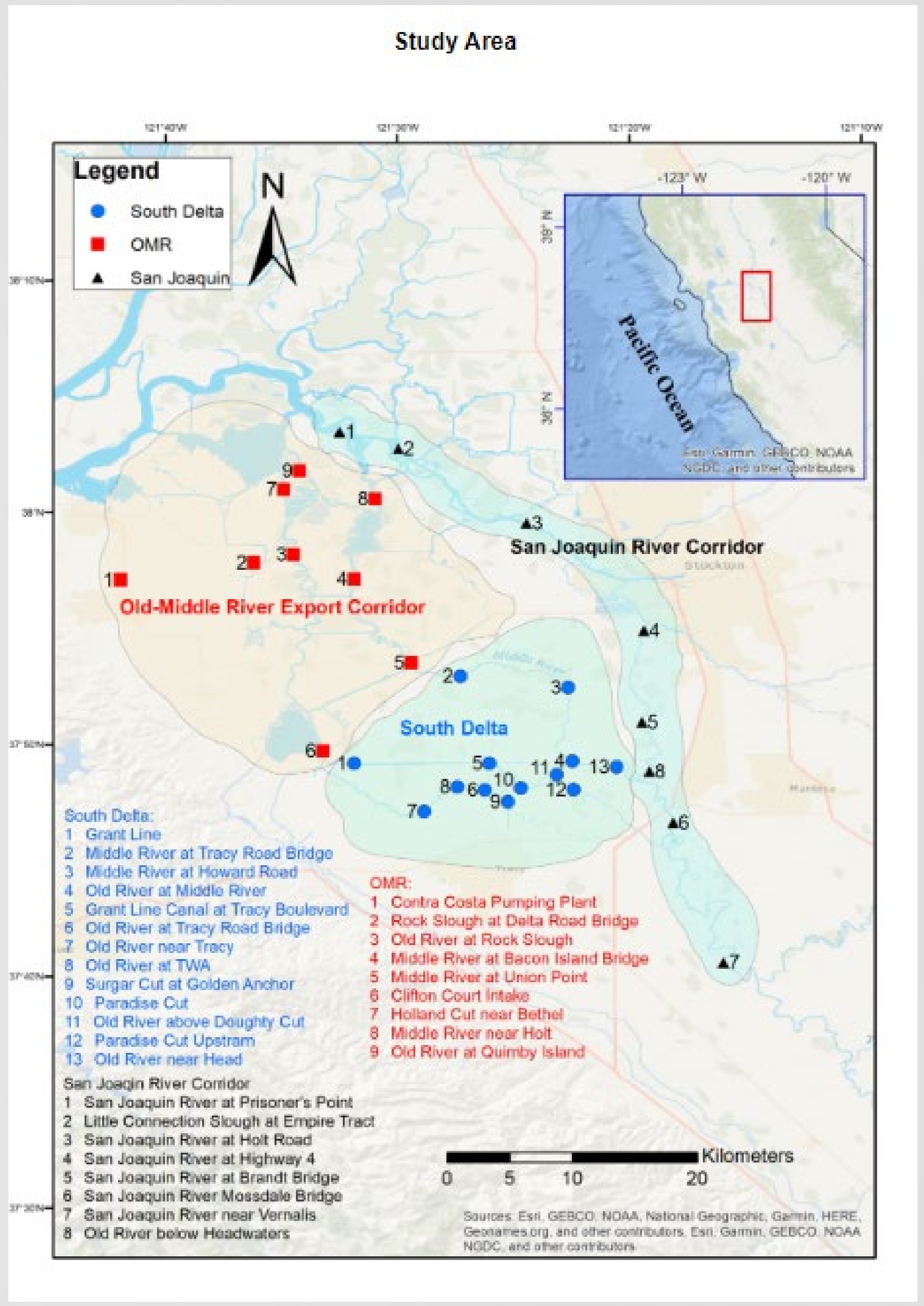
Hands-on

Goal: Build and evaluate ML models to predict ion concentrations in the Interior Delta

What will be covered:

1. **Data Loading**: Import water quality datasets
2. **Data Visualization**: Explore relationships between variables
3. **Preprocessing**: Feature engineering, scaling, and encoding
4. **Model Training**: Train multiple ML models
5. **Model Evaluation**: Compare performance across models
6. **Interactive Dashboard**: Create a user-friendly interface


Study Area
Legend
● South Delta
🟥 OMR
▲ San Joaquin
N
Pacific Ocean
San Joaquin River Corridor
Old-Middle River Export Corridor
South Delta
South Delta:
1 Grant Line
2 Middle River at Tracy Road Bridge
3 Middle River at Howard Road
4 Old River at Middle River
5 Grant Line Canal at Tracy Boulevard
6 Old River at Tracy Road Bridge
7 Old River near Tracy
8 Old River at TWA
9 Surgar Cut at Golden Anchor
10 Paradise Cut
11 Old River above Doughty Cut
12 Paradise Cut Upstram
13 Old River near Head
OMR:
1 Contra Costa Pumping Plant
2 Rock Slough at Delta Road Bridge
3 Old River at Rock Slough
4 Middle River at Bacon Island Bridge
5 Middle River at Union Point
6 Clifton Court Intake
7 Holland Cut near Bethel
8 Middle River near Holt
9 Old River at Quimby Island
San Joagin River Corridor
1 San Joaquin River at Prisoner's Point
2 Little Connection Slough at Empire Tract
3 San Joaquin River at Holt Road
4 San Joaquin River at Highway 4
5 San Joaquin River at Brandt Bridge
6 San Joaquin River Mossdale Bridge
7 San Joaquin River near Vernalis
8 Old River below Headwaters
Kilometers
0 5 10 20
Sources: Esri, GEBCO, NOAA, National Geographic, Garmin, HERE, Geonames.org, and other contributors. Esri, Garmin, GEBCO, NOAA NGDC, and other contributors

Disclaimer: This notebook is created for educational purposes as part of a machine learning workshop. The models and methods demonstrated here are simplified examples and not necessarily optimized for production use. The actual models used in production environments may incorporate additional features, more sophisticated techniques, and more extensive validation.

CALIFORNIA DEPARTMENT OF WATER RESOURCES

Data Preprocessing

Scaling, and encoding

Predictors:

- **1- EC**
- **2- Sacramento X2**
- **3- Month**
- **4- Water Year Type**
- **5- Location**

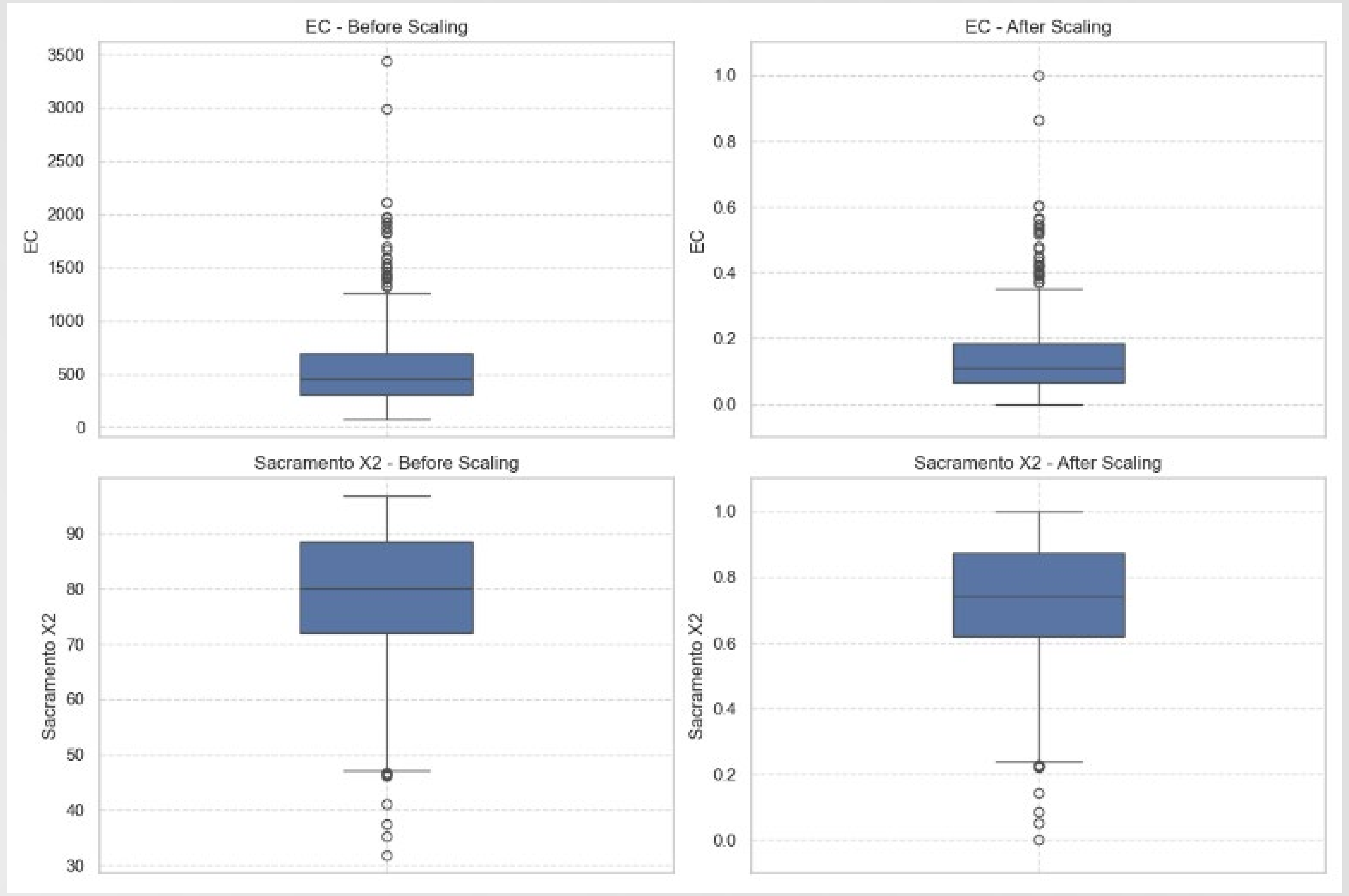
Numerical values: feature scaling

Prevent features with larger magnitudes from dominating those with smaller magnitudes

Categorical values: hot encoding

Enable machine learning models to interpret and utilize non-numeric data effectively

Before **After**



WYT
W
D
C
AN
BN

→

W	D	C	AN	BN
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

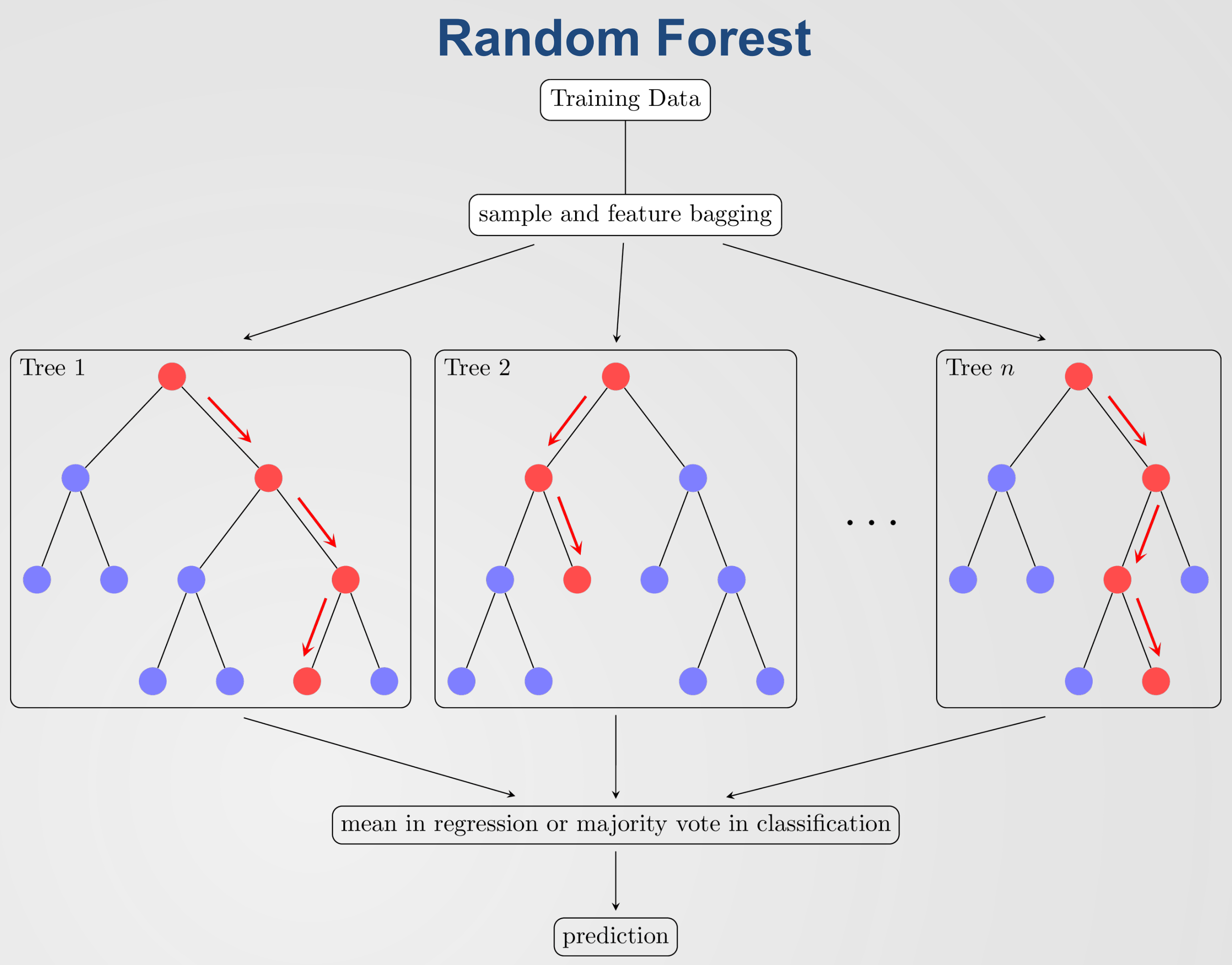
Binary encoded

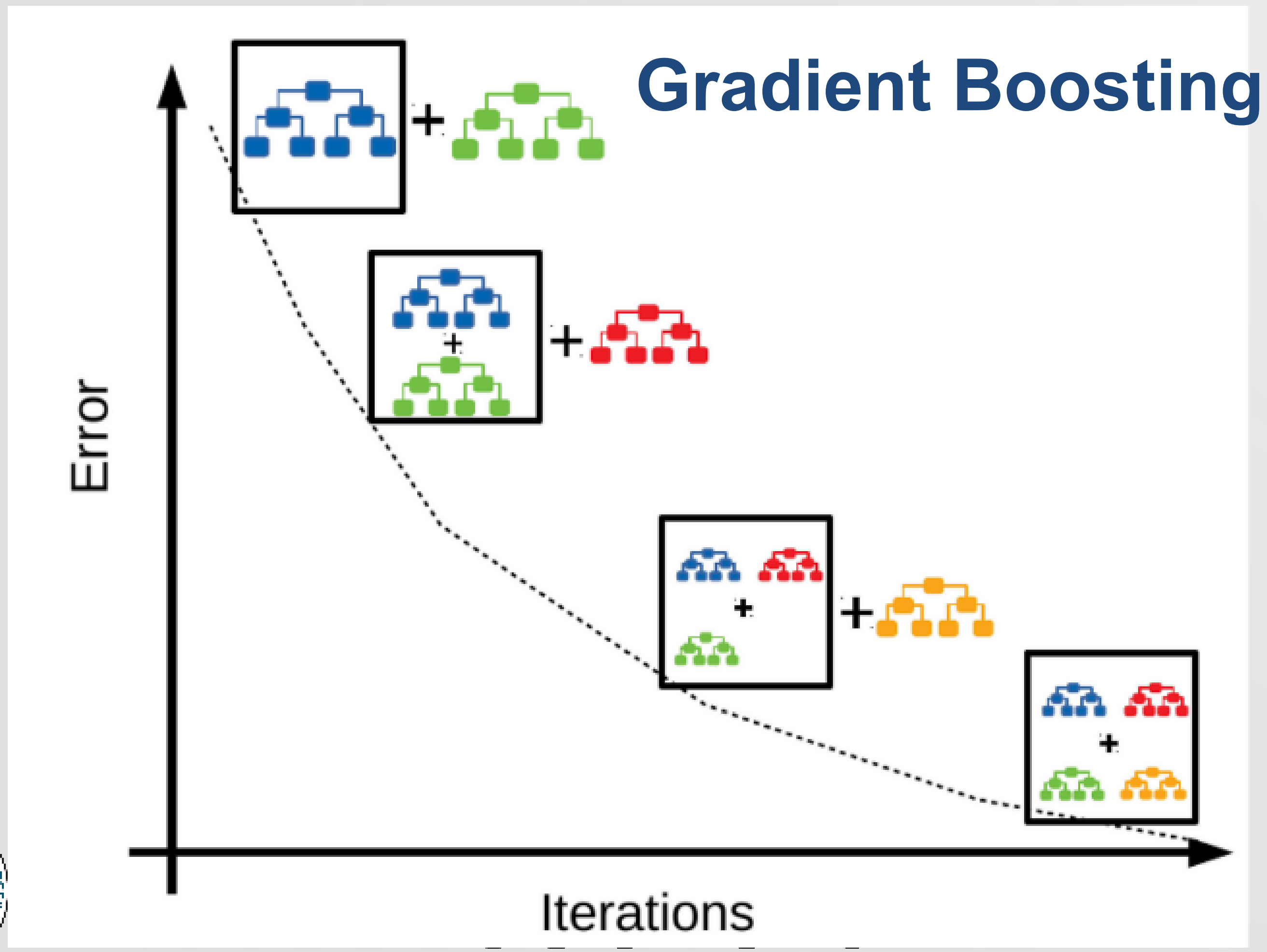
CALIFORNIA DEPARTMENT OF WATER RESOURCES

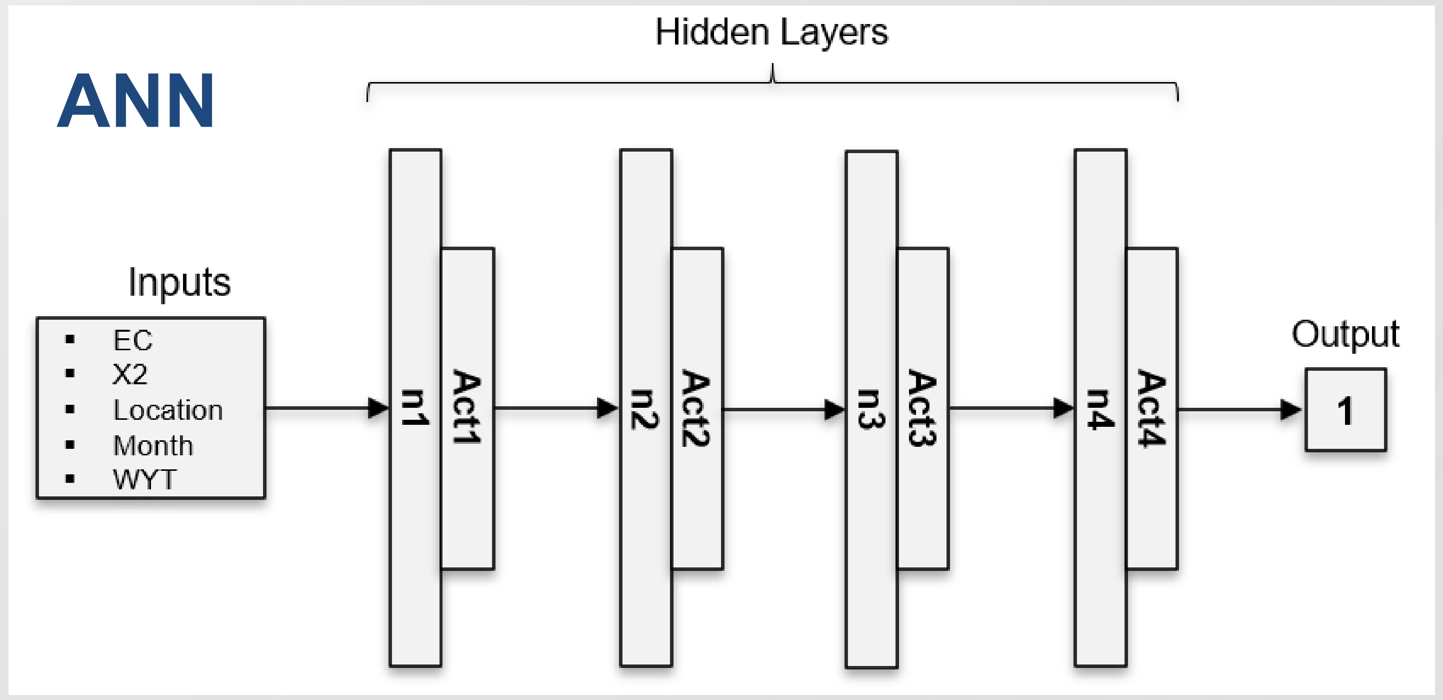
Model Selection

Models:

- Ensemble method:
  - Random Forest (RF)
  - Gradient Boosting (GB)
- Artificial Neural Networks (ANN)


Random Forest
Training Data
sample and feature bagging
Tree 1
Tree 2
. . .
Tree n
mean in regression or majority vote in classification
prediction

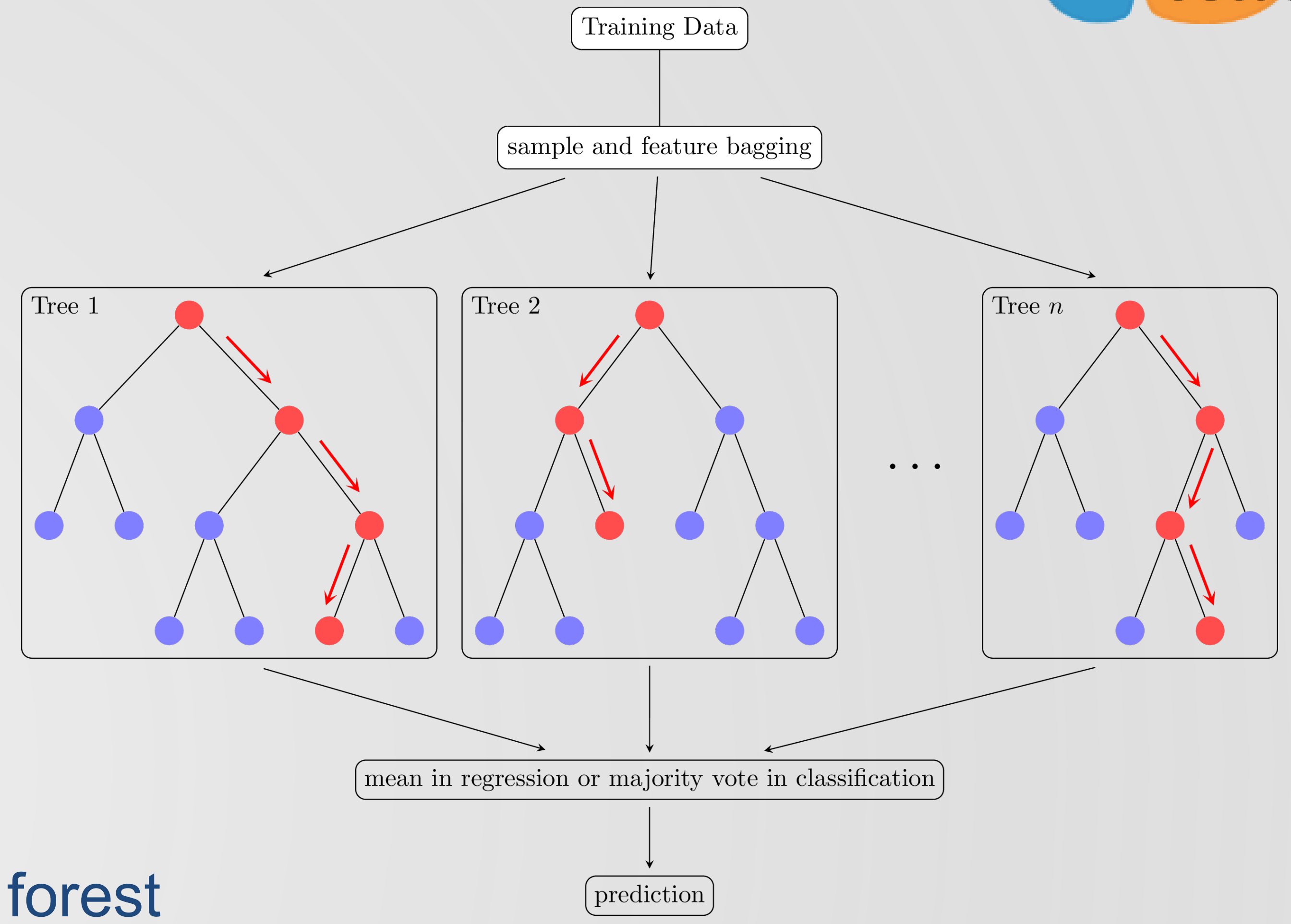


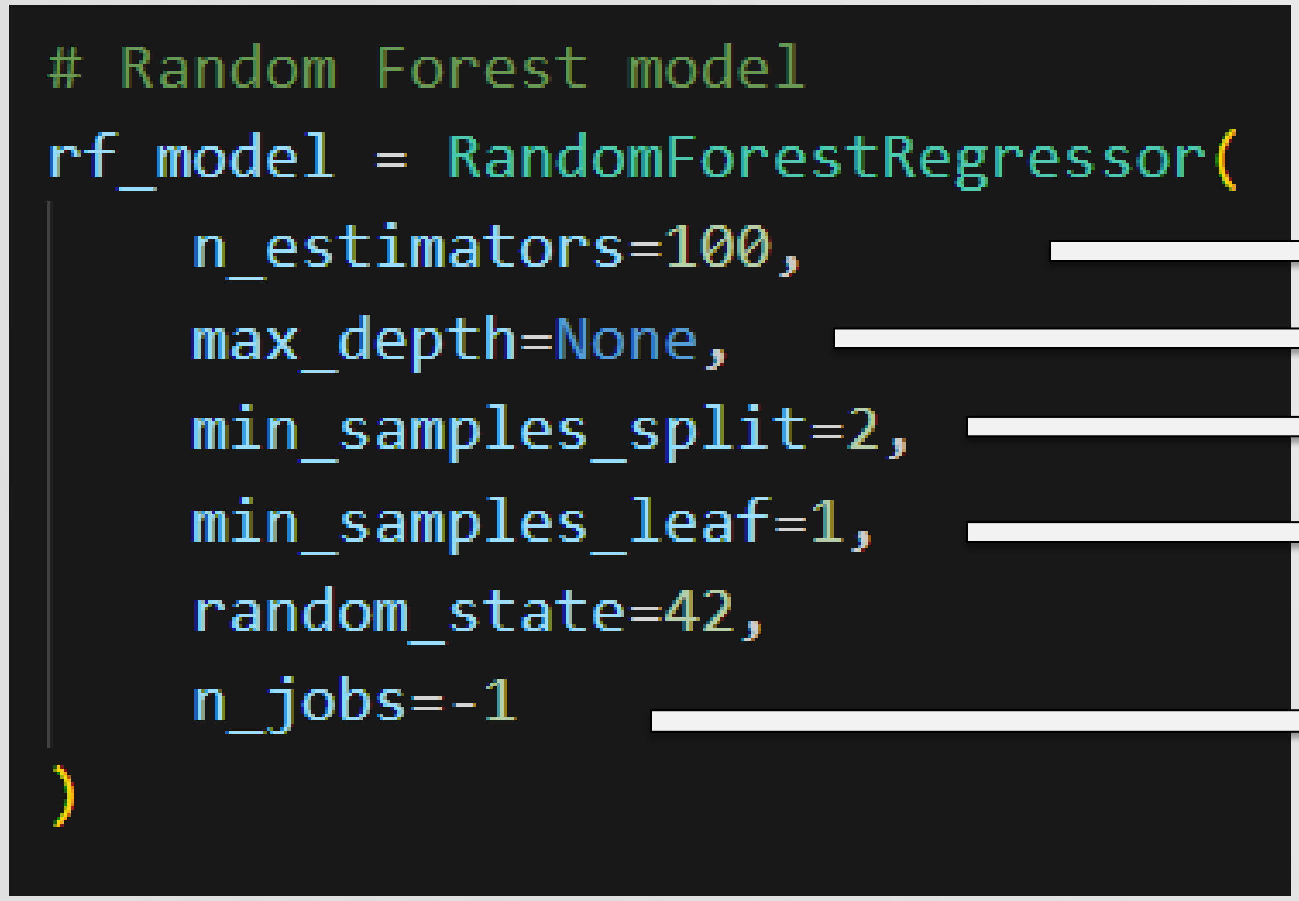


Random Forest (RF)

Key Characteristics:

- Trees built in parallel (independent)
- Random subset of data for each tree (bootstrapping)
- Random subset of features at each split
- Equal weight for all trees


Training Data
sample and feature bagging
Tree 1
Tree 2
Tree n
. . .
mean in regression or majority vote in classification
prediction



```
# Random Forest model
rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42,
    n_jobs=-1
)
```

n_estimators=100 → The number of trees in the forest

max_depth=None → The maximum depth of the tree.

min_samples_split=2 → The minimum number of samples required to split an internal node

min_samples_leaf=1 → The minimum number of samples required to be at a leaf node

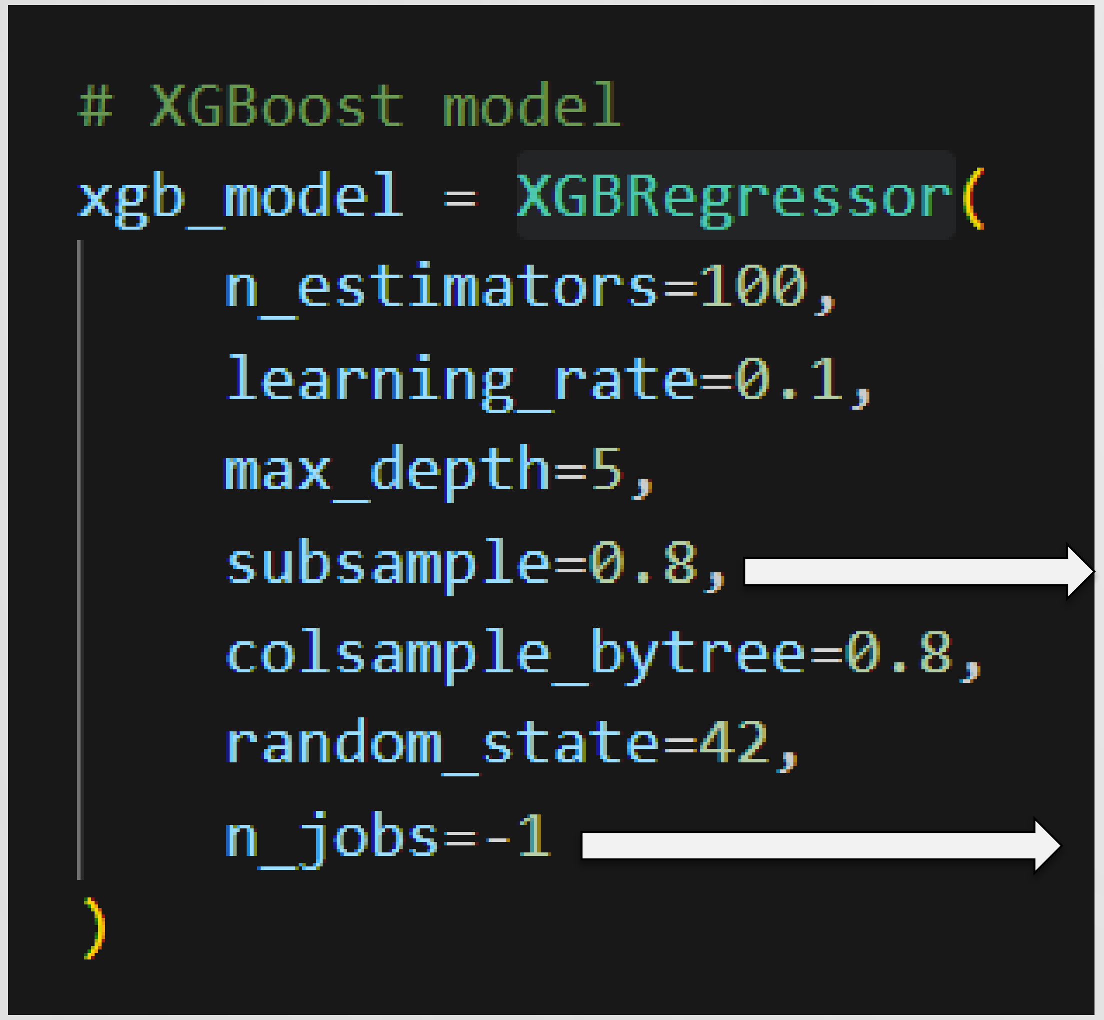
n_jobs=-1 → Means using all processors

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Gradient Boosting (GB)

Key Characteristics:

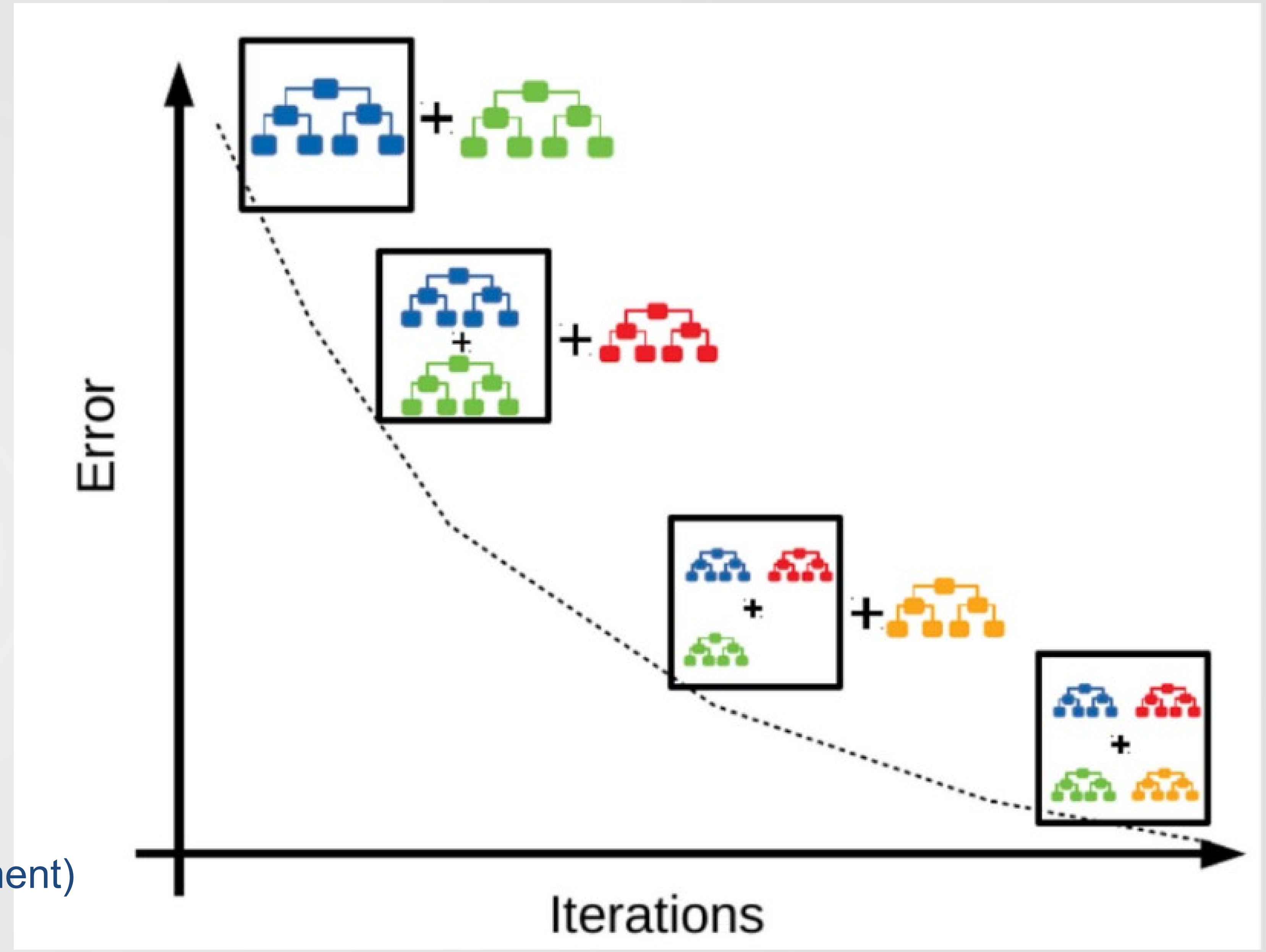
- Trees built sequentially (dependent)
- Each tree focuses on previous errors
- Weighted combination of trees
- Learning rate controls contribution of each tree



```
# XGBoost model
xgb_model = XGBRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    n_jobs=-1
)
```

Subsampling (without replacement)

Means using all processors

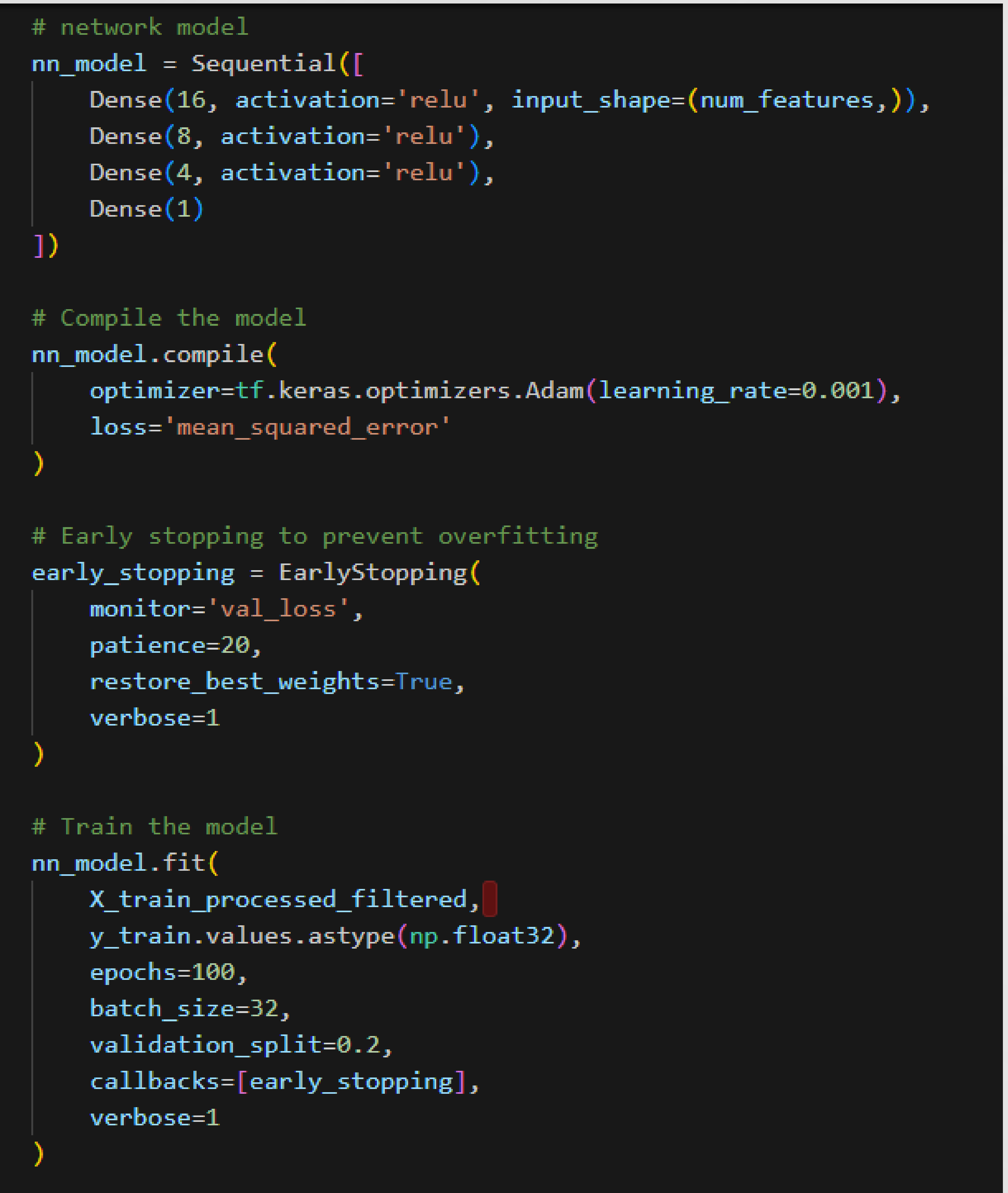


CALIFORNIA DEPARTMENT OF WATER RESOURCES

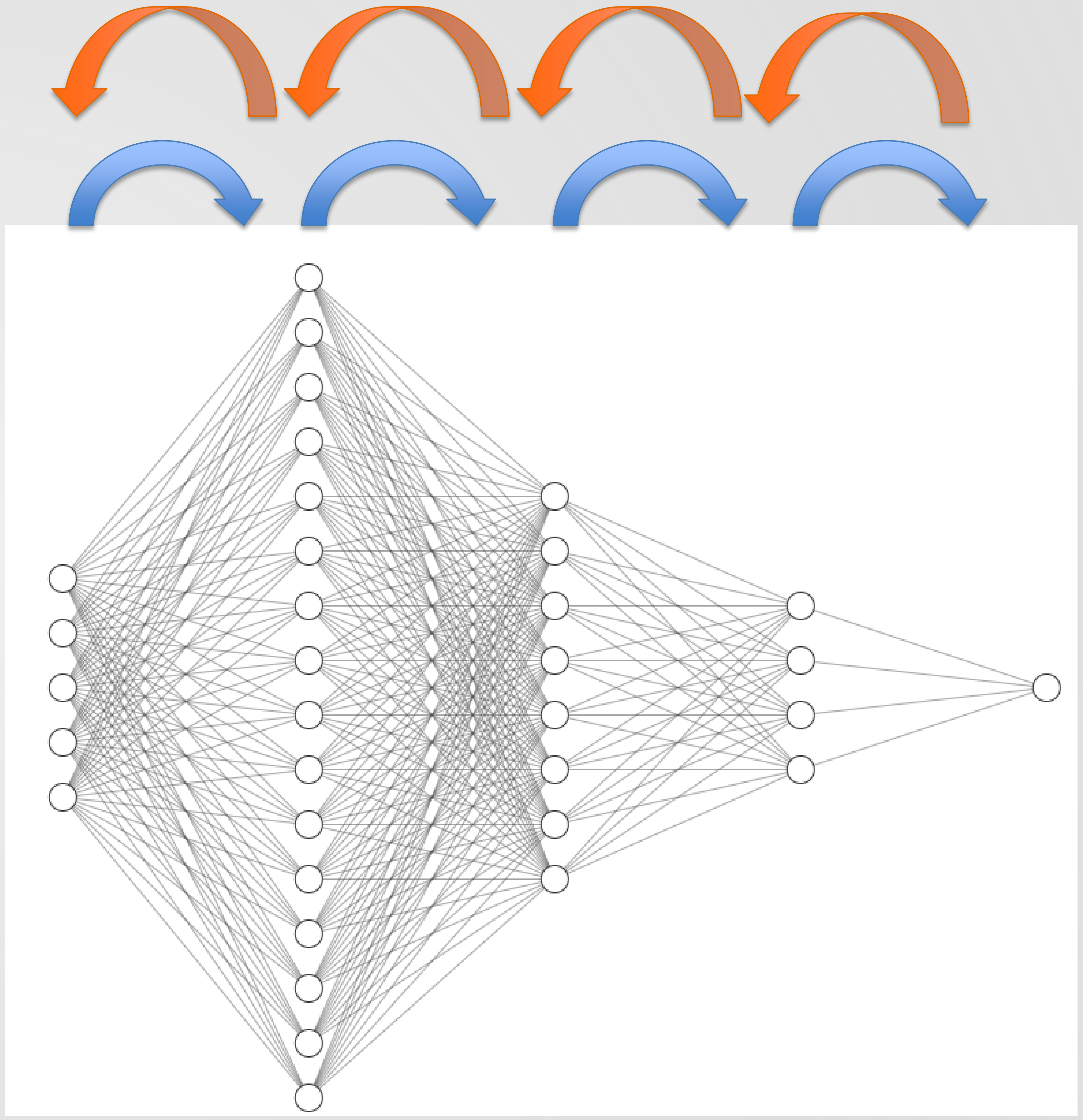
https://xgboost.readthedocs.io/en/release_3.0.0/python/python_api.html#xgboost.XGBRegressor.fit

Artificial Neural Networks (ANN)

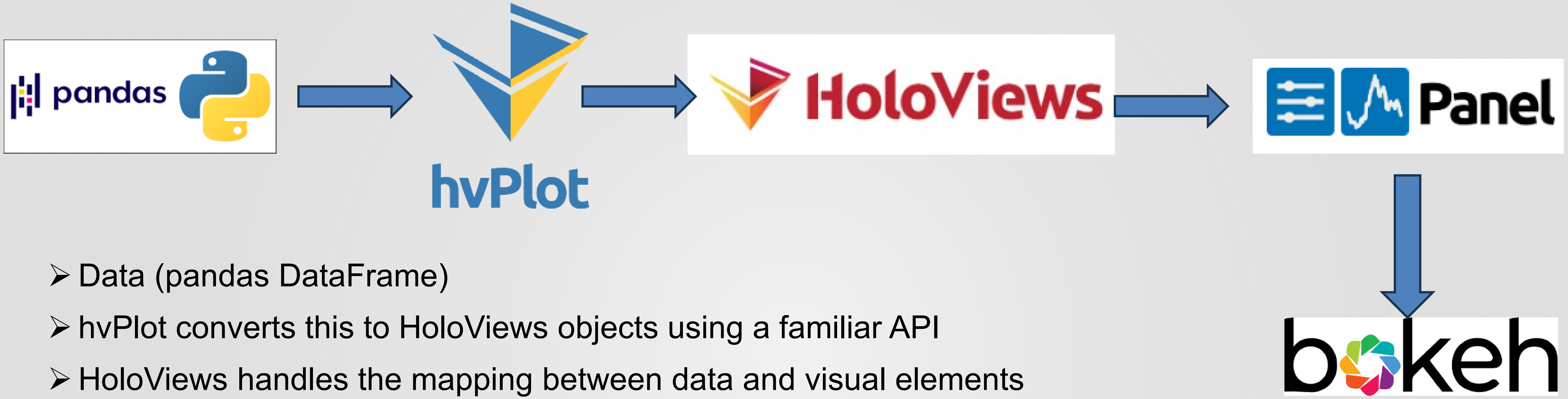
TensorFlow



```
# network model
nn_model = Sequential([
    Dense(16, activation='relu', input_shape=(num_features,)),
    Dense(8, activation='relu'),
    Dense(4, activation='relu'),
    Dense(1)
])

# Compile the model
nn_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='mean_squared_error'
)

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=20,
    restore_best_weights=True,
    verbose=1
)

# Train the model
nn_model.fit(
    X_train_processed_filtered,
    y_train.values.astype(np.float32),
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)
```



Dashboard



- ➢ Data (pandas DataFrame)
- ➢ hvPlot converts this to HoloViews objects using a familiar API
- ➢ HoloViews handles the mapping between data and visual elements
- ➢ Bokeh (as the backend renderer) creates the actual JavaScript-based visualization

  (https://docs.bokeh.org/en/latest/docs/gallery.html)
- ➢ Panel arranges these visualizations and adds interactive controls

  (https://panel.holoviz.org/reference/index.html#widgets)

Bokeh: Low-level JavaScript-based plotting (handles the actual rendering)
HoloViews: Mid-level data-centric visualization objects
hvPlot: High-level pandas-like plotting API
Panel: Dashboard composition and widget integration

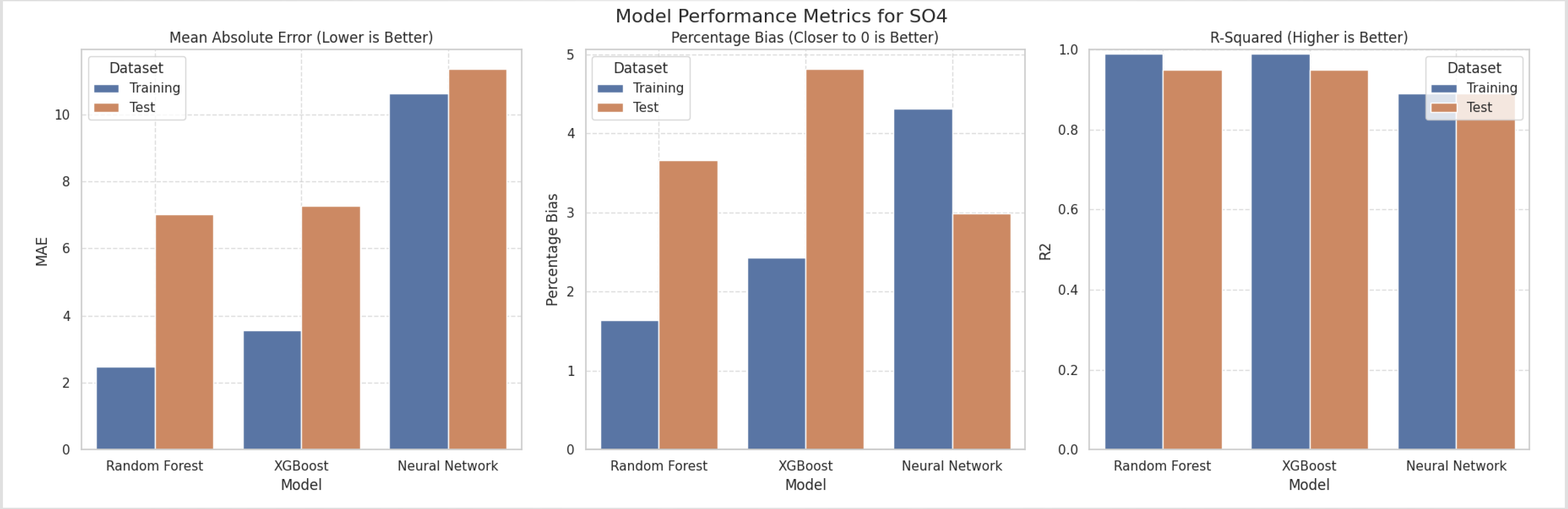
Practice Problem 1: Simulate SO4 Concentration

Objective: Repeat the modeling process we completed for Bromine (Br) but now for Sulfate (SO4).

Step 1: In the Part 3 (Preprocessing) Change TARGET_ION = 'Br' to TARGET_ION = 'SO4'

Step 2: Run all code cells from Section 3 (Preprocessing) to the end

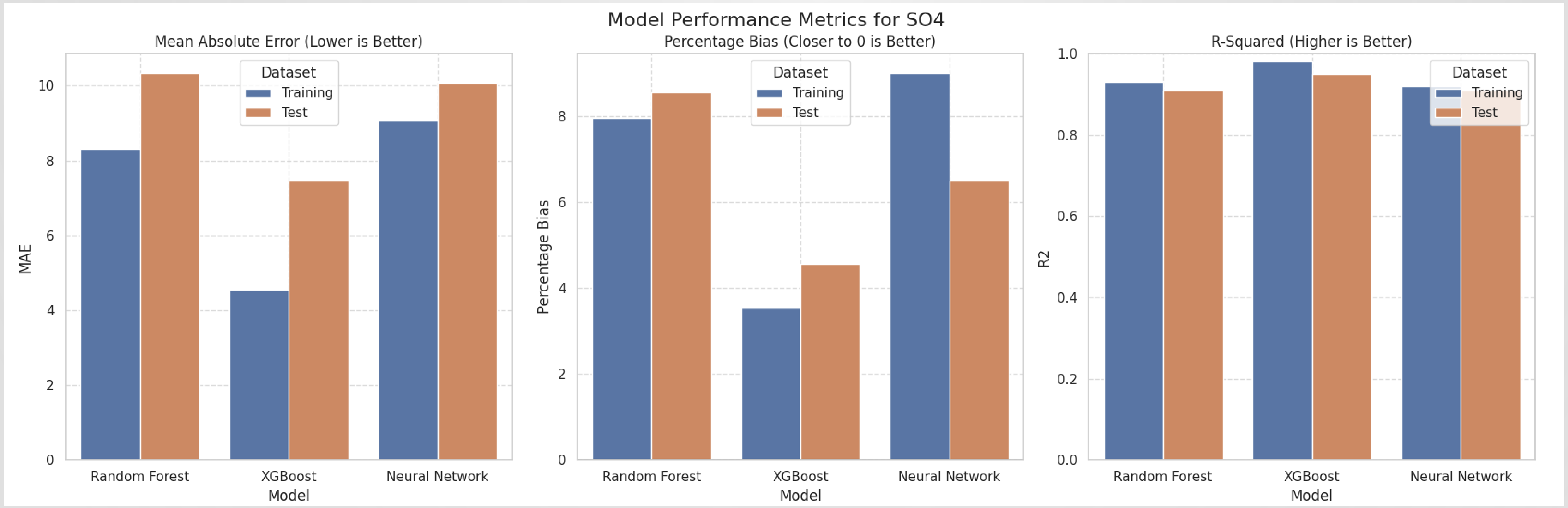
Step 3: Review the results and identify which model performs best for SO4 prediction



Practice Problem 2: Hyperparameter adjustment

- **Random Forest model**:
  - Limit tree depth to 4 (previously unlimited)
- **XGBoost model**:
  - Reduce maximum tree depth from 5 to 4
- **Neural Network model**:
  - Make the network bigger:

    (32→16→8 neurons instead of 16→8→4)
  - Train for more epochs (200 instead of 100)
  - Increase patience (30 instead of 20)

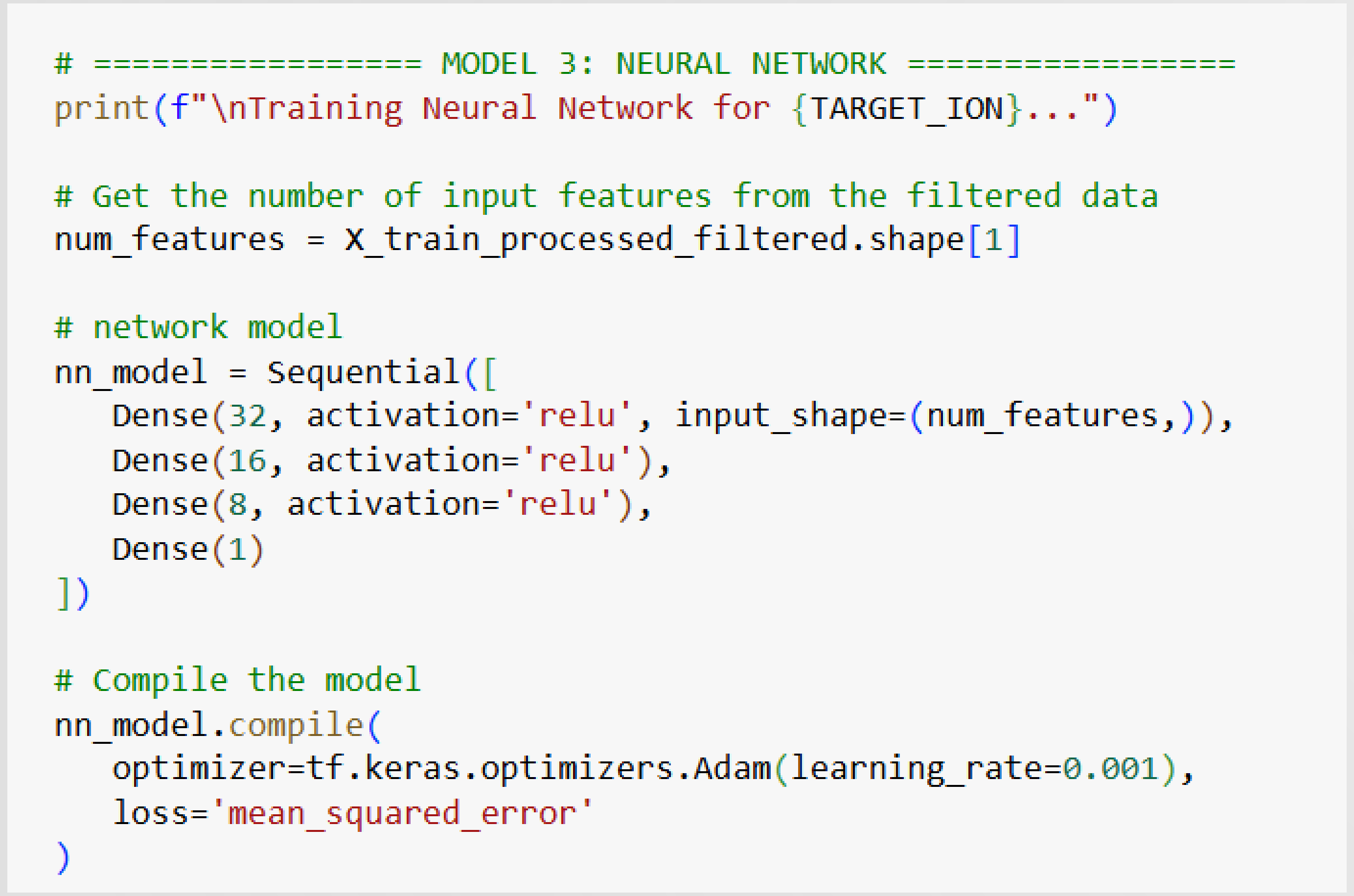


Practice Problem 3: Custom Loss Function for Neural Network

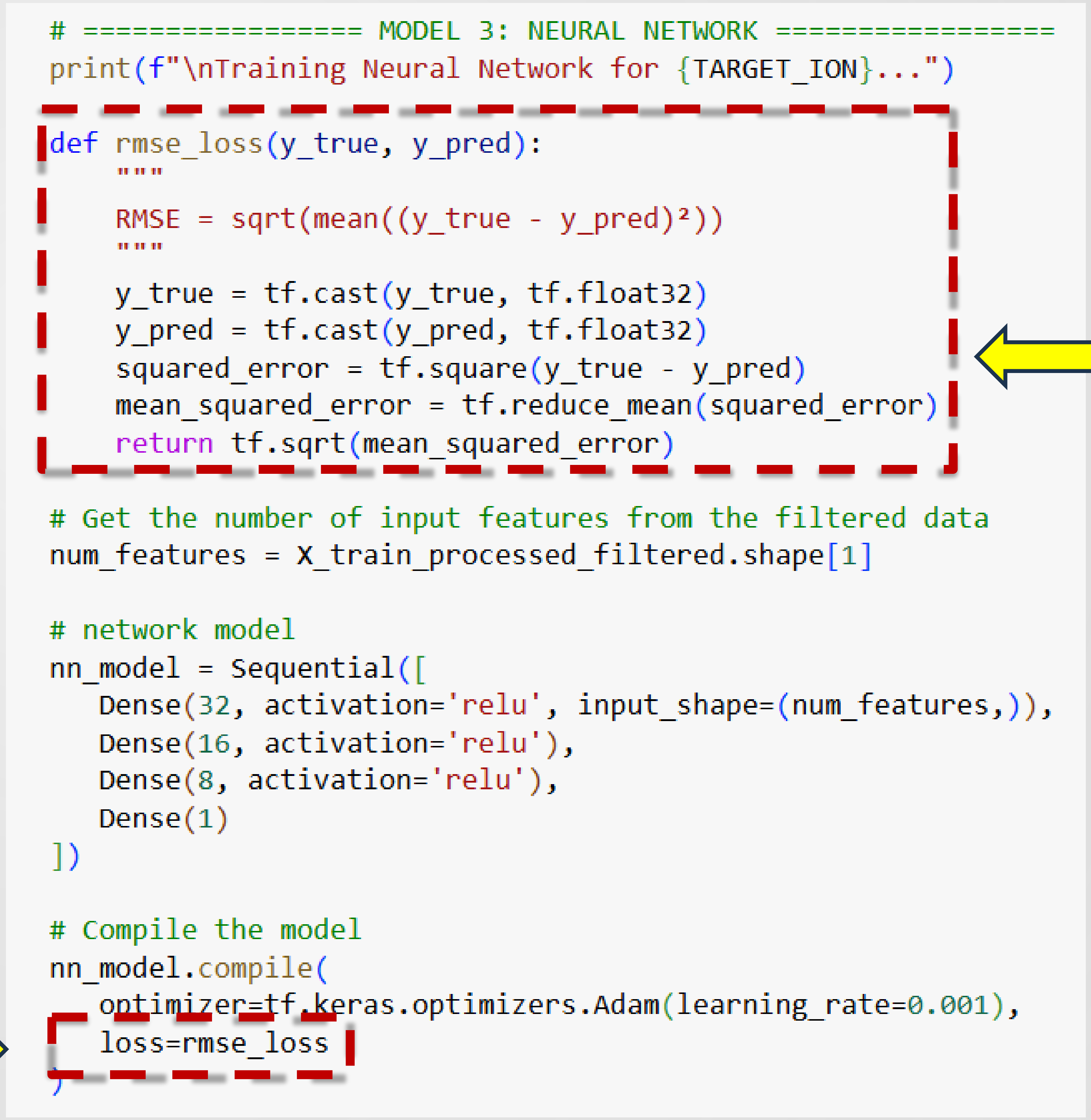
Objective: Implement and evaluate a simpler custom loss function for the Neural Network model.

RMSE instead of MSE. We want to do it with custom loss function.

4. Model Training



```python
# ================= MODEL 3: NEURAL NETWORK =================
print(f"\nTraining Neural Network for {TARGET_ION}...")

# Get the number of input features from the filtered data
num_features = X_train_processed_filtered.shape[1]

# network model
nn_model = Sequential([
   Dense(32, activation='relu', input_shape=(num_features,)),
   Dense(16, activation='relu'),
   Dense(8, activation='relu'),
   Dense(1)
])

# Compile the model
nn_model.compile(
   optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
   loss='mean_squared_error'
)
```

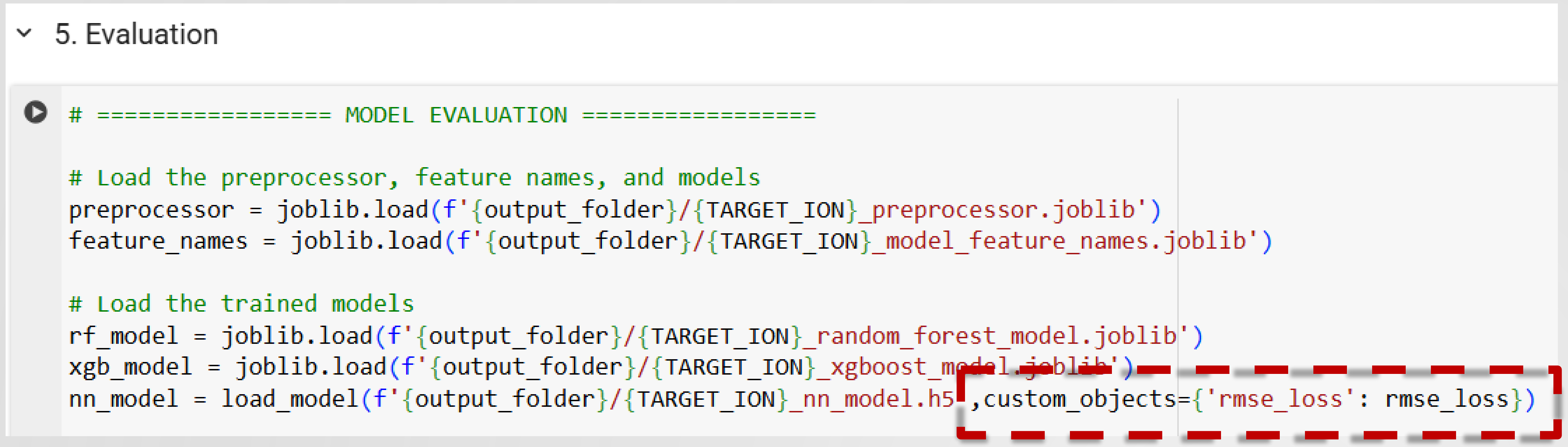


```python
# ================= MODEL 3: NEURAL NETWORK =================
print(f"\nTraining Neural Network for {TARGET_ION}...")

def rmse_loss(y_true, y_pred):
    """
    RMSE = sqrt(mean((y_true - y_pred)²))
    """
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)
    squared_error = tf.square(y_true - y_pred)
    mean_squared_error = tf.reduce_mean(squared_error)
    return tf.sqrt(mean_squared_error)

# Get the number of input features from the filtered data
num_features = X_train_processed_filtered.shape[1]

# network model
nn_model = Sequential([
   Dense(32, activation='relu', input_shape=(num_features,)),
   Dense(16, activation='relu'),
   Dense(8, activation='relu'),
   Dense(1)
])

# Compile the model
nn_model.compile(
   optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
   loss=rmse_loss
)
```

Step 1

Step 2

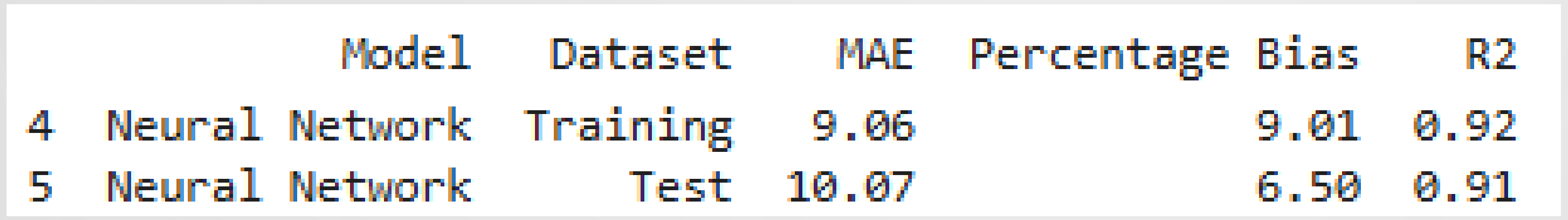
CALIFORNIA DEPARTMENT OF WATER RESOURCES



5. Evaluation

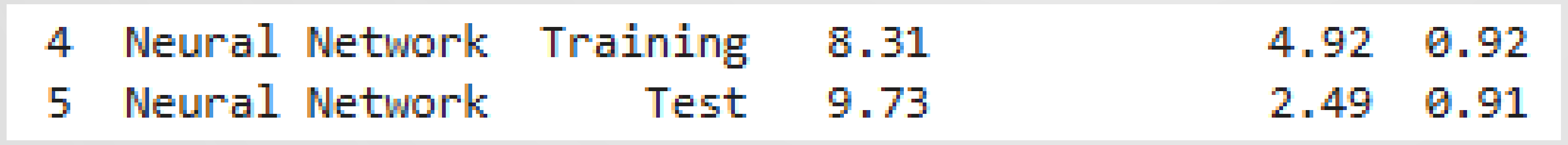
```
# ================= MODEL EVALUATION =================

# Load the preprocessor, feature names, and models
preprocessor = joblib.load(f'{output_folder}/{TARGET_ION}_preprocessor.joblib')
feature_names = joblib.load(f'{output_folder}/{TARGET_ION}_model_feature_names.joblib')

# Load the trained models
rf_model = joblib.load(f'{output_folder}/{TARGET_ION}_random_forest_model.joblib')
xgb_model = joblib.load(f'{output_folder}/{TARGET_ION}_xgboost_model.joblib')
nn_model = load_model(f'{output_folder}/{TARGET_ION}_nn_model.h5',custom_objects={'rmse_loss': rmse_loss})
```

Step 3



```
            Model   Dataset    MAE  Percentage Bias    R2
4  Neural Network  Training   9.06             9.01  0.92
5  Neural Network      Test  10.07             6.50  0.91
```

Loss=MSE



```
4  Neural Network  Training   8.31             4.92  0.92
5  Neural Network      Test   9.73             2.49  0.91
```

Loss=RMSE

CALIFORNIA DEPARTMENT OF
WATER RESOURCES

```python
def rmse_loss(y_true, y_pred):
    """
    RMSE = sqrt(mean((y_true - y_pred)²))
    """
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)
    squared_error = tf.square(y_true - y_pred)
    mean_squared_error = tf.reduce_mean(squared_error)
    return tf.sqrt(mean_squared_error)
```

CALIFORNIA DEPARTMENT OF
WATER RESOURCES

Questions?

CALIFORNIA DEPARTMENT OF WATER RESOURCES

Contact: Kevin He Kevin.He@water.ca.gov