C2W1_Assignment

October 7, 2021

1 Week 1 Assignment: Data Validation

Tensorflow Data Validation (TFDV) is an open-source library that helps to understand, validate, and monitor production machine learning (ML) data at scale. Common use-cases include comparing training, evaluation and serving datasets, as well as checking for training/serving skew. You have seen the core functionalities of this package in the previous ungraded lab and you will get to practice them in this week's assignment.

In this lab, you will use TFDV in order to:

- Generate and visualize statistics from a dataframe
- Infer a dataset schema
- Calculate, visualize and fix anomalies

Let's begin!

1.1 Table of Contents

1 - Setup and Imports

[1]:
```python
# Import packages
import os
import pandas as pd
import tensorflow as tf
import tempfile, urllib, zipfile
import tensorflow_data_validation as tfdv


from tensorflow.python.lib.io import file_io
from tensorflow_data_validation.utils import slicing_util
from tensorflow_metadata.proto.v0.statistics_pb2 import␣
↪DatasetFeatureStatisticsList, DatasetFeatureStatistics

# Set TF's logger to only display errors to avoid internal warnings being shown
tf.get_logger().setLevel('ERROR')
```

2 - Load the Dataset You will be using the Diabetes 130-US hospitals for years 1999-2008 Data Set donated to the University of California, Irvine (UCI) Machine Learning Repository. The dataset represents 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. It includes over 50 features representing patient and hospital outcomes.

This dataset has already been included in your Jupyter workspace so you can easily load it.

2.1 Read and Split the Dataset

[2]:
```python
# Read CSV data into a dataframe and recognize the missing data that is encoded␣
↪with '?' string as NaN
df = pd.read_csv('dataset_diabetes/diabetic_data.csv', header=0, na_values = '?
↪')

# Preview the dataset
df.head()
```

[2]:
```
   encounter_id  patient_nbr             race  gender      age weight  \
0       2278392      8222157        Caucasian  Female   [0-10)    NaN
1        149190     55629189        Caucasian  Female  [10-20)    NaN
2         64410     86047875  AfricanAmerican  Female  [20-30)    NaN
3        500364     82442376        Caucasian    Male  [30-40)    NaN
4         16680     42519267        Caucasian    Male  [40-50)    NaN

   admission_type_id  discharge_disposition_id  admission_source_id  \
0                  6                        25                    1
1                  1                         1                    7
2                  1                         1                    7
3                  1                         1                    7
4                  1                         1                    7
```

```
   time_in_hospital  … citoglipton insulin  glyburide-metformin  \
0                 1  …          No      No                   No
1                 3  …          No      Up                   No
2                 2  …          No      No                   No
3                 2  …          No      Up                   No
4                 1  …          No  Steady                   No

   glipizide-metformin  glimepiride-pioglitazone  metformin-rosiglitazone  \
0                   No                        No                       No
1                   No                        No                       No
2                   No                        No                       No
3                   No                        No                       No
4                   No                        No                       No

   metformin-pioglitazone  change diabetesMed readmitted
0                      No      No          No         NO
1                      No      Ch         Yes        >30
2                      No      No         Yes         NO
3                      No      Ch         Yes         NO
4                      No      Ch         Yes         NO

[5 rows x 50 columns]
```

Data splits

In a production ML system, the model performance can be negatively affected by anomalies and divergence between data splits for training, evaluation, and serving. To emulate a production system, you will split the dataset into:

- 70% training set
- 15% evaluation set
- 15% serving set

You will then use TFDV to visualize, analyze, and understand the data. You will create a data schema from the training dataset, then compare the evaluation and serving sets with this schema to detect anomalies and data drift/skew.

Label Column

This dataset has been prepared to analyze the factors related to readmission outcome. In this notebook, you will treat the `readmitted` column as the *target* or label column.

The target (or label) is important to know while splitting the data into training, evaluation and serving sets. In supervised learning, you need to include the target in the training and evaluation datasets. For the serving set however (i.e. the set that simulates the data coming from your users), the **label column needs to be dropped** since that is the feature that your model will be trying to predict.

The following function returns the training, evaluation and serving partitions of a given dataset:

[3]:
```python
def prepare_data_splits_from_dataframe(df):
    '''
    Splits a Pandas Dataframe into training, evaluation and serving sets.

    Parameters:
            df : pandas dataframe to split

    Returns:
            train_df: Training dataframe(70% of the entire dataset)
            eval_df: Evaluation dataframe (15% of the entire dataset)
            serving_df: Serving dataframe (15% of the entire dataset, label␣
↪column dropped)
    '''

    # 70% of records for generating the training set
    train_len = int(len(df) * 0.7)

    # Remaining 30% of records for generating the evaluation and serving sets
    eval_serv_len = len(df) - train_len

    # Half of the 30%, which makes up 15% of total records, for generating the␣
↪evaluation set
    eval_len = eval_serv_len // 2

    # Remaining 15% of total records for generating the serving set
    serv_len = eval_serv_len - eval_len

    # Sample the train, validation and serving sets. We specify a random state␣
↪for repeatable outcomes.
    train_df = df.iloc[:train_len].sample(frac=1, random_state=48).
↪reset_index(drop=True)
    eval_df = df.iloc[train_len: train_len + eval_len].sample(frac=1,␣
↪random_state=48).reset_index(drop=True)
    serving_df = df.iloc[train_len + eval_len: train_len + eval_len + serv_len].
↪sample(frac=1, random_state=48).reset_index(drop=True)

    # Serving data emulates the data that would be submitted for predictions,␣
↪so it should not have the label column.
    serving_df = serving_df.drop(['readmitted'], axis=1)

    return train_df, eval_df, serving_df
```

[4]:
```python
# Split the datasets
train_df, eval_df, serving_df = prepare_data_splits_from_dataframe(df)
print('Training dataset has {} records\nValidation dataset has {}␣
↪records\nServing dataset has {} records'.
↪format(len(train_df),len(eval_df),len(serving_df)))
```

```
Training dataset has 71236 records
Validation dataset has 15265 records
Serving dataset has 15265 records
```

3 - Generate and Visualize Training Data Statistics

In this section, you will be generating descriptive statistics from the dataset. This is usually the first step when dealing with a dataset you are not yet familiar with. It is also known as performing an *exploratory data analysis* and its purpose is to understand the data types, the data itself and any possible issues that need to be addressed.

It is important to mention that **exploratory data analysis should be perfomed on the training dataset** only. This is because getting information out of the evaluation or serving datasets can be seen as "cheating" since this data is used to emulate data that you have not collected yet and will try to predict using your ML algorithm. **In general, it is a good practice to avoid leaking information from your evaluation and serving data into your model.**

Removing Irrelevant Features

Before you generate the statistics, you may want to drop irrelevant features from your dataset. You can do that with TFDV with the tfdv.StatsOptions class. It is usually **not a good idea** to drop features without knowing what information they contain. However there are times when this can be fairly obvious.

One of the important parameters of the `StatsOptions` class is `feature_whitelist`, which defines the features to include while calculating the data statistics. You can check the documentation to learn more about the class arguments.

In this case, you will omit the statistics for `encounter_id` and `patient_nbr` since they are part of the internal tracking of patients in the hospital and they don't contain valuable information for the task at hand.

[5]:
```python
# Define features to remove
features_to_remove = {'encounter_id', 'patient_nbr'}

# Collect features to whitelist while computing the statistics
approved_cols = [col for col in df.columns if (col not in features_to_remove)]

# Instantiate a StatsOptions class and define the feature_whitelist property
stats_options = tfdv.StatsOptions(feature_whitelist=approved_cols)

# Review the features to generate the statistics
print(stats_options.feature_whitelist)
```

```
['race', 'gender', 'age', 'weight', 'admission_type_id',
'discharge_disposition_id', 'admission_source_id', 'time_in_hospital',
'payer_code', 'medical_specialty', 'num_lab_procedures', 'num_procedures',
'num_medications', 'number_outpatient', 'number_emergency', 'number_inpatient',
'diag_1', 'diag_2', 'diag_3', 'number_diagnoses', 'max_glu_serum', 'A1Cresult',
'metformin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride',
'acetohexamide', 'glipizide', 'glyburide', 'tolbutamide', 'pioglitazone',
'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone', 'tolazamide',
```

'examide', 'citoglipton', 'insulin', 'glyburide-metformin', 'glipizide-metformin', 'glimepiride-pioglitazone', 'metformin-rosiglitazone', 'metformin-pioglitazone', 'change', 'diabetesMed', 'readmitted']

Exercise 1: Generate Training Statistics

TFDV allows you to generate statistics from different data formats such as CSV or a Pandas DataFrame.

Since you already have the data stored in a DataFrame you can use the function `tfdv.generate_statistics_from_dataframe()` which, given a DataFrame and `stats_options`, generates an object of type `DatasetFeatureStatisticsList`. This object includes the computed statistics of the given dataset.

Complete the cell below to generate the statistics of the training set. Remember to pass the training dataframe and the `stats_options` that you defined above as arguments.

[6]:
```
### START CODE HERE
train_stats = tfdv.generate_statistics_from_dataframe(train_df,
                                                                                     ␣
 ↪stats_options=stats_options)
### END CODE HERE
```

[7]:
```
# TEST CODE

# get the number of features used to compute statistics
print(f"Number of features used: {len(train_stats.datasets[0].features)}")

# check the number of examples used
print(f"Number of examples used: {train_stats.datasets[0].num_examples}")

# check the column names of the first and last feature
print(f"First feature: {train_stats.datasets[0].features[0].path.step[0]}")
print(f"Last feature: {train_stats.datasets[0].features[-1].path.step[0]}")
```

```
Number of features used: 48
Number of examples used: 71236
First feature: race
Last feature: readmitted
```

Expected Output:

```
Number of features used: 48
Number of examples used: 71236
First feature: race
Last feature: readmitted
```

Exercise 2: Visualize Training Statistics

Now that you have the computed statistics in the `DatasetFeatureStatisticsList` instance, you will need a way to **visualize** these to get actual insights. TFDV provides this functionality through the method `tfdv.visualize_statistics()`.

Using this function in an interactive Python environment such as this one will output a very nice and convenient way to interact with the descriptive statistics you generated earlier.

Try it out yourself! Remember to pass in the generated training statistics in the previous exercise as an argument.

[8]:
```
### START CODE HERE
tfdv.visualize_statistics(train_stats)
### END CODE HERE
```

```
<IPython.core.display.HTML object>
```

4 - Infer a data schema

A schema defines the **properties of the data** and can thus be used to detect errors. Some of these properties include:

- which features are expected to be present
- feature type
- the number of values for a feature in each example
- the presence of each feature across all examples
- the expected domains of features

The schema is expected to be fairly static, whereas statistics can vary per data split. So, you will **infer the data schema from only the training dataset**. Later, you will generate statistics for evaluation and serving datasets and compare their state with the data schema to detect anomalies, drift and skew.

Exercise 3: Infer the training set schema

Schema inference is straightforward using `tfdv.infer_schema()`. This function needs only the **statistics** (an instance of `DatasetFeatureStatisticsList`) of your data as input. The output will be a Schema protocol buffer containing the results.

A complimentary function is `tfdv.display_schema()` for displaying the schema in a table. This accepts a **Schema** protocol buffer as input.

Fill the code below to infer the schema from the training statistics using TFDV and display the result.

[9]:
```
### START CODE HERE
# Infer the data schema by using the training statistics that you generated
schema = tfdv.infer_schema(train_stats)

# Display the data schema
tfdv.display_schema(schema)

### END CODE HERE
```

```
                              Type  Presence Valency  \
Feature name
'race'                      STRING  optional  single
'gender'                    STRING  required
```

'age'	STRING	required	
'weight'	STRING	optional	single
'admission_type_id'	INT	required	
'discharge_disposition_id'	INT	required	
'admission_source_id'	INT	required	
'time_in_hospital'	INT	required	
'payer_code'	STRING	optional	single
'medical_specialty'	STRING	optional	single
'num_lab_procedures'	INT	required	
'num_procedures'	INT	required	
'num_medications'	INT	required	
'number_outpatient'	INT	required	
'number_emergency'	INT	required	
'number_inpatient'	INT	required	
'diag_1'	BYTES	optional	single
'diag_2'	BYTES	optional	single
'diag_3'	BYTES	optional	single
'number_diagnoses'	INT	required	
'max_glu_serum'	STRING	required	
'A1Cresult'	STRING	required	
'metformin'	STRING	required	
'repaglinide'	STRING	required	
'nateglinide'	STRING	required	
'chlorpropamide'	STRING	required	
'glimepiride'	STRING	required	
'acetohexamide'	STRING	required	
'glipizide'	STRING	required	
'glyburide'	STRING	required	
'tolbutamide'	STRING	required	
'pioglitazone'	STRING	required	
'rosiglitazone'	STRING	required	
'acarbose'	STRING	required	
'miglitol'	STRING	required	
'troglitazone'	STRING	required	
'tolazamide'	STRING	required	
'examide'	STRING	required	
'citoglipton'	STRING	required	
'insulin'	STRING	required	
'glyburide-metformin'	STRING	required	
'glipizide-metformin'	STRING	required	
'glimepiride-pioglitazone'	STRING	required	
'metformin-rosiglitazone'	STRING	required	
'metformin-pioglitazone'	STRING	required	
'change'	STRING	required	
'diabetesMed'	STRING	required	
'readmitted'	STRING	required	

Domain

Feature name	
'race'	'race'
'gender'	'gender'
'age'	'age'
'weight'	'weight'
'admission_type_id'	-
'discharge_disposition_id'	-
'admission_source_id'	-
'time_in_hospital'	-
'payer_code'	'payer_code'
'medical_specialty'	'medical_specialty'
'num_lab_procedures'	-
'num_procedures'	-
'num_medications'	-
'number_outpatient'	-
'number_emergency'	-
'number_inpatient'	-
'diag_1'	-
'diag_2'	-
'diag_3'	-
'number_diagnoses'	-
'max_glu_serum'	'max_glu_serum'
'A1Cresult'	'A1Cresult'
'metformin'	'metformin'
'repaglinide'	'repaglinide'
'nateglinide'	'nateglinide'
'chlorpropamide'	'chlorpropamide'
'glimepiride'	'glimepiride'
'acetohexamide'	'acetohexamide'
'glipizide'	'glipizide'
'glyburide'	'glyburide'
'tolbutamide'	'tolbutamide'
'pioglitazone'	'pioglitazone'
'rosiglitazone'	'rosiglitazone'
'acarbose'	'acarbose'
'miglitol'	'miglitol'
'troglitazone'	'troglitazone'
'tolazamide'	'tolazamide'
'examide'	'examide'
'citoglipton'	'citoglipton'
'insulin'	'insulin'
'glyburide-metformin'	'glyburide-metformin'
'glipizide-metformin'	'glipizide-metformin'
'glimepiride-pioglitazone'	'glimepiride-pioglitazone'
'metformin-rosiglitazone'	'metformin-rosiglitazone'
'metformin-pioglitazone'	'metformin-pioglitazone'
'change'	'change'
'diabetesMed'	'diabetesMed'

```
'readmitted'                                 'readmitted'
                                                                                                
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                              
 ↪                                                                                           
 ↪Values
Domain
'race'                      'AfricanAmerican', 'Asian', 'Caucasian', 'Hispanic', 
 ↪'Other'
'gender'                    'Female', 'Male', 'Unknown/Invalid'
'age'                       '[0-10)', '[10-20)', '[20-30)', '[30-40)', 
 ↪'[40-50)', '[50-60)', '[60-70)', '[70-80)', '[80-90)', '[90-100)'
'weight'                    '>200', '[0-25)', '[100-125)', '[125-150)', 
 ↪'[150-175)', '[175-200)', '[25-50)', '[50-75)', '[75-100)'
'payer_code'                'BC', 'CH', 'CM', 'CP', 'DM', 'HM', 'MC', 'MD', 
 ↪'MP', 'OG', 'OT', 'PO', 'SI', 'SP', 'UN', 'WC'
```

```
'medical_specialty'         'AllergyandImmunology', 'Anesthesiology',␣
 ↪'Anesthesiology-Pediatric', 'Cardiology', 'Cardiology-Pediatric', 'Dentistry',␣
 ↪'Dermatology', 'Emergency/Trauma', 'Endocrinology', 'Family/GeneralPractice',␣
 ↪'Gastroenterology', 'Gynecology', 'Hematology', 'Hematology/Oncology',␣
 ↪'Hospitalist', 'InfectiousDiseases', 'InternalMedicine', 'Nephrology',␣
 ↪'Neurology', 'Obsterics&Gynecology-GynecologicOnco', 'Obstetrics',␣
 ↪'ObstetricsandGynecology', 'Oncology', 'Ophthalmology', 'Orthopedics',␣
 ↪'Orthopedics-Reconstructive', 'Osteopath', 'Otolaryngology',␣
 ↪'OutreachServices', 'Pathology', 'Pediatrics',␣
 ↪'Pediatrics-AllergyandImmunology', 'Pediatrics-CriticalCare',␣
 ↪'Pediatrics-EmergencyMedicine', 'Pediatrics-Endocrinology',␣
 ↪'Pediatrics-Hematology-Oncology', 'Pediatrics-InfectiousDiseases',␣
 ↪'Pediatrics-Neurology', 'Pediatrics-Pulmonology', 'Perinatology',␣
 ↪'PhysicalMedicineandRehabilitation', 'PhysicianNotFound', 'Podiatry',␣
 ↪'Proctology', 'Psychiatry', 'Psychiatry-Addictive', 'Psychiatry-Child/
 ↪Adolescent', 'Psychology', 'Pulmonology', 'Radiologist', 'Radiology',␣
 ↪'Rheumatology', 'Speech', 'SportsMedicine', 'Surgeon',␣
 ↪'Surgery-Cardiovascular', 'Surgery-Cardiovascular/Thoracic',␣
 ↪'Surgery-Colon&Rectal', 'Surgery-General', 'Surgery-Maxillofacial',␣
 ↪'Surgery-Neuro', 'Surgery-Pediatric', 'Surgery-Plastic',␣
 ↪'Surgery-PlasticwithinHeadandNeck', 'Surgery-Thoracic', 'Surgery-Vascular',␣
 ↪'SurgicalSpecialty', 'Urology'
'max_glu_serum'             '>200', '>300', 'None', 'Norm'
'A1Cresult'                 '>7', '>8', 'None', 'Norm'
'metformin'                 'Down', 'No', 'Steady', 'Up'
'repaglinide'               'Down', 'No', 'Steady', 'Up'
'nateglinide'               'Down', 'No', 'Steady', 'Up'
'chlorpropamide'            'Down', 'No', 'Steady', 'Up'
'glimepiride'               'Down', 'No', 'Steady', 'Up'
'acetohexamide'             'No', 'Steady'
'glipizide'                 'Down', 'No', 'Steady', 'Up'
'glyburide'                 'Down', 'No', 'Steady', 'Up'
'tolbutamide'               'No', 'Steady'
'pioglitazone'              'Down', 'No', 'Steady', 'Up'
'rosiglitazone'             'Down', 'No', 'Steady', 'Up'
'acarbose'                  'Down', 'No', 'Steady', 'Up'
'miglitol'                  'Down', 'No', 'Steady', 'Up'
'troglitazone'              'No', 'Steady'
'tolazamide'                'No', 'Steady', 'Up'
'examide'                   'No'
'citoglipton'               'No'
'insulin'                   'Down', 'No', 'Steady', 'Up'
'glyburide-metformin'       'Down', 'No', 'Steady', 'Up'
'glipizide-metformin'       'No', 'Steady'
'glimepiride-pioglitazone'  'No'
'metformin-rosiglitazone'   'No'
'metformin-pioglitazone'    'No'
```

```
'change'                     'Ch', 'No'
'diabetesMed'                'No', 'Yes'
'readmitted'                 '<30', '>30', 'NO'
```

[10]:
```python
# TEST CODE

# Check number of features
print(f"Number of features in schema: {len(schema.feature)}")

# Check domain name of 2nd feature
print(f"Second feature in schema: {list(schema.feature)[1].domain}")
```

```
Number of features in schema: 48
Second feature in schema: gender
```

Expected Output:

```
Number of features in schema: 48
Second feature in schema: gender
```

Be sure to check the information displayed before moving forward.

5 - Calculate, Visualize and Fix Evaluation Anomalies

It is important that the schema of the evaluation data is consistent with the training data since the data that your model is going to receive should be consistent to the one you used to train it with.

Moreover, it is also important that the **features of the evaluation data belong roughly to the same range as the training data**. This ensures that the model will be evaluated on a similar loss surface covered during training.

Exercise 4: Compare Training and Evaluation Statistics

Now you are going to generate the evaluation statistics and compare it with training statistics. You can use the `tfdv.generate_statistics_from_dataframe()` function for this. But this time, you'll need to pass the **evaluation data**. For the `stats_options` parameter, the list you used before works here too.

Remember that to visualize the evaluation statistics you can use `tfdv.visualize_statistics()`.

However, it is impractical to visualize both statistics separately and do your comparison from there. Fortunately, TFDV has got this covered. You can use the `visualize_statistics` function and pass additional parameters to overlay the statistics from both datasets (referenced as left-hand side and right-hand side statistics). Let's see what these parameters are:

- `lhs_statistics`: Required parameter. Expects an instance of `DatasetFeatureStatisticsList`.
- `rhs_statistics`: Expects an instance of `DatasetFeatureStatisticsList` to compare with `lhs_statistics`.
- `lhs_name`: Name of the `lhs_statistics` dataset.
- `rhs_name`: Name of the `rhs_statistics` dataset.

For this case, remember to define the `lhs_statistics` protocol with the `eval_stats`, and the optional `rhs_statistics` protocol with the `train_stats`.

Additionally, check the function for the protocol name declaration, and define the lhs and rhs names as `'EVAL_DATASET'` and `'TRAIN_DATASET'` respectively.

[11]:
```
### START CODE HERE
# Generate evaluation dataset statistics
# HINT: Remember to use the evaluation dataframe and to pass the stats_options␣
 ↪(that you defined before) as an argument
eval_stats = tfdv.generate_statistics_from_dataframe(eval_df,
                                                                           ␣
 ↪stats_options=stats_options)

# Compare evaluation data with training data
# HINT: Remember to use both the evaluation and training statistics with the␣
 ↪lhs_statistics and rhs_statistics arguments
# HINT: Assign the names of 'EVAL_DATASET' and 'TRAIN_DATASET' to the lhs and␣
 ↪rhs protocols
tfdv.visualize_statistics(lhs_statistics=eval_stats,
                          rhs_statistics=train_stats,
                          lhs_name='EVAL_DATASET',
                          rhs_name='TRAIN_DATASET')

### END CODE HERE
```

```
<IPython.core.display.HTML object>
```

[12]:
```
# TEST CODE

# get the number of features used to compute statistics
print(f"Number of features: {len(eval_stats.datasets[0].features)}")

# check the number of examples used
print(f"Number of examples: {eval_stats.datasets[0].num_examples}")

# check the column names of the first and last feature
print(f"First feature: {eval_stats.datasets[0].features[0].path.step[0]}")
print(f"Last feature: {eval_stats.datasets[0].features[-1].path.step[0]}")
```

```
Number of features: 48
Number of examples: 15265
First feature: race
Last feature: readmitted
```

Expected Output:

```
Number of features: 48
Number of examples: 15265
First feature: race
```

```
Last feature: readmitted
```

Exercise 5: Detecting Anomalies

At this point, you should ask if your evaluation dataset matches the schema from your training dataset. For instance, if you scroll through the output cell in the previous exercise, you can see that the categorical feature **glimepiride-pioglitazone** has 1 unique value in the training set while the evaluation dataset has 2. You can verify with the built-in Pandas `describe()` method as well.

[13]:
```python
train_df["glimepiride-pioglitazone"].describe()
```

[13]:
```
count      71236
unique     1
top        No
freq       71236
Name: glimepiride-pioglitazone, dtype: object
```

[14]:
```python
eval_df["glimepiride-pioglitazone"].describe()
```

[14]:
```
count      15265
unique     2
top        No
freq       15264
Name: glimepiride-pioglitazone, dtype: object
```

It is possible but highly inefficient to visually inspect and determine all the anomalies. So, let's instead use TFDV functions to detect and display these.

You can use the function `tfdv.validate_statistics()` for detecting anomalies and `tfdv.display_anomalies()` for displaying them.

The `validate_statistics()` method has two required arguments: - an instance of `DatasetFeatureStatisticsList` - an instance of `Schema`

Fill in the following graded function which, given the statistics and schema, displays the anomalies found.

[15]:
```python
def calculate_and_display_anomalies(statistics, schema):
    '''
    Calculate and display anomalies.

            Parameters:
                    statistics : Data statistics in statistics_pb2.
 ↪DatasetFeatureStatisticsList format
                    schema : Data schema in schema_pb2.Schema format

            Returns:
                    display of calculated anomalies
    '''
    ### START CODE HERE
```

```
    # HINTS: Pass the statistics and schema parameters into the validation␣
↪function
    anomalies = tfdv.validate_statistics(statistics, schema)
    # HINTS: Display input anomalies by using the calculated anomalies
    tfdv.display_anomalies(anomalies)
    ### END CODE HERE
```

You should see detected anomalies in the `medical_specialty` and `glimepiride-pioglitazone` features by running the cell below.

[16]:
```
# Check evaluation data for errors by validating the evaluation data staticss␣
↪using the previously inferred schema
calculate_and_display_anomalies(eval_stats, schema=schema)
```

```
                              Anomaly short description  \
Feature name
'medical_specialty'            Unexpected string values
'glimepiride-pioglitazone'     Unexpected string values

                                                                              ␣
↪Anomaly long description
Feature name
'medical_specialty'            Examples contain values missing from the schema:␣
↪Neurophysiology (<1%).
'glimepiride-pioglitazone'     Examples contain values missing from the schema:␣
↪Steady (<1%).
```

Exercise 6: Fix evaluation anomalies in the schema

The evaluation data has records with values for the features **glimepiride-pioglitazone** and **medical_speciality** that were not included in the schema generated from the training data. You can fix this by adding the new values that exist in the evaluation dataset to the domain of these features.

To get the `domain` of a particular feature you can use `tfdv.get_domain()`.

You can use the `append()` method to the `value` property of the returned `domain` to add strings to the valid list of values. To be more explicit, given a domain you can do something like:

```
domain.value.append("feature_value")
```

[17]:
```
### START CODE HERE

# Get the domain associated with the input feature, glimepiride-pioglitazone,␣
↪from the schema
glimepiride_pioglitazone_domain = tfdv.get_domain(schema,␣
↪'glimepiride-pioglitazone')

# HINT: Append the missing value 'Steady' to the domain
glimepiride_pioglitazone_domain.value.append('Steady')
```

```
# Get the domain associated with the input feature, medical_specialty, from the schema
medical_specialty_domain = tfdv.get_domain(schema, 'medical_specialty')

# HINT: Append the missing value 'Neurophysiology' to the domain
medical_specialty_domain.value.append('Neurophysiology')

# HINT: Re-calculate and re-display anomalies with the new schema
calculate_and_display_anomalies(eval_stats, schema=schema)
### END CODE HERE
```

```
<IPython.core.display.HTML object>
```

If you did the exercise correctly, you should see *"No anomalies found."* after running the cell above.

6 - Schema Environments

By default, all datasets in a pipeline should use the same schema. However, there are some exceptions.

For example, the **label column is dropped in the serving set** so this will be flagged when comparing with the training set schema.

In this case, introducing slight schema variations is necessary.

Exercise 7: Check anomalies in the serving set

Now you are going to check for anomalies in the **serving data**. The process is very similar to the one you previously did for the evaluation data with a little change.

Let's create a new `StatsOptions` that is aware of the information provided by the schema and use it when generating statistics from the serving DataFrame.

[18]:
```
# Define a new statistics options by the tfdv.StatsOptions class for the serving data by passing the previously inferred schema
options = tfdv.StatsOptions(schema=schema,
                            infer_type_from_schema=True,
                            feature_whitelist=approved_cols)
```

[19]:
```
### START CODE HERE
# Generate serving dataset statistics
# HINT: Remember to use the serving dataframe and to pass the newly defined statistics options
serving_stats = tfdv.generate_statistics_from_dataframe(serving_df, stats_options=options)
# HINT: Calculate and display anomalies using the generated serving statistics
calculate_and_display_anomalies(serving_stats, schema=schema)
### END CODE HERE
```

```
                         Anomaly short description  \
Feature name
'metformin-pioglitazone'   Unexpected string values
```

```
'payer_code'                 Unexpected string values
'medical_specialty'          Unexpected string values
'metformin-rosiglitazone'    Unexpected string values
'readmitted'                 Column dropped


                                                                         ␣
↪                                      Anomaly long description
Feature name
'metformin-pioglitazone'     Examples contain values missing from the schema:␣
↪Steady (<1%).
'payer_code'                 Examples contain values missing from the schema: FR␣
↪(<1%).
'medical_specialty'          Examples contain values missing from the schema:␣
↪DCPTEAM (<1%), Endocrinology-Metabolism (<1%), Resident (<1%).
'metformin-rosiglitazone'    Examples contain values missing from the schema:␣
↪Steady (<1%).
'readmitted'                 Column is completely missing
```

You should see that `metformin-rosiglitazone`, `metformin-pioglitazone`, `payer_code` and `medical_specialty` features have an anomaly (i.e. Unexpected string values) which is less than 1%.

Let's **relax the anomaly detection constraints** for the last two of these features by defining the `min_domain_mass` of the feature's distribution constraints.

[20]:
```
# This relaxes the minimum fraction of values that must come from the domain␣
↪for the feature.

# Get the feature and relax to match 90% of the domain
payer_code = tfdv.get_feature(schema, 'payer_code')
payer_code.distribution_constraints.min_domain_mass = 0.9

# Get the feature and relax to match 90% of the domain
medical_specialty = tfdv.get_feature(schema, 'medical_specialty')
medical_specialty.distribution_constraints.min_domain_mass = 0.9

# Detect anomalies with the updated constraints
calculate_and_display_anomalies(serving_stats, schema=schema)
```

```
                           Anomaly short description  \
Feature name
'readmitted'                 Column dropped
'metformin-pioglitazone'     Unexpected string values
'metformin-rosiglitazone'    Unexpected string values


                                                                  Anomaly long␣
↪description
Feature name
'readmitted'                 Column is completely missing
```

```
'metformin-pioglitazone'   Examples contain values missing from the schema:␣
 ↪Steady (<1%).
'metformin-rosiglitazone'  Examples contain values missing from the schema:␣
 ↪Steady (<1%).
```

If the payer_code and medical_specialty are no longer part of the output cell, then the relaxation worked!

Exercise 8: Modifying the Domain

Let's investigate the possible cause of the anomalies for the other features, namely metformin-pioglitazone and metformin-rosiglitazone. From the output of the previous exercise, you'll see that the anomaly long description says: "Examples contain values missing from the schema: Steady (<1%)". You can redisplay the schema and look at the domain of these features to verify this statement.

When you inferred the schema at the start of this lab, it's possible that some values were not detected in the training data so it was not included in the expected domain values of the feature's schema. In the case of metformin-rosiglitazone and metformin-pioglitazone, the value "Steady" is indeed missing. You will just see "No" in the domain of these two features after running the code cell below.

[21]:
```
tfdv.display_schema(schema)
```

```
                              Type  Presence Valency  \
Feature name
'race'                        STRING  optional  single
'gender'                      STRING  required
'age'                         STRING  required
'weight'                      STRING  optional  single
'admission_type_id'           INT     required
'discharge_disposition_id'    INT     required
'admission_source_id'         INT     required
'time_in_hospital'            INT     required
'payer_code'                  STRING  optional  single
'medical_specialty'           STRING  optional  single
'num_lab_procedures'          INT     required
'num_procedures'              INT     required
'num_medications'             INT     required
'number_outpatient'           INT     required
'number_emergency'            INT     required
'number_inpatient'            INT     required
'diag_1'                      BYTES   optional  single
'diag_2'                      BYTES   optional  single
'diag_3'                      BYTES   optional  single
'number_diagnoses'            INT     required
'max_glu_serum'               STRING  required
'A1Cresult'                   STRING  required
'metformin'                   STRING  required
'repaglinide'                 STRING  required
```

'nateglinide'	STRING	required
'chlorpropamide'	STRING	required
'glimepiride'	STRING	required
'acetohexamide'	STRING	required
'glipizide'	STRING	required
'glyburide'	STRING	required
'tolbutamide'	STRING	required
'pioglitazone'	STRING	required
'rosiglitazone'	STRING	required
'acarbose'	STRING	required
'miglitol'	STRING	required
'troglitazone'	STRING	required
'tolazamide'	STRING	required
'examide'	STRING	required
'citoglipton'	STRING	required
'insulin'	STRING	required
'glyburide-metformin'	STRING	required
'glipizide-metformin'	STRING	required
'glimepiride-pioglitazone'	STRING	required
'metformin-rosiglitazone'	STRING	required
'metformin-pioglitazone'	STRING	required
'change'	STRING	required
'diabetesMed'	STRING	required
'readmitted'	STRING	required

	Domain
Feature name	
'race'	'race'
'gender'	'gender'
'age'	'age'
'weight'	'weight'
'admission_type_id'	-
'discharge_disposition_id'	-
'admission_source_id'	-
'time_in_hospital'	-
'payer_code'	'payer_code'
'medical_specialty'	'medical_specialty'
'num_lab_procedures'	-
'num_procedures'	-
'num_medications'	-
'number_outpatient'	-
'number_emergency'	-
'number_inpatient'	-
'diag_1'	-
'diag_2'	-
'diag_3'	-
'number_diagnoses'	-
'max_glu_serum'	'max_glu_serum'

```
'A1Cresult'                  'A1Cresult'
'metformin'                  'metformin'
'repaglinide'                'repaglinide'
'nateglinide'                'nateglinide'
'chlorpropamide'             'chlorpropamide'
'glimepiride'                'glimepiride'
'acetohexamide'              'acetohexamide'
'glipizide'                  'glipizide'
'glyburide'                  'glyburide'
'tolbutamide'                'tolbutamide'
'pioglitazone'               'pioglitazone'
'rosiglitazone'              'rosiglitazone'
'acarbose'                   'acarbose'
'miglitol'                   'miglitol'
'troglitazone'               'troglitazone'
'tolazamide'                 'tolazamide'
'examide'                    'examide'
'citoglipton'                'citoglipton'
'insulin'                    'insulin'
'glyburide-metformin'        'glyburide-metformin'
'glipizide-metformin'        'glipizide-metformin'
'glimepiride-pioglitazone'   'glimepiride-pioglitazone'
'metformin-rosiglitazone'    'metformin-rosiglitazone'
'metformin-pioglitazone'     'metformin-pioglitazone'
'change'                     'change'
'diabetesMed'                'diabetesMed'
'readmitted'                 'readmitted'

                    Values
Domain
```

```
'race'                       'AfricanAmerican', 'Asian', 'Caucasian', 'Hispanic',␣
 ↪'Other'
'gender'                     'Female', 'Male', 'Unknown/Invalid'
'age'                        '[0-10)', '[10-20)', '[20-30)', '[30-40)',␣
 ↪'[40-50)', '[50-60)', '[60-70)', '[70-80)', '[80-90)', '[90-100)'
'weight'                     '>200', '[0-25)', '[100-125)', '[125-150)',␣
 ↪'[150-175)', '[175-200)', '[25-50)', '[50-75)', '[75-100)'
'payer_code'                 'BC', 'CH', 'CM', 'CP', 'DM', 'HM', 'MC', 'MD',␣
 ↪'MP', 'OG', 'OT', 'PO', 'SI', 'SP', 'UN', 'WC'
'medical_specialty'          'AllergyandImmunology', 'Anesthesiology',␣
 ↪'Anesthesiology-Pediatric', 'Cardiology', 'Cardiology-Pediatric', 'Dentistry',␣
 ↪'Dermatology', 'Emergency/Trauma', 'Endocrinology', 'Family/GeneralPractice',␣
 ↪'Gastroenterology', 'Gynecology', 'Hematology', 'Hematology/Oncology',␣
 ↪'Hospitalist', 'InfectiousDiseases', 'InternalMedicine', 'Nephrology',␣
 ↪'Neurology', 'Obsterics&Gynecology-GynecologicOnco', 'Obstetrics',␣
 ↪'ObstetricsandGynecology', 'Oncology', 'Ophthalmology', 'Orthopedics',␣
 ↪'Orthopedics-Reconstructive', 'Osteopath', 'Otolaryngology',␣
 ↪'OutreachServices', 'Pathology', 'Pediatrics',␣
 ↪'Pediatrics-AllergyandImmunology', 'Pediatrics-CriticalCare',␣
 ↪'Pediatrics-EmergencyMedicine', 'Pediatrics-Endocrinology',␣
 ↪'Pediatrics-Hematology-Oncology', 'Pediatrics-InfectiousDiseases',␣
 ↪'Pediatrics-Neurology', 'Pediatrics-Pulmonology', 'Perinatology',␣
 ↪'PhysicalMedicineandRehabilitation', 'PhysicianNotFound', 'Podiatry',␣
 ↪'Proctology', 'Psychiatry', 'Psychiatry-Addictive', 'Psychiatry-Child/
 ↪Adolescent', 'Psychology', 'Pulmonology', 'Radiologist', 'Radiology',␣
 ↪'Rheumatology', 'Speech', 'SportsMedicine', 'Surgeon',␣
 ↪'Surgery-Cardiovascular', 'Surgery-Cardiovascular/Thoracic',␣
 ↪'Surgery-Colon&Rectal', 'Surgery-General', 'Surgery-Maxillofacial',␣
 ↪'Surgery-Neuro', 'Surgery-Pediatric', 'Surgery-Plastic',␣
 ↪'Surgery-PlasticwithinHeadandNeck', 'Surgery-Thoracic', 'Surgery-Vascular',␣
 ↪'SurgicalSpecialty', 'Urology', 'Neurophysiology'
'max_glu_serum'              '>200', '>300', 'None', 'Norm'
'A1Cresult'                  '>7', '>8', 'None', 'Norm'
'metformin'                  'Down', 'No', 'Steady', 'Up'
'repaglinide'                'Down', 'No', 'Steady', 'Up'
'nateglinide'                'Down', 'No', 'Steady', 'Up'
'chlorpropamide'             'Down', 'No', 'Steady', 'Up'
'glimepiride'                'Down', 'No', 'Steady', 'Up'
'acetohexamide'              'No', 'Steady'
'glipizide'                  'Down', 'No', 'Steady', 'Up'
'glyburide'                  'Down', 'No', 'Steady', 'Up'
'tolbutamide'                'No', 'Steady'
'pioglitazone'               'Down', 'No', 'Steady', 'Up'
'rosiglitazone'              'Down', 'No', 'Steady', 'Up'
'acarbose'                   'Down', 'No', 'Steady', 'Up'
'miglitol'                   'Down', 'No', 'Steady', 'Up'
'troglitazone'               'No', 'Steady'
```

```
'tolazamide'                 'No', 'Steady', 'Up'
'examide'                    'No'
'citoglipton'                'No'
'insulin'                    'Down', 'No', 'Steady', 'Up'
'glyburide-metformin'        'Down', 'No', 'Steady', 'Up'
'glipizide-metformin'        'No', 'Steady'
'glimepiride-pioglitazone'   'No', 'Steady'
'metformin-rosiglitazone'    'No'
'metformin-pioglitazone'     'No'
'change'                     'Ch', 'No'
'diabetesMed'                'No', 'Yes'
'readmitted'                 '<30', '>30', 'NO'
```

Towards the bottom of the Domain-Values pairs of the cell above, you can see that many features (including **'metformin'**) have the same values: `['Down', 'No', 'Steady', 'Up']`. These values are common to many features including the ones with missing values during schema inference.

TFDV allows you to modify the domains of some features to match an existing domain. To address the detected anomaly, you can **set the domain** of these features to the domain of the `metformin` feature.

Complete the function below to set the domain of a feature list to an existing feature domain.

For this, use the `tfdv.set_domain()` function, which has the following parameters:

- `schema`: The schema
- `feature_path`: The name of the feature whose domain needs to be set.
- `domain`: A domain protocol buffer or the name of a global string domain present in the input schema.

[22]:
```
def modify_domain_of_features(features_list, schema, to_domain_name):
    '''
    Modify a list of features' domains.

            Parameters:
                    features_list : Features that need to be modified
                    schema: Inferred schema
                    to_domain_name : Target domain to be transferred to the␣
↪features list

            Returns:
                    schema: new schema
    '''
    ### START CODE HERE
    # HINT: Loop over the feature list and use set_domain with the inferred␣
↪schema, feature name and target domain name
    for feature in features_list:
        tfdv.set_domain(schema=schema, feature_path=feature,␣
↪domain=to_domain_name)
```

```
    ### END CODE HERE
    return schema
```

Using this function, set the domain of the features defined in the `domain_change_features` list below to be equal to **metformin's domain** to address the anomalies found.

Since you are overriding the existing domain of the features, it is normal to get a warning so you don't do this by accident.

[23]:
```
domain_change_features = ['repaglinide', 'nateglinide', 'chlorpropamide',␣
 ↪'glimepiride',
                          'acetohexamide', 'glipizide', 'glyburide',␣
 ↪'tolbutamide', 'pioglitazone',
                          'rosiglitazone', 'acarbose', 'miglitol',␣
 ↪'troglitazone', 'tolazamide',
                          'examide', 'citoglipton', 'insulin',␣
 ↪'glyburide-metformin', 'glipizide-metformin',
                          'glimepiride-pioglitazone',␣
 ↪'metformin-rosiglitazone', 'metformin-pioglitazone']


# Infer new schema by using your modify_domain_of_features function
# and the defined domain_change_features feature list
schema = modify_domain_of_features(domain_change_features, schema, 'metformin')

# Display new schema
tfdv.display_schema(schema)
```

```
WARNING:root:Replacing existing domain of feature "repaglinide".
WARNING:root:Replacing existing domain of feature "nateglinide".
WARNING:root:Replacing existing domain of feature "chlorpropamide".
WARNING:root:Replacing existing domain of feature "glimepiride".
WARNING:root:Replacing existing domain of feature "acetohexamide".
WARNING:root:Replacing existing domain of feature "glipizide".
WARNING:root:Replacing existing domain of feature "glyburide".
WARNING:root:Replacing existing domain of feature "tolbutamide".
WARNING:root:Replacing existing domain of feature "pioglitazone".
WARNING:root:Replacing existing domain of feature "rosiglitazone".
WARNING:root:Replacing existing domain of feature "acarbose".
WARNING:root:Replacing existing domain of feature "miglitol".
WARNING:root:Replacing existing domain of feature "troglitazone".
WARNING:root:Replacing existing domain of feature "tolazamide".
WARNING:root:Replacing existing domain of feature "examide".
WARNING:root:Replacing existing domain of feature "citoglipton".
WARNING:root:Replacing existing domain of feature "insulin".
WARNING:root:Replacing existing domain of feature "glyburide-metformin".
WARNING:root:Replacing existing domain of feature "glipizide-metformin".
WARNING:root:Replacing existing domain of feature "glimepiride-pioglitazone".
```

```
WARNING:root:Replacing existing domain of feature "metformin-rosiglitazone".
WARNING:root:Replacing existing domain of feature "metformin-pioglitazone".
```

Feature name	Type	Presence	Valency	Domain
'race'	STRING	optional	single	'race'
'gender'	STRING	required		'gender'
'age'	STRING	required		'age'
'weight'	STRING	optional	single	'weight'
'admission_type_id'	INT	required		-
'discharge_disposition_id'	INT	required		-
'admission_source_id'	INT	required		-
'time_in_hospital'	INT	required		-
'payer_code'	STRING	optional	single	'payer_code'
'medical_specialty'	STRING	optional	single	'medical_specialty'
'num_lab_procedures'	INT	required		-
'num_procedures'	INT	required		-
'num_medications'	INT	required		-
'number_outpatient'	INT	required		-
'number_emergency'	INT	required		-
'number_inpatient'	INT	required		-
'diag_1'	BYTES	optional	single	-
'diag_2'	BYTES	optional	single	-
'diag_3'	BYTES	optional	single	-
'number_diagnoses'	INT	required		-
'max_glu_serum'	STRING	required		'max_glu_serum'
'A1Cresult'	STRING	required		'A1Cresult'
'metformin'	STRING	required		'metformin'
'repaglinide'	STRING	required		'metformin'
'nateglinide'	STRING	required		'metformin'
'chlorpropamide'	STRING	required		'metformin'
'glimepiride'	STRING	required		'metformin'
'acetohexamide'	STRING	required		'metformin'
'glipizide'	STRING	required		'metformin'
'glyburide'	STRING	required		'metformin'
'tolbutamide'	STRING	required		'metformin'
'pioglitazone'	STRING	required		'metformin'
'rosiglitazone'	STRING	required		'metformin'
'acarbose'	STRING	required		'metformin'
'miglitol'	STRING	required		'metformin'
'troglitazone'	STRING	required		'metformin'
'tolazamide'	STRING	required		'metformin'
'examide'	STRING	required		'metformin'
'citoglipton'	STRING	required		'metformin'
'insulin'	STRING	required		'metformin'
'glyburide-metformin'	STRING	required		'metformin'
'glipizide-metformin'	STRING	required		'metformin'
'glimepiride-pioglitazone'	STRING	required		'metformin'

```
'metformin-rosiglitazone'   STRING  required          'metformin'
'metformin-pioglitazone'    STRING  required          'metformin'
'change'                    STRING  required          'change'
'diabetesMed'               STRING  required          'diabetesMed'
'readmitted'                STRING  required          'readmitted'

                                                                                   ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                                                                                 ␣
 ↪                  Values
Domain
'race'                      'AfricanAmerican', 'Asian', 'Caucasian', 'Hispanic',␣
 ↪'Other'
'gender'                    'Female', 'Male', 'Unknown/Invalid'
'age'                       '[0-10)', '[10-20)', '[20-30)', '[30-40)',␣
 ↪'[40-50)', '[50-60)', '[60-70)', '[70-80)', '[80-90)', '[90-100)'
'weight'                    '>200', '[0-25)', '[100-125)', '[125-150)',␣
 ↪'[150-175)', '[175-200)', '[25-50)', '[50-75)', '[75-100)'
'payer_code'                'BC', 'CH', 'CM', 'CP', 'DM', 'HM', 'MC', 'MD',␣
 ↪'MP', 'OG', 'OT', 'PO', 'SI', 'SP', 'UN', 'WC'
```

```
'medical_specialty'         'AllergyandImmunology', 'Anesthesiology',␣
↪'Anesthesiology-Pediatric', 'Cardiology', 'Cardiology-Pediatric', 'Dentistry',␣
↪'Dermatology', 'Emergency/Trauma', 'Endocrinology', 'Family/GeneralPractice',␣
↪'Gastroenterology', 'Gynecology', 'Hematology', 'Hematology/Oncology',␣
↪'Hospitalist', 'InfectiousDiseases', 'InternalMedicine', 'Nephrology',␣
↪'Neurology', 'Obsterics&Gynecology-GynecologicOnco', 'Obstetrics',␣
↪'ObstetricsandGynecology', 'Oncology', 'Ophthalmology', 'Orthopedics',␣
↪'Orthopedics-Reconstructive', 'Osteopath', 'Otolaryngology',␣
↪'OutreachServices', 'Pathology', 'Pediatrics',␣
↪'Pediatrics-AllergyandImmunology', 'Pediatrics-CriticalCare',␣
↪'Pediatrics-EmergencyMedicine', 'Pediatrics-Endocrinology',␣
↪'Pediatrics-Hematology-Oncology', 'Pediatrics-InfectiousDiseases',␣
↪'Pediatrics-Neurology', 'Pediatrics-Pulmonology', 'Perinatology',␣
↪'PhysicalMedicineandRehabilitation', 'PhysicianNotFound', 'Podiatry',␣
↪'Proctology', 'Psychiatry', 'Psychiatry-Addictive', 'Psychiatry-Child/
↪Adolescent', 'Psychology', 'Pulmonology', 'Radiologist', 'Radiology',␣
↪'Rheumatology', 'Speech', 'SportsMedicine', 'Surgeon',␣
↪'Surgery-Cardiovascular', 'Surgery-Cardiovascular/Thoracic',␣
↪'Surgery-Colon&Rectal', 'Surgery-General', 'Surgery-Maxillofacial',␣
↪'Surgery-Neuro', 'Surgery-Pediatric', 'Surgery-Plastic',␣
↪'Surgery-PlasticwithinHeadandNeck', 'Surgery-Thoracic', 'Surgery-Vascular',␣
↪'SurgicalSpecialty', 'Urology', 'Neurophysiology'
'max_glu_serum'             '>200', '>300', 'None', 'Norm'
'A1Cresult'                 '>7', '>8', 'None', 'Norm'
'metformin'                 'Down', 'No', 'Steady', 'Up'
'repaglinide'               'Down', 'No', 'Steady', 'Up'
'nateglinide'               'Down', 'No', 'Steady', 'Up'
'chlorpropamide'            'Down', 'No', 'Steady', 'Up'
'glimepiride'               'Down', 'No', 'Steady', 'Up'
'acetohexamide'             'No', 'Steady'
'glipizide'                 'Down', 'No', 'Steady', 'Up'
'glyburide'                 'Down', 'No', 'Steady', 'Up'
'tolbutamide'               'No', 'Steady'
'pioglitazone'              'Down', 'No', 'Steady', 'Up'
'rosiglitazone'             'Down', 'No', 'Steady', 'Up'
'acarbose'                  'Down', 'No', 'Steady', 'Up'
'miglitol'                  'Down', 'No', 'Steady', 'Up'
'troglitazone'              'No', 'Steady'
'tolazamide'                'No', 'Steady', 'Up'
'examide'                   'No'
'citoglipton'               'No'
'insulin'                   'Down', 'No', 'Steady', 'Up'
'glyburide-metformin'       'Down', 'No', 'Steady', 'Up'
'glipizide-metformin'       'No', 'Steady'
'glimepiride-pioglitazone'  'No', 'Steady'
'metformin-rosiglitazone'   'No'
'metformin-pioglitazone'    'No'
```

```
'change'                    'Ch', 'No'
'diabetesMed'               'No', 'Yes'
'readmitted'                '<30', '>30', 'NO'
```

[24]:
```
# TEST CODE

# check that the domain of some features are now switched to `metformin`
print(f"Domain name of 'chlorpropamide': {tfdv.get_feature(schema,␣
 ↪'chlorpropamide').domain}")
print(f"Domain values of 'chlorpropamide': {tfdv.get_domain(schema,␣
 ↪'chlorpropamide').value}")
print(f"Domain name of 'repaglinide': {tfdv.get_feature(schema, 'repaglinide').
 ↪domain}")
print(f"Domain values of 'repaglinide': {tfdv.get_domain(schema, 'repaglinide').
 ↪value}")
print(f"Domain name of 'nateglinide': {tfdv.get_feature(schema, 'nateglinide').
 ↪domain}")
print(f"Domain values of 'nateglinide': {tfdv.get_domain(schema, 'nateglinide').
 ↪value}")
```

```
Domain name of 'chlorpropamide': metformin
Domain values of 'chlorpropamide': ['Down', 'No', 'Steady', 'Up']
Domain name of 'repaglinide': metformin
Domain values of 'repaglinide': ['Down', 'No', 'Steady', 'Up']
Domain name of 'nateglinide': metformin
Domain values of 'nateglinide': ['Down', 'No', 'Steady', 'Up']
```

Expected Output:

```
Domain name of 'chlorpropamide': metformin
Domain values of 'chlorpropamide': ['Down', 'No', 'Steady', 'Up']
Domain name of 'repaglinide': metformin
Domain values of 'repaglinide': ['Down', 'No', 'Steady', 'Up']
Domain name of 'nateglinide': metformin
Domain values of 'nateglinide': ['Down', 'No', 'Steady', 'Up']
```

Let's do a final check of anomalies to see if this solved the issue.

[25]:
```
calculate_and_display_anomalies(serving_stats, schema=schema)
```

```
             Anomaly short description      Anomaly long description
Feature name
'readmitted'  Column dropped            Column is completely missing
```

You should now see the `metformin-pioglitazone` and `metformin-rosiglitazone` features dropped from the output anomalies.

Exercise 9: Detecting anomalies with environments

There is still one thing to address. The `readmitted` feature (which is the label column) showed up as an anomaly ('Column dropped'). Since labels are not expected in the serving data, let's tell

TFDV to ignore this detected anomaly.

This requirement of introducing slight schema variations can be expressed by using environments. In particular, features in the schema can be associated with a set of environments using `default_environment`, `in_environment` and `not_in_environment`.

[26]:
```
# All features are by default in both TRAINING and SERVING environments.
schema.default_environment.append('TRAINING')
schema.default_environment.append('SERVING')
```

Complete the code below to exclude the `readmitted` feature from the `SERVING` environment.

To achieve this, you can use the `tfdv.get_feature()` function to get the `readmitted` feature from the inferred schema and use its `not_in_environment` attribute to specify that `readmitted` should be removed from the `SERVING` environment's schema. This **attribute is a list** so you will have to **append** the name of the environment that you wish to omit this feature for.

To be more explicit, given a feature you can do something like:

`feature.not_in_environment.append('NAME_OF_ENVIRONMENT')`

The function `tfdv.get_feature` receives the following parameters:

- `schema`: The schema.
- `feature_path`: The path of the feature to obtain from the schema. In this case this is equal to the name of the feature.

[27]:
```
### START CODE HERE
# Specify that 'readmitted' feature is not in SERVING environment.
# HINT: Append the 'SERVING' environmnet to the not_in_environment attribute of␣
↪the feature
tfdv.get_feature(schema, 'readmitted').not_in_environment.append('SERVING')

# HINT: Calculate anomalies with the validate_statistics function by using the␣
↪serving statistics,
# inferred schema and the SERVING environment parameter.
serving_anomalies_with_env = tfdv.validate_statistics(serving_stats, schema,␣
↪environment='SERVING')
### END CODE HERE
```

You should see "No anomalies found" by running the cell below.

[28]:
```
# Display anomalies
tfdv.display_anomalies(serving_anomalies_with_env)
```

```
<IPython.core.display.HTML object>
```

Now you have succesfully addressed all anomaly-related issues!

7 - Check for Data Drift and Skew

During data validation, you also need to check for data drift and data skew between the training and serving data. You can do this by specifying the skew_comparator and drift_comparator in the schema.

Drift and skew is expressed in terms of L-infinity distance which evaluates the difference between vectors as the greatest of the differences along any coordinate dimension.

You can set the threshold distance so that you receive warnings when the drift is higher than is acceptable. Setting the correct distance is typically an iterative process requiring domain knowledge and experimentation.

Let's check for the skew in the **diabetesMed** feature and drift in the **payer_code** feature.

[29]:
```
# Calculate skew for the diabetesMed feature
diabetes_med = tfdv.get_feature(schema, 'diabetesMed')
diabetes_med.skew_comparator.infinity_norm.threshold = 0.03 # domain knowledge␣
↪helps to determine this threshold

# Calculate drift for the payer_code feature
payer_code = tfdv.get_feature(schema, 'payer_code')
payer_code.drift_comparator.infinity_norm.threshold = 0.03 # domain knowledge␣
↪helps to determine this threshold

# Calculate anomalies
skew_drift_anomalies = tfdv.validate_statistics(train_stats, schema,
                                          previous_statistics=eval_stats,
                                          serving_statistics=serving_stats)

# Display anomalies
tfdv.display_anomalies(skew_drift_anomalies)
```

```
                                       Anomaly short description  \
Feature name
'diabetesMed'  High Linfty distance between training and serving
'payer_code'   High Linfty distance between current and previous


                                                                  ␣
↪                                                                 ␣
↪Anomaly long description
Feature name
'diabetesMed'  The Linfty distance between training and serving is 0.0325464 (up␣
↪to six significant digits), above the threshold 0.03. The feature value with␣
↪maximum difference is: No
'payer_code'   The Linfty distance between current and previous is 0.0342144 (up␣
↪to six significant digits), above the threshold 0.03. The feature value with␣
↪maximum difference is: MC
```

In both of these cases, the detected anomaly distance is not too far from the threshold value of `0.03`. For this exercise, let's accept this as within bounds (i.e. you can set the distance to something like `0.035` instead).

However, if the anomaly truly indicates a skew and drift, then further investigation is

necessary as this could have a direct impact on model performance.

8 - Display Stats for Data Slices

Finally, you can slice the dataset and calculate the statistics for each unique value of a feature. By default, TFDV computes statistics for the overall dataset in addition to the configured slices. Each slice is identified by a unique name which is set as the dataset name in the DatasetFeatureStatistics protocol buffer. Generating and displaying statistics over different slices of data can help track model and anomaly metrics.

Let's first define a few helper functions to make our code in the exercise more neat.

[30]:
```
def split_datasets(dataset_list):
    '''
    split datasets.

            Parameters:
                    dataset_list: List of datasets to split

            Returns:
                    datasets: sliced data
    '''
    datasets = []
    for dataset in dataset_list.datasets:
        proto_list = DatasetFeatureStatisticsList()
        proto_list.datasets.extend([dataset])
        datasets.append(proto_list)
    return datasets


def display_stats_at_index(index, datasets):
    '''
    display statistics at the specified data index

            Parameters:
                    index : index to show the anomalies
                    datasets: split data

            Returns:
                    display of generated sliced data statistics at the␣
 ↪specified index
    '''
    if index < len(datasets):
        print(datasets[index].datasets[0].name)
        tfdv.visualize_statistics(datasets[index])
```

The function below returns a list of `DatasetFeatureStatisticsList` protocol buffers. As shown in the ungraded lab, the first one will be for `All Examples` followed by individual slices through the feature you specified.

To configure TFDV to generate statistics for dataset slices, you will use the function `tfdv.StatsOptions()` with the following 4 arguments:

- `schema`
- `slice_functions` passed as a list.
- `infer_type_from_schema` set to True.
- `feature_whitelist` set to the approved features.

Remember that `slice_functions` only work with `generate_statistics_from_csv()` so you will need to convert the dataframe to CSV.

[31]:
```python
def sliced_stats_for_slice_fn(slice_fn, approved_cols, dataframe, schema):
    '''
    generate statistics for the sliced data.

            Parameters:
                    slice_fn : slicing definition
                    approved_cols: list of features to pass to the statistics␣
↪options
                    dataframe: pandas dataframe to slice
                    schema: the schema

            Returns:
                    slice_info_datasets: statistics for the sliced dataset
    '''
    # Set the StatsOptions
    slice_stats_options = tfdv.StatsOptions(schema=schema,
                                            slice_functions=[slice_fn],
                                            infer_type_from_schema=True,
                                            feature_whitelist=approved_cols)

    # Convert Dataframe to CSV since `slice_functions` works only with `tfdv.
↪generate_statistics_from_csv`
    CSV_PATH = 'slice_sample.csv'
    dataframe.to_csv(CSV_PATH)

    # Calculate statistics for the sliced dataset
    sliced_stats = tfdv.generate_statistics_from_csv(CSV_PATH,␣
↪stats_options=slice_stats_options)

    # Split the dataset using the previously defined split_datasets function
    slice_info_datasets = split_datasets(sliced_stats)

    return slice_info_datasets
```

With that, you can now use the helper functions to generate and visualize statistics for the sliced datasets.

[32]:
```
# Generate slice function for the `medical_speciality` feature
slice_fn = slicing_util.get_feature_value_slicer(features={'medical_specialty':␣
↪None})

# Generate stats for the sliced dataset
slice_datasets = sliced_stats_for_slice_fn(slice_fn, approved_cols,␣
↪dataframe=train_df, schema=schema)

# Print name of slices for reference
print(f'Statistics generated for:\n')
print('\n'.join([sliced.datasets[0].name for sliced in slice_datasets]))

# Display at index 10, which corresponds to the slice named␣
↪`medical_specialty_Gastroenterology`
display_stats_at_index(10, slice_datasets)
```

```
Statistics generated for:

All Examples
medical_specialty_Orthopedics
medical_specialty_InternalMedicine
medical_specialty_Cardiology
medical_specialty_Family/GeneralPractice
medical_specialty_Surgery-General
medical_specialty_Emergency/Trauma
medical_specialty_Nephrology
medical_specialty_Surgery-Neuro
medical_specialty_Oncology
medical_specialty_Gastroenterology
medical_specialty_Orthopedics-Reconstructive
medical_specialty_ObstetricsandGynecology
medical_specialty_Surgery-Cardiovascular/Thoracic
medical_specialty_Radiologist
medical_specialty_Urology
medical_specialty_Surgery-Vascular
medical_specialty_Hematology/Oncology
medical_specialty_Neurology
medical_specialty_Psychology
medical_specialty_Psychiatry
medical_specialty_PhysicalMedicineandRehabilitation
medical_specialty_Pulmonology
medical_specialty_Otolaryngology
medical_specialty_Obsterics&Gynecology-GynecologicOnco
medical_specialty_Endocrinology
medical_specialty_Anesthesiology
medical_specialty_Pediatrics-Endocrinology
medical_specialty_Radiology
```

```
medical_specialty_Pediatrics
medical_specialty_Pediatrics-Pulmonology
medical_specialty_Osteopath
medical_specialty_Surgery-Plastic
medical_specialty_Podiatry
medical_specialty_Surgery-Thoracic
medical_specialty_Rheumatology
medical_specialty_Obstetrics
medical_specialty_Pediatrics-AllergyandImmunology
medical_specialty_Surgery-Cardiovascular
medical_specialty_Anesthesiology-Pediatric
medical_specialty_Pathology
medical_specialty_Pediatrics-CriticalCare
medical_specialty_PhysicianNotFound
medical_specialty_Gynecology
medical_specialty_AllergyandImmunology
medical_specialty_Surgery-Maxillofacial
medical_specialty_Hospitalist
medical_specialty_Hematology
medical_specialty_Surgeon
medical_specialty_Proctology
medical_specialty_InfectiousDiseases
medical_specialty_Psychiatry-Child/Adolescent
medical_specialty_SurgicalSpecialty
medical_specialty_Ophthalmology
medical_specialty_Surgery-Pediatric
medical_specialty_Pediatrics-Neurology
medical_specialty_Surgery-PlasticwithinHeadandNeck
medical_specialty_OutreachServices
medical_specialty_Pediatrics-Hematology-Oncology
medical_specialty_Dentistry
medical_specialty_Pediatrics-EmergencyMedicine
medical_specialty_Psychiatry-Addictive
medical_specialty_Surgery-Colon&Rectal
medical_specialty_Pediatrics-InfectiousDiseases
medical_specialty_Dermatology
medical_specialty_Perinatology
medical_specialty_SportsMedicine
medical_specialty_Cardiology-Pediatric
medical_specialty_Speech
medical_specialty_Gastroenterology
```

```
<IPython.core.display.HTML object>
```

If you are curious, try different slice indices to extract the group statistics. For instance, `index=5` corresponds to all `medical_specialty_Surgery-General` records. You can also try slicing through multiple features as shown in the ungraded lab.

Another challenge is to implement your own helper functions. For instance, you can make a `display_stats_for_slice_name()` function so you don't have to determine the index of a slice. If done correctly, you can just do `display_stats_for_slice_name('medical_specialty_Gastroenterology', slice_datasets)` and it will generate the same result as `display_stats_at_index(10, slice_datasets)`.

9 - Freeze the schema

Now that the schema has been reviewed, you will store the schema in a file in its "frozen" state. This can be used to validate incoming data once your application goes live to your users.

This is pretty straightforward using Tensorflow's `io` utils and TFDV's `write_schema_text()` function.

[33]:
```
# Create output directory
OUTPUT_DIR = "output"
file_io.recursive_create_dir(OUTPUT_DIR)

# Use TensorFlow text output format pbtxt to store the schema
schema_file = os.path.join(OUTPUT_DIR, 'schema.pbtxt')

# write_schema_text function expect the defined schema and output path as␣
↪parameters
tfdv.write_schema_text(schema, schema_file)
```

After submitting this assignment, you can click the Jupyter logo in the left upper corner of the screen to check the Jupyter filesystem. The `schema.pbtxt` file should be inside the `output` directory.

Congratulations on finishing this week's assignment! A lot of concepts where introduced and now you should feel more familiar with using TFDV for inferring schemas, anomaly detection and other data-related tasks.

Keep it up!