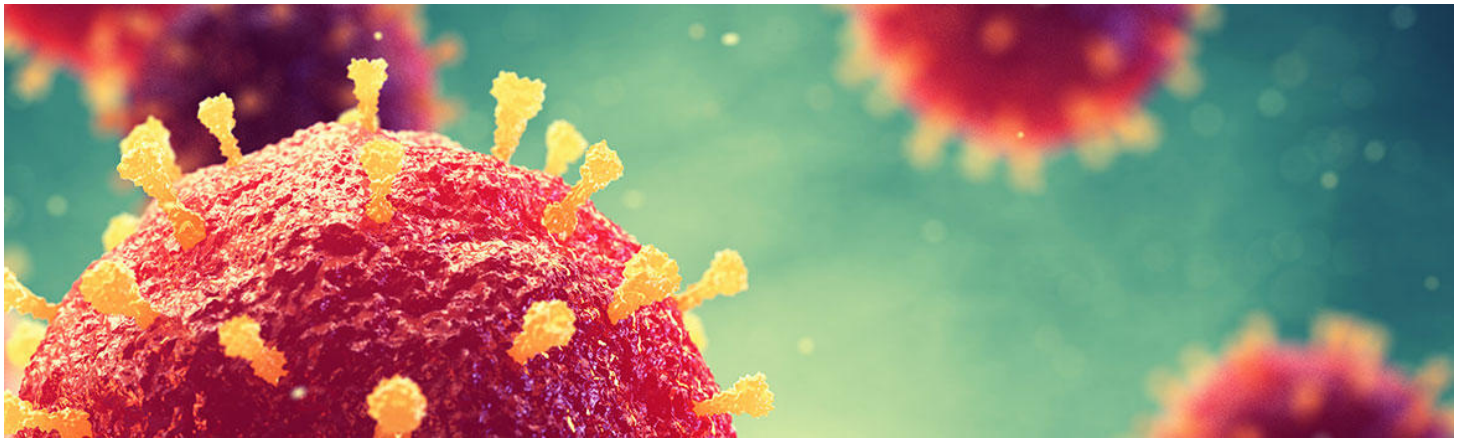


PART 3

Analyzing the epidemiological outbreak of COVID-19

Now we'll try to predict confirmed cases per country using regression techniques.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
import theme

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

%matplotlib inline
```

Data loading and wrangling

We will load COVID-19 data from the GitHub data repository (https://github.com/CSSEGISandData/COVID-19) for the 2019 Novel Coronavirus Visual Dashboard operated by the Johns Hopkins University Center for Systems Science and Engineering (JHU CSSE). Also, Supported by ESRI Living Atlas Team and the Johns Hopkins University Applied Physics Lab (JHU APL).

As we already known the data, we'll go faster now:

In [2]:

```
COVID_CONFIRMED_URL = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Confirmed.csv'
covid_confirmed = pd.read_csv(COVID_CONFIRMED_URL)
```

In [3]:

```
print(covid_confirmed.shape)
```

```
(463, 59)
```

In [4]:

```
covid_confirmed.head()
```

Out[4]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20
0	NaN	Thailand	15.000	101.000	2	3	5	7	8
1	NaN	Japan	36.000	138.000	2	1	2	2	4
2	NaN	Singapore	1.283	103.833	0	1	3	3	4
3	NaN	Nepal	28.167	84.250	0	0	0	1	1
4	NaN	Malaysia	2.500	112.500	0	0	0	3	4

First convert all the data to long format:

In [5]:

```
covid_confirmed_long = pd.melt(covid_confirmed,
                               id_vars=covid_confirmed.iloc[:, :4],
                               var_name='date',
                               value_name='confirmed')
```

In [6]:

```
covid_confirmed_long.shape
```

Out[6]:

```
(25465, 6)
```

In [7]:

```
covid_confirmed_long.head()
```

Out[7]:

	Province/State	Country/Region	Lat	Long	date	confirmed
0	NaN	Thailand	15.000	101.000	1/22/20	2
1	NaN	Japan	36.000	138.000	1/22/20	2
2	NaN	Singapore	1.283	103.833	1/22/20	0
3	NaN	Nepal	28.167	84.250	1/22/20	0
4	NaN	Malaysia	2.500	112.500	1/22/20	0

Data cleaning

As we did before replace `Mainland china` with just `China` , and fill some missing values.

In [8]:

```
covid_confirmed_long['Country/Region'].replace('Mainland China', 'China', inplac
e=True)
```

In [9]:

```
covid_confirmed_long[['Province/State']] = covid_confirmed_long[['Province/State
']].fillna('')
```

In [10]:

```
covid_confirmed_long.fillna(0, inplace=True)
```

Final checks:

In [11]:

```
covid_confirmed_long.isna().sum().sum()
```

Out[11]:

0

Country analysis over the time

Now we'll aggregate values by `Country/Region` and `date`.

> Hint! the `sort=False` parameter will keep our dates ordered.

Also, remove unused columns.

In [12]:

```
covid_countries_date_df = covid_confirmed_long.groupby(['Country/Region', 'date'
], sort=False).sum().reset_index()

covid_countries_date_df.drop(['Lat', 'Long'], axis=1, inplace=True)
```

In [13]:

```
covid_countries_date_df
```

Out[13]:

	Country/Region	date	confirmed
0	Thailand	1/22/20	2
1	Japan	1/22/20	2
2	Singapore	1/22/20	0
3	Nepal	1/22/20	0
4	Malaysia	1/22/20	0
...	...	...	...
8575	Mayotte	3/16/20	1
8576	Republic of the Congo	3/16/20	1
8577	Somalia	3/16/20	1
8578	Tanzania	3/16/20	1
8579	The Bahamas	3/16/20	1

8580 rows × 3 columns

Now just filter the daily data from the country you want to analyze:

In [14]:

```
COUNTRY = 'US'
```

In [15]:

```
covid_country = covid_countries_date_df[covid_countries_date_df['Country/Region'
] == COUNTRY]

covid_country.head()
```

Out[15]:

	Country/Region	date	confirmed
88	US	1/22/20	1
244	US	1/23/20	1
400	US	1/24/20	2
556	US	1/25/20	2
712	US	1/26/20	5

And create a `days` list with the amount of days since 1/22 for each day.

In [16]:

```
days = np.array([i for i in range(len(covid_country['date']))])

days
```

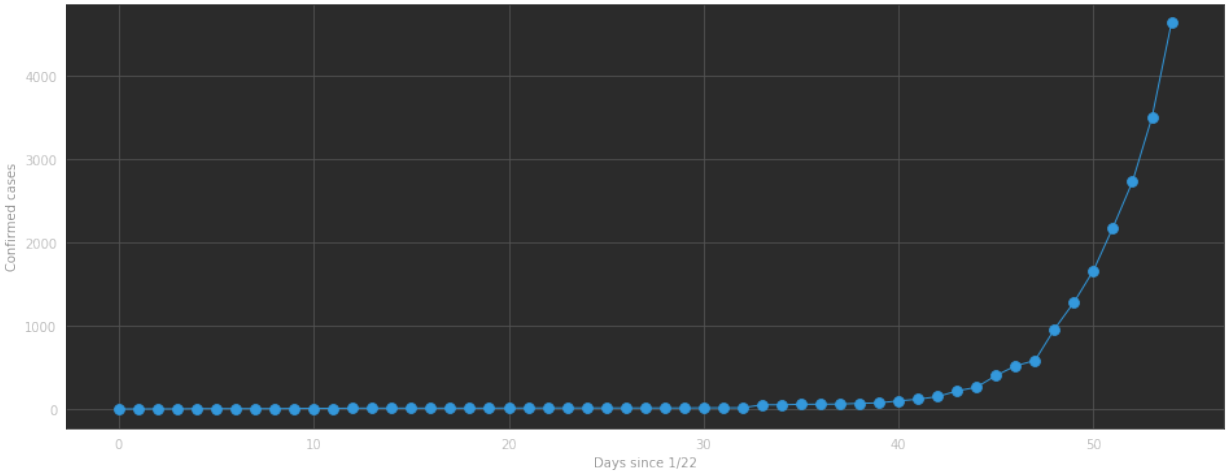
Out[16]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 1
5, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 3
2, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 4
9, 50,
       51, 52, 53, 54])
```

Let's create a plot showing confirmed cases with our country data:

In [17]:

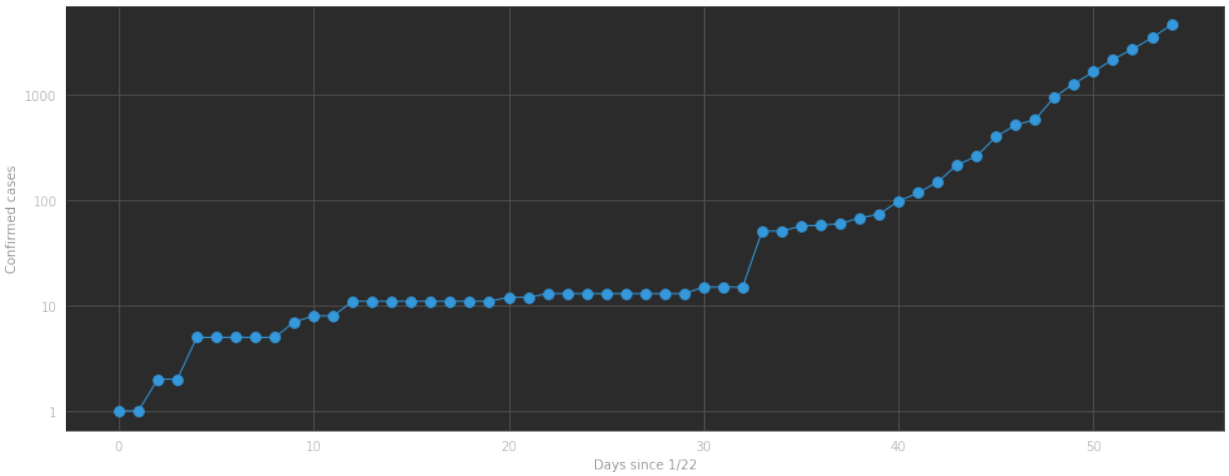
```
fig, ax = plt.subplots(figsize=(16, 6))

sns.lineplot(x=days, y=covid_country['confirmed'],
             markeredgecolor="#3498db", markerfacecolor="#3498db", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#3498db")

plt.suptitle(f"COVID-19 confirmed cases in {COUNTRY} over the time", fontsize=16
, fontweight='bold', color='white')

plt.ylabel('Confirmed cases')
plt.xlabel('Days since 1/22')

plt.show()
```



Show the same plot using a logarithmic scale to see if we got a more linear shape.

In [18]:

```
fig, ax = plt.subplots(figsize=(16, 6))

ax.set(yscale="log")
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda y, _: '{:g}'.format(y))
)
plt.title("(logarithmic scale)", color='white')

sns.lineplot(x=days, y=covid_country['confirmed'],
             markeredgecolor="#3498db", markerfacecolor="#3498db", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#3498db")

plt.suptitle(f"COVID-19 confirmed cases in {COUNTRY} over the time", fontsize=16
, fontweight='bold', color='white')

plt.ylabel('Confirmed cases')
plt.xlabel('Days since 1/22')

plt.show()
```



As the cases have grown with more volume in the last weeks, we won't use the first 30 values (first ~4 weeks).

In [19]:

```
SKIP_DAYS = 30
```

In [20]:

```
covid_country_confirmed_sm = list(covid_country['confirmed'][SKIP_DAYS:])

covid_country_confirmed_sm[:15]
```

Out[20]:

```
[15, 15, 15, 51, 51, 57, 58, 60, 68, 74, 98, 118, 149, 217, 262]
```

Data representation

We'll be using scikit-learn (https://scikit-learn.org/) as tool for our predictive analysis.

Let's define `X` features and `y` labels:

In [21]:

```
X = days[SKIP_DAYS:].reshape(-1, 1)

X
```

Out[21]:

```
array([[30],
       [31],
       [32],
       [33],
       [34],
       [35],
       [36],
       [37],
       [38],
       [39],
       [40],
       [41],
       [42],
       [43],
       [44],
       [45],
       [46],
       [47],
       [48],
       [49],
       [50],
       [51],
       [52],
       [53],
       [54]])
```

In [22]:

```
y = list(np.log(covid_country_confirmed_sm))

y
```

Out[22]:

```
[2.70805020110221,
 2.70805020110221,
 2.70805020110221,
 3.9318256327243257,
 3.9318256327243257,
 4.04305126783455,
 4.060443010546419,
 4.0943445622221,
 4.219507705176107,
 4.30406509320417,
 4.584967478670572,
 4.770684624465665,
 5.003946305945459,
 5.37989735354046,
 5.568344503761097,
 5.996452088619021,
 6.249975242259483,
 6.368187186350492,
 6.8658910748834385,
 7.155396301896734,
 7.416378479192928,
 7.68662133494462,
 7.910957382845589,
 8.160232492367689,
 8.440744019252831]
```

Train and Test split

Now that we have our features and labels defined, let's split them into train and test sets.

In [23]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.1,
                                                    shuffle=False)
```

Our first Machine Learning model

Let's create a simple Linear Regression model using the `LinearRegression` method from `scikit-learn`.

After creating our model, we'll train it using our `X_train` and `y_train` data using the `fit()` method.

In [24]:

```
linear_model = LinearRegression(fit_intercept=True)

linear_model.fit(X_train, y_train)
```

Out[24]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, norma
lize=False)
```

After our model is trained, let's use it to get some predictions:

In [25]:

```
y_pred = linear_model.predict(X_test)

y_pred
```

Out[25]:

```
array([7.62138986, 7.85030141, 8.07921296])
```

We can also check our model's error.

In [26]:

```
print('MAE:', mean_absolute_error(y_pred, y_test))
print('MSE:',mean_squared_error(y_pred, y_test))
```

```
MAE: 0.3203432199895569
MSE: 0.10353711012769458
```

Forecasting next 2 weeks COVID-19 cases

As linear regression formula is:
$$ y = a.x + b $$

We can get the `a` coefficient and `b` interceptor from our model:

In [27]:

```
a = linear_model.coef_
b = linear_model.intercept_
```

With that values, we can forecast new `y` values.

We'll forecast the next 14 days.

In [28]:

```
X_fore = list(np.arange(len(days), len(days) + 14))
y_fore = [(a*x+b)[0] for x in X_fore]

X_fore, y_fore
```

Out[28]:

```
([55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68],
 [8.308124506558874,
  8.537036054088738,
  8.765947601618603,
  8.994859149148466,
  9.223770696678331,
  9.452682244208194,
  9.68159379173806,
  9.910505339267923,
  10.139416886797788,
  10.368328434327651,
  10.597239981857514,
  10.82615152938738,
  11.055063076917243,
  11.283974624447108])
```

Showing our predictions

As our model worked with logarithmic data, we'll need to format that data back to linear scale to interpret it correctly.

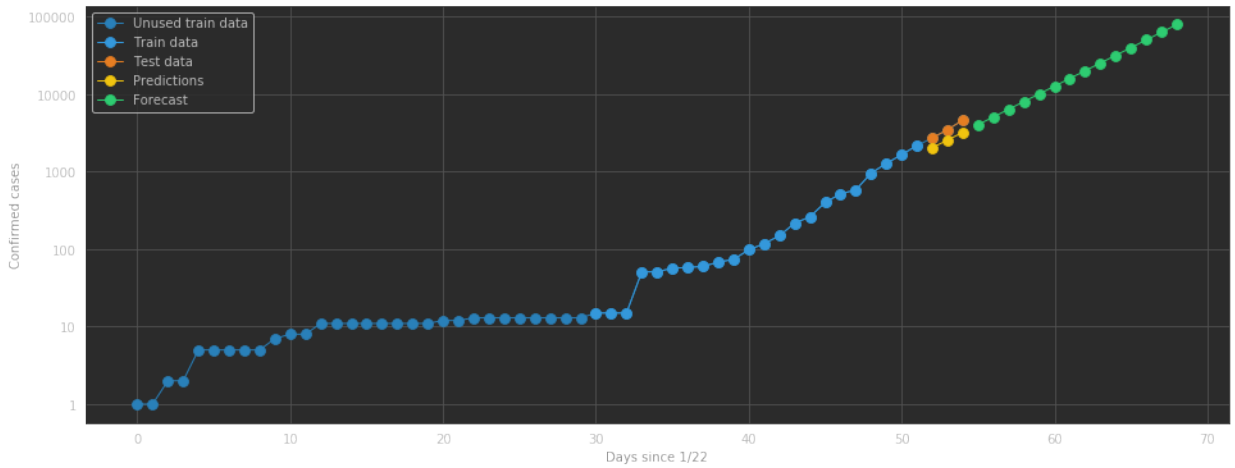
In [29]:

```
y_train_l = list(np.exp(y_train))
y_test_l = list(np.exp(y_test))
y_pred_l = list(np.exp(y_pred))
y_fore_l = list(np.exp(y_fore))
```

If we keep logarithmic scale we won't really understand what the plot is showing and what are the actual number of COVID-19 confirmed cases.

In [33]:

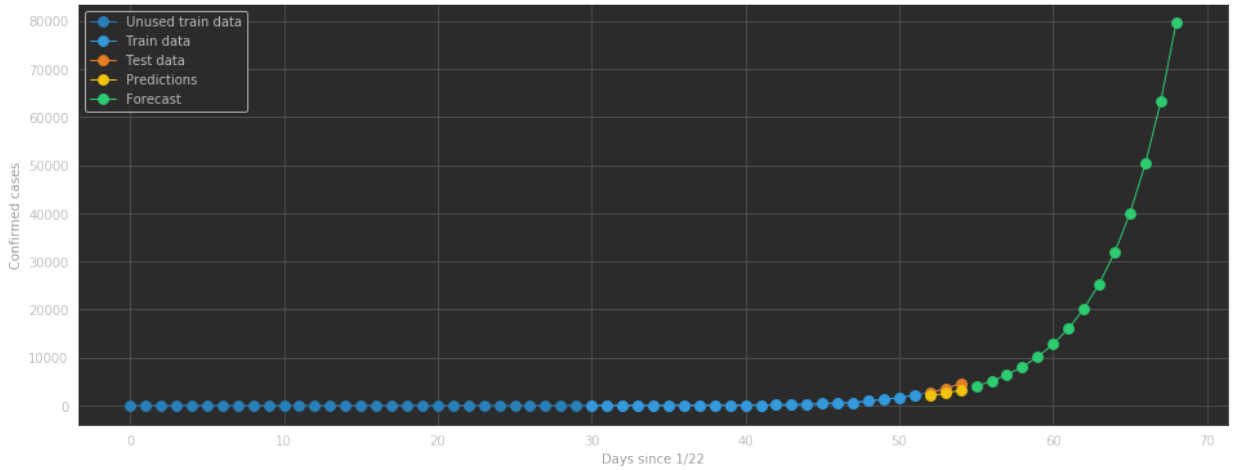
```
fig, ax = plt.subplots(figsize=(16, 6))

ax.set(yscale="log")
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda y, _: '{:g}'.format(y))
)
plt.title("(logarithmic scale)", color='white')

sns.lineplot(x=days, y=covid_country['confirmed'],
             markeredgecolor="#2980b9", markerfacecolor="#2980b9", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#2980b9")

sns.lineplot(x=X_train.reshape(-1), y=y_train_l,
             markeredgecolor="#3498db", markerfacecolor="#3498db", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#3498db")

sns.lineplot(x=X_test.reshape(-1), y=y_test_l,
             markeredgecolor="#e67e22", markerfacecolor="#e67e22", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#e67e22")

sns.lineplot(x=X_test.reshape(-1), y=y_pred_l,
             markeredgecolor="#f1c40f", markerfacecolor="#f1c40f", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#f1c40f")

sns.lineplot(x=X_fore, y=y_fore_l,
             markeredgecolor="#2ecc71", markerfacecolor="#2ecc71", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#2ecc71")

plt.suptitle(f"COVID-19 confirmed cases and forecasting in {COUNTRY} over the ti
me", fontsize=16, fontweight='bold', color='white')

plt.ylabel('Confirmed cases')
plt.xlabel('Days since 1/22')

plt.legend(['Unused train data', 'Train data', 'Test data', 'Predictions', 'Fore
cast'])

plt.show()
```



Now in linear scale we can see corretly the evolution and forecast of COVID-19 confirmed cases.

In [35]:

```
fig, ax = plt.subplots(figsize=(16, 6))

sns.lineplot(x=days, y=covid_country['confirmed'],
             markeredgecolor="#2980b9", markerfacecolor="#2980b9", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#2980b9")

sns.lineplot(x=X_train.reshape(-1), y=y_train_l,
             markeredgecolor="#3498db", markerfacecolor="#3498db", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#3498db")

sns.lineplot(x=X_test.reshape(-1), y=y_test_l,
             markeredgecolor="#e67e22", markerfacecolor="#e67e22", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#e67e22")

sns.lineplot(x=X_test.reshape(-1), y=y_pred_l,
             markeredgecolor="#f1c40f", markerfacecolor="#f1c40f", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#f1c40f")

sns.lineplot(x=X_fore, y=y_fore_l,
             markeredgecolor="#2ecc71", markerfacecolor="#2ecc71", markersize=8,
marker="o",
             sort=False, linewidth=1, color="#2ecc71")

plt.suptitle(f"COVID-19 confirmed cases and forecasting in {COUNTRY} over the ti
me", fontsize=16, fontweight='bold', color='white')

plt.ylabel('Confirmed cases')
plt.xlabel('Days since 1/22')

plt.legend(['Unused train data', 'Train data', 'Test data', 'Predictions', 'Fore
cast'])
plt.savefig('reg.svg', format='svg', dpi=1200)
plt.show()
```


Unused train data
Train data
Test data
Predictions
Forecast
Confirmed cases
Days since 1/22
0
10000
20000
30000
40000
50000
60000
70000
80000
0
10
20
30
40
50
60
70