5. Plot the bar chart with the variable 'date' on the x axis and closing price on the y axis (see Figure 38-12).

```
# plot the bar chart
litcoin_crypto.plot(kind='bar', x='date', y='close')
plt.show()
```

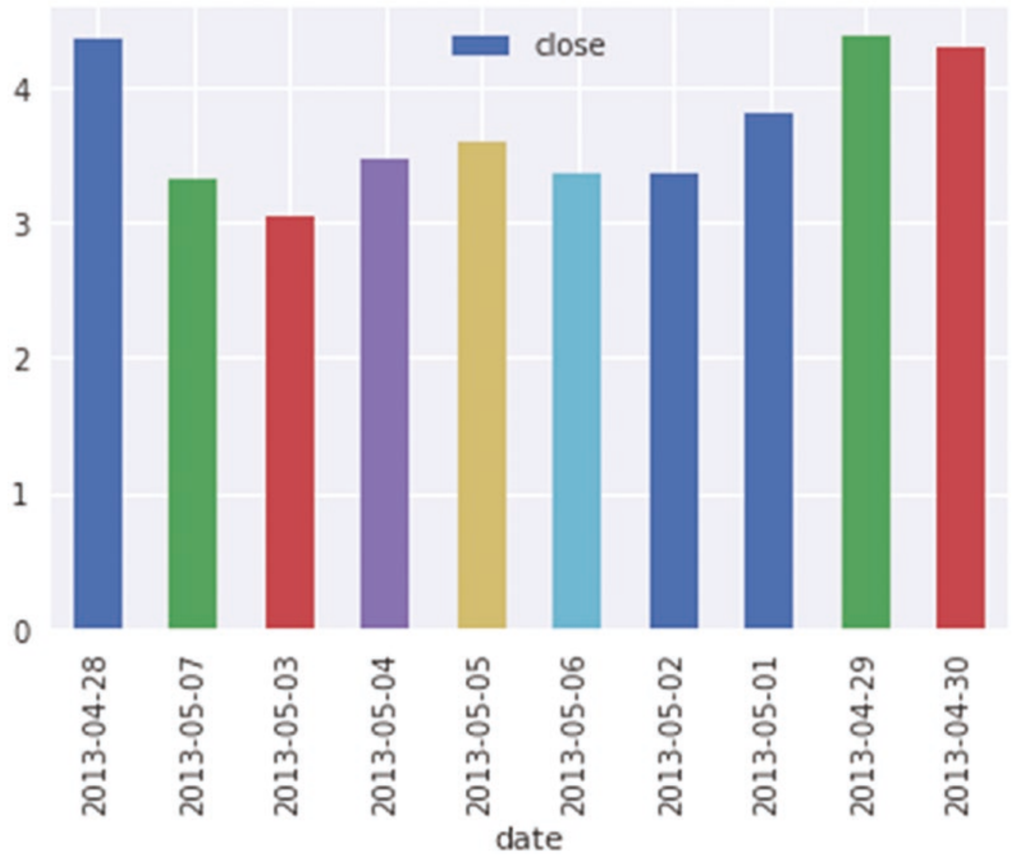

***Figure 38-12.*** *Litcoin crypto-currency bar chart plot*

## BigQueryML

BigQuery machine learning makes it quick and easy to harness the power of machine learning on your datasets in BigQuery by using simple standard SQL commands. This functionality includes the capability to train and test models on the datasets by using subsets of the data, as well as the capability for automatic hyper-parameter tuning of the learning models.

At this time of writing, the following learning models are available in BigQuery:

- Linear regression
- Binary and multi-class logistic regression

In this section, we'll work with BigQuery ML using the Notebook instance on Colab on Google AI VMs to build a predictive model using the 'market' table in the 'crypto_data' dataset that we earlier imported into BigQuery. This model will attempt to predict the next day's closing price of the Bitcoin crypto-currency given a set of market attributes. The data processing and machine learning modeling is all done using standard SQL:

1. Open a new notebook.
2. Select features for training the ML model. In the SQL code, we use the 'LEAD()' function to return the value of the next row. The offset of 1 indicates that we want to get the next value that is one step ahead in the query. With this, it is easy to adjust the query to predict a 2- to n-day window. The LEAD() function is a window function that moves over a rowset. Hence, the OVER() function is used to define a window within a query, while the PARTITION BY and ORDER BY clauses divide the query results into partitions and define the arrangement of the rows within each partition.

   We use the 'params' variable to sample half of the data and store it in the 'TRAIN' set. This makes sure that the rest of the dataset is not used in model training and can be used to check that the model generalizes well during the model evaluation phase.

   Be sure to update the FROM field with your dataset and table IDs.

```
%%bigquery --project ekabasandbox btc_market
WITH
  params AS (
  SELECT
    1 AS TRAIN,
    2 AS EVAL ),

  btc_market AS (
  SELECT
    symbol,
    date,
    open,
    high,
    low,
    close,
    spread,
    cast(LEAD(close, 1) OVER (PARTITION BY symbol ORDER BY symbol
    DESC) AS NUMERIC) AS next_day_close
  FROM
    `crypto_data.markets`,
    params
  WHERE
    symbol = 'BTC'
    AND MOD(ABS(FARM_FINGERPRINT(CAST(date AS STRING))),4) =
    params.TRAIN )

SELECT
  *
FROM
  btc_market
WHERE
  next_day_close IS NOT NULL
```

3. Display the first ten rows of the query.

```
btc_market.head(10)
```

```
symbol  date    open    high    low close   spread  next_day_close
0   BTC 2013-05-05  112.9   118.8   107.14  115.91  11.66 112.3
1   BTC 2013-05-06  115.98  124.66  106.64  112.3   18.02 112.67
2   BTC 2013-05-09  113.2   113.46  109.26  112.67  4.2   115.24
3   BTC 2013-05-11  117.7   118.68  113.01  115.24  5.67  111.5
4   BTC 2013-05-14  117.98  119.8   110.25  111.5   9.55  114.22
5   BTC 2013-05-15  111.4   115.81  103.5   114.22  12.31 121.99
6   BTC 2013-05-19  123.21  124.5   119.57  121.99  4.93  123.89
7   BTC 2013-05-22  122.89  124     122     123.89  2     133.2
8   BTC 2013-05-24  126.3   133.85  125.72  133.2   8.13  131.98
9   BTC 2013-05-25  133.1   133.22  128.9   131.98  4.32  133.48
```

4. The trained model is stored in a BigQuery dataset. In this case, we'll create a BigQuery dataset to store the model.

```
from google.cloud import bigquery
client = bigquery.Client(project='ekabasandbox')
# create a BigQuery dataset to store your ML model
dataset = client.create_dataset('btc_crypto')
print('Dataset: `{}` created.'.format(dataset.dataset_id))
```

```
Dataset: `btc_crypto` created.
```

5. After preparing our training dataset, now it is time to train the model. Be sure to update the FROM field with your dataset and table IDs.

```
%%bigquery --project ekabasandbox model
CREATE OR REPLACE MODEL `btc_crypto.market_closing_model`
OPTIONS
  (model_type='linear_reg',
    labels=['next_day_close']) AS
WITH
  params AS (
  SELECT
    1 AS TRAIN,
    2 AS EVAL ),
  btc_market AS (
  SELECT
    CAST(open AS NUMERIC) AS open,
    CAST(high AS NUMERIC) AS high,
    CAST(low AS NUMERIC) AS low,
    CAST(close AS NUMERIC) AS close,
    CAST(spread AS NUMERIC) AS spread,
    CAST(LEAD(close, 1) OVER (PARTITION BY symbol ORDER BY symbol
    DESC) AS NUMERIC) AS next_day_close
  FROM
    `crypto_data.markets`,
    params
  WHERE
    symbol = 'BTC'
    AND MOD(ABS(FARM_FINGERPRINT(CAST(date AS STRING))),4) =
    params.TRAIN )
SELECT
  *
FROM
  btc_market
WHERE
  next_day_close IS NOT NULL
```

6. Check that the created model exists in the Dataset 'btc_crypto.' We prefix the exclamation sign ('!') in a Notebook cell to execute bash commands.

```
!bq ls btc_crypto

        tableId          Type    Labels   Time Partitioning
 ---------------------- ------- -------- -------------------
  market_closing_model   MODEL
```

7. Evaluate the model to estimate the performance of the model. The RMSE metric is evaluated in BigQuery calling the 'mean_squared_error' field of the trained model and passing it through the 'SQRT()' function. To evaluate the model, pass the model through the function 'ML.EVALUATE().' This time we select the remaining subset of the dataset and store it in 'params.EVAL.'

8. Be sure to update the FROM field with your dataset and table IDs.

```
%%bigquery --project ekabasandbox rmse
SELECT
  SQRT(mean_squared_error) AS rmse
FROM
  ML.EVALUATE(MODEL `btc_crypto.market_closing_model`,
    (
    WITH
      params AS (
          SELECT
            1 AS TRAIN,
            2 AS EVAL ),
          btc_market AS (
          SELECT
            CAST(open AS NUMERIC) AS open,
            CAST(high AS NUMERIC) AS high,
            CAST(low AS NUMERIC) AS low,
            CAST(close AS NUMERIC) AS close,
            CAST(spread AS NUMERIC) AS spread,
            CAST(LEAD(close, 1) OVER (PARTITION BY symbol ORDER BY
            symbol DESC) AS NUMERIC) AS next_day_close
          FROM
            `crypto_data.markets`,
            params
          WHERE
            symbol = 'BTC'
            AND MOD(ABS(FARM_FINGERPRINT(CAST(date AS STRING))),4) =
            params.EVAL )
        SELECT
          *
        FROM
          btc_market
        WHERE
          next_day_close IS NOT NULL ))

        rmse
    0   393.265715
```

9. Predict the next day's closing prices for the Bitcoin crypto-currency using the trained model. Be sure to update the FROM field with your dataset and table IDs.

```
%%bigquery --project ekabasandbox predict
SELECT
  *
FROM
```

```
ml.PREDICT(MODEL `btc_crypto.market_closing_model`,
  (
  WITH
    params AS (
    SELECT
      1 AS TRAIN,
      2 AS EVAL ),
    btc_market AS (
    SELECT
      CAST(close AS NUMERIC) AS close,
      date,
      CAST(open AS NUMERIC) AS open,
      CAST(high AS NUMERIC) AS high,
      CAST(low AS NUMERIC) AS low,
      CAST(spread AS NUMERIC) AS spread,
      CAST(LEAD(close, 1) OVER (PARTITION BY symbol ORDER BY
      symbol DESC) AS NUMERIC) AS next_day_close
    FROM
      `crypto_data.markets`,
      params
    WHERE
      symbol = 'BTC'
      AND MOD(ABS(FARM_FINGERPRINT(CAST(date AS STRING))),4) =
      params.EVAL )
  SELECT
    *
  FROM
    btc_market
  WHERE
    next_day_close IS NOT NULL ))
```

```
predict  predicted_next_day_close  close    date        open     high     low      spread  next_day_close

0        193.523361                116.99   2013-05-01  139      139.89   107.72   32.17   112.5
1        162.505189                112.5    2013-05-04  98.1     115      92.5     22.5    111.5
2        158.389055                111.5    2013-05-07  112.25   113.44   97.7     15.74   117.2
3        158.700481                117.2    2013-05-10  112.8    122      111.55   10.45   115
...      ...                       ...      ...         ...      ...      ...      ...     ...
388      4491.052680               4703.39  2017-08-31  4555.59  4736.05  4549.4   186.65  4597.12
389      4422.931411               4597.12  2017-09-06  4376.59  4617.25  4376.59  240.66  4122.94
390      4163.348876               4122.94  2017-09-10  4229.34  4245.44  3951.04  294.4   4161.27
391      4029.355833               4161.27  2017-09-11  4122.47  4261.67  4099.4   162.27  4130.81
...      ...                       ...      ...         ...      ...      ...      ...     ...
416      14723.798445              15201    2018-01-03  14978.2  15572.8  14844.5  728.3   15599.2
417      15421.170791              15599.2  2018-01-04  15270.7  15739.7  14522.2  1217.5  14595.4
```

This chapter provided an overview of working with Google BigQuery as a data warehouse and analytics platform on GCP. It covered working with BigQuery from Notebooks hosted on Google Colab or on GCP AI Instances and included how to work with BigQuery ML to build machine learning predictive models using SQL commands.

The next chapter will introduce Cloud Dataprep for visually exploring and transforming large datasets on GCP.

CHAPTER 39

# Google Cloud Dataprep

Google Cloud Dataprep is a managed cloud service for quick data exploration and transformation. Dataprep makes it easy to clean and transform large datasets for analysis. It is auto-scalable as it takes advantage of the distributed processing capabilities of Google Cloud Dataflow.

Typically Cloud Dataprep is aimed at easing the data preparation process. Datasets from real-world use cases are often messy and untidy. In this form, it cannot be used for downstream analytics or machine learning modeling. Hence, a large portion of the modeling process involves preparing and cleaning the data. Programming libraries earlier discussed like Pandas are centrally used for carrying out data preparation. However, Google Cloud Dataprep provides a simple visual interface for performing data cleaning. The ability to re-organize the dataset for modeling quickly without coding provides an instant appeal for Dataprep, as this can greatly speed up the time spent in data preparation as part of the overall modeling pipeline. The other good part is that Dataprep can work with petabyte scale data as it is built on a serverless infrastructure. Dataprep can be used for processing structured and unstructured datasets.

In this section, we'll go through a brief tour of Google Dataprep by using it to prepare our 'crypto_markets.csv' dataset already stored on Google Cloud Storage.

## Getting Started with Cloud Dataprep

From the GCP dashboard, click the triple dash at the top-left corner and scroll down to 'Dataprep' under the **BIG DATA** section as seen in Figure 39-1.

© Ekaba Bisong 2019
E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*,
https://doi.org/10.1007/978-1-4842-4470-8_39