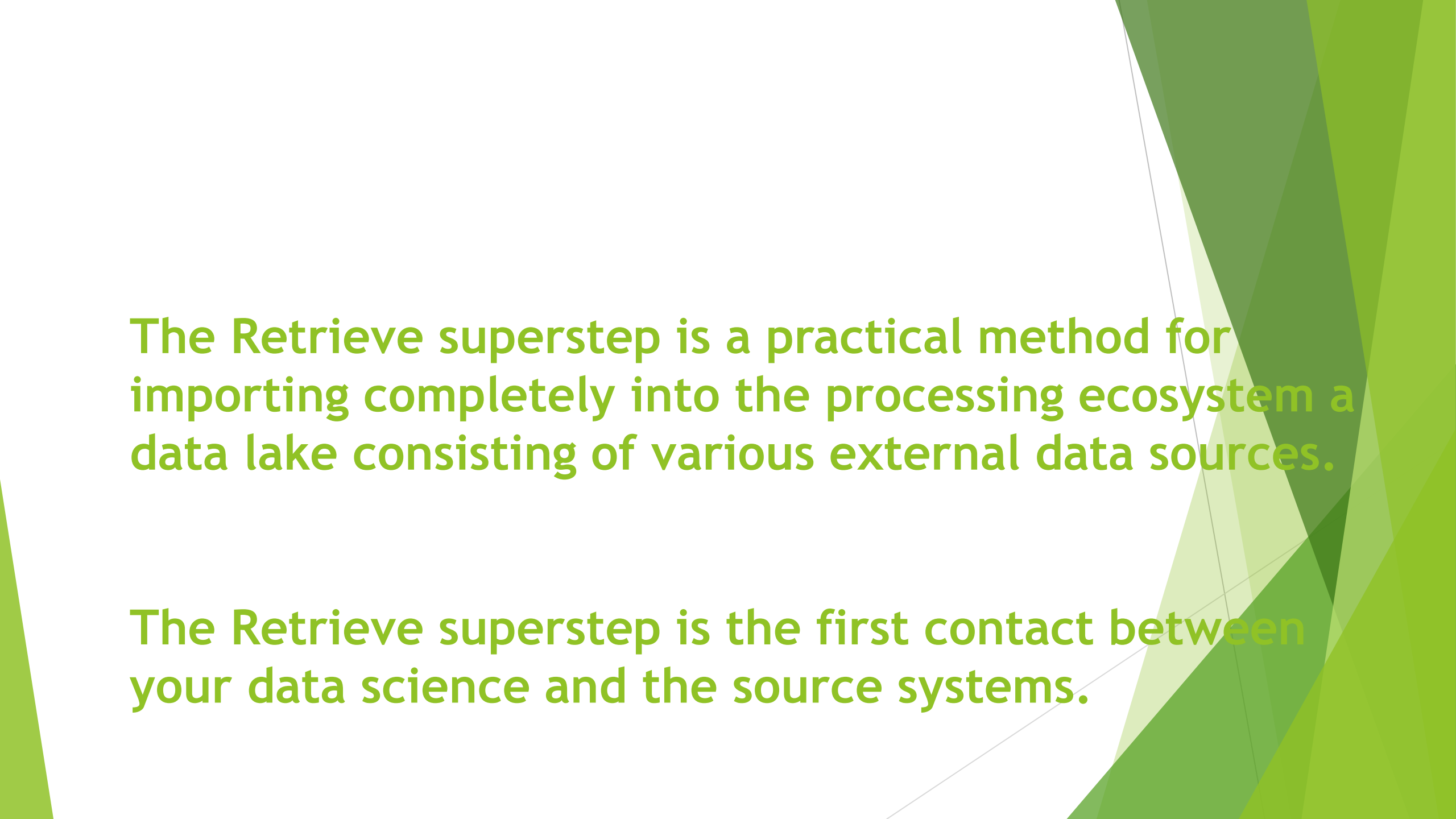


# UNIT-2 PART-2

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

**RETRIEVE SUPERSTEP**

The background of the slide features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

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

- ▶ 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.

# DATA SWAMPS

- ▶ A data swamp is a data lake containing unstructured, ungoverned data that has gotten out of hand.
- ▶ A data swamp is usually the result of a lack of processes and standards.
- ▶ Data in a data swamp is difficult to find, manipulate, and—invariably—analyze.

# Data Lakes vs Data Swamps



## Data Lake

- Proper data organisation, metadata management and data governance in place.
- Data is scalable and easy to find.



## Data Swamp

- Lack of data organisation, metadata management and data governance.
- Data is irrelevant, outdated and/or faulty

# STEPS TO AVOID A DATA SWAMP

- ▶ Start with Concrete Business Questions
- ▶ Data Quality
- ▶ Audit and Version Management
- ▶ Data Governance

# 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.
- ▶ The data lake must be enabled to collect the data required to answer your business questions.



**It is suggested to 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.**

# DATA QUALITY

- ▶ Bad quality data will damage even the best designed data science model. So, spend the time to ensure that you understand any data issues early in the process.
- ▶ Data quality can cause the invalidation of a complete data set, if not dealt with correctly.

# AUDIT AND VERSION MANAGEMENT

You must always report the following:

- ▶ Who used the process?
- ▶ When was it used?
- ▶ Which version of code was used?

# 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.
- ▶ Finding a piece of unmanaged data in a petabyte, or even terabyte, data lake will cost you hours of time.

# WAYS TO IMPLEMENT DATA GOVERNANCE

- ▶ DATA SOURCE CATALOG
- ▶ BUSINESS GLOSSARY
- ▶ ANALYTICAL MODEL USAGE

# DATA SOURCE CATALOG

- ▶ Metadata that link ingested data-to-data sources are a must-have for any data lake.
- ▶ Data processing should include the following rules:
- ▶ **Unique data catalog number**
- ▶ 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.

# DATA SOURCE CATALOG

- ▶ **Short description (It should be under 100 characters)**
  - ▶ Country codes and country names
  - ▶ Ex. ISO 3166 defines Country Codes as per United Nations Sources
- 
- ▶ **Long description (It should kept as complete as possible)**
  - ▶ ▪ Country codes and country names used by your organization as
  - ▶ standard for country entries

# DATA SOURCE CATALOG

- ▶ **Contact information for external data source**
- ▶ ISO 3166-1:2013 code lists from [www.iso.org/iso-3166-country-codes.html](http://www.iso.org/iso-3166-country-codes.html)
- ▶ codes.html



# DATA SOURCE CATALOG

- ▶ **Expected frequency**

- ▶ Irregular i.e., no fixed frequency, also known as ad hoc, every minute, hourly, daily, weekly, monthly, or yearly.

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.

# BUSINESS GLOSSARY

- ▶ The business glossary records the data sources ready for the retrieve processing to load the data.
- ▶ 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.

- **Unique data catalog number:** use YYYYMMDD/NNNNNN/NNN.
- **Unique data mapping number:** 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:** use YYYYMMDD-HHMMSS-SSS that supports timestamp down to a thousandth of a second.

# 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.

# DATA ANALYTICAL MODELS

- Data Field Name Verification
- Unique Identifier of Each Data Entry
- Data Type of Each Data Column
- Histograms of Each Column
- Minimum Value
- Maximum Value
- Mean
- Median
- Mode
- Range
- Quartiles
- Standard Deviation
- Skewness
- Missing or Unknown Values
- Data Pattern

# DATAFIELD NAME VERIFICATION

- ▶ This is used to validate and verify the data field's names in the retrieve processing in an easy manner.
- ▶ Reveals field names that are not easy to use

# DATAFIELD NAME VERIFICATION

- ▶ To add an extra library, you will have to run the following commands:
- ▶ `library(tibble)`
- ▶ `set_tidy_names(INPUT_DATA, 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)`

# DATAFIELD NAME VERIFICATION

- By using command `View(IP_DATA_ALL_FIX)`, you can check that you have fixed the columns.



# UNIQUE IDENTIFIER OF EACH DATA ENTRY

- ▶ Allocate a unique identifier within the system that is independent of the given file name.
- ▶ This ensures that the system can handle different files from different paths and keep track of all data entries in an effective manner.
- ▶ Then allocate a unique identifier for each record or data element in the files that are retrieved

# UNIQUE IDENTIFIER OF EACH DATA ENTRY

- ▶ 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)`
- ▶ This will show you that every record has a unique ID named “Row ID.”

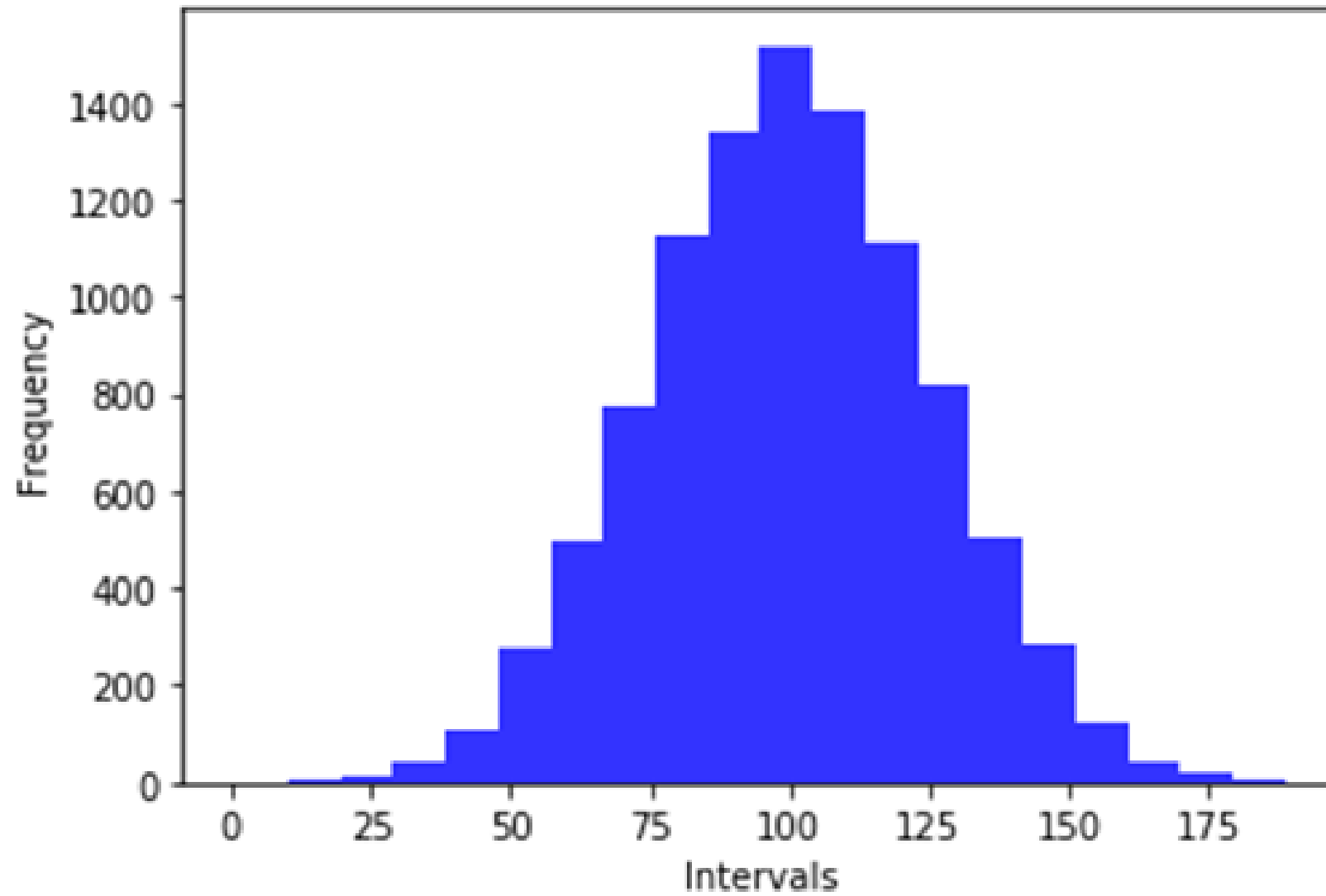
# DATA TYPE OF EACH DATA COLUMN

- ▶ 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)`

# HISTOGRAMS OF EACH COLUMN

- ▶ Always generate a histogram across every column, to determine the
- ▶ spread of the data value.
- ▶ Example: to compute histogram

10000 random numbers, 20 intervals, mean=100, *stddev* = 25



# HISTOGRAMS OF EACH COLUMN

```
library(data.table)

hist_country=data.table(Country=unique(IP_DATA_ALL_with_ID[is.na(IP_DATA_
ALL_with_ID ['Country']) == 0, ]$Country))
setorder(hist_country,'Country')
hist_country_with_id=rowid_to_column(hist_country, var = "RowIDCountry")
View(hist_country_with_id)

IP_DATA_COUNTRY_FREQ=data.table(with(IP_DATA_ALL_with_ID, table(Country)))
View(IP_DATA_COUNTRY_FREQ)
```

Top-Two Country Codes	Frequency
US	512714
GB	127069

# HISTOGRAMS OF EACH COLUMN

► Business Insights:

- The two biggest subset volumes are from the US and GB.
- The US has just over four times the data as GB

# MINIMUM VALUE

- ▶ Determine the minimum value in a specific column.
- ▶ Example: find minimum value

```
min(country_histogram$Country)
```

or

```
sapply(country_histogram[, 'Country'], min, na.rm=TRUE)
```



# MAXIMUM VALUE

- ▶ Determine the maximum value in a specific column.
- ▶ Example: find maximum value

```
max(country_histogram$Country)
```

or

```
sapply(country_histogram[, 'Country'], max, na.rm=TRUE)
```

# MEAN

- ▶ If the column is numeric in nature, determine the average value in a specific column.
- ▶ Example: find mean of latitude
- ▶ `sapply(lattitue_histogram_with_id[, 'Latitude'], mean, na.rm=TRUE)`

# MEDIAN

- Determine the value that splits the data set into two parts in a specific column.

Example: find median of latitude

```
sapply(lattitue_histogram_with_id[, 'Latitude'], median, na.rm=TRUE)
```

# MODE

- ▶ Determine the value that appears most in a specific column.

Example: Find mode for column country

- ▶ `INPUT_DATA_COUNTRY_FREQ =`
- ▶ `data.table(with(INPUT_DATA_with_ID, table(Country)))`

# RANGE

- ▶ For numeric values, you determine the range of the values by taking the maximum value and subtracting the minimum value.
- ▶ Example: find range of latitude
- ▶ `sapply(lattitue_histogram_with_id[, 'Latitude'], range, na.rm=TRUE)`

# QUARTILES

- ▶ These are the base values that divide a data set in quarters. This is done by sorting the data column first and then splitting it in groups of four equal parts.
- ▶ Example: find quartile of latitude
- ▶ `sapply(lattitue_histogram_with_id[, 'Latitude'], quartile,`
- ▶ `na.rm=TRUE)`

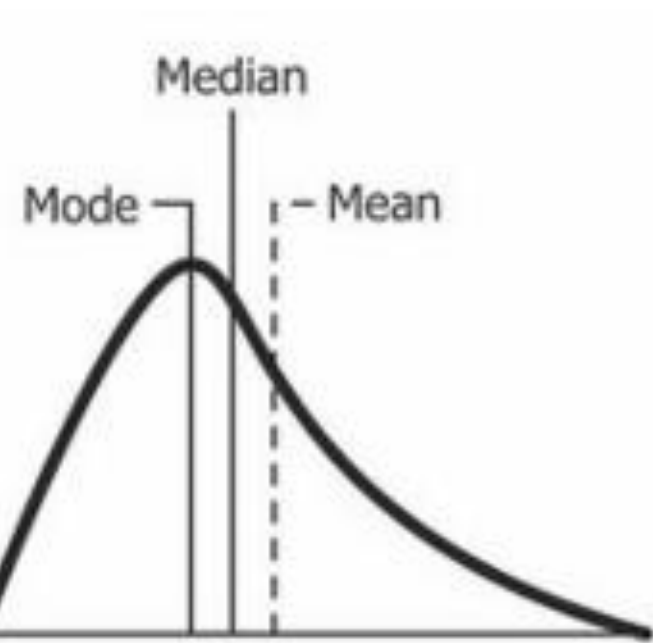
# STANDARD DEVIATION

- ▶ the standard deviation is a measure of the amount of variation or dispersion of a set of values.
- ▶ Example: find standard deviation of latitude
- ▶ `sapply(lattitue_histogram_with_id[, 'Latitude'], sd, na.rm=TRUE)`

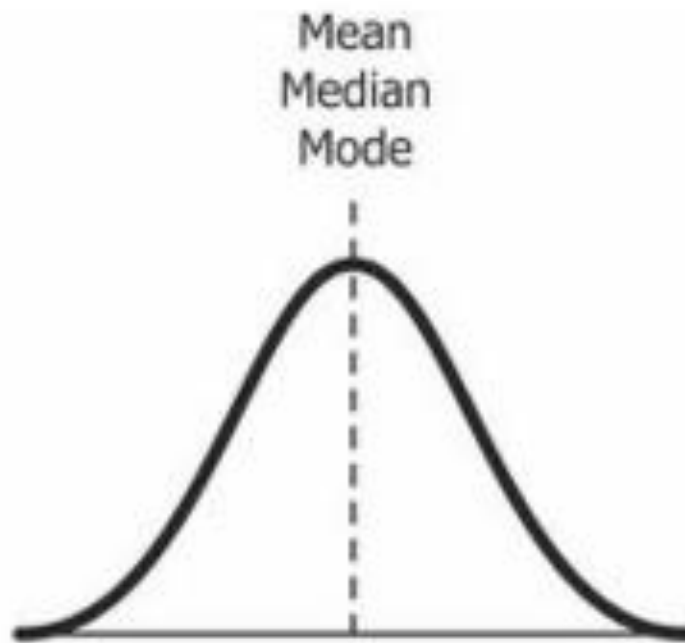
# SKEWNESS

- ▶ Skewness describes the shape or profile of the distribution of the data in the column.
- ▶ Example: find skewness of latitude
- ▶ `library(e1071)`
- ▶ `skewness(lattitue_histogram_with_id$Latitude, na.rm = FALSE,`
- ▶ `type = 2)`

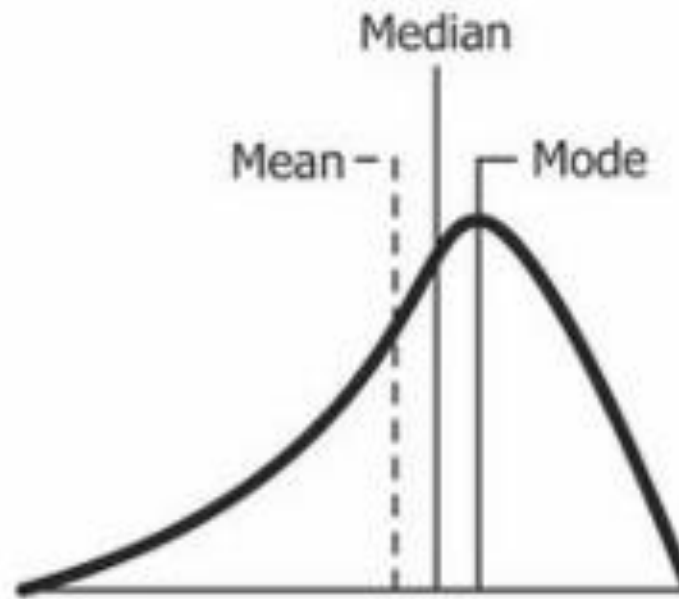




Positive  
Skew



Symmetrical  
Distribution



Negative  
Skew

# MISSING OR UNKNOWN VALUES

- ▶ Identify if you have missing or unknown values in the data sets.
- ▶ Example: find missing value in country column
- ▶ `missing_country=data.table(Country=unique(INPUT_DATA_with_ID[is.na(INPUT_DATA_with_ID ['Country']) == 1, ]))`

# 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 band
- ▶ all other unknown characters with a lowercase u.
- ▶ As a result, “Data Science 102” becomes “AAAAbAAAAAAAbNNNu.”
- ▶ This pattern creation is beneficial for designing any specific assess rules.

# EXAMPLE

- ▶ If you have two patterns—NNNNuNNuNN and uuNNuNNuNN—
- ▶ for a date column, it is easy to see that you have an issue with the uuNNuNNuNN pattern.
- ▶ It is also easy to create a quality matching grid in regular expressions:

# 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.
- ▶ 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

We now have the basic techniques of how to handle a retrieve step in the data science process.

Execute the following command via RStudio:

```
getwd()
```

Test if your workspace directory is correct. You should see the directory you set.

# STEPS FOR THE RETRIEVE SOLUTION

- ▶ Loading the different datasets.
- ▶ Then find the retrieve solution using R or python.
- ▶ Here, we will discuss python retrieve solution.

# LOADING DIFFERENT DATASETS

- ▶ Loading IP\_DATA\_ALL :This data set contains all the IP address allocations in the world. It will help you to locate your customers when interacting with them online.
- ▶ Loading IP\_DATA\_C\_VKHCG This is a specific data load, as it only contains one class C address that VKHCG is using for its data systems.
- ▶ Loading IP\_DATA\_CORE The next data source supplies all the core router's addresses within VKHCG. You will now load IP\_DATA\_CORE.csv into Retrieve\_IP\_DATA\_CORE.csv



# PYTHON RETRIEVE SOLUTION

Start your Python editor. Create a text file named Retrieve-IP\_DATA\_ALL.py in directory ..\VKHCG\01-Vermeulen\01-Retrieve.

Here is the Python code you must copy into the file:

```
#####  
# -*- coding: utf-8 -*-  
#####  
import os  
import pandas as pd  
#####  
if sys.platform == 'linux':  
    Base=os.path.expanduser('~') + '/VKHCG'  
else:  
    Base='C:/VKHCG'
```

UTF-8 is a variable length encoding standard used for electronic communication.

UTF stands for Unicode Transformation Format.

`os.path` module is always the path module suitable for the operating system python is running on, and therefore usable for local paths

`expanduser()` is used for shell like path expansion and `os.path.expanduser()` returns the argument with an initial component of tilde(~) or (~user) replaced by the user's home directory.

```
#####  
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 #####')
```

Low\_memory option is a Boolean parameter that can be set to True or False.

When low\_memory is set to true then panda reads the csv file in chunks which can reduce memory usage but also slow down the reading process.

```
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(IP_DATA_ALL_FIX.head())
#####
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)

#####
print('### Done!! #####')
#####
```

The Boolean values like 1 & 0 can be used as indexes in panda dataframe.

They can help us filter out the required records.

# CONNECTING TO OTHER DATA SOURCES

- Now, we will discuss how to handle non-CSV data sources. The basic data flow and processing stays the same; hence, the data science stays the same. It is only the connection methods that change.



# CONNECTING TO OTHER DATA SOURCES

- ▶ SQLite
- ▶ Date and Time
- ▶ Other Databases
- ▶ PostgreSQL
- ▶ Microsoft SQL server
- ▶ MySQL

# CONNECTING TO OTHER DATA SOURCES

- ▶ Oracle
- ▶ Microsoft Excel
- ▶ Apache Spark
- ▶ Apache Cassandra
- ▶ Apache Hive
- ▶ Apache Hadoop

# CONNECTING TO OTHER DATA SOURCES

- ▶ Amazon S3 storage
- ▶ Amazon Redshift
- ▶ Amazon web services.



# SQLITE

- ▶ **SQLite** is a database engine written in the C programming language.
- ▶ It is a library that software developers embed in their apps. As such, it belongs to the family of embedded databases. It is the most widely deployed database engine, as it is used by several of the top web browsers, operating systems, mobile phones, and other embedded systems.
- ▶ This requires a package named `sqlite3`.



PostgreSQL

# PostgreSQL

- ▶ PostgreSQL is used in numerous companies, and it enables connections to the database.
- ▶ **PostgreSQL** also known as Postgres, is a free and open-source relational database management system (RDBMS) emphasizing extensibility and SQL compliance.
- ▶ `from sqlalchemy import create_engine`
- ▶ `engine =  
create_engine('mssql+pymssql://scott:tiger@hostname:port  
/folder')`

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.



**The Engine is the starting point for any SQLAlchemy application.**

**It's “home base” for the actual database and its DBAPI, delivered to the SQLAlchemy application through a connection pool and a Dialect, which describes how to talk to a specific kind of database/DBAPI combination.**



# Oracle

- ▶ **Oracle Corporation** is an American multinational computer technology company headquartered in Austin, Texas, United States.
- ▶ The company sells database software and technology cloud engineered systems, and enterprise software products, such as enterprise resource planning (ERP) software.

# Oracle

- ▶ Oracle is a common database storage option in bigger companies. It enables you to load data from the following data source with ease:

# Oracle

```
▶ from sqlalchemy import create_engine  
engine  
=create_engine('oracle://andre:vermeulen@127.  
0.0.1:1521/vermeulen')
```

# MySQL

- ▶ MySQL is the world's most popular open source database. According to DB-Engines, MySQL ranks as the second-most-popular database, behind Oracle Database.
- ▶ 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/ve  
rmeulen')
```

# MySQL

- ▶ For connection via the DSN service, use from sqlalchemy import create\_engine
- ▶ engine =  
create\_engine('mssql+pyodbc://mydsnvermeulen')





# Apache Hadoop

- ▶ Hadoop is one of the most successful data lake ecosystems in highly distributed data Science. The pydoop package includes a Python MapReduce and HDFS API for 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 s3 - Python's s3 module connects to Amazon's S3 REST API
- ▶ o Package Boto - The Boto package is another useful tool that connects to Amazon's S3 REST API

aws



# Amazon Web Services

- ▶ Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud, offering over 200 fully featured services from data centers globally. Millions of customers—including the fastest-growing startups, largest enterprises, and leading government agencies—are using AWS to lower costs, become more agile, and innovate faster.
- ▶ The boto3 package is an Amazon Web Services Library Python package that provides interfaces to Amazon Web Services.

# Amazon Web Services

- ▶ **Amazon Web Service**, or AWS, is an online platform providing cost-effective, scalable cloud computing solutions.

# ENGINEERING A PRACTICAL RETRIEVE SUPERSTEP

## Retrieve Super Step

Data Science

# STEPS

- ▶ 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).

JSON (JavaScript Object Notation) is a text-based, human-readable data interchange format used to exchange data between web clients and web servers.

The format defines a set of structuring rules for the representation of structured data. JSON is used as an alternative to Extensible Markup Language (XML).



XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

# Vermeulen PLC

- ▶ The company has two main jobs on which to focus your attention:
- ▶ Designing a routing diagram for company
- ▶ Planning a schedule of jobs to be performed for the router network.

# Designing a Routing Diagram for the Company

- ▶ A network routing diagram is a map of all potential routes through the company's network, like a road map.
- ▶ Create a set of start and end locations, using longitude and latitude.
- ▶ Then implement feature extraction, by calculating the distance between locations in kilometers and miles.

```
#####
```

```
import os
```

```
import pandas as pd
```

```
from math import radians, cos, sin, asin, sqrt
```

```
#####
```

```
def haversine(lon1, lat1, lon2, lat2, stype):
```

```
    """ convert decimal degrees to radians
```

```
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
```

Stype :It is a simple module which is now with seven functions :

"count\_vowels" ,"count\_consonants"

"collect\_vowels","collect\_consonants","count\_symbols","collect\_symbols" "count\_space" and "do\_everything" which will count and collect out the vowels,consonants and symbols/punctuation and spaces in your string.

```
dlat = lat2 - lat1
a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
c = 2 * asin(sqrt(a))
### Type of Distance (Kilometers/Miles)
if stype == 'km':
    r = 6371 # Radius of earth in kilometers
else:
    r = 3956 # Radius of earth in miles
d=round(c * r,3)
return d

#####
Base='C:/VKHCG'
sFileName=Base + '/01-Vermeulen/00-RawData/IP_DATA_CORE.csv'
print('Loading :',sFileName)
IP_DATA_ALL=pd.read_csv(sFileName,header=0,low_memory=False,
    usecols=['Country','Place Name','Latitude','Longitude'])
#####
sFileDir=Base + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python'
if not os.path.exists(sFileDir):
    os.makedirs(sFileDir)
```

```
IP_DATA = IP_DATA_ALL.drop_duplicates(subset=None, keep='first',
inplace=False)
IP_DATA.rename(columns={'Place Name': 'Place_Name'}, inplace=True)
IP_DATA1 = IP_DATA
IP_DATA1.insert(0, 'K', 1)
IP_DATA2 = IP_DATA1

print(IP_DATA1.shape)

IP_CROSS=pd.merge(right=IP_DATA1,left=IP_DATA2,on='K')
IP_CROSS.drop('K', axis=1, inplace=True)
IP_CROSS.rename(columns={'Longitude_x': 'Longitude_from', 'Longitude_y':
'Longitude_to'}, inplace=True)
```

***on : label or list***

Column or index level names to join on. These must be found in both DataFrames. If *on* is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames.



# pandas.DataFrame.shape

*property* DataFrame.**shape**

Return a tuple representing the dimensionality of the DataFrame.

# pandas.DataFrame.drop\_duplicates

```
DataFrame.drop_duplicates(subset=None, *, keep='first', inplace=False, ignore_index=False) \[source\]
```

Return DataFrame with duplicate rows removed.

Considering certain columns is optional. Indexes, including time indexes are ignored.

## Parameters:

**subset** : *column label or sequence of labels, optional*

Only consider certain columns for identifying duplicates, by default use all of the columns.

**keep** : *{'first', 'last', False}, default 'first'*

Determines which duplicates (if any) to keep.

- 'first' : Drop duplicates except for the first occurrence.
- 'last' : Drop duplicates except for the last occurrence.
- False : Drop all duplicates.

**inplace** : *bool, default False*

```
IP_CROSS.rename(columns={'Latitude_x': 'Latitude_from', 'Latitude_y':  
    'Latitude_to'}, inplace=True)  
IP_CROSS.rename(columns={'Place_Name_x': 'Place_Name_from', 'Place_Name_y':  
    'Place_Name_to'}, inplace=True)  
IP_CROSS.rename(columns={'Country_x': 'Country_from', 'Country_y':  
    'Country_to'}, inplace=True)  
  
IP_CROSS['DistanceBetweenKilometers'] = IP_CROSS.apply(lambda row:  
    haversine(  
        row['Longitude_from'],  
        row['Latitude_from'],  
        row['Longitude_to'],  
        row['Latitude_to'],  
        'km')  
    ,axis=1)
```

```
IP_CROSS['DistanceBetweenMiles'] = IP_CROSS.apply(lambda row:
    haversine(
        row['Longitude_from'],
        row['Latitude_from'],
        row['Longitude_to'],
        row['Latitude_to'],
        'miles')
    ,axis=1)
print(IP_CROSS.shape)
sFileName2=sFileDir + '/Retrieve_IP_Routing.csv'
IP_CROSS.to_csv(sFileName2, index = False)

#####
print('### Done!! #####')
#####
```

Now, save the text file.

Pandas.apply allow the users to pass a function and apply it on every single value of the Pandas series.

It comes as a huge improvement for the pandas library as this function helps to segregate data according to the conditions required due to which it is efficiently used in data science and machine learning.

# RETRIEVE OUTPUT

Country_ from	Place_ Name_ from	Latitude_ from	Longitude_ from	Country_ to	Place_ Name_ to	Latitude_ to	Longitude_ to	Distance Between Kilometers	Distance Between Miles
US	New York	40.7528	-73.9725	US	New York	40.7528	-73.9725	0	0
US	New York	40.7528	-73.9725	US	New York	40.7214	-74.0052	4.448	2.762

So, the distance between New York (40.7528, -73.9725) to New York (40.7214, -74.0052) is 4.45 kilometers, or 2.77 miles.

# Building a Diagram for the Scheduling of Jobs

- ▶ You can extract core routers locations to schedule maintenance jobs. Now that we know where the routers are, we must set up a schedule for maintenance jobs that have to be completed every month.
- ▶ To accomplish this, we must retrieve the location of each router, to prepare a schedule for the staff maintaining them.

```
# -*- coding: utf-8 -*-
```

```
#####
```

```
import os
```

```
import pandas as pd
```

```
#####
```

```
InputFileName='IP_DATA_CORE.csv'
```

```
OutputFileName='Retrieve_Router_Location.csv'
```

```
#####
```



```
if sys.platform == 'linux':
    Base=os.path.expanduser('~') + '/VKHCG'
else:
    Base='C:/VKHCG'
#####
sFileName=Base + '/01-Vermeulen/00-RawData/' + InputFileName
print('Loading :',sFileName)
IP_DATA_ALL=pd.read_csv(sFileName,header=0,low_memory=False,
    usecols=['Country','Place Name','Latitude','Longitude'])

IP_DATA_ALL.rename(columns={'Place Name': 'Place_Name'}, inplace=True)
#####
sFileDir=Base + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python'
if not os.path.exists(sFileDir):
    os.makedirs(sFileDir)

ROUTERLOC = IP_DATA_ALL.drop_duplicates(subset=None, keep='first',
inplace=False)

print('Rows :',ROUTERLOC.shape[0])
print('Columns :',ROUTERLOC.shape[1])
```

Pandas provide a unique method to retrieve rows from a Data frame.

**DataFrame.loc[]** method is a method that takes only index labels and returns row or dataframe if the index label exists in the caller data frame.

*Syntax: pandas.DataFrame.loc[]*

*Parameters:*

*Index label: String or list of string of index label of rows*

```
sFileName2=sFileDir + '/' + OutputFileName
```

```
ROUTERLOC.to_csv(sFileName2, index = False)
```

```
#####
```

```
print('### Done!! #####')
```

```
#####
```

# OUTPUT OF FINAL RETRIEVE SUPERSTEP

Open this file, and you should see a data set like the following:

Country	Place_Name	Latitude	Longitude
US	New York	40.7528	-73.9725
US	New York	40.7214	-74.0052
US	New York	40.7662	-73.9862

You have now successfully retrieved the location of 150 routers.

**THANKS**