Firstly, you will need to install Visual Studio code, which can be found [here](https://code.visualstudio.com/): https://code.visualstudio.com/. Once Visual Studio Code is installed, you will need the Python Extension for Visual Studio Code, which can be found [here](https://marketplace.visualstudio.com/items?itemName=ms-python.python): https://marketplace.visualstudio.com/items?itemName=ms-python.python and also by searching for it in the Extensions Marketplace in VS Code, as shown in Figure 18-1.

You will also need a stable release of Python. For this demo, I have used Python 3.9, which can be found [here](https://www.python.org/downloads/): <https://www.python.org/downloads/.The> initial 64-bit installation UI can be seen in Figure 18-2.

Next, open a new Python terminal in VS Code, as seen in Figure 18-5, and run the following command: py -3 –version to verify the version of Python and confirm that it matches the version that you just installed and set.

Here are the commands shown in Figure 18-6 that you will need to run.

mkdir PythonWheelDemo

cd PythonWheelDemo

code .

The first file that you will need to create is the setup.py file, shown in Figure 18-7. This file will contain all your package metadata information, which is typically similar to the script shown below.

import setuptools

with open("README.md", "r") as fh:

long\_description = fh.read()

setuptools.setup(

name="hive",

version="0.0.1",

author="Ron LEsteve",

author\_email="ronlesteve@ronlesteve.com",

description="Package to create Hive",

long\_description=long\_description,

long\_description\_content\_type="text/markdown",

packages=setuptools.find\_packages(),

classifiers=[

"Programming Language :: Python :: 3",

"License :: OSI Approved :: MIT License",

"Operating System :: OS Independent",

],

python\_requires='>=3.7',

)

The next file that you will need to create is the README.md file shown in Figure 18-7. Here are the typical contents of this file which might contain a description of the package, along with any supporting materials.

# Example Package

This is a simple example package. You can use

[Github-flavored Markdown](https://guides.github.com/features/mastering-markdown/)

to write your content.

You will also need to create a License file, shown in Figure 18-7. This file will contain verbiage as follows and can be customized through the following website: https://choosealicense.com/ It will be important for every package that is uploaded to the Python Package Index to include a license. This tells users who will install your package the terms under which they can use your package.

Copyright (c) 2018 The Python Packaging Authority

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

You will also need an \_\_init\_\_.py file, shown in Figure 18-7. This file provides a mechanism for you to group separate python scripts into a single importable module. For more details on how to properly create an \_\_init\_\_.py file that best suits your needs, read https://timothybramlett.com/How\_to\_create\_a\_Python\_Package\_with\_\_\_init\_\_py.html and <https://towardsdatascience.com/whats-init-for-me-d70a312da583>. The following script is a sample of what is typically included in this file.

from .hive import registerHive

Finally, you will need a python package function file which will contain the python code that will need to be converted to a function. In this demo, we are simply creating a function for a create table statement that can be run in Synapse or Databricks which we will call hive.py, as shown in Figure 18-7. It will accept the database, table. Spark will be used to simply define the spark.sql code section. Here are the contents of this file. You can create a variety of highly customized functions depending on your use case.

def registerHive(spark, database, table, location):

cmd = f"CREATE TABLE IF NOT EXISTS {database}.{table} USING PARQUET LOCATION '{location}'"

spark.sql(cmd)

print(f"Executed: {cmd}")

Begin by running the following command: pip install wheel within the terminal to install the Wheel Package. Figure 18-8 illustrates the messages that are displayed once the command runs successfully

You will also need to run this command: python -m pip install --upgrade pip to update the pip. Figure 18-9 shows the messages that are displayed once the command runs successfully.

You will also need to install a package to check the Wheel contents by running the following command: pip install check-wheel-contents. Check wheel contents will fail and notify you if any of several common errors & mistakes are detected. More details on check wheel contents can be found [here](https://pypi.org/project/check-wheel-contents/): https://pypi.org/project/check-wheel-contents/. Figure 18-10 shows the messages that are displayed once the command runs successfully.

To create the wheel file, run the following command in a python terminal: python setup.py bdist\_wheel. Once the command completes running, notice from Figure 18-11 that it has created the Wheel file and also added hive.py to it.

At this point, you can run the following check wheel contents command: check-wheel-contents C:\PythonWheelDemo\dist\ to verify that that a status of OK is received, which can be seen in Figure 18-12.

More information on uploading wheel files and managing libraries for Apache Spark in Azure Synapse Analytics can be found [here](https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-azure-portal-add-libraries): <https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-azure-portal-add-libraries>.

Here is the code that you will need to run to begin the process of mounting your ADLSgen2 account.

spark.conf.set(

"fs.azure.account.key.adls2001rl.dfs.core.windows.net",

"ENTER-ACCESS-KEY”

)

Next, run the following code to specify the location path containing your AdventureWorksLT2019 database within ADLS gen2. You will need to pre-upload this database to your account. The database can be found here and would need to be converted to parquet format: github.com/microsoft/sql-server-samples/tree/master/samples/databases

dbutils.fs.ls("abfss://data@adls2001rl.dfs.core.windows.net/raw/AdventureWorksLT2019/SalesLT")

Now that you have mounted your data lake folder, create a new AdventureWorksLT2019 Spark database by running the following code, which will generate an OK message once the database is created.

%sql

CREATE DATABASE adventureworkslt2019

Here is the code that you will need to run to re-create the results shown in Figure 18-17.

%sh

/databricks/python/bin/pip freeze

Next, let’s define registerHive() function’s parameters so that we can pass it to the function.

database = "AdventureWorksLT2019"

table = "Customer"

location = f"abfss://data@adls2001rl.dfs.core.windows.net/raw/AdventureWorksLT2019/SalesLT/Customer"

Run the following function from the wheel package rldemopkg.registerHive(spark,database,table,location).

Here is the code that you will need to run to re-create the results shown in Figure 18-19.

%sql

SHOW TABLES FROM AdventureWorksLT2019;