Table of Contents

config

```
[core]
remote = myremote
['remote "myremote"']
url = C:\Users\Asus\AppData\Local\Temp
```

.dvcignore

```
# Add patterns of files dvc should ignore, which could improve
# the performance. Learn more at
# https://dvc.org/doc/user-guide/dvcignore
```

LICENSE

Makefile

```
.PHONY: clean data lint requirements sync_data_to_s3
sync_data_from_s3

############################################################################
# GLOBALS
#

############################################################################
PROJECT_DIR := $(shell dirname $(realpath $(lastword
$(MAKEFILE_LIST))))
BUCKET = [OPTIONAL] your-bucket-for-syncing-data (do not include
's3://')
PROFILE = default
PROJECT_NAME = MlOps-Mini_Project
PYTHON_INTERPRETER = python3
ifeq (,$(shell which conda))
HAS_CONDA=False
else
HAS_CONDA=True
endif

############################################################################
# COMMANDS
#

############################################################################
## Install Python Dependencies
requirements: test_environment
    $(PYTHON_INTERPRETER) -m pip install -U pip setuptools wheel
    $(PYTHON_INTERPRETER) -m pip install -r requirements.txt
## Make Dataset
data: requirements
    $(PYTHON_INTERPRETER) src/data/make_dataset.py data/raw
data/processed
## Delete all compiled Python files
clean:
    find . -type f -name "*.py[co]" -delete
    find . -type d -name "__pycache__" -delete
## Lint using flake8
lint:
    flake8 src
## Upload Data to S3
sync_data_to_s3:
ifeq (default,$(PROFILE))
    aws s3 sync data/ s3://$(BUCKET)/data/
else
    aws s3 sync data/ s3://$(BUCKET)/data/ --profile $(PROFILE)
endif
## Download Data from S3
```

```
sync_data_from_s3:
ifeq (default,$(PROFILE))
    aws s3 sync s3://$(BUCKET)/data/ data/
else
    aws s3 sync s3://$(BUCKET)/data/ data/ --profile $(PROFILE)
endif
## Set up python interpreter environment
create_environment:
ifeq (True,$(HAS_CONDA))
        @echo ">>> Detected conda, creating conda environment."
ifeq (3,$(findstring 3,$(PYTHON_INTERPRETER)))
    conda create --name $(PROJECT_NAME) python=3
else
    conda create --name $(PROJECT_NAME) python=2.7
endif
        @echo ">>> New conda env created. Activate with:\nsource activate
$(PROJECT_NAME)"
else
    $(PYTHON_INTERPRETER) -m pip install -q virtualenv virtualenvwrapper
    @echo ">>> Installing virtualenvwrapper if not already
installed.\nMake sure the following lines are in shell startup
file\n\
    export WORKON_HOME=$$HOME/.virtualenvs\nexport
PROJECT_HOME=$$HOME/Devel\nsource
/usr/local/bin/virtualenvwrapper.sh\n"
    @bash -c "source `which virtualenvwrapper.sh`;mkvirtualenv
$(PROJECT_NAME) --python=$(PYTHON_INTERPRETER)"
    @echo ">>> New virtualenv created. Activate with:\nworkon
$(PROJECT_NAME)"
endif
## Test python environment is setup correctly
test_environment:
    $(PYTHON_INTERPRETER) test_environment.py

#################################################################################
# PROJECT RULES
#

#################################################################################

#################################################################################
# Self Documenting Commands
#

#################################################################################
.DEFAULT_GOAL := help
# Inspired by
<http://marmelab.com/blog/2016/02/29/auto-documented-makefile.html>
# sed script explained:
# /^##/:
#     * save line in hold space
#     * purge line
#     * Loop:
#         * append newline + line to hold space
#         * go to next line
#         * if line starts with doc comment, strip comment character off
and loop
#     * remove target prerequisites
#     * append hold space (+ newline) to line
#     * replace newline plus comments by `---`
#     * print line
# Separate expressions are necessary because labels cannot be
delimited by
# semicolon; see <http://stackoverflow.com/a/11799865/1968>
.PHONY: help
help:
	@echo "$$(tput bold)Available rules:$$(tput sgr0)"
	@echo
	@sed -n -e "/^## / { \
		h; \
		s/.*//; \
		:doc" \
		-e "H; \
		n; \
		s/^## //; \
		t doc" \
		-e "s/:.*//; \
		G; \
		s/\\n## /---/; \
		s/\\n/ /g; \
		p; \
	}" ${MAKEFILE_LIST} \
	| LC_ALL='C' sort --ignore-case \
	| awk -F '---' \
		-v ncol=$$(tput cols) \
		-v indent=19 \
		-v col_on="$$(tput setaf 6)" \
		-v col_off="$$(tput sgr0)" \
	'{ \
		printf "%s%*s%s ", col_on, -indent, $$1, col_off; \
		n = split($$2, words, " "); \
		line_length = ncol - indent; \
		for (i = 1; i <= n; i++) { \
			line_length -= length(words[i]) + 1; \
			if (line_length <= 0) { \
				line_length = ncol - indent - length(words[i]) - 1; \
				printf "\n%*s ", -indent, " "; \
			} \
			printf "%s ", words[i]; \
		} \
		printf "\n"; \
	}' \
    | more $(shell test $(shell uname) = Darwin && echo '--no-init
--raw-control-chars')
```

README.md

```
MLOps Mini Project
==============================
A short description of the project.
Project Organization
------------
LICENSE
Makefile           <- Makefile with commands like `make data` or
`make train`
README.md          <- The top-level README for developers using this
project.
data
external       <- Data from third party sources.
interim        <- Intermediate data that has been transformed.
processed      <- The final, canonical data sets for modeling.
raw            <- The original, immutable data dump.
docs               <- A default Sphinx project; see sphinx-doc.org
for details
models             <- Trained and serialized models, model
predictions, or model summaries
notebooks          <- Jupyter notebooks. Naming convention is a
number (for ordering),
the creator's initials, and a short `-` delimited description, e.g.
`1.0-jqp-initial-data-exploration`.
references         <- Data dictionaries, manuals, and all other
explanatory materials.
reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
figures        <- Generated graphics and figures to be used in
reporting
requirements.txt   <- The requirements file for reproducing the
analysis environment, e.g.
generated with `pip freeze > requirements.txt`
setup.py           <- makes project pip installable (pip install -e
.) so src can be imported
src                <- Source code for use in this project.
__init__.py    <- Makes src a Python module
data           <- Scripts to download or generate data
make_dataset.py
features       <- Scripts to turn raw data into features for modeling
build_features.py
models         <- Scripts to train models and then use trained models
to make
predictions
predict_model.py
train_model.py
visualization  <- Scripts to create exploratory and results oriented
visualizations
visualize.py
tox.ini            <- tox file with settings for running tox; see
tox.readthedocs.io
```

```
--------
<p><small>Project based on the <a target="_blank"
href="https://drivendata.github.io/cookiecutter-data-science/">cookiecutter
data science project template</a>.
#cookiecutterdatascience</small></p>
```

Makefile

```
# Makefile for Sphinx documentation
#
# You can set these variables from the command line.
SPHINXOPTS    =
SPHINXBUILD   = sphinx-build
PAPER         =
BUILDDIR      = _build
# Internal variables.
PAPEROPT_a4     = -D latex_paper_size=a4
PAPEROPT_letter = -D latex_paper_size=letter
ALLSPHINXOPTS   = -d $(BUILDDIR)/doctrees $(PAPEROPT_$(PAPER))
$(SPHINXOPTS) .
# the i18n builder cannot share the environment and doctrees with the
others
I18NSPHINXOPTS  = $(PAPEROPT_$(PAPER)) $(SPHINXOPTS) .
.PHONY: help clean html dirhtml singlehtml pickle json htmlhelp
qthelp devhelp epub latex latexpdf text man changes linkcheck doctest
gettext
help:
    @echo "Please use \`make <target>' where <target> is one of"
    @echo "  html       to make standalone HTML files"
    @echo "  dirhtml    to make HTML files named index.html in
directories"
    @echo "  singlehtml to make a single large HTML file"
    @echo "  pickle     to make pickle files"
    @echo "  json       to make JSON files"
    @echo "  htmlhelp   to make HTML files and a HTML help project"
    @echo "  qthelp     to make HTML files and a qthelp project"
    @echo "  devhelp    to make HTML files and a Devhelp project"
    @echo "  epub       to make an epub"
    @echo "  latex      to make LaTeX files, you can set PAPER=a4 or
PAPER=letter"
    @echo "  latexpdf   to make LaTeX files and run them through
pdflatex"
    @echo "  text       to make text files"
    @echo "  man        to make manual pages"
    @echo "  texinfo    to make Texinfo files"
    @echo "  info       to make Texinfo files and run them through
makeinfo"
    @echo "  gettext    to make PO message catalogs"
    @echo "  changes    to make an overview of all
changed/added/deprecated items"
    @echo "  linkcheck  to check all external links for integrity"
    @echo "  doctest    to run all doctests embedded in the
documentation (if enabled)"
clean:
    -rm -rf $(BUILDDIR)/*
html:
    $(SPHINXBUILD) -b html $(ALLSPHINXOPTS) $(BUILDDIR)/html
```

```
	@echo
	@echo "Build finished. The HTML pages are in $(BUILDDIR)/html."
dirhtml:
	$(SPHINXBUILD) -b dirhtml $(ALLSPHINXOPTS) $(BUILDDIR)/dirhtml
	@echo
	@echo "Build finished. The HTML pages are in $(BUILDDIR)/dirhtml."
singlehtml:
	$(SPHINXBUILD) -b singlehtml $(ALLSPHINXOPTS) $(BUILDDIR)/singlehtml
	@echo
	@echo "Build finished. The HTML page is in $(BUILDDIR)/singlehtml."
pickle:
	$(SPHINXBUILD) -b pickle $(ALLSPHINXOPTS) $(BUILDDIR)/pickle
	@echo
	@echo "Build finished; now you can process the pickle files."
json:
	$(SPHINXBUILD) -b json $(ALLSPHINXOPTS) $(BUILDDIR)/json
	@echo
	@echo "Build finished; now you can process the JSON files."
htmlhelp:
	$(SPHINXBUILD) -b htmlhelp $(ALLSPHINXOPTS) $(BUILDDIR)/htmlhelp
	@echo
	@echo "Build finished; now you can run HTML Help Workshop with the"
\
	      ".hhp project file in $(BUILDDIR)/htmlhelp."
qthelp:
	$(SPHINXBUILD) -b qthelp $(ALLSPHINXOPTS) $(BUILDDIR)/qthelp
	@echo
	@echo "Build finished; now you can run "qcollectiongenerator" with
the" \
	      ".qhcp project file in $(BUILDDIR)/qthelp, like this:"
	@echo "# qcollectiongenerator
$(BUILDDIR)/qthelp/MlOps-Mini_Project.qhcp"
	@echo "To view the help file:"
	@echo "# assistant -collectionFile
$(BUILDDIR)/qthelp/MlOps-Mini_Project.qhc"
devhelp:
	$(SPHINXBUILD) -b devhelp $(ALLSPHINXOPTS) $(BUILDDIR)/devhelp
	@echo
	@echo "Build finished."
	@echo "To view the help file:"
	@echo "# mkdir -p $$HOME/.local/share/devhelp/MlOps-Mini_Project"
	@echo "# ln -s $(BUILDDIR)/devhelp
$$HOME/.local/share/devhelp/MlOps-Mini_Project"
	@echo "# devhelp"
epub:
	$(SPHINXBUILD) -b epub $(ALLSPHINXOPTS) $(BUILDDIR)/epub
	@echo
	@echo "Build finished. The epub file is in $(BUILDDIR)/epub."
latex:
	$(SPHINXBUILD) -b latex $(ALLSPHINXOPTS) $(BUILDDIR)/latex
	@echo
```

```
    @echo "Build finished; the LaTeX files are in $(BUILDDIR)/latex."
    @echo "Run \`make' in that directory to run these through
(pdf)latex" \
          "(use \`make latexpdf' here to do that automatically)."
latexpdf:
    $(SPHINXBUILD) -b latex $(ALLSPHINXOPTS) $(BUILDDIR)/latex
    @echo "Running LaTeX files through pdflatex..."
    $(MAKE) -C $(BUILDDIR)/latex all-pdf
    @echo "pdflatex finished; the PDF files are in $(BUILDDIR)/latex."
text:
    $(SPHINXBUILD) -b text $(ALLSPHINXOPTS) $(BUILDDIR)/text
    @echo
    @echo "Build finished. The text files are in $(BUILDDIR)/text."
man:
    $(SPHINXBUILD) -b man $(ALLSPHINXOPTS) $(BUILDDIR)/man
    @echo
    @echo "Build finished. The manual pages are in $(BUILDDIR)/man."
texinfo:
    $(SPHINXBUILD) -b texinfo $(ALLSPHINXOPTS) $(BUILDDIR)/texinfo
    @echo
    @echo "Build finished. The Texinfo files are in
$(BUILDDIR)/texinfo."
    @echo "Run \`make' in that directory to run these through makeinfo"
\
          "(use \`make info' here to do that automatically)."
info:
    $(SPHINXBUILD) -b texinfo $(ALLSPHINXOPTS) $(BUILDDIR)/texinfo
    @echo "Running Texinfo files through makeinfo..."
    make -C $(BUILDDIR)/texinfo info
    @echo "makeinfo finished; the Info files are in
$(BUILDDIR)/texinfo."
gettext:
    $(SPHINXBUILD) -b gettext $(I18NSPHINXOPTS) $(BUILDDIR)/locale
    @echo
    @echo "Build finished. The message catalogs are in
$(BUILDDIR)/locale."
changes:
    $(SPHINXBUILD) -b changes $(ALLSPHINXOPTS) $(BUILDDIR)/changes
    @echo
    @echo "The overview file is in $(BUILDDIR)/changes."
linkcheck:
    $(SPHINXBUILD) -b linkcheck $(ALLSPHINXOPTS) $(BUILDDIR)/linkcheck
    @echo
    @echo "Link check complete; look for any errors in the above output
" \
          "or in $(BUILDDIR)/linkcheck/output.txt."
doctest:
    $(SPHINXBUILD) -b doctest $(ALLSPHINXOPTS) $(BUILDDIR)/doctest
    @echo "Testing of doctests in the sources finished, look at the " \
          "results in $(BUILDDIR)/doctest/output.txt."
```

commands.rst

```
Commands
========
The Makefile contains the central entry points for common tasks
related to this project.
Syncing data to S3
^^^^^^^^^^^^^^^^^^
* `make sync_data_to_s3` will use `aws s3 sync` to recursively sync
files in `data/` up to `s3://[OPTIONAL] your-bucket-for-syncing-data
(do not include 's3://')/data/`.
* `make sync_data_from_s3` will use `aws s3 sync` to recursively sync
files from `s3://[OPTIONAL] your-bucket-for-syncing-data (do not
include 's3://')/data/` to `data/`.
```

conf.py

```
# -*- coding: utf-8 -*-
#
# MLOps Mini Project documentation build configuration file, created
by
# sphinx-quickstart.
#
# This file is execfile()d with the current directory set to its
containing dir.
#
# Note that not all possible configuration values are present in this
# autogenerated file.
#
# All configuration values have a default; values that are commented
out
# serve to show the default.
import os
import sys
# If extensions (or modules to document with autodoc) are in another
directory,
# add these directories to sys.path here. If the directory is
relative to the
# documentation root, use os.path.abspath to make it absolute, like
shown here.
# sys.path.insert(0, os.path.abspath('.'))
# -- General configuration
-----------------------------------------------------
# If your documentation needs a minimal Sphinx version, state it
here.
# needs_sphinx = '1.0'
# Add any Sphinx extension module names here, as strings. They can be
extensions
# coming with Sphinx (named 'sphinx.ext.*') or your custom ones.
extensions = []
# Add any paths that contain templates here, relative to this
directory.
templates_path = ['_templates']
# The suffix of source filenames.
source_suffix = '.rst'
# The encoding of source files.
# source_encoding = 'utf-8-sig'
# The master toctree document.
master_doc = 'index'
# General information about the project.
project = u'MLOps Mini Project'
# The version info for the project you're documenting, acts as
replacement for
# |version| and |release|, also used in various other places
throughout the
# built documents.
```

```
#
# The short X.Y version.
version = '0.1'
# The full version, including alpha/beta/rc tags.
release = '0.1'
# The language for content autogenerated by Sphinx. Refer to
documentation
# for a list of supported languages.
# language = None
# There are two options for replacing |today|: either, you set today
to some
# non-false value, then it is used:
# today = ''
# Else, today_fmt is used as the format for a strftime call.
# today_fmt = '%B %d, %Y'
# List of patterns, relative to source directory, that match files
and
# directories to ignore when looking for source files.
exclude_patterns = ['_build']
# The reST default role (used for this markup: `text`) to use for all
documents.
# default_role = None
# If true, '()' will be appended to :func: etc. cross-reference text.
# add_function_parentheses = True
# If true, the current module name will be prepended to all
description
# unit titles (such as .. function::).
# add_module_names = True
# If true, sectionauthor and moduleauthor directives will be shown in
the
# output. They are ignored by default.
# show_authors = False
# The name of the Pygments (syntax highlighting) style to use.
pygments_style = 'sphinx'
# A list of ignored prefixes for module index sorting.
# modindex_common_prefix = []
# -- Options for HTML output
---------------------------------------------------
# The theme to use for HTML and HTML Help pages.  See the
documentation for
# a list of builtin themes.
html_theme = 'default'
# Theme options are theme-specific and customize the look and feel of
a theme
# further.  For a list of options available for each theme, see the
# documentation.
# html_theme_options = {}
# Add any paths that contain custom themes here, relative to this
directory.
# html_theme_path = []
# The name for this set of Sphinx documents.  If None, it defaults to
# "<project> v<release> documentation".
# html_title = None
# A shorter title for the navigation bar.  Default is the same as
html_title.
# html_short_title = None
# The name of an image file (relative to this directory) to place at
the top
# of the sidebar.
# html_logo = None
# The name of an image file (within the static path) to use as
favicon of the
# docs.  This file should be a Windows icon file (.ico) being 16x16
or 32x32
# pixels large.
# html_favicon = None
# Add any paths that contain custom static files (such as style
sheets) here,
# relative to this directory. They are copied after the builtin
static files,
# so a file named "default.css" will overwrite the builtin
"default.css".
html_static_path = ['_static']
# If not '', a 'Last updated on:' timestamp is inserted at every page
bottom,
# using the given strftime format.
# html_last_updated_fmt = '%b %d, %Y'
# If true, SmartyPants will be used to convert quotes and dashes to
# typographically correct entities.
# html_use_smartypants = True
# Custom sidebar templates, maps document names to template names.
# html_sidebars = {}
# Additional templates that should be rendered to pages, maps page
names to
# template names.
# html_additional_pages = {}
# If false, no module index is generated.
# html_domain_indices = True
# If false, no index is generated.
# html_use_index = True
# If true, the index is split into individual pages for each letter.
# html_split_index = False
# If true, links to the reST sources are added to the pages.
# html_show_sourcelink = True
# If true, "Created using Sphinx" is shown in the HTML footer.
Default is True.
# html_show_sphinx = True
# If true, "(C) Copyright ..." is shown in the HTML footer. Default
is True.
# html_show_copyright = True
# If true, an OpenSearch description file will be output, and all
pages will
```

```
# contain a <link> tag referring to it.  The value of this option
must be the
# base URL from which the finished HTML is served.
# html_use_opensearch = ''
# This is the file name suffix for HTML files (e.g. ".xhtml").
# html_file_suffix = None
# Output file base name for HTML help builder.
htmlhelp_basename = 'MlOps-Mini_Projectdoc'
# -- Options for LaTeX output
--------------------------------------------------
latex_elements = {
# The paper size ('letterpaper' or 'a4paper').
# 'papersize': 'letterpaper',
# The font size ('10pt', '11pt' or '12pt').
# 'pointsize': '10pt',
# Additional stuff for the LaTeX preamble.
# 'preamble': '',
}
# Grouping the document tree into LaTeX files. List of tuples
# (source start file, target name, title, author, documentclass
[howto/manual]).
latex_documents = [
('index',
'MlOps-Mini_Project.tex',
u'MLOps Mini Project Documentation',
u"Joel", 'manual'),
]
# The name of an image file (relative to this directory) to place at
the top of
# the title page.
# latex_logo = None
# For "manual" documents, if this is true, then toplevel headings are
parts,
# not chapters.
# latex_use_parts = False
# If true, show page references after internal links.
# latex_show_pagerefs = False
# If true, show URL addresses after external links.
# latex_show_urls = False
# Documents to append as an appendix to all manuals.
# latex_appendices = []
# If false, no module index is generated.
# latex_domain_indices = True
# -- Options for manual page output
--------------------------------------------
# One entry per manual page. List of tuples
# (source start file, name, description, authors, manual section).
man_pages = [
('index', 'MlOps-Mini_Project', u'MLOps Mini Project Documentation',
[u"Joel"], 1)
]
```

```
# If true, show URL addresses after external links.
# man_show_urls = False
# -- Options for Texinfo output
------------------------------------------------
# Grouping the document tree into Texinfo files. List of tuples
# (source start file, target name, title, author,
#  dir menu entry, description, category)
texinfo_documents = [
('index', 'MlOps-Mini_Project', u'MLOps Mini Project Documentation',
u"Joel", 'MLOps Mini Project',
'A short description of the project.', 'Miscellaneous'),
]
# Documents to append as an appendix to all manuals.
# texinfo_appendices = []
# If false, no module index is generated.
# texinfo_domain_indices = True
# How to display URL addresses: 'footnote', 'no', or 'inline'.
# texinfo_show_urls = 'footnote'
```

getting-started.rst

```
Getting started
===============
This is where you describe how to get set up on a clean install,
including the
commands necessary to get the raw data (using the `sync_data_from_s3`
command,
for example), and then how to make the cleaned, final data sets.
```

index.rst

```
.. MLOps Mini Project documentation master file, created by
sphinx-quickstart.
You can adapt this file completely to your liking, but it should at
least
contain the root `toctree` directive.
MLOps Mini Project documentation!
==============================================
Contents:
.. toctree::
:maxdepth: 2
getting-started
commands
Indices and tables
==================
* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

make.bat

```
@ECHO OFF
REM Command file for Sphinx documentation
if "%SPHINXBUILD%" == "" (
    set SPHINXBUILD=sphinx-build
)
set BUILDDIR=_build
set ALLSPHINXOPTS=-d %BUILDDIR%/doctrees %SPHINXOPTS% .
set I18NSPHINXOPTS=%SPHINXOPTS% .
if NOT "%PAPER%" == "" (
    set ALLSPHINXOPTS=-D latex_paper_size=%PAPER% %ALLSPHINXOPTS%
    set I18NSPHINXOPTS=-D latex_paper_size=%PAPER% %I18NSPHINXOPTS%
)
if "%1" == "" goto help
if "%1" == "help" (
    :help
    echo.Please use `make ^<target^>` where ^<target^> is one of
    echo.  html       to make standalone HTML files
    echo.  dirhtml    to make HTML files named index.html in directories
    echo.  singlehtml to make a single large HTML file
    echo.  pickle     to make pickle files
    echo.  json       to make JSON files
    echo.  htmlhelp   to make HTML files and a HTML help project
    echo.  qthelp     to make HTML files and a qthelp project
    echo.  devhelp    to make HTML files and a Devhelp project
    echo.  epub       to make an epub
    echo.  latex      to make LaTeX files, you can set PAPER=a4 or
PAPER=letter
    echo.  text       to make text files
    echo.  man        to make manual pages
    echo.  texinfo    to make Texinfo files
    echo.  gettext    to make PO message catalogs
    echo.  changes    to make an overview over all
changed/added/deprecated items
    echo.  linkcheck  to check all external links for integrity
    echo.  doctest    to run all doctests embedded in the documentation
if enabled
    goto end
)
if "%1" == "clean" (
    for /d %%i in (%BUILDDIR%\*) do rmdir /q /s %%i
    del /q /s %BUILDDIR%\*
    goto end
)
if "%1" == "html" (
    %SPHINXBUILD% -b html %ALLSPHINXOPTS% %BUILDDIR%/html
    if errorlevel 1 exit /b 1
    echo.
    echo.Build finished. The HTML pages are in %BUILDDIR%/html.
    goto end
)
if "%1" == "dirhtml" (
	%SPHINXBUILD% -b dirhtml %ALLSPHINXOPTS% %BUILDDIR%/dirhtml
	if errorlevel 1 exit /b 1
	echo.
	echo.Build finished. The HTML pages are in %BUILDDIR%/dirhtml.
	goto end
)
if "%1" == "singlehtml" (
	%SPHINXBUILD% -b singlehtml %ALLSPHINXOPTS% %BUILDDIR%/singlehtml
	if errorlevel 1 exit /b 1
	echo.
	echo.Build finished. The HTML pages are in %BUILDDIR%/singlehtml.
	goto end
)
if "%1" == "pickle" (
	%SPHINXBUILD% -b pickle %ALLSPHINXOPTS% %BUILDDIR%/pickle
	if errorlevel 1 exit /b 1
	echo.
	echo.Build finished; now you can process the pickle files.
	goto end
)
if "%1" == "json" (
	%SPHINXBUILD% -b json %ALLSPHINXOPTS% %BUILDDIR%/json
	if errorlevel 1 exit /b 1
	echo.
	echo.Build finished; now you can process the JSON files.
	goto end
)
if "%1" == "htmlhelp" (
	%SPHINXBUILD% -b htmlhelp %ALLSPHINXOPTS% %BUILDDIR%/htmlhelp
	if errorlevel 1 exit /b 1
	echo.
	echo.Build finished; now you can run HTML Help Workshop with the ^
.hhp project file in %BUILDDIR%/htmlhelp.
	goto end
)
if "%1" == "qthelp" (
	%SPHINXBUILD% -b qthelp %ALLSPHINXOPTS% %BUILDDIR%/qthelp
	if errorlevel 1 exit /b 1
	echo.
	echo.Build finished; now you can run "qcollectiongenerator" with the
^
.qhcp project file in %BUILDDIR%/qthelp, like this:
	echo.^> qcollectiongenerator
%BUILDDIR%\qthelp\MlOps-Mini_Project.qhcp
	echo.To view the help file:
	echo.^> assistant -collectionFile
%BUILDDIR%\qthelp\MlOps-Mini_Project.ghc
	goto end
)
```

```
if "%1" == "devhelp" (
    %SPHINXBUILD% -b devhelp %ALLSPHINXOPTS% %BUILDDIR%/devhelp
    if errorlevel 1 exit /b 1
    echo.
    echo.Build finished.
    goto end
)
if "%1" == "epub" (
    %SPHINXBUILD% -b epub %ALLSPHINXOPTS% %BUILDDIR%/epub
    if errorlevel 1 exit /b 1
    echo.
    echo.Build finished. The epub file is in %BUILDDIR%/epub.
    goto end
)
if "%1" == "latex" (
    %SPHINXBUILD% -b latex %ALLSPHINXOPTS% %BUILDDIR%/latex
    if errorlevel 1 exit /b 1
    echo.
    echo.Build finished; the LaTeX files are in %BUILDDIR%/latex.
    goto end
)
if "%1" == "text" (
    %SPHINXBUILD% -b text %ALLSPHINXOPTS% %BUILDDIR%/text
    if errorlevel 1 exit /b 1
    echo.
    echo.Build finished. The text files are in %BUILDDIR%/text.
    goto end
)
if "%1" == "man" (
    %SPHINXBUILD% -b man %ALLSPHINXOPTS% %BUILDDIR%/man
    if errorlevel 1 exit /b 1
    echo.
    echo.Build finished. The manual pages are in %BUILDDIR%/man.
    goto end
)
if "%1" == "texinfo" (
    %SPHINXBUILD% -b texinfo %ALLSPHINXOPTS% %BUILDDIR%/texinfo
    if errorlevel 1 exit /b 1
    echo.
    echo.Build finished. The Texinfo files are in %BUILDDIR%/texinfo.
    goto end
)
if "%1" == "gettext" (
    %SPHINXBUILD% -b gettext %I18NSPHINXOPTS% %BUILDDIR%/locale
    if errorlevel 1 exit /b 1
    echo.
    echo.Build finished. The message catalogs are in %BUILDDIR%/locale.
    goto end
)
if "%1" == "changes" (
    %SPHINXBUILD% -b changes %ALLSPHINXOPTS% %BUILDDIR%/changes
    if errorlevel 1 exit /b 1
    echo.
    echo.The overview file is in %BUILDDIR%/changes.
    goto end
)
if "%1" == "linkcheck" (
    %SPHINXBUILD% -b linkcheck %ALLSPHINXOPTS% %BUILDDIR%/linkcheck
    if errorlevel 1 exit /b 1
    echo.
    echo.Link check complete; look for any errors in the above output ^
or in %BUILDDIR%/linkcheck/output.txt.
    goto end
)
if "%1" == "doctest" (
    %SPHINXBUILD% -b doctest %ALLSPHINXOPTS% %BUILDDIR%/doctest
    if errorlevel 1 exit /b 1
    echo.
    echo.Testing of doctests in the sources finished, look at the ^
results in %BUILDDIR%/doctest/output.txt.
    goto end
)
:end
```

dvc.lock

```
schema: '2.0'
stages:
data_ingestion:
cmd: python src/data/data_ingestion.py
deps:
- path: src/data/data_ingestion.py
hash: md5
md5: ee08a4c31ae467cebbcce70b5cc5491a
size: 3545
params:
params.yaml:
data_ingestion.test_size: 0.25
outs:
- path: data/raw
hash: md5
md5: 3a301dd6a0b3caa261222635788eb1a5.dir
size: 827973
nfiles: 2
data_preprocessing:
cmd: python src/data/data_preprocessing.py
deps:
- path: data/raw
hash: md5
md5: 3a301dd6a0b3caa261222635788eb1a5.dir
size: 827973
nfiles: 2
- path: src/data/data_preprocessing.py
hash: md5
md5: c1e18bf95fcc1e3d80cae9ee7c4a6383
size: 4014
outs:
- path: data/interim
hash: md5
md5: edc3efae1413d803c66f5da2a5e95764.dir
size: 572486
nfiles: 2
feature_engineering:
cmd: python src/features/feature_engineering.py
deps:
- path: data/interim
hash: md5
md5: edc3efae1413d803c66f5da2a5e95764.dir
size: 572486
nfiles: 2
- path: src/features/feature_engineering.py
hash: md5
md5: c1888aef256c204017522c2ce5fd36a0
size: 3883
params:
```

```
params.yaml:
feature_engineering.max_features: 3500
outs:
- path: data/processed
hash: md5
md5: ad12a2be65f5d7c35633cb796eed8362.dir
size: 72681916
nfiles: 2
- path: models/vectorizer.pkl
hash: md5
md5: 1f65028238db7f7738ff745dea57a742
size: 83204
model_building:
cmd: python src/model/model_building.py
deps:
- path: data/processed
hash: md5
md5: ad12a2be65f5d7c35633cb796eed8362.dir
size: 72681916
nfiles: 2
- path: src/model/model_building.py
hash: md5
md5: 809681a98d52938b43c9befd0b00a422
size: 2373
outs:
- path: models/model.pkl
hash: md5
md5: 8775b4db6fb95f077289fcbc47219d8b
size: 28716
model_evaluation:
cmd: python src/model/model_evaluation.py
deps:
- path: models/model.pkl
hash: md5
md5: 8775b4db6fb95f077289fcbc47219d8b
size: 28716
- path: src/model/model_evaluation.py
hash: md5
md5: 29267b632dccb26952d0a1d6dc81c841
size: 5318
outs:
- path: reports/experiment_info.json
hash: md5
md5: 348f515ddf54815f78cb838ecebdc46d
size: 82
- path: reports/metrics.json
hash: md5
md5: b313db5181179cf92e682c08a8f346f1
size: 144
model_registration:
cmd: python src/model/register_model.py
```

```
deps:
- path: reports/experiment_info.json
hash: md5
md5: 348f515ddf54815f78cb838ecebdc46d
size: 82
- path: src/model/register_model.py
hash: md5
md5: 584de993043619f557dfd129a8fe0274
size: 2550
```

dvc.yaml

```
# updated dvc.yaml
stages:
data_ingestion:
cmd: python src/data/data_ingestion.py
deps:
- src/data/data_ingestion.py
params:
- data_ingestion.test_size
outs:
- data/raw
data_preprocessing:
cmd: python src/data/data_preprocessing.py
deps:
- data/raw
- src/data/data_preprocessing.py
outs:
- data/interim
feature_engineering:
cmd: python src/features/feature_engineering.py
deps:
- data/interim
- src/features/feature_engineering.py
params:
- feature_engineering.max_features
outs:
- data/processed
- models/vectorizer.pkl
model_building:
cmd: python src/model/model_building.py
deps:
- data/processed
- src/model/model_building.py
outs:
- models/model.pkl
model_evaluation:
cmd: python src/model/model_evaluation.py
deps:
- models/model.pkl
- src/model/model_evaluation.py
metrics:
- reports/metrics.json
outs:
- reports/experiment_info.json  # Add the model_info.json file as an
output
model_registration:
cmd: python src/model/register_model.py
deps:
- reports/experiment_info.json
- src/model/register_model.py
```

app.py

```
from flask import Flask, render_template,request
import mlflow
from preprocessing_utility import normalize_text
import dagshub
import pickle

mlflow.set_tracking_uri("https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-Pro
dagshub.init(repo_owner='sudeepjoelbayyee',
repo_name='MLOps-Mini-Project', mlflow=True)
app = Flask(__name__)
# load model from model registry
def get_latest_model_version(model_name):
client = mlflow.MlflowClient()
latest_version = client.get_latest_versions(model_name,
stages=["Production"])
if not latest_version:
latest_version = client.get_latest_versions(model_name,
stages=["None"])
return latest_version[0].version if latest_version else None
model_name = "my_model"
model_version = get_latest_model_version(model_name)
model_uri = f'models:/{model_name}/{model_version}'
model = mlflow.pyfunc.load_model(model_uri)
vectorizer = pickle.load(open('models/vectorizer.pkl','rb'))
@app.route("/")
def home():
return render_template("index.html",result=None)
@app.route("/predict", methods=['POST'])
def predict():
text = request.form['text']
# clean
text = normalize_text(text)
# BOW
features = vectorizer.transform([text])
# prediction
result = model.predict(features)
return render_template("index.html",result=result[0])
return text
app.run(debug=True)
```

preprocessing_utility.py

```
import numpy as np
import pandas as pd
import os
import re
import nltk
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
def lemmatization(text):
"""Lemmatize the text."""
lemmatizer = WordNetLemmatizer()
text = text.split()
text = [lemmatizer.lemmatize(word) for word in text]
return " ".join(text)
def remove_stop_words(text):
"""Remove stop words from the text."""
stop_words = set(stopwords.words("english"))
text = [word for word in str(text).split() if word not in stop_words]
return " ".join(text)
def removing_numbers(text):
"""Remove numbers from the text."""
text = ''.join([char for char in text if not char.isdigit()])
return text
def lower_case(text):
"""Convert text to lower case."""
text = text.split()
text = [word.lower() for word in text]
return " ".join(text)
def removing_punctuations(text):
"""Remove punctuations from the text."""
text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
text = text.replace('', "")
text = re.sub('\s+', ' ', text).strip()
return text
def removing_urls(text):
"""Remove URLs from the text."""
url_pattern = re.compile(r'https?://\S+|www\.\S+')
return url_pattern.sub(r'', text)
def remove_small_sentences(df):
"""Remove sentences with less than 3 words."""
for i in range(len(df)):
if len(df.text.iloc[i].split()) < 3:
df.text.iloc[i] = np.nan
def normalize_text(text):
text = lower_case(text)
text = remove_stop_words(text)
text = removing_numbers(text)
text = removing_punctuations(text)
text = removing_urls(text)
text = lemmatization(text)
return text
```

index.html

```html
<!DOCTYPE html>
<html>
<head>
<title>Sentiment Analysis</title>
</head>
<body>
<h1>Sentiment Analysis</h1>
<form action="/predict" method="POST">
<label>Write text:</label><br>
<textarea name="text" rows="10" cols="40"></textarea><br>
<input type="submit" value="Predict">
</form>
{% if result is not none %}
{% if result == 1 %}
<h2>Happy</h2>
{% else %}
<h2>Sad</h2>
{% endif %}
{% endif %}
</body>
</html>
```

meta.yaml

```
artifact_location:
file:///D:/Projects/MLOps/MlOps-Mini_Project/mlruns/0
creation_time: 1723598035686
experiment_id: '0'
last_update_time: 1723598035686
lifecycle_stage: active
name: Default
```

.gitkeep

.gitkeep

dagshub_setup.py

```
import dagshub
import mlflow

mlflow.set_tracking_uri("https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-Pro
dagshub.init(repo_owner='sudeepjoelbayyee',
repo_name='MLOps-Mini-Project', mlflow=True)
with mlflow.start_run():
mlflow.log_param('parameter name', 'value')
mlflow.log_metric('metric name', 1)
```

exp1_baseline_model.ipynb

```
{
"cells": [
{
"cell_type": "code",
"execution_count": 1,
"metadata": {},
"outputs": [],
"source": [
"import mlflow\n",
"import pandas as pd\n",
"import mlflow.sklearn\n",
"from sklearn.feature_extraction.text import CountVectorizer\n",
"from sklearn.model_selection import train_test_split\n",
"from sklearn.linear_model import LogisticRegression\n",
"from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score\n",
"import pandas as pd\n",
"import re\n",
"import string\n",
"from nltk.corpus import stopwords\n",
"from nltk.stem import WordNetLemmatizer\n",
"import numpy as np"
]
},
{
"cell_type": "markdown",
"metadata": {},
"source": [
"## Get Data"
]
},
{
"cell_type": "code",
"execution_count": 2,
"metadata": {},
"outputs": [
{
"data": {
"text/html": [
"<div>\n",
"<style scoped>\n",
"    .dataframe tbody tr th:only-of-type {\n",
"        vertical-align: middle;\n",
"    }\n",
"\n",
"    .dataframe tbody tr th {\n",
"        vertical-align: top;\n",
"    }\n",
"\n",
"    .dataframe thead th {\n",
"        text-align: right;\n",
"    }\n",
"</style>\n",
"<table border=\"1\" class=\"dataframe\">\n",
"  <thead>\n",
"    <tr style=\"text-align: right;\">\n",
"      <th></th>\n",
"      <th>sentiment</th>\n",
"      <th>content</th>\n",
"    </tr>\n",
"  </thead>\n",
"  <tbody>\n",
"    <tr>\n",
"      <th>0</th>\n",
"      <td>empty</td>\n",
"      <td>@tiffanylue i know  i was listenin to bad habi...</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>1</th>\n",
"      <td>sadness</td>\n",
"      <td>Layin n bed with a headache  ughhhh...waitin o...</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>2</th>\n",
"      <td>sadness</td>\n",
"      <td>Funeral ceremony...gloomy friday...</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>3</th>\n",
"      <td>enthusiasm</td>\n",
"      <td>wants to hang out with friends SOON!</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>4</th>\n",
"      <td>neutral</td>\n",
"      <td>@dannycastillo We want to trade with someone w...</td>\n",
"    </tr>\n",
"  </tbody>\n",
"</table>\n",
"</div>"
],
"text/plain": [
"    sentiment                                            content\n",
"0       empty  @tiffanylue i know  i was listenin to bad habi...\n",
"1     sadness  Layin n bed with a headache  ughhhh...waitin o...\n",
"2     sadness                Funeral ceremony...gloomy friday...\n",
"3  enthusiasm               wants to hang out with friends SOON!\n",
"4     neutral  @dannycastillo We want to trade with someone w..."
]
},
"execution_count": 2,
"metadata": {},
"output_type": "execute_result"
}
],
"source": [
"df =
pd.read_csv('https://raw.githubusercontent.com/campusx-official/jupyter-mast
"df.head()"
]
},
{
"cell_type": "markdown",
"metadata": {},
"source": [
"## Data preprocessing"
]
},
{
"cell_type": "code",
"execution_count": 3,
"metadata": {},
"outputs": [
{
"name": "stderr",
"output_type": "stream",
"text": [
"<>:30: SyntaxWarning: invalid escape sequence '\\s'\n",
"<>:30: SyntaxWarning: invalid escape sequence '\\s'\n",

"C:\\Users\\Asus\\AppData\\Local\\Temp\\ipykernel_7756\\4275467378.py:30:
SyntaxWarning: invalid escape sequence '\\s'\n",
"  text = re.sub('\\s+', ' ', text).strip()\n"
]
}
],
"source": [
"# Define text preprocessing functions\n",
"def lemmatization(text):\n",
"    \"\"\"Lemmatize the text.\"\"\"\n",
"    lemmatizer = WordNetLemmatizer()\n",
"    text = text.split()\n",
"    text = [lemmatizer.lemmatize(word) for word in text]\n",
"    return \" \".join(text)\n",
"\n",
"def remove_stop_words(text):\n",
"    \"\"\"Remove stop words from the text.\"\"\"\n",
"    stop_words = set(stopwords.words(\"english\"))\n",
"    text = [word for word in str(text).split() if word not in
stop_words]\n",
"    return \" \".join(text)\n",
"\n",
"def removing_numbers(text):\n",
"    \"\"\"Remove numbers from the text.\"\"\"\n",
"    text = ''.join([char for char in text if not
char.isdigit()])\n",
"    return text\n",
"\n",
"def lower_case(text):\n",
"    \"\"\"Convert text to lower case.\"\"\"\n",
"    text = text.split()\n",
"    text = [word.lower() for word in text]\n",
"    return \" \".join(text)\n",
"\n",
"def removing_punctuations(text):\n",
"    \"\"\"Remove punctuations from the text.\"\"\"\n",
"    text = re.sub('[%s]' % re.escape(string.punctuation), ' ',
text)\n",
"    text = text.replace('', \"\")\n",
"    text = re.sub('\\s+', ' ', text).strip()\n",
"    return text\n",
"\n",
"def removing_urls(text):\n",
"    \"\"\"Remove URLs from the text.\"\"\"\n",
"    url_pattern = re.compile(r'https?://\\S+|www\\.\\S+')\n",
"    return url_pattern.sub(r'', text)\n",
"\n",
"def normalize_text(df):\n",
"    \"\"\"Normalize the text data.\"\"\"\n",
"    try:\n",
"        df['content'] = df['content'].apply(lower_case)\n",
"        df['content'] = df['content'].apply(remove_stop_words)\n",
"        df['content'] = df['content'].apply(removing_numbers)\n",
"        df['content'] =
df['content'].apply(removing_punctuations)\n",
"        df['content'] = df['content'].apply(removing_urls)\n",
"        df['content'] = df['content'].apply(lemmatization)\n",
"        return df\n",
"    except Exception as e:\n",
"        print(f'Error during text normalization: {e}')\n",
"        raise"
]
},
{
"cell_type": "code",
"execution_count": 4,
"metadata": {},
"outputs": [
{
"data": {
"text/html": [
"<div>\n",
"<style scoped>\n",
"    .dataframe tbody tr th:only-of-type {\n",
"        vertical-align: middle;\n",
"    }\n",
"\n",
"    .dataframe tbody tr th {\n",
"        vertical-align: top;\n",
"    }\n",
"\n",
"    .dataframe thead th {\n",
"        text-align: right;\n",
"    }\n",
"</style>\n",
"<table border=\"1\" class=\"dataframe\">\n",
"  <thead>\n",
"    <tr style=\"text-align: right;\">\n",
"      <th></th>\n",
"      <th>sentiment</th>\n",
"      <th>content</th>\n",
"    </tr>\n",
"  </thead>\n",
"  <tbody>\n",
"    <tr>\n",
"      <th>0</th>\n",
"      <td>empty</td>\n",
"      <td>tiffanylue know listenin bad habit earlier sta...</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>1</th>\n",
"      <td>sadness</td>\n",
"      <td>layin n bed headache ughhhh waitin call</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>2</th>\n",
"      <td>sadness</td>\n",
"      <td>funeral ceremony gloomy friday</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>3</th>\n",
"      <td>enthusiasm</td>\n",
"      <td>want hang friend soon</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>4</th>\n",
"      <td>neutral</td>\n",
"      <td>dannycastillo want trade someone houston ticke...</td>\n",
"    </tr>\n",
"  </tbody>\n",
"</table>\n",
"</div>"
],
"text/plain": [
"    sentiment                                           content\n",
"0       empty  tiffanylue know listenin bad habit earlier sta...\n",
"1     sadness            layin n bed headache ughhhh waitin call\n",
"2     sadness                    funeral ceremony gloomy friday\n",
"3  enthusiasm                             want hang friend soon\n",
"4     neutral  dannycastillo want trade someone houston ticke..."
]
},
"execution_count": 4,
"metadata": {},
"output_type": "execute_result"
}
],
"source": [
"df = normalize_text(df)\n",
"df.head()"
]
},
{
"cell_type": "code",
"execution_count": 5,
"metadata": {},
"outputs": [
{
"data": {
"text/plain": [
"sentiment\n",
"neutral       8638\n",
"worry         8459\n",
"happiness     5209\n",
"sadness       5165\n",
"love          3842\n",
"surprise      2187\n",
"fun           1776\n",
"relief        1526\n",
"hate          1323\n",
"empty          827\n",
"enthusiasm     759\n",
"boredom        179\n",
"anger          110\n",
"Name: count, dtype: int64"
]
},
"execution_count": 5,
"metadata": {},
"output_type": "execute_result"
}
],
"source": [
"df['sentiment'].value_counts()"
```

```
]
},
{
"cell_type": "code",
"execution_count": 6,
"metadata": {},
"outputs": [],
"source": [
"x = df['sentiment'].isin(['happiness','sadness'])\n",
"df = df[x]"
]
},
{
"cell_type": "code",
"execution_count": 7,
"metadata": {},
"outputs": [
{
"name": "stderr",
"output_type": "stream",
"text": [

"C:\\Users\\Asus\\AppData\\Local\\Temp\\ipykernel_7756\\1089524538.py:1:
FutureWarning: Downcasting behavior in `replace` is deprecated and
will be removed in a future version. To retain the old behavior,
explicitly call `result.infer_objects(copy=False)`. To opt-in to the
future behavior, set `pd.set_option('future.no_silent_downcasting',
True)`\n",
"  df['sentiment'] = df['sentiment'].replace({'sadness':0,
'happiness':1})\n"
]
},
{
"data": {
"text/html": [
"<div>\n",
"<style scoped>\n",
"    .dataframe tbody tr th:only-of-type {\n",
"        vertical-align: middle;\n",
"    }\n",
"\n",
"    .dataframe tbody tr th {\n",
"        vertical-align: top;\n",
"    }\n",
"\n",
"    .dataframe thead th {\n",
"        text-align: right;\n",
"    }\n",
"</style>\n",
"<table border=\"1\" class=\"dataframe\">\n",
"  <thead>\n",
"    <tr style=\"text-align: right;\">\n",
"      <th></th>\n",
"      <th>sentiment</th>\n",
"      <th>content</th>\n",
"    </tr>\n",
"  </thead>\n",
"  <tbody>\n",
"    <tr>\n",
"      <th>1</th>\n",
"      <td>0</td>\n",
"      <td>layin n bed headache ughhhh waitin call</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>2</th>\n",
"      <td>0</td>\n",
"      <td>funeral ceremony gloomy friday</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>6</th>\n",
"      <td>0</td>\n",
"      <td>sleep im not thinking old friend want he s mar...</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>8</th>\n",
"      <td>0</td>\n",
"      <td>charviray charlene love miss</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>9</th>\n",
"      <td>0</td>\n",
"      <td>kelcouch i m sorry least friday</td>\n",
"    </tr>\n",
"  </tbody>\n",
"</table>\n",
"</div>"
],
"text/plain": [
"   sentiment                                           content\n",
"1          0           layin n bed headache ughhhh waitin call\n",
"2          0                    funeral ceremony gloomy friday\n",
"6          0  sleep im not thinking old friend want he s mar...\n",
"8          0                      charviray charlene love miss\n",
"9          0                   kelcouch i m sorry least friday"
]
},
"execution_count": 7,
"metadata": {},
"output_type": "execute_result"
}
],
"source": [
"df['sentiment'] = df['sentiment'].replace({'sadness':0,
'happiness':1})\n",
"df.head()"
]
},
{
"cell_type": "code",
"execution_count": 8,
"metadata": {},
"outputs": [],
"source": [
"vectorizer = CountVectorizer(max_features=1000)\n",
"X = vectorizer.fit_transform(df['content'])\n",
"y = df['sentiment']"
]
},
{
"cell_type": "code",
"execution_count": 9,
"metadata": {},
"outputs": [],
"source": [
"X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)"
]
},
{
"cell_type": "code",
"execution_count": 10,
"metadata": {},
"outputs": [
{
"data": {
"text/html": [
"<pre
style=\"white-space:pre;overflow-x:auto;line-height:normal;font-family:Menlo
Sans Mono',consolas,'Courier New',monospace\">
<span style=\"font-weight: bold\"> AUTHORIZATION REQUIRED </span>
\n",
"</pre>\n"
],
"text/plain": [
"                                        \u001b[1m AUTHORIZATION
REQUIRED \u001b[0m                                        \n"
]
},
"metadata": {},
"output_type": "display_data"
},
{
"data": {
```

```
"text/html": [
"<pre
style=\"white-space:pre;overflow-x:auto;line-height:normal;font-family:Menlo
Sans Mono',consolas,'Courier
New',monospace\">d:\\Projects\\MLOps\\MlOps-Mini_Project\\mlopsmini\\Lib\\si
UserWarning: install \n",
"\"ipywidgets\" for Jupyter support\n",
"  warnings.warn('install \"ipywidgets\" for Jupyter support')\n",
"</pre>\n"
],
"text/plain": [

"d:\\Projects\\MLOps\\MlOps-Mini_Project\\mlopsmini\\Lib\\site-packages\\ric
UserWarning: install \n",
"\"ipywidgets\" for Jupyter support\n",
"  warnings.warn('install \"ipywidgets\" for Jupyter support')\n"
]
},
"metadata": {},
"output_type": "display_data"
},
{
"name": "stdout",
"output_type": "stream",
"text": [
"\n",
"\n",
"Open the following link in your browser to authorize the client:\n",

"https://dagshub.com/login/oauth/authorize?state=931e1b82-433f-4cbf-8c14-94f
"\n",
"\n"
]
},
{
"data": {
"text/html": [
"<pre
style=\"white-space:pre;overflow-x:auto;line-height:normal;font-family:Menlo
Sans Mono',consolas,'Courier New',monospace\"></pre>\n"
],
"text/plain": []
},
"metadata": {},
"output_type": "display_data"
},
{
"data": {
"text/html": [
"<pre
style=\"white-space:pre;overflow-x:auto;line-height:normal;font-family:Menlo
```

```
Sans Mono',consolas,'Courier New',monospace\">Accessing as
sudeepjoelbayyee\n",
"</pre>\n"
],
"text/plain": [
"Accessing as sudeepjoelbayyee\n"
]
},
"metadata": {},
"output_type": "display_data"
},
{
"data": {
"text/html": [
"<pre
style=\"white-space:pre;overflow-x:auto;line-height:normal;font-family:Menlo
Sans Mono',consolas,'Courier New',monospace\">Initialized MLflow to
track repo <span style=\"color: #008000; text-decoration-color:
#008000\">\"sudeepjoelbayyee/MLOps-Mini-Project\"</span>\n",
"</pre>\n"
],
"text/plain": [
"Initialized MLflow to track repo
\u001b[32m\"sudeepjoelbayyee/MLOps-Mini-Project\"\u001b[0m\n"
]
},
"metadata": {},
"output_type": "display_data"
},
{
"data": {
"text/html": [
"<pre
style=\"white-space:pre;overflow-x:auto;line-height:normal;font-family:Menlo
Sans Mono',consolas,'Courier New',monospace\">Repository
sudeepjoelbayyee/MLOps-Mini-Project initialized!\n",
"</pre>\n"
],
"text/plain": [
"Repository sudeepjoelbayyee/MLOps-Mini-Project initialized!\n"
]
},
"metadata": {},
"output_type": "display_data"
},
{
"name": "stderr",
"output_type": "stream",
"text": [
"2024/08/14 01:30:23 INFO mlflow.tracking.fluent: Experiment with
name 'Logistic Regression Baseline' does not exist. Creating a new
```

experiment.\n"
]
},
{
"data": {
"text/plain": [
"<Experiment:
artifact_location='mlflow-artifacts:/5c330e6290c74d48a2337087ff598535',
creation_time=1723579227490, experiment_id='0',
last_update_time=1723579227490, lifecycle_stage='active',
name='Logistic Regression Baseline', tags={}>"
]
},
"execution_count": 10,
"metadata": {},
"output_type": "execute_result"
}
],
"source": [
"import dagshub\n",
"\n",

"mlflow.set_tracking_uri(\"https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-P
"dagshub.init(repo_owner='sudeepjoelbayyee',
repo_name='MLOps-Mini-Project', mlflow=True)\n",
"\n",
"mlflow.set_experiment(\"Logistic Regression Baseline\")"
]
},
{
"cell_type": "code",
"execution_count": 11,
"metadata": {},
"outputs": [
{
"name": "stderr",
"output_type": "stream",
"text": [
"2024/08/14 01:32:13 WARNING mlflow.models.model: Input example
should be provided to infer model signature if the model signature is
not provided when logging the model.\n"
]
},
{
"name": "stdout",
"output_type": "stream",
"text": [
"Accuracy: 0.779277108433735\n",
"Precision: 0.7701260911736179\n",
"Recall: 0.7822660098522167\n",
"F1 Score: 0.7761485826001955\n"

```
]
},
{
"name": "stderr",
"output_type": "stream",
"text": [
"2024/08/14 01:33:05 INFO mlflow.tracking._tracking_service.client:
View run spiffy-shark-454 at:
https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-Project.mlflow/#/experiments
"2024/08/14 01:33:05 INFO mlflow.tracking._tracking_service.client:
View experiment at:
https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-Project.mlflow/#/experiments
]
}
],
"source": [
"with mlflow.start_run():\n",
"    \n",
"    # Log preprocessing parameters\n",
"    mlflow.log_param(\"vectorizer\", \"Bag of Words\")\n",
"    mlflow.log_param(\"num_features\", 1000)\n",
"    mlflow.log_param(\"test_size\", 0.2)\n",
"    \n",
"    # Model building and training\n",
"    model = LogisticRegression()\n",
"    model.fit(X_train, y_train)\n",
"    \n",
"    # Log model parameters\n",
"    mlflow.log_param(\"model\", \"Logistic Regression\")\n",
"    \n",
"    # Model evaluation\n",
"    y_pred = model.predict(X_test)\n",
"    accuracy = accuracy_score(y_test, y_pred)\n",
"    precision = precision_score(y_test, y_pred)\n",
"    recall = recall_score(y_test, y_pred)\n",
"    f1 = f1_score(y_test, y_pred)\n",
"    \n",
"    # Log evaluation metrics\n",
"    mlflow.log_metric(\"accuracy\", accuracy)\n",
"    mlflow.log_metric(\"precision\", precision)\n",
"    mlflow.log_metric(\"recall\", recall)\n",
"    mlflow.log_metric(\"f1_score\", f1)\n",
"    \n",
"    # Log model\n",
"    mlflow.sklearn.log_model(model, \"model\")\n",
"\n",
"    # Save and log the notebook\n",
"    import os\n",
"    notebook_path = \"exp1_baseline_model.ipynb\"\n",
"    os.system(f\"jupyter nbconvert --to notebook --execute --inplace
{notebook_path}\")\n",
"    mlflow.log_artifact(notebook_path)\n",
"    \n",
"    # Print the results for verification\n",
"    print(f\"Accuracy: {accuracy}\")\n",
"    print(f\"Precision: {precision}\")\n",
"    print(f\"Recall: {recall}\")\n",
"    print(f\"F1 Score: {f1}\")"
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {},
"outputs": [],
"source": []
}
],
"metadata": {
"kernelspec": {
"display_name": "mlopsmini",
"language": "python",
"name": "python3"
},
"language_info": {
"codemirror_mode": {
"name": "ipython",
"version": 3
},
"file_extension": ".py",
"mimetype": "text/x-python",
"name": "python",
"nbconvert_exporter": "python",
"pygments_lexer": "ipython3",
"version": "3.12.4"
}
},
"nbformat": 4,
"nbformat_minor": 2
}
```

exp2_dif_models_f_extraction.py

```
# Import necessary libraries
import mlflow
import mlflow.sklearn
import mlflow.keras
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
import pandas as pd
import re
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import numpy as np
import os
import dagshub
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding,
SpatialDropout1D
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

mlflow.set_tracking_uri("https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-Pro
dagshub.init(repo_owner='sudeepjoelbayyee',
repo_name='MLOps-Mini-Project', mlflow=True)
# Load the data
df =
pd.read_csv('https://raw.githubusercontent.com/campusx-official/jupyter-mast
df.head()
# Define text preprocessing functions
def lemmatization(text):
"""Lemmatize the text."""
lemmatizer = WordNetLemmatizer()
text = text.split()
text = [lemmatizer.lemmatize(word) for word in text]
return " ".join(text)
def remove_stop_words(text):
"""Remove stop words from the text."""
stop_words = set(stopwords.words("english"))
text = [word for word in str(text).split() if word not in stop_words]
return " ".join(text)
def removing_numbers(text):
"""Remove numbers from the text."""
```

```
text = ''.join([char for char in text if not char.isdigit()])
return text
def lower_case(text):
"""Convert text to lower case."""
text = text.split()
text = [word.lower() for word in text]
return " ".join(text)
def removing_punctuations(text):
"""Remove punctuations from the text."""
text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
text = text.replace('', "")
text = re.sub('\s+', ' ', text).strip()
return text
def removing_urls(text):
"""Remove URLs from the text."""
url_pattern = re.compile(r'https?://\S+|www\.\S+')
return url_pattern.sub(r'', text)
def normalize_text(df):
"""Normalize the text data."""
try:
df['content'] = df['content'].apply(lower_case)
df['content'] = df['content'].apply(remove_stop_words)
df['content'] = df['content'].apply(removing_numbers)
df['content'] = df['content'].apply(removing_punctuations)
df['content'] = df['content'].apply(removing_urls)
df['content'] = df['content'].apply(lemmatization)
return df
except Exception as e:
print(f'Error during text normalization: {e}')
raise
# Normalize the text data
df = normalize_text(df)
x = df['sentiment'].isin(['happiness','sadness'])
df = df[x]
df['sentiment'] = df['sentiment'].replace({'sadness':0,
'happiness':1})
# Set the experiment name
mlflow.set_experiment("Bow vs TfIdf vs LSTM")
def build_lstm_model(input_length):
model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128,
input_length=input_length))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
return model
# Define feature extraction methods
vectorizers = {
'BoW': CountVectorizer(),
'TF-IDF': TfidfVectorizer()
}
# Define algorithms
algorithms = {
'LogisticRegression': LogisticRegression(),
'MultinomialNB': MultinomialNB(),
'XGBoost': XGBClassifier(),
'RandomForest': RandomForestClassifier(),
'GradientBoosting': GradientBoostingClassifier(),
'LSTM':build_lstm_model
}
# LSTM-specific preprocessing
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(df['content'])
X_lstm = tokenizer.texts_to_sequences(df['content'])
X_lstm = pad_sequences(X_lstm, maxlen=100)
# Start the parent run
with mlflow.start_run(run_name="All Experiments") as parent_run:
# Loop through algorithms and feature extraction methods (Child Runs)
for algo_name, algorithm in algorithms.items():
if algo_name == 'LSTM':
# Use LSTM-specific data
X_train, X_test, y_train, y_test = train_test_split(X_lstm,
df['sentiment'], test_size=0.2, random_state=42)
input_length = X_train.shape[1]
model = algorithm(input_length)  # Build the LSTM model
with mlflow.start_run(run_name=f"{algo_name}", nested=True) as
child_run:
# LSTM Model training
model.fit(X_train, y_train, epochs=5, batch_size=64,
validation_data=(X_test, y_test), verbose=2)
# Model evaluation
y_pred = (model.predict(X_test) > 0.5).astype("int32")
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
# Log evaluation metrics
mlflow.log_metric("accuracy", accuracy)
mlflow.log_metric("precision", precision)
mlflow.log_metric("recall", recall)
mlflow.log_metric("f1_score", f1)
# Log LSTM model
mlflow.keras.log_model(model, "model")
# Print the results for verification
print(f"Algorithm: {algo_name}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
else:
```

```
for vec_name, vectorizer in vectorizers.items():
with mlflow.start_run(run_name=f"{algo_name} with {vec_name}",
nested=True) as child_run:
X = vectorizer.fit_transform(df['content'])
y = df['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Log preprocessing parameters
mlflow.log_param("vectorizer", vec_name)
mlflow.log_param("algorithm", algo_name)
mlflow.log_param("test_size", 0.2)
# Model training
model = algorithm
model.fit(X_train, y_train)
# Log model parameters
if algo_name == 'LogisticRegression':
mlflow.log_param("C", model.C)
elif algo_name == 'MultinomialNB':
mlflow.log_param("alpha", model.alpha)
elif algo_name == 'XGBoost':
mlflow.log_param("n_estimators", model.n_estimators)
mlflow.log_param("learning_rate", model.learning_rate)
elif algo_name == 'RandomForest':
mlflow.log_param("n_estimators", model.n_estimators)
mlflow.log_param("max_depth", model.max_depth)
elif algo_name == 'GradientBoosting':
mlflow.log_param("n_estimators", model.n_estimators)
mlflow.log_param("learning_rate", model.learning_rate)
mlflow.log_param("max_depth", model.max_depth)
# Model evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
# Log evaluation metrics
mlflow.log_metric("accuracy", accuracy)
mlflow.log_metric("precision", precision)
mlflow.log_metric("recall", recall)
mlflow.log_metric("f1_score", f1)
# Log model
mlflow.sklearn.log_model(model, "model")
# Save and log the notebook
mlflow.log_artifact(__file__)
# Print the results for verification
print(f"Algorithm: {algo_name}, Feature Engineering: {vec_name}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

exp3_lor_bow_hp.py

```
# hyperparameter tuning
# Import necessary libraries
import mlflow
import mlflow.sklearn
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
import pandas as pd
import re
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import numpy as np
import os
import dagshub

mlflow.set_tracking_uri("https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-Pro
dagshub.init(repo_owner='sudeepjoelbayyee',
repo_name='MLOps-Mini-Project', mlflow=True)
# Load the data
# Load the data
df =
pd.read_csv('https://raw.githubusercontent.com/campusx-official/jupyter-mast
# Define text preprocessing functions
def lemmatization(text):
"""Lemmatize the text."""
lemmatizer = WordNetLemmatizer()
text = text.split()
text = [lemmatizer.lemmatize(word) for word in text]
return " ".join(text)
def remove_stop_words(text):
"""Remove stop words from the text."""
stop_words = set(stopwords.words("english"))
text = [word for word in str(text).split() if word not in stop_words]
return " ".join(text)
def removing_numbers(text):
"""Remove numbers from the text."""
text = ''.join([char for char in text if not char.isdigit()])
return text
def lower_case(text):
"""Convert text to lower case."""
text = text.split()
text = [word.lower() for word in text]
return " ".join(text)
def removing_punctuations(text):
"""Remove punctuations from the text."""
```

```
text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
text = text.replace('', "")
text = re.sub('\s+', ' ', text).strip()
return text
def removing_urls(text):
"""Remove URLs from the text."""
url_pattern = re.compile(r'https?://\S+|www\.\S+')
return url_pattern.sub(r'', text)
def normalize_text(df):
"""Normalize the text data."""
try:
df['content'] = df['content'].apply(lower_case)
df['content'] = df['content'].apply(remove_stop_words)
df['content'] = df['content'].apply(removing_numbers)
df['content'] = df['content'].apply(removing_punctuations)
df['content'] = df['content'].apply(removing_urls)
df['content'] = df['content'].apply(lemmatization)
return df
except Exception as e:
print(f'Error during text normalization: {e}')
raise
# Normalize the text data
df = normalize_text(df)
x = df['sentiment'].isin(['happiness','sadness'])
df = df[x]
df['sentiment'] = df['sentiment'].replace({'sadness':0,
'happiness':1})
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['content'])
y = df['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Set the experiment name
mlflow.set_experiment("LoR Hyperparameter Tuning")
# Define hyperparameter grid for Logistic Regression
param_grid = {
'C': [0.1, 1, 10],
'penalty': ['l1', 'l2'],
'solver': ['liblinear']
}
# Start the parent run for hyperparameter tuning
with mlflow.start_run():
# Perform grid search
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5,
scoring='f1', n_jobs=-1)
grid_search.fit(X_train, y_train)
# Log each parameter combination as a child run
for params, mean_score, std_score in
zip(grid_search.cv_results_['params'],
grid_search.cv_results_['mean_test_score'],
grid_search.cv_results_['std_test_score']):
```

```
with mlflow.start_run(run_name=f"LR with params: {params}",
nested=True):
model = LogisticRegression(**params)
model.fit(X_train, y_train)
# Model evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
# Log parameters and metrics
mlflow.log_params(params)
mlflow.log_metric("mean_cv_score", mean_score)
mlflow.log_metric("std_cv_score", std_score)
mlflow.log_metric("accuracy", accuracy)
mlflow.log_metric("precision", precision)
mlflow.log_metric("recall", recall)
mlflow.log_metric("f1_score", f1)
# Print the results for verification
print(f"Mean CV Score: {mean_score}, Std CV Score: {std_score}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
# Log the best run details in the parent run
best_params = grid_search.best_params_
best_score = grid_search.best_score_
mlflow.log_params(best_params)
mlflow.log_metric("best_f1_score", best_score)
print(f"Best Params: {best_params}")
print(f"Best F1 Score: {best_score}")
# Save and log the notebook
mlflow.log_artifact(__file__)
# Log model
mlflow.sklearn.log_model(grid_search.best_estimator_, "model")
```

params.yaml

```
# params.yaml
data_ingestion:
test_size: 0.25
feature_engineering:
max_features: 3500
```

.gitkeep

.gitkeep

.gitkeep

requirements.txt

```
mlflow
dagshub
dvc
nltk
ipykernel
ipywidgets
xgboost
flask
```

setup.py

```
from setuptools import find_packages, setup
setup(
name='src',
packages=find_packages(),
version='0.1.0',
description='A short description of the project.',
author='Joel',
license='MIT',
)
```

.gitkeep

data_ingestion.py

```
# data ingestion
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import yaml
import logging
# logging configuration
logger = logging.getLogger('data_ingestion')
logger.setLevel('DEBUG')
console_handler = logging.StreamHandler()
console_handler.setLevel('DEBUG')
file_handler = logging.FileHandler('errors.log')
file_handler.setLevel('ERROR')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
- %(message)s')
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)
logger.addHandler(console_handler)
logger.addHandler(file_handler)
def load_params(params_path: str) -> dict:
"""Load parameters from a YAML file."""
try:
with open(params_path, 'r') as file:
params = yaml.safe_load(file)
logger.debug('Parameters retrieved from %s', params_path)
return params
except FileNotFoundError:
logger.error('File not found: %s', params_path)
raise
except yaml.YAMLError as e:
logger.error('YAML error: %s', e)
raise
except Exception as e:
logger.error('Unexpected error: %s', e)
raise
def load_data(data_url: str) -> pd.DataFrame:
"""Load data from a CSV file."""
try:
df = pd.read_csv(data_url)
logger.debug('Data loaded from %s', data_url)
return df
except pd.errors.ParserError as e:
logger.error('Failed to parse the CSV file: %s', e)
raise
except Exception as e:
logger.error('Unexpected error occurred while loading the data: %s',
e)
raise
```

```
def preprocess_data(df: pd.DataFrame) -> pd.DataFrame:
"""Preprocess the data."""
try:
df.drop(columns=['tweet_id'], inplace=True)
final_df = df[df['sentiment'].isin(['happiness', 'sadness'])]
final_df['sentiment'].replace({'happiness': 1, 'sadness': 0},
inplace=True)
logger.debug('Data preprocessing completed')
return final_df
except KeyError as e:
logger.error('Missing column in the dataframe: %s', e)
raise
except Exception as e:
logger.error('Unexpected error during preprocessing: %s', e)
raise
def save_data(train_data: pd.DataFrame, test_data: pd.DataFrame,
data_path: str) -> None:
"""Save the train and test datasets."""
try:
raw_data_path = os.path.join(data_path, 'raw')
os.makedirs(raw_data_path, exist_ok=True)
train_data.to_csv(os.path.join(raw_data_path, "train.csv"),
index=False)
test_data.to_csv(os.path.join(raw_data_path, "test.csv"),
index=False)
logger.debug('Train and test data saved to %s', raw_data_path)
except Exception as e:
logger.error('Unexpected error occurred while saving the data: %s',
e)
raise
def main():
try:
params = load_params(params_path='params.yaml')
test_size = params['data_ingestion']['test_size']
df =
load_data(data_url='https://raw.githubusercontent.com/campusx-official/jupyt
final_df = preprocess_data(df)
train_data, test_data = train_test_split(final_df,
test_size=test_size, random_state=42)
save_data(train_data, test_data, data_path='./data')
except Exception as e:
logger.error('Failed to complete the data ingestion process: %s', e)
print(f"Error: {e}")
if __name__ == '__main__':
main()
```

data_preprocessing.py

```
# data preprocessing
import numpy as np
import pandas as pd
import os
import re
import nltk
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import logging
# logging configuration
logger = logging.getLogger('data_transformation')
logger.setLevel('DEBUG')
console_handler = logging.StreamHandler()
console_handler.setLevel('DEBUG')
file_handler = logging.FileHandler('transformation_errors.log')
file_handler.setLevel('ERROR')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
- %(message)s')
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)
logger.addHandler(console_handler)
logger.addHandler(file_handler)
nltk.download('wordnet')
nltk.download('stopwords')
def lemmatization(text):
"""Lemmatize the text."""
lemmatizer = WordNetLemmatizer()
text = text.split()
text = [lemmatizer.lemmatize(word) for word in text]
return " ".join(text)
def remove_stop_words(text):
"""Remove stop words from the text."""
stop_words = set(stopwords.words("english"))
text = [word for word in str(text).split() if word not in stop_words]
return " ".join(text)
def removing_numbers(text):
"""Remove numbers from the text."""
text = ''.join([char for char in text if not char.isdigit()])
return text
def lower_case(text):
"""Convert text to lower case."""
text = text.split()
text = [word.lower() for word in text]
return " ".join(text)
def removing_punctuations(text):
"""Remove punctuations from the text."""
text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
text = text.replace('', "")
text = re.sub('\s+', ' ', text).strip()
return text
def removing_urls(text):
"""Remove URLs from the text."""
url_pattern = re.compile(r'https?://\S+|www\.\S+')
return url_pattern.sub(r'', text)
def remove_small_sentences(df):
"""Remove sentences with less than 3 words."""
for i in range(len(df)):
if len(df.text.iloc[i].split()) < 3:
df.text.iloc[i] = np.nan
def normalize_text(df):
"""Normalize the text data."""
try:
df['content'] = df['content'].apply(lower_case)
logger.debug('converted to lower case')
df['content'] = df['content'].apply(remove_stop_words)
logger.debug('stop words removed')
df['content'] = df['content'].apply(removing_numbers)
logger.debug('numbers removed')
df['content'] = df['content'].apply(removing_punctuations)
logger.debug('punctuations removed')
df['content'] = df['content'].apply(removing_urls)
logger.debug('urls')
df['content'] = df['content'].apply(lemmatization)
logger.debug('lemmatization performed')
logger.debug('Text normalization completed')
return df
except Exception as e:
logger.error('Error during text normalization: %s', e)
raise
def main():
try:
# Fetch the data from data/raw
train_data = pd.read_csv('./data/raw/train.csv')
test_data = pd.read_csv('./data/raw/test.csv')
logger.debug('data loaded properly')
# Transform the data
train_processed_data = normalize_text(train_data)
test_processed_data = normalize_text(test_data)
# Store the data inside data/processed
data_path = os.path.join("./data", "interim")
os.makedirs(data_path, exist_ok=True)
train_processed_data.to_csv(os.path.join(data_path,
"train_processed.csv"), index=False)
test_processed_data.to_csv(os.path.join(data_path,
"test_processed.csv"), index=False)
logger.debug('Processed data saved to %s', data_path)
except Exception as e:
logger.error('Failed to complete the data transformation process:
%s', e)
```

```
print(f"Error: {e}")
if __name__ == '__main__':
main()
```

.gitkeep

feature_engineering.py

```
# feature engineering
import numpy as np
import pandas as pd
import os
from sklearn.feature_extraction.text import CountVectorizer
import yaml
import logging
import pickle
# logging configuration
logger = logging.getLogger('feature_engineering')
logger.setLevel('DEBUG')
console_handler = logging.StreamHandler()
console_handler.setLevel('DEBUG')
file_handler = logging.FileHandler('feature_engineering_errors.log')
file_handler.setLevel('ERROR')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
- %(message)s')
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)
logger.addHandler(console_handler)
logger.addHandler(file_handler)
def load_params(params_path: str) -> dict:
"""Load parameters from a YAML file."""
try:
with open(params_path, 'r') as file:
params = yaml.safe_load(file)
logger.debug('Parameters retrieved from %s', params_path)
return params
except FileNotFoundError:
logger.error('File not found: %s', params_path)
raise
except yaml.YAMLError as e:
logger.error('YAML error: %s', e)
raise
except Exception as e:
logger.error('Unexpected error: %s', e)
raise
def load_data(file_path: str) -> pd.DataFrame:
"""Load data from a CSV file."""
try:
df = pd.read_csv(file_path)
df.fillna('', inplace=True)
logger.debug('Data loaded and NaNs filled from %s', file_path)
return df
except pd.errors.ParserError as e:
logger.error('Failed to parse the CSV file: %s', e)
raise
except Exception as e:
logger.error('Unexpected error occurred while loading the data: %s',
e)
raise
def apply_bow(train_data: pd.DataFrame, test_data: pd.DataFrame,
max_features: int) -> tuple:
"""Apply Count Vectorizer to the data."""
try:
vectorizer = CountVectorizer(max_features=max_features)
X_train = train_data['content'].values
y_train = train_data['sentiment'].values
X_test = test_data['content'].values
y_test = test_data['sentiment'].values
X_train_bow = vectorizer.fit_transform(X_train)
X_test_bow = vectorizer.transform(X_test)
train_df = pd.DataFrame(X_train_bow.toarray())
train_df['label'] = y_train
test_df = pd.DataFrame(X_test_bow.toarray())
test_df['label'] = y_test
pickle.dump(vectorizer, open('models/vectorizer.pkl', 'wb'))
logger.debug('Bag of Words applied and data transformed')
return train_df, test_df
except Exception as e:
logger.error('Error during Bag of Words transformation: %s', e)
raise
def save_data(df: pd.DataFrame, file_path: str) -> None:
"""Save the dataframe to a CSV file."""
try:
os.makedirs(os.path.dirname(file_path), exist_ok=True)
df.to_csv(file_path, index=False)
logger.debug('Data saved to %s', file_path)
except Exception as e:
logger.error('Unexpected error occurred while saving the data: %s',
e)
raise
def main():
try:
params = load_params('params.yaml')
max_features = params['feature_engineering']['max_features']
train_data = load_data('./data/interim/train_processed.csv')
test_data = load_data('./data/interim/test_processed.csv')
train_df, test_df = apply_bow(train_data, test_data, max_features)
save_data(train_df, os.path.join("./data", "processed",
"train_bow.csv"))
save_data(test_df, os.path.join("./data", "processed",
"test_bow.csv"))
except Exception as e:
logger.error('Failed to complete the feature engineering process:
%s', e)
print(f"Error: {e}")
if __name__ == '__main__':
main()
```

.gitkeep

model_building.py

```
# model building
import numpy as np
import pandas as pd
import pickle
from sklearn.linear_model import LogisticRegression
import yaml
import logging
# logging configuration
logger = logging.getLogger('model_building')
logger.setLevel('DEBUG')
console_handler = logging.StreamHandler()
console_handler.setLevel('DEBUG')
file_handler = logging.FileHandler('model_building_errors.log')
file_handler.setLevel('ERROR')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
- %(message)s')
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)
logger.addHandler(console_handler)
logger.addHandler(file_handler)
def load_data(file_path: str) -> pd.DataFrame:
"""Load data from a CSV file."""
try:
df = pd.read_csv(file_path)
logger.debug('Data loaded from %s', file_path)
return df
except pd.errors.ParserError as e:
logger.error('Failed to parse the CSV file: %s', e)
raise
except Exception as e:
logger.error('Unexpected error occurred while loading the data: %s',
e)
raise
def train_model(X_train: np.ndarray, y_train: np.ndarray) ->
LogisticRegression:
"""Train the Logistic Regression model."""
try:
clf = LogisticRegression(C=1, solver='liblinear', penalty='l2')
clf.fit(X_train, y_train)
logger.debug('Model training completed')
return clf
except Exception as e:
logger.error('Error during model training: %s', e)
raise
def save_model(model, file_path: str) -> None:
"""Save the trained model to a file."""
try:
with open(file_path, 'wb') as file:
pickle.dump(model, file)
```

```
logger.debug('Model saved to %s', file_path)
except Exception as e:
logger.error('Error occurred while saving the model: %s', e)
raise
def main():
try:
train_data = load_data('./data/processed/train_bow.csv')
X_train = train_data.iloc[:, :-1].values
y_train = train_data.iloc[:, -1].values
clf = train_model(X_train, y_train)
save_model(clf, 'models/model.pkl')
except Exception as e:
logger.error('Failed to complete the model building process: %s', e)
print(f"Error: {e}")
if __name__ == '__main__':
main()
```

model_evaluation.py

```
# updated model evaluation
import numpy as np
import pandas as pd
import pickle
import json
from sklearn.metrics import accuracy_score, precision_score,
recall_score, roc_auc_score
import logging
import mlflow
import mlflow.sklearn
import dagshub
import os

mlflow.set_tracking_uri("https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-Pro
dagshub.init(repo_owner='sudeepjoelbayyee',
repo_name='MLOps-Mini-Project', mlflow=True)
# logging configuration
logger = logging.getLogger('model_evaluation')
logger.setLevel('DEBUG')
console_handler = logging.StreamHandler()
console_handler.setLevel('DEBUG')
file_handler = logging.FileHandler('model_evaluation_errors.log')
file_handler.setLevel('ERROR')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
- %(message)s')
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)
logger.addHandler(console_handler)
logger.addHandler(file_handler)
def load_model(file_path: str):
"""Load the trained model from a file."""
try:
with open(file_path, 'rb') as file:
model = pickle.load(file)
logger.debug('Model loaded from %s', file_path)
return model
except FileNotFoundError:
logger.error('File not found: %s', file_path)
raise
except Exception as e:
logger.error('Unexpected error occurred while loading the model: %s',
e)
raise
def load_data(file_path: str) -> pd.DataFrame:
"""Load data from a CSV file."""
try:
df = pd.read_csv(file_path)
logger.debug('Data loaded from %s', file_path)
return df
```

```
except pd.errors.ParserError as e:
logger.error('Failed to parse the CSV file: %s', e)
raise
except Exception as e:
logger.error('Unexpected error occurred while loading the data: %s',
e)
raise
def evaluate_model(clf, X_test: np.ndarray, y_test: np.ndarray) ->
dict:
"""Evaluate the model and return the evaluation metrics."""
try:
y_pred = clf.predict(X_test)
y_pred_proba = clf.predict_proba(X_test)[:, 1]
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_proba)
metrics_dict = {
'accuracy': accuracy,
'precision': precision,
'recall': recall,
'auc': auc
}
logger.debug('Model evaluation metrics calculated')
return metrics_dict
except Exception as e:
logger.error('Error during model evaluation: %s', e)
raise
def save_metrics(metrics: dict, file_path: str) -> None:
"""Save the evaluation metrics to a JSON file."""
try:
with open(file_path, 'w') as file:
json.dump(metrics, file, indent=4)
logger.debug('Metrics saved to %s', file_path)
except Exception as e:
logger.error('Error occurred while saving the metrics: %s', e)
raise
def save_model_info(run_id: str, model_path: str, file_path: str) ->
None:
"""Save the model run ID and path to a JSON file."""
try:
model_info = {'run_id': run_id, 'model_path': model_path}
with open(file_path, 'w') as file:
json.dump(model_info, file, indent=4)
logger.debug('Model info saved to %s', file_path)
except Exception as e:
logger.error('Error occurred while saving the model info: %s', e)
raise
def main():
mlflow.set_experiment("dvc-pipeline")
with mlflow.start_run() as run:  # Start an MLflow run
```

```
try:
clf = load_model('./models/model.pkl')
test_data = load_data('./data/processed/test_bow.csv')
X_test = test_data.iloc[:, :-1].values
y_test = test_data.iloc[:, -1].values
metrics = evaluate_model(clf, X_test, y_test)
save_metrics(metrics, 'reports/metrics.json')
# Log metrics to MLflow
for metric_name, metric_value in metrics.items():
mlflow.log_metric(metric_name, metric_value)
# Log model parameters to MLflow
if hasattr(clf, 'get_params'):
params = clf.get_params()
for param_name, param_value in params.items():
mlflow.log_param(param_name, param_value)
# Log model to MLflow
mlflow.sklearn.log_model(clf, "model")
# Save model info
save_model_info(run.info.run_id, "model",
'reports/experiment_info.json')
# Log the metrics file to MLflow
mlflow.log_artifact('reports/metrics.json')
# Log the model info file to MLflow
mlflow.log_artifact('reports/model_info.json')
# Log the evaluation errors log file to MLflow
mlflow.log_artifact('model_evaluation_errors.log')
except Exception as e:
logger.error('Failed to complete the model evaluation process: %s',
e)
print(f"Error: {e}")
if __name__ == '__main__':
main()
```

register_model.py

```
# register model
import json
import mlflow
import logging
import os
import dagshub

mlflow.set_tracking_uri("https://dagshub.com/sudeepjoelbayyee/MLOps-Mini-Pro
dagshub.init(repo_owner='sudeepjoelbayyee',
repo_name='MLOps-Mini-Project', mlflow=True)
# logging configuration
logger = logging.getLogger('model_registration')
logger.setLevel('DEBUG')
console_handler = logging.StreamHandler()
console_handler.setLevel('DEBUG')
file_handler = logging.FileHandler('model_registration_errors.log')
file_handler.setLevel('ERROR')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
- %(message)s')
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)
logger.addHandler(console_handler)
logger.addHandler(file_handler)
def load_model_info(file_path: str) -> dict:
"""Load the model info from a JSON file."""
try:
with open(file_path, 'r') as file:
model_info = json.load(file)
logger.debug('Model info loaded from %s', file_path)
return model_info
except FileNotFoundError:
logger.error('File not found: %s', file_path)
raise
except Exception as e:
logger.error('Unexpected error occurred while loading the model info:
%s', e)
raise
def register_model(model_name: str, model_info: dict):
"""Register the model to the MLflow Model Registry."""
try:
model_uri =
f"runs:/{model_info['run_id']}/{model_info['model_path']}"
# Register the model
model_version = mlflow.register_model(model_uri, model_name)
# Transition the model to "Staging" stage
client = mlflow.tracking.MlflowClient()
client.transition_model_version_stage(
name=model_name,
version=model_version.version,
stage="Production"
)
logger.debug(f'Model {model_name} version {model_version.version}
registered and transitioned to Staging.')
except Exception as e:
logger.error('Error during model registration: %s', e)
raise
def main():
try:
model_info_path = 'reports/experiment_info.json'
model_info = load_model_info(model_info_path)
model_name = "my_model"
register_model(model_name, model_info)
except Exception as e:
logger.error('Failed to complete the model registration process: %s',
e)
print(f"Error: {e}")
if __name__ == '__main__':
main()
```

.gitkeep

test_environment.py

```
import sys
REQUIRED_PYTHON = "python3"
def main():
system_major = sys.version_info.major
if REQUIRED_PYTHON == "python":
required_major = 2
elif REQUIRED_PYTHON == "python3":
required_major = 3
else:
raise ValueError("Unrecognized python interpreter: {}".format(
REQUIRED_PYTHON))
if system_major != required_major:
raise TypeError(
"This project requires Python {}. Found: Python {}".format(
required_major, sys.version))
else:
print(">>> Development environment passes all tests!")
if __name__ == '__main__':
main()
```

tox.ini

```
[flake8]
max-line-length = 79
max-complexity = 10
```