Vacation Recommendation

Human Learning

2025-12-14

I. Introduction

Choosing an ideal travel destination is a complicated decision making process. Travelers must balance personal preferences, such as climate, travel style, and budget, with contextual factors like trip duration, companionship, and geographic constraints. Therefore, we developed a machine learning driven vacation recommendation engine that predicts destination ratings and produces personalized Top-5 travel suggestions.

Our system integrates a set of user level features (demographics, preferences, behavioral patterns) and location-level characteristics (climate zone, city type, average temperature, and environmental attributes). Past user ratings are parsed and transformed into an expanded training dataset, enabling the model to learn relationships between user profiles and detination desirability.

To model these relationships, we use a diverse ensemble of predictive methods, including: - K-Neareast Neighbors (KNN) - Decision Tree - Random Forest - Support Vector Machine (SVM) - K-means clustering - LightGBM gradient boosting regression - XGBoost Regression

Rather than relying on a single model, we combine all using a stacking meta-learner, trained through cross-validation predictions. This meta-model, implemented as a linear regression, automatically learns the optimal weighting of each base model, yielding a more stable and accurate final rating predictor compared to soft voting or individual models.

The system supports two recommendation scenarios:

1. Destination ranking without user input, based only on aggregated historical ratings.
2. Personalized Top-5 recommendations, where each destination's predicted rating is computed using stacking ensemble. Optional user-specified filters allow further customization.

Overall, this project delivers a practical, data-driven recommendation system that generates personalized travel suggestions using a stacked ensemble of machine learning models. The combination of diverse predictors and careful feature engineering allows the system to produce reliable and adaptable destination rankings for a wide range of travelers.

Dataset

We use two datasets: a user dataset containing individual characteristics and past travel ratings, and a destination dataset describing attributes of each location.

```
users = read.csv('src/Vacation_UsersData.csv')
dests = read.csv('src/Vacation_DestinationData.csv')
```

Key variables include:

User Dataset - Age, ParentalEducationYears: demographic information - CEOAnnualSalary, Budget: socioeconomic and financial factors - IdealTripDurationDays: preferred trip length - CurrentLatitude, CurrentLongtitude: geographic context - ClimatePreference, TravelStylePreference, TravelCompanionPreference: categorical travel preferences - FavoriteColor, LocalSportsTeamRecord: additional preference indicators - PastTripsWithRatings: list of visited destinations and corresponding ratings.

Destination Dataset - LocationID: identifier for each destination - ClimateZone: environmental category of the destination - CityType: type of urban environment - AvgTemperature_C: average local temperature - Latitude & Longtitude

II. Methods

We begin by importing all the functions (will be attached in the back) and packages used in this project.

```
source('src/knn.R')
source('src/kmc.R')
library(rpart)
library(randomForest)
library(e1071)
library(xgboost)
library(lightgbm)
```

Note: For the MLR and SVM parts, we use package implementations instead of our own course functions. Because the one-hot encoded design matrix is very sparse and $X^\top X$ is not invertible, our custom MLR function based on $(X^\top X)^{-1}$ cannot be used. Therefore, for both MLR and SVM we rely on the corresponding package implementations, which are numerically stable under high dimensional and sigular design matrices.

User preference variable are converted to factors so they can be properly encoded in the design matrix during model fitting.

```
users$ClimatePreference = factor(users$ClimatePreference)
users$TravelStylePreference = factor(users$TravelStylePreference)
users$TravelCompanionPreference = factor(users$TravelCompanionPreference)
users$FavoriteColor = factor(users$FavoriteColor)
users$LocalSportsTeamRecord = factor(users$LocalSportsTeamRecord)
```

1. Data Preprocession

We split the user dataset into 65% training and 35% testing with a fixed random seed for reproducibility. These subsets are used throughout model training and evaluation.

```
set.seed(1234)
n = nrow(users)
ptrain = 0.65
ntrain = floor(n*ptrain)
idx = seq(1, n, 1)
itrain = sample(idx, ntrain, replace=F)
itest = setdiff(idx, itrain)
Xtrain = users[itrain,]
Xtest = users[itest,]
```

Each user's past travel history is stored as a string containing destination IDs and corresponding ratings. We parse this field into a structured format containing one row per (user, destination) pair. We then expand the dataset so that each user appears once per visited destination and merge these records with destination attributes. We remove identifier and non-predictive variable, then construct a fully encoded design matrix using model.matrix. The final training features (Xtrain) and response vector (Ytrain) are extracted for model fitting. We remove identifier and non-predictive variable, then construct a fully encoded design matrix using model.matrix. The final training features (Xtrain) and response vector (Ytrain) are extracted for model fitting.

```
preprocess = function(Xtrain, dests) {
  parse_trips = function(trips) {
```

```
    parts = unlist(strsplit(trips, ", "))

    loc = c()
    rat = c()
    for (p in parts) {
      reg = regexec("Destination_([0-9]+)\\s*-\\s*([0-9]+)", p)
      output = unlist(regmatches(p, reg))
      loc = c(loc, as.integer(output[2]))
      rat = c(rat, as.integer(output[3]))
    }
    return(data.frame(LocationID = loc, Rating = rat))
  }

  Xtrain_list = lapply(1:nrow(Xtrain), function(i) {
    trips = parse_trips(Xtrain$PastTripsWithRatings[i])
    base = Xtrain[i,]
    base = base[rep(1, nrow(trips)), , drop=F]
    return(cbind(base, trips))
  })

  Xtrain_all = do.call(rbind, Xtrain_list)
  Xtrain_merge = merge(Xtrain_all, dests, by = "LocationID")

  drop_cols = c("UserID", "PastTripsWithRatings", "Rating", "CityName", "LocationID")
  feature_cols = setdiff(names(Xtrain_merge), drop_cols)

  Xtrain = model.matrix(~ . -1, data = Xtrain_merge[, feature_cols, drop=F])
  Ytrain = Xtrain_merge$Rating
  return(list(Xtrain = Xtrain, Ytrain = Ytrain,
              Xtrain_all = Xtrain_all, feature_cols = feature_cols))
}

output = preprocess(Xtrain, dests)
feature_cols = output$feature_cols
Xtrain_all = output$Xtrain_all
Xtrain = output$Xtrain
Ytrain = output$Ytrain
```

Because the set of one-hot-encoded columns must remain consistent across training and testing, we define a helper function to align any new matrix to the training column stucture.

```
train_cols = colnames(Xtrain)
align_matrix = function(X_new, train_cols) {
  missing_cols = setdiff(train_cols, colnames(X_new))
  if (length(missing_cols) > 0) {
    add = matrix(0, nrow = nrow(X_new), ncol = length(missing_cols))
    colnames(add) = missing_cols
    X_new = cbind(X_new, add)
  }
  return(X_new[, train_cols, drop=F])
}
```

2. Train the 7 base models

To capture different aspects of the relationship between user-destination features and ratings, we fit a diverse set of regression models on the same design matrix Xtrain and response vector Ytrain. The function below trains all seven base models and returns them in a single list object:

```
## All hyperparameters used below (e.g., k_knn = 17, k_kmc = 3, ntree_rf = 1000)
## are the final values selected from our hyperparameter tuning procedures shown
## in Section IV (Sidenote).
train_all_rating = function(Xtrain, Ytrain, k_knn = 17, k_kmc = 3,
                            ntree_rf = 1000, seed = 1234) {

  # 1) KNN regression:
  knn_model = list(X = Xtrain, Y = Ytrain, k = k_knn)

  # 2) DT regression
  dt_df = data.frame(Rating = Ytrain, Xtrain)
  dt_model = rpart(Rating ~ ., data = dt_df, method = "anova",
                   control = rpart.control(cp = 0.001, minsplit = 10))

  # 3) RF regression
  rf_model = randomForest(x = Xtrain, y = Ytrain,
                          ntree = ntree_rf,
                          mtry = floor(sqrt(ncol(Xtrain))))

  # 4) SVM regression
  svm_model = e1071::svm(x = Xtrain, y = Ytrain,
                         type  = "eps-regression", kernel = "radial")

  # 5) KMC
  kmc_model_raw = kmc(Xtrain, k = k_kmc, init = "Forgy", iter = "Lloyd", seed = seed)
  mean_rate = tapply(Ytrain, kmc_model_raw$cids, mean)
  kmc_model = list(kmc = kmc_model_raw, mean_rate = mean_rate)

  # 6) LightGBM regression
  lgbm_train = lgb.Dataset(data = Xtrain, label = Ytrain)
  lgbm_params = list(objective        = "regression",
                     metric           = "rmse",
                     learning_rate    = 0.05,
                     num_leaves       = 31,
                     feature_fraction = 0.8,
                     bagging_fraction = 0.8,
                     bagging_freq     = 1)

  lgbm_model = lgb.train(params  = lgbm_params,
                         data    = lgbm_train,
                         nrounds = 400,
                         verbose = -1)

  # 7) XGBoost regression
  dtrain = xgb.DMatrix(data = Xtrain, label = Ytrain)
  params = list(objective        = "reg:squarederror",
                eval_metric      = "rmse",
                max_depth        = 4,
                eta              = 0.05,
```

```
                subsample        = 0.8,
                colsample_bytree = 0.8)
  xgb_model = xgb.train(params = params, data = dtrain, nrounds = 400, verbose = 0)

  return(list("knn"   = knn_model,
              "dt"    = dt_model,
              "rf"    = rf_model,
              "svm"   = svm_model,
              "kmc"   = kmc_model,
              "lgbm"  = lgbm_model,
              "xgb"   = xgb_model))
}

models = train_all_rating(Xtrain, Ytrain)
```

3. Per-model rating predictors

After fitting the seven base models, we define a corresponding prediction function for each one. These functions take a new design matrix Xnew and return the model specific estimated ratings:

1) KNN average neighbor ratings

Computes the average rating among the k nearest neighbors of each new observation in feature space.

```
knn_rating = function(model, Xnew) {
  n_new = nrow(Xnew)
  ratings = numeric(n_new)

  for (i in 1:n_new) {
    KNN = knn(model$X, model$Y, Xnew[i, ], k = model$k, type = 2, ztype = "z-score")
    labs = as.numeric(KNN$k_nearest_labels)
    ratings[i] = mean(labs)
  }
  return(ratings)
}
```

2) Decision Tree regression

Uses the fitted regression tree to assign each observation to a leaf node and return the node's predicted value.

```
dt_rating = function(model, Xnew) {
  as.numeric(predict(model, newdata = as.data.frame(Xnew)))
}
```

3) Random Forest regression

Aggregates predictions across all trees in the ensemble, getting a stable estimate.

```
rf_rating = function(model, Xnew) {
  as.numeric(predict(model, newdata = Xnew))
}
```

4) SVM regression

Uses a SVM model to estimate ratings based on patterns it learned from the training data.

```
svm_rating = function(model, Xnew) {
  as.numeric(predict(model, newdata = Xnew))
}
```

5) KMC cluster mean rating

Assigns each new point to the nearest cluster and uses that cluster's mean rating as the prediction.

```
kmc_rating = function(model, Xnew) {
  km = model$kmc
  mr = model$mean_rate
  n_new = nrow(Xnew)
  ratings = numeric(n_new)

  for (i in 1:n_new) {
    cid = km$nearest_cids(Xnew[i, ])
    v = mr[as.character(cid)]
    ratings[i] = ifelse(is.na(v), mean(mr, na.rm=T), v)
  }
  return(ratings)
}
```

6) LightGBM regression

Applies the trained gradient boosting model to obtain fast, high-accuracy predictions.

```
lgbm_rating = function(model, Xnew) {
  as.numeric(predict(model, Xnew))
}
```

7) XGB regression

Uses the trained XGBoost model to predict ratings based on many small decision trees working together.

```
xgb_rating = function(model, Xnew) {
  as.numeric(predict(model, xgb.DMatrix(Xnew)))
}
```

These functions standardize the output across models so that each returns a numeric predicted rating for any new input matrix.

4. Meta-Learner for Stacking by using MLR

In our stacking framework, the seven base models are trained in parallel and each produces its own predicted rating. Instead of selecting a single model or averaging their outputs, we train a second-stage model using a simple linear regression (MLR), using these predictions as inputs.

This meta-learner effectively learns how much weight to assign to each base model. The coefficients of the MLR represent these learned weights, allowing the ensemble to automatically emphasize models that perform better and down weight those that are less accurate.

In short, the stacking layer refines the final prediction by optimally combining all seven model outputs, leading to improved accuracy and robustness.

```
base_models = c("knn", "dt", "rf", "svm", "kmc", "lgbm", "xgb")
M = length(base_models)
```

```r
build_meta = function(Xtrain, Ytrain, Kfolds = 5, seed = 1234) {
  set.seed(seed)
  n = nrow(Xtrain)
  folds = sample(rep(1:Kfolds, length.out = n))

  P = matrix(NA, nrow = n, ncol = M)
  colnames(P) = base_models

  for (k in 1:Kfolds) {
    idx_tr  = which(folds != k)
    idx_val = which(folds == k)

    model_k = train_all_rating(Xtrain[idx_tr, , drop=F], Ytrain[idx_tr], seed = seed)

    P[idx_val, "knn"] = knn_rating(model_k$knn, Xtrain[idx_val, , drop=F])
    P[idx_val, "dt"] = dt_rating(model_k$dt, Xtrain[idx_val, , drop=F])
    P[idx_val, "rf"] = rf_rating(model_k$rf, Xtrain[idx_val, , drop=F])
    P[idx_val, "svm"] = svm_rating(model_k$svm, Xtrain[idx_val, , drop=F])
    P[idx_val, "kmc"] = kmc_rating(model_k$kmc, Xtrain[idx_val, , drop=F])
    P[idx_val, "lgbm"] = lgbm_rating(model_k$lgbm, Xtrain[idx_val, , drop=F])
    P[idx_val, "xgb"] = xgb_rating(model_k$xgb, Xtrain[idx_val, , drop=F])
  }

  return(list(P = P, Y = Ytrain))
}

meta_data = build_meta(Xtrain, Ytrain, Kfolds = 5, seed = 1234)
meta_df = data.frame(Rating = meta_data$Y, meta_data$P)
head(meta_df, 20)
```

```
##    Rating      knn       dt       rf      svm      kmc     lgbm      xgb
## 1      10 7.235294 7.647059 7.667082 8.315394 7.421168 8.413183 8.005407
## 2       6 6.470588 7.750000 7.388985 7.798489 7.419682 7.105734 7.357471
## 3       7 6.294118 7.444444 7.087188 7.103953 7.419682 7.250753 7.267148
## 4      10 7.705882 6.666667 7.672137 8.654648 7.421168 7.100886 7.599708
## 5       7 6.235294 6.500000 7.113321 6.990511 7.395314 7.323868 7.036290
## 6       7 6.411765 8.000000 7.537060 7.072915 7.384202 7.979719 7.736133
## 7      10 6.588235 8.500000 7.028219 7.179973 7.412500 6.940021 7.123943
## 8       8 6.647059 8.076923 7.849550 8.600311 7.443478 7.878992 7.744804
## 9       6 6.117647 7.642857 7.057572 6.590417 7.513193 6.965237 7.063685
## 10      9 6.647059 8.000000 7.649567 6.980838 7.447721 7.608576 7.498855
## 11      8 6.352941 7.833333 7.396225 7.822066 7.395314 7.550858 7.982465
## 12      7 6.235294 4.500000 7.089713 7.263723 7.414634 6.948599 6.827232
## 13      9 7.176471 7.809524 7.765364 8.171397 7.467681 7.996282 8.043987
## 14     10 6.823529 9.285714 8.103673 8.678237 7.467681 8.448211 8.390060
## 15      7 6.470588 6.111111 6.791069 7.561998 7.395314 6.712765 6.967513
## 16      7 6.176471 7.679245 7.637720 7.235802 7.501289 8.247518 7.766054
## 17      6 6.588235 8.500000 8.018707 7.430957 7.443478 8.649328 7.997388
## 18      8 6.529412 6.952381 7.618717 7.499143 7.421168 8.308688 7.682660
## 19      8 5.823529 6.737705 6.922524 7.034095 7.384202 7.135156 7.096141
## 20      7 7.176471 7.000000 7.709931 7.934858 7.513193 8.034692 7.509025
```

```r
stack = lm(Rating ~ . -1, data = meta_df)
stack$coef
```

```
##         knn          dt          rf         svm         kmc        lgbm
##  0.20526174 -0.02888128 -0.20610484  0.33691618  0.46903451 -0.03946671
##         xgb
##  0.28915871
```

5. Models Ensemble: Stacking

For each new input, we first collect predictions from all seven base models. These values are then passed to the meta-model, which combines them using the learned weights to produce the final rating.

```
stack_predict_rating = function(models, stack_model, Xnew) {
  comps = data.frame(knn = knn_rating(models$knn, Xnew),
                     dt  = dt_rating(models$dt, Xnew),
                     rf  = rf_rating(models$rf, Xnew),
                     svm = svm_rating(models$svm, Xnew),
                     kmc = kmc_rating(models$kmc, Xnew),
                     lgbm  = lgbm_rating(models$lgbm, Xnew),
                     xgb = xgb_rating(models$xgb, Xnew))

  return(as.numeric(predict(stack_model, newdata = comps)))
}
```

III. Recommendation Scenerios

Here we show how the recommendation system works in two different situations.

Case I: No user information

When no user preferences are available, the system relies solely on historical data. We compute the average rating received by each destination in the training set and recommend the Top-5 destinations with the highest mean scores.

```
top5_no_info = function(Xtrain_all, dests, topk = 5) {
  avg = aggregate(Rating ~ LocationID, data = Xtrain_all, FUN = mean)
  avg = avg[order(-avg$Rating),]
  top = head(avg, topk)
  result = merge(top, dests, by = "LocationID")
  return(result[order(-result$Rating),])
}

top5_no_info(Xtrain_all, dests, topk = 5)
```

```
##   LocationID   Rating        CityName Latitude Longitude AvgTemperature_C
## 1         23 7.941176 Destination_023 -55.7815   77.1782             11.4
## 5         64 7.897436 Destination_064  19.5634   94.8588             25.2
## 3         37 7.857143 Destination_037 -22.3589 -125.1804             29.8
## 4         39 7.809524 Destination_039 -40.6049   73.2384             16.7
## 2         33 7.805556 Destination_033  43.6099 -128.2623             12.5
##   AvgCostOfLivingIndex ClimateZone CityType
## 1                 81.1 Continental  Natural
## 5                133.1         Dry    Urban
## 3                136.9    Tropical   Resort
## 4                127.2   Temperate    Beach
## 2                153.9   Temperate    Urban
```

Case II: User information available

When user data are provided, the system generates personalized destination recommendations. For each new user, we first cross join the user with all destinations to create a full user–destination grid. Every row in this grid represents "this user visiting this destination," including both user features and destination features. The grid is then converted into the exact model, matrix structure used during training.

```
### Case II: Have user info
prepare_Xnew = function(Xnew, dests, models, stack) {
  Xnew$id = seq_len(nrow(Xnew))
  grid0 = merge(Xnew, dests, by = NULL)

  prepare_grid = function(grid) {
    Xg = model.matrix(~ . -1, data = grid[, feature_cols, drop=F])
    Xg = align_matrix(Xg, train_cols)
    list(grid = grid, Xg = Xg)
  }

  prep = prepare_grid(grid0)
  grid = prep$grid
  Xg = prep$Xg

  grid$PredRating = stack_predict_rating(models, stack, Xg)
  out = split(grid, grid$id)
  return(out)
}
```

Next, each base model produces a predicted rating for every destination. The stacking meta-learner then combines these predictions into a single final rating. Users may also apply optional filters, such as climate preference, city type, or temperature range to restrict the candidate destinations before ranking.

```
recommend_topk_rating = function(Xnew, dests, topk = 5, models, stack,
                                 climate_choice = NULL, city_choice = NULL, temp_choice = NULL) {
  out = prepare_Xnew(Xnew, dests, models, stack)
  result = lapply(out, function(df) {
    if (!is.null(climate_choice)) {
      df = df[which(df$ClimateZone == climate_choice),]
    }
    if (!is.null(city_choice)) {
      df = df[which(df$CityType == city_choice),]
    }
    if (!is.null(temp_choice)) {
      reg = gregexpr("-?\\d+\\.?\\d*", temp_choice)
      nums = as.numeric(unlist(regmatches(temp_choice, reg)))
      df = df[which(df$AvgTemperature_C > nums[1] &
                    df$AvgTemperature_C <= nums[2]),]
    }
    df = df[order(df$PredRating, decreasing=T), , drop=F]
    return(head(data.frame(LocationID = df$LocationID, PredRating = df$PredRating), topk))
  })
  return(result)
}
```

Finally, the top-ranked destinations are merged back with their metadata (city type, climate zone, etc.) and packaged together with the corresponding user. This provides an interpretable output structure: for each user, the system reports the 5 destinations with the highest predicted ratings according to the ensemble

model.

```
top5_have_info = function(Xnew, dests, topk = 5, models, stack,
                          climate_choice = NULL, city_choice = NULL, temp_choice = NULL) {
  model = recommend_topk_rating(Xnew, dests, topk, models, stack,
                                climate_choice, city_choice, temp_choice)

  loc_list = vector("list", length = nrow(Xnew))
  for (i in seq_len(nrow(Xnew))) {
    top5 = model[[i]]
    merged  = merge(top5, dests, by = "LocationID")
    loc_list[[i]]  = merged[order(merged$PredRating, decreasing=T), ]
  }

  user_list <- split(Xnew, seq_len(nrow(Xnew)))
  result <- Map(function(user, dests) {
      list(user = user, dests = dests)
    }, user_list, loc_list)

  return(result)
}
```

Examples

Example 1 (recommend based on test set) We first demonstrate how the system performs personalized recommendations for several users drawn directly from the test set. For each user profile, the model evaluates all destinations, predicts a rating through the stacked ensemble, and returns the Top-5 destinations with the highest predicted scores.

```
new_users = Xtest[1:5,]
top5_have_info(Xnew = new_users, dests, topk = 5, models, stack = stack)
```

```
## $`1`
## $`1`$user
##   UserID Age ParentalEducationYears CEOAnnualSalary  Budget
## 1      1  25                   14.7        39719.36 1411.74
##   IdealTripDurationDays CurrentLatitude CurrentLongitude ClimatePreference
## 1                     5         40.8327         -74.8982              Warm
##   TravelStylePreference TravelCompanionPreference FavoriteColor
## 1             Adventure                   Couples          Blue
##   LocalSportsTeamRecord
## 1               Neutral
##                                                    PastTripsWithRatings
## 1 Destination_061-7, Destination_012-6, Destination_031-7, Destination_015-4
##
## $`1`$dests
##   LocationID PredRating        CityName Latitude Longitude AvgTemperature_C
## 5         64   7.950657 Destination_064  19.5634   94.8588             25.2
## 2         31   7.931582 Destination_031   1.3201   92.4842             30.2
## 3         40   7.924381 Destination_040  16.7289 -127.9237             25.1
## 4         56   7.922985 Destination_056 -31.1223 -151.8501             25.9
## 1          1   7.919995 Destination_001 -15.3031   55.0516             25.6
##   AvgCostOfLivingIndex ClimateZone CityType
## 5                133.1         Dry    Urban
## 2                138.0         Dry    Urban
```

```
## 3                156.6         Dry   Resort
## 4                107.3         Dry Cultural
## 1                132.3         Dry Cultural
##
##
## $`2`
## $`2`$user
##   UserID Age ParentalEducationYears CEOAnnualSalary  Budget
## 2      2  34                  13.3        70391.75 1217.15
##   IdealTripDurationDays CurrentLatitude CurrentLongitude ClimatePreference
## 2                     7          25.433         -80.0823         Temperate
##   TravelStylePreference TravelCompanionPreference FavoriteColor
## 2                 Party                      Solo        Orange
##   LocalSportsTeamRecord                PastTripsWithRatings
## 2               Winning Destination_059-6, Destination_064-5
##
## $`2`$dests
##   LocationID PredRating        CityName Latitude Longitude AvgTemperature_C
## 3         35   7.925950 Destination_035 -40.6784   94.3176             14.4
## 1         23   7.784360 Destination_023 -55.7815   77.1782             11.4
## 5         57   7.751646 Destination_057 -30.0243  -61.8830             18.6
## 2         30   7.722603 Destination_030  17.4026  133.7719             25.6
## 4         42   7.595805 Destination_042 -45.6657  144.4349              9.2
##   AvgCostOfLivingIndex ClimateZone CityType
## 3                 84.7   Temperate  Natural
## 1                 81.1 Continental  Natural
## 5                 93.3   Temperate  Natural
## 2                 94.7         Dry  Natural
## 4                103.2 Continental  Natural
##
##
## $`3`
## $`3`$user
##   UserID Age ParentalEducationYears CEOAnnualSalary  Budget
## 3      3  18                  14.9        60139.63 1225.82
##   IdealTripDurationDays CurrentLatitude CurrentLongitude ClimatePreference
## 3                    13         33.5377         -117.786             Mixed
##   TravelStylePreference TravelCompanionPreference FavoriteColor
## 3                 Party                    Couples        Orange
##   LocalSportsTeamRecord                                    PastTripsWithRatings
## 3               Losing Destination_069-5, Destination_068-5, Destination_034-8
##
## $`3`$dests
##   LocationID PredRating        CityName Latitude Longitude AvgTemperature_C
## 2         23   7.486717 Destination_023 -55.7815   77.1782             11.4
## 1          7   7.395800 Destination_007  -9.6731 -129.8780             19.2
## 5         61   7.390258 Destination_061  -8.8470 -135.4316             26.0
## 4         55   7.361203 Destination_055 -24.2471 -162.8097             19.5
## 3         33   7.339900 Destination_033  43.6099 -128.2623             12.5
##   AvgCostOfLivingIndex ClimateZone CityType
## 2                 81.1 Continental  Natural
## 1                110.1         Dry    Urban
## 5                116.2         Dry    Urban
## 4                156.1         Dry    Urban
```

```
## 3               153.9   Temperate    Urban
##
##
## $`4`
## $`4`$user
##   UserID Age ParentalEducationYears CEOAnnualSalary  Budget
## 5      5  60                  12.1        19466.88 1106.52
##   IdealTripDurationDays CurrentLatitude CurrentLongitude ClimatePreference
## 5                     5          25.467         -80.1078         Temperate
##   TravelStylePreference TravelCompanionPreference FavoriteColor
## 5            Relaxation                    Family           Red
##   LocalSportsTeamRecord PastTripsWithRatings
## 5               Neutral    Destination_006-7
##
## $`4`$dests
##   LocationID PredRating        CityName Latitude Longitude AvgTemperature_C
## 1         17   8.258628 Destination_017  36.0001  155.9101             17.1
## 4         47   8.115377 Destination_047 -39.8819   36.0454             17.3
## 5         67   8.114603 Destination_067  33.5133  -90.9262             19.3
## 3         44   8.068978 Destination_044  26.0875  158.5189             18.9
## 2         41   8.057370 Destination_041 -36.2533   -3.8814             16.7
##   AvgCostOfLivingIndex ClimateZone  CityType
## 1                119.0   Temperate  Cultural
## 4                111.1   Temperate Adventure
## 5                146.6   Temperate     Beach
## 3                120.5   Temperate    Resort
## 2                125.4   Temperate  Cultural
##
##
## $`5`
## $`5`$user
##   UserID Age ParentalEducationYears CEOAnnualSalary  Budget
## 7      7  18                  14.8        43032.41 1000.76
##   IdealTripDurationDays CurrentLatitude CurrentLongitude ClimatePreference
## 7                     9         41.7619         -87.9972             Mixed
##   TravelStylePreference TravelCompanionPreference FavoriteColor
## 7                 Party                   Friends        Purple
##   LocalSportsTeamRecord                PastTripsWithRatings
## 7               Winning Destination_010-7, Destination_049-7
##
## $`5`$dests
##   LocationID PredRating        CityName Latitude Longitude AvgTemperature_C
## 3         23   7.641085 Destination_023 -55.7815   77.1782             11.4
## 5         30   7.631255 Destination_030  17.4026  133.7719             25.6
## 1          9   7.593175 Destination_009 -10.7700   94.7753             29.6
## 4         29   7.520109 Destination_029  -2.9855  133.1703             24.4
## 2         13   7.519712 Destination_013 -71.3581   28.5933             -4.9
##   AvgCostOfLivingIndex ClimateZone CityType
## 3                 81.1 Continental  Natural
## 5                 94.7         Dry  Natural
## 1                117.8    Tropical    Urban
## 4                124.4    Tropical    Urban
## 2                115.7       Polar Cultural
```

Example 2 (interaction to get user info) Note: The follow-up filtering step uses menu(), which is only available in interactive R sessions. When knitting this document, default choices are used to avoid interrupting the render process.

The system can also generate recommendations for a user who provides their information interactively. After collecting numeric and categorical inputs, the profile is encoded to match the training feature space and passed into the stacked model to produce personalized Top-5 recommendations.

```
top5_ask_info = function(dests, topk = 5, models, stack) {

  # ---------- Numeric inputs ----------
  if (interactive()) {
    age     = readline("Please enter your age: ")
    parent  = readline("Please enter your parental education years: ")
    ceo     = readline("Please enter your CEO annual salary: ")
    budget  = readline("Please enter your budget: ")
    duration= readline("Please enter your ideal trip duration days: ")
    lat     = readline("Please enter your current latitude: ")
    long    = readline("Please enter your current longitude: ")
  } else {
    # defaults when knitting
    age = parent = ceo = budget = duration = lat = long = 0
  }

  # ---------- Climate ----------
  if (interactive()) {
    climate_idx = menu(c("Cold","Mixed","Temperate","Warm"),
                       title = "Please choose climate:")
  } else {
    climate_idx = 3   # default = Temperate
  }
  climate = c("Cold","Mixed","Temperate","Warm")[climate_idx]

  # ---------- Style ----------
  if (interactive()) {
    style_idx = menu(c("Adventure","Cultural","Luxury","Party","Relaxation"),
                     title = "Please choose travel style:")
  } else {
    style_idx = 2
  }
  style = c("Adventure","Cultural","Luxury","Party","Relaxation")[style_idx]

  # ---------- Companion ----------
  if (interactive()) {
    compan_idx = menu(c("Couples","Family","Friends","Solo","Work"),
                      title = "Please choose companion type:")
  } else {
    compan_idx = 1
  }
  compan = c("Couples","Family","Friends","Solo","Work")[compan_idx]

  # ---------- Color ----------
  if (interactive()) {
    color_idx = menu(c("Black","Blue","Green","Orange","Purple","Red","White","Yellow"),
                     title = "Please choose favorite color:")
```

```
  } else {
    color_idx = 1
  }
  color = c("Black","Blue","Green","Orange","Purple","Red","White","Yellow")[color_idx]

  # ---------- Sports team ----------
  if (interactive()) {
    sport_idx = menu(c("Losing","Neutral","Winning"),
                     title = "Local sports team record:")
  } else {
    sport_idx = 2   # Neutral
  }
  sport = c("Losing","Neutral","Winning")[sport_idx]


  new_users = data.frame(
    Age                      = as.numeric(age),
    ParentalEducationYears   = as.numeric(parent),
    CEOAnnualSalary          = as.numeric(ceo),
    Budget                   = as.numeric(budget),
    IdealTripDurationDays    = as.numeric(duration),
    CurrentLatitude          = as.numeric(lat),
    CurrentLongitude         = as.numeric(long),
    ClimatePreference        = climate,
    TravelStylePreference    = style,
    TravelCcompanionPreference = compan,
    FavoriteColor            = color,
    LocalSportsTeamRecord    = sport
  )

  new_users$ClimatePreference       = factor(new_users$ClimatePreference,
                                             levels = levels(users$ClimatePreference))
  new_users$TravelStylePreference   = factor(new_users$TravelStylePreference,
                                             levels = levels(users$TravelStylePreference))
  new_users$TravelCompanionPreference = factor(new_users$TravelCompanionPreference,
                                               levels = levels(users$TravelCompanionPreference))
  new_users$FavoriteColor           = factor(new_users$FavoriteColor,
                                             levels = levels(users$FavoriteColor))
  new_users$LocalSportsTeamRecord   = factor(new_users$LocalSportsTeamRecord,
                                             levels = levels(users$LocalSportsTeamRecord))

  top5_have_info(new_users, dests, topk, models, stack)
}

top5_ask_info(dests, topk = 5, models = models, stack = stack)
```

```
## $`1`
## $`1`$user
##   Age ParentalEducationYears CEOAnnualSalary Budget IdealTripDurationDays
## 1   0                      0               0      0                     0
##   CurrentLatitude CurrentLongitude ClimatePreference TravelStylePreference
## 1               0                0         Temperate               Cultural
##   TravelCcompanionPreference FavoriteColor LocalSportsTeamRecord
## 1                    Couples         Black               Neutral
```

```
##
## $`1`$dests
##   LocationID PredRating        CityName Latitude Longitude AvgTemperature_C
## 4         23   8.119029 Destination_023 -55.7815   77.1782             11.4
## 5         38   8.027057 Destination_038 -42.3197   66.0492             17.8
## 3         16   7.989086 Destination_016 -41.2470   27.5608             14.3
## 1         10   7.939860 Destination_010 -35.1140 -137.2271             17.2
## 2         14   7.935411 Destination_014 -58.7333 -155.8512             10.9
##   AvgCostOfLivingIndex ClimateZone CityType
## 4                 81.1 Continental  Natural
## 5                126.4   Temperate Cultural
## 3                139.9   Temperate Cultural
## 1                 85.0   Temperate Cultural
## 2                154.0 Continental   Resort
```

Case III: Have user info (can ask follow up questions about destination info)

Beyond the initial personalized recommendations, the system allows optional follow-up filters, such as climate, city type, or temperature range to refine the results. If the user selects "I'm fine with all," no additional filtering is applied.

```
top5_followup_info = function(new_users, dests, topk = 5, models, stack) {
if (interactive()) {
  climate_options = menu(c("Dry","Temperate","Continental","Tropical","Polar",
                           "I'm fine with all"),
                         title = "Please choose one:")
} else {
  climate_options = 6  # skip interaction when knitting
}

climate_choice = if (climate_options == 6) NULL
                 else c("Dry","Temperate","Continental","Tropical",
                        "Polar")[climate_options]


if (interactive()) {
  city_options = menu(c("Cultural","Resort","Beach","Urban","Natural",
                        "Adventure","I'm fine with all"),
                      title = "Please choose one:")
} else {
  city_options = 7
}

city_choice = if (city_options == 7) NULL
              else c("Cultural","Resort","Beach","Urban","Natural",
                     "Adventure")[city_options]


if (interactive()) {
  temp_options = menu(c("-10 < temp <= 0","0 < temp <= 10","10 < temp <= 20",
                        "20 < temp <= 33","I'm fine with all"),
                      title = "Please choose one:")
} else {
  temp_options = 5
```

```
}

temp_choice = if (temp_options == 5) NULL
              else c("-10 < temp <= 0","0 < temp <= 10","10 < temp <= 20",
                     "20 < temp <= 33")[temp_options]

  return(top5_have_info(new_users, dests, topk, models, stack,
                        climate_choice, city_choice, temp_choice))
}

new_users = Xtest[1,]
top5_followup_info(new_users, dests, topk = 5, models, stack)
```

```
## $`1`
## $`1`$user
##   UserID Age ParentalEducationYears CEOAnnualSalary  Budget
## 1      1  25                   14.7        39719.36 1411.74
##   IdealTripDurationDays CurrentLatitude CurrentLongitude ClimatePreference
## 1                     5         40.8327         -74.8982              Warm
##   TravelStylePreference TravelCcompanionPreference FavoriteColor
## 1             Adventure                    Couples          Blue
##   LocalSportsTeamRecord
## 1               Neutral
##                                                    PastTripsWithRatings
## 1 Destination_061-7, Destination_012-6, Destination_031-7, Destination_015-4
##
## $`1`$dests
##   LocationID PredRating        CityName Latitude Longitude AvgTemperature_C
## 5         64   7.950657 Destination_064  19.5634   94.8588             25.2
## 2         31   7.931582 Destination_031   1.3201   92.4842             30.2
## 3         40   7.924381 Destination_040  16.7289 -127.9237             25.1
## 4         56   7.922985 Destination_056 -31.1223 -151.8501             25.9
## 1          1   7.919995 Destination_001 -15.3031   55.0516             25.6
##   AvgCostOfLivingIndex ClimateZone CityType
## 5                133.1         Dry    Urban
## 2                138.0         Dry    Urban
## 3                156.6         Dry   Resort
## 4                107.3         Dry Cultural
## 1                132.3         Dry Cultural
```

Case IV: Partial Predictors (have same predictors as another team)

In this case, we consider a scenario where only a partial set of user information is available and shared with another team. The predictors include basic trip preferences, current location, climate preference, and past trips with ratings.

Using this limited information, we train rating models and generate the top 5 destination recommendations for a new user. This case evaluates whether reasonable recommendations can still be produced with commonly available user predictors.

```
X1 = users$Budget
X2 = users$IdealTripDurationDays
X3 = users$CurrentLatitude
X4 = users$CurrentLongitude
X5 = users$ClimatePreference
```

```r
X6 = users$PastTripsWithRatings
users4 = data.frame("Budget" = X1,
                    "IdealTripDurationDays" = X2,
                    "CurrentLatitude" = X3,
                    "CurrentLongitude" = X4,
                    "ClimatePreference" = X5,
                    "PastTripsWithRatings" = X6)
users4$ClimatePreference = factor(users4$ClimatePreference)

Xtrain4 = users[itrain,]
Xtest4 = users[itest,]
output4 = preprocess(Xtrain4, dests)
feature_cols4 = output4$feature_cols
Xtrain4 = output$Xtrain
Ytrain4 = output$Ytrain
train_cols4 = colnames(Xtrain4)

models4 = train_all_rating(Xtrain4, Ytrain4)
meta_data4 = build_meta(Xtrain4, Ytrain4, Kfolds = 5, seed = 1234)
meta_df4 = data.frame(Rating = meta_data4$Y, meta_data4$P)
stack4 = lm(Rating ~ . -1, data = meta_df4)

new_users = Xtest4[1,]
top5_have_info(new_users, dests, topk = 5, models4, stack)
```

```
## $`1`
## $`1`$user
##   UserID Age ParentalEducationYears CEOAnnualSalary  Budget
## 1      1  25                   14.7        39719.36 1411.74
##   IdealTripDurationDays CurrentLatitude CurrentLongitude ClimatePreference
## 1                     5         40.8327         -74.8982              Warm
##   TravelStylePreference TravelCompanionPreference FavoriteColor
## 1             Adventure                   Couples          Blue
##   LocalSportsTeamRecord
## 1               Neutral
##                                                  PastTripsWithRatings
## 1 Destination_061-7, Destination_012-6, Destination_031-7, Destination_015-4
##
## $`1`$dests
##   LocationID PredRating        CityName Latitude Longitude AvgTemperature_C
## 5         64   7.951541 Destination_064  19.5634   94.8588             25.2
## 2         31   7.942865 Destination_031   1.3201   92.4842             30.2
## 3         56   7.940141 Destination_056 -31.1223 -151.8501             25.9
## 4         61   7.925864 Destination_061  -8.8470 -135.4316             26.0
## 1          1   7.924999 Destination_001 -15.3031   55.0516             25.6
##   AvgCostOfLivingIndex ClimateZone CityType
## 5                133.1         Dry    Urban
## 2                138.0         Dry    Urban
## 3                107.3         Dry Cultural
## 4                116.2         Dry    Urban
## 1                132.3         Dry Cultural
```

IV. Sidenote: Hyperparameter Tuning

This section shows how the final hyperparameters were selected via cross-validation. These tuned values are used in the main model training.

Hyperparameter Tuning for KNN

```
rmse = function(y, yhat) {
  sqrt(mean((y - yhat)^2))
}

tune_knn_k = function(X, y, k_grid = seq(3, 31, 2), Kfolds = 5, seed = 1234) {
  set.seed(seed)
  n = nrow(X)
  folds = sample(rep(1:Kfolds, length.out = n))
  results = data.frame(k = k_grid, RMSE = NA_real_)

  for (i in seq_along(k_grid)) {
    k_val = k_grid[i]
    rmse_vec = numeric(Kfolds)

    for (f in 1:Kfolds) {
      idx_tr  = which(folds != f)
      idx_val = which(folds == f)

      model_fold = list(X = X[idx_tr, , drop=F],
                        Y = y[idx_tr],
                        k = k_val)

      preds = knn_rating(model_fold, X[idx_val, , drop=F])
      rmse_vec[f] = rmse(y[idx_val], preds)
    }

    results$RMSE[i] = mean(rmse_vec)
  }

  best_idx = which.min(results$RMSE)
  return(list(best_k = results$k[best_idx], cv_results = results))
}

# knn_tuning = tune_knn_k(Xtrain, Ytrain)
# knn_tuning$best_k
# knn_tuning$cv_results
```

Hyperparameter Tuning for KMC

```
tune_kmc_k = function(X, y, k_grid = c(3, 5, 7, 10, 15),
                      Kfolds = 5, seed = 1234) {
  set.seed(seed)
  n = nrow(X)
  folds = sample(rep(1:Kfolds, length.out = n))
  results = data.frame(k = k_grid, RMSE = NA_real_)
```

```
  for (i in seq_along(k_grid)) {
    k_val = k_grid[i]
    rmse_vec = numeric(Kfolds)

    for (f in 1:Kfolds) {
      idx_tr  = which(folds != f)
      idx_val = which(folds == f)

      kmc_model_raw = kmc(X[idx_tr, , drop=F], k = k_val,
                          init = "Forgy", iter = "Lloyd", seed = seed)
      mean_rate = tapply(y[idx_tr], kmc_model_raw$cids, mean)
      kmc_model = list(kmc = kmc_model_raw, mean_rate = mean_rate)

      preds = kmc_rating(kmc_model, X[idx_val, , drop=F])
      rmse_vec[f] = rmse(y[idx_val], preds)
    }

    results$RMSE[i] = mean(rmse_vec)
  }

  best_idx = which.min(results$RMSE)
  return(list(best_k = results$k[best_idx], cv_results = results))
}

# kmc_tuning = tune_kmc_k(Xtrain, Ytrain)
# kmc_tuning$best_k
# kmc_tuning$cv_results
```

Hyperparameter Tuning for Random Forest

```
tune_rf_ntree = function(X, y, ntree_grid = c(500, 1000, 1500, 2000),
                         Kfolds = 5, seed = 1234) {
  set.seed(seed)
  n = nrow(X)
  folds = sample(rep(1:Kfolds, length.out = n))
  results = data.frame(ntree = ntree_grid, RMSE = NA_real_)

  for (i in seq_along(ntree_grid)) {
    ntree_val = ntree_grid[i]
    rmse_vec = numeric(Kfolds)

    for (f in 1:Kfolds) {
      idx_tr  = which(folds != f)
      idx_val = which(folds == f)

      rf_model = randomForest(x = X[idx_tr, , drop=F],
                              y = y[idx_tr],
                              ntree = ntree_val,
                              mtry = floor(sqrt(ncol(X))))

      preds = predict(rf_model, newdata = X[idx_val, , drop=F])
      rmse_vec[f] = rmse(y[idx_val], preds)
    }
```

```
    results$RMSE[i] = mean(rmse_vec)
  }

  best_idx = which.min(results$RMSE)
  return(list(best_ntree = results$ntree[best_idx], cv_results = results))
}

# rf_tuning = tune_rf_ntree(Xtrain, Ytrain)
# rf_tuning$best_ntree
# rf_tuning$cv_results
```

V Two Groups Comparison Analysis

No-User-Info Comparison

Our Team — No Info Top 5

Rank	Destination	Notable Attributes
1	Destination_023	Mild temp, natural scenery
2	Destination_064	Warm, urban
3	Destination_037	Tropical resort
4	Destination_039	Coastal temperate
5	Destination_033	Temperate urban

Other Team — No Info Top 5

Rank	Destination	FinalScore Components
1	Destination_030	High intrinsic quality (PCA), safe
2	Destination_023	High intrinsic quality, same pick as our team's #1
3	Destination_018	Strong PCA score
4	Destination_022	Good PCA and cluster quality
5	Destination_064	Appears in both teams' lists

Without user information, the two systems produce different results because they define "population preference" differently. Our team relies on historical average ratings, so our recommendations reflect what the average user genuinely enjoyed, leading to a more diverse set of destinations. The other team focuses on destination attributes like safety and cost, which prioritizes structurally "high-quality" places but may not represent what people actually prefer. In situations where no user information is available, our approach is arguably better because it is grounded directly in real user behavior, making it a more realistic estimate of population-level preferences.

With-User-Info Comparison

Our Team - With User Info

Rank	Destination	PredRating	ClimateZone	CityType
1	064	7.9515	Dry	Urban
2	031	7.9429	Dry	Urban
3	056	7.9404	Dry	Cultural

Rank	Destination	PredRating	ClimateZone	CityType
4	061	7.9259	Dry	Urban
5	001	7.9250	Dry	Cultural

Other Team - With User Info

Rank	Destination	CFScore	PopScore	FinalScore
1	044	1.00	0.7405	0.9222
2	013	0.875	0.6804	0.8166
3	027	0.7486	0.5348	0.6845
4	050	0.7568	0.3576	0.6370
5	069	0.8169	0.1804	0.6260

When both systems use the same user information (UserID = 1), our model personalizes recommendations more effectively. By using the user's warm - climate preference, budget, and past ratings, our model consistently returns warm/dry destinations that match the user's profile. The other team's hybrid CF system relies mainly on similar users' behavior and cluster popularity, which can overlook explicit preferences such as climate. As a result, our system provides recommendations that align more closely with what our testing user clearly likes. Therefore, for this specific user, our personalized results are more targeted and better justified.

VI. Conclusion

This project built a personalized vacation recommendation system using a stacked ensemble of seven machine learning models. By combining diverse predictors and user–destination features, the model produces more stable and accurate rating estimates than any single method. The system supports both general and personalized Top-5 recommendations, with optional filters for user-specific preferences. Overall, it provides a flexible foundation for destination ranking and can be extended with additional data or user behavior signals.

Appendix of Sourced Functions

1. K-Nearest Neighbors (KNN)

```
knn = function(X, Y, myx, k, type=2, ztype="z-score", weight=NULL) {
  # X is a matrix, Y and myx are vectors
  n = nrow(X)
  p = ncol(X)

  # ------------------- Normalization --------------------
  if(ztype == 'z-score') {
    Xs = scale(X)
    myx_s = (myx - colMeans(X)) / apply(X, 2, sd)
  } else if(ztype == 'min-max') {
    xmin = apply(X, 2, min)
    xmax = apply(X, 2, max)
    rng = xmax - xmin
    Xs = sweep(sweep(X, 2, xmin, FUN = "-"), 2, rng, FUN = "/")
    myx_s = (myx - xmin) / rng
  } else if(ztype == 'none') {
    Xs = X
    myx_s = myx
```

```
  } else {
    stop("Unknown target space. Use 'z-score', 'min-max', or 'none'.")
  }

  # find the distances from the myx to all training points
  diff = Xs - matrix(myx_s, nrow = n, ncol = p, byrow=T)

  # ----------------- Apply Weights (optional) -----------------
  if (!is.null(weight)) {
    if (length(weight) == p) {
      # per-feature weights -> scale columns
      diff = sweep(diff, 2, weight, `*`)
    } else if (length(weight) == n) {
      # per-observation weights -> scale rows
      diff = diff * matrix(weight, nrow = n, ncol = p, byrow=F)
    }
  }

  # ----------------- Distance Metrics -----------------
  if(type == 1) {
    dist = rowSums(abs(diff))
  } else if(type == 2) {
    dist = sqrt(rowSums(diff^2))
  } else if(type == 'infty') {
    dist = apply(abs(diff), 1, max)
  } else {
    dist = (rowSums(abs(diff))^type)^(1/type)
  }

  # find the k-nearest neighbors
  Y = factor(Y)
  idx = order(dist)[1:k]
  ylabel = Y[idx]
  ydist = dist[idx]

  # majority vote
  tab = table(ylabel)
  major_vote = names(tab)[tab == max(tab)][1]

  # distribution table
  distri_table = data.frame(class = names(tab),
                            count = as.integer(tab),
                            probs = as.integer(tab)/k)

  result = list('k_nearest_labels'   = ylabel,
                'k_nearest_dist'     = ydist,
                'distribution_table' = distri_table,
                'prob'               = max(as.integer(tab)/k),
                'yhat'               = major_vote)

  return(result)
}
```

```
##############################################################################
### KNN accuracy o for test/train split
myknnacc = function(Xtrain, Ytrain, Xtest, Ytest, k, type=2, ztype="z-score", weight=NULL) {
  ntest = nrow(Xtest)
  myclass = c()
  myprobs = c()
  for (i in 1:ntest) {
    myknn = knn(Xtrain, Ytrain, Xtest[i,], k, type, ztype, weight)
    myclass = c(myclass, myknn$yhat)
    myprobs = c(myprobs, myknn$prob)
  }
  acc = mean(myclass == Ytest)

  result = list('yhat'     = myclass,
                'probs'    = myprobs,
                'accuracy' = acc)
  return(result)
}
```

2. K-Means Clustering (KMC)

```
kmc = function(X, k, init='Forgy', iter='Lloyd', seed) {
  # X is n x p matrix with n observation and p features
  # k is the number of clusters we want
  set.seed(seed)
  n = nrow(X)
  p = ncol(X)

  # check whether k here is a valid parameter
  if(k < 1 || k > n) {
    stop("invalid k value: k should be = or < than the sample size n")
  }

  # ----------------- Initialization of Centroids ------------------------
  if(init == 'Forgy') {
    # choose k random existing points as initial centers
    centroids = X[sample.int(n, k, replace=F), , drop=F]
  } else if (init == 'Bounding-Box') {
    # find the bounding box for the entire dataset and uniformly random select
    # smaple k centroids inside the box
    mins = apply(X, 2, min)
    maxs = apply(X, 2, max)
    centroids = matrix(NA, nrow = k, ncol = p)
    for(j in 1:p) {
      centroids[,j] = runif(k, mins[j], maxs[j])
    }
  } else if (init == 'Random-Partition') {
    # randomly assign labels (1, 2, ..., k) to all points, then take the mean
    # of each label as the initial centroids
    init = sample.int(k, n, replace=T)
    centroids = matrix(NA, nrow = k, ncol = p)
    for (j in 1:k) {
      idx = (init == j)
```

```
    centroids[j,] = colMeans(X[idx, , drop=F])
  }
} else {
  stop("Unknown initialization metric. Use 'Forgy', 'Bounding-Box', or 'Random-Partition'.")
}

initial_centroids = centroids

# ------------------------- Iterations --------------------------------
# initialize the clusters vector, which should contains only labels
if (iter == 'Lloyd') {
  clusters = rep(0, n)
  iter = 0
  repeat {
    iter = iter + 1
    dist = matrix(0, nrow = n, ncol = k)

    # compute the 2-norm distance between each points and the k centers
    for (j in 1:k) {
      diff = X - matrix(centroids[j, ], nrow = n, ncol = p, byrow=T)
      dist[, j] = sqrt(rowSums(diff^2))
    }

    # assign cluster label to every other points
    new_clusters = apply(dist, 1, which.min)
    change = sum(new_clusters != clusters)
    clusters = new_clusters

    # move centers to the mean of assigned points
    for (j in 1:k) {
      idx = (clusters == j)
      if (any(idx)) {
        centroids[j, ] = colMeans(X[idx, , drop=F])
      }
    }

    # convergence check: labels unchanged
    if (change == 0) break
  }
} else if (iter == 'Mac-Queen') {
  clusters = rep(0, n)
  prev_clusters = rep(0, n)
  iter = 0
  repeat {
    iter = iter + 1
    counts = rep(0, k)
    prev_clusters = clusters

    for (i in 1:n) {
      # distance from Xi to each center
      diff = centroids - matrix(X[i,], nrow = k, ncol = p, byrow=T)
      dist = sqrt(rowSums(diff^2))
```

```
      # nearest cluster; j here is the temporary centroids
      j = which.min(dist)
      clusters[i] = j

      # Mac-Queen incremental updates of centroids
      counts[j] = counts[j] + 1
      centroids[j,] = centroids[j,] + (X[i,] - centroids[j,])/counts[j]
    }

    if (sum(prev_clusters != clusters) == 0) break
  }
} else if (iter == 'hierarchy') {
  clusters = rep(0, n)
  iter = 1
  for(i in 1:n) {
    diff = centroids - matrix(X[i,], nrow = k, ncol = p, byrow=T)
    dist = sqrt(rowSums(diff^2))
    clusters[i] = which.min(dist)
  }
} else {
  stop("Unknown iteration metric. Use 'Lloyd', 'Mac-Queen', or 'hierarchy'.")
}

# ------------------------ Output and Result -----------------------------
# build a distribution table: how many points in each cluster
tab = table(clusters)
nj = as.numeric(tab)

Xdf = data.frame(X)
Xs = split(Xdf, clusters)

SSC = numeric(k)
for (j in 1:k) {
  idx = (clusters == j)
  if (any(idx)) {
    Xj = X[idx, , drop=F]
    diff = Xj - matrix(centroids[j, ], nrow=nrow(Xj), ncol=p, byrow=T)
    SSC[j] = sum(rowSums(diff^2))
  }
}
TSSC = sum(SSC)

nearest_cluster = function(myx) {
  diff = centroids - matrix(myx, nrow = k, ncol = p, byrow=T)
  dist = sqrt(rowSums(diff^2))
  return(which.min(dist))
}

result = list('iteration_num'     = iter,
              'initial_centroids' = initial_centroids,
              'final_centroids'   = centroids,
              'cids'              = clusters,
              'nj'                = nj,
```

```
                'SSC'               = SSC,
                'aSSC'              = SSC/nj,
                'TSSC'              = TSSC,
                'aTSSC'             = TSSC/sum(nj),
                'nearest_cids'      = nearest_cluster)
  return(result)
}
```