Porfolio

Edición Interactiva

Roberto Ramirez Martinez

2024-12-31

Table of contents

1 Porfolio **5**

2 Particules **6**

4 Introduction **8**

5 Introduccion **9**

6 Desarrollo **10**

7 Conclusion **11**

I Python **12**

8 python **13**

9 Python **14**

9.1 header 14
9.2 py1 14
9.3 py2 15
9.4 py3 15
9.5 py4 16
9.6 py5 16
9.7 py6 16

10 Code example 1-1 **17**

11 Note: you can ignore the rest of this notebook, it just generates many of the figures in chapter 1. **24**

12 Load and prepare Life satisfaction data **25**

13 Load and prepare GDP per capita data **26**

II R 40

14 html 41

III Bash 42

15 bash 43

IV Javascript 44

16 title 45
16.1 g1 45
16.2 g2 45
16.3 g3 45
16.4 g4 45
16.5 g5 45
16.6 g6 45

V D3 46

17 Graph1 47
17.1 graph1 47
17.2 graph2 47
17.3 graph3 47
17.4 graph4 47
17.5 graph5 47
17.6 graph6 47

18 Index-Graph-3D 48
18.1 graph1 48
18.2 graph2 48
18.3 graph3 48
18.4 graph4 48
18.5 graph5 48
18.6 graph6 48

19 title 49
19.1 g1 49
19.2 g2 49
19.3 g3 49
19.4 g4 49

19.5 g5 49
19.6 g6 49

VI HTML 50

20 HTML 1 51
20.1 D1 51
20.2 D2 51
20.3 D3 51
20.4 D4 51
20.5 D5 51
20.6 D6 51
20.7 D7 51
20.8 D8 51
20.9 D9 51
20.10D10 51

21 H2 HTML 52
21.1 H1 52
21.2 H2 52
21.3 H3 52
21.4 H4 52
21.5 H5 52
21.6 H6 52
21.7 H7 52
21.8 H8 52
21.9 H9 52
21.10H10 52

22 Animacion and AWS 53
22.1 H1 54
22.2 H2 54
22.3 H3 54
22.4 H4 54
22.5 H5 54
22.6 H6 54

1 Porfolio

Edición Interactiva

2 Particules

```python
import numpy as np

a = np.array([2, 4])
b = np.array([-1, 3])
v = a + b
print(f"Vector resultante de los vectores a + b {v}")
```

```
Vector resultante de los vectores a + b [1 7]
```

3

4 Introduction

5 Introduccion

6 Desarrollo

7 Conclusion

Part I

Python

8 python

9 Python

9.1 header

9.2 py1

```
# Load plotly
import plotly.graph_objects as go

# Sample data
x = [1.5, 2.9, 3, 4.2, 5.6]
y = [2.2, 13.3, 4.4, 55.3, 52.1]

# Initialize a figure
fig = go.Figure()

# Add the scatter trace
fig.add_trace(go.Scatter(
    x=x, # Variable in the x-axis
    y=y, # Variable in the y-axis
    mode='markers', # This explicitly states that we want our observations to be represented
))

# Show
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

Note

This is an example of an tip callout

9.3 py2

```
# Load plotly
import plotly.graph_objects as go

# Sample data
x = [1.5, 2.9, 3, 4.2, 5.6]
y = [2.2, 13.3, 4.4, 55.3, 52.1]

# Initialize a figure
fig = go.Figure()

# Add the scatter trace
fig.add_trace(go.Scatter(
    x=x, # Variable in the x-axis
    y=y, # Variable in the y-axis
    mode='markers', # This explicitly states that we want our observations to be represented
))

# Show
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

9.4 py3

```
# import the plotly express library
import plotly.express as px

# Some dummy data
categories = ['A', 'B', 'C', 'D', 'E']
values = [15, 22, 18, 12, 28]

# Plot
fig = px.bar(
  x=categories,
  y=values,
)

fig.show()
```

```
Unable to display output for mime type(s): text/html
```

9.5 py4

9.6 py5

9.7 py6

10 Code example 1-1

Chapter 1 – The Machine Learning landscape

This is the code used to generate some of the figures in chapter 1.

Although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead.

```
# Python  3.5 is required
import sys
assert sys.version_info >= (3, 5)
```

```
# Scikit-Learn  0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"
```

This function just merges the OECD's life satisfaction data and the IMF's GDP per capita data. It's a bit too long and boring and it's not specific to Machine Learning, which is why I left it out of the book.

```
def prepare_country_stats(oecd_bli, gdp_per_capita):
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
    gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
    gdp_per_capita.set_index("Country", inplace=True)
    full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                                  left_index=True, right_index=True)
    full_country_stats.sort_values(by="GDP per capita", inplace=True)
    remove_indices = [0, 1, 6, 8, 33, 34, 35]
    keep_indices = list(set(range(36)) - set(remove_indices))
    return full_country_stats[["GDP per capita", 'Life satisfaction']].iloc[keep_indices]
```

The code in the book expects the data files to be located in the current directory. I just tweaked it here to fetch the files in `datasets/lifesat`.

```
import os
datapath = os.path.join("datasets", "lifesat", "")
```

```
# To plot pretty figures directly within Jupyter
%matplotlib inline
import matplotlib as mpl
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

```
# Download the data
import urllib.request
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
os.makedirs(datapath, exist_ok=True)
for filename in ("oecd_bli_2015.csv", "gdp_per_capita.csv"):
    print("Downloading", filename)
    url = DOWNLOAD_ROOT + "datasets/lifesat/" + filename
    urllib.request.urlretrieve(url, datapath + filename)
```

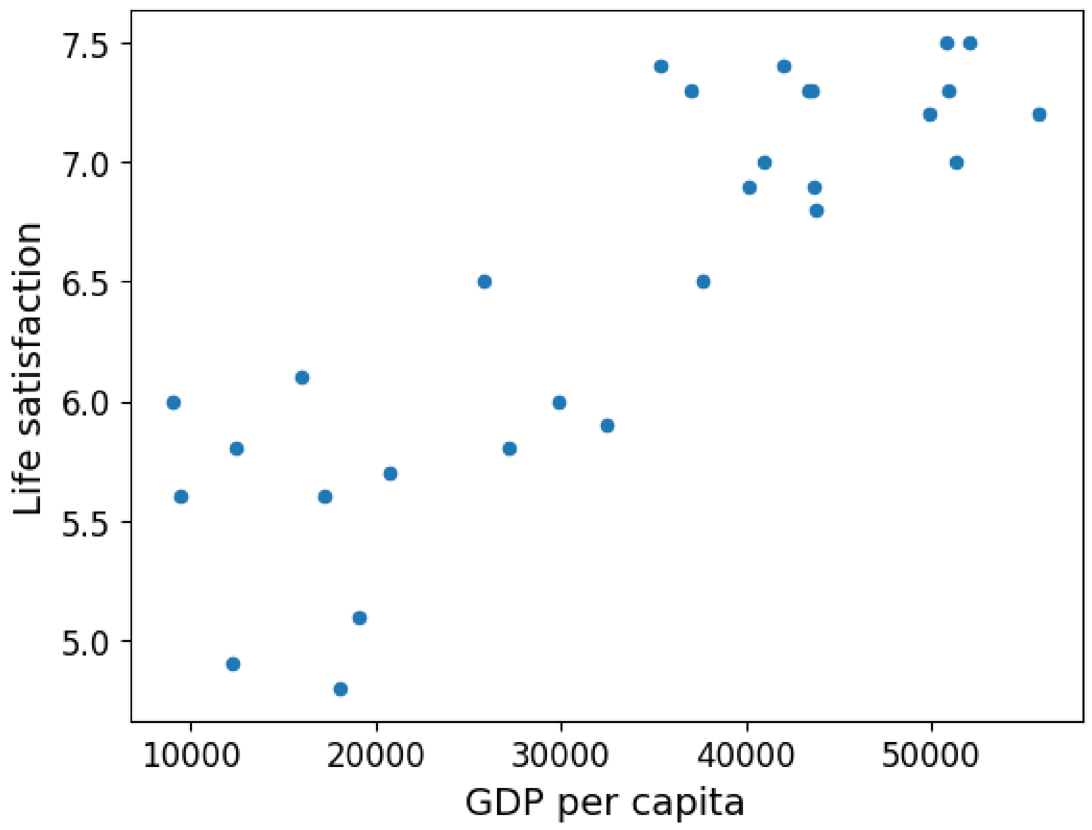
```
Downloading oecd_bli_2015.csv

URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable
---------------------------------------------------------------------------
SSLCertVerificationError                  Traceback (most recent call last)
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/urllib/request.py:131
   1318 try:
-> 1319     h.request(req.get_method(), req.selector, req.data, headers,
   1320               encode_chunked=req.has_header('Transfer-encoding'))
   1321 except OSError as err: # timeout error
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/http/client.py:1336,
   1335 """Send a complete request to the server."""
-> 1336 self._send_request(method, url, body, headers, encode_chunked)
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/http/client.py:1382,
   1381     body = _encode(body, 'body')
-> 1382 self.endheaders(body, encode_chunked=encode_chunked)
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/http/client.py:1331,
   1330     raise CannotSendHeader()
-> 1331 self._send_output(message_body, encode_chunked=encode_chunked)
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/http/client.py:1091,
   1090 del self._buffer[:]
-> 1091 self.send(msg)
```

```
   1093 if message_body is not None:
   1094
   1095     # create a consistent interface to message_body
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/http/client.py:1035, i
   1034 if self.auto_open:
-> 1035     self.connect()
   1036 else:
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/http/client.py:1477, i
   1475     server_hostname = self.host
-> 1477 self.sock = self._context.wrap_socket(self.sock,
   1478                                       server_hostname=server_hostname)
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/ssl.py:455, in SSLCon
    449 def wrap_socket(self, sock, server_side=False,
    450                 do_handshake_on_connect=True,
    451                 suppress_ragged_eofs=True,
    452                 server_hostname=None, session=None):
    453     # SSLSocket class handles server_hostname encoding before it calls
    454     # ctx._wrap_socket()
--> 455     return self.sslsocket_class._create(
    456         sock=sock,
    457         server_side=server_side,
    458         do_handshake_on_connect=do_handshake_on_connect,
    459         suppress_ragged_eofs=suppress_ragged_eofs,
    460         server_hostname=server_hostname,
    461         context=self,
    462         session=session
    463     )
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/ssl.py:1076, in SSLSo
   1075                 raise ValueError("do_handshake_on_connect should not be specified fo
-> 1076             self.do_handshake()
   1077 except:
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/ssl.py:1372, in SSLSo
   1371         self.settimeout(None)
-> 1372     self._sslobj.do_handshake()
   1373 finally:
SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable
During handling of the above exception, another exception occurred:
URLError                                  Traceback (most recent call last)
Cell In[14], line 8
      6 print("Downloading", filename)
      7 url = DOWNLOAD_ROOT + "datasets/lifesat/" + filename
----> 8 urllib.request.urlretrieve(url, datapath + filename)
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/urllib/request.py:214
```

```
    197 """
    198 Retrieve a URL into a temporary location on disk.
    199
   (...)
    210 data file as well as the resulting HTTPMessage object.
    211 """
    212 url_type, path = _splittype(url)
--> 214 with contextlib.closing(urlopen(url, data)) as fp:
    215     headers = fp.info()
    217     # Just return the local path and the "headers" for file://
    218     # URLs. No sense in performing a copy unless requested.
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/urllib/request.py:189
    187 else:
    188     opener = _opener
--> 189 return opener.open(url, data, timeout)
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/urllib/request.py:489
    486     req = meth(req)
    488 sys.audit('urllib.Request', req.full_url, req.data, req.headers, req.get_method())
--> 489 response = self._open(req, data)
    491 # post-process response
    492 meth_name = protocol+"_response"
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/urllib/request.py:506
    503     return result
    505 protocol = req.type
--> 506 result = self._call_chain(self.handle_open, protocol, protocol +
    507                           '_open', req)
    508 if result:
    509     return result
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/urllib/request.py:466
    464 for handler in handlers:
    465     func = getattr(handler, meth_name)
--> 466     result = func(*args)
    467     if result is not None:
    468         return result
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/urllib/request.py:136
   1366 def https_open(self, req):
-> 1367     return self.do_open(http.client.HTTPSConnection, req,
   1368                         context=self._context)
File /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/urllib/request.py:132
   1319         h.request(req.get_method(), req.selector, req.data, headers,
   1320                   encode_chunked=req.has_header('Transfer-encoding'))
   1321     except OSError as err: # timeout error
-> 1322         raise URLError(err)
```

```
   1323     r = h.getresponse()
   1324 except:
URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable
```

```python
# Code example
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model

# Load the data
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv(datapath + "gdp_per_capita.csv",thousands=',',delimiter='\t',
                             encoding='latin1', na_values="n/a")

# Prepare the data
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]

# Visualize the data
country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction')
plt.show()

# Select a linear model
model = sklearn.linear_model.LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[22587]]  # Cyprus' GDP per capita
print(model.predict(X_new)) # outputs [[ 5.96242338]]
```



```
[[5.96242338]]
```

Replacing the Linear Regression model with k-Nearest Neighbors (in this example, k = 3) regression in the previous code is as simple as replacing these two lines:

```
import sklearn.linear_model
model = sklearn.linear_model.LinearRegression()
```

with these two:

```
import sklearn.neighbors
model = sklearn.neighbors.KNeighborsRegressor(n_neighbors=3)
```

```
# Select a 3-Nearest Neighbors regression model
import sklearn.neighbors
model1 = sklearn.neighbors.KNeighborsRegressor(n_neighbors=3)
```

```
# Train the model
model1.fit(X,y)

# Make a prediction for Cyprus
print(model1.predict(X_new)) # outputs [[5.76666667]]
```

```
[[5.76666667]]
```

11 Note: you can ignore the rest of this notebook, it just generates many of the figures in chapter 1.

Create a function to save the figures.

```python
# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "fundamentals"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Make this notebook's output stable across runs:

```python
np.random.seed(42)
```

12 Load and prepare Life satisfaction data

If you want, you can get fresh data from the OECD's website. Download the CSV from http://stats.oecd.org/index.aspx?DataSetCode=BLI and save it to `datasets/lifesat/`.

```python
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
oecd_bli.head(2)
```

Indicator	Air pollution	Assault rate	Consultation on rule-making	Dwellings without basic facilities	Ed
Country					
Australia	13.0	2.1	10.5	1.1	76.
Austria	27.0	3.4	7.1	1.0	83.

```python
oecd_bli["Life satisfaction"].head()
```

	Life satisfaction
Country	
Australia	7.3
Austria	6.9
Belgium	6.9
Brazil	7.0
Canada	7.3

13 Load and prepare GDP per capita data

Just like above, you can update the GDP per capita data if you want. Just download data from http://goo.gl/j1MSKe (=> imf.org) and save it to `datasets/lifesat/`.

```
gdp_per_capita = pd.read_csv(datapath+"gdp_per_capita.csv", thousands=',', delimiter='\t',
                             encoding='latin1', na_values="n/a")
gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
gdp_per_capita.set_index("Country", inplace=True)
gdp_per_capita.head(2)
```

	Subject Descriptor	Units	Scale	Country/Series-specifi
Country				
Afghanistan	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross d
Albania	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross d

```
full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita, left_index=True, right_in
full_country_stats.sort_values(by="GDP per capita", inplace=True)
full_country_stats
```

	Air pollution	Assault rate	Consultation on rule-making	Dwellings without basic facilit
Country				
Brazil	18.0	7.9	4.0	6.7
Mexico	30.0	12.8	9.0	4.2
Russia	15.0	3.8	2.5	15.1
Turkey	35.0	5.0	5.5	12.7
Hungary	15.0	3.6	7.9	4.8
Poland	33.0	1.4	10.8	3.2
Chile	46.0	6.9	2.0	9.4
Slovak Republic	13.0	3.0	6.6	0.6
Czech Republic	16.0	2.8	6.8	0.9
Estonia	9.0	5.5	3.3	8.1
Greece	27.0	3.7	6.5	0.7

Country	Air pollution	Assault rate	Consultation on rule-making	Dwellings without basic facilit
Portugal	18.0	5.7	6.5	0.9
Slovenia	26.0	3.9	10.3	0.5
Spain	24.0	4.2	7.3	0.1
Korea	30.0	2.1	10.4	4.2
Italy	21.0	4.7	5.0	1.1
Japan	24.0	1.4	7.3	6.4
Israel	21.0	6.4	2.5	3.7
New Zealand	11.0	2.2	10.3	0.2
France	12.0	5.0	3.5	0.5
Belgium	21.0	6.6	4.5	2.0
Germany	16.0	3.6	4.5	0.1
Finland	15.0	2.4	9.0	0.6
Canada	15.0	1.3	10.5	0.2
Netherlands	30.0	4.9	6.1	0.0
Austria	27.0	3.4	7.1	1.0
United Kingdom	13.0	1.9	11.5	0.2
Sweden	10.0	5.1	10.9	0.0
Iceland	18.0	2.7	5.1	0.4
Australia	13.0	2.1	10.5	1.1
Ireland	13.0	2.6	9.0	0.2
Denmark	15.0	3.9	7.0	0.9
United States	18.0	1.5	8.3	0.1
Norway	16.0	3.3	8.1	0.3
Switzerland	20.0	4.2	8.4	0.0
Luxembourg	12.0	4.3	6.0	0.1

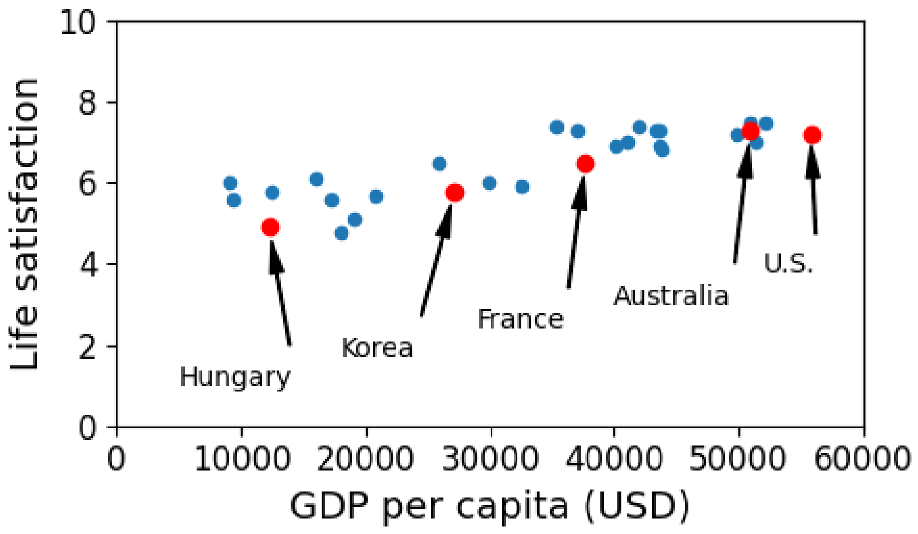
```
full_country_stats[["GDP per capita", 'Life satisfaction']].loc["United States"]
```

	United States
GDP per capita	55805.204
Life satisfaction	7.200

```
remove_indices = [0, 1, 6, 8, 33, 34, 35]
keep_indices = list(set(range(36)) - set(remove_indices))

sample_data = full_country_stats[["GDP per capita", 'Life satisfaction']].iloc[keep_indices]
missing_data = full_country_stats[["GDP per capita", 'Life satisfaction']].iloc[remove_indic
```

```
sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction', figsize=(5,3))
plt.axis([0, 60000, 0, 10])
position_text = {
    "Hungary": (5000, 1),
    "Korea": (18000, 1.7),
    "France": (29000, 2.4),
    "Australia": (40000, 3.0),
    "United States": (52000, 3.8),
}
for country, pos_text in position_text.items():
    pos_data_x, pos_data_y = sample_data.loc[country]
    country = "U.S." if country == "United States" else country
    plt.annotate(country, xy=(pos_data_x, pos_data_y), xytext=pos_text,
            arrowprops=dict(facecolor='black', width=0.5, shrink=0.1, headwidth=5))
    plt.plot(pos_data_x, pos_data_y, "ro")
plt.xlabel("GDP per capita (USD)")
save_fig('money_happy_scatterplot')
plt.show()
```

```
Saving figure money_happy_scatterplot
```

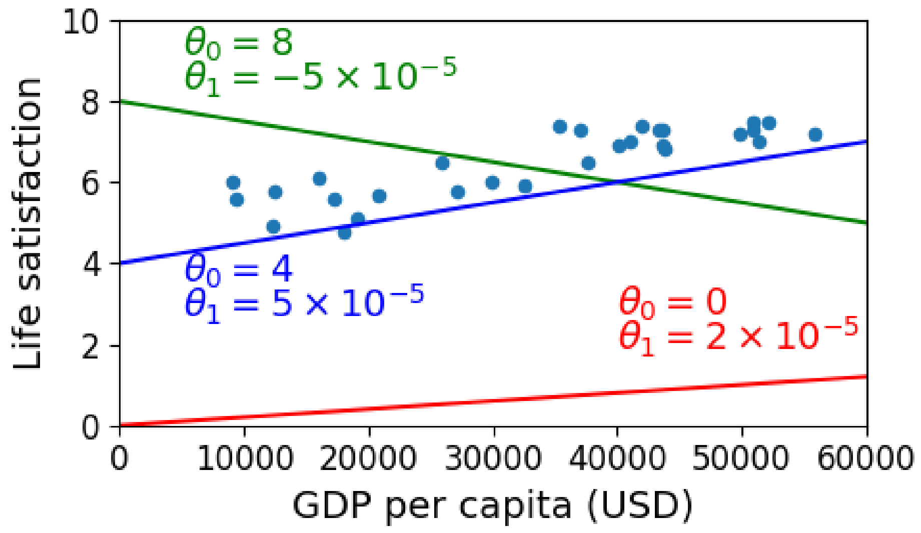


```
sample_data.to_csv(os.path.join("datasets", "lifesat", "lifesat.csv"))
sample_data.loc[list(position_text.keys())]
```

	GDP per capita	Life satisfaction
Country		
Hungary	12239.894	4.9
Korea	27195.197	5.8
France	37675.006	6.5
Australia	50961.865	7.3
United States	55805.204	7.2

```
import numpy as np

sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction', figsize=(5,3))
plt.xlabel("GDP per capita (USD)")
plt.axis([0, 60000, 0, 10])
X=np.linspace(0, 60000, 1000)
plt.plot(X, 2*X/100000, "r")
plt.text(40000, 2.7, r"$\theta_0 = 0$", fontsize=14, color="r")
plt.text(40000, 1.8, r"$\theta_1 = 2 \times 10^{-5}$", fontsize=14, color="r")
plt.plot(X, 8 - 5*X/100000, "g")
plt.text(5000, 9.1, r"$\theta_0 = 8$", fontsize=14, color="g")
plt.text(5000, 8.2, r"$\theta_1 = -5 \times 10^{-5}$", fontsize=14, color="g")
plt.plot(X, 4 + 5*X/100000, "b")
plt.text(5000, 3.5, r"$\theta_0 = 4$", fontsize=14, color="b")
plt.text(5000, 2.6, r"$\theta_1 = 5 \times 10^{-5}$", fontsize=14, color="b")
save_fig('tweaking_model_params_plot')
plt.show()
```

```
Saving figure tweaking_model_params_plot
```

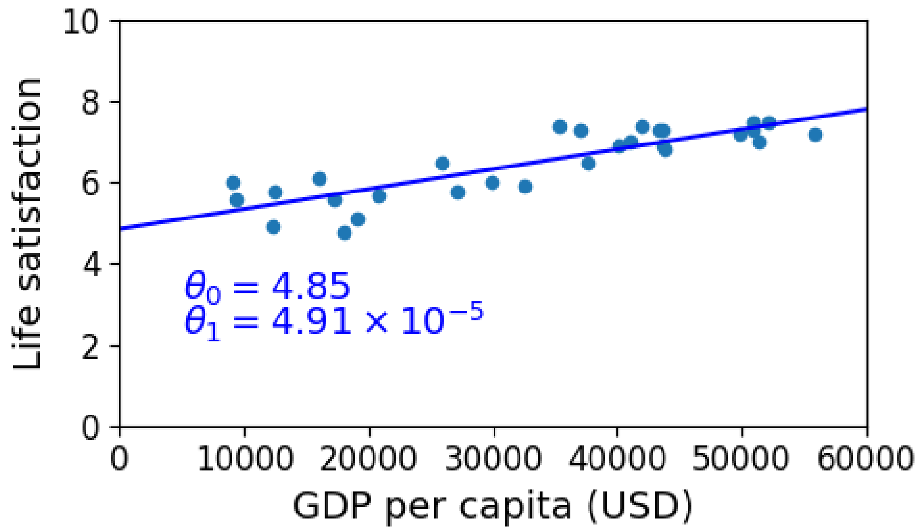


```
from sklearn import linear_model
lin1 = linear_model.LinearRegression()
Xsample = np.c_[sample_data["GDP per capita"]]
ysample = np.c_[sample_data["Life satisfaction"]]
lin1.fit(Xsample, ysample)
t0, t1 = lin1.intercept_[0], lin1.coef_[0][0]
t0, t1
```

```
(np.float64(4.853052800266436), np.float64(4.911544589158484e-05))
```

```
sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction', figsize=(5,3))
plt.xlabel("GDP per capita (USD)")
plt.axis([0, 60000, 0, 10])
X=np.linspace(0, 60000, 1000)
plt.plot(X, t0 + t1*X, "b")
plt.text(5000, 3.1, r"$\theta_0 = 4.85$", fontsize=14, color="b")
plt.text(5000, 2.2, r"$\theta_1 = 4.91 \times 10^{-5}$", fontsize=14, color="b")
save_fig('best_fit_model_plot')
plt.show()
```

```
Saving figure best_fit_model_plot
```



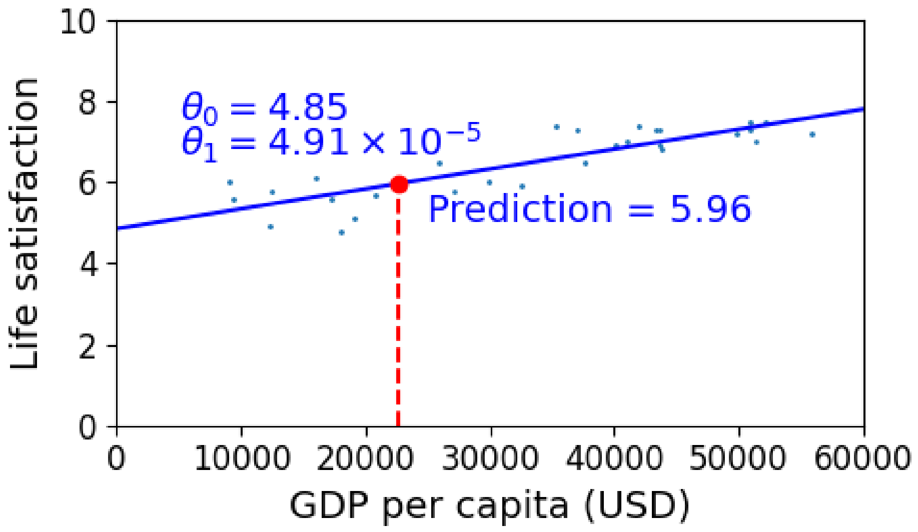
```
cyprus_gdp_per_capita = gdp_per_capita.loc["Cyprus"]["GDP per capita"]
print(cyprus_gdp_per_capita)
cyprus_predicted_life_satisfaction = lin1.predict([[cyprus_gdp_per_capita]])[0][0]
cyprus_predicted_life_satisfaction
```

```
22587.49
```

```
np.float64(5.96244744318815)
```

```
sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction', figsize=(5,3), s
plt.xlabel("GDP per capita (USD)")
X=np.linspace(0, 60000, 1000)
plt.plot(X, t0 + t1*X, "b")
plt.axis([0, 60000, 0, 10])
plt.text(5000, 7.5, r"$\theta_0 = 4.85$", fontsize=14, color="b")
plt.text(5000, 6.6, r"$\theta_1 = 4.91 \times 10^{-5}$", fontsize=14, color="b")
plt.plot([cyprus_gdp_per_capita, cyprus_gdp_per_capita], [0, cyprus_predicted_life_satisfact
plt.text(25000, 5.0, r"Prediction = 5.96", fontsize=14, color="b")
plt.plot(cyprus_gdp_per_capita, cyprus_predicted_life_satisfaction, "ro")
save_fig('cyprus_prediction_plot')
plt.show()
```

```
Saving figure cyprus_prediction_plot
```



```
sample_data[7:10]
```

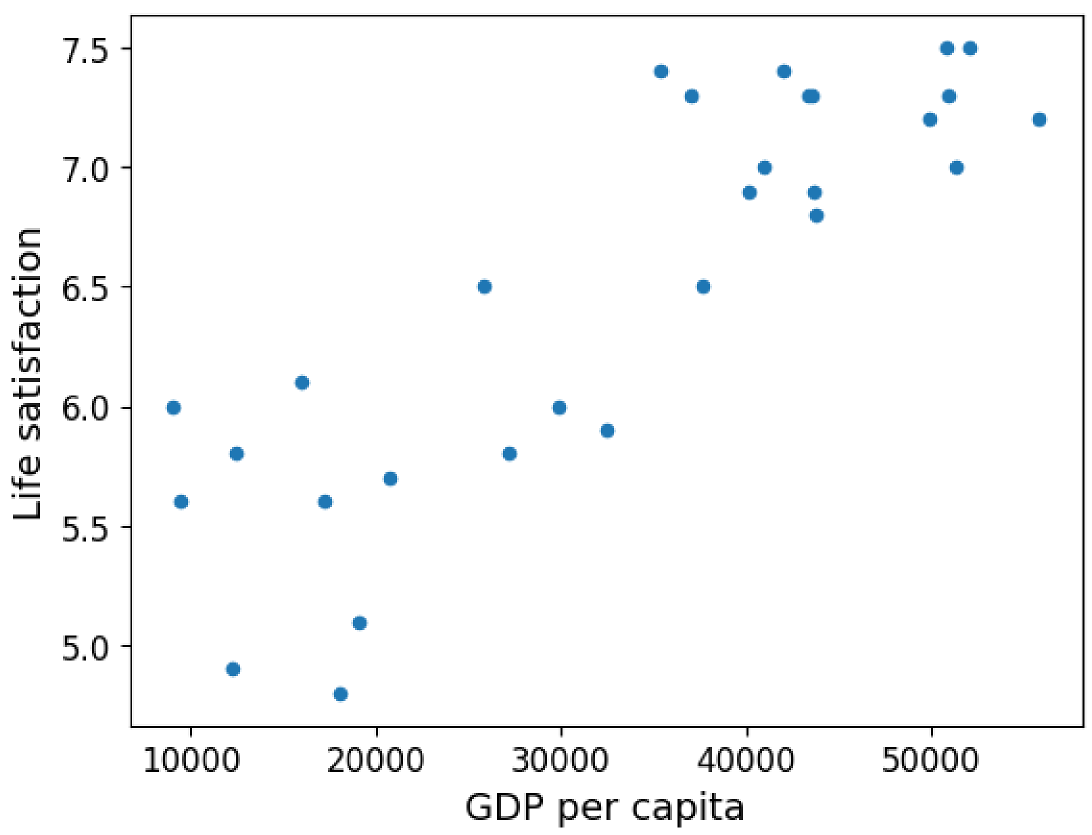
	GDP per capita	Life satisfaction
Country		
Portugal	19121.592	5.1
Slovenia	20732.482	5.7
Spain	25864.721	6.5

```
(5.1+5.7+6.5)/3
```

```
5.766666666666667
```

```
backup = oecd_bli, gdp_per_capita

def prepare_country_stats(oecd_bli, gdp_per_capita):
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
    gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
    gdp_per_capita.set_index("Country", inplace=True)
    full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                                  left_index=True, right_index=True)
    full_country_stats.sort_values(by="GDP per capita", inplace=True)
    remove_indices = [0, 1, 6, 8, 33, 34, 35]
    keep_indices = list(set(range(36)) - set(remove_indices))
    return full_country_stats[["GDP per capita", 'Life satisfaction']].iloc[keep_indices]
```

```
# Code example
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model

# Load the data
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv(datapath + "gdp_per_capita.csv",thousands=',',delimiter='\t',
                             encoding='latin1', na_values="n/a")

# Prepare the data
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]

# Visualize the data
country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction')
plt.show()

# Select a linear model
model = sklearn.linear_model.LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[22587]]  # Cyprus' GDP per capita
print(model.predict(X_new)) # outputs [[ 5.96242338]]
```



```
[[5.96242338]]
```

```
oecd_bli, gdp_per_capita = backup
```

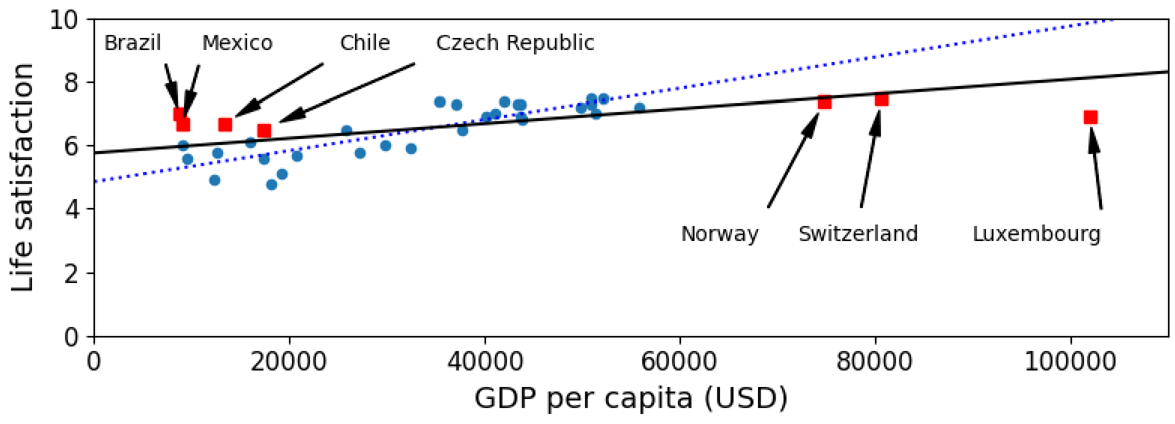
```
missing_data
```

	GDP per capita	Life satisfaction
Country		
Brazil	8669.998	7.0
Mexico	9009.280	6.7
Chile	13340.905	6.7
Czech Republic	17256.918	6.5
Norway	74822.106	7.4
Switzerland	80675.308	7.5
Luxembourg	101994.093	6.9

```
position_text2 = {
    "Brazil": (1000, 9.0),
    "Mexico": (11000, 9.0),
    "Chile": (25000, 9.0),
    "Czech Republic": (35000, 9.0),
    "Norway": (60000, 3),
    "Switzerland": (72000, 3.0),
    "Luxembourg": (90000, 3.0),
}
```

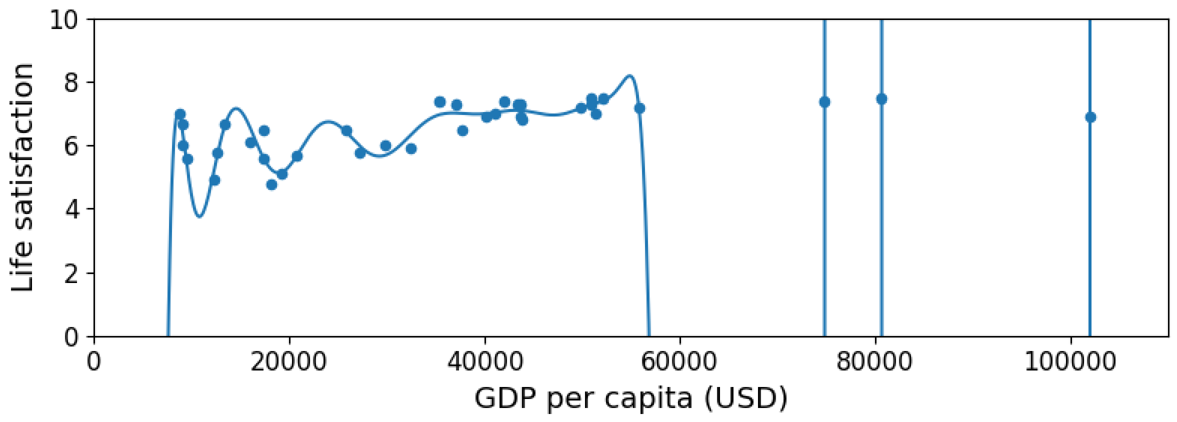
```
sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction', figsize=(8,3))
plt.axis([0, 110000, 0, 10])

for country, pos_text in position_text2.items():
    pos_data_x, pos_data_y = missing_data.loc[country]
    plt.annotate(country, xy=(pos_data_x, pos_data_y), xytext=pos_text,
            arrowprops=dict(facecolor='black', width=0.5, shrink=0.1, headwidth=5))
    plt.plot(pos_data_x, pos_data_y, "rs")

X=np.linspace(0, 110000, 1000)
plt.plot(X, t0 + t1*X, "b:")

lin_reg_full = linear_model.LinearRegression()
Xfull = np.c_[full_country_stats["GDP per capita"]]
yfull = np.c_[full_country_stats["Life satisfaction"]]
lin_reg_full.fit(Xfull, yfull)

t0full, t1full = lin_reg_full.intercept_[0], lin_reg_full.coef_[0][0]
X = np.linspace(0, 110000, 1000)
plt.plot(X, t0full + t1full * X, "k")
plt.xlabel("GDP per capita (USD)")

save_fig('representative_training_data_scatterplot')
plt.show()
```

```
Saving figure representative_training_data_scatterplot
```



```python
full_country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction', figsize=(
plt.axis([0, 110000, 0, 10])

from sklearn import preprocessing
from sklearn import pipeline

poly = preprocessing.PolynomialFeatures(degree=30, include_bias=False)
scaler = preprocessing.StandardScaler()
lin_reg2 = linear_model.LinearRegression()

pipeline_reg = pipeline.Pipeline([('poly', poly), ('scal', scaler), ('lin', lin_reg2)])
pipeline_reg.fit(Xfull, yfull)
curve = pipeline_reg.predict(X[:, np.newaxis])
plt.plot(X, curve)
plt.xlabel("GDP per capita (USD)")
save_fig('overfitting_model_plot')
plt.show()
```

```
Saving figure overfitting_model_plot
```



```
full_country_stats.loc[[c for c in full_country_stats.index if "W" in c.upper()]]["Life satis
```

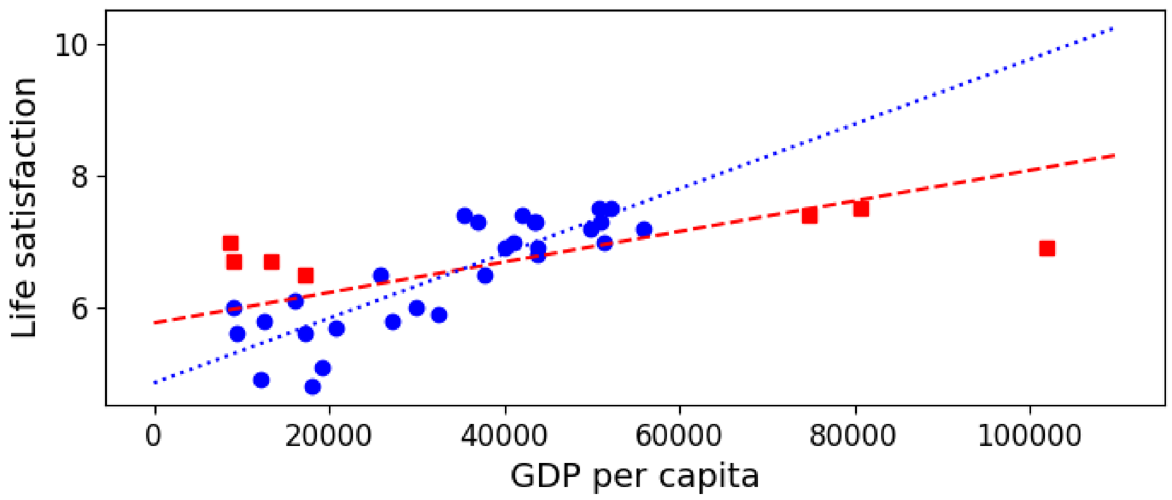
Country	Life satisfaction
New Zealand	7.3
Sweden	7.2
Norway	7.4
Switzerland	7.5

```
gdp_per_capita.loc[[c for c in gdp_per_capita.index if "W" in c.upper()]].head()
```

Country	Subject Descriptor	Units	Scale	Country/Series-speci
Botswana	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross
Kuwait	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross
Malawi	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross
New Zealand	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross
Norway	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross

```
plt.figure(figsize=(8,3))

plt.xlabel("GDP per capita")
plt.ylabel('Life satisfaction')

plt.plot(list(sample_data["GDP per capita"]), list(sample_data["Life satisfaction"]), "bo")
plt.plot(list(missing_data["GDP per capita"]), list(missing_data["Life satisfaction"]), "rs")

X = np.linspace(0, 110000, 1000)
plt.plot(X, t0full + t1full * X, "r--", label="Linear model on all data")
plt.plot(X, t0 + t1*X, "b:", label="Linear model on partial data")

ridge = linear_model.Ridge(alpha=10**9.5)
Xsample = np.c_[sample_data["GDP per capita"]]
ysample = np.c_[sample_data["Life satisfaction"]]
ridge.fit(Xsample, ysample)
t0ridge, t1ridge = ridge.intercept_[0], ridge.coef_[0][0]
plt.plot(X, t0ridge + t1ridge * X, "b", label="Regularized linear model on partial data")

plt.legend(loc="lower right")
plt.axis([0, 110000, 0, 10])
plt.xlabel("GDP per capita (USD)")
save_fig('ridge_model_plot')
plt.show()
```

```
IndexError: invalid index to scalar variable.
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipython-input-3720486063.py in <cell line: 0>()
     15 ysample = np.c_[sample_data["Life satisfaction"]]
     16 ridge.fit(Xsample, ysample)
---> 17 t0ridge, t1ridge = ridge.intercept_[0], ridge.coef_[0][0]
     18 plt.plot(X, t0ridge + t1ridge * X, "b", label="Regularized linear model on partial da
     19
IndexError: invalid index to scalar variable.
```


Life satisfaction
10
8
6
0
20000
40000
60000
80000
100000
GDP per capita

Part II

R

14 html

Part III

Bash

15 bash

```
```

Part IV

Javascript

16 title

16.1 g1

16.2 g2

16.3 g3

16.4 g4

16.5 g5

16.6 g6

Part V

D3

17 Graph1

17.1 graph1

17.2 graph2

17.3 graph3

17.4 graph4

17.5 graph5

17.6 graph6

18 Index-Graph-3D

This is a Quarto website.

To learn more about Quarto websites visit https://quarto.org/docs/websites.

18.1 graph1

18.2 graph2

18.3 graph3

18.4 graph4

18.5 graph5

18.6 graph6

19 title

19.1 g1

19.2 g2

19.3 g3

19.4 g4

19.5 g5

19.6 g6

Part VI

HTML

20 HTML 1

20.1 D1

20.2 D2

20.3 D3

20.4 D4

20.5 D5

20.6 D6

20.7 D7

20.8 D8

20.9 D9

20.10 D10

21 H2 HTML

21.1 H1

21.2 H2

21.3 H3

21.4 H4

21.5 H5

21.6 H6

21.7 H7

21.8 H8

21.9 H9

21.10 H10

22 Animacion and AWS

22.1 H1

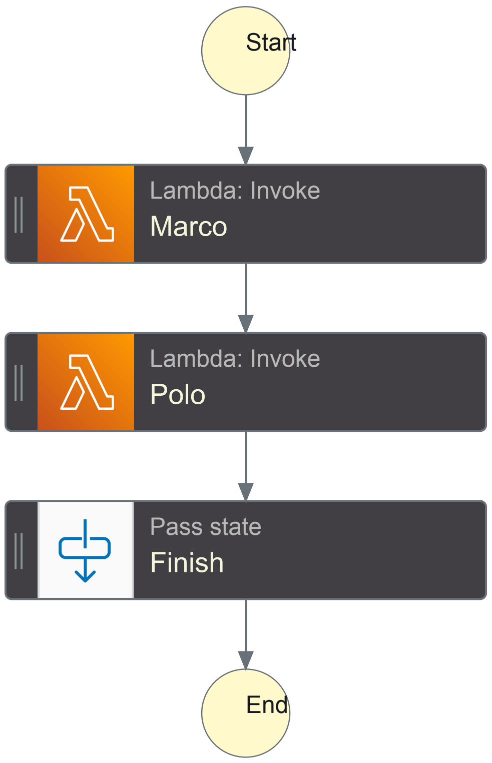
22.2 H2

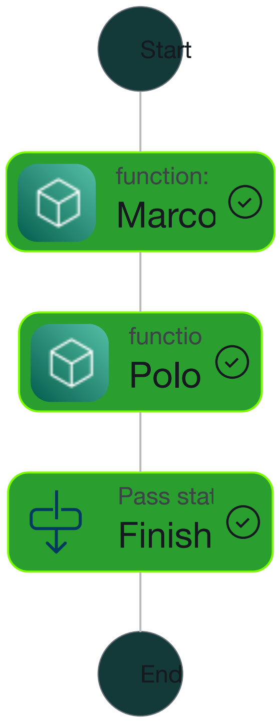
22.3 H3

22.4 H4

22.5 H5

22.6 H6




Start
function:
Marco
functio
Polo
Pass stat
Finish
End

23

24

25

26