Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID-19. The COVID-19 resource centre is hosted on Elsevier Connect, the company's public news and information website.

Elsevier hereby grants permission to make all its COVID-19-related research that is available on the COVID-19 resource centre - including this research content - immediately available in PubMed Central and other publicly funded repositories, such as the WHO COVID database with rights for unrestricted research re-use and analyses in any form or by any means with acknowledgement of the original source. These permissions are granted for free by Elsevier for as long as the COVID-19 resource centre remains active.
A data science perspective of real-world COVID-19 databases

Shivika Prasanna¹,², Praveen Rao¹,²,³

¹ DEPARTMENT OF HEALTH MANAGEMENT AND INFORMATICS, UNIVERSITY OF MISSOURI, COLUMBIA, MO, UNITED STATES ² DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, UNIVERSITY OF MISSOURI, COLUMBIA, MO, UNITED STATES ³ CENTER FOR BIOMEDICAL INFORMATICS, UNIVERSITY OF MISSOURI, COLUMBIA, MO, UNITED STATES

Key points

- Publicly available COVID-19 datasets useful for data science activities are discussed.
- An overview of Cerner’s Real-World Data™ is provided along with Spark SQL and Python examples for data extraction and processing.
- Overview and implementation of some popularly used machine learning algorithms are presented.
- Visualization of the data and results using Python is presented.

5.1 Introduction

Since the 1918 Spanish flu, the COVID-19 pandemic is the biggest public health crisis faced by mankind. As of January 2021, more than 91 million people have been infected by COVID-19 and more than 1.9 million people have died worldwide (Johns Hopkins University, 2021). The economy of many countries has been damaged by COVID-19 due to mandatory lockdowns and billions of dollars are being invested by governments to tackle the pandemic. Several research efforts are underway in the areas of rapid testing, vaccine development, drug repurposing, genomics, contact tracing, supply-chain management, and so on. It is a sigh of relief that government-approved vaccines are now being administered in different parts of the world. While the negative impact of the pandemic on mankind is unimaginable, this pandemic has triggered new research and innovation in the use of artificial intelligence (AI) for developing solutions to address the current public health crisis.

In recent years, the interdisciplinary field of data science has emerged as the next frontier for data-driven decision making, innovation, and discovery (Mckinsey, 2016). Broadly, data science deals with the extraction of knowledge or meaningful insights from data using automated methods such as statistical and machine learning (ML) approaches (NYU, 2013; edX, 2006; Wikipedia 2018).
Touted as the sexiest job of the 21st century in 2012 (Harvard Business Review, 2012), data scientists are in huge demand in several domains including retail, healthcare, finance, defense, manufacturing, and public sector (Mckinsey, 2016). With AI and machine learning achieving tremendous commercial success, companies like Amazon, Facebook, Baidu, Google, IBM, Microsoft, and many others have gained significant business advantage by extracting meaningful insights from massive amounts of data (Forbes, 2017). Several universities are offering new educational programs to prepare the next generation of data scientists. Needless to say, data science will provide transformative benefits to many domains including healthcare delivery and public health.

Several clinical and nonclinical studies have emerged to better understand COVID-19 and develop effective treatment strategies (Digala et al., 2021; Ye et al., 2020; Beigel et al., 2020; Ren et al., 2020; Li et al., 2020; Penn Medicine, 2021; FDA, 2020; NIH, 2020). This has resulted in valuable real-world datasets that can be consumed by a data science pipeline to create new applications and solutions. These datasets range from medical imaging datasets to biomedical literature on coronaviruses, to genome sequences of COVID-19 viruses and infected patients. The availability of open-source tools for data science (e.g., Apache Spark, Python) coupled with real-world COVID-19 datasets provide new opportunities for any aspiring data scientist. A typical workflow of a data scientist involves data collection, data storage and retrieval, data wrangling, data analysis, and visualization. However, given the variety and number of different COVID-19 datasets, it can be a nontrivial task for an aspiring data scientist to begin using them.

Motivated by the aforementioned reasons, in this chapter, we first provide an overview and comparison of different publicly available datasets for COVID-19. Next, we delve deep into Cerner’s HealtheLab (Cerner, 2021), a cloud-based portal containing de-identified patient data of those tested for COVID-19 collected from more than 60 contributing health systems in the United States. The associated dataset is called Cerner Real-World Data and does not require institutional review board (IRB) approval to conduct secondary analysis. The main purpose of Cerner HealtheLab and Cerner Real-World Data is to empower clinical researchers with easy access to electronic medical records (EMR) data for retrospective analysis as well as post-market surveillance. The data can be queried, visualized, and analyzed in a cloud-based environment such as Amazon Web Services (AWS).

We assume the reader has a basic knowledge of relational database concepts and machine learning techniques. Therefore in this chapter, we describe the schema of the database, which includes several tables to store patient demographics, patient conditions, medications, lab results, etc. This provides the reader a view of how patient data are organized in an enterprise healthcare setting and how patient data can be extracted by writing database queries. We also discuss the data quality issues that were encountered due to the required de-identification process and incomplete data collection. We provide examples of how data wrangling and querying can be done via Spark SQL queries. After that, we present examples of how machine learning techniques can be applied to the extracted data for classification and regression. We provide a few visualization examples to better understand the data and predictions. All of the examples are based on Python and Apache Spark. We also discuss the technical challenges that were encountered and how we overcame them to complete multiple clinical studies on COVID-19 patients.
The rest of the chapter is organized as follows: Section 5.2 provides an overview and comparison of publicly available COVID-19 data sources. Section 5.3 introduces Cerner Real-World Data followed by the data extraction and wrangling steps, and challenges with real-world data. Section 5.4 discusses different machine learning techniques for classification and regression as well as visualization. Finally, we conclude in Section 5.5.

5.2 Publicly available data sources for COVID-19: an overview

COVID-19 is an infectious disease caused by a new coronavirus that has caused a global pandemic. Various organizations have come together to bundle data from different sources to provide free and publicly accessible data tools on COVID-19. Data play a critical role in research and specifically to combat public health crises. Access to such datasets is vital in providing healthcare researchers the right tools to develop effective responses and mitigation strategies for the pandemic.

COVID-19 datasets have been broadly classified into “Data Type,” “Applications,” “Methods,” and “Repositories” by Shuja et al. (2020). The taxonomy defined by the authors divides different data sources into categories. “Data Type” can be further classified into medical images (e.g., CT scans and X-rays), textual data (e.g., case reports, tweets, scholarly articles), and speech data (e.g., cough- and breath-based diagnoses). “Applications” can be further classified into image-based, visual cases, natural language processing of scholarly articles, and so on. “Methods” can be classified into machine learning, statistical, and big data. Finally, “Repositories” include GitHub and Kaggle.

One of the most popular data tools for COVID-19 is provided by the Centers for Disease Control and Prevention (CDC) (CDC, 2020). CDC provides an extremely user-friendly dashboard that shows case trends, vaccinations, global cases, and deaths, and many more. They also provide a state-by-state count of the number of cases, number of recoveries, and number of deaths. According to CDC, more than 30% of patients hospitalized due to COVID-19 required intensive care and more than 1 out of 10 hospitalizations ended in death. Such insights help researchers understand the trend of how the virus is spreading and its harmful effects on the population. Koonin et al. (2020) have utilized the data provided by CDC to analyze the trends in telehealth—a two-way telecommunication to provide long-distance clinical healthcare and patient education using information and communication technologies. They examined changes in the frequency of use of telehealth services, especially during the period of January to March 2020, where the proportion of COVID-19 related encounters had significantly increased in March 2020. They were also able to calculate the p-values for a majority of the attributes, such as encounters, using the obtained frequencies.

The dataset provided by CDC consists of data collated from four participating, leading US telehealth providers. It contains de-identified information of patients, mainly from January 2020, and includes encounter dates, demographic information such as age, gender, county and state of residence, disposition after a visit, reason for the visit, and ICD-10 diagnosis
codes. Since the data are being used to understand telehealth services trends, the patient encounters were classified as COVID-19-related or not COVID-19-related. The dataset only represents a small portion of the population and the interpretation does not include data from virtual encounters in all the telehealth providers available.

Another popular resource is from Google Cloud that offers a cloud-hosted public repository of public datasets like COVID-19 Open Data, Global Health Data from World Bank, and OpenStreetMap (Jennings et al., 2020) datasets. The purpose of these datasets and tools is to bolster COVID-19 research and provide BigQuery ML to train advanced ML models within BigQuery itself. Researchers can access the datasets via the Google Cloud Console as it reduces the time taken to search for and download large files. The COVID-19 Open Data dataset offers global information such as latitude, longitude, geometric location, confirmed cases, number of deaths, and recoveries. This data can be used to represent location-specific cases on a map (BigQuery Public Datasets Program, 2020).

National Institute of Health (NIH) (NIH, 2020b) offers open-access COVID-19 data and computational resources such as AWS Data Lake for analysis of COVID-19 data, the COVID-19 Open Research Dataset (CORD-19), COVID-19 viral genome sequences, whole slide images of COVID-19 histopathological samples, and many more. All the resources are free, publicly available, and may also have various tools to support the user in faster extraction of the data.

The CORD-19 dataset (CORD-19) is a popular dataset that was created by the Allen Institute of AI, in partnership with leading research groups. Wang et al. (2020) have described the mechanics of the dataset construction and have provided an overview of how the CORD-19 dataset has been used to aid in building effective treatments and management policies for COVID-19. The CORD-19 dataset integrates papers and preprints on COVID-19 and coronaviruses from various sources such as PubMed Central, the bioRxiv, medRxiv preprint servers, and the World Health Organization (WHO) COVID-19 database. A paper is the fundamental unit of published knowledge, and a preprint is an unpublished counterpart of a paper that is not peer-reviewed. The metadata within the dataset contains the title of the paper and abstract among other columns. The dataset also consists of tables that contain patient information such as the severity of the disease, how many patients have been included in the study, factors affecting the study sample, methods used in the study, results of the study, measure of evidence, etc.

One can only imagine the plethora of scientific and clinical questions (on COVID-19) that can be answered using the CORD-19 dataset. As reported by Wang et al. (2020), the dataset can be used to find evidence for answering queries such as “Does hypertension increase the risks associated with COVID-19?” Kricka et al. (2020) have explored the CORD-19 dataset along with AI-powered search tools such as WellAI and contact tracing based on mobile communication to increase the awareness of the importance of these tools for further research.

WHO has gathered the latest developments, responses and findings, and knowledge on COVID-19 to develop a database that is freely accessible (WHO, 2021; PAHO, 2020). The database consists of a search interface that provides a global research database on COVID-19 by reusing the Global Index Medicus platform. The database contains the latest findings and international multilingual scientific knowledge about COVID-19 where the literature is global.
and updated regularly from searches in bibliographic databases, manual searches, and articles referred by experts.

John Hopkins’ Coronavirus Resource Center (Johns Hopkins University, 2021) is an excellent example of how users can understand COVID-19 data trends using intuitive dashboards. A Geomap is used to display the cases in colored dots that increase in size depending on the intensity of number of cases in that area. The dashboards also show the number of cases, fatalities, testing rate, and other metrics, per county, country, region, and sovereignty. These dashboards are useful to data scientists who wish to explore the current trends on how the COVID-19 infection is spreading across the globe.

5.3 Cerner real-world data

The following note is based on Cerner’s License Agreement. “Cerner Real-World Data is extracted from the EMR of hospitals in which Cerner has a data use agreement. Encounters may include pharmacy, clinical and microbiology laboratory, admission, and billing information from affiliated patient care locations. All admissions, medication orders and dispensing, laboratory orders and specimens are date and time stamped, providing a temporal relationship between treatment patterns and clinical information. Cerner Corporation has established Health Insurance Portability and Accountability Act-compliant operating policies to establish de-identification for Cerner Real-World Data.”

In this section, we will introduce Cerner’s HealtheDataLab portal (Cerner, 2021) that was created to provide users access to Cerner’s cloud-hosted COVID-19 data. The data can be accessed through the Jupyter Notebook that supports various languages like PySpark, R, Python. Digala, et al. (2021a,b,c) conducted multiple clinical studies related to neuromuscular diseases using HealtheDataLab. For example, to understand the impact of COVID-19 infection on amyotrophic lateral sclerosis (ALS) patients (Digala et al., 2021a,b,c), they first identified all ALS patients that tested positive for COVID-19, the various comorbidities of these patients, and those whose conditions were aggravated due to COVID-19. It was also possible to extract the hospitalization frequency of these patients that was required for calculating the mean length of hospitalization for the patients. Using this data, they outlined the intensity of the impact of COVID-19 infection in a small number of ALS patients as ALS is a rare disease.

5.3.1 Data extraction

Before we dive into how to work with the database in HealtheDataLab, let us first discuss the different terminologies that are often used in relational database systems. Structured Query Language (SQL) is the language used for writing and querying structured data. Data are stored in a database with a predefined structure as specified by its schema. Data are organized as records; records are stored in tables; each record contains a tuple of values. Data records can be easily accessed, managed, modified, updated, or controlled. Most databases contain multiple tables with different fields or attributes. A popular example of a database is
a “Company database”, where the information is stored in multiple tables, such as employees, inventory, customers, sales, etc.

A table consists of columns, and each column is of a single data type that could be an integer, character string, date or date and time, or any other valid data type. A single record of data is stored as a row in the table. For example, in the “employees” table of the “Company database” with columns or attributes “employee_id”, “first_name”, “last_name”, and “address”, a row would contain a tuple of values for the attributes. One or more columns of a table can be used as a key to identify each record. If the key is unique across all the records in a table, it is called a primary key. A good example of a primary key would be “employee_id”, where each employee is assigned a unique ID in an organization. The other type of key is the foreign key that provide a “link” between records in two tables. A foreign key in a table is a primary key in another table. For example, if the sales table contains products sold by employees, then the attribute “employee_id” in the sales table is a foreign key.

Cerner Real-World Data is hosted on AWS and can be accessed using the Jupyter Notebook. Several languages can be used for data extraction and analysis such as PySpark, Python, and R. We will be showing examples using both PySpark and Python in this chapter. Although we will be using Cerner Real-World Data in subsequent examples, the examples can be adapted to other databases stored using Apache Spark.

Cerner Real-World Data for COVID-19 is stored into multiple tables. The demographics table contains all demographic information of patients such as gender, race, ethnicity, etc. In addition, if a patient was tested for or had exposure to the COVID-19 infection, then the status (alive or deceased) is stored. The encounter table contains the details for encounters that patients had, such as ‘Emergency’, ‘Inpatient’, ‘Admitted for Observation’. The covid_labs table consists of qualitative COVID-19 lab results associated with the encounters. The condition table consists of diagnoses from qualifying or supplemental encounters. Each condition has an associated condition code and code type that can be ICD-9-CM, ICD-10-CM, or SNOMED-CT. Note that ICD-9-CM, ICD-10-CM, and SNOMED-CT are standardized clinical terminologies for healthcare systems. The medication table consists of medication orders associated with the encounters. The procedure table includes the procedures a patient has undergone, where each procedure is associated with a Current Procedural Terminology (CPT) code. Finally, the result table includes result records such as vital signs, lab results, and personal characteristics.

The schema for Cerner Real-World Data is given in Fig. 5–1, where the primary key in demographics has been highlighted in bold, and the foreign keys have been depicted using the arrow lines (the grayed-out lines show the relation between the demographics table and the other tables). It is worth noting that all the tables contain “personid”, which is a unique, de-identified ID for each patient. We utilize this attribute to perform joins or filter information like comorbidities (i.e., other medical/disease conditions), medications, or procedures for a particular patient of interest.

Before we begin writing the queries for data extraction, we must first import the required libraries and packages. Pandas is a software library written for Python and is used to create data structures and operations for manipulating data. A Pandas DataFrame is a
FIGURE 5-1 Schema for Cerner real-world data.
2-dimensional data structure with columns of different data types. We utilize an additional package from the IPython library to display the output in a readable format as shown below.

```python
import pandas as pd
from IPython.display import display

with pd.option_context('display.max_rows', 1000, 'display.max_columns', 10, 'display.max_colwidth', 500):
    display(dataframe_name)
```

As Apache Spark and Spark SQL are used to store and query the tables, these tables are stored internally as Spark DataFrames. In Spark, a DataFrame is a distributed collection of data with named columns. To manipulate the result of a Spark SQL statement such as “find the number of patients with a particular comorbidity,” we need to convert the results into a Pandas DataFrame, which is not distributed. This can be done using toPandas() function. Similarly, show() is used to display all the rows in a Spark DataFrame and printSchema() is used to print the schema of a particular table.

Using a Jupyter Notebook instance and PySpark, we can view all the databases available in Cerner HealtheDataLab using the following query.

```python
spark.sql("SHOW databases").show()
```

To view all the tables in a database, the following statement can be executed.

```python
spark.sql("SHOW tables").toPandas()
```

To view the schema of a table, the following statement can be executed.

```python
spark.sql("""SELECT *
           FROM demographics"""").printSchema()
```

To find the number of patients with a given clinical condition, we must first know the codes and the code type for that condition. We can then form a query to select all the patients having the condition. Consider the condition “Acute respiratory distress syndrome,” which is represented by codes 67782005 (in SNOMED-CT) and J80 (in ICD-10-CM). To find all the unique codes for the condition in the database, a select query is performed on the condition table.

```python
spark.sql("""SELECT DISTINCT(conditioncode), codetype
           FROM condition
           WHERE condition LIKE "%Acute respiratory distress syndrome%"").toPandas()
```
Once the condition codes and the code types are known, a SQL query is executed on the `condition` table to extract all patients that have the given condition.

```sql
spark.sql(""
SELECT DISTINCT(condition.personid), condition.condition
FROM condition
WHERE (condition.conditioncode = "J80" AND condition.codetype = "ICD-10-CM") OR
  (condition.conditioncode = "67782005" AND condition.codetype = "SNOMED-CT") """ ).toPandas()
```

We perform an ‘AND’ operation between `conditioncode` and `codetype` so that only patients that have the given condition are included.

We can further refine our query to find all patients that have the aforementioned condition and have tested positive for the COVID-19 infection. We can do a join operation between the `condition` table and the `covid_labs` table on “personid” and select those where the lab result is “Positive”. Since patient ID is unique and present in all the tables, the join operation helps retrieve all patients for a given condition that tested positive for the COVID-19 infection. A JOIN operation specifies a join between two or more tables by combining rows between the tables, based on a common, related column.

```sql
spark.sql(""
SELECT DISTINCT(condition.personid), condition.condition, covid_labs.result
FROM condition JOIN covid_labs ON condition.personid = covid_labs.personid
WHERE COVID_LABS.result = "Positive" AND
  (condition.conditioncode = "J80" AND condition.codetype = "ICD-10-CM") OR
  (condition.conditioncode = "67782005" AND condition.codetype = "SNOMED-CT") """ ).toPandas()
```

In clinical studies, it is important to know the demographical information of the patients, that is, their “gender,” “race,” and “ethnicity”. This information is generally stored in the `demographics` table and can be queried as follows.

```sql
spark.sql(""
SELECT DISTINCT(condition.personid), demographics.gender, demographics.race, demographics.ethnicity
FROM condition JOIN covid_labs ON condition.personid = covid_labs.personid
JOIN demographics ON demographics.personid = condition.personid
WHERE COVID_LABS.result = "Positive" AND (condition.conditioncode = "J80" AND condition.codetype = "ICD-10-CM") OR
  (condition.conditioncode = "67782005" AND condition.codetype = "SNOMED-CT") """ ).toPandas()
```

The above query will output the gender, race, and ethnicity of each patient. Due to `DISTINCT`, there will be only one output record for each patient. This will change if we do not project “personid” as each patient may undergo multiple COVID-19 tests and test positive multiple times. So, for each “Positive” test result, there will be an output record.
of gender, race, and ethnicity. The results of the above query can be stored in a DataFrame (say df_demographics). We can obtain the count for each of the attributes using Python.

```python
with pd.option_context('display.max_rows', 1000, 'display.max_columns', 10, 'display.max_colwidth', 500):
    display(df_demographics.groupby('gender').count())
```

In the above statements, we are grouping the DataFrame by “gender.” The output will include the patient count for each gender. In Cerner Real-World Data, the gender attribute is assigned either ‘Male’, ‘Female’, or ‘Other’.

A patient’s age is a continuous variable; it will increase with every yearly visit of the patient. This information may be stored in the encounter table so that the age is recorded for every encounter and can be accessed by doing a join between condition, covid_labs, and encounter tables on “personid”. In the following query, demographics table is also included to retrieve all the demographics information.

```sql
spark.sql("SELECT DISTINCT(condition.personid), demographics.gender, demographics.race, demographics.ethnicity, encounter.age_at_encounter FROM condition JOIN covid_labs ON condition.personid = covid_labs.personid JOIN demographics ON demographics.personid = condition.personid JOIN encounter ON encounter.personid = condition.personid WHERE COVID_LABS.result = "Positive" AND (condition.conditioncode = "J80" AND condition.codetype = "ICD-10-CM") OR (condition.conditioncode = "67782005" AND condition.codetype = "SNOMED-CT") ").toPandas()
```

The query above will list all the recorded ages of each patient with the desired condition. One way to retrieve their age statistics (e.g., most recent age) is by using Python and storing the results from the query into a DataFrame and then performing a `groupby` operation on “personid”. (Another way is to extend the query itself by finding the maximum of a patient’s age and grouping by “personid”.) Suppose the result of the above query is stored in a DataFrame called “df”. As shown below, the DataFrame is transformed by a group by operation on “personid”. The function max() is applied on “age_at_encounter” to get the maximum age for each patient after grouping. The results are stored in “df_age”. We can then find the maximum age among all the patients using another max() operation on “df_age”. Similarly, we can obtain minimum age using min(), mean age using mean(), and standard deviation using std().

```python
df_age = df.groupby('personid')['age_at_encounter'].max()
display(df_age.max())
```
Another important attribute that clinicians are interested in for COVID-19 studies is a patient’s length of stay in a hospital. To calculate this attribute, it is important to know the patients’ hospitalization start date, discharge date, admission types as well as their discharge disposition. As we are interested in the patients’ hospitalization timeline after they were infected by COVID-19, we must consider all hospitalization start dates after the first COVID-19 positive test. Assuming this information is stored in the *encounter* table, the query will be as follows.

```sql
spark.sql(""
SELECT DISTINCT(condition.personid), encounter.encountertype, 
encounter.dischargedisposition, covid_labs.servicedate AS COVIDdate, encounter.admissiontype, encounter.dischargedate, 
encounter.servicedate, DATEDIFF(encounter.dischargedate, 
encounter.servicedate) AS LENGTHOFSTAY
FROM condition JOIN covid_labs ON condition.personid = 
covid_labs.personid JOIN encounter ON condition.personid = 
encounter.personid
WHERE covid_labs.result = "Positive" AND 
(encounter.servicedate >= covid_labs.servicedate AND 
encounter.dischargedate >= covid_labs.servicedate) AND 
(condition.conditioncode = "J80" AND 
condition.codetype = "ICD-10-CM") AND 
covid_labs.servicedate IN 
(SELECT covid_labs.servicedate 
FROM condition JOIN covid_labs ON condition.personid = 
covid_labs.personid 
WHERE covid_labs.result = "Positive" AND 
(condition.conditioncode = "J80" AND 
condition.codetype = "ICD-10-CM")
"").toPandas()
```

In the above query, there is a subquery that retrieves the “servicedate” attribute from *covid_labs* table containing the dates for all positive COVID-19 test results of the patients with the desired condition. Using “servicedate”, the query checks that the COVID-19 test was before the admission date denoted by the “servicedate” attribute in the *encounter* table. For all patients that satisfy this condition, the query will then output the attributes “encountertype” and “dischargedisposition” along with the patient’s length of hospitalization using SQL’s DATEDIFF function. Note that the “encountertype” attributes takes values “Inpatient,” “Emergency,” “Day visit,” or None (Null value). The “discharge disposition” attribute takes values “routine discharge,” “nonroutine discharge” or “expired” if the patient has deceased. As a word of caution, it is advisable to verify if the patient has expired using the “deceased” attribute from the *demographics* table, if available.

Sometimes the latest disposition of a patient is also of interest for some conditions that are majorly affected by the COVID-19 infection. To find the latest disposition, we need to do a SQL query to retrieve all the dispositions for the patients after their COVID-19 test date and then use Python to only view the latest records. After executing the above query and storing the results in a DataFrame (df), the following Python statement can be used. By using `tail(1)`, we are extracting the last row from all the rows for each patient.

```python
df.sort_values('dischargedate').groupby('personid').tail(1)
```
To find the medications that were given to the patients, a straightforward join query between *condition* and *medication* table and selecting the required attributes can be executed. However, to find medications that were given after a patient tested positive for the COVID-19 infection, the date of the first positive result test and the start date of the medications should be known. This is similar to finding the length of stay.

Many times, a patient would go for multiple tests and may have tested positive before testing negative or vice-versa. To ensure only one positive test result is taken into consideration, the first date of positive result is used. A query, similar to the above query, is written to extract all “personid” and test dates, which in this database, is stored in the attribute “service_date” and the condition and result are given. The results are then stored in a DataFrame (df). Using Python, the first date of the positive test can be found using `min()`.

```python
df.groupby('personid').min()
```

Suppose we want to find out the various procedures a patient has undergone. We must first know the code type and the procedure code that could be an ICD-9 or CPT code. Similar to the above scenario, procedures after the COVID-19 test result can also be found once we know the date of the first positive COVID-19 test.

### 5.3.2 Challenges

Real-world data, although structured, has its own limitations and challenges. Often the data may be incomplete where only a portion of a patient’s medical information is available. Sometimes a patient may visit one of the health centers that does not have an agreement with any healthcare corporations, such as Cerner, and so the information of that visit may not be recorded. This brings us to a limitation of understanding a patient’s illness and medical history with the data available in the given dataset.

Another challenge often observed is with dates such as hospitalization start date, discharge date, COVID-19 test date, medication start and end dates. For de-identifying data, date shifting is employed consistently across a patient’s data and may be of different intervals such as ±7 or multiples of 7 between two different patients. As date shifting is managed by the underlying health information system, there is no way to overcome this challenge. Instead, we can use the data to get an estimate of how many days a patient was hospitalized or was on a certain prescribed drug.

As the data contain many records, it is important to ensure that the queries written are yielding the correct/required results. Often duplicate records or missing values are overseen which can skew the findings. Fortunately, a data scientist can drill through the data using popular tools like Jupyter Notebook, which supports PySpark allow SQL to be used for querying and Python to display the results in a better, readable format.
5.4 Machine learning

Machine learning enables a data scientist to gain meaningful insights from data. Currently, it is being used in several fields like medical diagnosis, image processing, speech recognition, financial services, and e-commerce. While data extraction is the first step in understanding how an infection like COVID-19 is affecting a population, it is valuable to develop prediction models to understand how a disease will impact an individual or the population as a whole. We can use machine learning to find hidden patterns, predict an outcome, or find solutions that would help boost healthcare decision making for new as well as existing patients. Machine learning focuses on efficiently learning mathematical models that best represent the data. Using these models, one can identify the patterns and make decisions on new input data. For the COVID-19 pandemic, machine learning can empower scientists and clinicians to better predict the survivability outcome of patients, the severity of COVID-19 in patients as well as better understand why individuals may be symptomatic or asymptomatic.

Machine learning algorithms are broadly categorized into supervised and unsupervised learning. In supervised learning, algorithms learn on labeled data and give predictions on unseen or new data. The model is trained on a labeled dataset with input features and output labels. The learning algorithm can compare its output with the true output to modify the model accordingly by minimizing a loss function. There are two types of supervised learning algorithms:

- **Classification**: The output has defined labels, that is, discrete values. The prediction can be either binary (e.g., 0 or 1, yes or no) or multi-class classification if it is more than two classes. Popular classification algorithms include Logistic Regression (Cramer, 2002), Support Vector Machine (Noble, 2006), Decision Tree classifier (Safavian and Landgrebe, 1991; Swain and Hauska, 1977), Random Forest classifier (Ho, 1995; Liaw and Wiener, 2002), and XGBoost classifier (Chen and Guestrin, 2016; XGBoost eXtreme Gradient Boosting).

- **Regression**: The output label takes continuous values. The prediction is a value that would be closer to the actual output value, and the evaluation is done by calculating an error metric (e.g., mean-squared effort). Linear regression (Montgomery et al., 2021), Decision Tree regression (Utgoff, 1989), Random Forest regression (Liaw and Wiener, 2002), and XGBoost regression (Chen and Guestrin, 2016) are some widely used regression algorithms.

In unsupervised learning, the algorithm learns on data that do not have any labels. The algorithm explores the data and draws inferences to identify hidden patterns from unlabeled data. There are three types of unsupervised learning algorithms:

- **Clustering**: The model aims to identify inherent structure/groupings in the data. K-Means is a popular clustering algorithm.

- **Association**: The model aims to identify rules that describe the dependencies between two attributes in a dataset. Apriori (Bodon, 2003), Eclat (Borgelt, 2003), and FP-Growth (Borgelt, 2005) are some popular association algorithms.
• Dimensionality reduction: The model aims to provide a succinct representation of the data, which may have high dimensionality. Principal component analysis is a popular dimensionality reduction algorithm.

The third category of machine learning is semi-supervised learning where, in a large amount of data, only some data are labeled. In this case, a mixture of unsupervised and supervised learning algorithms can be used.

Next, we discuss some classification and regression algorithms and explain how to quickly implement them using code snippets. The typical machine learning pipeline can be divided into seven steps:

1. Data gathering: quality and quantity of the data gathered will dictate how good the machine learning model will perform.
2. Data preparation: impute missing data; load and split the data into a training set for training the model, a validation set for tuning the hyperparameters, and a test set for evaluating the model.
3. Choosing a model: depending on the type of data, a particular model can be selected.
4. Training and testing the model: the training set is used to train the model and then the test set is used to test the model on new or unseen data.
5. Parameter tuning: this allows the model to perform better by choosing or “tuning” the parameters or hyperparameters for the selected model.
6. Evaluation: once the model has been trained, the performance of the model can then be evaluated by testing it on the test set.
7. Prediction: the model’s predictions are then computed to understand how well the model can predict the outcomes.

Consider the problem of identifying the survivability outcome of a patient who tested positive for COVID-19. Suppose the demographic information of all the patients are available. In addition, we may have information on the comorbidities of the patients in the past and current, the medications they were administered as well as their hospitalization durations. For simplicity, we consider only the demographic information. This can be obtained using the below query.

```python
df_ml = spark.sql("SELECT DISTINCT(condition.personid), demographics.gender, demographics.race, demographics.ethnicity, demographics.deceased FROM condition JOIN covid_labs ON condition.personid = covid_labs.personid JOIN demographics ON condition.personid = demographics.personid WHERE covid_labs.result = "Positive"").toPandas()
```

5.5 Classification

5.5.1 Logistic regression

Logistic Regression is used to identify the association of independent variables with one dependent variable. This algorithm uses a logistic function at its base, to model a binary dependent variable.
We will apply the Logistic Regression model to the dataframe that contains the results from the above query. Logistic Regression classifier can be imported from `sklearn.linear_model` package as in Line 1 of the code snippet shown in Fig. 5–2.

Once the required libraries and functions are imported, we can proceed with splitting the dataset into the train and test set. However, real-world data may not always be numerical and therefore it is necessary to map the data for each column to its numerical form. For example, Hypertension could be mapped to 0 and Type2 diabetes mellitus can be mapped to 1. To map this correctly and efficiently, we can utilize `LabelEncoder()` as shown in Line 7. Once we convert the data into a numerical form, we drop the target column and assign the remaining data from the dataframe to a variable “X”. In Line 9, we are also dropping “personid” as this column is irrelevant to the features required to predict the outcome. We assign the target column to a variable “y” as shown in Line 10.

To split the dataset into test and train split, we use the `train_test_split()` function (Line 12) that takes the data to learn from “X” as well as the target column “y”. `random_state()` ensures the splits can be regenerated. The last argument is the train size, which in this case is 0.7. This means 70% of the data is split into train set and the remaining 30% of the data will be used for testing.

Lines 13–16 are used to create the model, fit the Xtrain and ytrain parts of the dataset and then predict the output of the model using the Xtest portion of the dataset. The accuracy of the prediction can be calculated by comparing the predicted values with the actual values from ytest. As shown in Line 18, the `accuracy_score()` function is used to compute the accuracy of the model.

In real-world healthcare data, like Cerner Real-World Data, the ratio between patients that are Alive, and Deceased is imbalanced, there may be more patients that are Alive, and this may cause the machine learning model to lean more towards the higher variable, in this case, Alive. For example, for a given condition, the survivability rate could be higher, where

```python
1. from sklearn.linear_model import LogisticRegression
2. from sklearn.model_selection import train_test_split
3. from sklearn.metrics import accuracy_score
4. from sklearn.preprocessing import LabelEncoder
5. from sklearn.model_selection import GridSearchCV
6.
7. df_RF = df_ml.apply(LabelEncoder().fit_transform)
8.
9. X = df_RF.drop(['deceased', 'personid'], axis=1)
10. y = df_RF['deceased']
11.
12. Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0, train_size=0.7)
13. model = LogisticRegression(random_state=0)
14. model.fit(Xtrain, ytrain)
15.
16. y_predict = model.predict(Xtest)
17. print(y_predict)
18. print("Accuracy [0-1]: ", accuracy_score(ytest, y_predict))
```

**FIGURE 5–2** Logistic regression implementation.
the number of deceased patients may be 10 out of a set of 100 patients. In this case, the model may incorrectly predict the status as most patients are Alive. One way to check this is by implementing the code snippet in Fig. 5–3.

This calculates how many times the model correctly predicted the target variable by comparing the output with its actual value. Line 5 prints this information for every row in ytest.

We can similarly apply other machine learning models on the same dataset.

5.5.2 Decision tree classification

Decision Tree classification models build trees by breaking down a data set into smaller subsets continuously based on a set parameter. Decision trees learn data with a set of if-then-else decision rules and the tree is incrementally developed. The output of this model is in a tree with a root node, decision nodes, and leaf nodes. A decision node has two or more branches, and a leaf node represents a decision or a classification. Root node corresponds to the topmost decision node.

The first step in building a decision tree is to partition the data into multiple subsets. The splits are formed based on an attribute value. To avoid overfitting, a simpler tree is often advisable. A tree can be shortened by limiting the branches or “pruning”. Pruning reduces the size of the tree by converting some branch nodes into leaf nodes and thereby removing leaf nodes under the original branch. Classification trees may fit training data well but may underperform on new test data. Pruning is useful in avoiding this.

To build the Decision Tree Classification model, we first load DecisionTreeClassifier() class from Scikit-learn library’s tree package. The import statements for building the model by training and testing it on the data queried above are shown in Fig. 5–4.

In a similar fashion to Logistic Regression, we split the resulting data from the query into X (data) and y (target) variables and then split both X and y into training and test sets (Lines
We can then proceed with creating the Decision Tree Classifier using the `DecisionTreeClassifier()` class that takes multiple parameters such as `criterion`, `max_depth`, and `max_features`, as shown in Line 11, Fig. 5–5.

Decision Tree Classifier takes two criteria—Gini Index and Entropy. Gini Index calculates the probability of a specific feature being misclassified when randomly selected. Entropy helps the model build an appropriate decision tree by selecting the best splitter. Gini Index is more efficient than Entropy in terms of computation power. `max_depth` is the maximum depth of the tree. Once we set the parameters that would be best suited for building the decision tree, we can fit the model on `Xtrain` and `ytrain` sets of the data (Line 12, Fig. 5–5). We can then predict the accuracy of the model in a similar way as we did for Logistic Regression (Lines 14–16, Fig. 5–5).

One way to identify the best parameters is to use `GridSearchCV` which performs an Exhaustive Search over the specified parameter values for an estimator. `GridSearchCV` implements a “fit” and “score” method and combines an estimator with a grid search to tune hyperparameters. `GridSearchCV()` can be imported from Scikit-learn’s `model_selection()` package as in Line 1, Fig. 5–6.

The first step to implement `GridSearchCV` (Fig. 5–6) is to define the estimator by passing the model instance as in Line 2. Then, the predefined values for the hyperparameters are passed as a dictionary to the `GridSearchCV` class (Line 3), for example, `criterion` and `max_depth`. The estimator, parameter grid and cross-validation folds are then passed to the

```python
5. df_RF = df_ml.apply(LabelEncoder().fit_transform)
6.
7. X = df_RF.drop(['deceased', 'personid'], axis=1)
8. y = df_RF['deceased']
9.
10. Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0, train_size=0.7)
11. model = DecisionTreeClassifier(criterion="gini", max_depth=2, random_state=0)
12. model.fit(Xtrain, ytrain)
13.
14. y_predict = model.predict(Xtest)
15. print(y_predict)
16. print("Accuracy [0-1]: ", accuracy_score(ytest, y_predict))

FIGURE 5–5 Decision tree classifier implementation.

1. from sklearn.model_selection import GridSearchCV
2. model_DTC = DecisionTreeClassifier()
3. parameters_DTC = {'criterion': ['gini', 'entropy'], 'max_depth': [2, 4, 6]}
4. clf_DTC = GridSearchCV(model_DTC, parameters_DTC, cv=5)
5. clf_DTC.fit(X, y)
6. print("Best parameters for Decision Tree Classifier: ", clf_DTC.best_params_, "
\n")
7. print("Mean test score for Decision Tree Classifier: ", clf_DTC.cv_results_['mean_test_score'], "\n")

FIGURE 5–6 GridSearchCV implementation for decision tree classifier.

7–10, Fig. 5–5).
GridSearchCV class (Line 4). The number of cross-validation set defines how many times the train and test data samples will be divided and tested on. Cross-validation trains the model on several subsets of the input data and then evaluates on the remaining subsets of the data. Line 6 displays the combination of parameters that yield the highest accuracy, using the given parameter grid and Line 7 prints the mean test score for each cross-validation fold.

5.5.3 Random forest classification

Random Forest is another supervised learning algorithm that builds multiple decision trees and merges them to develop a more accurate and stable prediction. The ensemble of decision trees this algorithm builds can also be referred to as a “forest” that is usually trained by a “bagging” method. The idea behind Bagging is that a combination of learning models increases the overall accuracy and result. Random Forests can be used for both classification and regression problems.

The first step to building a Random Forest Classifier model is importing all the required packages and libraries. Random Forest model can be built by using Scikit-learn’s RandomForestClassifier(). All the packages we will require to build the model are in Lines 1–4 in Fig. 5–7.

As in Fig. 5–7, we split the dataframe with the queried results into data and target variables, X and y respectively (Lines 6–9), and then split X and y into train and test sets (Line 11).

RandomForestClassifier() takes multiple hyperparameters that are similar to DecisionTreeClassifier() such as n_estimators that define the number of trees in the forest, max_depth that defines the maximum depth of the trees, criterion that can be Gini Index or Entropy. Here, in Line 12, we are only setting the number of trees in the forest and the maximum depth of the trees. Once the parameters are set, we can then proceed with fitting the

```python
1. from sklearn.ensemble import RandomForestClassifier
2. from sklearn.model_selection import train_test_split
3. from sklearn.metrics import accuracy_score
4. from sklearn.preprocessing import LabelEncoder
5.
6. df_RF = df_ml.apply(LabelEncoder().fit_transform)
7.
8. X = df_RF.drop(['deceased', 'personid'], axis=1)
9. y = df_RF['deceased']
10.
11. Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0, train_size=0.7)
12. model = RandomForestClassifier(n_estimators=15, max_depth=2, random_state=0)
13. model.fit(Xtrain, ytrain)
14.
15. y_predict = model.predict(Xtest)
16. print(y_predict)
17. print("Accuracy [0-1]: ", accuracy_score(ytest, y_predict))
```

FIGURE 5–7 Random forest classifier implementation.
model with the X and y train sets and then predict on the X test set. The accuracy is then calculated by comparing y_test set and the prediction done on X_test set, as in Line 17.

Similar to finding the best suitable hyperparameters for Decision Tree Classifier, we can use GridSearchCV() to find the best hyperparameters for Random Forest Classifier as well, as shown in Fig. 5–8. The parameter grid consists of n_estimators, max_features, max_depth, and criterion. n_estimators takes a list of numbers for the number of trees in the forest. max_features provide the maximum number of features to consider for each tree when searching for the best split. It can be “auto” where all the features that may apply to every tree are considered, “sqrt” where the square root of the total number of features will be considered, or “log2” where the random forest only takes 20% of the variables per tree. max_depth is also a list of numbers that the random forest can use, to decide the best depth for the trees. criterion, similar to that of Decision Tree, can be either Gini Index or Entropy.

5.5.4 XGBoost classification

XGBoost algorithm has been recently dominating the field of machine Learning and is an implementation of gradient boosted decision trees intended for improved speed and performance. XGBoost stands for “eXtreme Gradient Boosting” and is a software library that needs to be installed in order to be implemented. Gradient Boosting is a type of machine learning boosting that is based on the idea that the best possible model combined with the previous models minimizes the overall prediction error. XGBoost is better than Gradient Boosting as it performs much better and faster and can be distributed across clusters.

To get started with installing the XGBoost package, the following code in Fig. 5–9 can be used.

Another way to install XGBoost using Python is by using the pip command in Terminal, as shown in Fig. 5–10.

Once the package has been successfully installed, the required packages, shown in Fig. 5–11, can be imported.

Note that XGBClassifier() belongs to the xgboost library.
In Fig. 5–12, Lines 7–11 are similar to the other models discussed above. XGBClassifier() also takes multiple parameters like Random Forest or Decision Tree. However, for simplicity, we are only setting the max_depth for the trees built by the model. Similar to the other algorithms, once the model is created the data is fit, the output is predicted, and the accuracy is calculated as shown in Lines 13–17.

5.5.5 Regression

5.5.5.1 Linear regression
Linear Regression is a linear approach to modeling a relationship or finding a straight line between a target or dependent variable and a set of covariant or independent variables. In simple terms, it is the process of finding a line that best fits the data points on the plot that can be used to predict data points for values in the test set. The idea here is that the output would fall on the line. Linear Regression makes the assumption that there is a linear relation between the input variable (x) and a single output variable (y) such that y can be calculated from a linear combination of the input variable x. It is used to predict values that are in a continuous range such as age instead of classifying them into categories such as yes/no or 0/1.
To build the model, we import the `LinearRegression()` class from Scikit-learn’s `linear_model` package, as shown in Line 1, Fig. 5–13. All the other steps of the algorithm (Lines 2–15) remain the same as the other models discussed above.

It is important to note that the evaluation metrics (Lines 16–19) for Regression models are different from Classification models since Regression models are used for continuous ranges. In Classification models, `accuracy_score()` would provide exactly how correctly the model could predict the outcome. Regression models use `max_error()` that calculates the maximum residual error, `mean_absolute_error()` or “mae” that is the mean of the absolute values of individual prediction errors over all instances of the test set, `mean_squared_error()` or “mse” gives the average of the squares of errors that defines how close a regression line is to the data points, and `r2_score()` that gives a value between 0 and 1 that tells how close the data are fitted to the regression line.

### 5.5.5.2 Decision tree regression

As we have covered earlier, decision tree is a supervised learning model that is used to predict a target outcome by learning decision rules from the features in the dataset. Decision Trees can be applied to classification as well as regression problems. To implement `DecisionTreeRegressor()`, we use Scikit-learn’s `tree` package.

The data splitting, model fitting, and prediction of the outcome (Lines 8–18) are similar to Classification algorithms, as in Fig. 5–14. The evaluation metrics are shown in Lines 20–23.

To decide the best set of hyperparameters for achieving a stable and efficient model, we can use `GridSearchCV()`, similar to the other models previously discussed. The implementation is shown in Fig. 5–15.

```python
1. from sklearn.linear_model import LinearRegression
2. from sklearn.model_selection import train_test_split
3. from sklearn.metrics import accuracy_score
4. from sklearn.preprocessing import LabelEncoder
5. df_RF = df_ml.apply(LabelEncoder().fit_transform)
6. X = df_RF.drop(['deceased', 'personid'], axis=1)
7. y = df_RF['deceased']
8. Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0, train_size=0.7)
9. model = LogisticRegression(random_state=0)
10. model.fit(Xtrain, ytrain)
11. y_predict = model.predict(Xtest)
12. print("Max error: ", max_error(ytest, y_predict))
13. print("Mean absolute error: ", mean_absolute_error(ytest, y_predict))
14. print("Mean squared error: ", mean_squared_error(ytest, y_predict))
15. print("R2 score: ", r2_score(ytest, y_predict))
```

**FIGURE 5–13** Linear regression implementation.
As we already know, the idea behind generating a Random Forest is to combine multiple “weak” Decision Trees to create a more stable and efficient model. Similar to RandomForestClassifier(), we import RandomForestRegressor() from Scikit-learn’s ensemble package, as in Fig. 5-16.

The data is split using the train_test_split() function, the model is then fit with the data and the model then predicts on the test set (Lines 13–23). The hyperparameters for RandomForestRegressor are similar to DecisionTreeRegressor. To find the best hyperparameters, GridSearchCV() (Lines 25–31) can be used, similar to the models above.
5.5.6 Data visualization

Data can be visualized in the form of graphs, pie charts, and plots. Consider the following query that extracts the demographic information for all patients that tested positive for COVID-19 and have the condition “Acute respiratory distress syndrome” that is associated with two codes-67782005 (SNOMED-CT) and J80 (ICD-10-CM). A JOIN operation has been performed on condition, covid_labs, and demographics table on “personid”. The constraints are mentioned in the “WHERE” part of the query with the COVID-19 result and the condition codes.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import max_error, mean_absolute_error, mean_squared_error, r2_score

df_RF = df_ml.apply(LabelEncoder().fit_transform)

X = df_RF.drop(['deceased', 'personid'], axis=1)
y = df_RF['deceased']

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0, train_size=0.7)

model = RandomForestRegressor(n_estimators=15, max_depth=2, random_state=0)
model.fit(Xtrain, ytrain)

y_predict = model.predict(Xtest)
print(y_predict)

print("Max error: ", max_error(ytest, y_predict))
print("Mean absolute error: ", mean_absolute_error(ytest, y_predict))
print("Mean squared error: ", mean_squared_error(ytest, y_predict))
print("R2 score: ", r2_score(ytest, y_predict))

model_RPC = RandomForestRegressor()
parameters_RPC = [{'n_estimators': [5, 10, 15], 'max_features': ['log2', 'sqrt']}, {'max_depth': [2, 4], 'criterion': ['mse', 'mae']}

clf_RPC = GridSearchCV(model_RPC, parameters_RPC, cv=5)
clf_RPC.fit(X, y)

print("Best parameters for Random Forest Regressor: ", clf_RPC.best_params_, 
"
"
print("Mean test score for Random Forest Regressor: ", clf_RPC.cv_results_['mean_test_score'], 
"
```

FIGURE 5-16 Random forest regression implementation.

```sql
df_demographics = spark.sql("""SELECT DISTINCT(condition.personid), demographics.gender, demographics.race, demographics.ethnicity, demographics.deceased FROM condition JOIN covid_labs ON condition.personid = covid_labs.personid JOIN demographics ON condition.personid = demographics.personid WHERE covid_labs.result = "Positive" AND (condition.conditioncode = "J80" AND condition.codetype = "ICD-10-CM") OR condition.conditioncode = "67782005" AND condition.codetype = "SNOMED-CT") """".toPandas()
```
This query will yield multiple rows of unique patients that have tested positive for the COVID-19 infection at some point during their visits and have the given condition. Sometimes, a patient may have visited two different hospitals, one of which may have coded the condition using the ICD-10-CM code and the other hospital may have used SNOMED-CT codes. Hence, it is important to use the `groupby()` function with an aggregate function on the personid.

The queried results can be visualized in many different ways. This is important for the user to understand the patterns in the data or to simply understand the data. A histogram is a graphical display of data using bars that represent ranges of grouped numbers. Though histogram is similar to a bar chart, it is used when frequency distributions are to be shown. Bar charts, on the other hand, also use bars to represent the data but are used to compare the frequency or numbers for discrete categories of data.

Another way to represent data is by using pie charts that divide the frequency or numbers of the data into proportions and display them in a circular graphical chart. Each portion of the pie chart is called a “slice” and represents a distinct category. Line plots are used for non-cyclic data, where the data can be represented as points on or above a line. Area plots provide maximum information for time-series data such as the timeline for a COVID-19 positive patient or to display the number of patients that tested positive for COVID-19 during a date range. When the data consists of numerical data that can be represented using quadrants that identify measures of spread such as minimum, maximum, and mean, box plots are used. Box plots can be displayed horizontally or vertically. A good example would be the ages of a group of patients.

Python provides in-built functions to use these charts or “plots” in a very convenient manner. Consider a table that contains insulin administration on patients hospitalized due to COVID-19. This table contains a column labeled “insulin” that contains 4 unique values—“Down”, “No”, “Steady”, and “Up”. In this case, we can use the `hist()` function provided by Pandas. This function comes from the hierarchy `pandas.DataFrame.hist()` and therefore, does not require us to specifically call any library.

```
df['insulin'].hist()
```

This function finds the counts for each of the column values and plots them in the histogram chart.

When the race categories of the patients from the above query are to be represented graphically, a bar chart would be most informative since the types of races are discrete. In the code shown in Fig. 5–17, Line 1 groups the results by “race” for each “personid” and displays the counts in a bar chart. The argument “title” in `plot()` is used to give the chart a title. This chart information is stored in a variable “ax” that allows us to call other sub-functions

```
1. ax = df_demographics.groupby('race')['personid'].count().plot(kind='bar', title ="Demographics - Race Bar Plot")
2. ax.set_xlabel("Race")
3. ax.set_ylabel("Frequency")
```

**FIGURE 5–17** Code snippet to display data in a bar chart.
such as `set_xlabel()` to set the x-axis label and `set_ylabel()` to set the y-axis label. The bar chart is shown in Fig. 5–18.

We can use a bar chart to display the ethnicities of the group of patients. However, we can also display this categorical information using a Pie chart. In Fig. 5–19, Lines 1 and 2 depict another way of writing the same code as above. The arguments passed to the `plot()` function are additional for a pie chart to allow more visual information. Like any other chart, pie chart also requires an x-axis and y-axis, and therefore, Line 3 allows us to hide the y-axis label, which in this case would be “personid”.

Each slice in Fig. 5–20 represents the proportion of an Ethnic group in the queried data results.

The database contains ages for the patients at every “encounter”, that is, for every visit the age is updated and the age information can be depicted using a Box plot. A boxplot is

![Demographics - Race Bar Plot](image)

**FIGURE 5–18** Bar chart representing the frequency of race for acute respiratory distress syndrome patients that tested positive for COVID-19 infection.

```python
1. df_plot = df_demographics.groupby('ethnicity')['personid'].count()
2. ax = df_plot.plot.pie(title = "Demographics - Ethnicity Pie Plot", autopct='%.1f%%', startangle=270, fontsize=10)
3. ax.yaxis.set_visible(False)
```

**FIGURE 5–19** Code snippet to display data in a pie chart.
used to show the spread (interquartile range and mean of dataset) and centers (mean and median) of the dataset.

Consider the following query that has been expanded from the previous query, where there is a JOIN operation done on `condition`, `covid_labs`, `demographics` as well as on `encounter` table as encounter table contains the attribute “age_at_encounter”. This will yield multiple rows for each patient, as the age is updated on every visit.

```sql
def = spark.sql("""
SELECT DISTINCT(condition.personid), encounter.age_at_encounter,
demographics.gender, demographics.race, demographics.ethnicity,
demographics.deceased FROM condition JOIN covid_labs ON condition.personid =
covid_labs.personid JOIN demographics ON condition.personid = demographics.personid
JOIN encounter ON condition.personid = encounter.personid
WHERE covid_labs.result = "Positive" AND (condition.conditioncode = "J80" AND condition.codetype = "ICD-10-CM")
OR condition.conditioncode = "67782005" AND condition.codetype = "SNOMED-CT")""
toPandas()
```

To get the maximum age for each patient, we can use Python as follows.

```python
df.groupby('personid')['age_at_encounter'].max()
```

However, we can expand our box plot to represent the maximum age for each patient belonging to one of the three genders, “Male”, “Female” and “Unknown”. This can be done by modifying the above Python code as follows:

```python
df.groupby('personid')['age_at_encounter', 'gender'].max()
```

To plot this information in a box plot, shown in Fig. 5–21, the following code shown in Fig. 5–22 is used. The function `boxplot()` takes the argument “by” where we specify which
This graph information is stored in a variable “ax” that can be used to call other functions (Lines 2–6) such as `set_title()` to set the graph title, `set_xlabel()`, and `set_ylabel()` to set the x- and y-axis labels. The `get_figure()` function has a sub-function `suptitle()` that allows us to set the boldfaced super title of the box plot. If this title is not required, it can be set as an empty string as shown in Line 5. By default, box plots show the gridlines which can be set to False if not required, as in Line 6.

Box plots give 5 main pieces of information, generally called “the five-number summary” (Wikipedia, 2020), that are included in the chart:

- Minimum: the data point shown at the bottom-most of the chart.
- First quartile, Q1: the bottom-most of the box.
- Median or middle value: the line shown in the center of the box.
- Second quartile, Q2: the top-most of the box.
- Maximum: the data point shown at the top-most of the chart.

The points at the bottom-most of the chart indicate the outliers of the dataset. They may also lie in the top-most part of the chart.
Applying this information to the box plot in Fig. 5-22, we can observe the minimum, maximum, and median ages for all the genders. We can also observe the outliers in “Female” and “Male” categories.

5.6 Review questions

1. Explain what tables are typically used to store the medical data of patients.
2. Explain what is a foreign key in a table.
3. Write a SQL query to extract the conditions of a patient after 2020-10-10 from the condition table.
4. What is the difference between supervised and unsupervised machine learning?
5. What is the difference between classification and regression?
6. What is a box plot?

5.7 Discussion questions

1. We learned how to extract patients’ information from their first COVID-19 positive test date using Spark SQL/Python. How can we extend this approach to extracting all distinct medications for patients after they have recovered from COVID-19 infection, that is, the last time they tested positive after which all subsequent test results were negative?
2. We learned the basics of implementing a machine learning algorithm using Python and discussed some popularly used ones. Which algorithms would you choose (and why) to accurately predict whether a patient can acquire COVID-19 infection given the COVID-19 symptoms and their underlying historical conditions?
3. How can you visualize the timeline of a given patient to understand how a certain condition, such as hypertension or atrial fibrillation, is impacted by COVID-19? The plot would include their historical and current comorbidities, medications, procedures, and hospitalization details. Which kind of plot would best depict this nature of data?

5.8 Problem statements

When a pandemic occurs, patient information becomes a critical resource for conducting large-scale clinical studies. However, the reality is that healthcare vendors can offer only partial data for these studies as they must act in a timely manner to enable new research and discoveries. The significance and impact of research studies will depend on how large the patient population is and what patient information is available. Certain variables that a clinician expects may not be available. Data may also have certain artifacts that must be well-understood before using them for data science tasks. It is worth exploring if a formal approach can be developed to describe the significance of the available data and the appropriateness of a clinical study and the hypotheses that are being tested.
Another important problem with regard to machine learning is the coverage of the COVID-19 datasets from participating healthcare centers. This is important because a model’s accuracy would depend on how well a training dataset represents new data used for prediction. Sometimes healthcare centers may not be willing to share their data. Sometimes the de-identification processes could strip away valuable information. We believe there are new opportunities to leverage privacy-preserving machine learning in such cases.

There are many clinical perspectives that can be derived from COVID-19 datasets. For example, medications that are most frequently administered for patients that have the COVID-19 infection, and have been hospitalized as Inpatient or Emergency, can help understand the population ratio to each drug and the impact the drug has. This can be extended to understanding how well a particular drug has worked on a sample population, given the mortality rate for that sample set.

5.9 Conclusion

Data are incredibly important as they help us understand the magnitude of a public health crisis such as COVID-19. Using rich data, one can identify time-constrained and feasible solutions to detect, monitor, and mitigate the spread of COVID-19. There are various sources that offer free and public COVID-19 datasets that may be de-identified. Some of these data are real-time and frequently updated. Such real-world data can provide some important insights on how healthcare officials and clinicians can introduce new practices or improve existing ones. Data scientists can understand the trends of how a pandemic is spreading using the various tools and dashboards available to them.

In this chapter, we utilized Cerner Real-World Data through their cloud-based portal. We presented some of the challenges using this large-scale data (e.g., due to de-identification) in clinical studies. We showed how queries can be posed to extract relevant data of patients and their conditions. We also showed how various machine learning models can be applied on the data to find patterns and predict possible outcomes. Both classification and regression use cases were covered. The results and data can be plotted using various visualization tools available in open-source frameworks like Jupyter Notebook.

Acknowledgments

We would like to acknowledge the support of the Center for Biomedical Informatics (CBMI) at University of Missouri-Columbia. We would like to thank Cerner and Tiger Institute for this collaboration. The second author (P.R.) would like to acknowledge the partial support of the National Science Foundation under Grant No. 2034247.

References

Beigel, J.H., Tomashek, K.M., Dodd, L.E., Mehta, A.K., Zingman, B.S., Kalil, A.C., et al., 2020. Remdesivir for the treatment of Covid-19—preliminary report. The New England Journal of Medicine.
BigQuery Public Datasets Program, 2020. COVID-19 Open Data. <https://console.cloud.google.com/marketplace/product/bigquery-public-datasets/covid19-open-data?_ga=2.152850233.-1862715341.1610298430&pli=1&project=loginreg-f935b&folder=&organizationId=> (accessed 05.01.21).

Bodon, F., 2003. A fast APRIORI implementation. FIMI 3, 63.

Borgelt, C., 2003 Efficient implementations of apriori and eclat. In FIMI’03: Proceedings of the IEEE ICDM Workshop On Frequent Itemset Mining Implementations (p. 90).

Borgelt, C., 2005 An implementation of the FP-growth algorithm. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations (pp. 1–5).

CDC, 2020 United States COVID-19 Cases and Deaths by State. <https://covid.cdc.gov/covid-data-tracker/#cases_casesper100klast7days> (accessed 10.01.21).

Cerner, 2021. Real-World Data. <https://www.cerner.com/solutions/real-world-data> (accessed 19.01.21).

Chen, T. and Guestrin, C., 2016. Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794).

CORD-19. COVID-19 Open Research Dataset <https://www.semanticscholar.org/cord19> (accessed 05.01.21).

Cramer, J.S., 2002. The Origins of Logistic Regression.

Digala, L., Prasanna, S., Rao, P., Govindarajan, R., Qureshi, A., 2021a. Impact of COVID-19 Infection Among Hospitalized Amyotrophic Lateral Sclerosis Patients. Journal of Clinical Neuromuscular Disease 22 (1), 180—181. 2021.

Digala, L., Prasanna, S., Rao, P., Govindarajan, R., Qureshi, A., 2021b. Cerner Real-World Data™ Study of Spinal Muscular Atrophy Patients With Positive COVID-19 Infection. Journal of Clinical Neuromuscular Disease 22 (4), 239—240. 2021.

Digala, L., Prasanna, S., Rao, P., Govindarajan, R., Qureshi, A., 2021c. Five cases of charcot-marie-tooth disease with positive COVID-19 infection reported using Cerner Real-World Data™. Journal of Clinical Neuromuscular Disease in press.

edX, 2016 The Future of Data Science: Q&A with MIT Professional Educations Devavrat Shah. <https://blog.edx.org/future-data-science-qa-mit-professional-educations-devavrat-shah> (accessed 12.01.21).

FDA, 2020. Coronavirus Treatment Acceleration Program (CTAP). <https://www.fda.gov/drugs/coronavirus-covid-19-drugs/coronavirus-treatment-acceleration-program-ctap> (accessed 26.01.21).

Forbes, 2017. Five Things to Watch in AI and Machine Learning in 2017. <https://www.forbes.com/sites/moorinsights/2017/01/06/five-things-to-watch-in-ai-and-machine-learning-in-2017/?sh=2ca5df9c455a> (accessed 19.01.21).

Harvard Business Review, 2012 Data Scientist: The Sexiest Job of the 21st Century. <https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century> (accessed 12.01.21).

Ho, T.K., 1995. Random decision forests. Proceedings of 3rd International Conference On Document Analysis And Recognition 1, 278–282. IEEE.

Jennings, W.C., Glass, S., 2020. COVID-19 public dataset program: making data freely accessible for better public outcomes. <https://cloud.google.com/blog/products/data-analytics/free-public-datasets-for-covid19> (accessed 05.01.21).

Johns Hopkins University, 2021. COVID-19 Dashboard by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU). <https://coronavirus.jhu.edu/map.html> (accessed 05.01.21).

Koonin, L.M., Hoots, B., Tsang, C.A., Leroy, Z., Farris, K., Jolly, B., et al., 2020. Trends in the use of telehealth during the emergence of the COVID-19 pandemic—United States, January–March 2020. Morbidity and Mortality Weekly Report 69 (43), 1595.

Kricka, L.J., Polevikov, S., Park, J.Y., Fortina, P., Bernardini, S., Satchkov, D., et al., 2020. Artificial intelligence-powered search tools and resources in the fight against COVID-19. Ejjfccc 31 (2), 106.
Li, L., Zhang, W., Hu, Y., Tong, X., Zheng, S., Yang, J., et al., 2020. Effect of convalescent plasma therapy on time to clinical improvement in patients with severe and life-threatening COVID-19: a randomized clinical trial. Jama 324 (5), 460–470.

Liaw, A., Wiener, M., 2002. Classification and regression by random forest. R News 2 (3), 18–22.

Mckinsey, 2016 The Age of Analytics: Competing In A Data-Driven World. <https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/the-age-of-analytics-competing-in-a-data-driven-world> (accessed 12.01.21).

Montgomery, D.C., Peck, E.A., Vining, G.G., 2021. Introduction to Linear Regression Analysis. John Wiley & Sons.

NIH, 2020a. NIH study aims to identify promising COVID-19 treatments for larger clinical trials. <https://www.nih.gov/news-events/news-releases/nih-study-aims-identify-promising-covid-19-treatments-larger-clinical-trials> (accessed 26.01.21).

NIH, 2020b. Open-Access Data and Computational Resources to Address COVID-19. <https://datascience.nih.gov/covid-19-open-access-resources> (accessed 05.01.21).

Noble, W.S., 2006. What is a support. vector machine? Nature Biotechnology 24 (12), 1565–1567.

NYU, 2013. <https://datascience.nyu.edu/what-is-data-science> (accessed 12.01.21).

PAHO, 2020. Global research on COVID-19 is available in new WHO database. <https://www.paho.org/en/news/24-4-2020-global-research-covid-19-available-new-who-database> (accessed 25.01.21).

Penn Medicine, 2021. Plasmapheresis Donor Registry for Patients Recovered from Confirmed COVID-19. <https://clinicalresearch.itmat.upenn.edu/clinicaltrial/6404/covid-19-covid19-coronavirus-plasma-donor-registry-for-patients-recovered-from-confirmed-covid19/?qd = 1460104>, (accessed 26.01.21).

Ren, J.L., Zhang, A.H., Wang, X.J., 2020. Traditional chinese medicine for COVID-19 treatment. Pharmacological Research 155, 104743.

Safavian, S.R., Landgrebe, D., 1991. A survey of decision tree classifier methodology. IEEE Transactions on Systems, Man, And Cybernetics 21 (3), 660–674.

Shuja, J., Alanazi, E., Alasmary, W., Alashaikh, A., 2020. Covid-19 open source data sets: a comprehensive survey. Applied Intelligence 1–30.

Swain, P.H., Hauska, H., 1977. The decision tree classifier: design and potential. IEEE Transactions on Systems, Man, And Cybernetics 15 (3), 142–147.

Utgoff, P.E., 1989. Incremental induction of decision trees. Machine Learning 4 (2), 161–186.

Wang, L.L., Lo, K., Chandrasekhar, Y., Reas, R., Yang, J., Eide, D., et al., 2020. CORD-19: Covid-19 Open Research Dataset ArXiv.

WHO, 2021. Global research on coronavirus disease (COVID-19). <https://www.who.int/emergencies/diseases/novel-coronavirus-2019/global-research-on-novel-coronavirus-2019-ncov> (accessed 05.01.21).

Wikipedia, 2018. <https://en.wikipedia.org/wiki/Data_science> (accessed 12.01.21).

Wikipedia, 2020. <https://en.wikipedia.org/wiki/Five-number_summary> (accessed 05.01.21).

XGBoost eXtreme Gradient Boosting. <https://github.com/dmlc/xgboost> (accessed 15.02.21).

Ye, Q., Wang, B., Mao, J., 2020. The pathogenesis and treatment of the ‘cytokine storm’ in COVID-19. J. Infect. 80 (6), 607–613.