```
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import precision_score, recall_score,
    recall_score, accuracy_score, roc_auc_score, classification_
from sklearn.tree.export_graphviz report, reportf, f_score
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

Step 1

```
df = pd.read_csv("Employee_Hopping.csv")
df.head()
df.shape
df.columns
df.dtypes
df.value_counts()
```

Step: 2

```
x = df.drop(['Attrition'], axis=1)
y = df.Attrition
y = y.apply(lambda x: 1 if x == 'Yes' else 0)
df.select_dtypes(include=['object']).dtypes
```

Lab9. Employee Hopping Prediction using Random Forests

Objectives

In this lab, you will build a classification model to predict whether employee will continue to work or quit his job in the company using Random Forest classifier.

Learning Outcomes

After completing this lab, you will be able to

- Perform One Hot Encoding on categorical columns
- Create RandomForestClassifier, perform training and testing
- Print accuracy score and classification report
- Print feature importance values
- Select the best number of trees based on out-of-bag error values
- Compare against Decision Tree model
- Visualize trees using graphviz

Business Use Case

You are a data scientist. Heber Software Solutions is a leading IT industry in your city. They have collected the details of employees and if they work or left the company. They ask you to build a prediction model so that they can use your model to check if employees will continue to work or leave. Based on this analysis, they will understand what make them to quit and accordingly they will design employee welfare management schemes.

Step1. [Understand Data].

- Using Pandas, import **"Employee_Hopping.csv "** file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Extract X and y]

- Create X and y columns from the dataframe

Step3. [Feature Engineering]

- There are 8 categorical columns (where dtype="object"). Perform one hot encoding and create new columns

Step4. Now, check shape of X and y.

Step5. [Model Development]

- Split X and y for training and testing
- Create RandomForestClassifier model, fit (no need to scale) and predict

Step6. [Testing]

- Print accuracy score between y_test and y_pred
- Print classification report between y_test and y_pred and observe the results

Step7. [Feature importance value]

- You can look at feature importance values using the property, rf.feature_importances_
- Print feature name and its rf.feature_importances_ values and understand important features
- Show a Bar plot between feature column names and feature_importances_ score.

Step8. [Visualize your RF Decision Tree using graphviz]

- Figure below show a segment of the RF tree which is visualized at http://www.webgraphviz.com/. You should copy and paste .dot file in this page.

Step: 3 [Feature Engineering]

```
df = pd.get_dummies(df, columns=["BusinessTravel", "Department",
                                 "EducationField", "Gender", "JobRole",
                                 "MaritalStatus", "over18", "OverTime"])
df.head()
```

Step: 4:

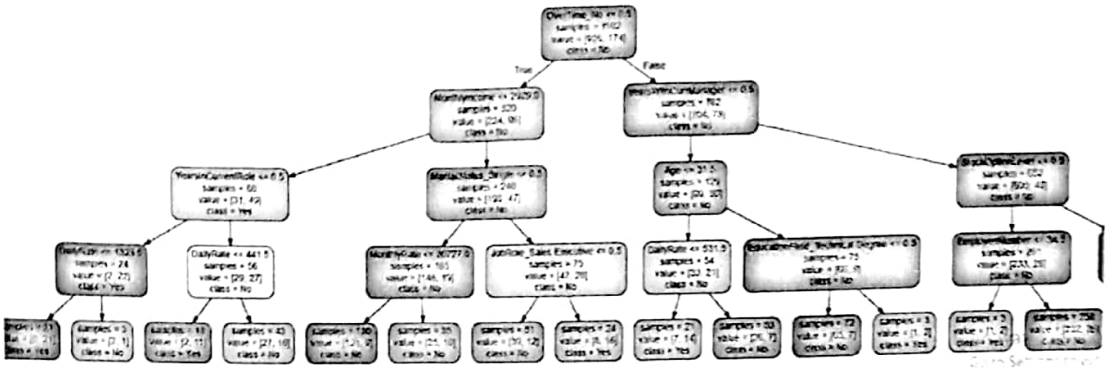
```
x = df.drop("Attrition", axis=1)
x.shape
y.shape
```

Step: 5

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
                                                    random_state=0)
seed=0
rfc = RandomForestClassifier(n_estimators=1000, max_features=0.3,
                             max_depth=4, min_samples_leaf=2,
                             n_jobs=-1, random_state=seed, warm_start=True,
                             verbose=0)
rfc.fit(x_train, y_train)
y_pred = rfc.predict(x_test)
y_pred
```

Step: 6

```
accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```



Step9. [RF with a range of trees]

Fit random forest models with a range of tree numbers [15, 20, 30, 40, 50, 100, 150, 200, 300, 400] and print Out-Of-Bag error for each of these model. Use model.oob_score_ to get score and subtract this score from 1 to get the oob-error. That is, oob-error = 1 - model.oob_score_.

Hint: since the only thing changing is the number of trees, the `warm_start` flag can be used so that the model just adds more trees to the existing model each time. Use the `set_params` method to update the number of trees. The following code many help to understand this part.

```
rf2 = RandomForestClassifier(oob_score=True,
                             random_state=42,
                             warm_start=True,
                             n_jobs=-1)

oob_list = list()

# Iterate through all of the possibilities for number of trees
for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:

    # Use this to set the number of trees
    rf2.set_params(n_estimators=n_trees)

    # Fit the model
    rf2.fit(X_train, y_train)

    # Get the oob error
    oob_error = 1 - rf2.oob_score_

    # Store it
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')

rf_oob_df
```

Step10. [Plot oob-error for each tree]

The following lines will help you

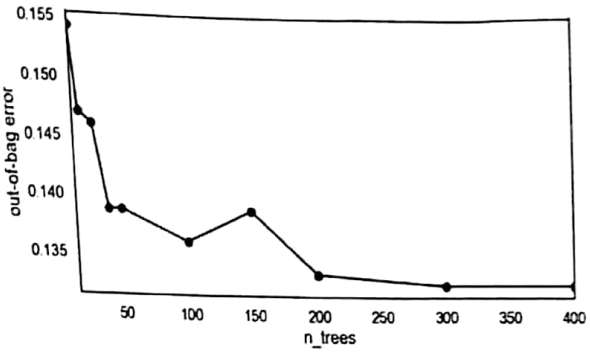
```
ax = rf_oob_df.plot(legend=False, marker='o', figsize=(10,5))
ax.set(ylabel='out-of-bag error')
```

Step 7:

```
print(rfc.feature_importances_)
feature_imp = pd.DataFrame(rfc.feature_importances_, index=X_train.
                           columns, columns=['Important score'])
feature_imp
plt.figure(figsize=(20,10))
sns.barplot(x=feature_imp.index, y=feature_imp['Important score'])
plt.ylabel("Feature Importance Score")
plt.xlabel('Features')
plt.title("Visualizing Important features")
plt.yticks(rotation=45)
plt.show()
```

Step 8:

```
estimator = rfc.estimators_[5]
export_graphviz(estimator, out_file='tree1.txt',
                feature_names = X_train.columns.values,
                class_names=['Yes', 'No'], rounded = true,
                proportion = false, precision=2, filled = true
```



Step11. [Compare with DecisionTreeClassifier]

- Create DecisionTreeClassifier, fit and predict on test set
- Visualize the tree using graphviz
- Print accuracy score
- Print classification report
- What is the result of the comparision between RF and DT models? Which gives best accuracy?.
- What is your comment on precision, recall, f1 score values?

Step:9

```
rf2 = RandomForestClassifier(oob_score=True, random_state=42,
                             warm_start=True, n_jobs=-1)
oob_list = list()
for n_trees in [15,20,30,40,50,100,150,200,300,400]:
    rf2.set_params(n_estimators=n_trees)
    rf2.fit(x_train, y_train)
    oob_error = 1 - rf2.oob_score_
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))
rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')
rf_oob_df
ax = rf_oob_df.plot(legend=False, marker='o', figsize=(10,5))
ax.set(ylabel='out-of-bag error')
```

Step : 11

```
clf_1 = DecisionTreeClassifier(criterion='gini', max_depth=4,
                               random_state=42)
clf_1.fit(X_train, y_train)

y_pred1 = clf_1.predict(x_test)
y_pred1

with open("tree2.txt", 'w') as f:
    f = tree.export_graphviz(clf_1, out_file=f, max_depth=4,
                             impurity=False, feature_names=
                             X_train.columns.values,
                             class_names=['yes','no'], filled=True)

accuracy = score(y_test, y_pred1)
print(classification_report(y_test, y_pred1))

print("RF Model: ", accuracy_score(y_test, y_pred))
print("RF Precision: ", precision_score(y_test, y_pred))
print("RF Recall: ", recall_score(y_test, y_pred))
print("RF F1 score: ", f1_score(y_test, y_pred))
print("\n")
print("DT model: ", accuracy_score(y_test, y_pred1))
print("DT Precision: ", precision_score(y_test, y_pred1))
print("DT Recall: ", recall_score(y_test, y_pred1))
print("DT F1 score: ", f1_score(y_test, y_pred1))
```