Summary of the Project Proposal

The project utilizes machine learning (ML) and deep learning (DL) models to predict **retention times (RTs)** for compounds in Hydrophilic Interaction Liquid Chromatography (HILIC). The analysis integrates multiple datasets:

1. **hilic_true_negatives.csv** - Contains true negative samples to enhance model robustness.
2. **hilic_data_high_quality.csv** - Provides high-quality annotated retention times for training and testing.
3. **hilic_meta_all.csv** - Offers metadata, possibly descriptors or additional contextual information for compounds.
4. **hilic_data_to_check.csv** - Data requiring validation or cross-referencing.
5. **identifier_utils.py** - A Python script, likely for pre-processing (e.g., descriptor extraction or InChIKey alignment).

This analysis involves:

1. **Data Integration and Cleaning**: Merging datasets, aligning features, and ensuring compatibility.
2. **Feature Engineering**: Using SMILES and metadata to derive molecular descriptors.
3. **Model Implementation**: Applying baseline and advanced models like Random Forest, XGBoost, LASSO, and Keras neural networks.
4. **Error Analysis**: Computing metrics like **MAE** and **RMSE**.
5. **Comparison with Literature**: Validating 3Rtip implementation against established benchmarks.

Steps for Completing the Part 3 of the Project: 3Rtip and Comparison of Literature Methods

1. **Setup the R Environment**
   - Install required packages:
     tidyverse, data.table, caret, randomForest, xgboost, keras, and igraph (for GNN).
   - Load your data (hilic_true_negatives.csv and hilic_data_high_quality.csv) into R, ensuring proper data formatting.

2. **Feature Engineering**

- Use RDKit or similar tools to extract molecular descriptors from the SMILES column. Export descriptors as a .csv or .txt file for input into R.
- Merge the descriptor dataset with annotated RTs for model training.

3. **Implement 3Rtip**
   - Refer to Bonini et al. (2020) for methodology.
   - Set up an ensemble ML pipeline incorporating **Random Forest**, **Bayesian Neural Networks**, **XGBoost**, **LightGBM**, and **Keras**.
   - Tune hyperparameters for each model using **cross-validation** to achieve optimal performance.

4. **Apply Literature Models**
   - Implement LASSO and Ridge regression as baselines.
   - Recreate ensemble methods (Gradient Boosting, Adaptive Boosting) based on the described studies.
   - Integrate **GNN-TL** if feasible, utilizing transfer learning with pre-trained molecular graph models.

5. **Error Distribution Analysis**
   - Compute and visualize MAE, **root mean square error (RMSE)**, and **error distributions** for all models.
   - Compare error metrics across models to assess predictive accuracy.

6. **Checklist for Results**
   - Ensure **model reproducibility** by saving seeds and documenting steps.
   - Validate key assumptions: Are features significant predictors of RTs? Are residuals normally distributed?
   - Present validation results, including comparative MAEs.

7. **Documentation and Reporting**
   - Include a detailed write-up of methods and results.
   - Discuss findings in the context of literature (e.g., does your 3Rtip implementation outperform published models?).

Step 1: Setup and Load Files

```
install.packages(c("tidyverse", "data.table", "caret", "randomForest", "xgboost", "glmnet",
"keras", "e1071", "ggplot2"))

library(tidyverse)
library(data.table)
library(caret)
library(randomForest)
library(xgboost)
library(glmnet)
library(keras)
library(e1071)
library(ggplot2)

hilic_neg <- read.csv("hilic_true_negatives.csv")
hilic_high <- read.csv("hilic_data_high_quality.csv")
hilic_meta <- read.csv("hilic_meta_all.csv")
hilic_to_check <- read.csv("hilic_data_to_check.csv")

str(hilic_neg)
str(hilic_high)
str(hilic_meta)
str(hilic_to_check)
```

Step 2: Merge and Clean Data

```
data <- hilic_high %>%
  left_join(hilic_meta, by = "InChIKey") %>%
  left_join(hilic_neg, by = "InChIKey") %>%
  filter(!is.na(RetentionTime))  # Remove rows without retention times
```

```
head(data)
```

Step 3: Feature Engineering

```
descriptors <- read.csv("descriptors.csv")

data <- left_join(data, descriptors, by = "InChIKey")

target <- "RetentionTime"
predictors <- setdiff(names(data), c("RetentionTime", "SMILES", "InChIKey"))
```

Step 4: Split Data

```
set.seed(123)
trainIndex <- createDataPartition(data$RetentionTime, p = 0.8, list = FALSE)
train_data <- data[trainIndex, ]
test_data <- data[-trainIndex, ]
```

Step 5: Implement Models

```
rf_model <- randomForest(RetentionTime ~ ., data = train_data[, c(predictors, target)], ntree = 500)
rf_predictions <- predict(rf_model, test_data[, predictors])

xgb_train <- xgb.DMatrix(data = as.matrix(train_data[, predictors]), label = train_data$RetentionTime)
xgb_test <- xgb.DMatrix(data = as.matrix(test_data[, predictors]))

params <- list(booster = "gbtree", eta = 0.1, max_depth = 6, objective = "reg:squarederror")
xgb_model <- xgb.train(params = params, data = xgb_train, nrounds = 100)
xgb_predictions <- predict(xgb_model, xgb_test)
```

```
lasso_model <- cv.glmnet(as.matrix(train_data[, predictors]), train_data$RetentionTime,
alpha = 1)
lasso_predictions <- predict(lasso_model, as.matrix(test_data[, predictors]), s = "lambda.min")

keras_model <- keras_model_sequential() %>%
  layer_dense(units = 128, activation = "relu", input_shape = ncol(train_data[, predictors]))
%>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 1)

keras_model %>% compile(optimizer = "adam", loss = "mean_squared_error")
history <- keras_model %>% fit(as.matrix(train_data[, predictors]),
train_data$RetentionTime,
                          epochs = 50, batch_size = 32, validation_split = 0.2)

keras_predictions <- keras_model %>% predict(as.matrix(test_data[, predictors]))
```

Step 6: Evaluate Models

```
evaluate <- function(actual, predicted) {
  mae <- mean(abs(actual - predicted))
  rmse <- sqrt(mean((actual - predicted)^2))
  list(MAE = mae, RMSE = rmse)
}

rf_eval <- evaluate(test_data$RetentionTime, rf_predictions)
xgb_eval <- evaluate(test_data$RetentionTime, xgb_predictions)
lasso_eval <- evaluate(test_data$RetentionTime, lasso_predictions)
keras_eval <- evaluate(test_data$RetentionTime, keras_predictions)

results <- data.frame(
  Model = c("Random Forest", "XGBoost", "LASSO", "Keras"),
  MAE = c(rf_eval$MAE, xgb_eval$MAE, lasso_eval$MAE, keras_eval$MAE),
  RMSE = c(rf_eval$RMSE, xgb_eval$RMSE, lasso_eval$RMSE, keras_eval$RMSE)
)

print(results)
```

Step 7: Visualize Results

```
predictions <- data.frame(
  Actual = test_data$RetentionTime,
  RF = rf_predictions,
  XGBoost = xgb_predictions,
  LASSO = lasso_predictions,
  Keras = keras_predictions
)

ggplot(predictions, aes(x = Actual)) +
  geom_point(aes(y = RF, color = "Random Forest")) +
  geom_point(aes(y = XGBoost, color = "XGBoost")) +
  geom_point(aes(y = LASSO, color = "LASSO")) +
  geom_point(aes(y = Keras, color = "Keras")) +
  labs(title = "Predicted vs Actual Retention Times", x = "Actual Retention Time", y =
"Predicted Retention Time") +
  theme_minimal()
```