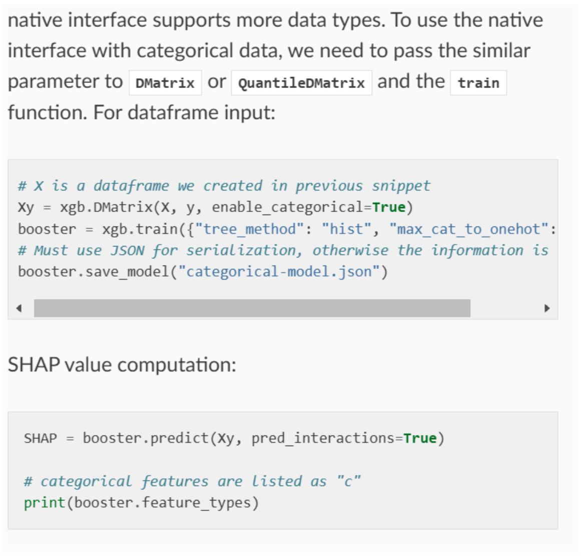
Categorical Data

- Training with scikit-learn interface
    - `X["cat_feature"].astype("category")`
    - Need to specify data type as category
    - Also need to specify enable_categorical parameter in the xgb classifier model
        - `xgb.XGBClassifier(tree_method="hist", enable_categorical=True, device="cuda")`
    - One hot encoding is also an option instead of using xgb categorization

- Optimal Partitioning
    - Technique for partitioning the categorical predictors for each node split
    - To find the best way to split the data in order to create a strong model
    - One only needs to look at sorted partitions instead of all possible permutations to accomplish this
    - XGB splits the data, separating features
        - Measures good splits based on how well it separates to make predictions accurate
        - Uses algorithms instead of testing every single split
    - `max_cat_to_onehot`, which controls whether one-hot encoding or partitioning should be used for each feature

- Using Native Interface
    - Scikit-learn interface does not allow SHAP value to be computed directly



native interface supports more data types. To use the native interface with categorical data, we need to pass the similar parameter to `DMatrix` or `QuantileDMatrix` and the `train` function. For dataframe input:

```
# X is a dataframe we created in previous snippet
Xy = xgb.DMatrix(X, y, enable_categorical=True)
booster = xgb.train({"tree_method": "hist", "max_cat_to_onehot":
# Must use JSON for serialization, otherwise the information is
booster.save_model("categorical-model.json")
```

SHAP value computation:

```
SHAP = booster.predict(Xy, pred_interactions=True)

# categorical features are listed as "c"
print(booster.feature_types)
```

        - For numerical data the feature can be q or float, while categorical feature is specified as c
            - `# "q" is numerical feature, while "c" is categorical feature`
            - `ft = ["q", "c", "c"]`
            - `X: np.ndarray = load_my_data()`
            - `assert X.shape[1] == 3`
            - `Xy = xgb.DMatrix(X, y, feature_types=ft, enable_categorical=True)`

- Data Consistency
    - To keep data consistent, encoding should be used on all data types together
        - For example, do not encode train and test data separately
    - Encoder is often integrated into the data engineering pipeline

- Miscellaneous
    - By default, XGB assumes input categories are integers
    - However, user might provide inputs with invalid values due to mistakes or missing values in training dataset. It can be negative value, integer values that can not be accurately represented by 32-bit floating point, or values that are larger than actual number of unique categories. During training this is validated but for prediction it's treated as the same as not-chosen category for performance reasons.