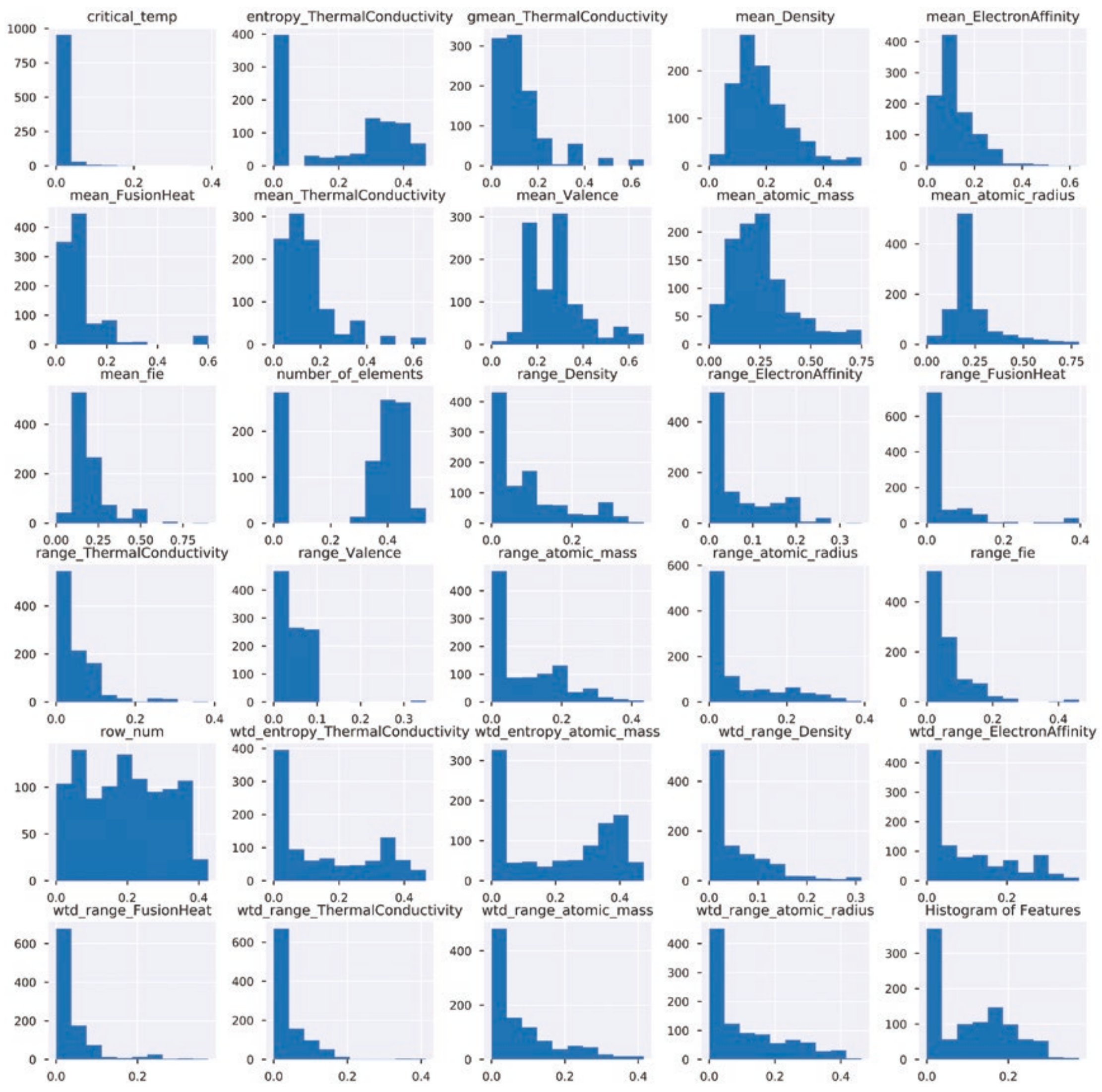

Figure 44-2. *Histogram showing variable distribution*

Spot Checking Machine Learning Algorithms

With our reduced dataset, let's sample a few candidate algorithms to have an idea on their performance and which is more likely to work best for this problem domain. Let's take the following steps:

- The dataset is split into a design matrix and their corresponding label vector.

```
# split features and labels
dataset_y = dataset['critical_temp']
dataset_X = dataset.drop(['critical_temp', 'row_num'], axis=1)
```

- Randomly split the dataset into a training set and a test set.

```
# train-test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(dataset_X,
dataset_y, shuffle=True)
```

- Outline the candidate algorithms to create a model.

```
# spot-check ML algorithms
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt
```

- Create a dictionary of the candidate algorithms.

```
ml_models = {
    'Linear Reg.': LinearRegression(),
    'Dec. Trees': DecisionTreeRegressor(),
    'Rand. Forest': RandomForestRegressor(),
    'SVM': SVR(),
    'XGBoost': XGBRegressor(),
    'NNets': MLPRegressor(warm_start=True, early_stopping=True,
             learning_rate='adaptive')
}
```

- For each candidate algorithm, train with the training set and evaluate on the hold-out test set.

```
ml_results = {}
for name, model in ml_models.items():
    # fit model on training data
    model.fit(X_train, y_train)
    # make predictions for test data
    prediction = model.predict(X_test)
    # evaluate predictions
    rmse = sqrt(mean_squared_error(y_test, prediction))
    # append accuracy results to dictionary
    ml_results[name] = rmse
    print('RMSE: {} -> {}'.format(name, rmse))
```

```
'Output':
RMSE: SVM -> 0.0748587427887
RMSE: XGBoost -> 0.0222440358318
RMSE: Rand. Forest -> 0.0227742725953
RMSE: Linear Reg. -> 0.025615918858
RMSE: Dec. Trees -> 0.0269103025639
RMSE: NNets -> 0.0289585489638
```

- The plots of the model performances are shown in Figure 44-3.

```
plt.plot(ml_results.keys(), ml_results.values(), 'o')
plt.title("RMSE estimates for ML algorithms")
plt.xlabel('Algorithms')
plt.ylabel('RMSE')
```

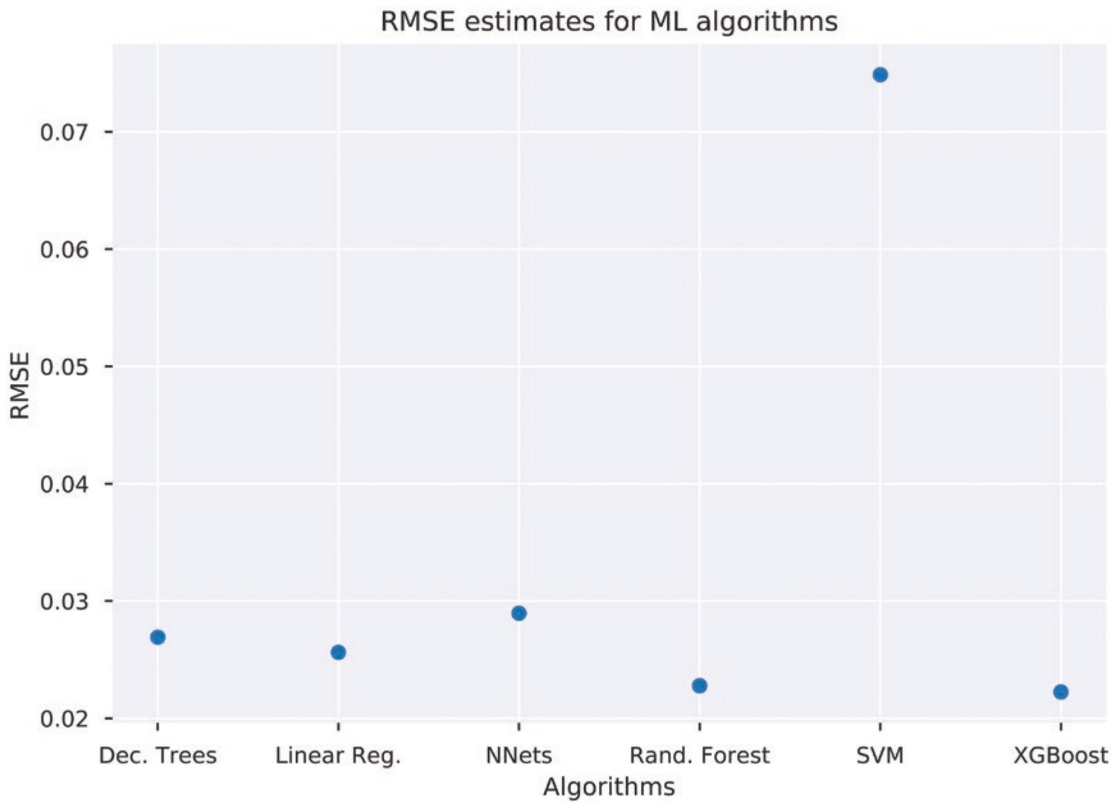

Figure 44-3. *RMSE estimates for ML algorithms*

Dataflow and TensorFlow Transform for Large-Scale Data Processing

In this section, we use Google Cloud Dataflow to carry out large-scale data processing on humongous datasets. Google Dataflow as earlier discussed is a serverless, parallel, and distributed infrastructure for running jobs for batch and stream data processing. Dataflow is a vital component in architecting a production pipeline for building and deploying large-scale machine learning products. In conjunction with Cloud Dataflow, we use TensorFlow Transform (TFT), a library built for preprocessing with Tensorflow. The goal of using TFT is to have a consistent set of transformation operations applied to the dataset when the model is trained and when it is served or deployed for consumption. In the following steps, each code block is executed in a Notebook cell: