Demo: Data Wrangler Titanic Dataset Walkthrough

The following sections provide a walkthrough to help you get started using Data Wrangler. This walkthrough assumes that you have already followed the steps in Access Data Wrangler and have a new data flow file open that you intend to use for the demo. You may want to rename this .flow file to something similar to `titanic-demo.flow`.

This walkthrough uses the Titanic dataset. This data set contains the survival status, age, gender, and class (which serves as a proxy for economic status) of passengers aboard the maiden voyage of the *RMS Titanic* in 1912.

In this tutorial, you perform the following steps.

- Upload the Titanic dataset to Amazon Simple Storage Service (Amazon S3), and then import this dataset into Data Wrangler.
- Analyze this dataset using Data Wrangler analyses.
- Define a data flow using Data Wrangler data transforms.
- Export your flow to a Jupyter Notebook that you can use to create a Data Wrangler job.
- Process your data, and kick off a SageMaker training job to train a XGBoost Binary Classifier.

Upload Dataset to S3 and Import

To get started, download the Titanic dataset and upload it to an Amazon S3 (Amazon S3) bucket in the AWS Region in which you want to complete this demo.

If you are a new user of Amazon S3, you can do this using drag and drop in the Amazon S3 console. To learn how, see Uploading Files and Folders by Using Drag and Drop in the Amazon Simple Storage Service User Guide.

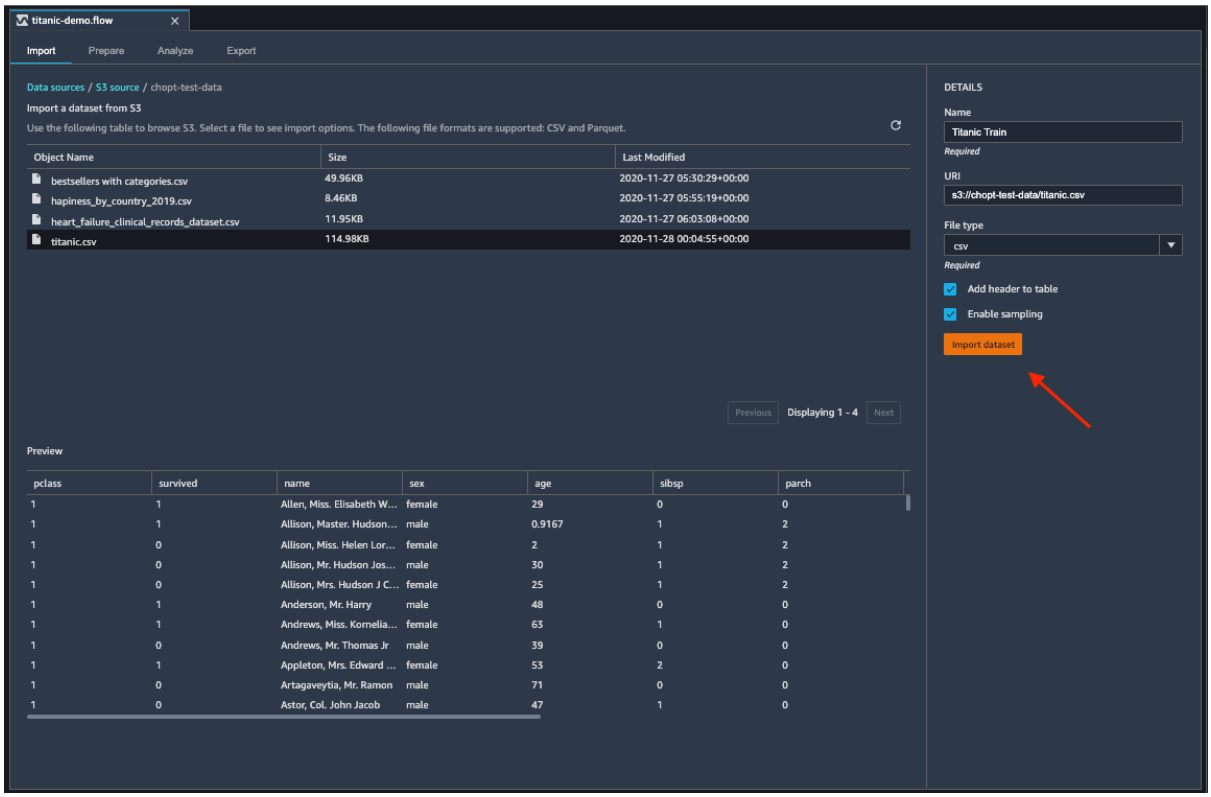
Important

Upload your dataset to an S3 bucket in the same AWS Region you want to use to complete this demo.

When your dataset has been successfully uploaded to Amazon S3, it you can import it into Data Wrangler.

Import the Titanic dataset to Data Wrangler

1. Select the **Import** tab in your Data Wrangler flow file.
2. Select **Amazon S3**.
3. Use the **Import a dataset from S3** table to find the bucket to which you added the Titanic dataset. Choose the Titanic dataset CSV file to open the **Details** pane.
4. Under **Details**, the **File type** should be CSV. Choose **Add header to table** to specify that the first row of the dataset is a header. You can also name the dataset something more friendly, such as `Titanic-train`.
5. Select **Import dataset**.



When your dataset is imported into Data Wrangler, it appears in your data flow. You can view your data flow at any time by selecting the **Data Flow** tab. You can double click on a node to enter the node detail view, which allows you to add transformations or analysis; otherwise, you can use the plus icon to navigate for a

quick navigation. In the next section, you use this data flow to add analysis and transform steps.

Data Flow

In the data flow section, the only steps in the data flow are your recently imported dataset and a **Data type** step. After applying transformations, you can come back to this tab see what the data flow looks like. Now, add some basic transformations under the **Prepare** and **Analyze** tabs.

Prepare and Visualize

Data Wrangler has built-in transformations and visualizations that you can use to analyze, clean, and transform your data.

The **Data** tab of the node detail view lists all built-in transformations in the right panel, which also contains an area in which you can add custom transformations. The following use case showcases how to use these transformations.

Data Exploration

First, create a table summary of the data using an analysis. Do the following:

1. Choose the **+** next to the **Data type** step in your data flow and select **Add analysis**.
2. In the **Analysis** area, select **Table summary** from the dropdown list.
3. Give the table summary a **Name**.
4. Select **Preview** to preview the table that will be created.
5. Choose **Create** to save it to your data flow. It appears under **All Analyses**.

Using the statistics you see, you can make observations similar to the following about this dataset:

- Fare average (mean) is around $33, while the max is over $500. This column likely has outliers.
- This dataset uses *?* to indicate missing values. A number of columns have missing values: *cabin*, *embarked*, and *home.dest*
- The age category is missing over 250 values.

Choose **Prepare** to go back to the data flow. Next, clean your data using the insights gained from these stats.

Drop Unused Columns

Using the analysis from the previous section, clean up the dataset to prepare it for training. To add a new transform to your data flow, choose **+** next to the **Data type** step in your data flow and choose **Add transform**.

First, drop columns that you don't want to use for training. You can use Pandas data analysis library to do this, or you can use one of the built-in transforms.

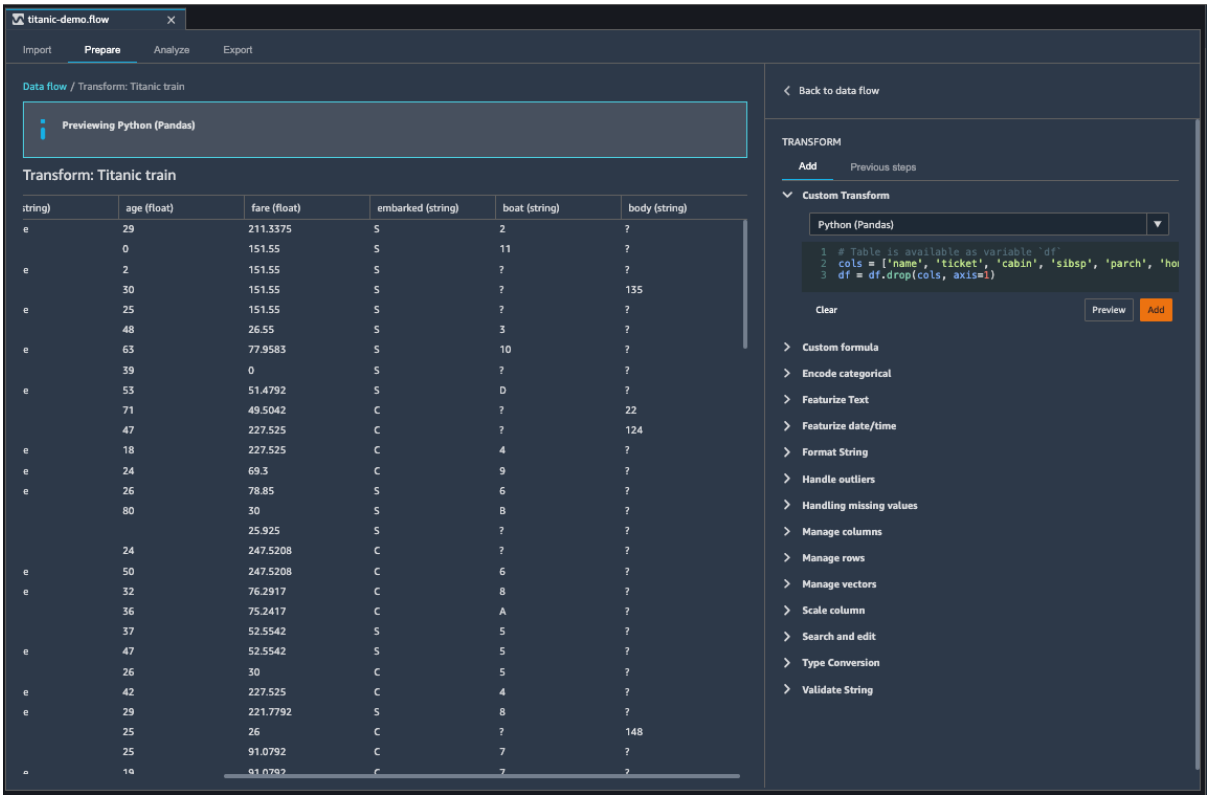
To do this using Pandas, follow these steps.

1. In the **Custom Transform** section, select **Python (Pandas)** from the dropdown list.
2. Enter the following in the code box.

```
cols = ['name', 'ticket', 'cabin', 'sibsp', 'parch',
'home.dest','boat', 'body']
```

```
df = df.drop(cols, axis=1)
```

4. Choose **Preview** to preview the change and then choose **Add** to add the transformation.



To use the built-in transformations, do the following:

1. Choose **Manage columns** from the right panel.
2. For **Input column**, choose **cabin**, and choose **Preview**.
3. Verify that the **cabin** column has been dropped, then choose **Add**.
4. Repeat these steps for the following columns: **ticket**, **name**, **sibsp**, **parch**, **home.dest**, **boat**, and **body**.

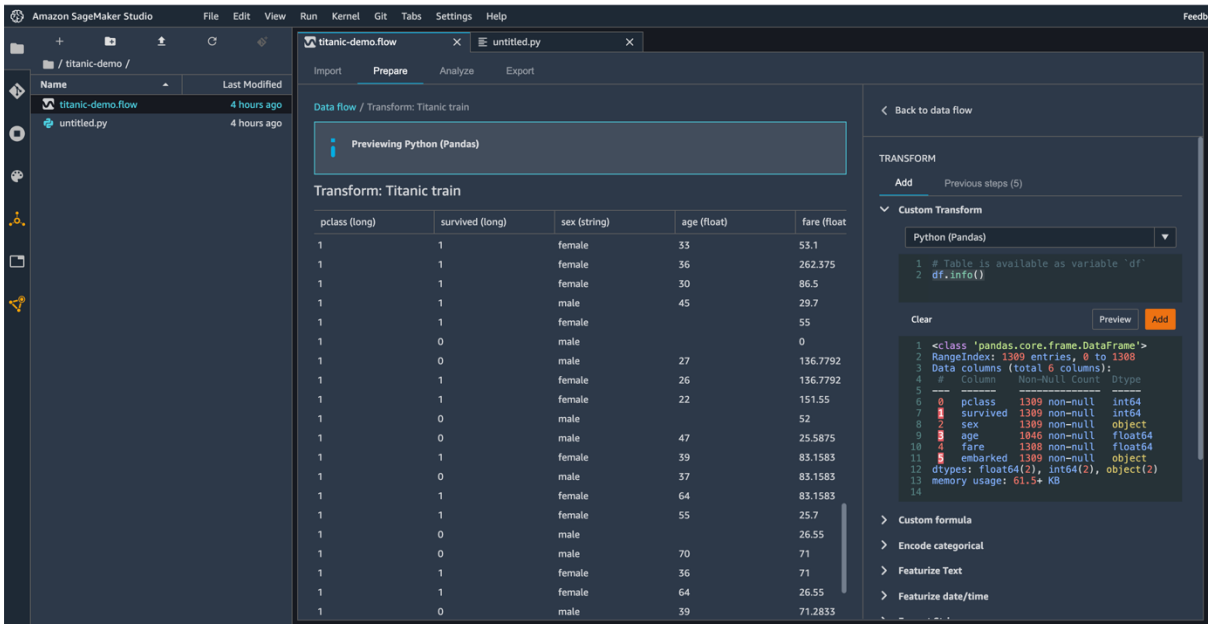
Clean up Missing Values

Now, clean up missing values. You can do this with the **Handling missing values** transform group.

A number of columns have missing values. Of the remaining columns, *age* and *fare* contain missing values. Inspect this using the **Custom Transform**.

Using the **Python (Pandas)** option, use the following to quickly review the number of entries in each column:

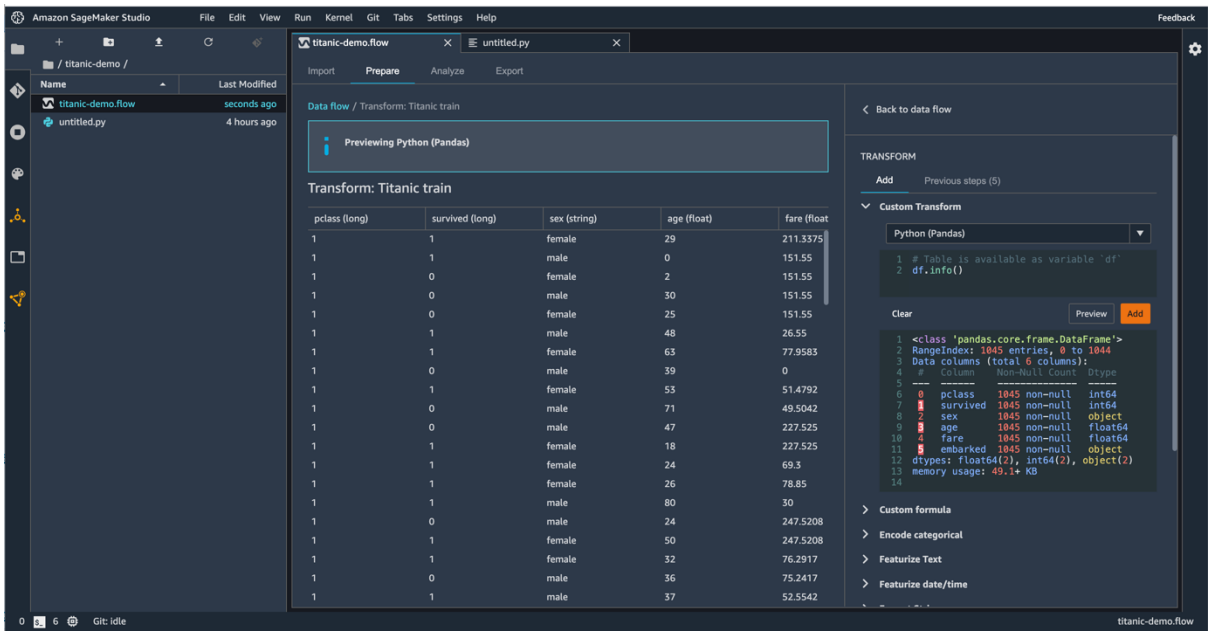
```
df.info()
```



To drop rows with missing values in the *age* category, do the following:

1. Choose **Handling missing values**.
2. Choose **Drop missing** for the **Transformer**.
3. Choose **Drop Rows** for the **Dimension**.
4. Choose *age* for the **Input column**.
5. Choose **Preview** to see the new data frame, and then choose **Add** to add the transform to your flow.
6. Repeat the same process for *fare*.

You can use `df.info()` in the **Custom transform** section to confirm that all rows now have 1,045 values.



Custom Pandas: Encode

Try flat encoding using Pandas. Encoding categorical data is the process of creating a numerical representation for categories. For example, if your categories are `Dog` and `Cat`, you may encode this information into two vectors: `[1,0]` to represent `Dog`, and `[0,1]` to represent `Cat`.

1. In the **Custom Transform** section, choose **Python (Pandas)** from the dropdown list.
2. Enter the following in the code box.

```
import pandas as pd

dummies = []
cols = ['pclass','sex','embarked']
for col in cols:
    dummies.append(pd.get_dummies(df[col]))

encoded = pd.concat(dummies, axis=1)

df = pd.concat((df, encoded),axis=1)
```

12. Choose **Preview** to preview the change. The encoded version of each column is added to the dataset.
13. Choose **Add** to add the transformation.

Custom SQL: SELECT Columns

Now, select the columns you want to keep using SQL. For this demo, select the columns listed in the following `SELECT` statement. Because *survived* is your target column for training, put that column first.

1. In the **Custom Transform** section, select **SQL (PySpark SQL)** from the dropdown list.
2. Enter the following in the code box.

```
SELECT survived, age, fare, 1, 2, 3, female, male, C, Q,
S FROM df;
```

3. Choose **Preview** to preview the change. The columns listed in your `SELECT` statement are the only remaining columns.
4. Choose **Add** to add the transformation.

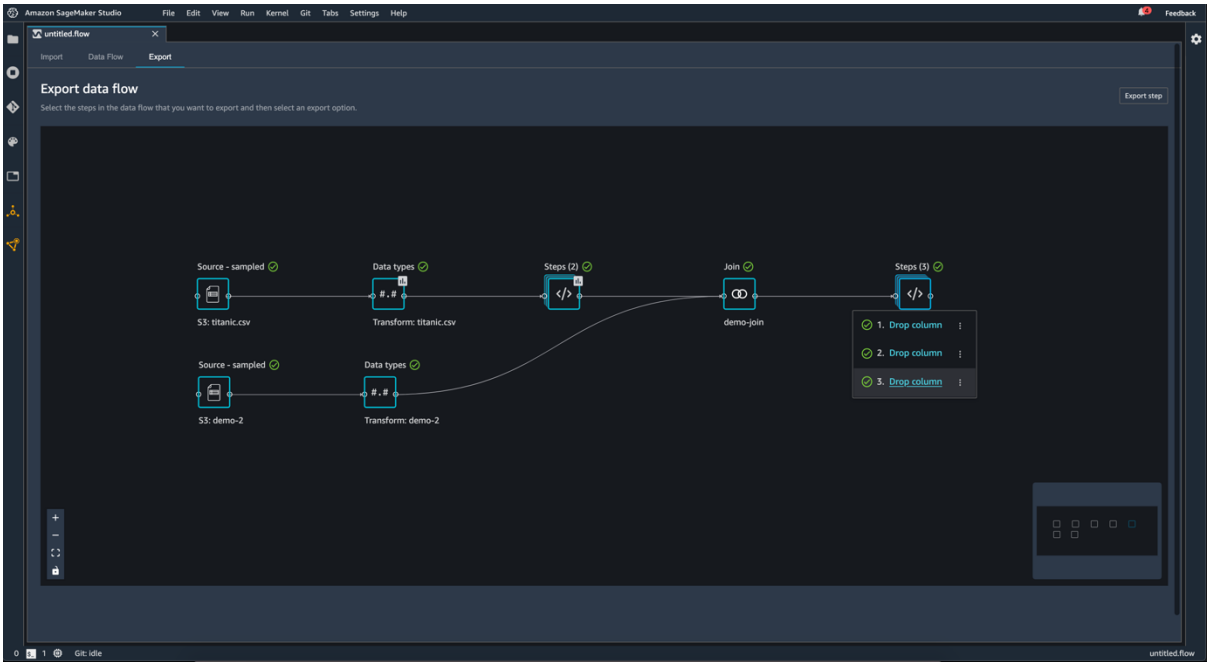
Export

When you've finished creating a data flow, you have a number of export options. The following section explains how to export to a Data Wrangler job notebook. A Data Wrangler job is used to process your data using the steps defined in your data flow. To learn more about all export options, see Export.

Export to Data Wrangler Job Notebook

When you export your data flow using a **Data Wrangler job**, the process automatically creates a Jupyter Notebook. This notebook automatically opens in your Studio instance and is configured to run a SageMaker processing job to run your Data Wrangler data flow, which is referred to as a Data Wrangler job.

1. Save your data flow. Select **File** and then select **Save Data Wrangler Flow**.
2. Choose the **Export** tab.
3. Select the last step in your data flow.



4. Choose **Data Wrangler Job**. This opens a Jupyter Notebook.

5. Choose any **Python 3 (Data Science)** kernel for the **Kernel**.
6. When the kernel starts, run the cells in the notebook book until **Kick off SageMaker Training Job (Optional)**.
7. Optionally, you can run the cells in **Kick off SageMaker Training Job (Optional)** if you want to create a SageMaker training job to train an XGboost classifier. You can find the cost to run an SageMaker training job in Amazon SageMaker Pricing.

   Alternatively, you can add the code blocks found in Training XGBoost Classifier to the notebook and run them to use the XGBoost open source library to train an XGBoost classifier.
8. Uncomment and run the cell under **Cleanup** and run it to revert the SageMaker Python SDK to its original version.

You can monitor your Data Wrangler job status in the SageMaker console in the **Processing** tab. Additionally, you can monitor your Data Wrangler job using Amazon CloudWatch. For additional information, see Monitor Amazon SageMaker Processing Jobs with CloudWatch Logs and Metrics.

If you kicked off a training job, you can monitor its status using the SageMaker console under **Training jobs** in the **Training section**.

Training XGBoost Classifier

In the same notebook that kicked off the Data Wrangler job, you can pull the data and train a XGBoost Binary Classifier using the prepared data with minimal data preparation.

1. First, upgrade necessary modules using `pip` and remove the _SUCCESS file (this last file is problematic when using `awswrangler`).

```
! pip install --upgrade awscli awswrangler boto sklearn

! aws s3 rm {output_path} --recursive  --exclude "*" --
include "*_SUCCESS*"
```

3. Read the data from Amazon S3. You can use `awswrangler` to recursively read all the CSV files in the S3 prefix. The data is then split into features and labels. The label is the first column of the dataframe.

```
4. import awswrangler as wr
5.
6. df = wr.s3.read_csv(path=output_path, dataset=True)
```

```
X, y = df.iloc[:,:-1],df.iloc[:,-1]
```

- Finally, create DMatrices (the XGBoost primitive structure for data) and do cross-validation using the XGBoost binary classification.

```
- import xgboost as xgb
-
- dmatrix = xgb.DMatrix(data=X, label=y)
-
- params = {"objective":"binary:logistic",'learning_rate': 0.1, 'max_depth': 5, 'alpha': 10}
-
- xgb.cv(
-     dtrain=dmatrix,
-     params=params,
-     nfold=3,
-     num_boost_round=50,
-     early_stopping_rounds=10,
-     metrics="rmse",
-     as_pandas=True,
      seed=123)
```

Shut down Data Wrangler

When you are finished using Data Wrangler, we recommend that you shut down the instance it runs on to avoid incurring additional charges. To learn how to shut down the Data Wrangler app and associated instance, see Shut Down Data Wrangler.