```
import numpy as np
x = np.array([80, 65, 95, 95, 85, 75, 90, 65])  # Attendance
x2 = np.array([75, 70, 85, 100, 65, 55, 90, 80]) # Homework
y = np.array([1, 0, 1, 1, 0 , 0, 1, 1]) # Pass


# Separamos los conjuntos por prueba y entrenamiento
x_1 = x[:6]
x_1v = x[-2:]
x2_1 = x2[:6]
x2_v = x2[-2:]
y_t = y[:6]
y_v = y[-2:]
```

Algoritmo de regresión logística con columna Attendance

Para la inicialización de nuestros valores θ, nos basamos en la inicialización de Xavier-Glorot : https://www.tensorflow.org/api_docs/python/tf/keras/initializers/GlorotNormal

```
import tensorflow as tf
tf.random.set_seed(24)
initializer = tf.keras.initializers.GlorotNormal()
values = initializer(shape=(2, 1))
n = x_1.size
```

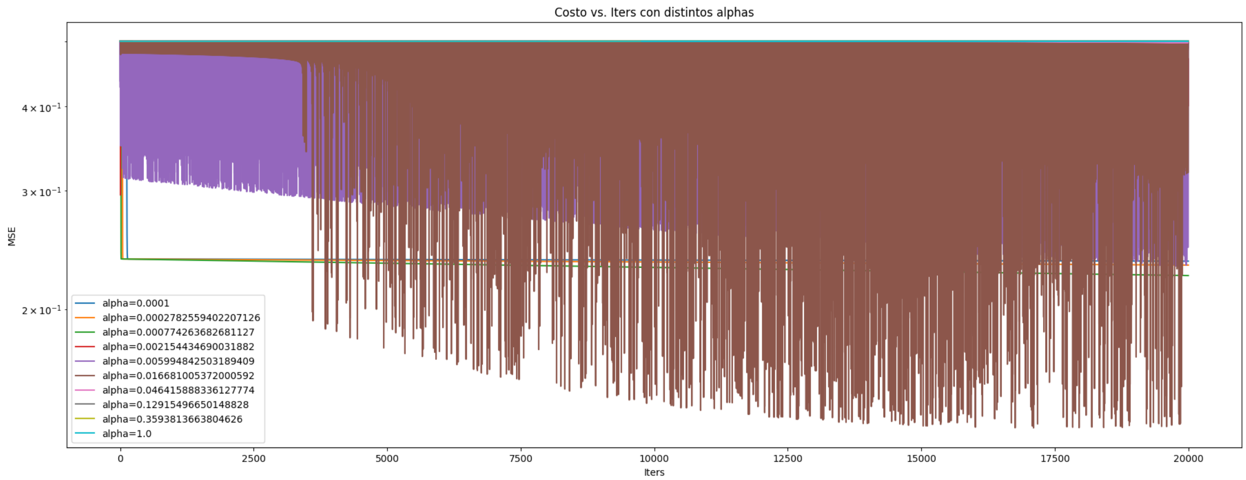
Crearemos un grid search para encontrar el α óptimo dentro de un rango.

```
learning_rates = np.logspace(-4, 0, 10)  # Se crean 10 valores 
logarítimicos entre 10e-4 a 1
resultados = [] # Aquí presentaremos nuestros resultados
for alpha in learning_rates: # Probaremos con cada alpha

  # Jalamos los pesos del inicializador
    theta_0 = (values[0].numpy())[0]
    theta_1 = (values[1].numpy())[0]
    costos = [] # Creamos la lista de los costos

  # Nuestro Algoritmo de Gradiente Descendente

    for i in range(20000):
    # Definimos nuestra función h_0, nuestra delta y nuestra delta * 
x1
        h_0 = 1 / (1 + np.exp(-(theta_0 + theta_1 * x_1)))
        delta = h_0 - y_t
        delta_x1 = delta * x_1

    # Actualizamos nuestros pesos
```

```
        theta_0 -= alpha * (1/n) * np.sum(delta)
        theta_1 -= alpha * (1/n) * np.sum(delta_x1)


        costo = np.mean((h_0 - y_t) ** 2)   # Calculamos costo con MSE
        costos.append(costo) # Agregamos el costo a la lista de los
costos

# Adjuntamos los valores relevantes encontrados con alpha
    resultados.append({
        "learning_rate": alpha,
        "costo_final": costos[-1],
        "theta_0": theta_0,
        "theta_1": theta_1,
        "costos": costos
    })
```

```
<ipython-input-79-f0f734002dca>:14: RuntimeWarning: overflow
encountered in exp
  h_0 = 1 / (1 + np.exp(-(theta_0 + theta_1 * x_1)))
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(22, 8)) # Graficamos los resultados

for result in resultados:
    plt.plot(result["costos"],
label=f"alpha={result['learning_rate']}")

plt.title('Costo vs. Iters con distintos alphas')
plt.xlabel('Iters')
plt.ylabel('MSE')
plt.legend()
plt.yscale('log')

plt.show()

for result in resultados: # Mostramos los resultados finales
    print(f"Learning Rate: {result['learning_rate']:.4f}")
    print(f"  Costo Final: {result['costo_final']:.6f}")
    print(f"  theta_0: {result['theta_0']:.6f}, theta_1:
{result['theta_1']:.6f}\n")
```




```
Learning Rate: 0.0001
  Costo Final: 0.235728
  theta_0: -0.567574, theta_1: 0.008943

Learning Rate: 0.0003
  Costo Final: 0.232604
  theta_0: -0.717516, theta_1: 0.010732

Learning Rate: 0.0008
  Costo Final: 0.224401
  theta_0: -1.123385, theta_1: 0.015577

Learning Rate: 0.0022
  Costo Final: 0.401319
  theta_0: -2.428242, theta_1: 0.081532

Learning Rate: 0.0060
  Costo Final: 0.499998
  theta_0: -5.827451, theta_1: 0.054428

Learning Rate: 0.0167
  Costo Final: 0.500000
  theta_0: -14.867152, theta_1: 0.744374

Learning Rate: 0.0464
  Costo Final: 0.500000
  theta_0: -42.668919, theta_1: 0.386173

Learning Rate: 0.1292
  Costo Final: 0.500000
  theta_0: -117.867876, theta_1: 1.924886

Learning Rate: 0.3594
  Costo Final: 0.500000
  theta_0: -327.120706, theta_1: 5.144160
```

```
Learning Rate: 1.0000
  Costo Final: 0.500000
  theta_0: -909.379149, theta_1: 14.101982
```

Observamos que el gráfico con $\alpha = 0.0167$ muestra los mejores resultados. En este caso, utilizaremos el $\alpha = 0.0167$.

```python
# Guardaremos todos los pesos
x_theta_0 = []
x_theta_1 = []
for result in resultados:
  x_theta_0.append(result['theta_0'])
  x_theta_1.append(result['theta_1'])

# Extraemos solo los de alpha = 0.0167
theta_0 = x_theta_0[5]
theta_1 = x_theta_1[5]
print(theta_0, theta_1)
```

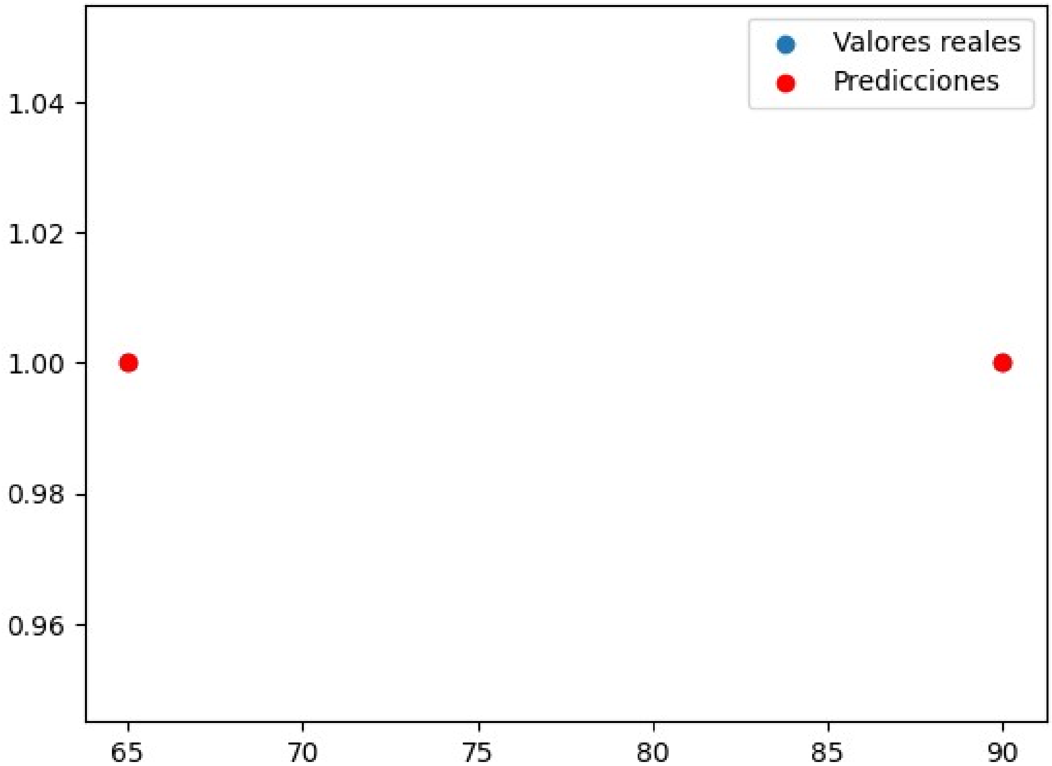
```
-14.86715247594826 0.7443741046620592
```

```python
y_pred = []
for i in range(0, len(x_1v)):
  y_pred.append(round(1/(1 + np.exp(-(theta_0 + theta_1*x_1v[i])))))

y_pred
```

```
[1, 1]
```

```python
import matplotlib.pyplot as plt
plt.scatter(x_1v, y_v)
plt.scatter(x_1v, y_pred, color='red')
plt.legend(['Valores reales', 'Predicciones'], loc = 'best')
```

```
<matplotlib.legend.Legend at 0x7c0790ff1c30>
```



Observamos que se predicen correctamente los dos valores.

```
# Scores
true_pos = 0
for i in range(0, len (y_pred)):
  if y_pred[i] == 1 and y_pred[i] == y_v[i]: true_pos +=1

true_neg = 0
for i in range(0, len (y_pred)):
  if y_pred[i] == 0 and y_pred[i] == y_v[i]: true_neg +=1

false_pos = 0
for i in range(0, len (y_pred)):
  if y_pred[i] == 1 and y_pred[i] != y_v[i]: false_pos +=1


false_neg = 0
for i in range(0, len (y_pred)):
  if y_pred[i] == 0 and y_pred[i] != y_v[i]: false_neg +=1

Accuracy = (true_pos + true_neg) / len(y_v)

if (true_pos + false_pos) == 0:
    Precision = 0
else:
    Precision = true_pos / (true_pos + false_pos)
```

```
if (true_pos + false_neg) == 0:
    Recall = 0
else:
    Recall = true_pos / (true_pos + false_neg)
if (Precision + Recall) == 0:
    F_1 = 0
else:
    F_1 = (2 * Precision * Recall) / (Precision + Recall)
print('Accuracy:', Accuracy, '\nPrecision:', Precision, '\nRecall:',
Recall, '\nF_1 Score:',F_1)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F_1 Score: 1.0
```

Algoritmo de regresión logística con columna Homework

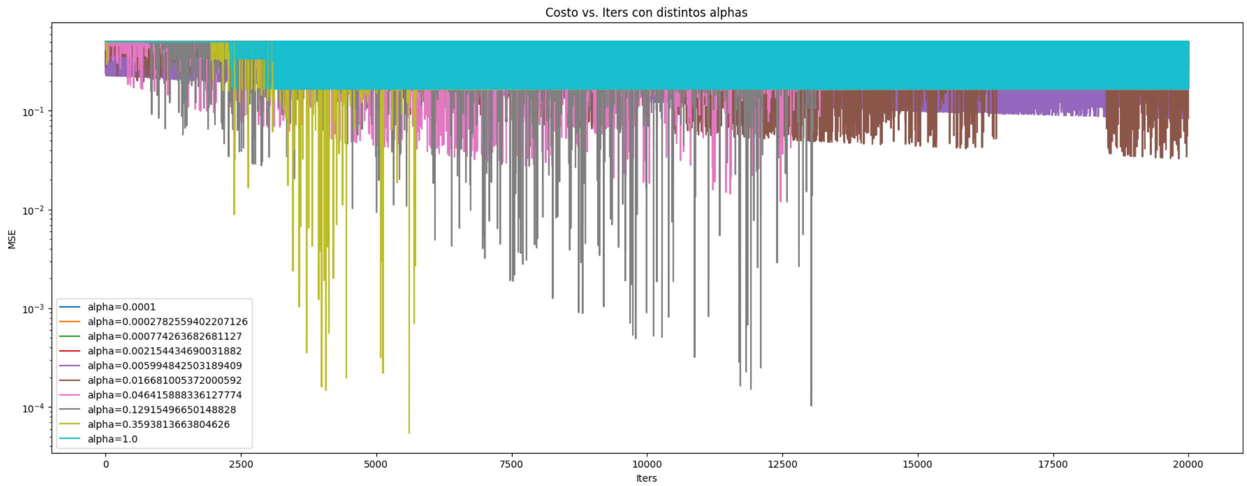
Se hará exactamente lo mismo que anteriormente, pero ahora con la columna Homework

```
learning_rates = np.logspace(-4, 0, 10)  # Se crean 10 valores
logarítimicos entre 10e-4 a 1
resultados = [] # Aquí presentaremos nuestros resultados
for alpha in learning_rates: # Probaremos con cada alpha

  # Jalamos los pesos del inicializador
    theta_0 = (values[0].numpy())[0]
    theta_1 = (values[1].numpy())[0]
    costos = [] # Creamos la lista de los costos

  # Nuestro Algoritmo de Gradiente Descendente

    for i in range(20000):
    # Definimos nuestra función h_0, nuestra delta y nuestra delta *
x1
        h_0 = 1 / (1 + np.exp(-(theta_0 + theta_1 * x2_1)))
        delta = h_0 - y_t
        delta_x2 = delta * x2_1

    # Actualizamos nuestros pesos
        theta_0 -= alpha * (1/n) * np.sum(delta)
        theta_1 -= alpha * (1/n) * np.sum(delta_x2)


        costo = np.mean((h_0 - y_t) ** 2)   # Calculamos costo con MSE
```

```python
        costos.append(costo) # Agregamos el costo a la lista de los costos

# Adjuntamos los valores relevantes encontrados con alpha
    resultados.append({
        "learning_rate": alpha,
        "costo_final": costos[-1],
        "theta_0": theta_0,
        "theta_1": theta_1,
        "costos": costos
    })
```

```
<ipython-input-87-ce27e66dd30f>:14: RuntimeWarning: overflow encountered in exp
  h_0 = 1 / (1 + np.exp(-(theta_0 + theta_1 * x2_1)))
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(22, 8)) # Graficamos los resultados

for result in resultados:
    plt.plot(result["costos"], label=f"alpha={result['learning_rate']}")

plt.title('Costo vs. Iters con distintos alphas')
plt.xlabel('Iters')
plt.ylabel('MSE')
plt.legend()
plt.yscale('log')

# Mostraremos este rango, pues es donde hubo realmente cambios significativos
plt.show()

for result in resultados: # Mostramos los resultados finales
    print(f"Learning Rate: {result['learning_rate']:.4f}")
    print(f"  Costo Final: {result['costo_final']:.6f}")
    print(f"  theta_0: {result['theta_0']:.6f}, theta_1: {result['theta_1']:.6f}\n")
```




```
Learning Rate: 0.0001
  Costo Final: 0.222361
  theta_0: -0.622314, theta_1: 0.012066

Learning Rate: 0.0003
  Costo Final: 0.214214
  theta_0: -0.866806, theta_1: 0.015229

Learning Rate: 0.0008
  Costo Final: 0.194199
  theta_0: -1.509403, theta_1: 0.023567

Learning Rate: 0.0022
  Costo Final: 0.369192
  theta_0: -3.554330, theta_1: 0.029384

Learning Rate: 0.0060
  Costo Final: 0.084029
  theta_0: -8.551582, theta_1: 0.128350

Learning Rate: 0.0167
  Costo Final: 0.500000
  theta_0: -23.738395, theta_1: 0.266830

Learning Rate: 0.0464
  Costo Final: 0.500000
  theta_0: -66.532692, theta_1: 1.147976

Learning Rate: 0.1292
  Costo Final: 0.500000
  theta_0: -172.643426, theta_1: 2.557258

Learning Rate: 0.3594
  Costo Final: 0.166599
  theta_0: -440.790174, theta_1: 2.225605
```

```
Learning Rate: 1.0000
  Costo Final: 0.500000
  theta_0: -1178.740020, theta_1: 48.634338
```

Observamos que el gráfico con $\alpha = 1$ muestra los mejores resultados. En este caso, utilizaremos el $\alpha = 1$.

```
# Guardaremos todos los pesos
x_theta_0 = []
x_theta_1 = []
for result in resultados:
  x_theta_0.append(result['theta_0'])
  x_theta_1.append(result['theta_1'])

# Extraemos solo los de alpha = 1
theta_0 = x_theta_0[-1]
theta_1 = x_theta_1[-1]
print(theta_0, theta_1)
```

```
-1178.7400198177247 48.634338242407
```

```
y_pred = []
for i in range(0, len(x_1v)):
  y_pred.append(round(1/(1 + np.exp(-(theta_0 + theta_1*x_1v[i])))))

y_pred
```

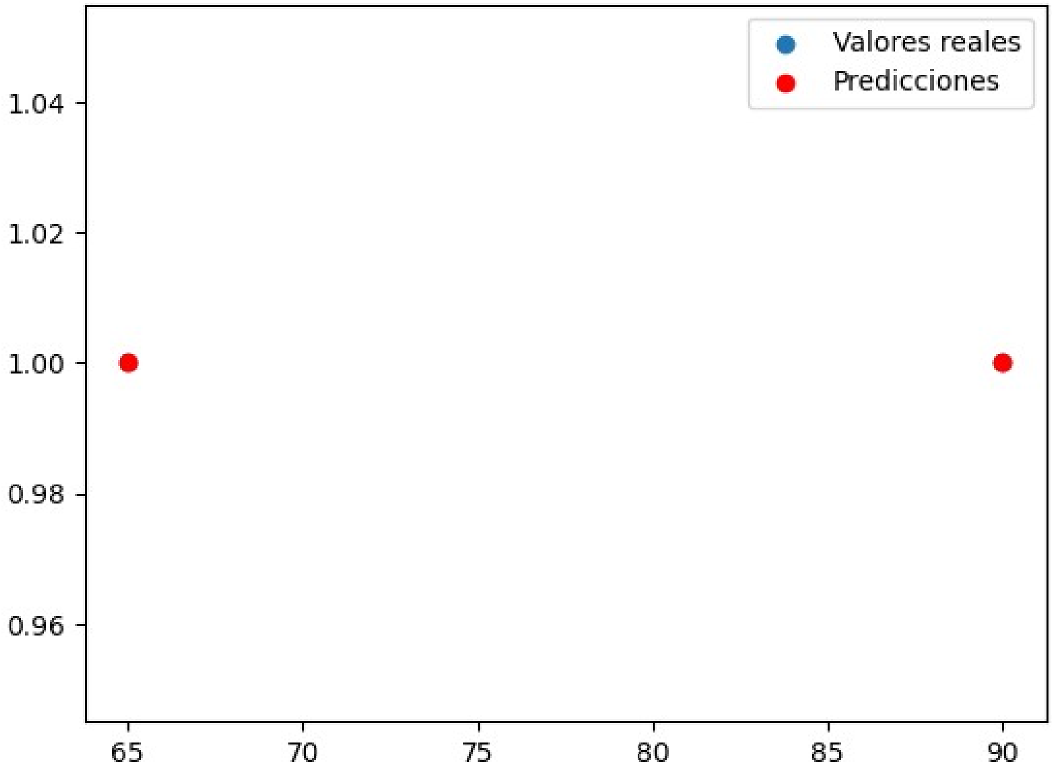
```
[1, 1]
```

```
len(y_v)
```

```
2
```

```
import matplotlib.pyplot as plt
plt.scatter(x_1v, y_v)
plt.scatter(x_1v, y_pred, color='red')
plt.legend(['Valores reales', 'Predicciones'], loc = 'best')
```

```
<matplotlib.legend.Legend at 0x7c079484dde0>
```



Observamos que se predicen correctamente los dos valores.

```
# Scores
true_pos = 0
for i in range(0, len (y_pred)):
  if y_pred[i] == 1 and y_pred[i] == y_v[i]: true_pos +=1

true_neg = 0
for i in range(0, len (y_pred)):
  if y_pred[i] == 0 and y_pred[i] == y_v[i]: true_neg +=1

false_pos = 0
for i in range(0, len (y_pred)):
  if y_pred[i] == 1 and y_pred[i] != y_v[i]: false_pos +=1


false_neg = 0
for i in range(0, len (y_pred)):
  if y_pred[i] == 0 and y_pred[i] != y_v[i]: false_neg +=1


Accuracy = (true_pos + true_neg )/ len(y_v)
Precision = (true_pos)/ (true_pos + false_pos)
Recall = (true_pos)/ (true_pos + false_neg)
F_1 = (2 * Precision * Recall) / (Precision + Recall)
```

```python
print('Accuracy:', Accuracy, '\nPrecision:', Precision, '\nRecall:',
Recall, '\nF_1 Score:',F_1)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F_1 Score: 1.0
```

```
!jupyter nbconvert Log_regression.ipynb --to html
```

```
[NbConvertApp] WARNING | pattern 'Log_regression.ipynb' matched no
files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE
RELEASES.

Options
=======
The options below are convenience aliases to configurable class-
options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an
error and include the error message in the cell output (the default
behaviour is to abort conversion). This flag is only relevant if '--
execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
```

```
--stdin
    read a single notebook file from stdin. Write the resulting
notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --
NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --
NbConvertApp.export_format=notebook --FilesWriter.build_directory= --
ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --
TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --
TemplateExporter.exclude_input=True --
TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is
found on the system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is
only useful for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be
sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN',
```

```
'ERROR', 'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path
for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme
distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be
sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a
```

```
time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each
notebook. To recover
                                  previous default behaviour
(outputting to the current
                                  working directory) use . as the flag
value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url
pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to
a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of
the
            current directory (from which the server is run).
            See the usage documentation

(https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-
slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex',
'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides',
'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb
```

```
            Both HTML and LaTeX support multiple output templates.
LaTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic',
'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab
mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post
serve

            Multiple notebooks can be given at the command line in a
couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file,
containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```