**Data Science**

**Unit 2**

## Chapter 1: Three Management Layers:

### Operational Management Layer:

- It is the core store for the data science ecosystem's complete processing capability.
- The layer stores every processing schedule and workflow for the all-inclusive ecosystem.
- This area enables you to see a singular view of the entire ecosystem.
- It reports the status of the processing.
- Following things are recorded here:
- **1) Processing- Stream Definition and Management:**
- The processing-stream definitions are the building block of the data science ecosystem.
- It stores all current active processing scripts.
- Definition management describes the workflow of the scripts through the system, ensuring that the correct execution order is managed, as per the data scientists' workflow design.
- **2) Parameters:**
- The parameters for the processing are stored, to ensure a single location for all the system parameters.
- Two main designs are used:
- 1. A simple text file that we then import into every Python script
- 2. A parameter database supported by a standard parameter setup script that we then include into every script.
- Example:
- if sys.platform == 'linux':
- Base=os.path.expanduser('~') + '/VKHCG'
- else:
- Base='C:/VKHCG'
- **3) Scheduling:**
- The scheduling plan is stored, to enable central control and visibility of the complete scheduling plan for the system.
- A Drum-Buffer-Rope methodology.
- It is a standard practice to identify the slowest process and then use this process to pace the complete pipeline.
- You then tie the rest of the pipeline to this process to control the eco-system's speed.
- We can use an independent Python program that employs the directed acyclic graph (DAG) that is provided by the network libraries' DiGraph structure.
- This automatically resolves duplicate dependencies and enables the use of a topological sort, which ensures that tasks are completed in the order of requirement.
-

- **4)Monitoring:**
- The central monitoring process is to ensure that there is a single view of the complete system.
- Always ensure that you monitor your data science from a single point.
- Example: (on windows system)
- conda install -c primer wmi
- import wmi
- c = wmi.WMI ()
- for process in c.Win32_Process ():
- print (process.ProcessId, process.Name)
- This will give you a full list of all running processes.
- **5)Communication:**
- All communication from the system is handled, to ensure that the system can communicate any activities that are happening.
- We are using a complex communication process via Jira, to ensure we have all our data science tracked.
- **6) Alerting:**
- The alerting section uses communications to inform the correct person, at the correct time, about the correct status of the complete system.
- We can use Jira for alerting, and it works well.
- If any issue is raised, alerting provides complete details of what the status was and the errors it generated.

- **Audit, Balance, and Control Layer:**
- This layer is the engine that ensures that each processing request is completed by the ecosystem as planned.
- It records:
- • Process-execution statistics
- • Balancing and controls
- • Rejects- and error-handling
- • Fault codes management
- The three subareas are utilized in following manner:
- **1) Audit:**
- An audit is a systematic and independent examination of the ecosystem.
- The audit sublayer records the processes that are running at any specific point within the environment.
- This information is used by data scientists and engineers to understand and plan future improvements to the processing.
- The audit consists of a series of observers that record preapproved processing indicators regarding the ecosystem.
- The following are good indicators for audit purposes:
- **A)Built-in Logging:**
- Design your logging into an organized preapproved location, to ensure that you capture every relevant log entry.
- Do not change the internal or built-in logging process of any of the data science tools, as this will make any future upgrades complex and costly.

- Deploy five independent watchers for each logging location, as logging usually has the following five layers:
- **1. Debug Watcher:**
- This is the maximum verbose logging level.
- Uses a precise processing cycles to perform low-level debugging.
- **2. Information Watcher:**
- The information level is normally utilized to output information that is beneficial to the running and management of a system.
- **3. Warning Watcher:**
- Warning is often used for handled "exceptions" or other important log events.
- Usually this means that the tool handled the issue and took corrective action for recovery.
- **4. Error Watcher:**
- Error is used to log all unhandled exceptions in the tool.
- This is not a good state for the overall processing to be in, as it means that a specific step in the planned processing did not complete as expected.
- Now, the ecosystem must handle the issue and take corrective action for recovery.
- **5. Fatal Watcher:**
- Fatal is reserved for special exceptions/conditions for which it is imperative that you quickly identify these events.
-
- **B)Process Tracking:**
- Build a controlled systematic and independent examination of the process for the hardware logging.
- There is numerous server-based software that monitors temperature sensors, voltage, fan speeds, and load and clock speeds of a computer system.
- **C) Data Provenance:**
- Keep records for every data entity in the data lake, by tracking it through all the transformations in the system.
- This ensures that you can reproduce the data, if needed, in the future and supplies a detailed history of the data's source in the system.
- **D) Data Lineage:**
- Keep records of every change that happens to the individual data values in the data lake.
- This enables you to know what the exact value of any data record was in the past.
- It is normally achieved by a valid-from and valid-to audit entry for each data set in the data science environment.
-
- **2) Balance:**
- The balance sublayer ensures that the ecosystem is balanced across the accessible processing capability or has the capability to top up capability during periods of extreme processing.
- The processing on-demand capability of a cloud ecosystem is highly desirable for this purpose.

- **3) Control:**
- The control sublayer controls the execution of the current active data science.
- The control elements are a combination of the control element within the Data Science Technology Stack's individual tools plus a custom interface to control the overarching work.
- The control sublayer also ensures that when processing experiences an error, it can try a recovery, as per your requirements, or schedule a clean-up utility to undo the error.

- **Yoke Solution:**
- The yoke solution is a custom design.
- Apache Kafka is an open source stream processing platform developed to deliver a unified, high-throughput, low-latency platform for handling real-time data feeds.
- Kafka provides a publish-subscribe solution that can handle all activity-stream data and processing.
- The Kafka environment enables you to send messages between producers and consumers that enable you to transfer control between different parts of your ecosystem while ensuring a stable process.
- **Producer:**
- The producer is the part of the system that generates the requests for data science processing, by creating structures messages for each type of data science process it requires.
- The producer is the end point of the pipeline that loads messages into Kafka.
- **Consumer:**
- The consumer is the part of the process that takes in messages and organizes them for processing by the data science tools.
- The consumer is the end point of the pipeline that offloads the messages from Kafka.

- **Directed Acyclic Graph Scheduling**
- This solution uses a combination of graph theory and publish-subscribe stream data processing to enable scheduling.
- You can use the Python NetworkX library to resolve any conflicts, by simply formulating the graph into a specific point before or after you send or receive messages via Kafka.
- That way, you ensure an effective and an efficient processing pipeline.

- **Cause-and-Effect Analysis System:**
- The cause-and-effect analysis system is the part of the ecosystem that collects all the logs, schedules, and other ecosystem-related information and enables data scientists to evaluate the quality of their system.

- **Functional Layer:**
- The functional layer of the data science ecosystem is the largest and most essential layer for programming and modelling.

- Any data science project must have processing elements in this layer.
- The layer performs all the data processing chains for the practical data science.

- **Data Science Process:**
- Following are the five fundamental data science process steps that are the core of approach to practical data science.
- **1) Start with a What-if Question:**
- Decide what you want to know, even if it is only the subset of the data lake you want to use for your data science, which is a good start.
- "What if I know what car my customer Bob will buy next?"
- **2) Take a Guess at a Potential Pattern**
- Use your experience or insights to guess a pattern you want to discover, to uncover additional insights from the data you already have.
- I guess Bob will buy a car every three years.
- **3) Gather Observations and Use Them to Produce a Hypothesis:**
- Hypothesis is a proposed explanation, prepared on the basis of limited evidence, as a starting point for further investigation.
- "I saw Bob looking at cars last weekend in his Audi" then becomes "Bob will buy an Audi next, as his normal three-year buying cycle is approaching."
- **4) Use Real-World Evidence to Verify the Hypothesis:**
- Now, we verify our hypothesis with real-world evidence.
- On our CCTV, I can see that Bob is looking only at Audis and returned to view a yellow Audi R8 five times over last two weeks.
- On the sales ledger, I see that Bob bought an Audi both three years previous and six previous.
- Bob's buying pattern, then, is every three years.
- So, our hypothesis is verified.
- Bob wants to buy my yellow Audi R8.
- **5) Collaborate Promptly and Regularly with Customers and Subject Matter Experts As You Gain Insights**
- The moment I discover Bob's intentions, I contact the salesperson, and we successfully sell Bob the yellow Audi R8.

- The functional layer is the part of the ecosystem that runs the comprehensive data science ecosystem.
- It consists of several structures, as follows:
- • Data schemas and data formats:
- Functional data schemas and data formats deploy onto the data lake's raw data, to perform the required schema-on-query via the functional layer.
- • Data models:
- These form the basis for future processing to enhance the processing capabilities of the data lake, by storing already processed data sources for future use by other processes against the data lake.
- • Processing algorithms:
- The functional processing is performed via a series of well-designed algorithms across the processing chain.

- • Provisioning of infrastructure:
- The functional infrastructure provision enables the framework to add processing capability to the ecosystem, using technology such as Apache Mesos, which enables the dynamic previsioning of processing work cells.

- The processing algorithms and data models are spread across six supersteps for processing the data lake.
- 1. Retrieve
- • This superstep contains all the processing chains for retrieving data from the raw data lake into a more structured format.
- 2. Assess
- • This superstep contains all the processing chains for quality assurance and additional data enhancements.
- 3. Process
- • This superstep contains all the processing chains for building the data vault.
- 4. Transform
- • This superstep contains all the processing chains for building the data warehouse from the core data vault.
- 5. Organize
- • This superstep contains all the processing chains for building the data marts from the core data warehouse.
- 6. Report
- • This superstep contains all the processing chains for building virtualization and reporting of the actionable knowledge

## Chapter 2: Retrieve Superstep

- **Retrieve Superstep :**
- The Retrieve superstep is a practical method for importing completely into the processing ecosystem a data lake consisting of various external data sources.
- The Retrieve superstep is the first contact between your data science and the source systems.

- **Data Lakes:**
- If you think of a datamart as a store of bottled water—cleansed and packaged and structured for easy consumption—the data lake is a large body of water in a more natural state.
- A company’s data lake covers all data that your business is authorized to process, to attain an improved profitability of your business’s core accomplishments.
- The data lake is the complete data world your company interacts with during its business life span.
- In simple terms, if you generate data or consume data to perform your business tasks, that data is in your company’s data lake.

- So, as a lake needs rivers and streams to feed it, the data lake will consume an unavoidable deluge (a large amount ) of data sources from upstream and deliver it to downstream partners.

- **Data Swamps:**
- Any unmanaged data structure is evil.
- Data swamps are simply data lakes that are not managed.
- Following are the four critical steps to avoid a data swamp:
- 1. Start with Concrete Business Questions:
- Simply dumping a horde of data into a data lake, with no tangible purpose in mind, will result in a big business risk.
- Perform a comprehensive analysis of the entire set of data you have and then apply a metadata classification for the data, stating full data lineage for allowing it into the data lake.
- The data lake must be enabled to collect the data required to answer your business questions.
- **2. Data Governance:**
- The role of data governance, data access, and data security does not go away with the volume of data in the data lake.
- It simply collects together into a worse problem, if not managed.
- Spend the time on data governance up front, as recovery is not easy.
- **3. Data Source Catalog:**
- Metadata that link ingested data-to-data sources are a must-have for any data lake.
- Following as general rules for the data you process:
- • Unique data catalog number: I normally use YYYYMMDD/NNNNNN/NNN.
- E.g. 20171230/000000001/001 for data first registered into the metadata registers on December 30, 2017, as data source 1 of data type 1. This is a critical requirement.
- • Short description (keep it under 100 characters): Country codes and country names (Country Codes—ISO 3166)
- • Long description (keep it as complete as possible): Country codes and country names used by VKHC as standard for country entries
- • Contact information for external data source: ISO 3166-1:2013 code lists from www.iso.org/iso-3166-country-codes.html
- • Expected frequency: Irregular (i.e., no fixed frequency, also known as ad hoc). Other options are near-real-time, every 5 seconds, every minute, hourly, daily, weekly, monthly, or yearly.
- • Internal business purpose: Validate country codes and names.
- **4.Business Glossary:**
- The business glossary maps the data-source fields and classifies them into respective lines of business.
- This glossary is a must-have for any good data lake.
- Create a data-mapping registry with the following information:
- • Unique data catalog number: normally use YYYYMMDD/NNNNNN/NNN.

- • Unique data mapping number: I normally use NNNNNNN/NNNNNNNNN. E.g., 0000001/000000001 for field 1 mapped to internal field 1
- • External data source field name: States the field as found in the raw data source
- • External data source field type: Records the full set of the field's data types when loading the data lake
- • Internal data source field name: Records every internal data field name to use once loaded from the data lake
- • Internal data source field type: Records the full set of the field's types to use internally once loaded
- • Timestamp of last verification of the data mapping: normally use YYYYMMDD-HHMMSS-SSS that supports timestamp down to a thousandth of a second.
- **5. Analytical Model Usage:**
- Data tagged in respective analytical models define the profile of the data that requires loading and guides the data scientist to what additional processing is required.
- Example:
- For R, perform the following commands: Start your RStudio editor.
- Execute the following to import a data loader package:
- library(readr)
- **1) Let's load a table named IP_DATA_ALL.csv.**
- Base=getwd()
- FileName=paste0(Base,'/01-Vermeulen/00-RawData/IP_DATA_ALL.csv')
- IP_DATA_ALL <- read_csv(FileName)
- You should have successfully loaded a data set with eight columns.
- Using the spec(IP_DATA_ALL) command should yield the following results:
- cols(
- ID = col_integer(),
- Country = col_character(),
- 'Place Name' = col_character(),
- 'Post Code' = col_character(),
- Latitude = col_double(),
- Longitude = col_double(),
- 'First IP Number' = col_integer(),
- 'Last IP Number' = col_integer()
- )
- **2) Data Field Name Verification**
- library(tibble)
- set_tidy_names(IP_DATA_ALL, syntactic = TRUE, quiet = FALSE)
- You can fix any detected invalid column names by executing
- IP_DATA_ALL_FIX=set_tidy_names(IP_DATA_ALL, syntactic = TRUE, quiet = TRUE)
- **3) Unique Identifier of Each Data Entry:**
- Allocate a unique identifier within the system that is independent of the given file name.
- To add the unique identifier, run the following command:
- IP_DATA_ALL_with_ID=rowid_to_column(IP_DATA_ALL_FIX, var = "RowID")

- Check if you successfully added the unique identifier, as follows:
- View(IP_DATA_ALL_with_ID)
- **4) Data Type of Each Data Column:**
- Determine the best data type for each column, to assist you in completing the business glossary, to ensure that you record the correct import processing rules.
- Use the following R command:
- sapply(IP_DATA_ALL_with_ID, typeof)
- **5) Minimum Value:**
- Determine the minimum value in a specific column.
- min(hist_country$Country)
- **6) Maximum Value:**
- Maximum Value
- Determine the maximum value in a specific column.
- In R, use
- max(hist_country$Country)
- **7) Mean**
- If the column is numeric in nature, determine the average value in a specific column.
- Let's examine latitude.
- In R, use
- sapply(hist_latitude_with_id[,'Latitude'], mean, na.rm=TRUE)
- **8) Median:**
- Determine the value that splits the data set into two parts in a specific column.
- In R, use the following command against the latitude column:
- sapply(hist_latitude_with_id[,'Latitude'], median, na.rm=TRUE)
- **9) Mode:**
- Determine the value that appears most in a specific column.
- In R, use the following command against the country column:
- IP_DATA_COUNTRY_FREQ=data.table(with(IP_DATA_ALL_with_ID, table(Country)))
- setorder(IP_DATA_COUNTRY_FREQ,-N)
- IP_DATA_COUNTRY_FREQ[1,'Country']
- **10) Range:**
- For numeric values, you determine the range of the values by taking the maximum value and subtracting the minimum value.
- In R, use the following command against the latitude column:
- sapply(hist_latitude_with_id[,'Latitude'], range, na.rm=TRUE)
- **11) Quartiles:**
- Quartiles are the base values dividing a data set into quarters.
- Simply sort the data column and split it into four groups that are of four equal parts.
- In R, use the following command against the latitude column:
- sapply(hist_latitude_with_id[,'Latitude'], quantile, na.rm=TRUE)
- **12) Standard Deviation:**
- In R, use the following command against the latitude column:
- sapply(hist_latitude_with_id[,'Latitude'], sd, na.rm=TRUE)

- **13) Skewness:**
- Skewness describes the shape or profile of the distribution of the data in the column.
- library(e1071)
- skewness(hist_latitude_with_id$Latitude, na.rm = FALSE, type = 2)
- Negative skew: The mass of the distribution is concentrated on the right, and the distribution is said to be left-skewed.
- Positive skew: The mass of the distribution is concentrated on the left, and the distribution is said to be right-skewed.
- **14) Missing or Unknown Values:**
- Identify if you have missing or unknown values in the data sets.
- In R, use the following command against the country column:
- missing_country=data.table(Country=unique(IP_DATA_ALL_with_ID[is.na(IP_DATA_ALL_with_ID ['Country']) == 1, ]))
- View(missing_country)
- **15) Data Pattern:**
- Replace all alphabet values with an uppercase case A, all numbers with an uppercase N, and replace any spaces with a lowercase letter b and all other unknown characters with a lowercase u.
- As a result, "Good Book 101" becomes "AAAAbAAAAbNNNu."
- **16) Data Quality:**
- Data quality can cause the invalidation of a complete data set, if not dealt with correctly.

- **6) Audit and Version Management:**
- You must always report the following:
- • Who used the process?
- • When was it used?
- • Which version of code was used?

- **Training the Trainer Model :**
- To prevent a data swamp, it is essential that you train your team also.
- Data science is a team effort.
- People, process, and technology are the three cornerstones to ensure that data is curated and protected.
- Technology requires that you invest time to understand it fully.
- A big part of this process is to ensure that business users and data scientists understand the need to start small, have concrete questions in mind, and realize that there is work to do with all data to achieve success.

- **Understanding the Business Dynamics of the Data Lake:**
- Python Retrieve Solution:
- Start your Python editor. Create a text file named Retrieve-IP_DATA_ALL.py in directory ..\VKHCG\01-Vermeulen\01-Retrieve.
- import os
- import pandas as pd

- if sys.platform == 'linux':
- Base=os.path.expanduser('~') + '/VKHCG'
- else:
- Base='C:/VKHCG'
- sFileName=Base + '/01-Vermeulen/00-RawData/IP_DATA_ALL.csv'
- print('Loading :',sFileName)
- IP_DATA_ALL=pd.read_csv(sFileName,header=0,low_memory=False)
- sFileDir=Base + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python'
- if not os.path.exists(sFileDir):
- os.makedirs(sFileDir)
- print('Rows:', IP_DATA_ALL.shape[0])
- print('Columns:', IP_DATA_ALL.shape[1])
- print('### Raw Data Set for i in range(0,len(IP_DATA_ALL.columns)):
- print(IP_DATA_ALL.columns[i],type(IP_DATA_ALL.columns[i]))
- print('### Fixed Data Set IP_DATA_ALL_FIX=IP_DATA_ALL
- for i in range(0,len(IP_DATA_ALL.columns)):
- cNameOld=IP_DATA_ALL_FIX.columns[i] + ' '
- cNameNew=cNameOld.strip().replace(" ", ".")
- IP_DATA_ALL_FIX.columns.values[i] = cNameNew
- print(IP_DATA_ALL.columns[i],type(IP_DATA_ALL.columns[i]))
- print('Fixed Data Set with ID')
- IP_DATA_ALL_with_ID=IP_DATA_ALL_FIX
- IP_DATA_ALL_with_ID.index.names = ['RowID']
- #print(IP_DATA_ALL_with_ID.head())
- sFileName2=sFileDir + '/Retrieve_IP_DATA.csv'
- IP_DATA_ALL_with_ID.to_csv(sFileName2, index = True)

- **Actionable Business Knowledge from Data Lakes:**
- Engineering a Practical Retrieve Superstep:
- The means are as follows:
- • Identify the data sources required.
- • Identify source data format (CSV, XML, JSON, or database).
- • Data profile the data distribution (Skew, Histogram, Min, Max).
- • Identify any loading characteristics (Columns Names, Data Types, Volumes).
- • Determine the delivery format (CSV, XML, JSON, or database).

- **Connecting to Other Data Sources:**
- **1) SQLite:**
- This requires a package named sqlite3.
- import sqlite3 as sq
- import pandas as pd
- Base='C:/VKHCG'
- sDatabaseName=Base + '/01-Vermeulen/00-RawData/SQLite/vermeulen.db'
- conn = sq.connect(sDatabaseName)
- sFileName=Base + '/01-Vermeulen/00-RawData/IP_DATA_ALL.csv'
- print('Loading :',sFileName)

- IP_DATA_ALL=pd.read_csv(sFileName,header=0,low_memory=False)
- IP_DATA_ALL.index.names = ['RowIDCSV']
- sTable='IP_DATA_ALL'
- print('Storing :',sDatabaseName,' Table:',sTable)
- IP_DATA_ALL.to_sql(sTable, conn, if_exists="replace")
- print('Loading :',sDatabaseName,' Table:',sTable)
- TestData=pd.read_sql_query("select * from IP_DATA_ALL;", conn)
- print(TestData)
- print('Rows :',TestData.shape[0])
- print('Columns :',TestData.shape[1])
-
- **Other Databases:**
- **PostgreSQL:**
- PostgreSQL is used in numerous companies, and it enables connections to the database.
- There are two options: a direct connection using from sqlalchemy import create_engine
- engine = create_engine('postgresql://scott:tiger@localhost:5432/vermeulen')
- and connection via the psycopg2 interface
- from sqlalchemy import create_engine
- engine = create_engine('postgresql+psycopg2://scott:tiger@localhost/mydatabase')
- **Microsoft SQL Server:**
- Microsoft SQL server is common in companies, and this connector supports your connection to the database.
- There are two options. Via the ODBC connection interface, use
- from sqlalchemy import create_engine
- engine = create_engine('mssql+pyodbc://scott:tiger@mydsnvermeulen')
- Via the direct connection, use
- from sqlalchemy import create_engine
- engine = create_engine('mssql+pymssql://scott:tiger@hostname:port/vermeulen')
- **MySQL**
- MySQL is widely used by lots of companies for storing data.
- This opens that data to your data science with the change of a simple connection string.
- There are two options. For direct connect to the database, use
- from sqlalchemy import create_engine
- engine = create_engine('mysql+mysqldb://scott:tiger@localhost/vermeulen')
- For connection via the DSN service, use
- from sqlalchemy import create_engine
- engine = create_engine('mssql+pyodbc://mydsnvermeulen')
- **Oracle**
- Oracle is a common database storage option in bigger companies.
- It enables you to load data from the following data source with ease:
- from sqlalchemy import create_engine
- engine = create_engine('oracle://andre:vermeulen@127.0.0.1:1521/vermeulen')

- **Microsoft Excel**
- Excel is common in the data sharing ecosystem, and it enables you to load files using this format with ease.
- import os
- import pandas as pd
- Base='C:/VKHCG'
- sFileDir=Base + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python'
- if not os.path.exists(sFileDir):
- os.makedirs(sFileDir)
- CurrencyRawData = pd.
- read_excel('C:/VKHCG/01-Vermeulen/00-RawData/Country_Currency.xlsx')
- sColumns = ['Country or territory', 'Currency', 'ISO-4217']
- CurrencyData = CurrencyRawData[sColumns]
- CurrencyData.rename(columns={'Country or territory': 'Country', 'ISO-4217':
- 'CurrencyCode'}, inplace=True)
- CurrencyData.dropna(subset=['Currency'],inplace=True)
- CurrencyData['Country'] = CurrencyData['Country'].map(lambda x: x.strip())
- CurrencyData['Currency'] = CurrencyData['Currency'].map(lambda x:x.strip())
- CurrencyData['CurrencyCode'] = CurrencyData['CurrencyCode'].map(lambda x:x.strip())
- print(CurrencyData)
- sFileName=sFileDir + '/Retrieve-Country-Currency.csv'
- CurrencyData.to_csv(sFileName, index = False)
- **Apache Spark**
- Apache Spark is now becoming the next standard for distributed data processing.
- import pyspark
- sc = pyspark.SparkContext()
- sql = SQLContext(sc)
- df = (sql.read
- .format("IP_DATA_ALL.csv")
- .option("header", "true")
- .load("/path/to_csv.csv"))
- **Apache Cassandra**
- Cassandra is becoming a widely distributed database engine in the corporate world.
- The advantage of this technology and ecosystem is that they can scale massively in a great scaling manner.
- This database can include and exclude nodes into the database ecosystem, without disruption to the data processing.
- To access it, use the Python package cassandra.
- from cassandra.cluster import Cluster
- cluster = Cluster()
- session = cluster.connect('vermeulen')
- **Apache Hive**
- Access to Hive opens its highly distributed ecosystem for use by data scientists.
- Import pyhs2

- with pyhs2.connect(host='localhost', port=10000, authMechanism="PLAIN",
- user='andre', password='vermeulen', database='vermeulen') as conn:
- with conn.cursor() as cur:
- #Execute query
- cur.execute("select * from IP_DATA_ALL")
- for i in cur.fetch():
- print (i)
- This enables you to read data from a data lake stored in Hive on top of Hadoop.
- **Amazon S3 Storage**
- S3, or Amazon Simple Storage Service (Amazon S3), creates simple and practical methods to collect, store, and analyze data, irrespective of format, completely at massive scale.
- **Package Boto:**
- from boto.s3.connection import S3Connection
- conn = S3Connection('<aws access key>', '<aws secret key>')
- **Amazon Redshift:**
- Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud.
- The Python package redshift-sqlalchemy, is an Amazon Redshift dialect for sqlalchemy that opens this data source to your data science.
- from sqlalchemy import create_engine
- engine = create_engine("redshift+psycopg2://username@host.amazonaws.
- com:5439/database"
- **Amazon Web Services:**
- It is an Amazon Web Services Library Python package that provides interfaces to Amazon Web Services. For example, the following Python code will create an EC2 cloud data processing ecosystem:
- import boto3
- ec2 = boto3.resource('ec2')
- instance = ec2.create_instances(ImageId='ami-1e299d7e',
- MinCount=1,MaxCount=1,InstanceType='t2.micro')
- print instance[0].id
- Once that is done, you can download a file from S3's data storage.
- import boto3
- import botocore
- BUCKET_NAME = 'vermeulen' # replace with your bucket name
- KEY = ' Key_IP_DATA_ALL'# replace with your object key
- s3 = boto3.resource('s3')
- try:
- s3.Bucket(BUCKET_NAME).download_file(KEY, 'IP_DATA_ALL.csv')
- except botocore.exceptions.ClientError as e:
- if e.response['Error']['Code'] == "404":
- print("The object does not exist.")
- else:
- raise