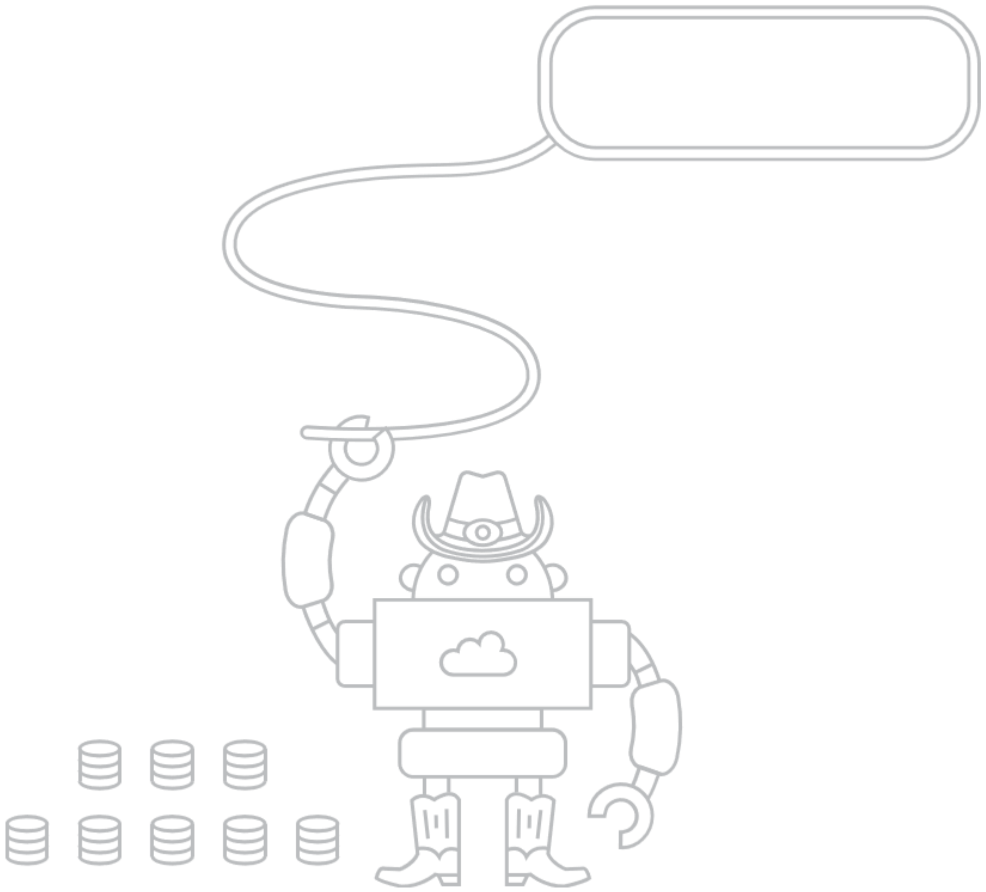

A Getting Started Guide
for the Oct 2016
release of Pendleton.

MICROSOFT PROJECT CODENAME “PENDLETON”

Getting Started Guide - Oct 2016

Contents

What’s New

This is the first release of this document so everything is new and there are no updates from previous releases of the document.

Introduction

Welcome to the Pendleton Getting Started Guide. This short tour of Pendleton gets you acquainted with how it works, demonstrates what it can do and accelerates your ability to build data preparation solutions.

Pendleton provides a set of flexible and scalable tools to help you explore, discover, understand and fix problems in your data. It allows you to consume data in many forms and to transform that data into new forms that are better suited for your usage.

Pendleton is a client app that is installed locally onto your machine. It installs a core application and then prompts you to allow the installer/app launcher to download dependencies.

The design time runtime uses Python and depends on various Python libraries including Pandas. In Windows installations, the Python libraries needed by Pendleton are all installed to a private application directory to avoid a clash with other versions of Python and its libraries that you may already be using.

After you install Pendleton on Windows there should be a shortcut on the desktop that can be found by pressing the Windows key and then starting to type Pendleton. On OS X/macOS there will be a new application.

When you first launch Pendleton you are taken to the home screen where you can read this document, watch videos on how to use Pendleton, get updated information about Pendleton, open existing Packages, create a new Package, or open a Data Source.

You can provide positive or negative feedback on Pendleton at any time via the happy and unhappy face icons in the top right of the main app. If possible, please provide your email address so the engineering team can follow up directly if needed.

Next to the happy and unhappy icons is a bell icon. The bell icon is gray by default. The bell icon turns red when an error occurs. You can click on the red icon to get more details about the error. A green bell icon indicates that an update is available and is currently downloading. Pendleton is a self-updating application. To successfully apply an update, first let the download finish and then exit Pendleton cleanly. Wait for a few minutes and then re-launch the application to see the new build.

Building Blocks of Pendleton

The Package

The Package is the primary container for your work. A Package is the artifact that is saved to disk and loaded from disk. While working inside the client, the Package is constantly AutoSaved in the background. It is given a default name if you do not choose a name through a Save action.

A Package may be exported to Python.

A Package contains 1 or more Dataflows.

The Dataflow

A Dataflow has a source and optional Transforms which are arranged through a series of Steps and optional destinations. When you click a Step in the UI, all the sources and Transforms prior to that Step are executed so that the data is represented as of the end of the Step. Steps can be added, moved and deleted within a Dataflow through the Step List.

The Step List on the right side of the client can be opened and closed to provide more screen space.

A Dataflow list on the left hand side of the client can be opened and closed to provide more screen space and to navigate between different Dataflows. Multiple Dataflows can exist in the UI at a time.

The Source

Appendix 2 provides the current list of supported sources. Each source has a different user experience to allow it to be configured. The source produces a “rectangular”/tabular view of the data. If the source data originally has a “ragged right”, then this is normalized to be “rectangular”.

The Transform

Appendix 3 provides the current list of supported Transforms. Each Transform has its own UI and behavior(s). Transforms consume data in a given format, perform some operation on the data (such as changing the data type) and then produce data in the new format. Chaining several Transforms together via Steps in the Dataflow is the core of Pendleton functionality.

The Inspector

Appendix 4 provides the current list of supported Inspectors. Inspectors are visualizations of the data. Inspectors help you understand the data so you can decide which actions (Transforms) you need to take to make the data better suited for your purpose. Some Inspectors support actions that generate Transforms. For example, the Value Count Inspector allows you to select a Value and then apply a filter to include that Value or to Exclude that Value. Inspectors can also provide context for Transforms. For example, selecting one or more columns changes the possible Transforms which can be applied.

A column may have multiple Inspectors at any point in time (e.g. Column Statistics and a Histogram). There can also be multiple instances of an Inspector against different columns. For example, all numeric columns could have Histograms at the same time.

The Inspectors appear in the Profiling Well at the bottom of the UI and also in the main view. The default Inspector is the Grid. Any Inspector can be expanded into the main view. In this scenario, whichever Inspector was already in the main view minimizes to the well. Inspectors can be configured by clicking on the pencil icon. Once minimized into the well, an inspector can be moved left to support reordering.

Some Inspectors support the option for a "Halo". This is the capability to remember the value/state before the last Transform was applied. The old value is displayed as a background grey with the current value in the foreground. This allows for easy comparison of the impact of a Transform.

The Destination

Appendix 5 provides the current list of supported destinations. A given Dataflow can have multiple destinations. Each destination has a different user experience to allow it to be configured. The destination consumes data in a "rectangular"/tabular format and writes it out to some location in a given format.

Using Pendleton

Pendleton assumes a basic 5 step methodology/approach to data preparation.

Step 1: Ingestion

You can import data into Pendleton through either the Open Data Source option on the home page or the Open Data Source option on the Dataflow menu. Once you select a source and sample size (the default is currently 10,000 rows), select the "Add Data Source" button above the preview grid to create a new Dataflow. You are now ready for data preparation via steps 2-4.

Step 2: Understand/Profile the Data

The first action is to look at the Data Quality Bar at the top of each column. For each column, green indicates the rows that have values. Grey indicates the rows with a missing value, null etc. There are tool tips to tell the exact numbers of rows in each of the 2 buckets. The UI is scaled logarithmically, so always check the actual numbers after using the UI to get a rough feel for the volume of missing data.

The next action is to use the various Inspectors from the Inspector's menu and also the grid to develop an understanding of the characteristics of the data and to start formulating hypotheses about the data preparation required for further analysis.

It's likely that several Inspectors across several columns will be needed to understand the data. You can scroll through various Inspectors in the Profiling Well. Within the well, you can also move Inspectors to the head of the list in order to see them in the immediately viewable area.

Different inspectors are provided for continuous vs categorical variables/columns. The Inspector menu enables and disables options depending on the type of variables/columns you have.

When working with very wide datasets that have many columns a pragmatic approach of working with subsets is advisable. This approach includes focusing on a small number of columns (e.g. 5-10), preparing them and then working through the remaining columns.

Step 3: Transform the Data

Transform(s) change the data and allow the execution of the data to support the current working hypothesis. Transforms appear as Steps in the Step List on the right hand side. It is possible to “time travel” through the Step List, to go backwards and forwards to any arbitrary point in the Step List simply by clicking on that Step.

A green icon to the left of a given Step indicates that it has run and the data reflects the execution of that Transform. A vertical bar to the left of the Step indicates the current state of the data in the Inspectors.

It is advisable to make small frequent changes to the data and to validate (Step 4) after each change as the hypothesis evolves.

Step 4: Verify the impact of the transformation.

Decide if the hypothesis was correct. If correct then develop the next hypothesis and repeat steps 2-3 for the new one. If incorrect then undo the last transformation and develop a new hypothesis and repeat steps 2-3.

The primary way to determine if the Transform had the right impact is to use the Inspectors. You can either use existing Inspectors with their Halo effect on or you can simply launch multiple Inspectors to view the data at given points in time.

To undo a Transformation, go the Steps List on the right hand side of the UI. The Steps List panel may need to be popped back out. To do this, click the double chevron pointing left. In the panel, select the Transform that was just executed (it is likely the last one in the list). Select the drop down on the right hand side of the UI block. Select either “Edit” to make changes or “Delete” to remove the Transform from the Steps List and the Dataflow.

Step 5: Output

A Dataflow can have many outputs. Select the Write CSV Transform to write the data out to a local file.

Appendix 1 – Supported client platforms for this release

Name	Description
Windows	Support for Windows 10 and higher
OS X/macOS	OS X 10.11 “El Capitan”/macOS Sierra

Appendix 2 – Supported sources of data for this release

Name	Description
Local CSV File	Read a Comma Separated Value file from a local or network mapped location. **Options:** Separator Comment Headers Decimal Symbol File Encoding Lines To Skip Sample Size
Local TSV File	Read a Tab Separated Value file from a local or network mapped location. **Options:** Comment Headers File Encoding Lines To Skip Sample Size
Local Excel File	Read an Excel file (.xlsx, .xls) from a local or network mapped location. **Options:** Sheet Name Headers Lines To Skip
Azure BLOB	Read .CSV file from an Azure BLOB Storage Account Azure Account Azure Subscription Storage Container All settings for local CSV file
Local JSON File	Read a JSON file from local or networked mapped location. Note the JSON will be "flattened" automatically **Options:** None

Appendix 3 – Supported transforms for this release

Name	Description/Usage
Add Column by Example	Add a new blank column, type an example of what you wish to see in the column (assuming it is derived from other columns) and the “By Example” technology will attempt to fill in all the other cells in the column. For complicated examples it may be necessary to provide more than one example, simple select another cell and type another example. If no columns are selected when this transform is created, then the “By Example” technology will use all the cells in the row which you type the example into to determine what your example means. If you select a subset of columns (one or more) then the “By Example” technology will use only those columns, this can sometimes lead to a better, more precise set of data being generated.
Add Column	Add a new blank column that uses a Python expression to derive it. The Python code has access to all other cells in the row so they can be referenced in the expression.
Split Column by Example	Takes an existing column and using the “By Example” engine attempts to split that column into n other columns. It is possible to run the Auto-Split on the subsequent generated columns
Replace Values	Choose a value to be replaced. Enter a value to use in the replacement. Match Entire Cell Contents. Special Character handling.
Replace NA Values	Allows the various different incarnations of NA (N/A, NA, null, NaN,) or empty strings with a null to make them consistent. Supports 1:n columns.
Trim Whitespace	Remove leading and trailing whitespaces.
Handle Missing Values	For a given column or set of columns find all the rows that have missing values and then delete all those rows or replace the missing value with a fixed value.
Rename Column	Changes the name of the column
Adjust Precision	Allows the number of decimal places for a numeric column to be set
Rename Column	
Remove Column	Removes a column from that point forward in the step list
Convert Field Type to Numeric	Change the column type
Convert Field Type to Date	Change the column type
Convert Field Type to Boolean	Change the column type
Convert Field Type to String	Change the column type
Use first row as headers	
Filter	Allows a Python row level filter to be written

	See Appendix 6 for more info
Join	Join 2 Dataflows together via a single column, provides the ability to handle the success rows and the left and right failing rows separately.
Summarize	Select a column(s) and compute aggregates for that combination of unique values. Aggregates supported: COUNT, SUM, MIN, MAX, MEAN, VARIANCE, STANDARD DEVIATION Analogous to an ANSI-SQL GROUP BY
Distinct Rows	For a given column(s) select one row to represent each unique value (or combination of unique values) for that column(s). Analogous to an ANSI-SQL SELECT DISTINCT
Sort	Reorder the dataset Analogous to an ANSI-SQL ORDER BY
Transform Frame	Write Python code to perform a transformation on the entire table. This transform allows multiple columns to be added and for the entire table to be operated on. See Appendix 7 for more info

Appendix 4 – Supported inspectors for this release

Name	Description
Column Statistics	For numeric columns provides a variety of different stats about the column including; Min, Max, Median, Average, Standard Deviation, Lower Quartile, Upper Quartile **Options:** None
Histogram	Histogram of a numeric column. Default number of buckets is calculated using Scott’s Rule. **Options:** Number of Buckets Halo Kernel Density Overlay **Actions:** Select buckets and Filter in or Filter out via toolbar.
Value Counts	Frequency table for string columns. **Options:** Number of values to include Ascending/Descending Halo **Actions:** Select values and Filter in or Filter out via toolbar.
Box Plot	A box whisker plot of a numeric column **Options:** Group By Column
Scatter Plot	A scatter plot for 2 numeric columns **Options:** Group By Column Sample Size
Time Series	A line graph with time awareness on the X- Axis **Options:** Sample Size **Actions:** Group By Column Select values and Filter in or Filter out via toolbar
Map	A map with points plotted assuming latitude and longitude have been specified. Latitude must be selected first.

	Options: Clustering on/off **Actions:** Latitude column Longitude column Group By Column Select (by CTRL Clicking and Dragging) and Filter in or Filter out via the toolbar.

Appendix 5 – Supported destinations of data for this release

Name	Description
Local CSV File	Write to a Std Comma Separated Value file on a local or network mapped drive. **Options:** Missing Value Replacement File Encoding Quoting Line Ending

Appendix 6 – Samples of filter expressions (Python)

Name	Description

	`row.Col2 > 4`
	`row.Col1 == 'Good' and row.Col2 == 1`
	`pd.isnull(row.Col1)`
	
	The following Python imports are provided
	`math, numbers, datetime, re, pandas (aliased as pd), numpy(aliased as np)`

Appendix 7 – Samples of custom transforms (Python)

Name	Description

Add Column	The following formats for referencing a column produce the same results
	`row.Col1 + row.Col2 is the same as row["Col1"] + row["Col2"]`
	
	If the value in Col1 is less then 4 then the new column should have a value of 1 else it has the value 2
	`1 if row.Col1 < 4 else 2`
	
	Insert the current date and time
	`datetime.datetime.now()`
	
	Typecast 2 numbers and then divide them and remove 1
	`float(row.Col1) / float(row.Col2 - 1)`
	
	If the Col1 contains a null then mark the new column as Bad otherwise mark it as Good
	`'Bad' if pandas.isnull(row.Col1) else 'Good'`
	
	New column is a logn of Col1
	`np.log(row.Col1)`

Transform Frame	Creates a column on the fly(city2) and reconciles multiple different versions of San Francisco to one. `df.loc[(df['city'] == 'San Francisco') | (df['city'] == 'SF') | (df['city'] == 'S.F.') | (df['city'] == 'SAN FRANCISCO'), 'city2'] = 'San Francisco'`
Transform Frame	Creates a new frame with the first and last aggs computed for the score column grouped by risk_category `df = df.groupby(['risk_category'])['Score'].agg(['first','last'])`