**Python Lab for Tree Based Algorithm**

**1. Required Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV, KFold

from sklearn.metrics import mean\_squared\_error

from sklearn.tree import DecisionTreeRegressor, plot\_tree

from sklearn.ensemble import BaggingRegressor, RandomForestRegressor

**2. Load and Preprocess Data**

df = pd.read\_csv("business\_data.csv")

df = df.apply(pd.to\_numeric, errors='coerce')

df.dropna(inplace=True)

X = df.drop(columns=["Revenue"])

y = df["Revenue"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

**3. Baseline Model: Mean Predictor**

y\_pred\_baseline = [y\_train.mean()] \* len(y\_test)

mse\_baseline = mean\_squared\_error(y\_test, y\_pred\_baseline)

print(f"Baseline MSE (mean predictor): {mse\_baseline:.2f}")

**4. Regression Tree Model (with Pruning)**

tree = DecisionTreeRegressor(random\_state=42)

path = tree.cost\_complexity\_pruning\_path(X\_train, y\_train)

ccp\_alphas = path.ccp\_alphas

trees = [DecisionTreeRegressor(random\_state=42, ccp\_alpha=alpha).fit(X\_train, y\_train) for alpha in ccp\_alphas]

mse\_values = [mean\_squared\_error(y\_test, t.predict(X\_test)) for t in trees]

best\_index = np.argmin(mse\_values)

pruned\_tree = trees[best\_index]

print(f"Best pruned tree index: {best\_index}, MSE: {mse\_values[best\_index]:.2f}")

# Plot: MSE vs alpha

plt.figure()

plt.plot(ccp\_alphas, mse\_values, marker='o')

plt.title("Pruning Path: Alpha vs MSE")

plt.xlabel("ccp\_alpha")

plt.ylabel("Test MSE")

plt.grid(True)

plt.show()

# Plot tree

plt.figure(figsize=(10, 6))

plot\_tree(pruned\_tree, feature\_names=X.columns, filled=True, fontsize=6)

plt.title(f"Pruned Decision Tree")

plt.show()

**5. Bagging with Cross-Validation for number of trees**

n\_estimators\_grid = [10, 50, 100, 200]

cv\_rmse = []

for n in n\_estimators\_grid:

model = BaggingRegressor(n\_estimators=n, random\_state=42)

scores = cross\_val\_score(model, X\_train, y\_train, scoring='neg\_root\_mean\_squared\_error', cv=10)

cv\_rmse.append(-scores.mean())

best\_n\_bag = n\_estimators\_grid[np.argmin(cv\_rmse)]

print(f"Best Bagging ntree = {best\_n\_bag}")

# Plot: CV RMSE vs ntree

plt.figure()

plt.plot(n\_estimators\_grid, cv\_rmse, marker='o')

plt.title("Bagging: CV RMSE vs #Trees")

plt.xlabel("Number of Trees")

plt.ylabel("CV RMSE")

plt.grid(True)

plt.show()

**6. Final Bagging Model**

bag\_model = BaggingRegressor(n\_estimators=best\_n\_bag, random\_state=42).fit(X\_train, y\_train)

y\_pred\_bag = bag\_model.predict(X\_test)

print(f"Final Bagging Model MSE: {mean\_squared\_error(y\_test, y\_pred\_bag):.2f}")

**7. Random Forest: Tune number of predictors**

param\_grid = {'max\_features': list(range(1, X.shape[1] + 1))}

rf\_grid = GridSearchCV(RandomForestRegressor(n\_estimators=200, random\_state=42),

param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

rf\_grid.fit(X\_train, y\_train)

best\_mtry = rf\_grid.best\_params\_['max\_features']

print(f"Best RF max\_features (mtry): {best\_mtry}")

**8. Random Forest: Tune number of trees via CV**

ntree\_grid = [50, 100, 150, 200, 250, 500, 1000]

cv\_mse\_rf = []

for nt in ntree\_grid:

rf = RandomForestRegressor(n\_estimators=nt, max\_features=best\_mtry, random\_state=42)

scores = cross\_val\_score(rf, X\_train, y\_train, scoring='neg\_mean\_squared\_error', cv=10)

cv\_mse\_rf.append(-scores.mean())

best\_ntree\_rf = ntree\_grid[np.argmin(cv\_mse\_rf)]

print(f"Best ntree for RF (CV MSE): {best\_ntree\_rf}")

# Plot: CV MSE vs ntree

plt.figure()

plt.plot(ntree\_grid, cv\_mse\_rf, marker='o')

plt.title("RF: CV MSE vs #Trees")

plt.xlabel("Number of Trees")

plt.ylabel("CV MSE")

plt.grid(True)

plt.show()

**9. Final Random Forest Model**

final\_rf = RandomForestRegressor(

n\_estimators=best\_ntree\_rf,

max\_features=best\_mtry,

random\_state=42,

oob\_score=True,

bootstrap=True

).fit(X\_train, y\_train)

print(f"OOB Score (R²): {final\_rf.oob\_score\_:.4f}")

# Plot: OOB MSE convergence

oob\_errors = [mean\_squared\_error(y\_train, tree.predict(X\_train)) for tree in final\_rf.estimators\_]

plt.figure()

plt.plot(range(1, best\_ntree\_rf+1), oob\_errors)

plt.title("RF: OOB MSE vs #Trees")

plt.xlabel("Tree Index")

plt.ylabel("Train MSE")

plt.grid(True)

plt.show()

# Variable importance

importances = pd.Series(final\_rf.feature\_importances\_, index=X.columns).sort\_values(ascending=False)

print("Top 5 important features:\n", importances.head())

# Plot importance

importances.head(10).plot(kind='barh', title="Top 10 RF Feature Importances")

plt.xlabel("Importance")

plt.gca().invert\_yaxis()

plt.show()

# Final test MSE

y\_pred\_rf = final\_rf.predict(X\_test)

print(f"Final RF Model MSE: {mean\_squared\_error(y\_test, y\_pred\_rf):.2f}")