DSC 478 Final Project Presentation

Project Title:

Potential Revenue Loss and Gain Prediction for Hotel Reservations

Team Members:

Bramhashree Manoharan, Hoda Masteri, Goutham Selvakumar

Fall 2022

Introduction

- One of the major reasons for potential revenue loss in the hotel industry is booking cancellations. However, not all cancellations lead to potential revenue loss, as some of the reservations are non-refundable or are canceled way ahead of arrival date.

- The goal of this project was to analyze a hotel booking dataset and train predictive classification models that could accurately predict whether a future reservation would lead to a potential revenue gain or loss. Such models can assist hoteliers in revenue management.

Dataset:

The dataset we used is the Hotel booking demand dataset from Kaggle. This dataset has 119390 rows and 32 columns. The columns include various aspects of hotel reservations.

Pre-processing

```
# Showing the columns containing Null values:
null_count = pd.DataFrame(hotel.isnull().sum()[hotel.isnull().sum()!=0], columns =["Null-count"])
print("Columns with Null values\n","-"*22,"\n", null_count)
```

```
Columns with Null values
 ----------------------
           Null-count
children            4
country           488
agent           16340
company        112593
```

- **Data cleaning & discretization:**

✓ **Children:** 4 null values: Can't be inferred -> dropping the 4 rows seemed to be the best move.
✓ **Country** has 488 null values: Binning: Replacing Nulls with "unknown". PRT is different from the rest (it gets its own bin). The rest can go into a single bin, the "other".
✓ **Agent** has 16340 null values: binning: Nulls -> "unknown". Agent 1 is different from the rest (it gets its own bin). The rest can go into a single bin, the "other".
✓ **Company** have nearly 94% missing values. No company stands out among others; the best action is to drop the company column entirely.

- **Feature engineering, data transformation and reduction:**

✓ Created a "**Revenue**" column using data from some of other columns, then selected Revenue as our target (label) and removed the columns we used to create the new target column.

✓ Created dummy variables from columns with categorical data type

✓ Used 12% of the dataset (randomly selected) and the 80-20 % rule to split the data into train and test sets

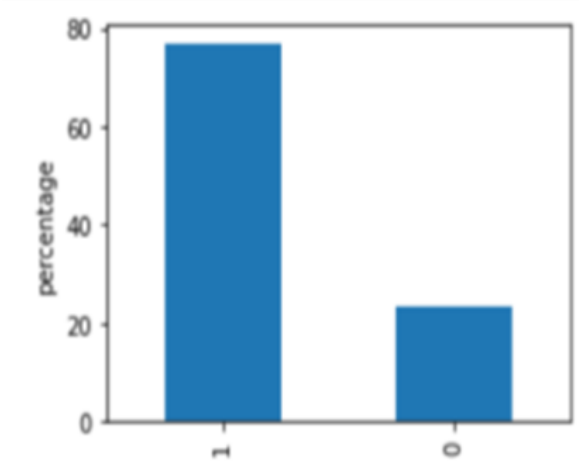
✓ Normalized the train and then test set using **MinMaxScaler**

```
profit = hotel.query('(reservation_status=="Canceled" &
                       (deposit_type == "Non Refund" |
                        Difference > 200)) | reservation_status=="Check-Out" |
                          (reservation_status=="No-Show" & deposit_type == "Non Refund")').copy()

# Loss is a collection of instances not captured by the above query
loss = hotel.loc[hotel.index.difference(profit.index)].copy()
```

```
# Creating a Revenue column for profit and loss dataframes and populating them:
profit["Revenue"] = 1
loss["Revenue"] = 0
hotel_2 = pd.concat([profit, loss], ignore_index=True)
hotel_2 = hotel_2.sample(frac=1).reset_index(drop=True)  # Shuffle and reset index
```

Exploratory Data Analysis

- To understand the dataset's main characteristics:



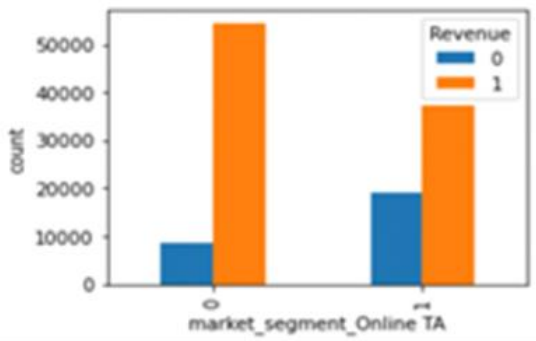
Bar chart of Revenue column (Profit:1 and loss:0)

```
corrMatrix = hotel_ssf.corr()
corrMatrix["Revenue"].sort_values(ascending=False)[:5]
```

```
Revenue                          1.00000
deposit_type_Non Refund          0.20554
required_car_parking_spaces      0.14034
market_segment_Offline TA/TO     0.11960
market_segment_Groups            0.10615
Name: Revenue, dtype: float64
```

```
corrMatrix["Revenue"].sort_values(ascending=False)[-5:]
```

```
distribution_channel_TA/TO   -0.09863
adr                          -0.12158
agent_listed_other           -0.12832
deposit_type_No Deposit      -0.20466
market_segment_Online TA     -0.24340
Name: Revenue, dtype: float64
```



Cross-tabulation for target variable "Revenue" and deposit types, and online market segment

Creating Classification Models

1. We utilized various classification algorithms (single and ensemble models) and by means of grid search and other algorithms we found the best parameters for each model.

2. Finally, we analyzed the accuracies of the various classifiers to select the best one that can predict the potential revenue gain or loss for each reservation with the highest accuracy.

K NEAREST NEIGHBOR

KNN is a supervised learning classifier, and it relies on data similarities and distance metrics to generate accurate predictions. Furthermore, we used KNeighborsClassifier from sci-kit-learn library to implement the k-nearest neighbors' vote. Since the knn follows only one parameter, this makes the parameter tuning to be easier. We performed a grid search for parameters of values [k=1, 3, 5, 7, & 9] and cross-validation of 5. The model accuracy that we got was increased slightly after the hyperparameter tuning by 4%. The optimal value of n_neighbors was 7, with which the **overall (cross-validation), the train, and the test accuracies as 0.79(+/- 0.01), 0.93%, and 0.78% respectively.**

```
from sklearn.model_selection import train_test_split
#Using 80/20 split
train, test, target_train, target_test = train_test_split(X_ssf, y, test_size=0.2, random_state=33)

from sklearn import preprocessing
#Normalizing the data
min_max_scaler = preprocessing.MinMaxScaler().fit(train)
#Min-Max normalization to scale all the variables between 0 & 1
train_norm = min_max_scaler.transform(train)
```

```
Overall Accuracy on X-Val: 0.79 (+/- 0.02)

Accuracy on Training:  0.932278923453338

Accuracy on Testing:  0.787836420831877
```

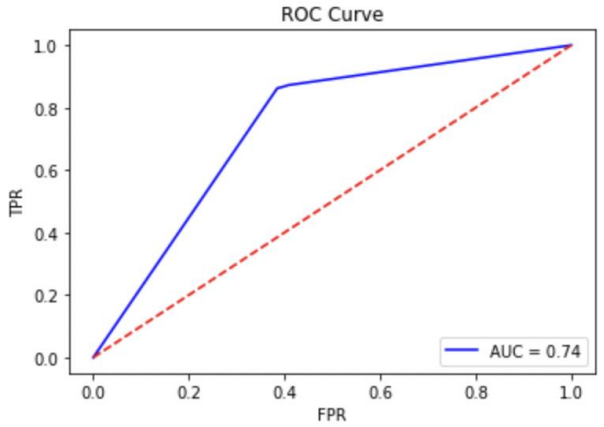
```
#Parameters for grid search (values of k=1, k=3, k=5, k=7, k=9)
parameters = {'n_neighbors': [1,3,5,7,9]}
#Initializing the grid search with specified parameters & 5-fold cross-validation
gs = GridSearchCV(clf, parameters, verbose=1, cv=5)
```

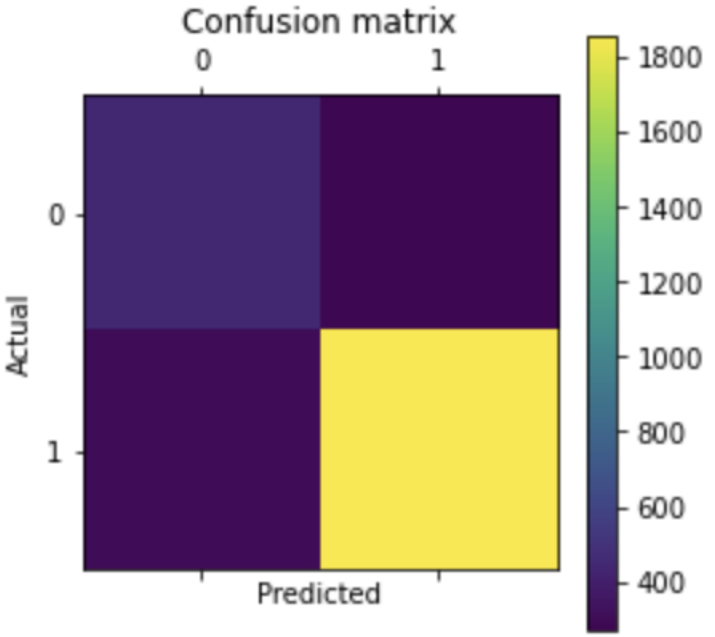
DECISION TREE

We used **tree.DecisionTreeClassifier** from Scikit-learn library

Accuracies with Default parameters:

```
In [82]: print (treeclf.score(train1, target_train1))
         0.9928346731911919

In [83]: print (treeclf.score(test1, target_test1))
         0.8014680181754631
```





Optimal parameters generated through Grid Search:

```
Fitting 5 folds for each of 3000 candidates, totalling 15000 fits
CPU times: user 8min 5s, sys: 1.19 s, total: 8min 6s
Wall time: 8min 7s

({'criterion': 'entropy',
  'max_depth': 11,
  'min_samples_leaf': 17,
  'min_samples_split': 4},
 0.8218290501547383)
```

```
Overall Accuracy on X-Val: 0.82 (+/- 0.02)

Accuracy on Training:  0.8405277874868927

Accuracy on Testing:  0.8276826284515904
```

There was not much difference in the accuracies after the hyperparameter tuning.

To visually represent the decision tree, we used the **graph viz**.

It helped us to explore and understand the data relationship in a tree hierarchy structure.

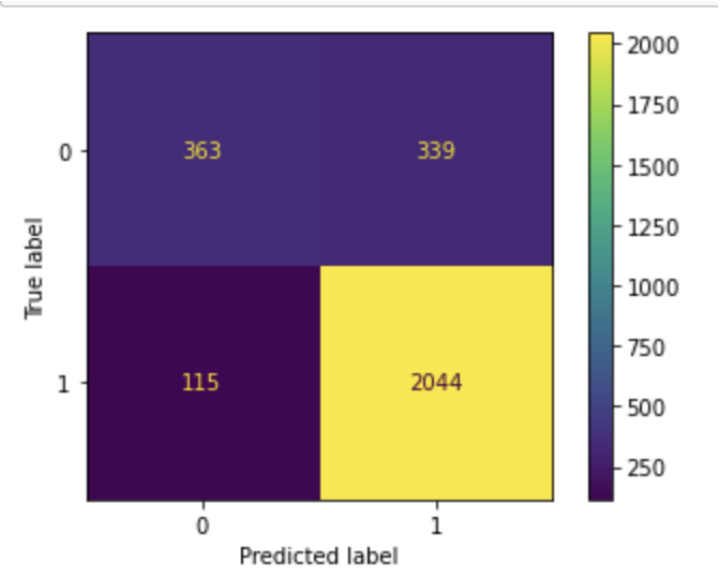
RANDOM FOREST

We used **ensemble.RandomForestClassifier** from Scikit-learn Library

Optimal parameters generated through grid search:

```
params = {
    'n_estimators':[100, 200, 300, 400, 500, 600],
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [5, 10, 15, 20],
}
model = RandomForestClassifier(random_state=33)
rf_best = grid_search(train, test, target_train, target_test, model, params, scoring='accuracy')
```

```
RandomForestClassifier(random_state=33)
Best parameter      : {'criterion': 'entropy', 'max_depth': 20, 'n_estimators': 600}
```



The test accuracy significantly increased after the hyperparameter tuning.

```
Overall Accuracy on X-Val: 0.85 (+/- 0.02)

Accuracy on Training:  0.9580566235581964

Accuracy on Testing:  0.9954561342188046
```

Random Forest is an ensemble model which can detect non-linearity and it resulted in better accuracy as we had a huge dataset.

SUPPORT VECTOR MACHINE

We used SVC from sci-kit-learn library to create an SVM classifier. We implemented grid search to optimize C, gamma, and kernel parameters. The SVM classifier was initialized with the balanced class weights. Setting up the parameters for our model and initializing the grid search with the cross-fold validation of 3. We are isolating the target variable as the 'Revenue'. Fitting the 3 folds for each of the 50 candidates, totaling 150 fits. Next, is to predict the classes of test data and print the classification report for SVM accordingly. Moreover, the SVM did not perform well with large datasets as the complexity of SVM is highly dependent on the size of the dataset. SVM is most effective in high dimensional spaces. The performance of the hyperparameter tuning did not show any significant performance improvement as from the classification report we can get the result of an **overall accuracy being 0.79 (+/- 0.01), the training and testing data accuracies being 0.95%, and 0.98%** respectively.

```
# grid search parameters
parameters = {'gamma': [0.1, .25, .4, .5, .7], 'C': [20, 40, 50, 75, 100], 'kernel': ['rbf', 'linear']}
# initialize grid search with 3-fold cv
gs4 = GridSearchCV(clf4, parameters, verbose=1, cv=3)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits

GridSearchCV(cv=3,
             estimator=SVC(cache_size=600, class_weight='balanced', tol=0.01),
             param_grid={'C': [20, 40, 50, 75, 100],
                         'gamma': [0.1, 0.25, 0.4, 0.5, 0.7],
                         'kernel': ['rbf', 'linear']},
             verbose=1)
```

```
Overall Accuracy on X-Val: 0.79 (+/- 0.01)

Accuracy on Training:  0.9529010835372247

Accuracy on Testing:  0.9849702901083537
```

NAIVE BAYES

- Used naive_bayes.GaussianNB from Scikit-learn Library
- Accuracies when default parameters used:

```
Overall Accuracy: 0.46 (+/- 0.02)

Accuracy on Training:  0.4556099265990912

Accuracy on 20% test set:  0.4627752534078993
```

```
# Hyperparameter tuning for Naive Bayes:
from sklearn.model_selection import GridSearchCV

parameters = {'var_smoothing': (1e-9,1e-7, 1e-5, 1e-4,1e-2)}

gs = GridSearchCV(nbclf, parameters, scoring = 'accuracy', cv=10)
```

```
_ = gs.fit(train, target_train)

gs.best_params_, gs.best_score_
```

```
({'var_smoothing': 0.01}, 0.5263936391119797)
```

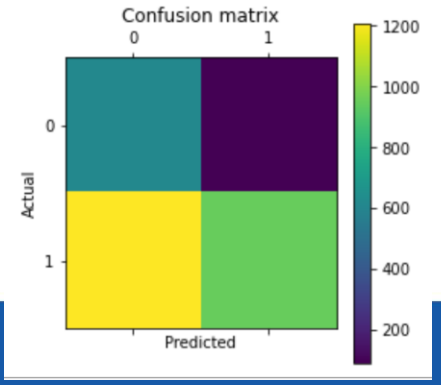
```
# Creating the best model based on the above grid search results:

best_nbclf = naive_bayes.GaussianNB(var_smoothing= 0.01)
```

- Train, cross-validation, and test accuracies after tuning (improved slightly) :

```
Overall Accuracy: 0.53 (+/- 0.03)

Accuracy on Training:  0.5297972736805313

Accuracy on 20% test set:  0.5456134218804614


classification report:

              precision    recall  f1-score   support

           0       0.34      0.87      0.48       702
           1       0.91      0.44      0.59      2159

    accuracy                           0.55      2861
   macro avg       0.62      0.66      0.54      2861
weighted avg       0.77      0.55      0.57      2861
```

- GaussianNB assumes the distributions of independent variables are normal (not very suitable for our dataset)

```
[[ 611   91]
 [1209  950]]
```



LINEAR DISCRIMINANT ANALYSIS (LDA)

- Used Linear Discriminant Analysis from scikit-learn library
- Accuracies when default parameters used:

```
Overall Accuracy: 0.78 (+/- 0.02)

Accuracy on Training:  0.7866130723523244

Accuracy on test set:  0.7790982174065012
```

- **Hyperparameter tuning :**

```
# hyperparameter tuning for LDA:
# The most important parameter to tune in LDA is the solver:

parameters = {'solver': ['svd', 'lsqr']}
gs = GridSearchCV(ldclf, parameters, scoring='accuracy', cv=10)
gs_res = gs.fit(train, target_train)

gs_res.best_params_, gs_res.best_score_
```

```
({'solver': 'svd'}, 0.7835525391638928)
```

```
# Checkig if auto Shrinkage combined with the two other solvers provides a higher accuracy:
# Side note: svd only works with shrinkage=None

ldclf_2 = LinearDiscriminantAnalysis(shrinkage='auto')
parameters_2 = {'solver': ('lsqr','eigen')}

gs = GridSearchCV(ldclf_2, parameters_2, scoring='accuracy', cv=10)
gs_res = gs.fit(train, target_train)

gs_res.best_params_, gs_res.best_score_
```

```
({'solver': 'lsqr'}, 0.778136317830641)
```

```
best_ldclf = LinearDiscriminantAnalysis(solver='svd')
```

- Train, cross-validation, and test accuracies after tuning (improved slightly) :

```
Overall Accuracy: 0.78 (+/- 0.02)

Accuracy on Training:  0.7866130723523244

Accuracy on test set:  0.7790982174065012


classification report:

              precision    recall  f1-score   support

           0       0.59      0.32      0.41       702
           1       0.81      0.93      0.86      2159

    accuracy                           0.78      2861
   macro avg       0.70      0.62      0.64      2861
weighted avg       0.75      0.78      0.75      2861
```

- we did not manage to increase the accuracy further. The reason may be that Linear Discriminant Analysis (LDA) is more suited to continuous numerical variables rather than dummy variables with few unique values.

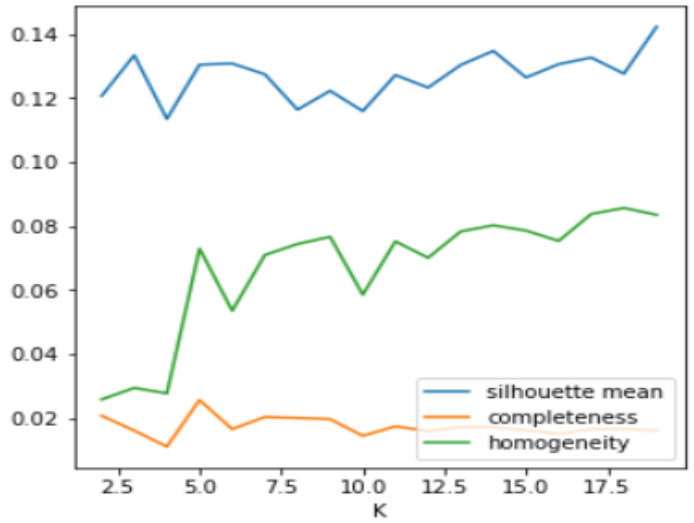
```
# generate the confusion matrix
ldcm = confusion_matrix(target_test, best_ldclf_test)
print(ldcm)
```

```
[[ 222  480]
 [ 152 2007]]
```

Unsupervised Knowledge Discovery(Clustering):

- Used **cluster.KMeans from** Scikit-learn library

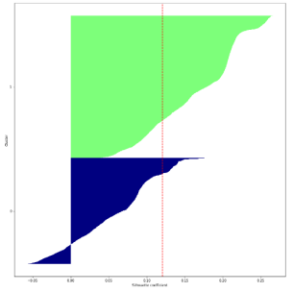
- Tried different values of k, from 2 to 20:



- Best k: k=5, k=7, and k=2.
- We picked k=2 because it enables us to compare only two centroids and to see how similar they are to the two classes (Revenue=1 and Revenue=0).

```
sil,comp,hmg,clust,cent=cluster_analysis(X_norm,X_ssf.columns,y,plot_silhouettes,k=2)
```

```
----------Cluster Analysis----------

Number of clusters: 2
Size of Cluster 0 =  6104
Size of Cluster 1 =  8201
```



- **Centroid 1**: has characteristics of revenue loss (Revenue=0) class: higher lead-time, lower stays_in_weekend_nights, lower number of children, higher previous_cancellations, higher days_in_waiting_list, lower number of total_of_special_requests, higher agent_1, etc.

- **Centroid 0:** has characteristics of a profit (Revenue=1) class: lower lead-time, higher stays_in_weekend_nights, higher number of children, lower previous_cancellations, lower days_in_waiting_list, higher number of total_of_special_requests, low agent_1, etc.

We cannot however conclude that the above findings hold for most of the data because the homogeneity and completeness scores are not high.

```
# Analysis of Cluster centroids 0 and 1:

cent.T   # centroid 0: shows characteristics of profit; Centroid 1: loss
```

	0	1
lead_time	0.17278	0.11522
arrival_date_year	0.48648	0.64181
arrival_date_week_number	0.52169	0.48953
arrival_date_day_of_month	0.49641	0.49122
stays_in_weekend_nights	0.04530	0.06727
stays_in_week_nights	0.05394	0.06863
adults	0.06369	0.07260

CONCLUSION

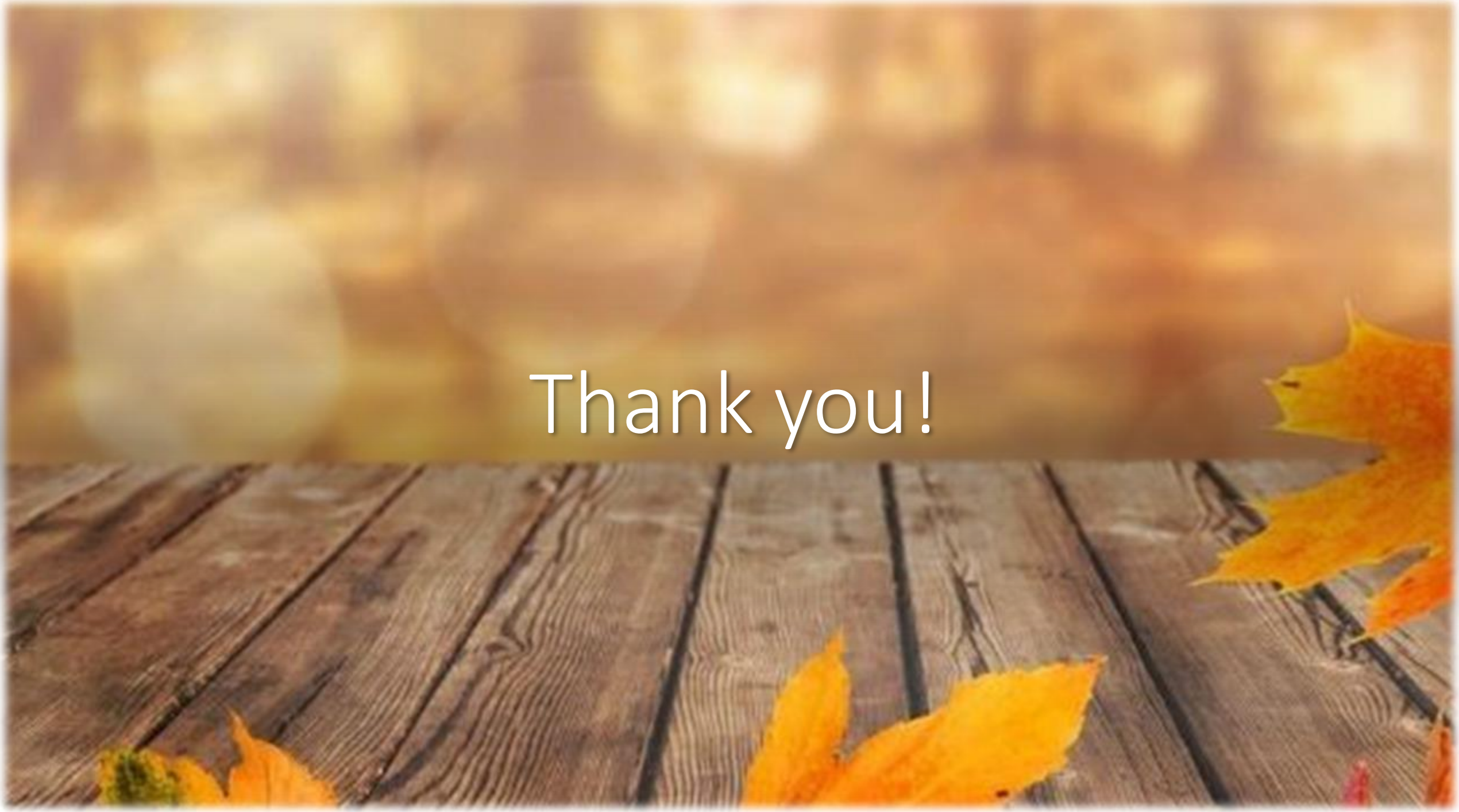
Below are the Overall Accuracies for each classifier:

Random Forest:	**0.85 (+/- 0.02)**
Decision Trees:	**0.82 (+/- 0.02)**
KNN:	**0.79 (+/- 0.01)**
SVM:	**0.79 (+/- 0.01)**
LDA:	**0.78 (+/- 0.02)**
Naïve Bayes:	**0.53 (+/- 0.03)**

Based on our analysis of each classifier,

- we conclude that **Random Forest**, being an ensemble model, outperformed the simple classifiers with an **overall accuracy of 85%**.

- In addition, the results, and observations from the unsupervised knowledge discovery (clustering) helped us in pattern recognition in the sense that the findings agreed with the information we had previously obtained from exploratory data analysis and visualizations.


Thank you!