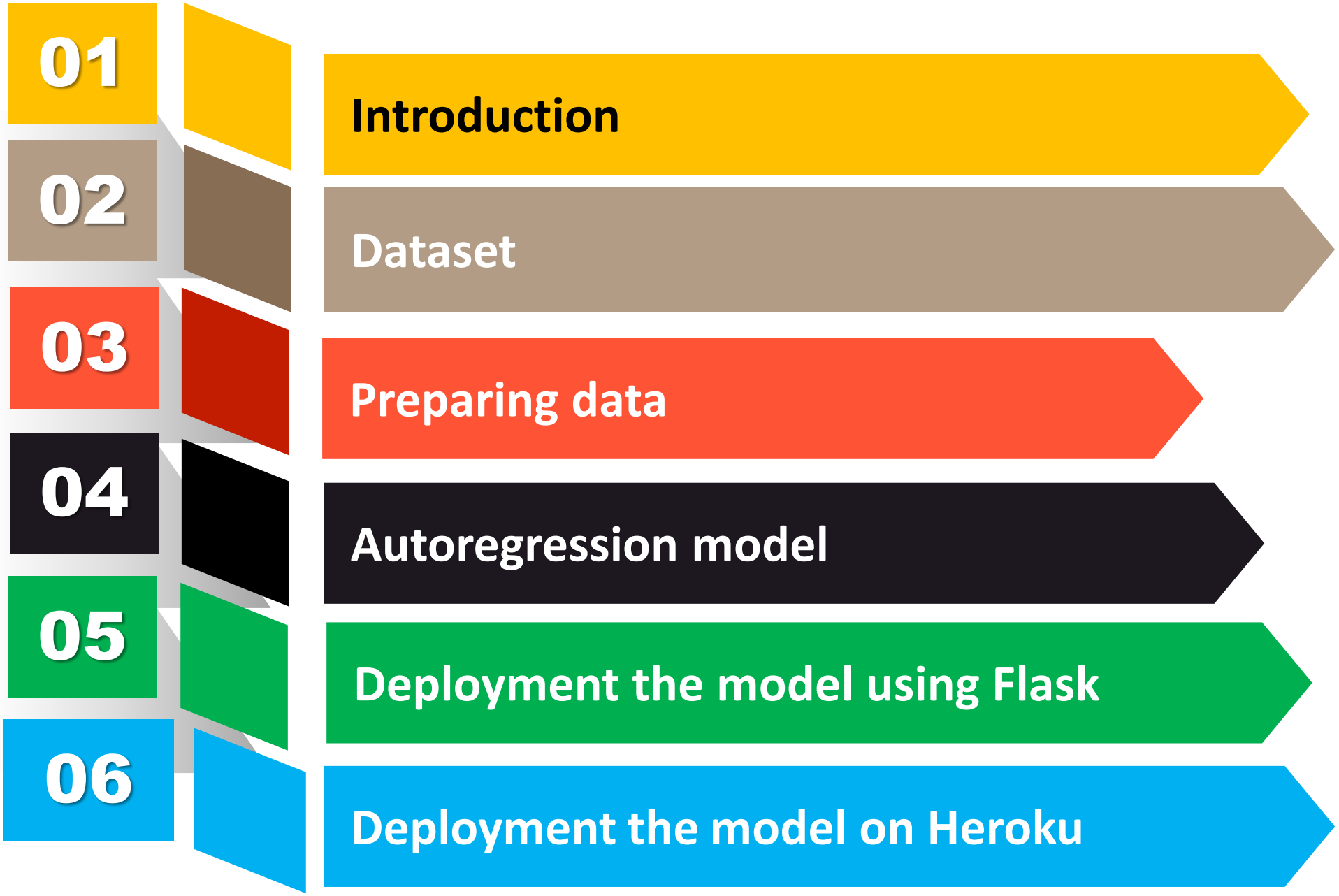
Working with Large Files & Data Ingestion Pipeline

Alireza Ehiaei

19-July-2021

Table Of Contents



Introduction

Loading Big Data or trying to read very big (more than 1 GB) CSV files usually arise Memory Errors while. There are some ways how to handle it in Python environment.

In this project different ways of reading of huge data files are reviewed and by creating a pipeline some basic data manipulation is provided.

Dataset

- Data set includes data of population, both sexes, of regions, subregions, countries or areas around the world from 1950 to 2020 including 289 rows.
- The data source is:

  https://dataverse.harvard.edu/dataverse/atlas/?q=country_hsproduct6digit_year

  https://dataverse.harvard.edu/dataverse/atlas/?q=country_partner_sitcproduct2digit_year
- Data and code are uploaded at: https://github.com/Alireza-Ehiaei/Data_Sciences/tree/main/Machine_learning/Deploy_on_Heroku.
- The web app created in this project:

  https://estimation-ml.herokuapp.com/

Dataset

- The Atlas of Economic Complexity maintains trade data in multiple international classification systems. This data set contains trade flows classified via Harmonized System (HS) 1992. HS data offers a contemporary and detailed classification of goods, but covers a relatively short time period:
- Categorizes approximately 5,000 goods
- Covers years from 1995–2018
- Categories break down to 1-, 2-, 4-, or 6-digit detail levels (though country reporting can be less reliable at the 6-digit level)
- Raw data on trade in goods is provided by United Nations Statistical Division (COMTRADE). The data is then cleaned by Growth Lab researchers using the Bustos-Yildirim Method which uses bilateral trade flows to account for inconsistent reporting and provides more reliable accounting.
- In addition to trade in goods, the data additionally contains unilateral data on services trade provided by the International Monetary Fund (IMF) and acquired through the World Development Indicators (WDI) of The World Bank.
- For further information, see the data information page on the Atlas website.

Process

First, we have a block of codes for downloading and preparing data, then some codes for the auto regression model, and the creating Flask app.

Finally, the steps of deploying the app on Heroku using git is explained.

Preparing data

In order to be able to run auto regression model we need time series data format preferably by indexing time, so that each column has population of one country. So, a function with name ‘panel’ is defined to convert data from their initial format into the desired format. The code is in next page and converts the initial format below:

Index	Variant	Region, subr	Notes	Country code	Type	Parent code	1950	1951	1952	1953	1
1	Estimates	WORLD		900	World	0	2 536 431	2 584 034	2 630 862	2 677 609	2 724 84
2	Estimates	UN developr	a	1803	Label/Separa	900	...	...	...	...	...
3	Estimates	More develc	b	901	Developmer	1803	814 819	824 004	833 720	843 788	854 060
4	Estimates	Less develop	c	902	Developmer	1803	1 721 612	1 760 031	1 797 142	1 833 822	1 870 78
5	Estimates	Least develo	d	941	Developmer	902	195 428	199 180	203 015	206 986	211 133
6	Estimates	Less develop	e	934	Developmer	902	1 526 184	1 560 850	1 594 126	1 626 836	1 659 65
7	Estimates	Less developed regions, e		948	Developmer	1803	1 157 420	1 179 933	1 203 963	1 229 440	1 256 30
8	Estimates	Land-locked	f	1636	Special othe	1803	103 803	105 870	108 079	110 423	112 894
9	Estimates	Small Island	g	1637	Special othe	1803	23 771	24 209	24 685	25 187	25 710
10	Estimates	World Bank income group		1802	Label/Separa	900	...	...	...	...	...

Preparing data

```
app = Flask(__name__)
#### Creating time series panel data

def get_df_name(df):
    name =[x for x in globals() if globals()[x] is df][0]
    return name

def panel(df):

    name = get_df_name(df)
    df = df.drop(columns=['Index', 'Type','Variant','Notes', 'Country code','Parent code' ])

     # rmove all rows with NaN cells
    df= df.dropna(axis=0)

    df = df.rename(columns={'Region, subregion, country or area':'country' })
    # rows having '...' are not deleted, delete by:
    #df =df.loc[~((df['country']=='...') | (df['year']=='...') | (df['Region, subregion, country or area']

    df['country'] = df['country'].replace({'Iran, Islamic Rep.': 'Iran'})
    dft = df.T
    new_header = dft.iloc[0]
    dft = dft[1:]
    dft.columns = new_header

    dft= dft.loc[:, ~(dft == '...').any()]

    for col in range(len(dft.columns)):
        dft.iloc[:,col] = dft.iloc[:,col].str.replace(' ','')

    return dft


### Reading data
pop = pd.read_csv(r"https://raw.githubusercontent.com/Alireza-Ehiaei/Machine_learning/master/WPP2019_POP_F

panel = panel(pop)
```

Preparing data

Data is ready to be used in model:

country	WORLD	More developed regions	Less developed regions	Least developed countries	Less developed regions, excluding least developed countries	Less developed regions, excluding China	Land-locked Developing Countries (LLDC)	Small Island Developing States (SIDS)
1950	2536431	814819	1721612	195428	1526184	1157420	103803	23771
1951	2584034	824004	1760031	199180	1560850	1179933	105870	24209
1952	2630862	833720	1797142	203015	1594126	1203963	108079	24685
1953	2677609	843788	1833822	206986	1626836	1229440	110423	25187
1954	2724847	854060	1870786	211133	1659653	1256303	112894	25710
...	...	...	...	...	...	...	...	...

Autoregression model

In this research, the AutoReg model is used to learn from previous data of the population of each country in order to build a forecasting model. Since each country has its own population dynamics, it may have a different autoregression model, so if we save the model of one country with pickle and use it to forecast the population of other countries, the estimations won't be reliable. In this research the function named 'autoreg' is defined to built the model and will be called for each country in order to build the model separately.

Autoregression model

autoreg model:

```
def autoreg(df,list_country,a,b):

    from statsmodels.tsa.ar_model import AutoReg
    from random import random
    #import pickle
    import json

    yhat =[]
    yhat= pd.DataFrame(yhat)
    # contrived dataset
    data = panel[panel.columns[panel.columns.isin(list_country)]]
    data = data.astype(int).reset_index(drop=True)
    # fit model
    for col in data.columns:
        model_AutoReg = AutoReg(data.loc[:,col], lags=1,old_names=False)
        model_fit = model_AutoReg.fit()

    #if there is just one model (one time series) save the model with pickle and call it in app:
    # pickle.dump(model_AutoReg , open('model.pkl','wb'))

    # make prediction
        yhat.loc[:,col] = model_fit.predict(a,b)
```

Autoregression model

The auto regression model can be tested before creating Flask app.

```
: 1 autoreg(panel,['WORLD', 'More developed regions'], 80,85)
```

For example, the prediction for the first two columns for the time periods 2030 – 2035 is:

years	WORLD	More developed regions
2030	8,745,733,000	1,304,051,000
2031	8,844,029,000	1,306,841,000
2032	8,942,922,000	1,309,584,000
2033	9,042,415,000	1,312,279,000
2034	9,142,512,000	1,314,927,000
2035	9,243,217,000	1,317,530,000

Deployment the model using Flask

After importing liberaries as well Flask, an instance of the *Flask* class is defined at tope of the code to be able to call the app on heroku. for this app the index.html, layout.html and view.html files are saved in templates folder and css file in static folder both whitin the main folder that includes the main file (app.py).

In the first part of app, ‘index’ function is called that asks from the user to enter the set of countries and years through the index.html file.

```
@app.route('/')
def index():
    return render_template(
        'index.html',
        data= list(panel.columns))
```

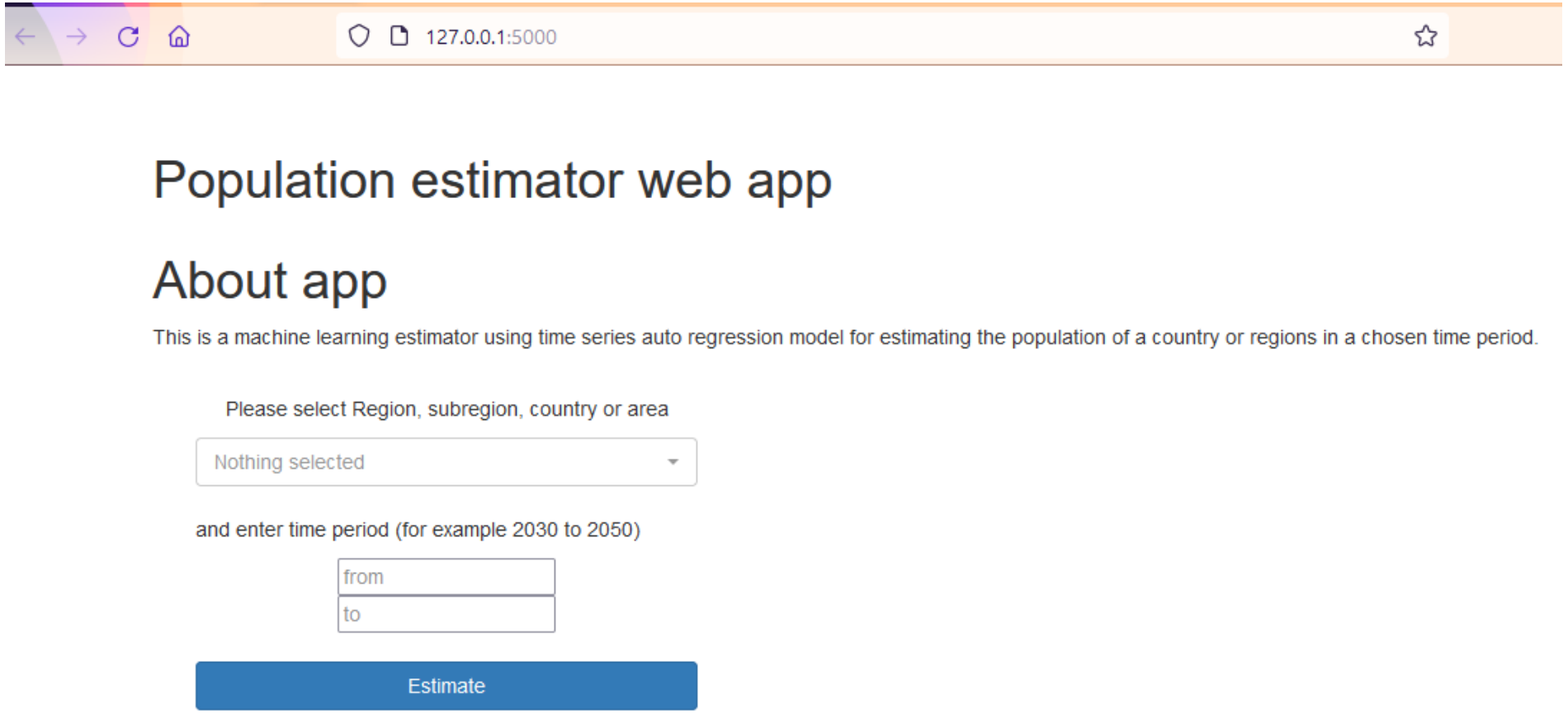
Deployment the model using Flask

index.html file:

```
<div class="container">

    <div class="col-md-4" style="text-align: center">
        <form action="{{ url_for('test') }}" method="post" id="multiple_select_form" 
        <br />
        <p>Please select Region, subregion, country or area. </p>
         <p>(sometimes Firefox does not allow multi selection, please choose another 
        <select multiple name="countries" class="form-control selectpicker" data-live-
            {% for o in data %}
               <option value="{{ o }}">{{ o }}</option>
            {% endfor %}

        </select>
        <br /><br />

        <p>and enter time period  (for example 2030 to 2050)</p>
        <input type="text" name="from" placeholder="from" required="required" /><br>
        <input type="text" name="to" placeholder="to" required="required" /><br>
        <br />

        <button type="submit" class="btn btn-primary btn-block">Estimate</button>
        </form>
        <br />
```

Deployment the model using Flask

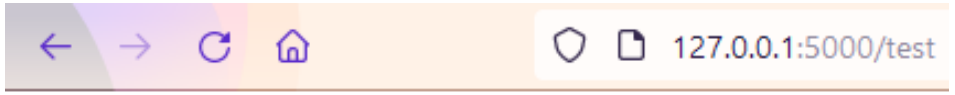
The page that is created locally by the codes above:



Deployment the model using Flask

Then the data entered by the user will be sent to the test route of the app and the autoregression model will be executed on the selected data and the result will be returned based on settings defined in layout.html and view.html and style.css.

For example, for the first two columns from 2030 to 2035 are:



Estimation

Estimation of your selected data is:

	WORLD	More developed regions
years		
2030	8,745,733,000	1,304,051,000
2031	8,844,029,000	1,306,841,000
2032	8,942,922,000	1,309,584,000
2033	9,042,415,000	1,312,279,000
2034	9,142,512,000	1,314,927,000
2035	9,243,217,000	1,317,530,000

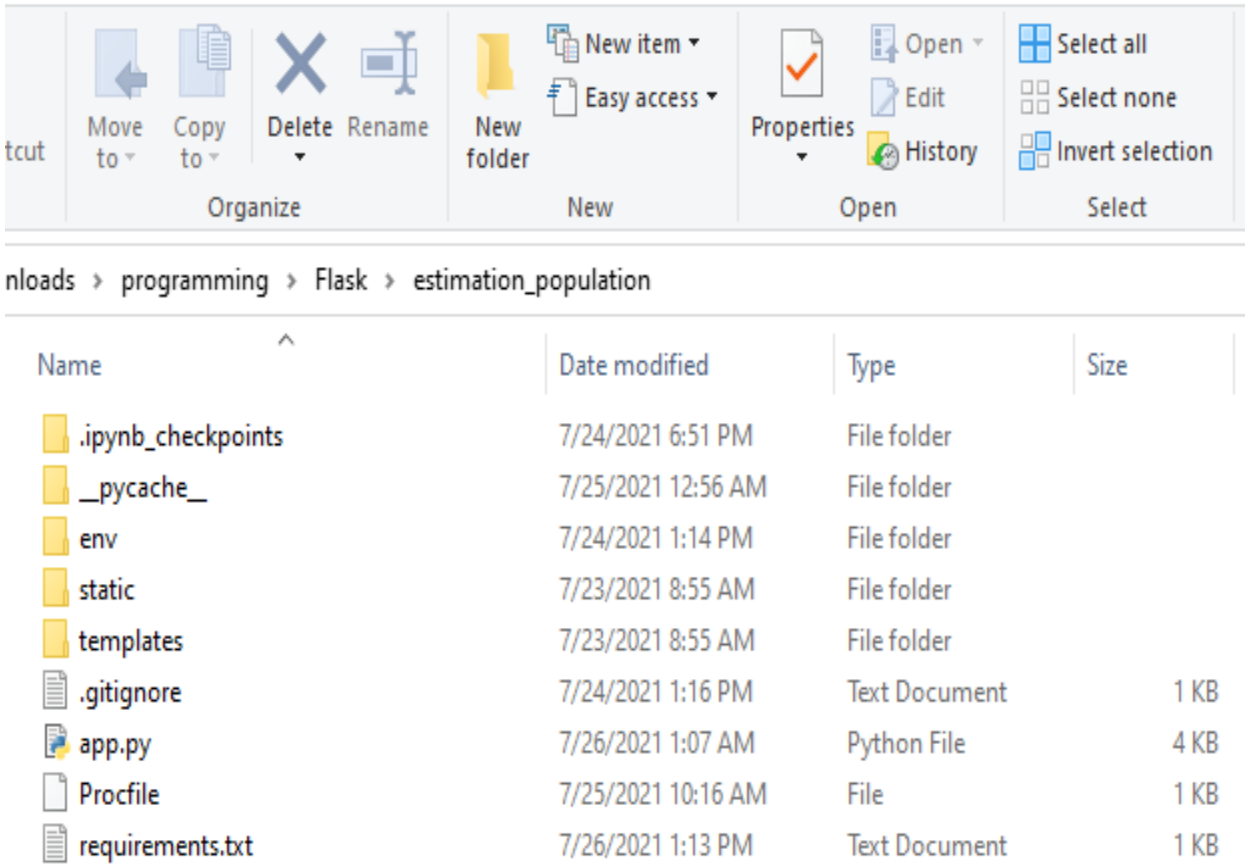
Deployment the flask app on Heroku

Heroku is a cloud platform as a service supporting several programming languages. Developers use Heroku to deploy, manage, and scale modern apps.

There are different ways to deploy an app on Heroku, it can be done using heroku website and connecting it to the github pages that all files and template and static folders are uploaded. We used git in this project.

Deployment the flask app on Heroku

First, the command prompt is applied and the path of the folder in personal devices (laptop) including all files is sent to it, then a virtual environment is created, and by using gunicorn in Procfile file Heroku receives the order of executing the app. Requirements.txt determine the libraries the should be installed by the app that Heroku needs to apply, the folder looks like this:

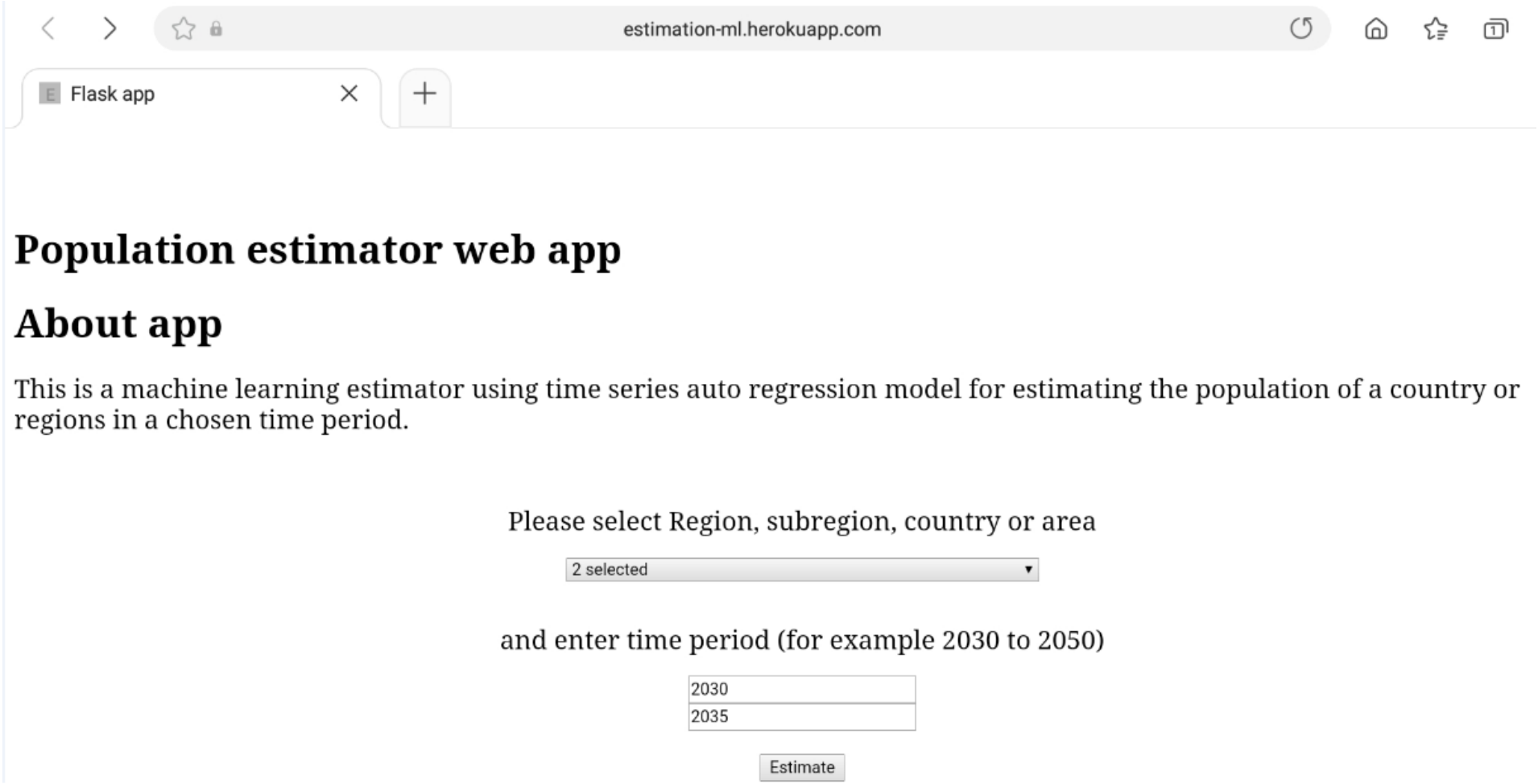


Deployment the flask app on Heroku

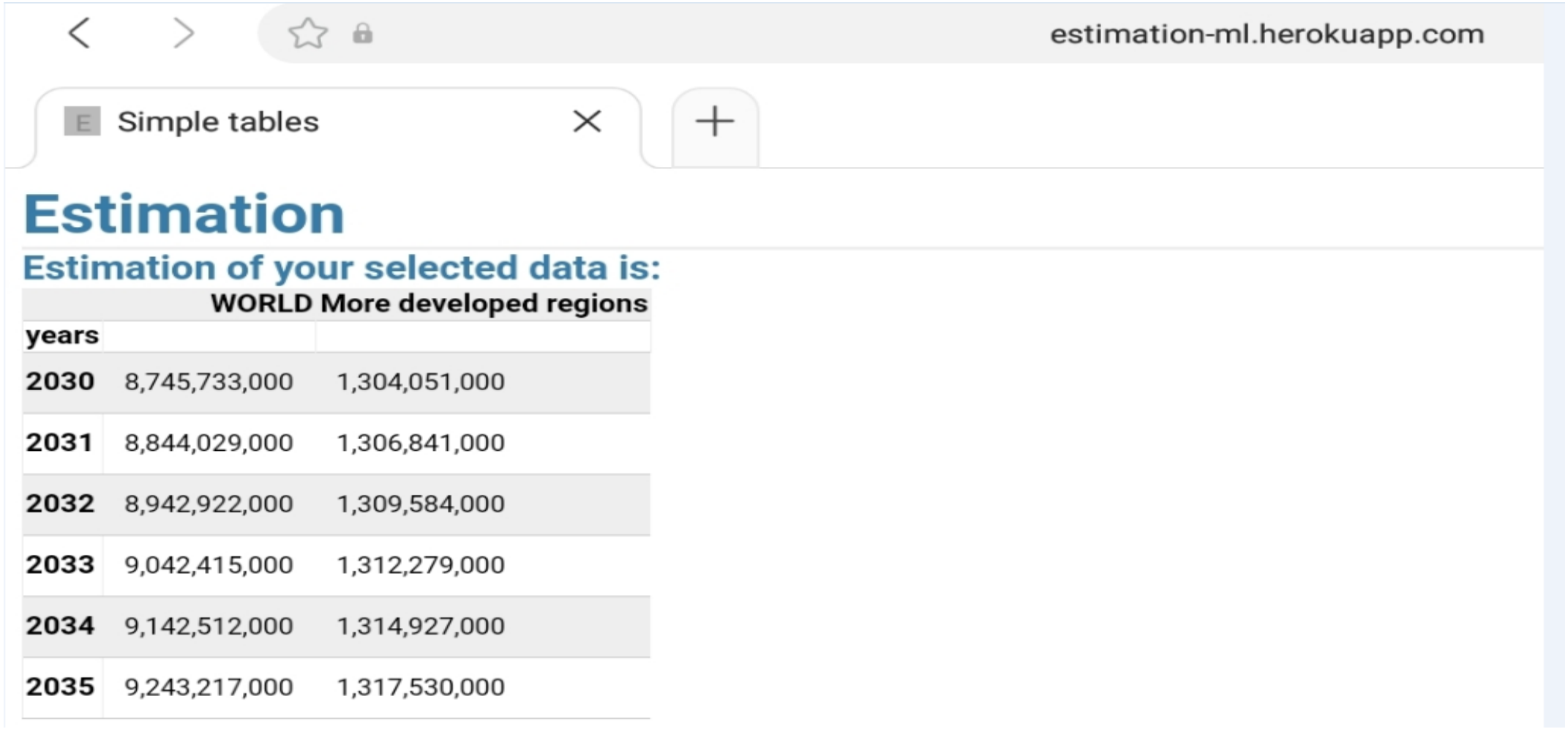
In the command prompt, we login to Heroku account that we should create it on the Heroku website, then an app is created on Heroku called estimation-ml, then add and commit all files and their changes to the remote git and by pushing them to the Heroku, the app will be created and is accessible through internet for others. The first page of the app and the result for the first two regions on Heroku are provided in the following slides.

Note, Firefox and internet explorer sometimes do not execute Heroku apps properly, please choose another internet browser.

Deployment the the flask app on Heroku



Deployment the the flask app on Heroku



	WORLD	More developed regions
years		
2030	8,745,733,000	1,304,051,000
2031	8,844,029,000	1,306,841,000
2032	8,942,922,000	1,309,584,000
2033	9,042,415,000	1,312,279,000
2034	9,142,512,000	1,314,927,000
2035	9,243,217,000	1,317,530,000

Thank You