```
# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import GridSearchCV

# Read Data
train = pd.read_hdf('train.h5', stop = 10000000) # Size 37,670,293 x 42
# Remove is_booking == 0 to match test set
train.drop(train[train['is_booking'] == 0 ].index , inplace=True)
train.drop('is_booking', axis=1, inplace = True)

#%%

# FEATURE SELECTION SECTION
predictors = [c for c in train.columns if c not in ["hotel_cluster"]]
xTrain, xTest, yTrain, yTest = train_test_split(train[predictors], train['hotel_cluster'], test_size =
0.2, random_state = 0)

num_feats = 10
#  Pearson's Correlation
def cor_selector(X, y,num_feats):
    cor_list = []
    feature_name = X.columns.tolist()
    # calculate the correlation with y for each feature
    for i in X.columns.tolist():
        cor = np.corrcoef(X[i], y)[0, 1]
        cor_list.append(cor)
    # replace NaN with 0
    cor_list = [0 if np.isnan(i) else i for i in cor_list]
```

```
    # feature name
    cor_feature = X.iloc[:,np.argsort(np.abs(cor_list))[-num_feats:]].columns.tolist()
    # feature selection? 0 for not select, 1 for select
    cor_support = [True if i in cor_feature else False for i in feature_name]
    return cor_support, cor_feature
cor_support, cor_feature = cor_selector(xTrain, yTrain,num_feats)

# Chi-squared
X_norm = MinMaxScaler().fit_transform(xTrain)
chi_selector = SelectKBest(chi2, k=num_feats)
chi_selector.fit(X_norm, yTrain)
chi_support = chi_selector.get_support()
chi_feature = xTrain.loc[:,chi_support].columns.tolist()

# Recursive feature elimination with logistic regression
rfe_selector = RFE(estimator=LogisticRegression(n_jobs= -1),
n_features_to_select=num_feats, step=10, verbose=1)
rfe_selector.fit(X_norm, yTrain)
rfe_support = rfe_selector.get_support()
rfe_feature = xTrain.loc[:,rfe_support].columns.tolist()

# lasso regression
embeded_lr_selector = SelectFromModel(LogisticRegression(penalty="l1", n_jobs = -1),
max_features=num_feats)
embeded_lr_selector.fit(X_norm, yTrain)
embeded_lr_support = embeded_lr_selector.get_support()
embeded_lr_feature = xTrain.loc[:,embeded_lr_support].columns.tolist()

# Random forest's feature importance
embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimators=100, n_jobs =
-1), max_features=num_feats)
embeded_rf_selector.fit(xTrain, yTrain)
embeded_rf_support = embeded_rf_selector.get_support()
embeded_rf_feature = xTrain.loc[:,embeded_rf_support].columns.tolist()

feature_name = xTrain.columns.tolist()
# put all selection together
feature_selection_df = pd.DataFrame({'Feature':feature_name, 'Pearson':cor_support,
'Chi-2':chi_support, 'RFE':rfe_support, 'Logistics':embeded_lr_support,
                                     'Random Forest':embeded_rf_support}) #,
'LightGBM':embeded_lgb_support
# count the selected times for each feature
feature_selection_df['Total'] = np.sum(feature_selection_df, axis=1)
```

```
feature_selection_df = feature_selection_df.sort_values(['Total','Feature'] , ascending=False)
feature_selection_df.index = range(1, len(feature_selection_df)+1)

# feature_selection_df.to_csv('feature_selection.csv')
feature_selection_df = pd.read_csv('feature_selection.csv')
print(feature_selection_df.columns)

# Only keep features in top 10 for at least 2 of the 5 methods
feats = []
for ind, row in feature_selection_df.iterrows():
    if row['Total']>= 2:
        feats.append(row['Feature'])

#%%

# THIS IS THE SECTION TO CREATE PREDICTIVE MODEL AND TUNE
HYPERPARAMETERS
scaler = MinMaxScaler()
train.columns = train.columns.astype(str)
train_rel = train[feats]
X_norm = scaler.fit_transform(train_rel)
xTrain, xTest, yTrain, yTest = train_test_split(X_norm, train['hotel_cluster'], test_size = 0.2,
random_state = 0)

# Create a knn model
knn = KNeighborsClassifier()
# Create a dictionary of all values we want to test for n_neighbors
params_knn = {'n_neighbors': np.arange(1,3,5)}
# Use gridsearch to test all values for n_neighbors
knn_gs = GridSearchCV(knn, params_knn, cv=3, verbose = 2, n_jobs = -1)
# Fit model to training data
knn_gs.fit(xTrain, yTrain)
# Save best model
knn_best = knn_gs.best_estimator_
# Check best n_neigbors value
print(knn_gs.best_params_)

# Create a random forest classifier
rf = RandomForestClassifier()
# Create a dictionary of all values we want to test for n_estimators
params_rf = {'n_estimators': [100],
             'max_depth': [5, 10],
             }
```

```
# Use gridsearch to test all values for n_estimators
rf_gs = GridSearchCV(rf, params_rf, cv=3, verbose = 2, n_jobs = -1)
# Fit model to training data
rf_gs.fit(xTrain, yTrain)
# Save best model
rf_best = rf_gs.best_estimator_
# Check best n_estimators value
print(rf_gs.best_params_)

# Create a new logistic regression model
log_reg = LogisticRegression()
params_log = {'penalty': ['l1', 'l2'],
              'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
             }
# Use gridsearch to test all values for n_estimators
log_gs = GridSearchCV(log_reg, params_log, cv=3, verbose = 2, n_jobs = -1)
# Fit the model to the training data
log_gs.fit(xTrain, yTrain)
# Save best model
log_best = log_gs.best_estimator_
# Check best n_estimators value
print(log_gs.best_params_)

# Create an MLP Classifier
mlp = MLPClassifier(max_iter=500)
params_log = {'hidden_layer_sizes': [(28,28,28), (14,14)]
              }
# Use gridsearch to test all values for n_estimators
mlp_gs = GridSearchCV(mlp, params_log, cv=3, verbose = 2, n_jobs = -1)
# Fit the model to the training data
mlp_gs.fit(xTrain, yTrain)
# Save best model
mlp_best = mlp_gs.best_estimator_
# Check best n_estimators value
print(mlp_gs.best_params_)

print('knn: {}'.format(knn_best.score(xTest, yTest)))
print('rf: {}'.format(rf_best.score(xTest, yTest)))
print('log_reg: {}'.format(log_best.score(xTest, yTest)))
print('mlp: {}'.format(mlp_best.score(xTest, yTest)))

# Create a dictionary of the best models
estimators=[('knn', knn_best), ('rf', rf_best), ('mlp', mlp_best)]
```

```
# Create voting classifier, inputting best models
ensemble = VotingClassifier(estimators, voting='hard')
# Fit model to training data
ensemble.fit(xTrain, yTrain)
# Test model on the cross-val data
ensemble.score(xTest, yTest)
#%%

# THIS IS THE SECTION TO MAKE PREDICTIONS
test = pd.read_hdf('test.h5') # Size 2,528,243 x 40
test.columns = test.columns.astype(str)
# Get only the 14 features used in training
test_rel = test[feats]
Y_norm = scaler.transform(test_rel)
predicts = ensemble.predict(Y_norm)
test['Predictions'] = predicts
test.to_excel('Hotel Predictions.xlsx')
```