1) Assignment on Linear Regression

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import mpl_toolkits

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

import datetime as dt

%matplotlib inline


data = pd.read_csv("house_data.csv")

data.head()

pd.DataFrame(data.isna().sum()).T

data.drop(['id','date'],axis=1,inplace=True)

#converting built year to actual age of the house data['built_age'] = 2021 - data.yr_built
data.drop('yr_built',axis=1,inplace=True)

sns.countplot(data.bedrooms,order=data.bedrooms.value_counts().index); plt.title("Bedrooms count");

plt.figure(figsize=(12,6)) sns.countplot(data.bathrooms,order=data.bathrooms.value_counts().index);
plt.title("Bathrooms count");

X = list(data.iloc[:,1:].values) #independent

y = data.price.values #dependent

#scaling X values

sn = StandardScaler()

X = sn.fit_transform(X)

#normalizing y values

sns.distplot(y);

plt.xticks(rotation=90);
```

```
plt.title("Before normalizing the dependent variable");

y = np.log10(y)

sns.distplot(y);

plt.xticks(rotation=90);

plt.title("After normalizing the dependent variable");

#train test split X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state)

lr = LinearRegression(normalize=True,fit_intercept=True,n_jobs=1)

lr.fit(X_train,y_train)

y_pred = lr.predict(X_test)

lr.score(X_test,y_test)

lr.intercept_

lr.coef_

#plotting test labels and predicted labels

plt.scatter(y_test,y_pred)

plt.tight_layout()

plt.show(
```

2) Assignment on Decision Tree Classifier

```
import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.tree import export_graphviz

from IPython.display import Image

from sklearn.compose import make_column_transformer

from sklearn.model_selection import train_test_split

from  sklearn.metrics  import accuracy_score

from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns

#loading titanic dataset

df = pd.read_csv("titanic.csv")

df

df.describe()

#dropping unnecessary values such as PassengerID, Name and Ticket

drop_elements = ['PassengerId', 'Name', 'Ticket']

df = df.drop(drop_elements, axis = 1)

#checking null values in the dataset

df.isnull().sum()

#filling age and embarked null values

cols = ['Pclass', 'Sex']

age_class_sex = df.groupby(cols)['Age'].mean().reset_index()

df['Age'] = df['Age'].fillna(df[cols].reset_index().merge(age_class_sex, how='left',
on=cols).set_index('index')['Age'])
```

```
df['Embarked'] = df['Embarked'].fillna('S')
#converting data attributes into categorial numerical form
df['Cabin'] = df["Cabin"].apply(lambda x: 0 if type(x) == float else 1)
df['Embarked'] = df['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)
df['Sex'] = df['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

df.loc[ df['Fare'] <= 7.91, 'Fare'] = 0
df.loc[(df['Fare'] > 7.91) & (df['Fare'] <= 14.454), 'Fare'] = 1
df.loc[(df['Fare'] > 14.454) & (df['Fare'] <= 31), 'Fare']   = 2
df.loc[ df['Fare'] > 31, 'Fare'] = 3
df['Fare'] = df['Fare'].astype(int)

df.loc[ df['Age'] <= 16, 'Age'] = 0
df.loc[(df['Age'] > 16) & (df['Age'] <= 32), 'Age'] = 1
df.loc[(df['Age'] > 32) & (df['Age'] <= 48), 'Age'] = 2
df.loc[(df['Age'] > 48) & (df['Age'] <= 64), 'Age'] = 3
df.loc[ df['Age'] > 64, 'Age'] = 4;
df['Age'] = df['Age'].astype(int)
df
y = df['Survived']
x = df.drop(['Survived'], axis=1).values
x_features = df.iloc[:,1:]
```

In [57]:

```
#split data into train and test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=1)
```

In [63]:

```
#train the decision tree classifier
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
```

y_pred = dt.predict(x_test)

In [65]:

print("Accuracy:",accuracy_score(y_test,y_pred))

Accuracy: 0.7661016949152543

In [67]:

#comparision of Actual and Predicted values

res = pd.DataFrame(list(zip(y_test, y_pred)), columns =['Actual', 'Predicted'])

res.head(100)

#Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(8,6))

sns.heatmap(conf_matrix,annot=**True**,cbar=**True**)

plt.ylabel('True Label')

plt.xlabel('Predicted Label')

plt.title('Confusion Matrix')

conf_matrix

export_graphviz(dt,out_file="data.dot",feature_names=x_features.columns,class_names=["Survived","Died"])

!dot -Tpng data.dot -o tree1.png

Image("tree1.png")

3) Assignment on k-NN Classification

```
import numpy as np

import pandas as pd

from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import preprocessing

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

In [21]:

df = pd.read_csv('iris.csv')

df

df['Species'].value_counts()

df.isnull().sum()

df.pop('Id')

df.head()

X = df.iloc[:,:4]

X = preprocessing.StandardScaler().fit_transform(X)

y = df.iloc[:,-1:]

In [27]:

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)

In [28]:

knnmodel = KNeighborsClassifier(n_neighbors = 3)

knnmodel.fit(X_train,y_train)

y_pred = knnmodel.predict(X_test)

from sklearn.metrics import accuracy_score

acc = accuracy_score(y_test,y_pred)

In [30]:

print('Accuracy',acc)
```

```
conf_matrix = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(8,6))

sns.heatmap(conf_matrix,annot=True,cbar=True)

plt.ylabel('True Label')

plt.xlabel('Predicted Label')

plt.title('Confusion Matrix')

conf_matrix
```

4) Assignment on K-Means Clustering

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
```

In [36]:

```
dataset = pd.read_csv('iris.csv')
x = dataset.iloc[:, [0, 1, 2, 3]].values
```

In [37]:

```
#Finding the optimum number of clusters for k-means classification
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
plt.scatter(x[:,0], x[:,1], c = dataset.iloc[:,4].astype('category').cat.codes, cmap='gist_rainbow')
plt.xlabel('Spea1 Length', fontsize=18)
plt.ylabel('Sepal Width', fontsize=18)
#Visualising the clusters
```

```
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')

plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolour')

plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')


#Plotting the centroids of the clusters

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label = 
'Centroids')

plt.xlabel('Spea1 Length', fontsize=18)

plt.ylabel('Sepal Width', fontsize=18)

plt.legend()
```

Ics1- Implementation of S-DES

```
P10 = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)
P8 = (6, 3, 7, 4, 8, 5, 10, 9)
P4 = (2, 4, 3, 1)

IP = (2, 6, 3, 1, 4, 8, 5, 7)
IPi = (4, 1, 3, 5, 7, 2, 8, 6)

E = (4, 1, 2, 3, 2, 3, 4, 1)

S0 = [
        [1, 0, 3, 2],
        [3, 2, 1, 0],
        [0, 2, 1, 3],
        [3, 1, 3, 2]
     ]

S1 = [
        [0, 1, 2, 3],
        [2, 0, 1, 3],
        [3, 0, 1, 0],
        [2, 1, 0, 3]
     ]

def permutation(pattern, key):
    permuted = ""
```

```
    for i in pattern:
        permuted += key[i-1]

    return permuted

def generate_first(left, right):
    left = left[1:] + left[:1]
    right = right[1:] + right[:1]
    key = left + right

    return permutation(P8, key)

def generate_second(left, right):
    left = left[3:] + left[:3]
    right = right[3:] + right[:3]
    key = left + right

    return permutation(P8, key)

def transform(right, key):
    extended = permutation(E, right)
    xor_cipher = bin(int(extended, 2) ^ int(key, 2))[2:].zfill(8)
    xor_left = xor_cipher[:4]
    xor_right = xor_cipher[4:]

    new_left = Sbox(xor_left, S0)
    new_right = Sbox(xor_right, S1)

    return permutation(P4, new_left + new_right)
```

```python
def Sbox(data, box):
    row = int(data[0] + data[3], 2)
    column = int(data[1] + data[2], 2)

    return bin(box[row][column])[2:].zfill(4)

def encrypt(left, right, key):
    cipher = int(left, 2) ^ int(transform(right, key), 2)

    return right, bin(cipher)[2:].zfill(4)

key = input("Enter a 10-bit key: ")
if len(key) != 10:
    raise Exception("Check the input")

plaintext = input("Enter 8-bit plaintext: ")
if len(plaintext) != 8:
    raise Exception("Check the input")

p10key = permutation(P10, key)
print("First Permutation")
print(p10key)
left_key = p10key[:len(p10key)//2]
print("Left key",left_key)
right_key = p10key[len(p10key)//2:]
print("Right key",right_key)
first_key = generate_first(left_key, right_key)
print("*****")
```

```
print("First key")
print(first_key)
second_key = generate_second(left_key, right_key)
print("*****")
print("Second key")
print(second_key)
initial_permutation = permutation(IP, plaintext)
print("Initial Permutation",initial_permutation)
left_data = initial_permutation[:len(initial_permutation)//2]
right_data = initial_permutation[len(initial_permutation)//2:]
left, right = encrypt(left_data, right_data, first_key)
left, right = encrypt(left, right, second_key)

print("Ciphertext:", permutation(IPi, left + right))
```

```
Enter a 10-bit key: 1010101010
Enter 8-bit plaintext: 10101010
First Permutation
1101001100
Left key 11010
Right key 01100
*****
First key
11100100
****
Second key
01010011
Initial Permutation 00110011
Ciphertext: 10101011
```

Java-

```
import java.util.Arrays;

import java.util.Scanner;

import java.util.*;

import java.util.regex.*;


public class SDES {


        private static final int[] P10 = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6 };

        private static final int[] P8 = { 6, 3, 7, 4, 8, 5, 10, 9 };

        private static final int[] IP = { 2, 6, 3, 1, 4, 8, 5, 7 };

        private static final int[] IP_INV = { 4, 1, 3, 5, 7, 2, 8, 6};

        private static final int[] EP = { 4, 1, 2, 3, 2, 3, 4, 1 };

        private static final int[] P4 = { 2,4,3,1 };

        private static final int[][] SBOX0 = { { 1,0,3,2 }, { 3,2,1,0 }, { 0,2,1,3 }, { 3,1,3,2 } };

        private static final int[][] SBOX1 = { { 0,1,2,3 }, { 2,0,1,3 }, { 3,0,1,0 }, { 2,1,0,3 } };


        private static int[] key1 = null;

        private static int[] key2 = null;


        public SDES(String key) {

                generateKeys(key);

        }


        private int[] permute(int key[], int type[]) {

                int res[] = new int[type.length];

                for(int i = 0 ; i < type.length ; i++) {

                        res[i] = key[type[i]-1];

                }
```

```
        return res;
}

private void shiftByOne(int key[]) {
        int temp = key[0];
        for(int i = 0 ; i < key.length-1 ; i++) {
                key[i] = key[i+1];
        }
        key[key.length-1] = temp;
}

private void shift(int[] key, int shiftSize) {
        for(int i = 0 ; i < shiftSize ; i++) {
                shiftByOne(key);
        }
}

private void generateKeys(String inputKey) {
        //converting string key into int array
        int[] key = new int[inputKey.length()];
        for(int i = 0 ; i < inputKey.length() ; i++) {
                key[i] = Integer.parseInt(String.valueOf(inputKey.charAt(i)));
        }
        //p10 permute
        key = permute(key, P10);

        //dividing the key into two halves - left and right half
        int leftHalf[] = new int[5];
        int rightHalf[] = new int[5];
```

```java
        System.arraycopy(key, 0, leftHalf, 0, 5);
        System.arraycopy(key, 5, rightHalf, 0, 5);

        //shifting the left and right halfs by 1
        shift(leftHalf, 1);
        shift(rightHalf, 1);

        //merging the left and right halfs
        System.arraycopy(leftHalf, 0, key, 0, 5);
        System.arraycopy(rightHalf, 0, key, 5, 5);

        //p8 permute and store the result into key1
        key1 = permute(key, P8);

        //shift the left and right half by 2
        shift(leftHalf, 2);
        shift(rightHalf, 2);

        System.arraycopy(leftHalf, 0, key, 0, 5);
        System.arraycopy(rightHalf, 0, key, 5, 5);

        //p8 permute and store the result into key2
        key2 = permute(key, P8);

        //generated keys
        System.out.println("Key1: "+Arrays.toString(key1));
        System.out.println("Key2: "+Arrays.toString(key2));
    }
```

```
private int binToDec(int a, int b) {
	if(a == 0 && b == 0) return 0;
	else if(a == 0 && b == 1) return 1;
	else if(a == 1 && b == 0) return 2;
	else return 3;
}

private int[] DecToBin(int num) {
	if(num == 0) return new int[] {0,0};
	else if(num == 1) return new int[] {0,1};
	else if(num == 2) return new int[] {1,0};
	else return new int[] {1,1};
}

private int[] getSBoxResult(int subBlock[], int[][] sbox) {
	int rowNo = binToDec(subBlock[0],subBlock[3]);
	int columnNo = binToDec(subBlock[1],subBlock[2]);
	return DecToBin(sbox[rowNo][columnNo]);
}

private int[] fk(int[] rBlock, int key[], int[] lBlock) {
	//expand and permute the right blocks
	int expandedSubBlock[] = permute(rBlock, EP);
	System.out.println("EP res: "+Arrays.toString(expandedSubBlock));

	//XOR the EP res with key1
	for(int i = 0 ; i < expandedSubBlock.length ; i++) {
		expandedSubBlock[i] = expandedSubBlock[i] ^ key[i];
	}
```

```
System.out.println("Res after XOR with key: "+Arrays.toString(expandedSubBlock));

int left[] = new int[4];
int right[] = new int[4];
System.arraycopy(expandedSubBlock, 0, left, 0, 4);
System.arraycopy(expandedSubBlock, 4, right, 0, 4);

//pass left block to SBOX0
int sBox0Res[] = getSBoxResult(left, SBOX0);
System.out.println("SBox0 res: "+Arrays.toString(sBox0Res));

//pass right block to SBOX1
int sBox1Res[] = getSBoxResult(right, SBOX1);
System.out.println("SBox1 res: "+Arrays.toString(sBox1Res));

int combineSBoxesRes[] = new int[4];
System.arraycopy(sBox0Res, 0, combineSBoxesRes, 0, 2);
System.arraycopy(sBox1Res, 0, combineSBoxesRes, 2, 2);
System.out.println("SBoxes combine res: "+Arrays.toString(combineSBoxesRes));

//p4 permutation
int P4PermRes[] =  permute(combineSBoxesRes, P4);
System.out.println("Result after P4 Permutation: "+Arrays.toString(P4PermRes));

//XOR P4 Perm result with left block
for(int i = 0 ; i < lBlock.length ; i++) {
        P4PermRes[i] = P4PermRes[i] ^ lBlock[i];
}
```

```
                System.out.println("Result after XOR with left half: "+Arrays.toString(P4PermRes));
                return P4PermRes;
        }

        private String convertStringToBinary(String input) {
    StringBuilder result = new StringBuilder();
    char[] chars = input.toCharArray();
    for (char aChar : chars) {
      result.append(
          String.format("%8s", Integer.toBinaryString(aChar))
              .replaceAll(" ", "0")
      );
    }
    return result.toString();
}

        private String convertBinaryToString(String input) {
                StringBuilder sb = new StringBuilder();
                Pattern p = Pattern.compile("[\\w ]{0,8}");
    Matcher m = p.matcher(input);
    while (m.find()) {
                        if(!m.group().isEmpty())
        sb.append(new Character((char)Integer.parseInt(m.group(), 2)).toString());
    }
                return sb.toString();
        }

private List<String> convertBinaryIntoBlocks(String binary, int blockSize, String separator) {
```

```
    List<String> result = new ArrayList<>();
    int index = 0;
    while (index < binary.length()) {
      result.add(binary.substring(index, Math.min(index + blockSize, binary.length())));
      index += blockSize;
    }

    return result;
  }

        public String encrypt(String input, String type) {
                //type - BINARY FORM | TEXT MESSAGE FORM
                List<String> blocks;
                StringBuilder builder = new StringBuilder();

                if(type.equals("TEXT_MESSAGE_FORM")) {
                        blocks = convertBinaryIntoBlocks(convertStringToBinary(input), 8, " ");
                }
                else {
                        blocks = new ArrayList<>();
                        blocks.add(input);
                }

                for(int i = 0 ; i < blocks.size() ; i++) {
                        String block = blocks.get(i);
                        //create 8bit msg block
                        int[] msgBlock = new int[8];
                        for(int j = 0 ; j < 8 ; j++) {
                                msgBlock[j] = Integer.parseInt(String.valueOf(block.charAt(j)));
```

```
}

//initial permutation
msgBlock = permute(msgBlock, IP);
System.out.println("Initial Permutation Res: "+Arrays.toString(msgBlock));

//divide 8bit msgblock into left and right halfs
int leftHalf[] = new int[4];
int rightHalf[] = new int[4];
System.arraycopy(msgBlock, 0, leftHalf, 0, 4);
System.arraycopy(msgBlock, 4, rightHalf, 0, 4);
System.out.println("left block: "+Arrays.toString(leftHalf));
System.out.println("right block: "+Arrays.toString(rightHalf));

System.out.println("\n***** First Round - fk1 *****");
//first round - fk function with key1
int fk1Res[] = fk(rightHalf, key1, leftHalf);
System.out.println("fk1 Result: "+Arrays.toString(fk1Res));

//swap function
leftHalf = rightHalf;
rightHalf = fk1Res;

System.out.println("\n***** Second Round - fk2 *****");
//second round - fk function with key2
int fk2Res[] = fk(fk1Res,key2,leftHalf);
System.out.println("fk2 Result: "+Arrays.toString(fk2Res));

int res[] = new int[8];
```

```
			System.arraycopy(fk2Res, 0, res, 0, 4);
			System.arraycopy(rightHalf, 0, res, 4, 4);
			System.out.println("\nResult before IP inverse: "+Arrays.toString(res));

			//inverse initial permutation
			int encrypedRes[] = permute(res, IP_INV);

			for(int ele: encrypedRes) {
				builder.append(ele);
			}
		}

		//return the encrypted message
		return builder.toString();
	}

	public String decrypt(String encryptedInput, String type) {
		StringBuilder builder = new StringBuilder();
		List<String> blocks = convertBinaryIntoBlocks(encryptedInput, 8, " ");
		//System.out.println("blocks:"+blocks.toString());

		for(int i = 0 ; i < blocks.size() ; i++) {
			String block = blocks.get(i);
			//create 8bit msg block
			int[] msgBlock = new int[8];
			for(int j = 0 ; j < 8 ; j++) {
				msgBlock[j] = Integer.parseInt(String.valueOf(block.charAt(j)));
			}
			//System.out.println("Msg block: "+Arrays.toString(msgBlock));
```

```
//initial permutation
msgBlock = permute(msgBlock, IP);
System.out.println("Initial Permutation Res: "+Arrays.toString(msgBlock));

//divide 8bit msgblock into left and right halfs
int leftHalf[] = new int[4];
int rightHalf[] = new int[4];
System.arraycopy(msgBlock, 0, leftHalf, 0, 4);
System.arraycopy(msgBlock, 4, rightHalf, 0, 4);
System.out.println("left block: "+Arrays.toString(leftHalf));
System.out.println("right block: "+Arrays.toString(rightHalf));

System.out.println("\n***** First Round - fk1 *****");
//first round - fk function with key1
int fk1Res[] = fk(rightHalf, key2, leftHalf);
System.out.println("fk1 Result: "+Arrays.toString(fk1Res));

//swap function
leftHalf = rightHalf;
rightHalf = fk1Res;

System.out.println("\n***** Second Round - fk2 *****");
//second round - fk function with key2
int fk2Res[] = fk(fk1Res,key1,leftHalf);
System.out.println("fk2 Result: "+Arrays.toString(fk2Res));

int res[] = new int[8];
System.arraycopy(fk2Res, 0, res, 0, 4);
```

```
			System.arraycopy(rightHalf, 0, res, 4, 4);
			System.out.println("\nResult before IP inverse: "+Arrays.toString(res));

			//inverse initial permutation
			int decryptedRes[] = permute(res, IP_INV);

			for(int ele: decryptedRes) {
				builder.append(ele);
			}
		}
		//return the decrypted message
		if(type.equals("TEXT_MESSAGE_FORM"))
			return convertBinaryToString(builder.toString());
		else
			return builder.toString();
	}

	public static void main(String[] args) {
		String key = null, msg = null, type = null;
		int typeInput = -1;
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter 10-bit key: ");
		key = sc.next();

		System.out.print("Enter plain text for encryption: ");
		msg = sc.nextLine() + sc.nextLine();

		System.out.println("Text message is in which form? ");
		System.out.println("1 - For binary form");
```

```
            System.out.println("2 - For Text message form");
            typeInput = sc.nextInt();
            if(typeInput == 1) type = "BINARY_FORM";
            else if(typeInput == 2)type = "TEXT_MESSAGE_FORM";

            SDES sdes = new SDES(key);

            System.out.println("\n***** ENCRYPTION *****");
            String encryptedMsg = sdes.encrypt(msg, type);
            System.out.println("Encrypted Message: "+encryptedMsg);

            System.out.println("\n***** DECRYPTION *****");
            String decryptedMsg = sdes.decrypt(encryptedMsg, type);
            System.out.println("Decrypted Message: "+decryptedMsg);
        }
}
```

Ics2-Implementation of S-AES

Java-

```
import java.util.Arrays;

import java.util.Scanner;

import java.util.*;

import java.util.regex.*;


public class SimplifiedAdvancedEncryptionStandard {


	private static final String[][] SBOX = {
{"1001","0100","1010","1011"},{"1101","0001","1000","0101"},{"0110","0010","0000","0011"},{"1100",
"1110","1111","0111"} };

	private static final String[][] SBOX_INV = {
{"1010","0101","1001","1011"},{"0001","0111","1000","1111"},{"0110","0000","0010","0011"},{"1100",
"0100","1101","1110"} };

	private static String key0 = null, key1 = null, key2 = null;

	private static int encryptionConstantMatrix[][] = { {1, 4}, {4, 1} };

	private static int decryptionConstantMatrix[][] = { {9, 2}, {2, 9} };


	public SimplifiedAdvancedEncryptionStandard(String key) {

		generateKeys(key);

	}


	private int binaryToDecimal(String binary) {

		return Integer.parseInt(binary, 2);

	}


	private String decimalToBinary(int decimal, int binaryStringSize) {

		return String.format("%" + binaryStringSize + "s", Integer.toBinaryString(decimal &
0xFF)).replace(' ', '0');
```

```java
        }

        public String stringXOR(String a, String b) {
                StringBuilder sb = new StringBuilder();
                for(int i = 0; i < a.length(); i++) {
                sb.append(a.charAt(i) ^ b.charAt(i));
                }
                return sb.toString();
        }

        // Galois field Multiplication
        private int gfMul(int a, int b) {
    int product = 0; //the product of the multiplication

    while (b > 0) {
      if ((b & 1) != 0) //if b is odd then add the first num i.e a into product result
        product = product ^ a;

      a = a << 1; //double first num

      //if a overflows beyond 4th bit
      if ((a & (1 << 4)) != 0)
        a = a ^ 0b10011; // XOR with irreducible polynomial with high term eliminated

      b = b >> 1; //reduce second num
    }
    return product;
  }
```

```java
	private String nibbleSubstitution(String input, String[][] SBOX) {
		StringBuilder sb = new StringBuilder();
		for(int i = 0 ; i < input.length() / 4 ; i++) {
			String str = input.substring(i*4, (i*4)+4);
	sb.append(SBOX[binaryToDecimal(str.substring(0,2))][binaryToDecimal(str.substring(2,4))]);
		}
		return sb.toString();
	}

	private String shiftRow(String str) {
		// Swap 2nd and 4th nibble
		StringBuilder sb = new StringBuilder();
		sb.append(str.substring(0,4));
		sb.append(str.substring(12, 16));
		sb.append(str.substring(8,12));
		sb.append(str.substring(4,8));
		return sb.toString();
	}

	private String rotateNibble(String word) {
		return word.substring(4,8) + word.substring(0,4);
	}

	private void generateKeys(String key) {
		String w0 = key.substring(0,8);
		String w1 = key.substring(8,16);
		String w2 = stringXOR(stringXOR(w0, "10000000"), nibbleSubstitution(rotateNibble(w1),
SBOX));
```

```
		String w3 = stringXOR(w2, w1);
		String w4 = stringXOR(stringXOR(w2, "00110000"), nibbleSubstitution(rotateNibble(w3), 
SBOX));
		String w5 = stringXOR(w4, w3);

		key0 = w0 + w1;
		key1 = w2 + w3;
		key2 = w4 + w5;
	}

	private String getKeys() {
		StringBuilder sb = new StringBuilder();
		sb.append("Key0: "+key0 + "\n");
		sb.append("Key1: "+key1 + "\n");
		sb.append("Key2: "+key2 + "\n");
		return sb.toString();
	}

	public String encrypt(String plainText) {
		// Round 0 - Add Key
		String roundZeroResult = stringXOR(plainText, key0);
		// Round 1 - Nibble Substitution -> Shift Row -> Mix Columns -> Add Key
		String shiftRowResult = shiftRow(nibbleSubstitution(roundZeroResult, SBOX));
		String matrix[][] = new String[2][2];
		matrix[0][0] = shiftRowResult.substring(0,4);
		matrix[0][1] = shiftRowResult.substring(8,12);
		matrix[1][0] = shiftRowResult.substring(4,8);
		matrix[1][1] = shiftRowResult.substring(12,16);
```

```
		StringBuilder sb = new StringBuilder();
		for(int i = 0 ; i < encryptionConstantMatrix.length ; i++) {
			for(int j = 0 ; j < matrix.length ; j++) {
				String tempResults[] = new String[2];
				for(int k = 0 ; k < 2 ; k++) {
					tempResults[k] = 
decimalToBinary(gfMul(encryptionConstantMatrix[i][k],binaryToDecimal(matrix[k][j])), 4);
				}
				sb.append(stringXOR(tempResults[0], tempResults[1]));
			}
		}
		String res = sb.toString();
		String mixColumnsResult = res.substring(0,4) + res.substring(8,12) + res.substring(4,8) + 
res.substring(12, 16);
		String roundOneResult = stringXOR(mixColumnsResult, key1);
		// Round 2 - Nibble Substitution -> Shift Row -> Add Key
		String roundTwoResult = stringXOR(shiftRow(nibbleSubstitution(roundOneResult, 
SBOX)), key2);
		return roundTwoResult;
	}

	public String decrypt(String cipherText) {
		// Round 0 - Add Key
		String roundZeroResult = stringXOR(cipherText, key2);
		// Round 1 - Shift Row -> Nibble Substitution -> Add Key -> Mix Columns
		String addKeyResult = stringXOR(nibbleSubstitution(shiftRow(roundZeroResult), 
SBOX_INV), key1);
		String matrix[][] = new String[2][2];
		matrix[0][0] = addKeyResult.substring(0,4);
		matrix[0][1] = addKeyResult.substring(8,12);
```

```java
		matrix[1][0] = addKeyResult.substring(4,8);
		matrix[1][1] = addKeyResult.substring(12,16);

		StringBuilder sb = new StringBuilder();
		for(int i = 0 ; i < decryptionConstantMatrix.length ; i++) {
			for(int j = 0 ; j < matrix.length ; j++) {
				String tempResults[] = new String[2];
				for(int k = 0 ; k < 2 ; k++) {
					tempResults[k] =
decimalToBinary(gfMul(decryptionConstantMatrix[i][k],binaryToDecimal(matrix[k][j])), 4);
				}
				sb.append(stringXOR(tempResults[0], tempResults[1]));
			}
		}
		String res = sb.toString();
		String mixColumnsResult = res.substring(0,4) + res.substring(8,12) + res.substring(4,8) +
res.substring(12, 16);
		// Round 2 - Shift Row -> Nibble Substitution -> Add Key
		String roundTwoResult = stringXOR(nibbleSubstitution(shiftRow(mixColumnsResult),
SBOX_INV), key0);
		return roundTwoResult;
	}

	public static void main(String[] args) {
		String key = null, msg = null;
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter 16-bit key: ");
		key = sc.next();

		System.out.print("Enter 16-bit binary form message for encryption: ");
```

```
            msg = sc.next();

            SimplifiedAdvancedEncryptionStandard simplifiedAdvancedEncryptionStandard = new
SimplifiedAdvancedEncryptionStandard(key);
            System.out.println(simplifiedAdvancedEncryptionStandard.getKeys());

            System.out.println("\n***** ENCRYPTION *****");
            String encryptedMsg = simplifiedAdvancedEncryptionStandard.encrypt(msg);
            System.out.println("Encrypted Message: "+encryptedMsg);

            System.out.println("\n***** DECRYPTION *****");
            String decryptedMsg = simplifiedAdvancedEncryptionStandard.decrypt(encryptedMsg);
            System.out.println("Decrypted Message: "+decryptedMsg);
      }
}
```

3) Implementation of Diffie-Hellman key exchange

```python
from random import randint


if __name__ == '__main__':


    # Both the persons will be agreed upon the
    # public keys G and P
    # A prime number P is taken
    P = 23

    # A primitve root for P, G is taken
    G = 9



    print('The Value of P is :%d'%(P))
    print('The Value of G is :%d'%(G))

    # Alice will choose the private key a
    a = 4
    print('The Private Key a for Alice is :%d'%(a))

    # gets the generated key
    x = int(pow(G,a,P))

    # Bob will choose the private key b
    b = 3
    print('The Private Key b for Bob is :%d'%(b))

    # gets the generated key
```

```python
    y = int(pow(G,b,P))



    # Secret key for Alice
    ka = int(pow(y,a,P))

    # Secret key for Bob
    kb = int(pow(x,b,P))

    print('Secret key for the Alice is : %d'%(ka))
    print('Secret Key for the Bob is : %d'%(kb))
```

The Value of P is :23

The Value of G is :9

The Private Key a for Alice is :4

The Private Key b for Bob is :3

Secret key for the Alice is : 9

Secret Key for the Bob is : 9

4) Implementation of RSA

```
import random



'''
Euclid's algorithm for determining the greatest common divisor
Use iteration to make it faster for larger integers
'''



def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a



'''
Euclid's extended algorithm for finding the multiplicative inverse of two numbers
'''



def multiplicative_inverse(e, phi):
    d = 0
    x1 = 0
    x2 = 1
    y1 = 1
    temp_phi = phi

    while e > 0:
```

```
        temp1 = temp_phi//e
        temp2 = temp_phi - temp1 * e
        temp_phi = e
        e = temp2

        x = x2 - temp1 * x1
        y = d - temp1 * y1

        x2 = x1
        x1 = x
        d = y1
        y1 = y

    if temp_phi == 1:
        return d + phi


'''
Tests to see if a number is prime.
'''


def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
```

```
        return False
    return True



def generate_key_pair(p, q):
    if not (is_prime(p) and is_prime(q)):
        raise ValueError('Both numbers must be prime.')
    elif p == q:
        raise ValueError('p and q cannot be equal')
    # n = pq
    n = p * q

    # Phi is the totient of n
    phi = (p-1) * (q-1)

    # Choose an integer e such that e and phi(n) are coprime
    e = random.randrange(1, phi)

    # Use Euclid's Algorithm to verify that e and phi(n) are coprime
    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(1, phi)
        g = gcd(e, phi)

    # Use Extended Euclid's Algorithm to generate the private key
    d = multiplicative_inverse(e, phi)

    # Return public and private key_pair
    # Public key is (e, n) and private key is (d, n)
```

```
    return ((e, n), (d, n))



def encrypt(pk, plaintext):
    # Unpack the key into it's components
    key, n = pk
    # Convert each letter in the plaintext to numbers based on the character using a^b mod m
    cipher = [pow(ord(char), key, n) for char in plaintext]
    # Return the array of bytes
    return cipher



def decrypt(pk, ciphertext):
    # Unpack the key into its components
    key, n = pk
    # Generate the plaintext based on the ciphertext and key using a^b mod m
    aux = [str(pow(char, key, n)) for char in ciphertext]
    # Return the array of bytes as a string
    plain = [chr(int(char2)) for char2 in aux]
    return ''.join(plain)



if __name__ == '__main__':
    '''
    Detect if the script is being run directly by the user
    '''

print("==============================================================================
==========================")
```

```
    print("=================================== RSA Encryptor / Decrypter
===============================================")

    print(" ")


    p = int(input(" - Enter a prime number (17, 19, 23, etc): "))

    q = int(input(" - Enter another prime number (Not one you entered above): "))


    print(" - Generating your public / private key-pairs now . . .")


    public, private = generate_key_pair(p, q)


    print(" - Your public key is ", public, " and your private key is ", private)


    message = input(" - Enter a message to encrypt with your public key: ")

    encrypted_msg = encrypt(public, message)


    print(" - Your encrypted message is: ", ''.join(map(lambda x: str(x), encrypted_msg)))

    print(" - Decrypting message with private key ", private, " . . .")

    print(" - Your message is: ", decrypt(private, encrypted_msg))


    print(" ")

    print("============================================== END
================================================================")

print("=======================================================================================
==============================")

==============================================================================================
========================

=================================== RSA Encryptor / Decrypter
===============================================
```

- Enter a prime number (17, 19, 23, etc): 17

- Enter another prime number (Not one you entered above): 23

- Generating your public / private key-pairs now . . .

- Your public key is (205, 391) and your private key is (261, 391)

- Enter a message to encrypt with your public key: omkarmankar

- Your encrypted message is: 314227241201252227201213241201252

- Decrypting message with private key (261, 391) . . .

- Your message is: omkarmankar

== END
==

==
=====================