China	11.88	36.26	37.6	16.60	1.16	18.8	90
298	Shanghai	43	11	MetroBike Giant	China	20.00	33.00	20	27.00	1.16	3.8	58
303	Dongguan	50	12	MetroBike Emerging	China	40.00	60.00	NaN	NaN	1.16	18.8	0.(
310	Ningbo	72	12	MetroBike Emerging	China	15.00	66.00	2	15.00	1.16	18.8	74
321	Wuxi	65	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	56

In [8]:
```
car =df_new.loc[df_new['Country'] == 'China', 'Car Modeshare (%)'].mean()

publictransit =df_new.loc[df_new['Country'] == 'China', 'Public Transit Modeshare (%)'].mean()

bike=df_new.loc[df_new['Country'] == 'China', 'Bicycle Modeshare (%)'].mean()

walking = df_new.loc[df_new['Country'] == 'China', 'Walking Modeshare (%)'].mean()

print(car,publictransit,bike,walking)
```

```
21.22533333333333 34.20125 22.509285714285713 27.345384615384617
```

In [9]:
```
#Identifying indexes of missing values
pd.DataFrame(df_china[df_china['Car Modeshare (%)'].isna() &df_china['Bicycle Modeshare (%)'].isna
() & df_china['Walking Modeshare (%)'].isna() & df_china['Public Transit Modeshare (%)'].isna() ])
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Sul Le
77	Wuhan	48	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	128
83	Jinan	60	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	0.0
124	Harbin	53	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	17.5
129	Shenyang	54	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	55.1
147	Xiamen	71	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	0.0
151	Zhengzhou	57	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	45.3
154	Taiyuan	61	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	0.0
182	Hangzhou	55	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	81.5
193	Qingdao	58	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	12.0
272	Suzhou	64	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	66.1
321	Wuxi	65	12	MetroBike Emerging	China	NaN	NaN	NaN	NaN	1.16	18.8	56.0

In [10]:
```
#imputing rows with nan values

indices = [77,83,124,129,147,151,154,182,193,272,321]
df_new.at[indices,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([21, 34, 22,23])
#imputing values
df_new.at[4,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([21.7, 54.7, 18,6.6])
df_new.at[32,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([20.6, 32.6, 0.5,46.3])
df_new.at[54,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([11, 15, 44,20])
df_new.at[16,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([42, 24.6, 2.7,19.3])
df_new.at[19,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([21, 43, 20,16])
df_new.at[78,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([23.8, 43, 20,3.2])
df_new.at[187,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([17, 23, 37,26])
df_new.at[272,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([17, 24, 39,20])
df_new.at[304,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([40, 60, 0,0])
```

In [11]:
```
df_new.loc[(df_new['Country'] == 'China')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	S L
4	Urumqi	67	12	MetroBike Emerging	China	21.70	54.70	18	6.60	1.16	18.8	0.
16	Hefei	68	12	MetroBike Emerging	China	42.00	24.60	2.7	19.30	1.16	18.8	24
19	Dalian	63	12	MetroBike Emerging	China	21.00	43.00	20	16.00	1.16	18.8	14
32	Chongqing	47	11	MetroBike Giant	China	20.60	32.60	0.5	46.30	1.18	18.8	20
54	Chengdu	51	12	MetroBike Emerging	China	11.00	15.00	44	20.00	1.16	18.8	10
77	Wuhan	48	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	12
78	Changsha	66	12	MetroBike Emerging	China	23.80	43.00	20	3.20	1.16	18.8	50
83	Jinan	60	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	0.
85	Shenzhen	46	11	MetroBike Giant	China	19.30	16.70	6.2	50.00	1.19	18.8	23
89	Guangzhou	45	11	MetroBike Giant	China	21.00	32.00	9	38.00	1.19	18.8	24
124	Harbin	53	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	17
129	Shenyang	54	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	55
147	Xiamen	71	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	0.
149	Beijing	44	11	MetroBike Giant	China	21.00	26.00	32	21.00	1.17	4.4	55
151	Zhengzhou	57	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	45
154	Taiyuan	61	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	0.
166	Shijiazhuang	70	12	MetroBike Emerging	China	5.00	7.16	47.28	26.82	1.16	18.8	0.
182	Hangzhou	55	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	81
186	Fuzhou	69	12	MetroBike Emerging	China	NaN	NaN	37	26.00	1.16	18.8	24
193	Qingdao	58	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	12
212	Kunming	62	12	MetroBike Emerging	China	22.10	25.20	55.3	27.50	1.16	18.8	40
221	Tianjin	49	12	MetroBike Emerging	China	27.80	42.00	13.85	14.27	1.16	18.8	13
249	Changchun	59	12	MetroBike Emerging	China	20.00	33.00	10	27.00	1.16	18.8	50
271	Nanjing	52	12	MetroBike Emerging	China	NaN	NaN	39	20.00	1.16	18.8	22
272	Suzhou	64	12	MetroBike Emerging	China	17.00	24.00	39	20.00	1.16	18.8	66
292	Xi,Äôan	56	12	MetroBike Emerging	China	11.88	36.26	37.6	16.60	1.16	18.8	90
298	Shanghai	43	11	MetroBike Giant	China	20.00	33.00	20	27.00	1.16	3.8	58
303	Dongguan	50	12	MetroBike Emerging	China	40.00	60.00	NaN	NaN	1.16	18.8	0.
310	Ningbo	72	12	MetroBike Emerging	China	15.00	66.00	2	15.00	1.16	18.8	74
321	Wuxi	65	12	MetroBike Emerging	China	21.00	34.00	22	23.00	1.16	18.8	56

Imputing modal shares for Indian cities

WE can use same methodology to impute modeshares for Indian cities, first lets find mean modal shares

In [12]:
```
car =df_new.loc[df_new['Country'] == 'India', 'Car Modeshare (%)'].mean()
bike=df_new.loc[df_new['Country'] == 'India', 'Bicycle Modeshare (%)'].mean()
publictransit =df_new.loc[df_new['Country'] == 'India', 'Public Transit Modeshare (%)'].mean()
walking = df_new.loc[df_new['Country'] == 'India', 'Walking Modeshare (%)'].mean()
modes_india_mean = [car, publictransit, bike, walking]
print(*modes_india_mean)
```

```
31.875 27.25 9.875 27.375
```

In [13]:
```
df_india=df_new.loc[(df_new['Country'] == 'India')]
df_india.head()
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Sub Le(
10	Pune	120	3	Congested Boomer	India	54.0	12.0	11	22.0	1.19	5.2	0.0
15	Delhi	113	3	Congested Boomer	India	19.0	42.0	12	21.0	1.14	9.1	213.(
35	Bangalore	117	3	Congested Boomer	India	25.0	35.0	7	26.0	1.19	8.9	31.5
80	Mumbai	114	3	Congested Boomer	India	15.0	45.0	6	27.0	1.27	3.2	11.4
127	Lucknow	122	1	Congested Emerging	India	NaN	NaN	NaN	NaN	1.19	16.6	0.0

In [14]:
```
df_new.at[10,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([32, 28, 10,30])
df_new.at[15,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([32, 28, 10,30])
df_new.at[35,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([32, 28, 10,30])
df_new.at[127,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([32, 28, 10,30])
```

Mexico

In [15]:
```
df_new.loc[(df_new['Country'] == 'Mexico')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Su L
47	Acapulco	173	4	BusTransit Dense	Mexico	NaN	67.2	NaN	NaN	0.86	17.7	0.C
58	Chihuahua	174	2	BusTransit Sprawl	Mexico	63.4	14.4	NaN	NaN	0.86	21.0	0.C
87	Mexico City	165	4	BusTransit Dense	Mexico	20.7	71.3	1	NaN	0.93	11.7	22(
107	Tijuana	169	2	BusTransit Sprawl	Mexico	NaN	NaN	NaN	NaN	0.89	15.3	0.C
210	Leon	171	4	BusTransit Dense	Mexico	27.0	33.9	NaN	NaN	0.88	21.0	0.C
229	Toluca	170	2	BusTransit Sprawl	Mexico	19.0	50.0	1	30.0	0.85	18.0	0.C
239	Puebla	168	2	BusTransit Sprawl	Mexico	45.0	40.6	NaN	NaN	0.86	16.7	0.C
257	Guadalajara	166	2	BusTransit Sprawl	Mexico	27.0	28.0	NaN	NaN	0.92	21.1	0.C
269	Ciudad Juarez	172	2	BusTransit Sprawl	Mexico	59.0	36.1	NaN	NaN	0.71	21.0	0.C
323	Monterrey	167	2	BusTransit Sprawl	Mexico	41.2	54.5	NaN	NaN	0.95	15.7	32.

In [16]:
```
car =df_new.loc[df_new['Country'] == 'Mexico', 'Car Modeshare (%)'].mean()
publictransit =df_new.loc[df_new['Country'] == 'Mexico', 'Public Transit Modeshare (%)'].mean()
bike=df_new.loc[df_new['Country'] == 'Mexico', 'Bicycle Modeshare (%)'].mean()
walking = df_new.loc[df_new['Country'] == 'Mexico', 'Walking Modeshare (%)'].mean()
modes_mexico_mean = [car, publictransit, bike, walking]
print(*modes_mexico_mean)
```

```
37.7875 44.00000000000001 1.0 30.0
```

In [17]:
```
df_new.at[47,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([30, 67.2, 1,2])
df_new.at[58,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([63.4, 14.6, 1,20])
df_new.at[107,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([38, 44, 1,17])
df_new.at[210,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([27, 34, 7,32])
df_new.at[239,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([45, 40.6, 4.4,10])
df_new.at[257,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([27, 28, 15,30])
df_new.at[269,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([59.0, 36.1, 1.9,3])
df_new.at[323,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([41.2, 54.5, 1,2.8])
```

Middle East

In Saudi Arabia, Bicycle and Walking is basically no option because of heat, lets increase pt share to 8% and impute 92% for car and 8% for public transport. Sharjah mode shares based on these statistics from Dubai: [https://www.statista.com/statistics/725806/dubai-share-of-motorized-trips-by-transport-mode/ (https://www.statista.com/statistics/725806/dubai-share-of-motorized-trips-by-transport-mode/)](https://www.statista.com/statistics/725806/dubai-share-of-motorized-trips-by-transport-mode/)

In [18]:
```
df_new.loc[(df_new['Country'] == 'Saudi Arabia')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Subwa Lengt (kn
122	Riyadh	214	2	BusTransit Sprawl	Saudi Arabia	92.0	2.0	NaN	NaN	0.54	27.4	0.0
174	Medina	216	2	BusTransit Sprawl	Saudi Arabia	NaN	NaN	NaN	NaN	0.54	27.4	0.0
180	Mecca	215	2	BusTransit Sprawl	Saudi Arabia	NaN	NaN	NaN	NaN	0.54	27.4	18.1
181	Jeddah	217	2	BusTransit Sprawl	Saudi Arabia	NaN	NaN	NaN	NaN	0.54	27.4	0.0

In [19]:
```
df_new.at[122,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([92, 8, 0,0])
df_new.at[174,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([92, 8, 0,0])
df_new.at[180,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([92, 8, 0,0])
df_new.at[181,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([92, 8, 0,0])
df_new.at[190,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([85.6, 14.4, 0,0])
```

Brazil

In [20]:
```
df_new.loc[(df_new['Country'] == 'Brazil')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Subv Len (l
79	Belo Horizonte	23	4	BusTransit Dense	Brazil	32.6000	28.1000	0.4	34.80	1.33	22.5	28.1
121	Sao Paulo	21	4	BusTransit Dense	Brazil	27.6000	39.0000	0.6	NaN	1.22	11.8	77.4
146	Brasilia	25	4	BusTransit Dense	Brazil	32.3675	37.2125	2.79	25.92	1.31	20.9	42.4
191	Salvador	24	1	Congested Emerging	Brazil	22.9000	44.2300	NaN	32.26	1.34	16.7	11.9
281	Rio de Janeiro	22	4	BusTransit Dense	Brazil	17.6900	45.3600	2.42	29.36	1.43	16.7	58.0
284	Recife	26	4	BusTransit Dense	Brazil	29.4400	44.3500	NaN	23.43	1.30	23.4	39.5

In [21]:
```
#These values are rough estimate
df_new.at[121,['Walking Modeshare (%)']] = 22.8
df_new.at[191,['Bicycle Modeshare (%)']] = 0.6
df_new.at[284,['Bicycle Modeshare (%)']] = 2.78
```

Turkey

Modeshares obtained from https://core.ac.uk/download/pdf/158369769.pdf (https://core.ac.uk/download/pdf/158369769.pdf), page 59 for Ankara, İstanbul and İzmir, rest will be imputed as mean values for these cities

In [22]:
```
df_new.loc[(df_new['Country'] == 'Turkey')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Subwa Leng (kr
44	Bursa	252	6	Hybrid Giant	Turkey	NaN	NaN	NaN	NaN	1.47	8.5	38.9
62	Istanbul	249	4	BusTransit Dense	Turkey	14.0	41.0	NaN	45.0	1.46	4.0	95.3
268	Adana	253	5	Hybrid Moderate	Turkey	NaN	NaN	NaN	NaN	1.47	11.4	13.9
328	Izmir	251	5	Hybrid Moderate	Turkey	NaN	NaN	NaN	NaN	1.47	8.3	0.0
329	Ankara	250	6	Hybrid Giant	Turkey	NaN	NaN	NaN	NaN	1.48	7.5	64.3

In [23]:
```
df_new.at[44,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([25, 56, 1,18])
df_new.at[268,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([25, 56, 1,18])
df_new.at[62,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([23, 54, 1,22])
df_new.at[328,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([22, 56, 3,19])
df_new.at[329,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([30, 57, 1,12])
```

Iran

Reculculation from Tehran:https://www.researchgate.net/publication/317591351_Iran_the_Urban_Transport_Crisis_in_Emerging_Economies (https://www.researchgate.net/publication/317591351_Iran_the_Urban_Transport_Crisis_in_Emerging_Economies) Isfahan: https://www.researchgate.net/publication/316273459_A_Statistical_Appraisal_of_Bus_Rapid_Transit_Based_on_Passengers_ (https://www.researchgate.net/publication/316273459_A_Statistical_Appraisal_of_Bus_Rapid_Transit_Based_on_Passengers Mashhad: https://www.researchgate.net/publication/319175220_Mode_Choice_Model_for_the_Elderly_Case_of_Mashhad/link/5b7856b (https://www.researchgate.net/publication/319175220_Mode_Choice_Model_for_the_Elderly_Case_of_Mashhad/link/5b7856b Shiraz: http://www.ccsenet.org/journal/index.php/jsd/article/download/33371/19246 (http://www.ccsenet.org/journal/index.php/jsd/article/download/33371/19246)

In [24]:
```
df_new.loc[(df_new['Country'] == 'Iran')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Subv Len (k
95	Tabriz	131	2	BusTransit Sprawl	Iran	32.0	23.0	NaN	3.0	0.36	32.1	7.0
222	Isfahan	130	2	BusTransit Sprawl	Iran	NaN	NaN	NaN	NaN	0.36	32.1	11.2
282	Shiraz	132	2	BusTransit Sprawl	Iran	NaN	NaN	NaN	NaN	0.36	32.1	10.5
294	Tehran	128	2	BusTransit Sprawl	Iran	35.0	13.0	1.5	36.0	0.36	32.1	178.0
317	Mashhad	129	2	BusTransit Sprawl	Iran	56.0	25.0	NaN	3.0	0.36	32.1	24.0

In [25]:
```
df_new.at[221,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([28.3, 65.7, 1,5])
df_new.at[282,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([53, 43, 0,7])
df_new.at[294,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([39, 48, 3,10])
df_new.at[317,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([25, 43, 2,30])

#Tabriz has unrealistic values, lets impute from means
car =df_new.loc[df_new['Country'] == 'Iran', 'Car Modeshare (%)'].mean()
publictransit =df_new.loc[df_new['Country'] == 'Iran', 'Public Transit Modeshare (%)'].mean()
bike=df_new.loc[df_new['Country'] == 'Iran', 'Bicycle Modeshare (%)'].mean()
walking = df_new.loc[df_new['Country'] == 'Iran', 'Walking Modeshare (%)'].mean()
modes_iran_mean = [car, publictransit, bike, walking]
print(*modes_iran_mean)

#To have sum to 100%, added 4% to car and public trans shares
df_new.at[95,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walking Modeshare (%)']]= np.array([40, 48, 1,11])
```

```
37.25 39.25 1.6666666666666667 12.5
```

South Korea

Korean statistics from : https://english.koti.re.kr/component/file/ND_fileDownload.do?q_fileSn=100633&q_fileId=c212f509-87df-42e6-b239-5db822995ae4

In [26]:
```
df_new.loc[(df_new['Country'] == 'South Korea')]
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Sub Le
38	Ulsan	233	7	Auto Sprawl	South Korea	NaN	NaN	NaN	NaN	1.47	9.1	0.0
125	Busan	229	6	Hybrid Giant	South Korea	NaN	NaN	NaN	NaN	1.47	9.1	130.
205	Daegu	230	6	Hybrid Giant	South Korea	NaN	NaN	NaN	NaN	1.47	9.1	81.2
255	Gwangju	232	6	Hybrid Giant	South Korea	36.0	69.0	0	0.0	1.47	9.1	20.1
260	Seoul-Incheon	228	9	MassTransit Heavyweight	South Korea	23.1	65.6	NaN	NaN	1.47	9.1	331.
277	Daejeon	231	6	Hybrid Giant	South Korea	44.0	28.0	2	26.0	1.47	9.1	22.7

In [27]:
```
df_new.at[38,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walkin
g Modeshare (%)']]= np.array([45.1, 26.2, 1.8,25.2])
df_new.at[125,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([29.8, 47.3, 0.9,23.5])
df_new.at[205,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([37.4, 30.7, 2.6,27.2])
df_new.at[255,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([40.5, 30.9, 1.2,26.4])
df_new.at[260,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([23.4, 66.6, 1.7,8.3])
df_new.at[277,['Car Modeshare (%)','Public Transit Modeshare (%)', 'Bicycle Modeshare (%)', 'Walki
ng Modeshare (%)']]= np.array([42.3, 27.7, 1.9,26.5])
```

For the rest of NAN values, linear interpolation is performed

In [28]:
```
df_new[['Car Modeshare (%)', 'Public Transit Modeshare (%)', 'Walking Modeshare (%)', 'Bicycle Mod
eshare (%)']]=df_new[['Car Modeshare (%)', 'Public Transit Modeshare (%)', 'Walking Modeshare (%)'
, 'Bicycle Modeshare (%)']].interpolate(method='linear', limit_direction='forward', axis=0)
```

Enriching the Dataset with Coordinates (Latitude and Longitude)

In our models, we will use geographical coordinates. In the following function, we use geopy plugin, to add coordinates for each city in a new column.

In [29]:
```
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="user_agent")

def add_coordinates(city):
    """
    This functon will add long. lat. coordinates for each city as a new column
    """

    if city != str:
        city = str(city)
    if city == "Baltimore(MD)":
        city = "Baltimore"
    if city == "Birmingham(AL)":
        city = "Birmingham"
    if city == "Valencia(VZL)":
        city = "Valencia"
    if city == "Tampa-St. Petersburg(FL)":
        city = "St. Petersburg, Florida"
    if city == "Denver-Aurora(CO)":
        city = "Aurora, Colorado"
    geolocator = Nominatim(user_agent="user_agent")
#     print(city)
    while True:
        try:
            location = geolocator.geocode(city)
            break
        except:
            continue

    if type(location)!=type(None):
        lat_long = (location.latitude, location.longitude)
    else:
        lat_long = np.nan
    return lat_long
    ### your code
```

- Then we apply our function to the dataset:

In [30]:
```
# Apply the add_coordinates function to the dataframe to produce coordinates for each city
df_new["coordinates"] = df_new["City"].apply(add_coordinates)
df_new[['City', 'coordinates']].head(3)
```

- Now we have a new column that is called coordinates. But it has both latitude and longitude combined. So we need to separate those values into two new column. We perform the following operations to achieve this goal.

In [31]:
```
#splitting coordinates to lattitude and longitude

df_new = df_new[df_new['coordinates']!='nan'] # To make sure that there's no null values amongst t
hat geopy returned.
df_new["coordinates"]= df_new["coordinates"].map(str)

split_data = df_new["coordinates"].str.split(", ")
data = split_data.to_list()
names = ["lattitude", "longitude"]
coordinates = pd.DataFrame(data, columns=names)
coordinates['lattitude'] = coordinates['lattitude'] .str[1:].astype(float)
coordinates['longitude'] = coordinates['longitude'] .str[:-1].astype(float)
coordinates.drop_duplicates()
df_new.drop_duplicates()
df_new = df_new[~df_new.index.duplicated()]
coordinates = coordinates[~coordinates.index.duplicated()]
coordinates['City']=df_new['City']
coordinates.head(3)
```

- As you can see, now we have latitude and longitude values in separate columns for each city now. **Please note that there was a mistake in the writing of "lattitude" but the code is written that way so we will keep it as it is.**

Importing Weather Variables:

Now we will import the weather variables that are going to be used in our Weather Anomalies Analysis.

In [32]:
```
# %pip install pyowm
```

In the following cells, using pyowm module, we will get contemporary temperature and humidity values for each city, with using geographical coordinates that we have found previously and enrich our dataset with the values obtained.

In [33]:
```
# get the temperature values
from pyowm import OWM
from pyowm.utils import config
from pyowm.utils import timestamps
owm = OWM('3951c3b4517f5b0f874efcee811b7571')
mgr = owm.weather_manager()
temperaturelist = []
for index, x in coordinates.iterrows():
    one_call = mgr.one_call(lat=float(x[0]), lon=float(x[1]))
    temperaturelist.append(str(one_call.forecast_daily[0].temperature('celsius').get('feels_like_m
orn', None)))

# get the humidity values
from pyowm import OWM
from pyowm.utils import config
from pyowm.utils import timestamps
owm = OWM('3951c3b4517f5b0f874efcee811b7571')
mgr = owm.weather_manager()
humiditylist = []
for index, x in coordinates.iterrows():
    one_call = mgr.one_call(lat=float(x[0]), lon=float(x[1]))
    humiditylist.append(str(one_call.current.humidity))

# Creating columns from lists
coordinates['Humidity'] = humiditylist
coordinates['Temperature'] = temperaturelist
coordinates.dropna()
```

- And we will save the final version of the dataset into a file for ease of use in the following analyses.

In [34]:
```
#Create a pickle file to backup dataframe
df_new.to_pickle("./df_new.pkl")
```

Gasoline Pump Price Analysis

Based on selected transportation infrastructure related variables, we are going to analyze the relationship between gasoline pump price and predict its values for different cities afterwards.

Used variables will be:

- Road Deaths Rate (per 1000)
- City
- Car Modeshare (%)
- Public Transit Modeshare (%)
- Bicycle Modeshare (%)
- Walking Modeshare (%)
- GDP per Capita (USD)
- Population
- Subway Length (km)
- Subway Length Density (per km)
- Subway Stations per Hundred Thousand
- Subway Ridership per Capita Subway Age (years)
- BRT Length (km)
- BRT System Length Density (per km)
- BRT Stations per Hundred Thousand Persons
- BRT Fleet per Hundred Thousand Persons
- BRT Annual Ridership per Capita BRT Age (years)
- Bikeshare Stations
- Congestion (%)
- Street length total (m)
- Street Length Density (m/sq. km)
- Intersection Count
- Vehicles per capita

In [35]:
```
df = pd.read_pickle("df_new.pkl")
#Data obtained from https://en.wikipedia.org/wiki/List_of_countries_by_vehicles_per_capita
vehiclespercapita = pd.read_csv('vehiclespercapita.csv',sep= ',',  thousands=',') # Read with exce
l index.
vehiclespercapita.head()
```

	#	Country or region	Motor vehiclesper 1,000 people	Total	Year
0	1	San Marino	1263	NaN	2013[5]
1	2	Monaco	899	NaN	2013[5]
2	3	New Zealand	860	4,240,000	2018[6]
3	4	United States	838	273,602,100[7]	2018
4	5	Iceland	824	278,924[8][9]	2016

In [36]:
```
vehiclespercapita['Motor vehiclesper 1,000 people'] = vehiclespercapita['Motor vehiclesper 1,000 p
eople'].astype(float)
vehiclespercapita['Cars per capita']=vehiclespercapita['Motor vehiclesper 1,000 people'].div(1000)
col2drop = ['Total', 'Year', '#','Motor vehiclesper 1,000 people']
vehiclespercapita.drop(col2drop, inplace=True, axis=1)
df = pd.merge(left=df, right=vehiclespercapita, how='left', left_on='Country', right_on='Country o
r region')
df.drop(['Country or region'], inplace=True, axis=1)
df.head()
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Su L
0	Baltimore(MD)	285	7	Auto Sprawl	United States	85.0	6.1	0.3	2.6	0.66	8.5	24.
1	Melbourne	10	8	Auto Innovative	Australia	80.0	14.0	2	4.0	1.11	5.4	0.0
2	Niamey	186	1	Congested Emerging	Niger	44.0	9.0	2	60.0	1.02	26.4	0.0
3	Hanoi	328	12	MetroBike Emerging	Vietnam	8.0	10.0	2	33.3	0.90	24.5	0.0
4	Urumqi	67	12	MetroBike Emerging	China	21.7	54.7	18	6.6	1.16	18.8	0.0

In [37]:
```
#Before any modelling begin it is neccesarry to take care of missing values
temp_df = pd.DataFrame(df.isnull().sum(axis=0), columns=['Missing Values'])/df.count()[0]*100
temp_df = temp_df.sort_values(by=temp_df.columns[0], ascending=False)
pd.set_option('display.max_rows', 1000)
temp_df.head(15)
cols2drop = temp_df[temp_df['Missing Values']>=25].index
df = df.drop(cols2drop, axis=1)
df.head(3)
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Subway Length (km)	Subw Leng Densi (p ki
0	Baltimore(MD)	285	7	Auto Sprawl	United States	85.0	6.1	2.6	0.66	8.5	24.9	0.0134
1	Melbourne	10	8	Auto Innovative	Australia	80.0	14.0	4.0	1.11	5.4	0.0	0.0000
2	Niamey	186	1	Congested Emerging	Niger	44.0	9.0	60.0	1.02	26.4	0.0	0.0000

In [38]:
```
#First lets use linear interpolation to fill missing values
df.interpolate(method ='linear', limit_direction ='forward', inplace=True)
```

The plan for the Gasoline pump price predictive model is following:

1) Select all transport infrastructure related variables + socio-economic and CO2 emissions per capita

2) Perform dimension reduction

3) Used reduced set as explanatory variables

4) Select Gasoline pump price as dependent variable

5) Perform regression or other suitable model

In [39]:
```
df_gasoline= df[['CO2 Emissions per Capita (metric tonnes)','GDP per Capita (USD)','Gasoline Pump
 Price (USD/liter)','Cars per capita','Street length total (m)', 'Street Length Density (m/sq. k
m)',\
'Street Length Average (m)', 'Intersection Count', 'Intersection Density (per sq. km)',\
'Degree Average', 'Streets per Node', 'Circuity', 'Self-Loop Proportion', 'Highway Proportion',\
'Metro Propensity Factor', 'BRT Propensity Factor', 'BikeShare Propensity Factor', 'Development Fa
ctor',\
 'Congestion Factor','Sprawl Factor', 'Network Density Factor']]
```

Let's take a look at how our correlation matrix looks like.

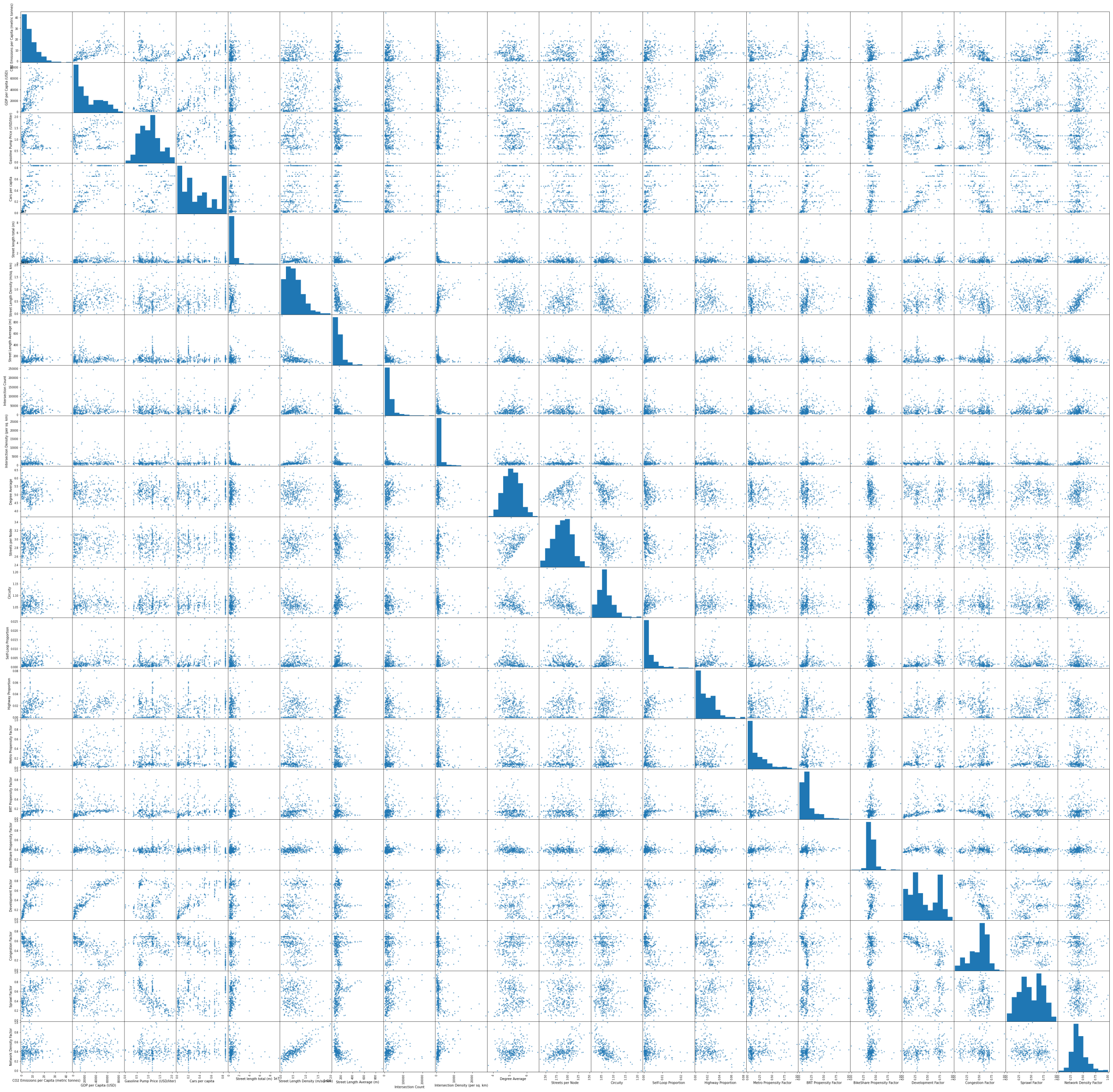
In [40]:
```
df_gasoline.corr()
```

	CO2 Emissions per Capita (metric tonnes)	GDP per Capita (USD)	Gasoline Pump Price (USD/liter)	Cars per capita	Street length total (m)	Street Length Density (m/sq. km)	Street Length Average (m)	Intersection Count	Intersection Density (per sq. km)	Degree Average	Stre N
CO2 Emissions per Capita (metric tonnes)	1.000000	0.701319	-0.143898	0.643845	0.043629	0.158597	0.093778	-0.011806	-0.004577	-0.153627	0.1254
GDP per Capita (USD)	0.701319	1.000000	0.174626	0.804273	0.053302	0.314590	-0.097854	0.059811	0.059067	-0.192285	0.0702
Gasoline Pump Price (USD/liter)	-0.143898	0.174626	1.000000	0.003229	-0.071845	0.075583	-0.021978	-0.065740	0.131954	-0.256270	-0.040
Cars per capita	0.643845	0.804273	0.003229	1.000000	0.029970	0.293457	-0.103364	0.031917	0.016519	-0.044548	0.0558
Street length total (m)	0.043629	0.053302	-0.071845	0.029970	1.000000	-0.160688	0.172659	0.895085	-0.239287	0.093450	0.0012
Street Length Density (m/sq. km)	0.158597	0.314590	0.075583	0.293457	-0.160688	1.000000	-0.437986	0.045149	0.565208	0.006270	0.2322
Street Length Average (m)	0.093778	-0.097854	-0.021978	-0.103364	0.172659	-0.437986	1.000000	-0.113118	-0.222806	-0.218080	0.0011
Intersection Count	-0.011806	0.059811	-0.065740	0.031917	0.895085	0.045149	-0.113118	1.000000	-0.240922	0.190718	0.0297
Intersection Density (per sq. km)	-0.004577	0.059067	0.131954	0.016519	-0.239287	0.565208	-0.222806	-0.240922	1.000000	-0.091068	0.1461
Degree Average	-0.153627	-0.192285	-0.256270	-0.044548	0.093450	0.006270	-0.218080	0.190718	-0.091068	1.000000	0.4241
Streets per Node	0.125445	0.070206	-0.040018	0.055877	0.001226	0.232271	0.001192	0.029716	0.146172	0.424123	1.0000
Circuity	0.006564	0.051006	0.111907	-0.014143	0.185586	-0.323661	0.234719	0.070021	-0.145773	-0.378995	-0.457
Self-Loop Proportion	0.320641	0.464750	0.059551	0.405176	-0.019337	0.082002	-0.044174	-0.045693	0.055915	-0.194239	-0.207
Highway Proportion	0.394573	0.415347	0.086366	0.329766	0.107816	-0.035574	0.501157	-0.036134	-0.087017	-0.378186	0.1678
Metro Propensity Factor	0.103467	0.316525	0.353949	0.140524	0.080948	0.104155	0.036749	0.110824	-0.094544	-0.209512	0.1420
BRT Propensity Factor	0.132038	0.275518	0.081295	0.227784	0.083507	0.229682	-0.128044	0.132547	0.089611	-0.159879	0.0584
BikeShare Propensity Factor	-0.007367	0.010004	0.233189	-0.044142	0.020835	0.114205	0.142457	0.005195	0.029913	-0.272746	0.1076
Development Factor	0.706203	0.935653	0.184467	0.843008	0.042572	0.352054	-0.098554	0.048077	0.122868	-0.217158	0.0598
Congestion Factor	-0.702860	-0.707596	0.081666	-0.758575	0.012202	-0.250125	0.060478	0.046693	-0.110825	0.066482	-0.024
Sprawl Factor	0.459980	0.134466	-0.720925	0.241927	0.320404	-0.133333	0.382972	0.220875	-0.292512	0.022176	0.1069
Network Density Factor	0.129987	0.271707	0.021159	0.271680	-0.062710	0.832578	-0.478496	0.138719	0.488623	0.229463	0.4533

In [41]:
```
#To see if the data are suitable for dimension reduction, lets count correlated features
correlated_features = set()
correlation_matrix = df_gasoline.corr()
for i in range(len(correlation_matrix .columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > 0.7:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)

print('Here we perform a quick check if there are variable that are highly correlated.')
print('There are {} pair of variables that are correlated with over 0.7 correlation score' \
      .format(len(correlated_features)))
```

```
Here we perform a quick check if there are variable that are highly correlated.
There are 7 pair of variables that are correlated with over 0.7 correlation score
```

In [42]:
```
#to have visual inputs of relationships between variables, plot scatter matrix
pd.plotting.scatter_matrix(df_gasoline,figsize=(60,60), alpha=0.8);
```



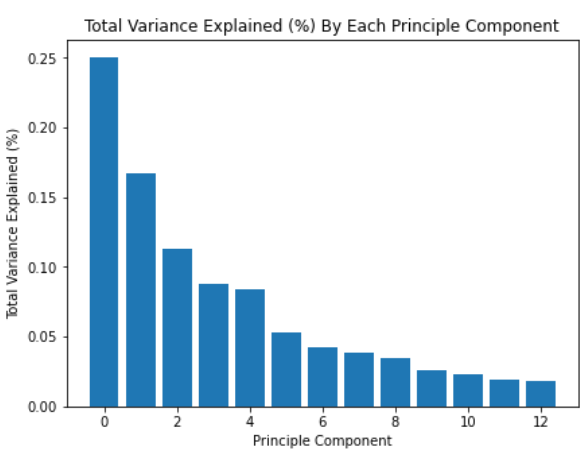
In the plots above (**you can double click to zoom in**), we see have many variables that seem to be correlated. Since correlated input variables can cause poorer performance increasing the bias of the model over these variables, while also increasing the time needed for the training, we need to handle them. Here we decide that it's best if we use PCA, because there so many columns and PCA by definition handles the correlated columns in a way that the outputted principle components are always orthogonal to each other.

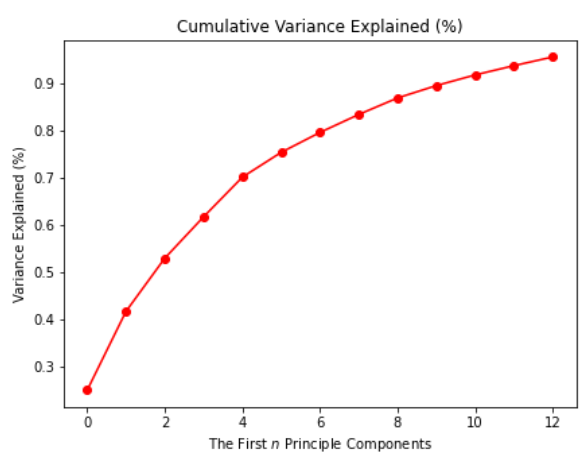
Below we will first standardize the dataset (as PCA requires it's inputs to be) then, we will perform a PCA to decide how what components that we will use.

In [43]:
```
from sklearn import decomposition
# Define the standardization function.
def standardize_dataframe(df_in):
    return (df_in-df_in.mean())/df_in.std()

# PCA inputs without filtering:
df_target = df_gasoline['Gasoline Pump Price (USD/liter)']
df_inputs = standardize_dataframe(df_gasoline.drop('Gasoline Pump Price (USD/liter)', axis=1))
```

In [44]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=.95)

pca.fit(df_inputs)
expl=pca.explained_variance_ratio_
cdf=[sum(expl[:i+1]) for i in range(len(expl))]

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15,5))

ax[0].bar(range(len(expl)),pca.explained_variance_ratio_)
ax[0].set_xlabel('Principle Component')
ax[0].set_ylabel('Total Variance Explained (%)')
ax[0].set_title('Total Variance Explained (%) By Each Principle Component')


ax[1].plot(range(len(expl)), cdf, marker='o', color='r');
ax[1].set_xlabel('The First $n$ Principle Components')
ax[1].set_ylabel('Variance Explained (%)')
ax[1].set_title('Cumulative Variance Explained (%)')
```

```
Text(0.5, 1.0, 'Cumulative Variance Explained (%)')
```





As you can see even by using 12 of the first principle components we are able to explain 95% of the total variance in the dataset.

We will use the first 10 components, which means 50% reduction dimensionality reduction. In the cell below we apply PCA on our standardized inputs and we create train and test splits.

In [45]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_inputs, df_target, test_size=0.25, random_s
tate=42)
```

We have used 25% of our dataset as test set, while using the other 75% as our training samples. Shuffling was open so the values found could change slightly everytime the code is run.

We tried Random Forest Regression and Linear Regression mainly. Tried to optimize the random forest model. But interestingly, a simple model like ordinary Linear Regression which needs no tuning seemed to perform similiar to the optimized random forest model.

In [46]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor


lin = LinearRegression()
lin.fit(X_train, y_train)

lin_R2 = lin.score(X_test, y_test)


rf = RandomForestRegressor(max_depth=15, min_samples_split=2, n_estimators=150)
rf.fit(X_train, y_train)

rf_R2 = rf.score(X_test, y_test)

print('R2-score obtained by Linear Regression: {}'.format(lin_R2))
print('R2-score obtained by Random Forest Regression: {}'.format(rf_R2))
```

```
R2-score obtained by Linear Regression: 0.7853661904853352
R2-score obtained by Random Forest Regression: 0.7893500536590882
```

Classification of Countries Based on Tansport Infrastructure Data

Lets reformulate the question to a classification problem. Based on, modeshares, transport, infrastructure and socio-economic statistics, can we identify the country? With what accuracy?

- Since the prediction models cannot run with string type data, we need to encode our Country values numerically. In the following cell we perform this operation with a function we have written:

In [47]:
```
from sklearn.preprocessing import LabelEncoder

def multi_label_encoder(df, cols2code):
        encoder = LabelEncoder()
        for col in cols2code:
            df[col+'(Encoded)'] = encoder.fit_transform(df[col])
        return df

df = multi_label_encoder(df, ['Country'])
df[['Country', 'Country(Encoded)']].head()
```

	Country	Country(Encoded)
0	United States	116
1	Australia	5
2	Niger	77
3	Vietnam	120
4	China	21

In [48]:
```
from sklearn.model_selection import train_test_split

transport_analysis = df[['Country(Encoded)','Gasoline Pump Price (USD/liter)','Car Modeshare (%)',
'Public Transit Modeshare (%)',\
                              'Walking Modeshare (%)', 'Subway Length (km)', 'BRT Length (km)',\
                              'Street length total (m)', 'Road Deaths Rate (per 1000)', 'GDP per Capita
(USD)']]

transport_analysis_nonan = df[['Country(Encoded)','Gasoline Pump Price (USD/liter)','Car Modeshare
(%)', 'Public Transit Modeshare (%)',\
                              'Walking Modeshare (%)', 'Subway Length (km)', 'BRT Length (km)',\
                              'Street length total (m)', 'Road Deaths Rate (per 1000)', 'GDP per Capita
(USD)']].dropna()

transport_analysis_target = transport_analysis_nonan['Country(Encoded)'].astype(int)
transport_analysis_inputs = transport_analysis_nonan.drop('Country(Encoded)' , axis = 1)
```

In [49]:
```
x_train,x_test,y_train,y_test = train_test_split(transport_analysis_inputs,transport_analysis_targ
et, test_size = 0.25 )
```

In [50]:
```
from sklearn.linear_model import SGDClassifier

regr = SGDClassifier()
regr.fit(x_train,y_train)
regr.score(x_test,y_test)
```

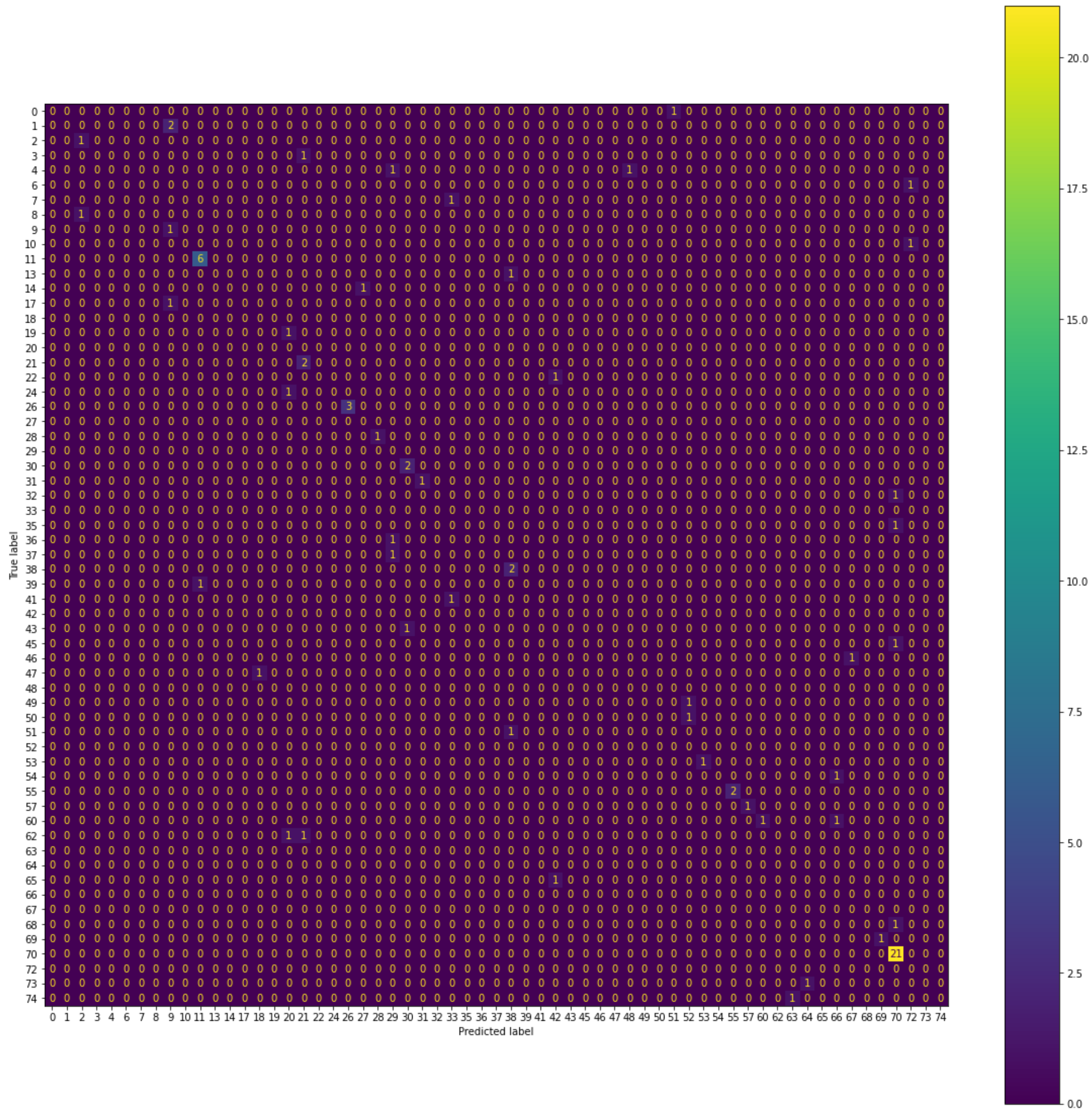
```
0.25301204819277107
```

In [51]:
```
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n_estimators=100, max_depth=15, random_state = 0)
rf.fit(x_train, y_train)
y_pred=rf.predict(x_test)
print(rf.score(x_test, y_test))
```

```
0.5542168674698795
```

In [52]:
```
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

fig, ax = plt.subplots(figsize=(20,20))
plot_confusion_matrix(rf, x_test, y_test, ax=ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x170e6616820>
```



In [53]:
```
# Getting the score for our model
rf.score(x_test, y_test)
```

```
0.5542168674698795
```

CO2 emmission of cities VS fossil fuel consumption of countries

In the prediction part of the project we attempted to merely predict the CO2 emmision per capita of a city. We are however curious about what impacts this CO2 emmission. We have found data which shows the total fossil fuel consumption per capita of every country over the last 50 years. We take the average of each country over the last three years and see if the highest fossil fuel consuming countries also have cities with high CO2 emmision. In other words we want to find out, is it the city itself which contributes to its own CO2 emission or is it the country which the city lies within which is contributing to how much the city emmites. We found the data for how much fossil fuel each country consumes per capita at the following link: [https://ourworldindata.org/fossil-fuels (https://ourworldindata.org/fossil-fuels)](https://ourworldindata.org/fossil-fuels).

In [54]:

```
fossil_fuels=pd.read_csv("fossil-fuels-per-capita.csv")
g = fossil_fuels[fossil_fuels['Year']>2015].groupby(fossil_fuels.Entity).mean()
consumption = list(g['Fossil fuels per capita (kWh)'])
column_names = ["Country", "Fossil fuel per capita (10MWh)"]

df = pd.DataFrame(columns=column_names)
df['Country'] = fossil_fuels['Entity'].unique()
df['Fossil fuel per capita (10MWh)'] = list(np.array(consumption)*1e-4)
df = df.sort_values(by=['Fossil fuel per capita (10MWh)'], ascending=False).head(10)
df.head()
```

	Country	Fossil fuel per capita (10MWh)
57	Qatar	20.039897
61	Singapore	17.193791
72	Trinidad and Tobago	14.475155
76	United Arab Emirates	13.759359
39	Kuwait	10.995292

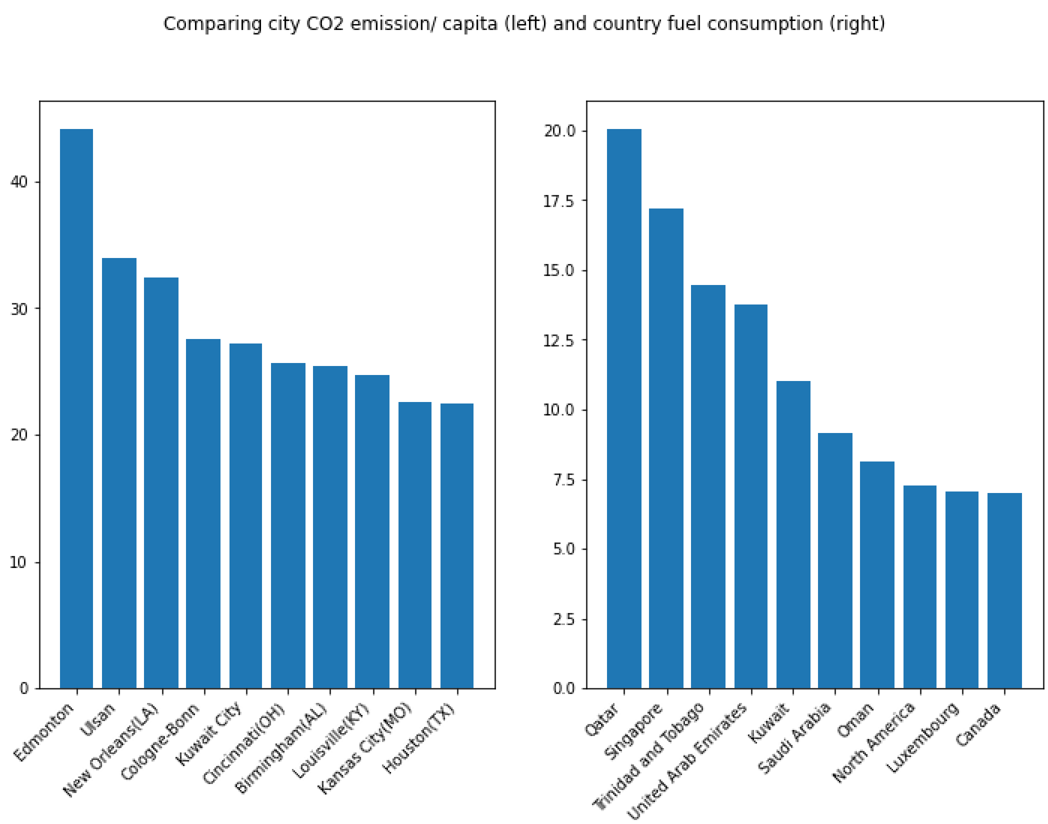
In [55]:

```
df2 = pd.read_excel("FINAL-COMBINED-DATASET.xlsx")
df2 = df2[['City', 'CO2 Emissions per Capita (metric tonnes)']]
df2 = df2.sort_values(by=['CO2 Emissions per Capita (metric tonnes)'], ascending=False).head(10)
df2.head()
```

	City	CO2 Emissions per Capita (metric tonnes)
36	Edmonton	44.100000
232	Ulsan	33.900000
310	New Orleans(LA)	32.400000
100	Cologne-Bonn	27.600000
153	Kuwait City	27.258964

In [56]:
```
# plot both
# df = df.set_index('Country')
# df2 = df2.set_index('City')
x1 = df2['City']
y1 = df2['CO2 Emissions per Capita (metric tonnes)']
x2 = df['Country']
y2 = df['Fossil fuel per capita (10MWh)']
fig, axs = plt.subplots(1,2, figsize=(12,8))
fig.autofmt_xdate(rotation=45)
fig.suptitle('Comparing city CO2 emission/ capita (left) and country fuel consumption (right)')
axs[0].bar(x1, y1)
axs[1].bar(x2,y2)
```

```
<BarContainer object of 10 artists>
```



The plot to the left are most polluting cities (CO2 emmission) per capita and on the right are the countries with the highest fossil fuel consuming countries per capita. There are a few of the highest polluting cities in the highest fuel consuming coutries which would indicate that there is a connection between the city and country. However if you look at the top four cities closly to see why they emmite so much CO2, you will find some interesting facts such as, Edmonton has some of the largest petrochemical and metal/ machinary industries in Canada. Ulsan is home to Korea's Hyundai motor manufacturing plant. New Orleans has some of the largest oil/ gas producers in the USA. Cologne-Bonn is one of euope's largest automotive industry.

So when acessing the CO2 emission of a city, the largest parameter which impacts emission is probably the unique activites/ industries which are located in the city itself and not solely the country which the city is located in.

Predicting city location

In this section we will attempt to find out if a cities coordinates impact various social parameters of a given city. We will also fit a model to some the social parameters and see if it is possible to predict with a certain accuracy the location of the city. We will start by finding the coordinates of all cities and then showing some parameter visualizations.

In [57]:
```
from geopy.geocoders import Nominatim
import seaborn as sns
from  matplotlib import pyplot
from matplotlib.pyplot import figure
from ast import literal_eval
import plotly.graph_objects as go
```

In [58]:
```
df = pd.read_excel('Cities.xls', index_col=0, skipinitialspace=True) # Read with excel index.

# def add_coordinates(city):
#     """
#     This functon will add long. lat. coordinates for each city as a new column
#     """

#     if city != str:
#         city = str(city)
#     if city == "Baltimore(MD)":
#         city = "Baltimore"
#     if city == "Birmingham(AL)":
#         city = "Birmingham"
#     if city == "Valencia(VZL)":
#         city = "Valencia"
#     if city == "Tampa-St. Petersburg(FL)":
#         city = "St. Petersburg, Florida"
#     if city == "Denver-Aurora(CO)":
#         city = "Aurora, Colorado"
#     geolocator = Nominatim(user_agent="user_agent")
# #     print(city)
#     while True:
#         try:
#             location = geolocator.geocode(city)
#             break
#         except:
#             continue

#     if type(location)!=type(None):
#         lat_long = (location.latitude, location.longitude)
#     else:
#         lat_long = np.nan
#     return lat_long
#     ### your code


# df["coordinates"] = df["City"].apply(add_coordinates)
# The function above takes a long time to run so we run it once
# and save the full set with coordinates as a csv for fast loading later

df = pd.read_csv('location_from_pickle.csv') # Previous saved dataset
```

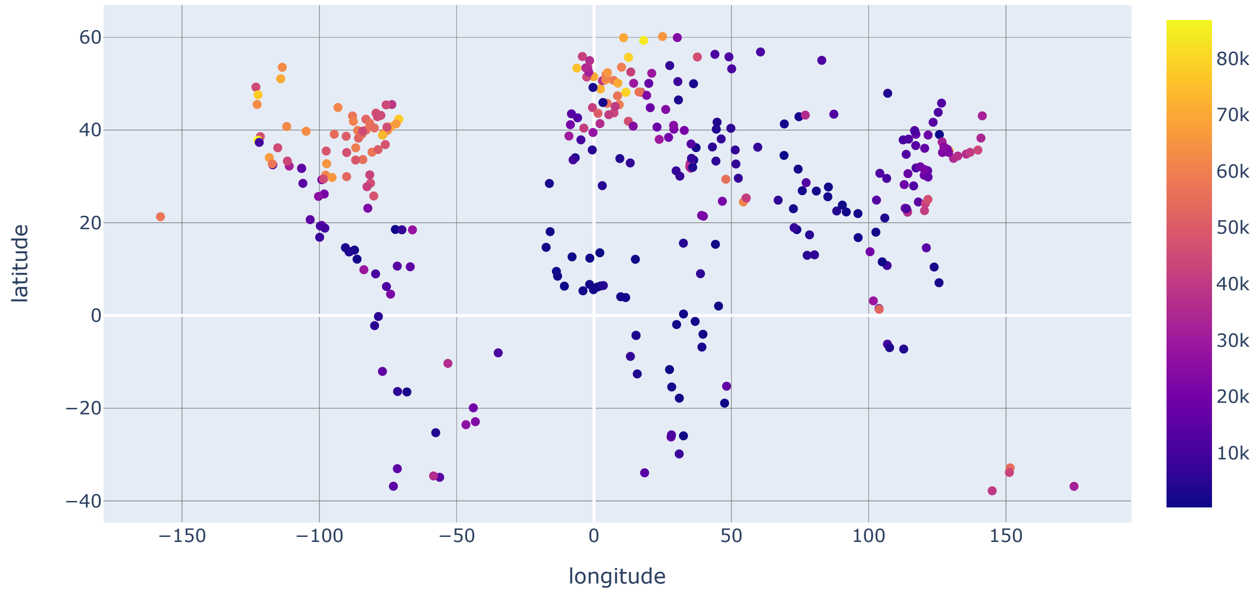
Now that we have the coordinates for all cities. We will proceed to try with predicting location, to start with we will see if we can predict the distance a city is from the equator using some social and economic parameters.

In [59]:
```
# Function to define distance in km from equater
def eq_dist(coord):
    # 1 degree is about 111.045 km
    dist = literal_eval(coord)[0]*111.045
    return int(abs(dist)) # Make distances south of equater positive.

df["eq_dist[km]"] = df["coordinates"].apply(eq_dist)
# Make sure everything looks ok.
df[['City', 'cityID', 'Country', 'coordinates', 'eq_dist[km]']].head()
```

	City	cityID	Country	coordinates	eq_dist[km]
0	Baltimore(MD)	285.0	United States	(39.2908816, -76.610759)	4363
1	Melbourne	10.0	Australia	(-37.8142176, 144.9631608)	4199
2	Niamey	186.0	Niger	(13.524834, 2.109823)	1501
3	Hanoi	328.0	Vietnam	(21.0294498, 105.8544441)	2335
4	Urumqi	67.0	China	(43.419754, 87.319461)	4821

Before making any models, lets visualize a much discussed parameter for each city, namely GDP/ capita.

In [60]:
```
# apply lat/ long to lists to further plot
lat_list = []
long_list = []
for i in df['coordinates']:
    lat_list.append(literal_eval(i)[0])
    long_list.append(literal_eval(i)[1])
# long_list
z = df['GDP per Capita (USD)']

# Make plot
import chart_studio.plotly as py
import plotly.graph_objects as go

scatter = go.Scatter(x=np.array(long_list).flatten(),
                     y=np.array(lat_list).flatten(),
                     marker={'color': np.array(z).flatten(),
                             'showscale': True},
                     mode='markers')
fig = go.FigureWidget(data=[scatter],
                      layout={'xaxis': {'title': 'longitude'},
                              'yaxis': {'title': 'latitude'}})
go.Figure(fig)
```



From the plot above showing the coordinates ploted with intensity being GDP per capita, it looks like northern cities have a higher GDP (along with Oceania regions). This is somewhat expected but lets investigate this further and see if the relationship between location and along with other parameters can be modeled.

In [61]:
```
# Start by choosing relevant features and normalizing them.
features = ['Population Density (per sq. km)', 'Urbanization Rate 2015 (%)', 'GDP per Capita (US
D)', 'Life Expectancy (years)']
X = df[features]
# X.isnull().sum()
X = X.fillna(X.mean())
X_n = (X-X.mean())/X.std()
y = df['eq_dist[km]']
X_n.shape

from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_n, y, test_size=0.33, random_state=123)
# define the keras model
model = Sequential()
model.add(Dense(500, input_dim=X_n.shape[1], activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(200, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(1, activation='linear'))

# compile the keras model
model.compile(loss='mse', optimizer='adam')

# fit the keras model on the dataset
model.fit(X_train, y_train, epochs=100, batch_size=100, validation_data=(X_test, y_test))

# evaluate the keras model
pred = model.predict(X_test)

# evaluate predictions
print("\nMAE=%f" % mean_absolute_error(y_test, pred))
print("\nRMSE=%f" % np.sqrt(mean_squared_error(y_test, pred)))
print("r^2=%f" % r2_score(y_test, pred))
```

```
Epoch 1/100
3/3 [==============================] - 0s 47ms/step - loss: 15173648.0000 - val_loss: 13601580.0000
Epoch 2/100
3/3 [==============================] - 0s 7ms/step - loss: 15168084.0000 - val_loss: 13596322.0000
Epoch 3/100
3/3 [==============================] - 0s 8ms/step - loss: 15162112.0000 - val_loss: 13590170.0000
Epoch 4/100
3/3 [==============================] - 0s 8ms/step - loss: 15154702.0000 - val_loss: 13582668.0000
Epoch 5/100
3/3 [==============================] - 0s 10ms/step - loss: 15146616.0000 - val_loss: 13573523.0000
Epoch 6/100
3/3 [==============================] - 0s 7ms/step - loss: 15136236.0000 - val_loss: 13562463.0000
Epoch 7/100
3/3 [==============================] - 0s 8ms/step - loss: 15123085.0000 - val_loss: 13549123.0000
Epoch 8/100
3/3 [==============================] - 0s 7ms/step - loss: 15106871.0000 - val_loss: 13533135.0000
Epoch 9/100
3/3 [==============================] - 0s 9ms/step - loss: 15089235.0000 - val_loss: 13514084.0000
Epoch 10/100
3/3 [==============================] - 0s 10ms/step - loss: 15067197.0000 - val_loss: 13491536.0000
Epoch 11/100
3/3 [==============================] - 0s 7ms/step - loss: 15043075.0000 - val_loss: 13465066.0000
Epoch 12/100
3/3 [==============================] - 0s 10ms/step - loss: 15010151.0000 - val_loss: 13433991.0000
Epoch 13/100
3/3 [==============================] - 0s 8ms/step - loss: 14971811.0000 - val_loss: 13397626.0000
Epoch 14/100
3/3 [==============================] - 0s 8ms/step - loss: 14933085.0000 - val_loss: 13355722.0000
Epoch 15/100
3/3 [==============================] - 0s 6ms/step - loss: 14882900.0000 - val_loss: 13307718.0000
Epoch 16/100
3/3 [==============================] - 0s 9ms/step - loss: 14834960.0000 - val_loss: 13253305.0000
Epoch 17/100
3/3 [==============================] - 0s 8ms/step - loss: 14769202.0000 - val_loss: 13191290.0000
Epoch 18/100
3/3 [==============================] - 0s 7ms/step - loss: 14699668.0000 - val_loss: 13121456.0000
Epoch 19/100
3/3 [==============================] - 0s 7ms/step - loss: 14618290.0000 - val_loss: 13042269.0000
Epoch 20/100
3/3 [==============================] - 0s 8ms/step - loss: 14534698.0000 - val_loss: 12953996.0000
Epoch 21/100
3/3 [==============================] - 0s 9ms/step - loss: 14429344.0000 - val_loss: 12857289.0000
Epoch 22/100
3/3 [==============================] - 0s 7ms/step - loss: 14308519.0000 - val_loss: 12749177.0000
Epoch 23/100
3/3 [==============================] - 0s 7ms/step - loss: 14194844.0000 - val_loss: 12630083.0000
Epoch 24/100
3/3 [==============================] - 0s 8ms/step - loss: 14076369.0000 - val_loss: 12499423.0000
Epoch 25/100
3/3 [==============================] - 0s 6ms/step - loss: 13905159.0000 - val_loss: 12356544.0000
Epoch 26/100
3/3 [==============================] - 0s 8ms/step - loss: 13755389.0000 - val_loss: 12200630.0000
Epoch 27/100
3/3 [==============================] - 0s 8ms/step - loss: 13590669.0000 - val_loss: 12033285.0000
Epoch 28/100
3/3 [==============================] - 0s 9ms/step - loss: 13390390.0000 - val_loss: 11853789.0000
Epoch 29/100
3/3 [==============================] - 0s 7ms/step - loss: 13191623.0000 - val_loss: 11662106.0000
Epoch 30/100
3/3 [==============================] - 0s 8ms/step - loss: 12974887.0000 - val_loss: 11456475.0000
Epoch 31/100
3/3 [==============================] - 0s 9ms/step - loss: 12702404.0000 - val_loss: 11237107.0000
Epoch 32/100
3/3 [==============================] - 0s 8ms/step - loss: 12483095.0000 - val_loss: 11003920.0000
Epoch 33/100
3/3 [==============================] - 0s 6ms/step - loss: 12210890.0000 - val_loss: 10758420.0000
Epoch 34/100
3/3 [==============================] - 0s 8ms/step - loss: 11936913.0000 - val_loss: 10501723.0000
Epoch 35/100
3/3 [==============================] - 0s 8ms/step - loss: 11629890.0000 - val_loss: 10233150.0000
Epoch 36/100
3/3 [==============================] - 0s 8ms/step - loss: 11341063.0000 - val_loss: 9954333.0000
Epoch 37/100
3/3 [==============================] - 0s 8ms/step - loss: 11068485.0000 - val_loss: 9664862.0000
Epoch 38/100
3/3 [==============================] - 0s 8ms/step - loss: 10687719.0000 - val_loss: 9365201.0000
```

```
Epoch 39/100
3/3 [==============================] - 0s 8ms/step - loss: 10413430.0000 - val_loss: 9058561.0000
Epoch 40/100
3/3 [==============================] - 0s 8ms/step - loss: 9993368.0000 - val_loss: 8748268.0000
Epoch 41/100
3/3 [==============================] - 0s 9ms/step - loss: 9676005.0000 - val_loss: 8433082.0000
Epoch 42/100
3/3 [==============================] - 0s 9ms/step - loss: 9292641.0000 - val_loss: 8114562.5000
Epoch 43/100
3/3 [==============================] - 0s 9ms/step - loss: 8889613.0000 - val_loss: 7790269.0000
Epoch 44/100
3/3 [==============================] - 0s 8ms/step - loss: 8498052.0000 - val_loss: 7464506.0000
Epoch 45/100
3/3 [==============================] - 0s 8ms/step - loss: 8176718.0000 - val_loss: 7141693.5000
Epoch 46/100
3/3 [==============================] - 0s 8ms/step - loss: 7815803.5000 - val_loss: 6823285.0000
Epoch 47/100
3/3 [==============================] - 0s 7ms/step - loss: 7478374.0000 - val_loss: 6509602.5000
Epoch 48/100
3/3 [==============================] - 0s 8ms/step - loss: 7025665.0000 - val_loss: 6200668.0000
Epoch 49/100
3/3 [==============================] - 0s 7ms/step - loss: 6629802.0000 - val_loss: 5902510.5000
Epoch 50/100
3/3 [==============================] - 0s 7ms/step - loss: 6443238.5000 - val_loss: 5620054.0000
Epoch 51/100
3/3 [==============================] - 0s 8ms/step - loss: 6070475.5000 - val_loss: 5351845.0000
Epoch 52/100
3/3 [==============================] - 0s 9ms/step - loss: 5817370.5000 - val_loss: 5094692.5000
Epoch 53/100
3/3 [==============================] - 0s 7ms/step - loss: 5384618.5000 - val_loss: 4856221.5000
Epoch 54/100
3/3 [==============================] - 0s 6ms/step - loss: 5137859.0000 - val_loss: 4635887.0000
Epoch 55/100
3/3 [==============================] - 0s 7ms/step - loss: 5039167.0000 - val_loss: 4431482.0000
Epoch 56/100
3/3 [==============================] - 0s 6ms/step - loss: 4771931.0000 - val_loss: 4241795.0000
Epoch 57/100
3/3 [==============================] - 0s 8ms/step - loss: 4479904.5000 - val_loss: 4070493.7500
Epoch 58/100
3/3 [==============================] - 0s 7ms/step - loss: 4322060.0000 - val_loss: 3915717.2500
Epoch 59/100
3/3 [==============================] - 0s 7ms/step - loss: 4134080.5000 - val_loss: 3775723.7500
Epoch 60/100
3/3 [==============================] - 0s 8ms/step - loss: 3896886.7500 - val_loss: 3652457.2500
Epoch 61/100
3/3 [==============================] - 0s 9ms/step - loss: 3695478.5000 - val_loss: 3543201.2500
Epoch 62/100
3/3 [==============================] - 0s 7ms/step - loss: 3662007.2500 - val_loss: 3449708.2500
Epoch 63/100
3/3 [==============================] - 0s 7ms/step - loss: 3568496.2500 - val_loss: 3367098.7500
Epoch 64/100
3/3 [==============================] - 0s 7ms/step - loss: 3475916.5000 - val_loss: 3292746.0000
Epoch 65/100
3/3 [==============================] - 0s 6ms/step - loss: 3356115.0000 - val_loss: 3229640.7500
Epoch 66/100
3/3 [==============================] - 0s 7ms/step - loss: 3337071.5000 - val_loss: 3174673.5000
Epoch 67/100
3/3 [==============================] - 0s 6ms/step - loss: 3090863.0000 - val_loss: 3124566.5000
Epoch 68/100
3/3 [==============================] - 0s 7ms/step - loss: 3159714.2500 - val_loss: 3079889.7500
Epoch 69/100
3/3 [==============================] - 0s 8ms/step - loss: 3088199.5000 - val_loss: 3040727.2500
Epoch 70/100
3/3 [==============================] - 0s 7ms/step - loss: 3047477.2500 - val_loss: 3006488.7500
Epoch 71/100
3/3 [==============================] - 0s 8ms/step - loss: 2988796.2500 - val_loss: 2974248.7500
Epoch 72/100
3/3 [==============================] - 0s 10ms/step - loss: 2910701.0000 - val_loss: 2943226.5000
Epoch 73/100
3/3 [==============================] - 0s 9ms/step - loss: 2828850.2500 - val_loss: 2913899.0000
Epoch 74/100
3/3 [==============================] - 0s 8ms/step - loss: 2823510.7500 - val_loss: 2885495.5000
Epoch 75/100
3/3 [==============================] - 0s 8ms/step - loss: 2880807.7500 - val_loss: 2859773.0000
Epoch 76/100
3/3 [==============================] - 0s 12ms/step - loss: 2803723.0000 - val_loss: 2834002.5000
```

```
Epoch 77/100
3/3 [==============================] - 0s 8ms/step - loss: 2750382.7500 - val_loss: 2807609.0000
Epoch 78/100
3/3 [==============================] - 0s 8ms/step - loss: 2776062.0000 - val_loss: 2778871.7500
Epoch 79/100
3/3 [==============================] - 0s 8ms/step - loss: 2732750.7500 - val_loss: 2752189.7500
Epoch 80/100
3/3 [==============================] - 0s 7ms/step - loss: 2657832.0000 - val_loss: 2728167.5000
Epoch 81/100
3/3 [==============================] - 0s 10ms/step - loss: 2738200.5000 - val_loss: 2704087.5000
Epoch 82/100
3/3 [==============================] - 0s 8ms/step - loss: 2612772.5000 - val_loss: 2679515.7500
Epoch 83/100
3/3 [==============================] - 0s 9ms/step - loss: 2532682.5000 - val_loss: 2655120.0000
Epoch 84/100
3/3 [==============================] - 0s 8ms/step - loss: 2586814.2500 - val_loss: 2629614.7500
Epoch 85/100
3/3 [==============================] - 0s 8ms/step - loss: 2473992.0000 - val_loss: 2603662.0000
Epoch 86/100
3/3 [==============================] - 0s 8ms/step - loss: 2546432.2500 - val_loss: 2578325.5000
Epoch 87/100
3/3 [==============================] - 0s 6ms/step - loss: 2474910.7500 - val_loss: 2554548.7500
Epoch 88/100
3/3 [==============================] - 0s 8ms/step - loss: 2601568.2500 - val_loss: 2529200.0000
Epoch 89/100
3/3 [==============================] - 0s 9ms/step - loss: 2439259.0000 - val_loss: 2503092.2500
Epoch 90/100
3/3 [==============================] - 0s 7ms/step - loss: 2382862.7500 - val_loss: 2477533.0000
Epoch 91/100
3/3 [==============================] - 0s 6ms/step - loss: 2518417.7500 - val_loss: 2453764.7500
Epoch 92/100
3/3 [==============================] - 0s 7ms/step - loss: 2413398.5000 - val_loss: 2433561.5000
Epoch 93/100
3/3 [==============================] - 0s 6ms/step - loss: 2318732.0000 - val_loss: 2414948.2500
Epoch 94/100
3/3 [==============================] - 0s 7ms/step - loss: 2368704.0000 - val_loss: 2398932.2500
Epoch 95/100
3/3 [==============================] - 0s 6ms/step - loss: 2162982.5000 - val_loss: 2383393.0000
Epoch 96/100
3/3 [==============================] - 0s 6ms/step - loss: 2336412.0000 - val_loss: 2368133.2500
Epoch 97/100
3/3 [==============================] - 0s 7ms/step - loss: 2324701.5000 - val_loss: 2353537.7500
Epoch 98/100
3/3 [==============================] - 0s 7ms/step - loss: 2316692.0000 - val_loss: 2338211.2500
Epoch 99/100
3/3 [==============================] - 0s 7ms/step - loss: 2132690.5000 - val_loss: 2321708.0000
Epoch 100/100
3/3 [==============================] - 0s 7ms/step - loss: 2251171.0000 - val_loss: 2306369.5000

MAE=1257.538493

RMSE=1518.673624
r^2=0.210660
```

Mean absolute error (MAE) for our model is about 1140 km which is about 10.4 degrees when it is evaluated on the test set (1/3 of total dataset). That means that given the 4 features used ('Population Density (per sq. km)', 'Urbanization Rate 2015 (%)', 'GDP per Capita (USD)', 'Life Expectancy (years)') we are able to predict the distance the city is from the equater within an error of 10.4 degrees! This is quite a small distance, see the map figure below for reference, the horizontal lines are spaced 15 degrees apart.

alt text

So now that we have seen that we can model the distance from the equator, lets take a step further and try to model the real coordinates of cities (latitude/ longitude). We will also try doing classification of the country regions, i.e. predict which continent the city is located. This time we will try to further analize the features which are used in the prediction model.

In [62]:
```python
# Again start by loading dataset with pre-calcualated city coordinates
df = pd.read_csv('location_from_pickle.csv')
df.describe()
```

	Unnamed: 0	cityID	clusterID	Car Modeshare (%)	Public Transit Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Subway Length (km)	Subway Length Density (per km)	S S H Th
count	332.000000	332.000000	332.000000	268.000000	269.000000	255.000000	332.000000	331.000000	332.000000	332.000000	332
mean	165.500000	165.801205	5.578313	45.611819	28.133913	17.510196	1.056024	14.674622	39.041416	0.038099	0.7
std	95.984374	95.619666	3.591263	27.787198	20.310731	14.842958	0.425800	8.726269	77.236480	0.068863	1.2
min	0.000000	1.000000	1.000000	0.000000	0.400000	0.000000	0.010000	0.600000	0.000000	0.000000	0.0
25%	82.750000	83.750000	2.000000	21.525000	11.000000	3.200000	0.707500	7.500000	0.000000	0.000000	0.0
50%	165.500000	165.500000	6.000000	38.000000	28.000000	17.000000	1.055000	13.900000	0.000000	0.000000	0.0
75%	248.250000	248.250000	8.000000	68.775000	42.000000	26.200000	1.322500	20.400000	42.825000	0.053334	1.1
max	331.000000	331.000000	12.000000	94.800000	82.500000	78.000000	2.120000	37.200000	588.000000	0.612982	9.7

Above you can see the main statistical features for each column. The dataset consists of 332 cities as it is right now. Since I saved the coordinates in a formatted string in a single column in the dataframe, first I parse them into two different **float** columns, namely: 'Latitude' and 'Longitude'.

In [63]:
```python
regions = pd.read_csv('country_region.csv')[['Country','Region']]
df = pd.merge(df, regions, on='Country', how='inner')

def get_lat(x):
    return float(x.split(',')[0][1:])
def get_lon(x):
    return float(x.split(',')[1][:-1])

df = df[~df['Location'].isnull()] # To make sure that there's no null values amongst that geopy re
turned.

df['Latitude'] = df['Location'].apply(get_lat)
df['Longitude'] = df['Location'].apply(get_lon)
df[['Region', 'Latitude', 'Longitude']].head(3)
```

	Region	Latitude	Longitude
0	North America	39.290882	-76.610759
1	North America	43.034993	-87.922497
2	North America	30.271129	-97.743700

In [64]:
```
from sklearn.preprocessing import LabelEncoder

def multi_label_encoder(df, cols2code):
        encoder = LabelEncoder()
        for col in cols2code:
            df[col+'(Encoded)'] = encoder.fit_transform(df[col])
        return df

df = multi_label_encoder(df, ['Region', 'Country'])
df.head()
```

	Unnamed: 0	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Bicycle Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)
0	0	Baltimore(MD)	285.0	7.0	Auto Sprawl	United States	85.0	6.1	0.3	2.6	0.66
1	5	Milwaukee(WI)	297.0	7.0	Auto Sprawl	United States	88.6	3.6	0.5	2.7	0.64
2	13	Austin(TX)	301.0	7.0	Auto Sprawl	United States	86.8	2.6	0.8	1.8	0.60
3	18	Chicago(IL)	269.0	8.0	Auto Innovative	United States	78.0	12.0	0.7	3.1	0.71
4	40	Atlanta(GA)	273.0	8.0	Auto Innovative	United States	86.8	3.1	0.3	1.3	0.65

Now, the dataset is ready to be investigated. First things first, we perform a percentage based check to see which columns are missing and how many values compared to the total number of rows in the dataset. If particular columns are missing relatively too large number of values, it's better to consider dropping the feature them instead of trying to impute the values. Below the top five features are shown, the missing values is shown in procent of total missing for the feature. If a feature is missing more than 25% of its values it is deemed not reliable and therfor dropped.

In [65]:
```
temp_df = pd.DataFrame(df.isnull().sum(axis=0), columns=['Missing Values'])/df.count()[0]*100
temp_df = temp_df.sort_values(by=temp_df.columns[0], ascending=False)
pd.set_option('display.max_rows', 1000)
print(temp_df.head(5))

# Drop features
cols2drop = temp_df[temp_df['Missing Values']>=25].index
df = df.drop(cols2drop, axis=1).drop('Unnamed: 0', axis=1).drop('coordinates', axis=1)
df.head(3)
```

```
                        Missing Values
AvgTemperature               67.441860
Traffic Index                55.149502
Inefficiency Index           55.149502
Travel Time Index            55.149502
Congestion PM Peak (%)       47.840532
```

	City	cityID	clusterID	Typology	Country	Car Modeshare (%)	Public Transit Modeshare (%)	Walking Modeshare (%)	Gasoline Pump Price (USD/liter)	Road Deaths Rate (per 1000)	Subway Length (km)	Subw Leng Dens (p k
0	Baltimore(MD)	285.0	7.0	Auto Sprawl	United States	85.0	6.1	2.6	0.66	8.5	24.9	0.0134
1	Milwaukee(WI)	297.0	7.0	Auto Sprawl	United States	88.6	3.6	2.7	0.64	9.8	0.0	0.0000
2	Austin(TX)	301.0	7.0	Auto Sprawl	United States	86.8	2.6	1.8	0.60	12.8	0.0	0.0000

Now that all viable features are left in our dataframe we will perform feature selection and select the most useful features to be used in the model.

Instead of trying to impute values for the remaining columns we will check the correlation, first correlation check between the coordinates and the Latitude/Longitude values - as a smart way of feature selection. The absolute value of correlation is used because the important aspect is the magnitude of the correlation to explain a change in a variable compared to another one.

In [66]:
```
# Correlation with latitude
pd.set_option('display.max_rows', 1000)
corr_Lat = abs(pd.DataFrame(df.corr()['Latitude'])).sort_values(by='Latitude', ascending=False)
corr_Lat.head(10)
```

	Latitude
Latitude	1.000000
Temperature	0.669497
clusterID	0.496319
Road Deaths Rate (per 1000)	0.494275
Life Expectancy (years)	0.460763
GDP per Capita (USD)	0.458740
Digital Penetration	0.458065
Development Factor	0.433200
Internet Penetration	0.404412
CO2 Emissions per Capita (metric tonnes)	0.399059

In [67]:
```
# Correlation with longitude
pd.set_option('display.max_rows', 1000)
corr_Lon = abs(pd.DataFrame(df.corr()['Longitude'])).sort_values(by='Longitude', ascending=False)
corr_Lon.head(10)
```

	Longitude
Longitude	1.000000
Region(Encoded)	0.686697
Car Modeshare (%)	0.538104
Urbanization Rate Change 2015 – 2025 (pp)	0.525583
Urbanization Rate 2015 (%)	0.437028
Sustainability Factor	0.433833
Congestion Factor	0.430325
Population Factor	0.429835
Walking Modeshare (%)	0.389066
Development Factor	0.380073

Interestingly, if you see the index of the outputs above, the features that the latitude and longitude values depend on the highest are completely different. Lattitude being correlated to the average temperature on 30th of November is quite logical. However, the other features do not represent a direct relation to both longitude and latitude.

Since there is no distinct features for both latitude and longitude, three of the features correlated to each will selected for the model. We will now attempt to perform continent classification of the cities.

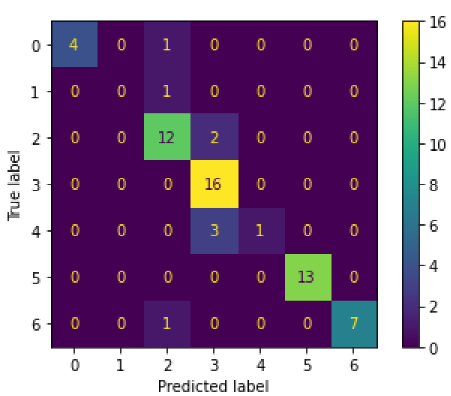
In [68]:
```
df_reg = df[['Road Deaths Rate (per 1000)','Urbanization Rate Change 2015 – 2025 (pp)', 'Congestio
n Factor',\
         'Digital Penetration', 'Car Modeshare (%)', 'Life Expectancy (years)', \
         'Region(Encoded)']].dropna()
X = df_reg[['Road Deaths Rate (per 1000)','Urbanization Rate Change 2015 – 2025 (pp)', 'Congestion
Factor',\
         'Digital Penetration', 'Car Modeshare (%)', 'Life Expectancy (years)']].values
y = df_reg['Region(Encoded)'].values
X_norm = (X - np.mean(X))/np.std(X) # Normalize values so we get better results!
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.25, random_state=42)
```

In [69]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

n_estimators = [int(x) for x in np.linspace(start = 100, stop = 600, num = 10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 50, num = 5)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

rf = RandomForestClassifier()
clf = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3,
verbose=2, random_state=42, n_jobs = -1)# Fit the random search model
clf.fit(X_train, y_train)

clf.score(X_test, y_test)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    4.1s
[Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed:   13.2s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:   25.9s finished

0.8688524590163934
```

In [70]:
```
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(clf, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x170e2a281c0>
```



Three fold cross validation has been used to evaluate the continent classification, as seen above the classification model was able to classify the cities continent quite well, acheiving about 87% accuracy when evaluated.

Now we will move on to the most challenging part, namely predicting the latidtude/ longitude coordinates of cities. To do this a multilayer feed-forward neural network will be used.

By seeing the correlation results above, we decided to use 6 features in total:

- 3 features that are correlated to Latitude the most (which are not directly related with geography).
- 3 features that are correlated to Longitude the most (which are not directly related with geography).

The chosen features are: Road Deaths Rate (per 1000), Digital Penetration, Life Expectancy (years), Car Modeshare (%), Urbanization Rate Change 2015 – 2025 (pp), Congestion Factor.

In [71]:
```
df_nn = df[['Road Deaths Rate (per 1000)','Urbanization Rate Change 2015 – 2025 (pp)', 'Congestion Factor',\
         'GDP per Capita (USD)', 'Car Modeshare (%)', 'Life Expectancy (years)', \
         'Latitude', 'Longitude']].dropna()

X = df_nn[['Road Deaths Rate (per 1000)','Urbanization Rate Change 2015 – 2025 (pp)', 'Congestion Factor',\
         'GDP per Capita (USD)', 'Car Modeshare (%)', 'Life Expectancy (years)']].values

y = df_nn[['Latitude', 'Longitude']].values

X_norm = (X - np.mean(X))/np.std(X) # Normalize values so we get better results!

X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.3, random_state=42)
```

In [72]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from keras.optimizers import Adam

# define the keras model
model = Sequential()
model.add(Dense(128, input_dim=np.shape(X_train)[1], activation='relu'))
model.add(Dense(128, activation='tanh'))
model.add(Dense(2, activation='linear')) # Output layer.


# compile the keras model
opt = Adam(learning_rate=1e-2)
model.compile(loss='mean_squared_error', optimizer=opt, metrics=['mean_squared_error'])
model.fit(X_train, y_train, epochs=150, batch_size=64, verbose=1, validation_split=0.25) # Do not
 print the progress since 3500 epochs.
```

```
Epoch 1/150
2/2 [==============================] - 0s 51ms/step - loss: 3763.8682 - mean_squared_error: 3763.8682 - val_loss: 2868.6621 - val_mean_squ
ared_error: 2868.6621
Epoch 2/150
2/2 [==============================] - 0s 8ms/step - loss: 3573.0178 - mean_squared_error: 3573.0178 - val_loss: 2755.3420 - val_mean_squa
red_error: 2755.3420
Epoch 3/150
2/2 [==============================] - 0s 8ms/step - loss: 3423.5024 - mean_squared_error: 3423.5024 - val_loss: 2677.7498 - val_mean_squa
red_error: 2677.7498
Epoch 4/150
2/2 [==============================] - 0s 9ms/step - loss: 3305.5146 - mean_squared_error: 3305.5146 - val_loss: 2611.2417 - val_mean_squa
red_error: 2611.2417
Epoch 5/150
2/2 [==============================] - 0s 8ms/step - loss: 3183.5374 - mean_squared_error: 3183.5374 - val_loss: 2546.0107 - val_mean_squa
red_error: 2546.0107
Epoch 6/150
2/2 [==============================] - 0s 8ms/step - loss: 3061.3372 - mean_squared_error: 3061.3372 - val_loss: 2494.0210 - val_mean_squa
red_error: 2494.0210
Epoch 7/150
2/2 [==============================] - 0s 9ms/step - loss: 2953.0466 - mean_squared_error: 2953.0466 - val_loss: 2470.0552 - val_mean_squa
red_error: 2470.0552
Epoch 8/150
2/2 [==============================] - 0s 9ms/step - loss: 2863.7903 - mean_squared_error: 2863.7903 - val_loss: 2457.6318 - val_mean_squa
red_error: 2457.6318
Epoch 9/150
2/2 [==============================] - 0s 10ms/step - loss: 2801.5522 - mean_squared_error: 2801.5522 - val_loss: 2447.9175 - val_mean_squ
ared_error: 2447.9175
Epoch 10/150
2/2 [==============================] - 0s 9ms/step - loss: 2743.8218 - mean_squared_error: 2743.8218 - val_loss: 2437.9768 - val_mean_squa
red_error: 2437.9768
Epoch 11/150
2/2 [==============================] - 0s 9ms/step - loss: 2695.0867 - mean_squared_error: 2695.0867 - val_loss: 2427.1089 - val_mean_squa
red_error: 2427.1089
Epoch 12/150
2/2 [==============================] - 0s 8ms/step - loss: 2654.0537 - mean_squared_error: 2654.0537 - val_loss: 2411.1843 - val_mean_squa
red_error: 2411.1843
Epoch 13/150
2/2 [==============================] - 0s 7ms/step - loss: 2602.1560 - mean_squared_error: 2602.1560 - val_loss: 2383.2471 - val_mean_squa
red_error: 2383.2471
Epoch 14/150
2/2 [==============================] - 0s 9ms/step - loss: 2554.4082 - mean_squared_error: 2554.4082 - val_loss: 2357.1504 - val_mean_squa
red_error: 2357.1504
Epoch 15/150
2/2 [==============================] - 0s 9ms/step - loss: 2515.2671 - mean_squared_error: 2515.2671 - val_loss: 2340.8728 - val_mean_squa
red_error: 2340.8728
Epoch 16/150
2/2 [==============================] - 0s 10ms/step - loss: 2472.0593 - mean_squared_error: 2472.0593 - val_loss: 2343.0505 - val_mean_squ
ared_error: 2343.0505
Epoch 17/150
2/2 [==============================] - 0s 9ms/step - loss: 2442.9583 - mean_squared_error: 2442.9583 - val_loss: 2334.5110 - val_mean_squa
red_error: 2334.5110
Epoch 18/150
2/2 [==============================] - 0s 9ms/step - loss: 2423.3076 - mean_squared_error: 2423.3076 - val_loss: 2340.2896 - val_mean_squa
red_error: 2340.2896
Epoch 19/150
2/2 [==============================] - 0s 9ms/step - loss: 2390.4395 - mean_squared_error: 2390.4395 - val_loss: 2331.8545 - val_mean_squa
red_error: 2331.8545
Epoch 20/150
2/2 [==============================] - 0s 10ms/step - loss: 2384.5054 - mean_squared_error: 2384.5054 - val_loss: 2351.2578 - val_mean_squ
ared_error: 2351.2578
Epoch 21/150
2/2 [==============================] - 0s 11ms/step - loss: 2349.2266 - mean_squared_error: 2349.2266 - val_loss: 2354.6743 - val_mean_squ
ared_error: 2354.6743
Epoch 22/150
2/2 [==============================] - 0s 9ms/step - loss: 2330.2776 - mean_squared_error: 2330.2776 - val_loss: 2371.9287 - val_mean_squa
red_error: 2371.9287
Epoch 23/150
2/2 [==============================] - 0s 12ms/step - loss: 2317.2871 - mean_squared_error: 2317.2871 - val_loss: 2383.9429 - val_mean_squ
ared_error: 2383.9429
Epoch 24/150
2/2 [==============================] - 0s 10ms/step - loss: 2308.8430 - mean_squared_error: 2308.8430 - val_loss: 2376.6294 - val_mean_squ
ared_error: 2376.6294
Epoch 25/150
2/2 [==============================] - 0s 9ms/step - loss: 2288.9797 - mean_squared_error: 2288.9797 - val_loss: 2391.9404 - val_mean_squa
red_error: 2391.9404
Epoch 26/150
```

```
2/2 [==============================] - 0s 9ms/step - loss: 2281.8726 - mean_squared_error: 2281.8726 - val_loss: 2410.5076 - val_mean_squa
red_error: 2410.5076
Epoch 27/150
2/2 [==============================] - 0s 8ms/step - loss: 2277.7864 - mean_squared_error: 2277.7864 - val_loss: 2406.5354 - val_mean_squa
red_error: 2406.5354
Epoch 28/150
2/2 [==============================] - 0s 9ms/step - loss: 2263.3201 - mean_squared_error: 2263.3201 - val_loss: 2400.6042 - val_mean_squa
red_error: 2400.6042
Epoch 29/150
2/2 [==============================] - 0s 11ms/step - loss: 2262.4458 - mean_squared_error: 2262.4458 - val_loss: 2422.2043 - val_mean_squ
ared_error: 2422.2043
Epoch 30/150
2/2 [==============================] - 0s 8ms/step - loss: 2251.3076 - mean_squared_error: 2251.3076 - val_loss: 2434.2458 - val_mean_squa
red_error: 2434.2458
Epoch 31/150
2/2 [==============================] - 0s 10ms/step - loss: 2248.3726 - mean_squared_error: 2248.3726 - val_loss: 2433.8972 - val_mean_squ
ared_error: 2433.8972
Epoch 32/150
2/2 [==============================] - 0s 12ms/step - loss: 2244.1536 - mean_squared_error: 2244.1536 - val_loss: 2443.3938 - val_mean_squ
ared_error: 2443.3938
Epoch 33/150
2/2 [==============================] - 0s 8ms/step - loss: 2237.6313 - mean_squared_error: 2237.6313 - val_loss: 2452.7581 - val_mean_squa
red_error: 2452.7581
Epoch 34/150
2/2 [==============================] - 0s 8ms/step - loss: 2243.7927 - mean_squared_error: 2243.7927 - val_loss: 2446.8499 - val_mean_squa
red_error: 2446.8499
Epoch 35/150
2/2 [==============================] - 0s 12ms/step - loss: 2229.4578 - mean_squared_error: 2229.4578 - val_loss: 2464.5188 - val_mean_squ
ared_error: 2464.5188
Epoch 36/150
2/2 [==============================] - 0s 9ms/step - loss: 2232.5969 - mean_squared_error: 2232.5969 - val_loss: 2455.2598 - val_mean_squa
red_error: 2455.2598
Epoch 37/150
2/2 [==============================] - 0s 13ms/step - loss: 2227.1245 - mean_squared_error: 2227.1245 - val_loss: 2458.6738 - val_mean_squ
ared_error: 2458.6738
Epoch 38/150
2/2 [==============================] - 0s 8ms/step - loss: 2220.2937 - mean_squared_error: 2220.2937 - val_loss: 2467.8049 - val_mean_squa
red_error: 2467.8049
Epoch 39/150
2/2 [==============================] - 0s 7ms/step - loss: 2217.0667 - mean_squared_error: 2217.0667 - val_loss: 2457.7061 - val_mean_squa
red_error: 2457.7061
Epoch 40/150
2/2 [==============================] - 0s 8ms/step - loss: 2239.2559 - mean_squared_error: 2239.2559 - val_loss: 2449.2104 - val_mean_squa
red_error: 2449.2104
Epoch 41/150
2/2 [==============================] - 0s 8ms/step - loss: 2212.9644 - mean_squared_error: 2212.9644 - val_loss: 2481.9277 - val_mean_squa
red_error: 2481.9277
Epoch 42/150
2/2 [==============================] - 0s 7ms/step - loss: 2230.9163 - mean_squared_error: 2230.9163 - val_loss: 2463.0349 - val_mean_squa
red_error: 2463.0349
Epoch 43/150
2/2 [==============================] - 0s 7ms/step - loss: 2211.1619 - mean_squared_error: 2211.1619 - val_loss: 2442.6353 - val_mean_squa
red_error: 2442.6353
Epoch 44/150
2/2 [==============================] - 0s 9ms/step - loss: 2258.4617 - mean_squared_error: 2258.4617 - val_loss: 2463.4976 - val_mean_squa
red_error: 2463.4976
Epoch 45/150
2/2 [==============================] - 0s 6ms/step - loss: 2218.1443 - mean_squared_error: 2218.1443 - val_loss: 2512.0361 - val_mean_squa
red_error: 2512.0361
Epoch 46/150
2/2 [==============================] - 0s 6ms/step - loss: 2256.0227 - mean_squared_error: 2256.0227 - val_loss: 2486.9678 - val_mean_squa
red_error: 2486.9678
Epoch 47/150
2/2 [==============================] - 0s 6ms/step - loss: 2189.6782 - mean_squared_error: 2189.6782 - val_loss: 2457.1223 - val_mean_squa
red_error: 2457.1223
Epoch 48/150
2/2 [==============================] - 0s 10ms/step - loss: 2267.9197 - mean_squared_error: 2267.9197 - val_loss: 2457.3713 - val_mean_squ
ared_error: 2457.3713
Epoch 49/150
2/2 [==============================] - 0s 7ms/step - loss: 2210.4836 - mean_squared_error: 2210.4836 - val_loss: 2530.8826 - val_mean_squa
red_error: 2530.8826
Epoch 50/150
2/2 [==============================] - ETA: 0s - loss: 2526.8586 - mean_squared_error: 2526.85 - 0s 7ms/step - loss: 2264.2717 - mean_squa
red_error: 2264.2717 - val_loss: 2497.4380 - val_mean_squared_error: 2497.4380
Epoch 51/150
2/2 [==============================] - 0s 7ms/step - loss: 2206.5251 - mean_squared_error: 2206.5251 - val_loss: 2457.5159 - val_mean_squa
```

```
red_error: 2457.5159
Epoch 52/150
2/2 [==============================] - 0s 7ms/step - loss: 2211.7083 - mean_squared_error: 2211.7083 - val_loss: 2454.0132 - val_mean_squa
red_error: 2454.0132
Epoch 53/150
2/2 [==============================] - 0s 7ms/step - loss: 2207.3901 - mean_squared_error: 2207.3901 - val_loss: 2482.4468 - val_mean_squa
red_error: 2482.4468
Epoch 54/150
2/2 [==============================] - 0s 9ms/step - loss: 2229.5635 - mean_squared_error: 2229.5635 - val_loss: 2488.8364 - val_mean_squa
red_error: 2488.8364
Epoch 55/150
2/2 [==============================] - 0s 8ms/step - loss: 2199.5513 - mean_squared_error: 2199.5513 - val_loss: 2451.9097 - val_mean_squa
red_error: 2451.9097
Epoch 56/150
2/2 [==============================] - 0s 8ms/step - loss: 2212.0784 - mean_squared_error: 2212.0784 - val_loss: 2458.2178 - val_mean_squa
red_error: 2458.2178
Epoch 57/150
2/2 [==============================] - 0s 7ms/step - loss: 2194.4536 - mean_squared_error: 2194.4536 - val_loss: 2499.5076 - val_mean_squa
red_error: 2499.5076
Epoch 58/150
2/2 [==============================] - 0s 8ms/step - loss: 2223.2310 - mean_squared_error: 2223.2310 - val_loss: 2495.2188 - val_mean_squa
red_error: 2495.2188
Epoch 59/150
2/2 [==============================] - 0s 9ms/step - loss: 2196.8445 - mean_squared_error: 2196.8445 - val_loss: 2458.8936 - val_mean_squa
red_error: 2458.8936
Epoch 60/150
2/2 [==============================] - 0s 7ms/step - loss: 2191.7996 - mean_squared_error: 2191.7996 - val_loss: 2452.1396 - val_mean_squa
red_error: 2452.1396
Epoch 61/150
2/2 [==============================] - 0s 8ms/step - loss: 2183.7751 - mean_squared_error: 2183.7751 - val_loss: 2473.6404 - val_mean_squa
red_error: 2473.6404
Epoch 62/150
2/2 [==============================] - 0s 8ms/step - loss: 2192.1270 - mean_squared_error: 2192.1270 - val_loss: 2486.0605 - val_mean_squa
red_error: 2486.0605
Epoch 63/150
2/2 [==============================] - 0s 8ms/step - loss: 2183.8289 - mean_squared_error: 2183.8289 - val_loss: 2469.7678 - val_mean_squa
red_error: 2469.7678
Epoch 64/150
2/2 [==============================] - 0s 7ms/step - loss: 2179.0789 - mean_squared_error: 2179.0789 - val_loss: 2461.1392 - val_mean_squa
red_error: 2461.1392
Epoch 65/150
2/2 [==============================] - 0s 9ms/step - loss: 2179.9685 - mean_squared_error: 2179.9685 - val_loss: 2464.1533 - val_mean_squa
red_error: 2464.1533
Epoch 66/150
2/2 [==============================] - 0s 7ms/step - loss: 2173.2039 - mean_squared_error: 2173.2039 - val_loss: 2463.8284 - val_mean_squa
red_error: 2463.8284
Epoch 67/150
2/2 [==============================] - 0s 7ms/step - loss: 2173.7859 - mean_squared_error: 2173.7859 - val_loss: 2470.7375 - val_mean_squa
red_error: 2470.7375
Epoch 68/150
2/2 [==============================] - 0s 8ms/step - loss: 2171.8513 - mean_squared_error: 2171.8513 - val_loss: 2472.7542 - val_mean_squa
red_error: 2472.7542
Epoch 69/150
2/2 [==============================] - 0s 10ms/step - loss: 2168.4055 - mean_squared_error: 2168.4055 - val_loss: 2469.9270 - val_mean_squ
ared_error: 2469.9270
Epoch 70/150
2/2 [==============================] - 0s 8ms/step - loss: 2170.2000 - mean_squared_error: 2170.2000 - val_loss: 2477.6475 - val_mean_squa
red_error: 2477.6475
Epoch 71/150
2/2 [==============================] - 0s 7ms/step - loss: 2165.7542 - mean_squared_error: 2165.7542 - val_loss: 2475.8110 - val_mean_squa
red_error: 2475.8110
Epoch 72/150
2/2 [==============================] - 0s 7ms/step - loss: 2165.7786 - mean_squared_error: 2165.7786 - val_loss: 2478.1316 - val_mean_squa
red_error: 2478.1316
Epoch 73/150
2/2 [==============================] - 0s 8ms/step - loss: 2166.0332 - mean_squared_error: 2166.0332 - val_loss: 2473.3818 - val_mean_squa
red_error: 2473.3818
Epoch 74/150
2/2 [==============================] - 0s 8ms/step - loss: 2165.3267 - mean_squared_error: 2165.3267 - val_loss: 2466.5610 - val_mean_squa
red_error: 2466.5610
Epoch 75/150
2/2 [==============================] - 0s 7ms/step - loss: 2162.2637 - mean_squared_error: 2162.2637 - val_loss: 2475.8892 - val_mean_squa
red_error: 2475.8892
Epoch 76/150
2/2 [==============================] - 0s 8ms/step - loss: 2159.0103 - mean_squared_error: 2159.0103 - val_loss: 2484.4241 - val_mean_squa
red_error: 2484.4241
```

```
Epoch 77/150
2/2 [==============================] - 0s 8ms/step - loss: 2159.4824 - mean_squared_error: 2159.4824 - val_loss: 2486.5034 - val_mean_squa
red_error: 2486.5034
Epoch 78/150
2/2 [==============================] - 0s 8ms/step - loss: 2156.3748 - mean_squared_error: 2156.3748 - val_loss: 2480.6638 - val_mean_squa
red_error: 2480.6638
Epoch 79/150
2/2 [==============================] - 0s 7ms/step - loss: 2156.9797 - mean_squared_error: 2156.9797 - val_loss: 2482.1033 - val_mean_squa
red_error: 2482.1033
Epoch 80/150
2/2 [==============================] - 0s 7ms/step - loss: 2170.4714 - mean_squared_error: 2170.4714 - val_loss: 2488.9153 - val_mean_squa
red_error: 2488.9153
Epoch 81/150
2/2 [==============================] - 0s 7ms/step - loss: 2159.5728 - mean_squared_error: 2159.5728 - val_loss: 2468.7446 - val_mean_squa
red_error: 2468.7446
Epoch 82/150
2/2 [==============================] - 0s 8ms/step - loss: 2164.3105 - mean_squared_error: 2164.3105 - val_loss: 2479.1794 - val_mean_squa
red_error: 2479.1794
Epoch 83/150
2/2 [==============================] - 0s 7ms/step - loss: 2150.9087 - mean_squared_error: 2150.9087 - val_loss: 2484.3137 - val_mean_squa
red_error: 2484.3137
Epoch 84/150
2/2 [==============================] - 0s 7ms/step - loss: 2152.1663 - mean_squared_error: 2152.1663 - val_loss: 2486.3535 - val_mean_squa
red_error: 2486.3535
Epoch 85/150
2/2 [==============================] - 0s 8ms/step - loss: 2164.7861 - mean_squared_error: 2164.7861 - val_loss: 2489.5515 - val_mean_squa
red_error: 2489.5515
Epoch 86/150
2/2 [==============================] - 0s 7ms/step - loss: 2145.6211 - mean_squared_error: 2145.6211 - val_loss: 2480.0586 - val_mean_squa
red_error: 2480.0586
Epoch 87/150
2/2 [==============================] - 0s 7ms/step - loss: 2150.8567 - mean_squared_error: 2150.8567 - val_loss: 2484.6479 - val_mean_squa
red_error: 2484.6479
Epoch 88/150
2/2 [==============================] - 0s 7ms/step - loss: 2146.5283 - mean_squared_error: 2146.5283 - val_loss: 2492.3728 - val_mean_squa
red_error: 2492.3728
Epoch 89/150
2/2 [==============================] - 0s 8ms/step - loss: 2145.2817 - mean_squared_error: 2145.2817 - val_loss: 2490.5552 - val_mean_squa
red_error: 2490.5552
Epoch 90/150
2/2 [==============================] - 0s 7ms/step - loss: 2147.7837 - mean_squared_error: 2147.7837 - val_loss: 2481.5239 - val_mean_squa
red_error: 2481.5239
Epoch 91/150
2/2 [==============================] - 0s 7ms/step - loss: 2142.1438 - mean_squared_error: 2142.1438 - val_loss: 2477.7090 - val_mean_squa
red_error: 2477.7090
Epoch 92/150
2/2 [==============================] - 0s 7ms/step - loss: 2147.1204 - mean_squared_error: 2147.1204 - val_loss: 2485.7646 - val_mean_squa
red_error: 2485.7646
Epoch 93/150
2/2 [==============================] - 0s 8ms/step - loss: 2138.9607 - mean_squared_error: 2138.9607 - val_loss: 2493.8018 - val_mean_squa
red_error: 2493.8018
Epoch 94/150
2/2 [==============================] - 0s 7ms/step - loss: 2144.1279 - mean_squared_error: 2144.1279 - val_loss: 2497.1006 - val_mean_squa
red_error: 2497.1006
Epoch 95/150
2/2 [==============================] - 0s 9ms/step - loss: 2144.1084 - mean_squared_error: 2144.1084 - val_loss: 2483.9893 - val_mean_squa
red_error: 2483.9893
Epoch 96/150
2/2 [==============================] - 0s 9ms/step - loss: 2149.2178 - mean_squared_error: 2149.2178 - val_loss: 2479.6304 - val_mean_squa
red_error: 2479.6304
Epoch 97/150
2/2 [==============================] - 0s 7ms/step - loss: 2137.4854 - mean_squared_error: 2137.4854 - val_loss: 2491.8350 - val_mean_squa
red_error: 2491.8350
Epoch 98/150
2/2 [==============================] - 0s 7ms/step - loss: 2138.2698 - mean_squared_error: 2138.2698 - val_loss: 2492.2319 - val_mean_squa
red_error: 2492.2319
Epoch 99/150
2/2 [==============================] - 0s 8ms/step - loss: 2134.3613 - mean_squared_error: 2134.3613 - val_loss: 2482.3687 - val_mean_squa
red_error: 2482.3687
Epoch 100/150
2/2 [==============================] - 0s 8ms/step - loss: 2134.5483 - mean_squared_error: 2134.5483 - val_loss: 2481.3223 - val_mean_squa
red_error: 2481.3223
Epoch 101/150
2/2 [==============================] - 0s 7ms/step - loss: 2135.2104 - mean_squared_error: 2135.2104 - val_loss: 2488.6387 - val_mean_squa
red_error: 2488.6387
Epoch 102/150
```

```
2/2 [==============================] - 0s 7ms/step - loss: 2136.7974 - mean_squared_error: 2136.7974 - val_loss: 2489.3208 - val_mean_squa
red_error: 2489.3208
Epoch 103/150
2/2 [==============================] - 0s 8ms/step - loss: 2132.1040 - mean_squared_error: 2132.1038 - val_loss: 2476.2478 - val_mean_squa
red_error: 2476.2478
Epoch 104/150
2/2 [==============================] - 0s 6ms/step - loss: 2137.5422 - mean_squared_error: 2137.5422 - val_loss: 2483.4871 - val_mean_squa
red_error: 2483.4871
Epoch 105/150
2/2 [==============================] - 0s 7ms/step - loss: 2130.2957 - mean_squared_error: 2130.2957 - val_loss: 2488.9541 - val_mean_squa
red_error: 2488.9541
Epoch 106/150
2/2 [==============================] - 0s 8ms/step - loss: 2129.4504 - mean_squared_error: 2129.4504 - val_loss: 2485.9692 - val_mean_squa
red_error: 2485.9692
Epoch 107/150
2/2 [==============================] - 0s 8ms/step - loss: 2128.7974 - mean_squared_error: 2128.7974 - val_loss: 2484.0686 - val_mean_squa
red_error: 2484.0686
Epoch 108/150
2/2 [==============================] - 0s 8ms/step - loss: 2127.2312 - mean_squared_error: 2127.2312 - val_loss: 2488.8928 - val_mean_squa
red_error: 2488.8928
Epoch 109/150
2/2 [==============================] - 0s 8ms/step - loss: 2125.6948 - mean_squared_error: 2125.6948 - val_loss: 2496.3811 - val_mean_squa
red_error: 2496.3811
Epoch 110/150
2/2 [==============================] - 0s 8ms/step - loss: 2128.9434 - mean_squared_error: 2128.9434 - val_loss: 2486.1570 - val_mean_squa
red_error: 2486.1570
Epoch 111/150
2/2 [==============================] - 0s 7ms/step - loss: 2129.0479 - mean_squared_error: 2129.0479 - val_loss: 2485.3357 - val_mean_squa
red_error: 2485.3357
Epoch 112/150
2/2 [==============================] - 0s 7ms/step - loss: 2135.4292 - mean_squared_error: 2135.4292 - val_loss: 2491.3794 - val_mean_squa
red_error: 2491.3794
Epoch 113/150
2/2 [==============================] - 0s 7ms/step - loss: 2123.2722 - mean_squared_error: 2123.2722 - val_loss: 2506.5400 - val_mean_squa
red_error: 2506.5400
Epoch 114/150
2/2 [==============================] - 0s 8ms/step - loss: 2134.4146 - mean_squared_error: 2134.4146 - val_loss: 2491.5916 - val_mean_squa
red_error: 2491.5916
Epoch 115/150
2/2 [==============================] - 0s 8ms/step - loss: 2124.0398 - mean_squared_error: 2124.0398 - val_loss: 2474.2344 - val_mean_squa
red_error: 2474.2344
Epoch 116/150
2/2 [==============================] - 0s 7ms/step - loss: 2130.6226 - mean_squared_error: 2130.6226 - val_loss: 2483.7764 - val_mean_squa
red_error: 2483.7764
Epoch 117/150
2/2 [==============================] - 0s 9ms/step - loss: 2132.2041 - mean_squared_error: 2132.2041 - val_loss: 2502.7444 - val_mean_squa
red_error: 2502.7444
Epoch 118/150
2/2 [==============================] - 0s 8ms/step - loss: 2126.0029 - mean_squared_error: 2126.0029 - val_loss: 2491.5537 - val_mean_squa
red_error: 2491.5537
Epoch 119/150
2/2 [==============================] - 0s 7ms/step - loss: 2122.9783 - mean_squared_error: 2122.9783 - val_loss: 2480.6365 - val_mean_squa
red_error: 2480.6365
Epoch 120/150
2/2 [==============================] - 0s 7ms/step - loss: 2128.7490 - mean_squared_error: 2128.7490 - val_loss: 2484.2078 - val_mean_squa
red_error: 2484.2078
Epoch 121/150
2/2 [==============================] - 0s 8ms/step - loss: 2118.1899 - mean_squared_error: 2118.1899 - val_loss: 2490.0911 - val_mean_squa
red_error: 2490.0911
Epoch 122/150
2/2 [==============================] - 0s 8ms/step - loss: 2123.3416 - mean_squared_error: 2123.3416 - val_loss: 2490.0017 - val_mean_squa
red_error: 2490.0017
Epoch 123/150
2/2 [==============================] - 0s 7ms/step - loss: 2121.9387 - mean_squared_error: 2121.9387 - val_loss: 2488.5659 - val_mean_squa
red_error: 2488.5659
Epoch 124/150
2/2 [==============================] - 0s 8ms/step - loss: 2114.3481 - mean_squared_error: 2114.3481 - val_loss: 2485.0547 - val_mean_squa
red_error: 2485.0547
Epoch 125/150
2/2 [==============================] - 0s 7ms/step - loss: 2120.8169 - mean_squared_error: 2120.8169 - val_loss: 2491.9434 - val_mean_squa
red_error: 2491.9434
Epoch 126/150
2/2 [==============================] - 0s 8ms/step - loss: 2119.4946 - mean_squared_error: 2119.4946 - val_loss: 2491.1309 - val_mean_squa
red_error: 2491.1309
Epoch 127/150
2/2 [==============================] - 0s 7ms/step - loss: 2117.5881 - mean_squared_error: 2117.5881 - val_loss: 2484.0513 - val_mean_squa
```

```
red_error: 2484.0513
Epoch 128/150
2/2 [==============================] - 0s 10ms/step - loss: 2117.6250 - mean_squared_error: 2117.6250 - val_loss: 2490.3811 - val_mean_squ
ared_error: 2490.3811
Epoch 129/150
2/2 [==============================] - 0s 8ms/step - loss: 2111.8882 - mean_squared_error: 2111.8882 - val_loss: 2499.3342 - val_mean_squa
red_error: 2499.3342
Epoch 130/150
2/2 [==============================] - 0s 9ms/step - loss: 2114.8596 - mean_squared_error: 2114.8596 - val_loss: 2495.5857 - val_mean_squa
red_error: 2495.5857
Epoch 131/150
2/2 [==============================] - 0s 9ms/step - loss: 2113.6846 - mean_squared_error: 2113.6846 - val_loss: 2493.2795 - val_mean_squa
red_error: 2493.2795
Epoch 132/150
2/2 [==============================] - 0s 7ms/step - loss: 2112.0286 - mean_squared_error: 2112.0286 - val_loss: 2483.0396 - val_mean_squa
red_error: 2483.0396
Epoch 133/150
2/2 [==============================] - 0s 7ms/step - loss: 2113.2520 - mean_squared_error: 2113.2520 - val_loss: 2486.0088 - val_mean_squa
red_error: 2486.0088
Epoch 134/150
2/2 [==============================] - 0s 9ms/step - loss: 2110.4753 - mean_squared_error: 2110.4753 - val_loss: 2498.9951 - val_mean_squa
red_error: 2498.9951
Epoch 135/150
2/2 [==============================] - 0s 8ms/step - loss: 2116.1807 - mean_squared_error: 2116.1807 - val_loss: 2499.3228 - val_mean_squa
red_error: 2499.3228
Epoch 136/150
2/2 [==============================] - 0s 8ms/step - loss: 2107.5339 - mean_squared_error: 2107.5339 - val_loss: 2485.9053 - val_mean_squa
red_error: 2485.9053
Epoch 137/150
2/2 [==============================] - 0s 8ms/step - loss: 2120.4351 - mean_squared_error: 2120.4351 - val_loss: 2483.2971 - val_mean_squa
red_error: 2483.2971
Epoch 138/150
2/2 [==============================] - 0s 9ms/step - loss: 2118.1680 - mean_squared_error: 2118.1680 - val_loss: 2509.2683 - val_mean_squa
red_error: 2509.2683
Epoch 139/150
2/2 [==============================] - 0s 7ms/step - loss: 2117.3948 - mean_squared_error: 2117.3948 - val_loss: 2499.0273 - val_mean_squa
red_error: 2499.0273
Epoch 140/150
2/2 [==============================] - 0s 8ms/step - loss: 2108.7043 - mean_squared_error: 2108.7043 - val_loss: 2485.6501 - val_mean_squa
red_error: 2485.6501
Epoch 141/150
2/2 [==============================] - 0s 6ms/step - loss: 2115.6223 - mean_squared_error: 2115.6223 - val_loss: 2490.7749 - val_mean_squa
red_error: 2490.7749
Epoch 142/150
2/2 [==============================] - 0s 9ms/step - loss: 2110.4883 - mean_squared_error: 2110.4883 - val_loss: 2507.6482 - val_mean_squa
red_error: 2507.6482
Epoch 143/150
2/2 [==============================] - 0s 7ms/step - loss: 2110.0137 - mean_squared_error: 2110.0137 - val_loss: 2500.4004 - val_mean_squa
red_error: 2500.4004
Epoch 144/150
2/2 [==============================] - 0s 8ms/step - loss: 2102.4683 - mean_squared_error: 2102.4683 - val_loss: 2490.6982 - val_mean_squa
red_error: 2490.6982
Epoch 145/150
2/2 [==============================] - 0s 8ms/step - loss: 2107.8696 - mean_squared_error: 2107.8696 - val_loss: 2493.0244 - val_mean_squa
red_error: 2493.0244
Epoch 146/150
2/2 [==============================] - 0s 9ms/step - loss: 2104.5281 - mean_squared_error: 2104.5281 - val_loss: 2499.7498 - val_mean_squa
red_error: 2499.7498
Epoch 147/150
2/2 [==============================] - 0s 8ms/step - loss: 2109.7358 - mean_squared_error: 2109.7358 - val_loss: 2499.7910 - val_mean_squa
red_error: 2499.7910
Epoch 148/150
2/2 [==============================] - 0s 7ms/step - loss: 2102.0623 - mean_squared_error: 2102.0623 - val_loss: 2487.0388 - val_mean_squa
red_error: 2487.0388
Epoch 149/150
2/2 [==============================] - 0s 8ms/step - loss: 2107.6714 - mean_squared_error: 2107.6714 - val_loss: 2488.2637 - val_mean_squa
red_error: 2488.2637
Epoch 150/150
2/2 [==============================] - 0s 8ms/step - loss: 2101.0576 - mean_squared_error: 2101.0576 - val_loss: 2499.7559 - val_mean_squa
red_error: 2499.7559

<tensorflow.python.keras.callbacks.History at 0x170e4b8acd0>
```

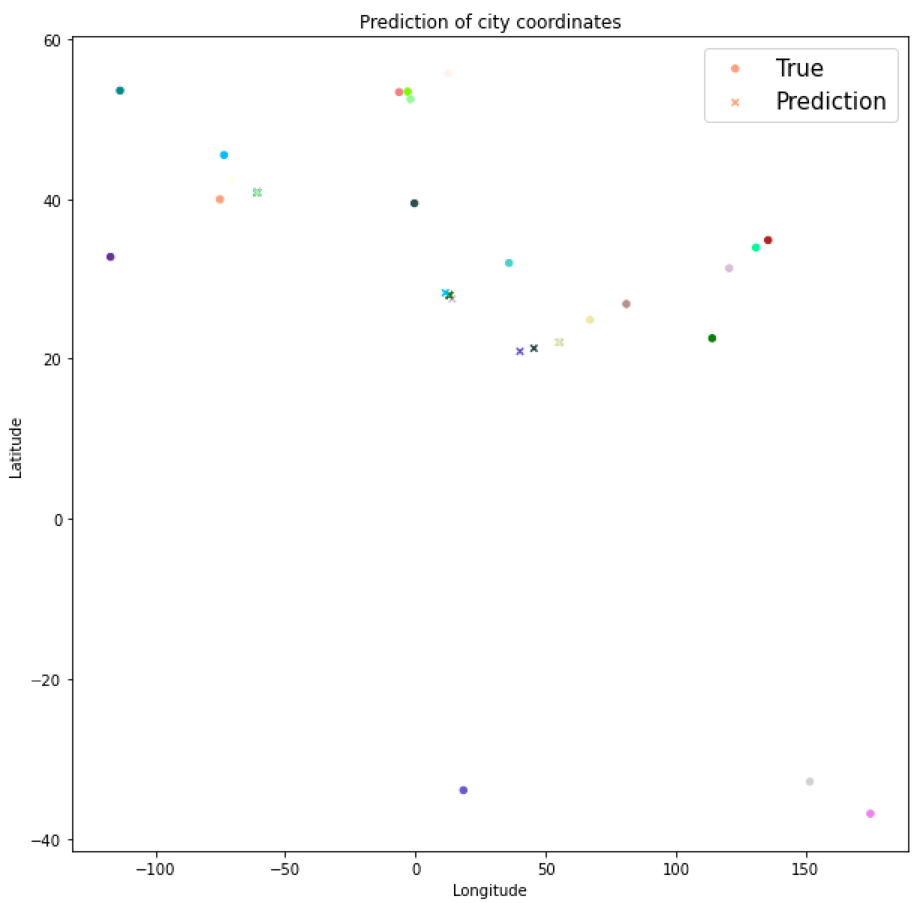
In [73]:
```
from sklearn.metrics import mean_absolute_error

y_pred = model.predict(X_test)
mean_absolute_error(y_test, y_pred)
```

```
41.00845453495208
```

The error above is the mean euclidean distance between the predicted coordinates and the true coordinates. Since we are dealing with a 2-D plane, it coul be nice if we could visualize this, lets do that below.

In [74]:
```
import matplotlib.colors as mcolors
import random

pred = pd.DataFrame(y_pred, columns=['Latitude', 'Longitude'])
true = pd.DataFrame(y_test, columns=['Latitude', 'Longitude'])

# Too much info when all preditcitons are plotted, just plot k of them
k = 20
pred_k = pred[:k]
true_k = true[:k]

color_dict = mcolors.CSS4_COLORS
color_list = list(color_dict.keys())
colors = random.sample(color_list, k=k) # pick the k nr. colors needed at random

# plot
fig, ax = plt.subplots(figsize=(10,10))
true_k.plot.scatter(x='Longitude', y='Latitude', marker='o', color=colors, ax=ax, label='True')
pred_k.plot.scatter(x='Longitude', y='Latitude', marker='x', color=colors, ax=ax, label='Predictio
n')
plt.title('Prediction of city coordinates')
ax.legend(prop={'size': 15})
```

```
<matplotlib.legend.Legend at 0x170e4f92f70>
```



The plot above gives good intuition of how our model behaves. The model is able to predict the coordinats very well for most cities that have a latitude greater than about 20 (above the equator). However for cities which lie below the equator or outlier cities, the model has difficulties predicting the coordinates.

Finding weather anamolies

Storms and harmful weather are common in some areas of the world, the storms can usually be attributed to anamolies in the weather. In this section we will look at finding weather anamolies in all the cities. The weather anamolies which will be analyzed are heavy rain and large pressure drops. Heavy rain will be analyzed because floods are often caused by sudden heavy rain. Pressure drops will be analyzed because sudden drops always occur before storms such as hurricans and tornados. An anamaly for rain weather will be defined as a day that the total rain fall is greater than five times the median level for the whole year. An anomaly for pressure drop is defined as a day pressure average being lower than 0.999 of the pressure median level for the whole year.

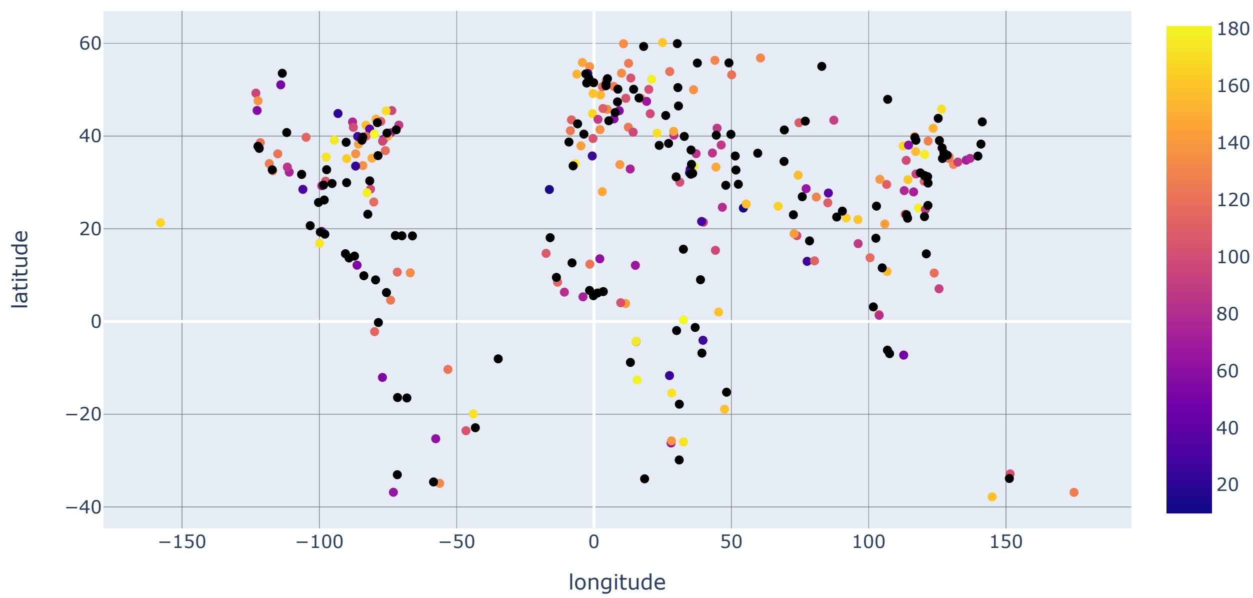
To find weather data a weather API from WorldWeatherOnline was used to get daily data every city between January 2018 to January 2019. The code below is the code which was used to parse and collect all the parameters of interest. Although most citie's weather could be found not all cities could be found using the weather API, so the weather of some of the cities are not evaluated.

In [ ]:
```
# from os import listdir
# from os.path import isfile, join
# from wwo_hist import retrieve_hist_data
# import pandas as pd
# from ast import literal_eval
# import re
# import os

# Get weather data for every city

# cities = pd.read_csv('location_from_pickle.csv', index_col=0, skipinitialspace=True)
# location_list1 = cities['City']
# frequency = 24
# start_date = '01-JAN-2018'
# end_date = '01-JAN-2019'
# api_key = '51f671d92fce4a8d9aa162950200612'

# for location in location_list1:
#     location_list = [location]
#     try:
#         hist_weather_data = retrieve_hist_data(api_key,
#                                         location_list,
#                                         start_date,
#                                         end_date,
#                                         frequency,
#                                         location_label = False,
#                                         export_csv = True,
#                                         store_df = True)
#     except:
#         pass


# Get only features of interest ('city', 'rain_anamolies', 'pressure_anamolies')

# weather_anamolies = pd.DataFrame(columns = ['city', 'rain_anamolies', 'pressure_anamolies'])
# path = 'weather data'
# files = [f for f in listdir(path) if isfile(join(path, f))]
# # print(files)
# nr = 0
# for file in files:
#     df = pd.read_csv(path+'/'+file)
#     prec_med = df['precipMM'].median()
#     pres_med = df['pressure'].median()
#     rain_anom = len(df['precipMM'][df['precipMM']>5*prec_med])
#     pres_anom = len(df['pressure'][df['pressure']<0.999*pres_med])
#     weather_anamolies.loc[nr] = df['location'][0], rain_anom, pres_anom
#     nr+=1
# weather_anamolies.to_csv(path+'/weather_anamolies.csv')
```

When the weather data is parsed and ready to go, we then plot the coordinates and intesity of the weather anamolies, this can be seen below.

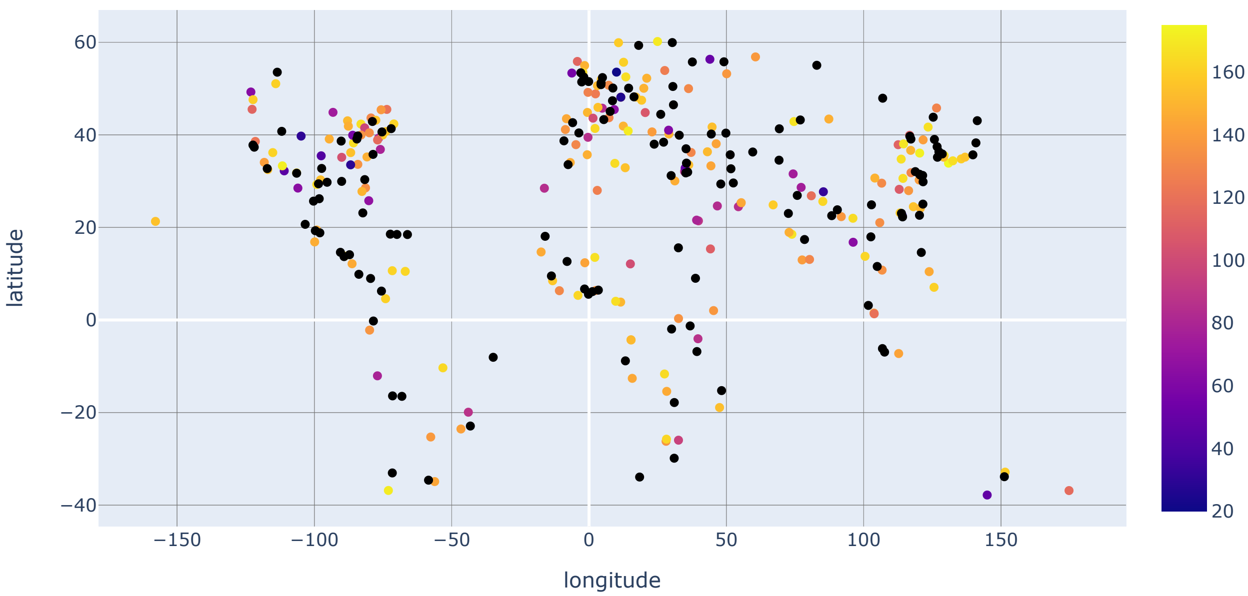
In [14]:
```
df = pd.read_csv('location_from_pickle.csv') # Previous saved dataset
df_weather = pd.read_csv('weather_anamolies.csv')
df_weather.drop(['Unnamed: 0'], axis=1)
# apply lat/ long to lists to further plot
lat_list = []
long_list = []
for i in df['coordinates']:
    lat_list.append(literal_eval(i)[0])
    long_list.append(literal_eval(i)[1])
# long_list
```

In [15]:
```
# Plot rain anamolies
z = df_weather['rain_anamolies']
scatter = go.Scatter(x=np.array(long_list).flatten(),
                     y=np.array(lat_list).flatten(),
                     marker={'color': np.array(z).flatten(),
                             'showscale': True},
                     mode='markers')
fig = go.FigureWidget(data=[scatter],
                      layout={'xaxis': {'title': 'longitude'},
                              'yaxis': {'title': 'latitude'}})

go.Figure(fig)
```



In [16]:

```
# Plot pressure anamolies
z = df_weather['pressure_anamolies']
scatter = go.Scatter(x=np.array(long_list).flatten(),
                     y=np.array(lat_list).flatten(),
                     marker={'color': np.array(z).flatten(),
                             'showscale': True},
                     mode='markers')
fig = go.FigureWidget(data=[scatter],
                      layout={'xaxis': {'title': 'longitude'},
                              'yaxis': {'title': 'latitude'}})
go.Figure(fig)
```



The rain and pressure anamolies are plotted above, but it is a bit hard to distinguish so the top 10 cities for both of these categories are shown below. For pressure differences it can be seen that many cities are located in China.

In [18]:

```
df_weather.sort_values(by=['rain_anamolies'], ascending=False).head(10)
```

	Unnamed: 0	city	rain_anamolies	pressure_anamolies
193	193	Washington(DC)	181	133
120	120	Memphis(TN)	180	135
27	27	Bratislava	179	144
192	192	Warsaw	179	160
164	164	Sendai	178	153
148	148	Providence(RI)	178	154
156	156	Richmond(VA)	177	149
105	105	Lille	177	148
194	194	Wuhan	176	172
119	119	Melbourne	175	147

In [19]:
```
df_weather.sort_values(by=['pressure_anamolies'], ascending=False).head(10)
```

	Unnamed: 0	city	rain_anamolies	pressure_anamolies
72	72	Harbin	132	175
194	194	Wuhan	176	172
180	180	Tianjin	84	171
65	65	Glasgow	64	170
38	38	Changsha	126	170
88	88	Jinan	96	170
43	43	Chongqing	161	169
167	167	Shijiazhuang	60	168
10	10	Auckland	100	167
196	196	Yekaterinburg	108	167

Conclusions

Transport Modal Shares

In the first section of the notebook, analysis with respect to transport variables was performed. It was neccessary to import the modeshares, to ensure correct performing of models. After dimension reduction, gasoline pump price regression models was created. The linear model and random forrest regression. It was interesting to see that both part performed similarly with score arround R2 ~0.6.

Coordinate predictions

The model was able to predict coordinates of the cities given six features for most cities, however for cities below the equator the model was not able to predict coordinates with high accuracy. In the template actually the MAE we got with the same parameters was 21, but for some reason while merging the notebooks the results was MAE 41. So the outputs here in the report are actually lower the what achieved individually but we couldn't have time enough to fix the bug.

Weather anamoly findings

When looking at the weather data, it was clear that many Chinese cities saw high ammounts of pressure drops which could mean that wind storms could be a some what common event in these cities. There was no exact area in the world which in particular saw high ammounts of rain anamolies, meaning that floods and sudden heavy rain could be common in many regions of the world.

Individual contributions

Individual contributions: You can see the table of contributions on this link: https://docs.google.com/document/d/1vM6VCNmWmQEN5QuODD59KJZ1mK_n7nQvPbErPJ19zJE/edit?fbclid=IwAR3f7iSPlesZQ1NrTiGnvJ6VzD6GMKsCaLANxgAo1JfKWs2bhAjPv0d17MI (https://docs.google.com/document/d/1vM6VCNmWmQEN5QuODD59KJZ1mK_n7nQvPbErPJ19zJE/edit?fbclid=IwAR3f7iSPlesZQ1NrTiGnvJ6VzD6GMKsCaLANxgAo1JfKWs2bhAjPv0d17MI)