Advanced Analytics with Transact-SQL

Exploring Hidden Patterns and Rules in Your Data

—

Dejan Sarka

Apress®

Advanced Analytics with Transact-SQL

Exploring Hidden Patterns and Rules in Your Data

Dejan Sarka

Apress®

Advanced Analytics with Transact-SQL: Exploring Hidden Patterns and Rules in Your Data

Dejan Sarka
Ljubjana, Slovenia

ISBN-13 (pbk): 978-1-4842-7172-8 ISBN-13 (electronic): 978-1-4842-7173-5
https://doi.org/10.1007/978-1-4842-7173-5

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Jonathan Gennick
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484271728. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author ix

About the Technical Reviewer xi

Acknowledgments xiii

Introduction xv

Part I: Statistics 1

Chapter 1: Descriptive Statistics 3

Variable Types 3

Demo Data 4

Frequency Distribution of Discrete Variables 8

Frequencies of Nominals 8

Frequencies of Ordinals 10

Descriptive Statistics for Continuous Variables 14

Centers of a Distribution 14

Measuring the Spread 19

Skewness and Kurtosis 24

Conclusion 28

Chapter 2: Associations Between Pairs of Variables 31

Associations Between Continuous Variables 35

Covariance 35

Correlation 38

Interpreting the Correlation 41

Associations Between Discrete Variables 43

Contingency Tables 44

Chi-Squared Test 51

Associations Between Discrete and Continuous Variables 56
Testing Continuous Variable Moments over a Discrete Variable 57
Analysis of Variance 58
Definite Integration 63
Conclusion 66

Part II: Data Preparation and Quality 67

Chapter 3: Data Preparation 69

Dealing with Missing Values 70
NULLs in T-SQL Functions 71
Handling NULLs 73
String Operations 79
Scalar String Functions 79
Aggregating and Splitting Strings 81
Derived Variables and Grouping Sets 84
Adding Computed Columns 85
Efficient Grouping 87
Data Normalization 91
Range and Z-score Normalization 91
Logistic and Hyperbolic Tangent Normalization 94
Recoding Variables 97
Converting Strings to Numerics 97
Discretizing Numerical Variables 100
Conclusion 105

Chapter 4: Data Quality and Information 107

Data Quality 108

Measuring Completeness 110

Finding Inaccurate Data 111

Measuring Data Quality over Time 118

Measuring the Information 123

Introducing Entropy 123

Mutual Information 127

Conditional Entropy 130

Conclusion 133

Part III: Dealing with Time 135

Chapter 5: Time-Oriented Data 137

Application and System Times 137

Inclusion Constraints 138

Demo Data 139

System-Versioned Tables and Issues 144

A Quick Introduction to System-Versioned Tables 144

Querying System-Versioned Tables Surprises 151

Optimizing Temporal Queries 165

Modifying the Filter Predicate 169

Using the Unpacked Form 172

Time Series 174

Moving Averages 176

Conclusion 180

Chapter 6: Time-Oriented Analyses 183
Demo Data 183
Exponential Moving Average 186
Calculating EMA Efficiently 188
Forecasting with EMA 191
ABC Analysis 194
Relational Division 195
Top Customers and Products 199
Duration of Loyalty 202
Survival Analysis 204
Hazard Analysis 208
Conclusion 211

Part IV: Data Science 213

Chapter 7: Data Mining 215
Demo Data 215
Linear Regression 218
Autoregression and Forecasting 220
Association Rules 225
Starting from the Negative Side 226
Frequency of Itemsets 229
Association Rules 232
Look-Alike Modeling 241
Training and Test Data Sets 242
Performing Predictions with LAM 250
Naïve Bayes 252
Training the NB Model 253
Performing Predictions with NB 255
Conclusion 261

Chapter 8: Text Mining 263

Demo Data 263

Introducing Full-Text Search 267

Full-Text Predicates 269

Full-Text Functions 272

Statistical Semantic Search 275

Quantitative Analysis 279

Analysis of Letters 279

Word Length Analysis 281

Advanced Analysis of Text 285

Term Extraction 285

Words Associations 287

Association Rules with Many Items 290

Conclusion 295

Index 297

About the Author

Dejan Sarka, MCT and Data Platform MVP, is an independent trainer and consultant who focuses on developing database and business intelligence applications, with more than 30 years of experience in this field. Besides projects, he spends about half of his time on training and mentoring. He is the founder of the Slovenian SQL Server and .NET Users Group. Sarka is the author or co-author of 19 books about databases and SQL Server. He has developed many courses and seminars for Microsoft, RADACAD, SolidQ, and Pluralsight.

About the Technical Reviewer

Ed Pollack has more than 20 years of experience in database and systems administration, developing a passion for performance optimization, database design, and wacky analytics. He has spoken at many user groups, data conferences, and summits. This led him to organize SQL Saturday Albany, which has become an annual event for New York's Capital Region. Sharing these experiences with the community is a passion. In his free time, Ed enjoys video games, backpacking, traveling, and cooking exceptionally spicy foods.

Acknowledgments

Many people helped me with this book, directly or indirectly. I am not mentioning the names explicitly because I could unintentionally omit some of them. However, I am pretty sure they are going to recognize themselves in the following text.

Thank you to my family, who understood that I could not spend so much time with them while I am writing the book.

Thank you to my friends, who were always encouraging me to finish the work.

Thank you to all the great people that work for or are engaged with Apress. Without your constant support, this book would probably never be finished.

Thank you to the reviewers. A well-reviewed book is as important as the writing itself. Thank you very much for your work.

Introduction

If you want to learn how to get information from your data with Transact-SQL, or the T-SQL language, this book is for you. It teaches you how to calculate statistical measures from descriptive statistics, including centers, spreads, skewness, and the kurtosis of a distribution, find the associations between pairs of variables, including calculating the linear regression formula, calculate the confidence level with definite integration, find the amount of information in your variables, and also do some machine learning or data science analysis, including predictive modeling and text mining.

The T-SQL language is in the latest editions of SQL Server, Azure SQL Database, and Azure Synapse Analytics. It has so many business intelligence (BI) improvements that it might become your primary analytic database system. Many database developers and administrators are already proficient with T-SQL. Occasionally they need to analyze the data with statistical or data science methods, but they do not want to or have time to learn a completely new language for these tasks. In addition, they need to analyze huge amounts of data, where specialized languages like R and Python might not be fast enough. SQL Server has been optimized for work with big datasets for decades.

To get the maximum out of these language constructs, you need to learn how to use them properly. This in-depth book shows extremely efficient statistical queries that use the window functions and are optimized through algorithms that use mathematical knowledge and creativity. The formulas and usage of those statistical procedures are explained as well.

Any serious analysis starts with data preparation. This book introduces some common data preparation tasks and shows how to implement them in T-SQL.

No analysis is good without good data quality. The book introduces data quality issues and shows how you can check for completeness and accuracy with T-SQL and measure improvements in data quality over time. It also shows how you can optimize queries with temporal data; for example, when you search for overlapping intervals. More advanced time-oriented information includes hazard and survival analysis.

Next, the book turns to data science. Some advanced algorithms can be implemented in T-SQL. You learn about the market basket analysis with association rules using different measures like support and confidence, and sequential market

basket analysis when there is a sequence in the basket. Then the book shows how to develop predictive models with a mixture of k-nearest neighbor and decision tree algorithms and Bayesian inference analysis.

Analyzing text, or text mining, is a popular topic. You can do a lot of text mining in pure T-SQL, and SQL Server can become a text mining engine. The book explains how to analyze text in multiple natural languages with pure T-SQL and features from full-text search (FTS).

In short, this book teaches you how to use T-SQL for

- statistical analysis
- data science methods
- text mining

Who Should Read This Book

Advanced Analytics with Transact-SQL is for database developers and database administrators who want to take their T-SQL programming skills to the max. It is for those who want to efficiently analyze huge amounts of data by using their existing knowledge of the T-SQL language. It is also for those who want to improve querying by learning new and original optimization techniques.

Assumptions

This book assumes that the reader already has good knowledge of the Transact-SQL language. A few years of coding experience is very welcome. A basic grasp of performance tuning and query optimization can help you better understand how the code works.

The Organization of This Book

There are eight chapters in this book, which are logically structured in four parts, each part with two chapters. The following is a brief description of the chapters.

Part I: Statistics Most advanced analytics starts with good old statistics. Sometimes statistical analysis might already provide the needed information, and sometimes statistics is only used in an overview of the data.

Chapter 1: Descriptive Statistics With descriptive statistics, the analyst gets an understanding of the distribution of a variable. One can analyze either continuous or discrete variables. Depending on the variable type, the analyst must choose the appropriate statistical measures.

Chapter 2: Associations Between Pairs of Variables When measuring associations between pairs of variables, there are three possibilities: both variables are continuous, both are discrete, or one is continuous and the other one is discrete. Based on the type of the variables, different measures of associations can be calculated. To calculate the statistical significance of associations, the calculation of the definite integrals is needed.

Part II: Data Quality and Preparation Before doing advanced analyses, it is crucial to understand the quality of the input data. A lot of additional work with appropriate data preparation is usually a big part of an analytics project in real life.

Chapter 3: Data Preparation There is no end to data preparation tasks. Some of the most common tasks include converting strings to numerical variables and discretizing continuous variables. Missing values are typical in analytical projects. Many times, derived variables help explain the values of a target variable more than the original input.

Chapter 4: Data Quality Garbage in, garbage out is a very old rule. Before doing advanced analyses, it is always recommendable to check for the data quality. Measuring improvements in data quality over time can help with understanding the factors that influence it.

Part III: Dealing with Time Queries that deal with time-oriented data are very frequent in analytical systems. Beyond a simple comparison of data in different time periods, much more complex problems and analyses can arise.

Chapter 5: Time-Oriented Data Understanding what kind of temporal data can appear in a database is very important. Some types of queries that deal with temporal data are hard to optimize. Data preparation of time series data has some own rules as well.

Chapter 6: Time-Oriented Analyses How long is a customer faithful to the supplier or the subscribed services and service provider? Which are the most hazardous days for losing a customer? What will be the sales amount in the next few periods? This chapter shows how to answer these questions with T-SQL.

Part IV: Data Science Some of the most advanced algorithms for analyzing data are many times mentioned with the term data science. Expressions as data mining, machine learning, and text mining are also very popular.

Chapter 7: Data Mining Every online or retail shop wants to know which products customers tend to buy together. Predicting a target discrete or continuous variable with few input variables is important in practically every type of business. This chapter introduces some of the most popular algorithms implemented with T-SQL.

Chapter 8: Text Mining The last chapter of the book introduced text mining with T-SQL. Text mining can include semantic search, term extraction, quantitative analysis of words and characters, and more. Data mining algorithms like association rules can also be used to understand analyzed text better.

System Requirements

You need the following software to run the code samples in this book.

- Microsoft SQL Server 2019 Developer of Enterprise edition, which is at www.microsoft.com/en-us/sql-server/sql-server-2019.
- Azure SQL Server Managed Instance, which is at https://azure.microsoft.com/en-us/services/azure-sql/sql-managed-instance/.
- Most of the code should run on the Azure SQL Database, which is at https://azure.microsoft.com/en-us/services/sql-database/.
- If you would like to try the code on unlimited resources, you can use Azure Synapse Analytics at https://azure.microsoft.com/en-us/services/synapse-analytics/.
- SQL Server Management Studio is the default client tool. You can download it for free at https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15.
- Another free client tool is the Azure Data Studio at https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver15.

- For demo data, you can find the AdventureWorks sample databases at https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms.
- Some demo data comes from R. I explain how to get it and show the R code for loading the demo data in SQL Server.
- You can download all the code used for the companion content in this book at https://github.com/Apress/adv-analytics-w-transact-sql.

Naming Conventions

When I create tables in SQL Server, I start with the column(s) that form the primary key, and I use Pascal case (e.g., FirstName) for the physical columns. For computed, typically aggregated columns from a query, I tend to use camel case (e.g., avgAmount). However, the book deals with the data from many sources. Demo data provided from Microsoft demo databases is not enough for all my examples. Two demo tables come from R. In R, the naming convention is not strict. I had a choice on how to proceed.

I decided to go with the original names when data comes from R, so the names of the columns in the table are all lowercase (e.g., carbrand). However, Microsoft demo data is far from perfect as well. Many dynamic management objects return all lowercase objects or even reserved keywords as the names of the columns. For example, in Chapter 8, I use two tabular functions provided by Microsoft, which return two columns with names [KEY] and [RANK]. Both in uppercase, and both are even reserved words in SQL, so they need to be enclosed in brackets. This is why sometimes the reader might get the impression that the naming convention is not good. I made an arbitrary decision, which I hope was the best in this situation.

PART I

Statistics

CHAPTER 1

Descriptive Statistics

Descriptive statistics summarize or quantitatively describe variables from a dataset. In a SQL Server table, a *dataset* is a set of the rows, or a *rowset*, that comes from a SQL Server table, view, or tabular expression. A *variable* is stored in a column of the rowset. In statistics, a variable is frequently called a *feature*.

When you analyze a variable, you first want to understand the *distribution* of its values. You can get a better understanding through graphical representation and descriptive statistics. Both are important. For most people, a graphical representation is easier to understand. However, with descriptive statistics, where you get information through numbers, it is simpler to analyze a lot of variables and compare their aggregated values; for example, their means and variability. You can always order numbers and quickly notice which variable has a higher mean, median, or other measure.

Transact-SQL is not very useful for graphing. Therefore, I focus on calculating descriptive statistics measures. I also include a few graphs, which I created with Power BI.

Variable Types

Before I calculate the summary values, I need to introduce the types of variables. Different types of variables require different calculations. The most basic division of the Variables are basically divided into two groups: discrete and continuous.

Discrete variables can only take a value from a limited pool. For example, there are only seven different or distinct values for the days of the week. Discrete variables can be further divided into two groups: nominal and ordinal.

If a value does not have a quantitative value (e.g., a label for a group), it is a nominal variable. For example, a variable that describes marital status could have three possible values: single, married, or divorced.

Discrete variables could also have an intrinsic order, which are called *ordinal* variables. If the values are represented as numbers, it is easy to notice the order. For example, evaluating a product purchased on a website could be expressed with numbers

D. Sarka, *Advanced Analytics with Transact-SQL*, https://doi.org/10.1007/978-1-4842-7173-5_1

from 1 to 7, where a higher number means greater satisfaction with the product. If the values of a variable are represented with strings, it is sometimes harder to notice the order. For example, education could be represented with strings, like high school degree, graduate degree, and so forth. You probably don't want to sort the values alphabetically because there is an order hidden in the values. With education, the order is defined through the years of schooling needed to get the degree.

If a discrete variable can take only two distinct values, it is a dichotomous variable called an *indicator*, a *flag*, or a *binary* variable. If the variable can only take a single value, it is a *constant*. Constants are not useful for analysis; there is no information in a constant. After all, variables are called *variables* because they introduce some variability.

Continuous variables can take a value from an unlimited, uncountable set of possible values. They are represented with integral or decimal numbers. They can be further divided into two classes: intervals or numerics (or true numerics).

Intervals are limited on the lower side, the upper side, or both sides. For example, temperature is an interval, limited with absolute zero on the lower side. On the other hand, true *numerics* have no limits on any side. For example, cashflow can be positive, negative, or zero.

It is not always completely clear if a variable is discrete or continuous. For example, the number of cars owned is an integer and can take any value between zero and infinite. You can use such variables in both ways—as discrete, when needed, or as continuous. For example, the naïve Bayes algorithm, which is explained in Chapter 7, uses only discrete variables so that you can treat the number of cars owned variable as discrete. But the linear regression algorithm, which is explained in the same chapter, uses only continuous variables, and you can treat the same variable as continuous.

Demo Data

I use a couple of demo datasets for the demos in this book. In this chapter, I use the *mtcars* demo dataset that comes from the R language; mtcars is an acronym for *MotorTrend* Car Road Tests. The dataset includes 32 *cases*, or rows, originally with 11 variables. For demo purposes, I add a few calculated variables. The data comes from a 1974 *MotorTrend* magazine and includes design and performance aspects for 32 cars, all 1973 and 1974 models. You can learn more about this dataset at www.rdocumentation.org/packages/datasets/versions/3.6.2/topics/mtcars.

I introduce variables when needed.

From SQL Server 2016, it is easy to execute R code inside SQL Server Database Engine. You can learn more about machine learning inside SQL Server with R or the Python language in official Microsoft documentation. A good introduction is at https://docs.microsoft.com/en-us/sql/machine-learning/sql-server-machine-learning-services?view=sql-server-ver15. Since this book is about T-SQL and not R, I will not spend more time explaining the R part of the code. I introduce the code that I used to import the mtcars dataset, with some additional calculated columns, in a SQL Server table.

First, you need to enable external scripts execution in SQL Server.

```
-- Configure SQL Server to enable external scripts
USE master;
EXEC sys.sp_configure 'show advanced options', 1;
RECONFIGURE
EXEC sys.sp_configure 'external scripts enabled', 1;
RECONFIGURE;
GO
```

I created a new table in the AdventureWorksDW2017 demo database, which is a Microsoft-provided demo database. I use the data from this database later in this book as well. You can find the AdventureWorks sample databases at https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms. For now, I won't spend more time on the content of this database. I just needed a database to create a table in, and because I use this database later, it seems like the best place for my first table with demo data. Listing 1-1 shows the T-SQL code for creating the demo table.

Listing 1-1. Creating the Demo Table

```
-- Create a new table in the AWDW database
USE AdventureWorksDW2017;
DROP TABLE IF EXISTS dbo.mtcars;
CREATE TABLE dbo.mtcars
(
 mpg numeric(8,2),
 cyl int,
 disp numeric(8,2),
```

```
 hp int,
 drat numeric(8,2),
 wt numeric(8,3),
 qsec numeric(8,2),
 vs int,
 am int,
 gear int,
 carb int,
 l100km numeric(8,2),
 dispcc numeric(8,2),
 kw numeric(8,2),
 weightkg numeric(8,2),
 transmission nvarchar(10),
 engine nvarchar(10),
 hpdescription nvarchar(10),
 carbrand nvarchar(20) PRIMARY KEY
)
GO
```

I want to discuss the naming conventions in this book. When I create tables in SQL Server, I start with the column(s) that form the primary key and use pascal case (e.g., FirstName) for the physical columns. For computed columns, typically aggregated columns from a query, I tend to use camel case (e.g., avgAmount). However, the book deals with data from many sources. Demo data provided from Microsoft demo databases is not enough for all of my examples. Two demo tables come from R. In R, the naming convention is not strict. I had a choice to make on how to proceed. I decided to go with the original names when data comes from R, so the names of the columns in the table in Listing 1-1 are all lowercase (e.g., carbrand).

Note Microsoft demo data is far from perfect. Many dynamic management objects return all lowercase objects or reserved keywords as the names of the columns. For example, in Chapter 8, I use two tabular functions by Microsoft that return two columns named [KEY] and [RANK]. Both are uppercase reserved words in SQL, so they need to be enclosed in brackets.

Now let's use the sys.sp_execute_external_script system stored procedure to execute the R code. Listing 1-2 shows how to execute the INSERT...EXECUTE T-SQL statement to get the R dataset in a SQL Server table.

Listing 1-2. Inserting R Data in the SQL Server Demo Table

```
-- Insert the mtcars dataset
INSERT INTO dbo.mtcars
EXECUTE sys.sp_execute_external_script
 @language=N'R',
 @script = N'
data("mtcars")
mtcars$l100km = round(235.214583 / mtcars$mpg, 2)
mtcars$dispcc = round(mtcars$disp * 16.38706, 2)
mtcars$kw = round(mtcars$hp * 0.7457, 2)
mtcars$weightkg = round(mtcars$wt * 1000 * 0.453592, 2)
mtcars$transmission = ifelse(mtcars$am == 0,
                             "Automatic", "Manual")
mtcars$engine = ifelse(mtcars$vs == 0,
                       "V-shape", "Straight")
mtcars$hpdescription =
  factor(ifelse(mtcars$hp > 175, "Strong",
                ifelse(mtcars$hp < 100, "Weak", "Medium")),
         order = TRUE,
         levels = c("Weak", "Medium", "Strong"))
mtcars$carbrand = row.names(mtcars)
 ',
 @output_data_1_name = N'mtcars';
GO
```

You can check if the demo data successfully imported with a simple SELECT statement.

```
SELECT *
FROM dbo.mtcars;
```

When the demo data is loaded, let's start analyzing it.

Frequency Distribution of Discrete Variables

You usually represent the distribution of a discrete variable with frequency distribution or *frequencies*. In the simplest example, you can calculate only the values' count. You can also express these value counts as percentages of the total number of rows or cases.

Frequencies of Nominals

The following is a simple example of calculating the counts and percentages for the *transmission* variable, which shows the transmission type.

```
-- Simple, nominals
SELECT c.transmission,
 COUNT(c.transmission) AS AbsFreq,
 CAST(ROUND(100. * (COUNT(c.transmission)) /
       (SELECT COUNT(*) FROM mtcars), 0) AS int) AS AbsPerc
FROM dbo.mtcars AS c
GROUP BY c.transmission;
```

The following is the result.

```
transmission AbsFreq     AbsPerc
------------ ----------- -----------
Automatic    19          59
Manual       13          41
```

I used a simple `GROUP BY` clause of the `SELECT` statement and the `COUNT()` aggregate function. Graphically, you can represent the distribution with vertical or horizontal bar charts. Figure 1-1 shows the bar charts for three variables from the mtcars dataset, created with Power BI.

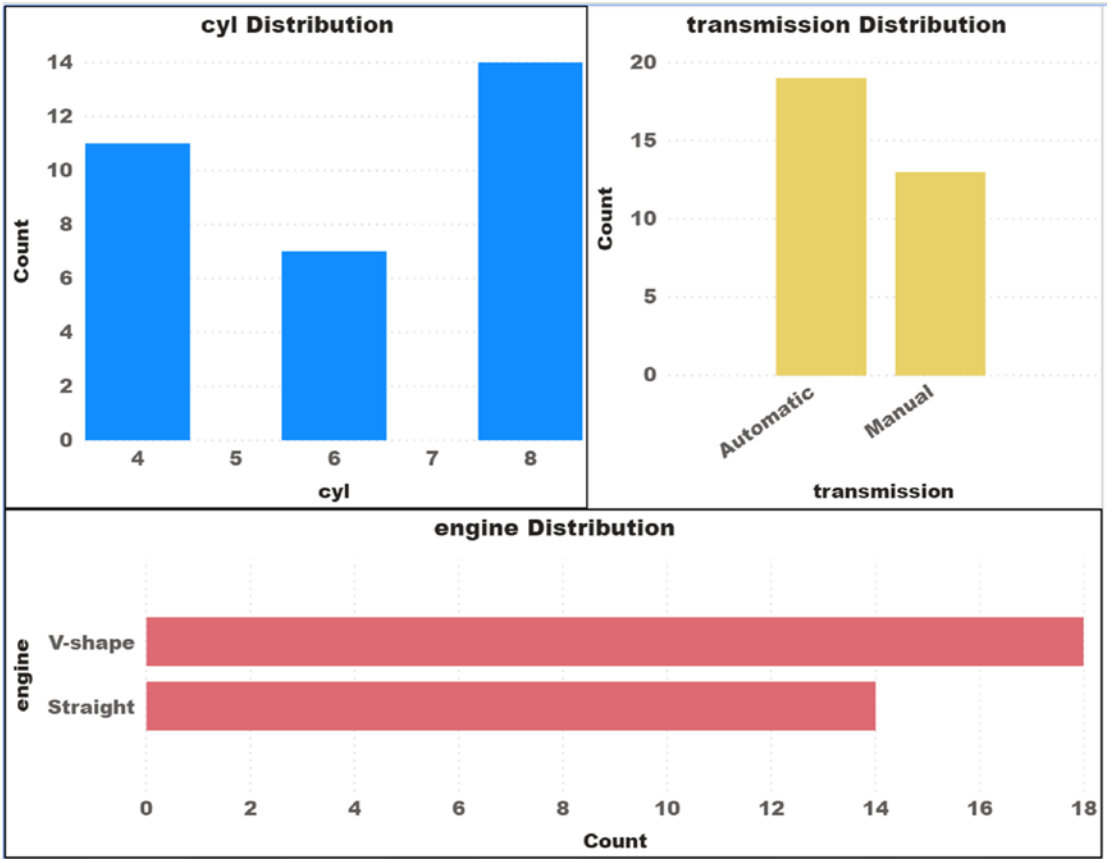

Figure 1-1. *Bar charts for discrete variables*

You can see the distribution of the `transmission`, `engine`, and `cyl` variables. The `cyl` variable is represented with the numbers 4, 6, and 8, which represent the number of engine cylinders. Can you create a bar chart with T-SQL? You can use the percentage number as a parameter to the `REPLICATE()` function and mimic the horizontal bar chart, or a horizontal histogram, as the following code shows.

```
WITH freqCTE AS
 (
SELECT c.transmission,
 COUNT(c.transmission) AS AbsFreq,
 CAST(ROUND(100. * (COUNT(c.transmission)) /
       (SELECT COUNT(*) FROM mtcars), 0) AS int) AS AbsPerc
```

```
FROM dbo.mtcars AS c
GROUP BY c.transmission
)
SELECT transmission,
 AbsFreq,
 AbsPerc,
 CAST(REPLICATE('*', AbsPerc) AS varchar(50)) AS Histogram
FROM freqCTE;
```

I used a common table expression to enclose the first query, which calculated the counts and the percentages, and then added the horizontal bars in the outer query. Figure 1-2 shows the result.

	transmission	AbsFreq	AbsPerc	Histogram
1	Automatic	19	59	**********
2	Manual	13	41	**********

Figure 1-2. *Counts with a horizontal bar*

For nominal variables, this is usually all that you calculate. For ordinals, you can also calculate running totals.

Frequencies of Ordinals

Ordinals have intrinsic order. When you sort the values in the correct order, it makes sense to also calculate the running totals. What is the total count of cases up to some specific value? What is the running total of percentages? You can use the T_SQL *window aggregate functions* to calculate the running totals. Listing 1-3 shows the calculation for the `cyl` variable.

Listing 1-3. Frequencies of an Ordinal Variable

```
-- Ordinals - simple with numerics
WITH frequency AS
(
SELECT v.cyl,
 COUNT(v.cyl) AS AbsFreq,
 CAST(ROUND(100. * (COUNT(v.cyl)) /
```

```
       (SELECT COUNT(*) FROM dbo.mtcars), 0) AS int) AS AbsPerc
FROM dbo.mtcars AS v
GROUP BY v.cyl
)
SELECT cyl,
 AbsFreq,
 SUM(AbsFreq)
  OVER(ORDER BY cyl
       ROWS BETWEEN UNBOUNDED PRECEDING
       AND CURRENT ROW) AS CumFreq,
 AbsPerc,
 SUM(AbsPerc)
  OVER(ORDER BY cyl
       ROWS BETWEEN UNBOUNDED PRECEDING
       AND CURRENT ROW) AS CumPerc,
 CAST(REPLICATE('*', AbsPerc) AS varchar(50)) AS Histogram
FROM frequency
ORDER BY cyl;
```

The query returns the result shown in Figure 1-3.

	cyl	AbsFreq	CumFreq	AbsPerc	CumPerc	Histogram
1	4	11	11	34	34	**********
2	6	7	18	22	56	**********
3	8	14	32	44	100	**********

Figure 1-3. *Frequencies of an ordinal variable*

Note If you are not familiar with the T-SQL window functions and the OVER() clause, please refer to the official SQL Server documentation at https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql?view=sql-server-ver15.

Ordering by the cyl variable was simple because the values are represented with integral numbers, and the order is automatically correct. But if an ordinal is represented with strings, you need to be careful with the proper order. You probably do not want to use alphabetical order.

For a demo, I created (already in the R code) a hpdescription derived variable (originally stored in the hp continuous variable), which shows engine horsepower in three classes: weak, medium, and strong. The following query incorrectly returns the result in alphabetical order.

```
-- Ordinals - incorrect order with strings
WITH frequency AS
(
SELECT v.hpdescription,
 COUNT(v.hpdescription) AS AbsFreq,
 CAST(ROUND(100. * (COUNT(v.hpdescription)) /
       (SELECT COUNT(*) FROM dbo.mtcars), 0) AS int) AS AbsPerc
FROM dbo.mtcars AS v
GROUP BY v.hpdescription
)
SELECT hpdescription,
 AbsFreq,
 SUM(AbsFreq)
  OVER(ORDER BY hpdescription
       ROWS BETWEEN UNBOUNDED PRECEDING
       AND CURRENT ROW) AS CumFreq,
 AbsPerc,
 SUM(AbsPerc)
  OVER(ORDER BY hpdescription
       ROWS BETWEEN UNBOUNDED PRECEDING
       AND CURRENT ROW) AS CumPerc,
 CAST(REPLICATE('*', AbsPerc) AS varchar(50)) AS Histogram
FROM frequency
ORDER BY hpdescription;
```

The results of this query are shown in Figure 1-4.

	hpdescription	AbsFreq	CumFreq	AbsPerc	CumPerc	Histogram
1	Medium	13	13	41	41	**********
2	Strong	10	23	31	72	********
3	Weak	9	32	28	100	*******

Figure 1-4. *Frequencies of the hpdescription variable with incorrect order*

You can use the CASE T-SQL expression to change the strings and include proper ordering with numbers at the beginning of the string. Listing 1-4 shows the calculation of the frequencies of a string ordinal with proper ordering.

Listing 1-4. Frequencies of an Ordinal with Proper Ordering

```
-- Ordinals - correct order
WITH frequency AS
(
SELECT
 CASE v.hpdescription
         WHEN N'Weak' THEN N'1 - Weak'
         WHEN N'Medium' THEN N'2 - Medium'
         WHEN N'Strong' THEN N'3 - Strong'
       END AS hpdescriptionord,
 COUNT(v.hpdescription) AS AbsFreq,
 CAST(ROUND(100. * (COUNT(v.hpdescription)) /
       (SELECT COUNT(*) FROM dbo.mtcars), 0) AS int) AS AbsPerc
FROM dbo.mtcars AS v
GROUP BY v.hpdescription
)
SELECT hpdescriptionord,
 AbsFreq,
 SUM(AbsFreq)
  OVER(ORDER BY hpdescriptionord
       ROWS BETWEEN UNBOUNDED PRECEDING
       AND CURRENT ROW) AS CumFreq,
 AbsPerc,
 SUM(AbsPerc)
  OVER(ORDER BY hpdescriptionord
       ROWS BETWEEN UNBOUNDED PRECEDING
       AND CURRENT ROW) AS CumPerc,
 CAST(REPLICATE('*', AbsPerc) AS varchar(50)) AS Histogram
FROM frequency
ORDER BY hpdescriptionord;
```

Figure 1-5 shows the result of the query from Listing 1-4.

	hpdescriptionord	AbsFreq	CumFreq	AbsPerc	CumPerc	Histogram
1	1 - Weak	9	9	28	28	**********
2	2 - Medium	13	22	41	69	**********
3	3 - Strong	10	32	31	100	**********

Figure 1-5. *Frequencies of the hpdescription ordinal variable*

With frequencies, I covered discrete variables. Now let's calculate some descriptive statistics for continuous variables.

Descriptive Statistics for Continuous Variables

You can calculate many statistical values for the distribution of a continuous variable. Next, I show you the calculation for the centers of distribution, spread, skewness, and "tailedness." I also explain the mathematical formulas for calculation and the meaning of the measures. These measures help describe the distribution of a continuous variable without graphs.

Centers of a Distribution

The most known and the most abused statistical measure is the *mean* or the average of a *variable*. How many times have you heard or read about "the average …"? Many times, this expression makes no sense, although it looks smart to use it. Let's discuss an example.

Take a group of random people in a bar. For the sake of the example, let's say they are all local people from the country where the bar is located. You want to estimate the wealth of these people.

The mean value is also called the *expected* value. It is used as the *estimator* for the target variable, in this case, wealth. It all depends on how you calculate the mean. You can ask every person in the group her or his income and then calculate the group's mean. This is the *sample* mean.

Your group is a sample of the broader population. You could also calculate the mean for the whole country. This would be the *population* mean. The population mean is a good estimator for the group. However, the sample mean could be very far from the actual wealth of the majority of people in the group. Imagine that there are 20 people in

the group, including one extremely rich person worth more than $20 billion. The sample mean would be more than a billion dollars, which seems like a group of billionaires are in the bar. This could be far from the truth.

Extreme values, especially if they are rare, are called *outliers*. Outliers can have a big impact on the mean value. This is clear from the formula for the mean.

$$\mu = \frac{1}{n} * \sum_{i=1}^{n} v_i$$

Each value, v_i, is part of the calculation of the mean, μ. A value of 100 adds a hundred times more to the mean than the value of 1. The mean of the sample is rarely useful if it is the only value you are measuring. The calculation of the mean involves every value on the first degree. That is why the mean is also called the *first population moment*.

Apparently, we need other measures. A very useful measure is the median. First, you order the rows (or the cases) by the target variable. Then, you split the population into two halves, or two tiles. The median is the value in the middle. If you have an odd number of rows, it is a single value because there is exactly one row in the middle. If you have an even number of rows, there are two rows in the middle. You can calculate the median as the lower of these two values, which is then the lower median.

Similarly, the upper median is the higher of the two values in the middle. I prefer to use the average of the two values in the middle, called the *median*. You can also split the cases into more than two tiles. If you split it into four tiles, the tiles are called *quartiles*. The median is the value on the second quartile, sometimes marked as Q_2.

In T-SQL, you can use the `PERCENTILE_CONT()` window analytic function to calculate the median and the `PERCENTILE_DISC()` function to calculate the lower median. The following code shows how these two functions work on a small sample dataset that consists of values 1, 2, 3, and 4.

```
-- Difference between PERCENTILE_CONT() and PERCENTILE_DISC()
SELECT DISTINCT       -- can also use TOP (1)
 PERCENTILE_DISC(0.5) WITHIN GROUP
  (ORDER BY val) OVER () AS MedianDisc,
 PERCENTILE_CONT(0.5) WITHIN GROUP
  (ORDER BY val) OVER () AS MedianCont
FROM (VALUES (1), (2), (3), (4)) AS a(val);
```