

## ASSIGNMENT – 1

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** Fibonacci Numbers (Recursive and Non-Recursive)

**Aim:**

Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

**Code:**

```
def recursive_fibonacci(n):
    if n<=1:
        return n
    else:
        return recursive_fibonacci(n-1) + recursive_fibonacci(n-2)

def non_recursive_fibonacci(n):
    first=0
    second=1
    print(first)
    print(second)
    while n-2>0:
        third = first + second
        first=second
        second=third
        print(third)
        n-=1
```

```
if __name__ == "__main__":
    n=10
    for i in range(n):
        print(recursive_fibonacci(i))

non_recursive_fibonacci(n)
```

## Output:

Enter the number n to find the nth Fibonacci number: 10

0  
1  
1  
2  
3  
5  
8  
13  
21  
34

Non-Recursive Fibonacci of 10 is: 34

Time Complexity (Non-Recursive): O(n)

Space Complexity (Non-Recursive): O(1)

Recursive Fibonacci of 10 is: 34

Time Complexity (Recursive): O( $2^n$ )

Space Complexity (Recursive): O(n)

## ASSIGNMENT – 2

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** Huffman Encoding (Greedy Strategy)

**Aim:**

Write a program to implement Huffman Encoding using a greedy strategy.

**Code:**

```
import heapq

class node:

    def __init__(self,freq,symbol,left=None,right=None):
        self.freq=freq
        self.symbol=symbol
        self.left=left
        self.right=right
        self.huff= ""

    def __lt__(self,nxt):
        return self.freq<nxt.freq

def printnodes(node,val=""):

    newval=val+str(node.huff)

    if node.left:
        printnodes(node.left,newval)

    printnodes(node.right,newval)
```

```

if node.right:
    printnodes(node.right,newval)

if not node.left and not node.right:
    print("{} -> {}".format(node.symbol,newval))

if __name__=="__main__":
    chars = ['a', 'b', 'c', 'd', 'e', 'f']
    freq = [ 5, 9, 12, 13, 16, 45]
    nodes=[]

for i in range(len(chars)):
    heapq.heappush(nodes, node(freq[i],chars[i]))

while len(nodes)>1:
    left=heapq.heappop(nodes)
    right=heapq.heappop(nodes)

    left.huff = 0
    right.huff = 1

    newnode = node(left.freq + right.freq , left.symbol + right.symbol , left , right)
    heapq.heappush(nodes, newnode)

printnodes(nodes[0])

```

## **Output:**

Huffman Codes:

Character | Code-Word

---

f	-> 0
c	-> 100
d	-> 101
a	-> 1100
b	-> 1101
e	-> 111

Time Complexity:  $O(n \log n)$  where  $n$  is the number of unique characters.

Space Complexity:  $O(n)$  to store the tree and codes.

## ASSIGNMENT – 3

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** Fractional Knapsack (Greedy Method)

**Aim:**

Write a program to solve a fractional Knapsack problem using a greedy method.

**Code:**

```
def fractional_knapsack():
    weights=[10,20,30]
    values=[60,100,120]
    capacity=50
    res=0

    for pair in sorted(zip(weights,values), key= lambda x: x[1]/x[0], reverse=True):
        if capacity<=0:
            break
        if pair[0]>capacity:
            res+=int(capacity * (pair[1]/pair[0]))
            capacity=0
        elif pair[0]<=capacity:
            res+=pair[1]
            capacity-=pair[0]
    print(res)

if __name__=="__main__":
    fractional_knapsack()
```

**Output:**

Maximum value in Knapsack for capacity 50 is: 240.00

Time Complexity:  $O(n \log n)$  due to sorting.

Space Complexity:  $O(n)$  for Item objects.

## ASSIGNMENT – 4

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** 0-1 Knapsack (Dynamic Programming)

**Aim:**

Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy

**Code:**

```
def solve_knapsack():

    val=[50,100,150,200] #value array
    wt=[8,16,32,40] # Weight array
    W=64
    n=len(val) - 1

    def knapsack(W,n):
        #base case
        if n<0 or W<=0:
            return 0

        if wt[n]>W:
            return knapsack(W, n-1)

        else:
            return max(val[n] + knapsack(W-wt[n],n-1),knapsack(W,n-1))

    print(knapsack(W,n))

if __name__=="__main__":
    solve_knapsack()
```

**Output:**

Maximum value for 0/1 Knapsack (DP) is: 350

Time Complexity:  $O(n*W)$  where n is number of items and W is the capacity.

Space Complexity:  $O(n*W)$  for the DP table.

## ASSIGNMENT – 5

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** N-Queens Problem (Backtracking)

**Aim:**

Design -Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final -queen\_s matrix.

**Code:**

```
def n_queens(n):
    col = set()
    posDiag=set() # (r+c)
    negDiag=set() # (r-c)

    res=[]

    board = [ ["0"]*n for i in range(n) ]

    def backtrack(r):
        if r==n:
            copy = " ".join(row) for row in board]
            res.append(copy)
            return

        for c in range(n):
            if c in col or (r+c) in posDiag or (r-c) in negDiag:
                continue

            col.add(c)
            posDiag.add(r+c)
            negDiag.add(r-c)

            backtrack(r+1)

            col.remove(c)
            posDiag.remove(r+c)
            negDiag.remove(r-c)
```

```

col.add(c)
posDiag.add(r+c)
negDiag.add(r-c)
board[r][c]="1"

backtrack(r+1)

col.remove(c)
posDiag.remove(r+c)
negDiag.remove(r-c)
board[r][c]="0"

backtrack(0)
for sol in res:
    for row in sol:
        print(row)
    print()

if __name__=="__main__":
    n_queens(4)

```

## **Output:**

Enter the board size N (e.g., 4, 8): 4

Enter the row for the pre-placed Queen (0 to N-1, e.g., 1 for 4x4): 1

Enter the column for the pre-placed Queen (0 to N-1, e.g., 0 for 4x4): 0

Solution 1:

0 1 0 0

0 0 0 1

1 0 0 0

0 0 1 0

Solution 2:

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

Time Complexity (Worst Case):  $O(N!)$

Space Complexity:  $O(N^2)$  for the board array +  $O(N)$  for recursion stack.

## ASSIGNMENT – 6

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** Uber Ride Price Prediction (Regression)

**Aim:**

Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform data preprocessing, identify outliers, check correlation, implement Linear Regression and Random Forest Regression, and evaluate models (R<sup>2</sup>, RMSE).

**Code:**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from geopy.distance import geodesic
# Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset
data = pd.read_csv("uber.csv")
print("Original Dataset Shape:", data.shape)
print(data.head())
# --- 2. Basic cleaning ---
data = data.dropna(subset=["pickup_longitude", "pickup_latitude",
                           "dropoff_longitude", "dropoff_latitude", "fare_amount"])

data = data[(data["fare_amount"] > 0) & (data["fare_amount"] < 500)]
data = data[(data["pickup_longitude"].between(-180, 180)) &
            (data["dropoff_longitude"].between(-180, 180)) &
            (data["pickup_latitude"].between(-90, 90)) &
            (data["dropoff_latitude"].between(-90, 90))]

def calculate_distance(row):
    pickup = (row["pickup_latitude"], row["pickup_longitude"])
    dropoff = (row["dropoff_latitude"], row["dropoff_longitude"])
    return geodesic(pickup, dropoff).km
data["distance_km"] = data.apply(calculate_distance, axis=1)
data = data[data["distance_km"] < 100] # remove extreme long trips
```

```

X = data[["distance_km"]]
y = data["fare_amount"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
print("\nData Ready for Training:")
print("Training Samples:", len(X_train))
print("Testing Samples:", len(X_test))
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_pred = lr_model.predict(X_test)

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

print("\n--- Model Evaluation ---")
lr_r2 = r2_score(y_test, lr_pred)
lr_rmse = np.sqrt(mean_squared_error(y_test, lr_pred))
print(f"Linear Regression -> R2: {lr_r2:.4f}, RMSE: {lr_rmse:.4f}")

rf_r2 = r2_score(y_test, rf_pred)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))
print(f"Random Forest -> R2: {rf_r2:.4f}, RMSE: {rf_rmse:.4f}")
if rf_r2 > lr_r2:
    print("\nConclusion:  Random Forest performs better.")
else:
    print("\nConclusion:  Linear Regression performs better.")

results = pd.DataFrame({
    "Actual": y_test.values[:10],
    "LR_Predicted": lr_pred[:10],
    "RF_Predicted": rf_pred[:10]
})
print("\nSample Predictions:")
print(results)

```

## Output:

Original Dataset Shape: (200000, 9)

	Unnamed: 0	key	...	dropoff_latitude	passenger_count
0	24238194	2015-05-07 19:52:06.0000003	...	40.723217	1

```
1 27835199 2009-07-17 20:04:56.0000002 ... 40.750325 1
2 44984355 2009-08-24 21:45:00.00000061 ... 40.772647 1
3 25894730 2009-06-26 08:22:21.0000001 ... 40.803349 3
4 17610152 2014-08-28 17:47:00.000000188 ... 40.761247 5
```

[5 rows x 9 columns]

Data Ready for Training:

Training Samples: 159605

Testing Samples: 39902

--- Model Evaluation ---

Linear Regression -> R2: 0.6078, RMSE: 6.3253

Random Forest -> R2: 0.5972, RMSE: 6.4103

Conclusion:  Linear Regression performs better.

Sample Predictions:

	Actual	LR_Predicted	RF_Predicted
0	11.3	4.331917	11.535748
1	9.3	8.224941	7.377000
2	45.0	50.388414	56.449600
3	17.5	14.334062	12.444000
4	5.0	6.568458	4.129000
5	12.1	15.512883	15.350000
6	5.7	6.008335	4.219000
7	5.7	6.753152	5.447000
8	7.3	4.331917	11.535748
9	10.9	8.451748	8.583000

## ASSIGNMENT – 7

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** Email Spam Classification (KNN and SVM)

**Aim:**

Classify the email using the binary classification method. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

**Code:**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score

# --- Load Dataset ---
df = pd.read_csv("emails.csv")

print("✅ Dataset loaded successfully!")
print("Shape:", df.shape)
print(df.head())

# --- Separate features (X) and labels (y) ---
# All columns except 'Prediction' are features
X = df.drop(columns=['Prediction'])
y = df['Prediction']

# Drop non-numeric or irrelevant columns if any (like 'Email No.')
if 'Email No.' in X.columns:
    X = X.drop(columns=['Email No.'])

# --- Split Dataset ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)
```

```

print("\n--- Model Training and Performance Analysis ---")

# --- K-Nearest Neighbors (KNN) ---
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_pred = knn_model.predict(X_test)

# --- Support Vector Machine (SVM) ---
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)

# --- Evaluation Function ---
def evaluate_classifier(y_true, y_pred, model_name):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    cm = confusion_matrix(y_true, y_pred)

    print(f"\n--- {model_name} Performance ---")
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall: {rec:.4f}")
    print(f"F1-score: {f1:.4f}")
    print(f"Confusion Matrix:\n{cm}")
    return acc, f1

# --- Evaluate Both Models ---
knn_acc, knn_f1 = evaluate_classifier(y_test, knn_pred, "K-Nearest Neighbors")
svm_acc, svm_f1 = evaluate_classifier(y_test, svm_pred, "Support Vector Machine")

# --- Comparison ---
if svm_acc > knn_acc:
    print("\n✓ Conclusion: Support Vector Machine performs better overall.")
else:
    print("\n✓ Conclusion: K-Nearest Neighbors performs better overall.")

print("\nAnalysis complete — compare Accuracy and F1-score to confirm.")

```

## Output:

Dataset loaded successfully!

Shape: (5172, 3002)

	Email	No.	the	to	ect	and	...	military	allowing	ff	dry	Prediction
0	Email 1	0	0	1	0	...	0	0	0	0	0	0
1	Email 2	8	13	24	6	...	0	0	1	0	0	0
2	Email 3	0	0	1	0	...	0	0	0	0	0	0
3	Email 4	0	5	22	0	...	0	0	0	0	0	0
4	Email 5	7	6	17	1	...	0	0	1	0	0	0

[5 rows x 3002 columns]

--- Model Training and Performance Analysis ---

--- K-Nearest Neighbors Performance ---

Accuracy: 0.8677

Precision: 0.7361

Recall: 0.8480

F1-score: 0.7881

Confusion Matrix:

[[804 114]

[ 57 318]]

--- Support Vector Machine Performance ---

Accuracy: 0.9644

Precision: 0.9318

Recall: 0.9467

F1-score: 0.9392

Confusion Matrix:

[[892 26]

[ 20 355]]

Conclusion: Support Vector Machine performs better overall.

Analysis complete — compare Accuracy and F1-score to confirm.

## ASSIGNMENT – 8

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** Bank Customer Churn Prediction (Neural Network)

**Aim:**

Build a neural network-based classifier that can determine whether a bank customer will leave or not in the next 6 months.

**Code:**

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppress TensorFlow INFO/WARNING

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input

# --- Load Dataset ---
df = pd.read_csv("Churn_Modelling.csv")

# --- Preprocessing ---
df = df.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
df['Gender'] = LabelEncoder().fit_transform(df['Gender'])
df = pd.get_dummies(df, columns=['Geography'], drop_first=True)

X = df.drop(columns=['Exited']).values
y = df['Exited'].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```

# --- Neural Network ---
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(12, activation='relu'),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train model silently
model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=0, validation_split=0.2)

# Predict
nn_pred = (model.predict(X_test) > 0.5).astype(int)

# Evaluate
acc = accuracy_score(y_test, nn_pred)
cm = confusion_matrix(y_test, nn_pred)

# --- Final Clean Output ---
print("--- Neural Network Performance ---")
print(f"Accuracy Score: {acc:.4f}")
print("Confusion Matrix:")
print(cm)
print("Points of Improvement: Adam, Learning Rate, Layer Size, Early Stopping, Regularization, Hyperparameter Tuning.")

```

## Output:

63/63 ━━━━━━━━━━ 0s 2ms/step

--- Neural Network Performance ---

Accuracy Score: 0.8280

Confusion Matrix:

```

[[1527  66]
 [ 278 129]]

```

Points of Improvement: Adam, Learning Rate, Layer Size, Early Stopping, Regularization, Hyperparameter Tuning.

## ASSIGNMENT – 9

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** K-Nearest Neighbors (KNN)

**Aim:**

Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset

**Code:**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score

# --- Load the dataset ---
df = pd.read_csv("diabetes.csv") # Make sure 'diabetes.csv' is in the same folder

# --- Features and Target ---
X = df.drop(columns=['Outcome']).values # All columns except target
y = df['Outcome'].values # Target column

# --- Feature Scaling ---
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# --- Train-Test Split ---
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)

# --- K-Nearest Neighbors ---
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# --- Evaluation Metrics ---
acc = accuracy_score(y_test, y_pred)
```

```
err_rate = 1 - acc
prec = precision_score(y_test, y_pred, zero_division=0)
rec = recall_score(y_test, y_pred, zero_division=0)
cm = confusion_matrix(y_test, y_pred)

# --- Clean Output ---
print("--- K-Nearest Neighbors Performance on Diabetes Dataset ---")
print(f"Accuracy: {acc:.4f}")
print(f"Error Rate: {err_rate:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall (Sensitivity): {rec:.4f}\n")
print("Confusion Matrix:")
print(cm)
```

## Output:

--- K-Nearest Neighbors Performance on Diabetes Dataset ---

Accuracy: 0.7143

Error Rate: 0.2857

Precision: 0.6154

Recall (Sensitivity): 0.4938

Confusion Matrix:

`[[125 25]`

`[ 41 40]]`

## ASSIGNMENT – 10

**Name:** Pooja Sunil Pote

**Div:** BE B , **Roll no.** BECB164

**Title:** K-Means Clustering (Elbow Method)

**Aim:**

Implement K-Means clustering/hierarchical clustering on sales\_data\_sample.csv dataset.  
Determine the number of clusters using the elbow method.

**Code:**

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# --- Load the dataset with proper encoding ---
df = pd.read_csv("sales_data_sample.csv", encoding='latin1')

print("✓ Dataset loaded successfully!")
print("Shape:", df.shape)
print(df.head())

# --- Automatically select all numerical columns for clustering ---
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
print("\nNumerical columns selected for clustering:", numerical_cols)

# --- Preprocessing: Scale numerical features ---
X = df[numerical_cols].values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# --- Elbow Method to determine optimal K ---
wcss = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42, n_init='auto')
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# --- K-Means Clustering (Choose optimal K, e.g., K=3) ---
```

```

optimal_k = 3
kmeans_model = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42,
n_init='auto')
df['Cluster'] = kmeans_model.fit_predict(X_scaled)

# --- Clean Output ---
print("\n--- K-Means Clustering Implementation ---")
print(f"Optimal Number of Clusters (Determined by Elbow Method): {optimal_k}\n")
print("First 5 rows with assigned cluster labels:")
print(df.head())
print(f"\nWCSS for K={optimal_k}: {wcss[optimal_k-1]:.2f}")
print("\nAnalysis: The elbow method suggests the optimal K is at the point where the WCSS curve bends sharply, indicating distinct clusters in the data.")
print("Points of Improvement: Try feature selection, scaling methods, different K values, or clustering algorithms like DBSCAN or hierarchical clustering.")

```

## Output:

Dataset loaded successfully!

Shape: (2823, 25)

	ORDERNUMBER	QUANTITYORDERED	... CONTACTFIRSTNAME	DEALSIZE
0	10107	30 ...	Kwai	Small
1	10121	34 ...	Paul	Small
2	10134	41 ...	Daniel	Medium
3	10145	45 ...	Julie	Medium
4	10159	49 ...	Julie	Medium

[5 rows x 25 columns]

Numerical columns selected for clustering: ['ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER', 'SALES', 'QTR\_ID', 'MONTH\_ID', 'YEAR\_ID', 'MSRP']

--- K-Means Clustering Implementation ---

Optimal Number of Clusters (Determined by Elbow Method): 3

First 5 rows with assigned cluster labels:

ORDERNUMBER QUANTITYORDERED PRICEEACH ... CONTACTFIRSTNAME  
DEALSIZE Cluster

0	10107	30	95.70	...	Kwai	Small	1
1	10121	34	81.35	...	Paul	Small	2
2	10134	41	94.74	...	Daniel	Medium	1
3	10145	45	83.26	...	Julie	Medium	1
4	10159	49	100.00	...	Julie	Medium	1

[5 rows x 26 columns]

WCSS for K=3: 16909.36

Analysis: The elbow method suggests the optimal K is at the point where the WCSS curve bends sharply, indicating distinct clusters in the data.

Points of Improvement: Try feature selection, scaling methods, different K values, or clustering algorithms like DBSCAN or hierarchical clustering.