Slip 1

```
import java.io.*;

public class UppercaseToLowercaseConverter {

    public static void main(String[] args) {
        try {
            // Create BufferedReader to read from System.in
            BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

            // Create BufferedWriter to write to System.out
            BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(System.out));

            System.out.println("Enter text (Ctrl+C to exit):");

            String line;
            while ((line = reader.readLine()) != null) {
                // Convert the input line to lowercase
                String lowercaseLine = line.toLowerCase();

                // Write the lowercase line to the output
                writer.write(lowercaseLine);
                writer.newLine();
                writer.flush();
            }

            // Close the reader and writer
            reader.close();
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
import numpy as np
from sklearn.decomposition import PCA

# Create a sample dataset
data = np.array([
    [1, 2, 3, 4, 5],
    [2, 3, 4, 5, 6],
    [3, 4, 5, 6, 7]
])

# Initialize a PCA transformer with the number of components you want
pca = PCA(n_components=2)

# Fit the transformer to the data and transform it
transformed_data = pca.fit_transform(data)

# Display the transformed data
print("Original Data:")
print(data)
print("\nTransformed Data (2 Principal Components):")
print(transformed_data)
```

Slip3

```
class factory{
    public static void main(String[] args) {
        OSFactory osf=new OSFactory();
        OS o1=osf.getInt("open");
        o1.show();

        OS o2=osf.getInt("closed");
        o2.show();

        OS o3=osf.getInt(" ");
        o3.show();
    }
}

interface OS{
    void show();
}

class Android implements OS{
    public void show(){
        System.out.println("I am Android");
    }
}
class Ios implements OS{
    public void show(){
        System.out.println("I am Ios");
    }
}
class Windows implements OS{
    public void show(){
        System.out.println("I am Windows");
    }
}

class OSFactory{
    public OS getInt(String str){
        if(str.equalsIgnoreCase("open"))
```

```
            return new Android();
        else if(str.equalsIgnoreCase("closed"))
            return new Ios();
        else
            return new Windows();
    }
}
```

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score


data = pd.read_csv('Housing.csv')

X = data['area'].values.reshape(-1, 1)

y = data['price'].values

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

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

slope = model.coef_[0]
```

```python
intercept = model.intercept_

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f"Linear Regression Equation: price = {slope:.2f} * area +
{intercept:.2f}")

print(f"Mean Squared Error: {mse:.2f}")

print(f"R-squared: {r2:.2f}")

plt.scatter(X_test, y_test, color='blue', label='Actual Data')

plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')

plt.xlabel('Area (sqft)')

plt.ylabel('Price ($)')

plt.title('House Price Prediction')

plt.legend()

plt.show()
```

Slip 4

```java
import java.util.Enumeration;
import java.util.Iterator;

public class EnumDemo {
    public static void main(String[] args) {
        // Create an Enumeration (e.g., from a Vector)
        java.util.Vector<String> vector = new java.util.Vector<>();
        vector.add("Item 1");
        vector.add("Item 2");
        vector.add("Item 3");
        Enumeration<String> enumeration = vector.elements();

        // Use the Enumeration as an Iterator using the Adapter
        Iterator<String> iterator = new EnumerationAdapter<>(enumeration);

        // Iterate and print elements using Iterator
        while (iterator.hasNext()) {
            String item = iterator.next();
            System.out.println(item);
        }
    }
}



// Adapter class to adapt Enumeration to Iterator
class EnumerationAdapter<T> implements Iterator<T> {
    private Enumeration<T> enumeration;

    public EnumerationAdapter(Enumeration<T> enumeration) {
        this.enumeration = enumeration;
    }

    @Override
    public boolean hasNext() {
        return enumeration.hasMoreElements();
    }

    @Override
    public T next() {
        return enumeration.nextElement();
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException("Remove operation is not
supported.");
    }
}
```

Slip 5

```
import java.util.Scanner;


public class TvRemoteControlDemo {
 public static void main(String[] args) {
     // Create the TV and commands
     Tv livingRoomTv = new Tv();
     TvOnCommand tvOn = new TvOnCommand(livingRoomTv);
     TvOffCommand tvOff = new TvOffCommand(livingRoomTv);


     // Create the remote control
     RemoteControl remote = new RemoteControl();

     // Menu-driven program
     Scanner scanner = new Scanner(System.in);
     int choice = 0; // Initialize choice with a default value

     do {
         System.out.println("\nTV Remote Control Menu:");
         System.out.println("1. Turn On TV");
         System.out.println("2. Turn Off TV");

         System.out.println("3. Exit");
         System.out.print("Enter your choice: ");
```

```
            if (scanner.hasNextInt()) {
                choice = scanner.nextInt();
                switch (choice) {
                    case 1:
                        remote.setCommand(tvOn);
                        remote.pressButton();
                        break;
                    case 2:
                        remote.setCommand(tvOff);
                        remote.pressButton();
                        break;

                    case 3:
                        System.out.println("Exiting the TV remote control.");
                        break;
                    default:
                        System.out.println("Invalid choice. Please try again.");
                }
            } else {
                System.out.println("Invalid input. Please enter a number.");
                scanner.next(); // Consume invalid input
            }
        } while (choice != 3);

        scanner.close();
 }
}
//Command interface
interface Command {
 void execute();
}

//Concrete command: Turn on the TV
class TvOnCommand implements Command {
 private Tv tv;

 public TvOnCommand(Tv tv) {
        this.tv = tv;
  }

  @Override
  public void execute() {
        tv.turnOn();
  }
}

//Concrete command: Turn off the TV
class TvOffCommand implements Command {
  private Tv tv;

  public TvOffCommand(Tv tv) {
        this.tv = tv;
  }

  @Override
  public void execute() {
        tv.turnOff();
  }
}



class Tv {

  private boolean isOn = false;

  public void turnOn() {
        isOn = true;
        System.out.println("TV is ON");
  }

  public void turnOff() {
        isOn = false;
        System.out.println("TV is OFF");
```

```
 }


}

//Invoker: Controls the commands
class RemoteControl {
 private Command command;

 public void setCommand(Command command) {
     this.command = command;
 }

 public void pressButton() {
     command.execute();
 }
}
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV data into a DataFrame
data= pd.read_csv("insurance_data.csv")


# Assuming your CSV file contains 'age' and 'bought_insurance' columns
X = data[['age']]
y = data['bought_insurance']

# Split the data into training and testing sets (80% training, 20% testing)
```

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

# Create a logistic regression model
model = LogisticRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not
Bought', 'Bought'], yticklabels=['Not Bought', 'Bought'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Slip 6

```
import java.util.Scanner;
public class CeilingFanTests {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CeilingFan fan = new CeilingFan();
        RemoteControl remote = new RemoteControl(fan);

        System.out.println("Ceiling Fan Test");
        while (true) {
            System.out.println("1. Turn On\n2. Turn Off\n3. Undo\n4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    remote.pressOn();
                    break;
                case 2:
                    remote.pressOff();
                    break;
                case 3:
                    remote.pressUndo();
                    break;
                case 4:
                    System.out.println("Exiting the program.");
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please try again.");
                    break;
            }
        }
    }
}



// Ceiling Fan class
class CeilingFan {
    private boolean isOn;
    public CeilingFan() {
        isOn = false;
    }

    public void turnOn() {
        isOn = true;
        System.out.println("Ceiling fan is turned on.");
    }

    public void turnOff() {
        isOn = false;
        System.out.println("Ceiling fan is turned off.");
    }

    public boolean isOn() {
        return isOn;
    }
}

// Remote Control class
class RemoteControl {
    private CeilingFan ceilingFan;
    private boolean previousState;

    public RemoteControl(CeilingFan fan) {
        ceilingFan = fan;
        previousState = false;
    }

    public void pressOn() {
        previousState = ceilingFan.isOn();
        ceilingFan.turnOn();
    }

    public void pressOff() {
        previousState = ceilingFan.isOn();
        ceilingFan.turnOff();
    }

    public void pressUndo() {
        if (previousState) {
            ceilingFan.turnOn();
        } else {
            ceilingFan.turnOff();
        }
    }
}
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load the CSV data into a DataFrame
data = pd.read_csv('position_sal.csv')

# Assuming your CSV file contains 'position', 'level', and 'salary' columns
X = data[['Level']]
y = data['Salary']

# Create a PolynomialFeatures object to generate polynomial features
poly_features = PolynomialFeatures(degree=4)  # You can adjust the degree as
needed

# Transform the input features into polynomial features
X_poly = poly_features.fit_transform(X)

# Create a linear regression model
model = LinearRegression()
```

```
# Fit the model to the polynomial features
model.fit(X_poly, y)

# Predict salary values using the polynomial regression model
y_pred = model.predict(X_poly)

# Plot the original data and the polynomial regression curve
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', label='Polynomial Regression')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.title('Polynomial Regression')
plt.legend()
plt.show()
```

Slip 7

```
//package GumballMachine;

//Main class
public class GumballMachineTestDrive {
 public static void main(String[] args) {
     GumballMachine gumballMachine = new GumballMachine(5);

     System.out.println(gumballMachine);

     gumballMachine.insertQuarter();
     gumballMachine.turnCrank();

     System.out.println(gumballMachine);

     gumballMachine.insertQuarter();
     gumballMachine.ejectQuarter();
```

```
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);
    }
}


//State interface
interface State {
    void insertQuarter();
    void ejectQuarter();
    void turnCrank();
    void dispense();
}

//Concrete State: NoQuarterState
class NoQuarterState implements State {
    private GumballMachine gumballMachine;

    public NoQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertQuarter() {
        System.out.println("You inserted a quarter.");
        gumballMachine.setState(gumballMachine.getHasQuarterState());
    }

    public void ejectQuarter() {
        System.out.println("You haven't inserted a quarter.");
    }

    public void turnCrank() {
        System.out.println("You turned, but there's no quarter.");
    }

    public void dispense() {
        System.out.println("You need to pay first.");
    }

    public String toString() {
        return "Waiting for quarter";
    }
}

//Concrete State: HasQuarterState
class HasQuarterState implements State {
    private GumballMachine gumballMachine;

    public HasQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertQuarter() {
        System.out.println("You can't insert another quarter.");
    }

    public void ejectQuarter() {
        System.out.println("Quarter returned.");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }
```

```
 public void turnCrank() {
     System.out.println("You turned...");
     gumballMachine.setState(gumballMachine.getSoldState());
 }

 public void dispense() {
     System.out.println("No gumball dispensed.");
 }

 public String toString() {
     return "Waiting for turn of crank";
 }
}

//Concrete State: SoldState
class SoldState implements State {
 private GumballMachine gumballMachine;

 public SoldState(GumballMachine gumballMachine) {
     this.gumballMachine = gumballMachine;
 }

 public void insertQuarter() {
     System.out.println("Please wait, we're already giving you a gumball.");
 }

 public void ejectQuarter() {
     System.out.println("Sorry, you already turned the crank.");
 }

 public void turnCrank() {
     System.out.println("Turning twice doesn't get you another gumball!");
 }

 public void dispense() {
     gumballMachine.releaseGumball();
     if (gumballMachine.getCount() > 0) {
         gumballMachine.setState(gumballMachine.getNoQuarterState());
     } else {
         System.out.println("Oops, out of gumballs!");
         gumballMachine.setState(gumballMachine.getSoldOutState());
     }
 }

 public String toString() {
     return "Dispensing a gumball";
 }
}

//Concrete State: SoldOutState
class SoldOutState implements State {
 private GumballMachine gumballMachine;

 public SoldOutState(GumballMachine gumballMachine) {
     this.gumballMachine = gumballMachine;
 }

 public void insertQuarter() {
     System.out.println("Sorry, gumballs are sold out.");
 }

 public void ejectQuarter() {
     System.out.println("You can't eject, you haven't inserted a quarter yet.");
 }

 public void turnCrank() {
     System.out.println("You turned, but there are no gumballs.");
 }

 public void dispense() {
     System.out.println("No gumball dispensed.");
 }

 public String toString() {
     return "Sold out";
 }
}

//Gumball Machine class
class GumballMachine {
 private State soldOutState;
 private State noQuarterState;
 private State hasQuarterState;
 private State soldState;

 private State state;
 private int count = 0;

 public GumballMachine(int numberOfGumballs) {
     soldOutState = new SoldOutState(this);
     noQuarterState = new NoQuarterState(this);
     hasQuarterState = new HasQuarterState(this);
     soldState = new SoldState(this);

     this.count = numberOfGumballs;
     if (numberOfGumballs > 0) {
         state = noQuarterState;
     } else {
         state = soldOutState;
     }
 }

 public void insertQuarter() {
     state.insertQuarter();
 }

 public void ejectQuarter() {
     state.ejectQuarter();
 }

 public void turnCrank() {
     state.turnCrank();
     state.dispense();
 }

 void setState(State state) {
     this.state = state;
 }

 void releaseGumball() {
     System.out.println("A gumball comes rolling out the slot...");
     if (count != 0) {
         count--;
     }
 }

 int getCount() {
     return count;
 }

 State getSoldOutState() {
     return soldOutState;
 }

 State getNoQuarterState() {
     return noQuarterState;
 }

 State getHasQuarterState() {
     return hasQuarterState;
 }

 State getSoldState() {
     return soldState;
 }

 public String toString() {
     StringBuilder result = new StringBuilder();
     result.append("\nMighty Gumball, Inc.");
     result.append("\nJava-enabled Standing Gumball Model #2023");
     result.append("\nInventory: " + count + " gumball");
     if (count != 1) {
         result.append("s");
     }
     result.append("\n");
     result.append("Machine is " + state + "\n");
     return result.toString();
 }
}
```

....

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
data = {
    'company': [
        'google', 'google', 'google', 'google', 'google', 'google',
        'abc pharma', 'abc pharma', 'abc pharma', 'abc pharma',
        'facebook', 'facebook', 'facebook', 'facebook', 'facebook', 'facebook'
    ],
    'job': [
        'sales executive', 'sales executive', 'business manager', 'business
manager', 'computer programmer', 'computer programmer',
        'sales executive', 'computer programmer', 'business manager', 'business
manager',
        'sales executive', 'sales executive', 'business manager', 'business
manager', 'computer programmer', 'computer programmer'
    ],
    'degree': [
        'bachelors', 'masters', 'bachelors', 'masters', 'bachelors', 'masters',
        'masters', 'bachelors', 'bachelors', 'masters',
        'bachelors', 'masters', 'bachelors', 'masters', 'bachelors', 'masters'
    ],
    'salary_more_then_100k': [0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
}

df = pd.DataFrame(data)

# Encode categorical variables using Label Encoding
label_encoder = LabelEncoder()
df['company'] = label_encoder.fit_transform(df['company'])
df['job'] = label_encoder.fit_transform(df['job'])
df['degree'] = label_encoder.fit_transform(df['degree'])
# Separate the features (X) and the target variable (y)
X = df.drop('salary_more_then_100k', axis=1)
y = df['salary_more_then_100k']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a Decision Tree classification model
decision_tree = DecisionTreeClassifier()

# Fit the model to the training data
decision_tree.fit(X_train, y_train)

# Make predictions on the test data
y_pred = decision_tree.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print the evaluation results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", report)
```

Slip 8

```
//Main class
public class AdapterPatternDemo {
    public static void main(String[] args) {
        Heart heart = new Heart();
        Beat beat = new Beat();
        HeartAdapter adapter = new HeartAdapter(heart);

        // Use the Heart Model as if it were a Beat Model

        int bpm = adapter.getBPM();
        System.out.println("Initial BPM: " + bpm);

        // Simulate changing heart rate data and updating the BPM
        heart.generateHeartRate();
        bpm = adapter.getBPM();
        beat.setBPM(bpm);
    }
}

// Heart Model interface
interface HeartModel {
    int getHeartRate();
}

// Heart Model Implementation
class Heart implements HeartModel {
    private int heartRate;

    public Heart() {
        // Simulate heart rate data (e.g., from a sensor)
        this.heartRate = 0;
    }

    public int getHeartRate() {
        return heartRate;
    }

    // Simulate changing heart rate data
    public void generateHeartRate() {
        heartRate = (int) (Math.random() * 100 + 60); // Generate a random heart
rate between 60 and 160
        System.out.println("Heart rate: " + heartRate + " bpm");
    }
}

// Beat Model interface
interface BeatModel {
    void setBPM(int bpm);

    int getBPM();
}

// Beat Model Implementation
class Beat implements BeatModel {
    private int bpm;

    public void setBPM(int bpm) {
        this.bpm = bpm;
        System.out.println("BPM set to " + bpm);
    }

    public int getBPM() {
        return bpm;
    }
}

// Adapter to adapt Heart Model to Beat Model
class HeartAdapter implements BeatModel {
    private HeartModel heartModel;

    public HeartAdapter(HeartModel heartModel) {
        this.heartModel = heartModel;
    }

    public void setBPM(int bpm) {
        // Heart rate doesn't have a set BPM, so we'll ignore this method
        System.out.println("Cannot set BPM on Heart Model");
    }

    public int getBPM() {
        int heartRate = heartModel.getHeartRate();
        return heartRate;
    }
}
```

………………

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, SVR
from sklearn.metrics import accuracy_score, mean_squared_error

# Classification Example
# Load the Iris dataset for classification
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a Linear SVM classifier
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)
```

```
# Evaluate the classifier's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Classification Accuracy: {accuracy:.2f}")

# Regression Example
# Generate synthetic data for regression
X_reg = np.random.rand(100, 1)
y_reg = 2 * X_reg + 1 + 0.1 * np.random.randn(100, 1)

# Split the regression data into training and testing sets
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_reg, y_reg,
test_size=0.2, random_state=42)

# Create a Linear SVR model
svr = SVR(kernel='linear')
svr.fit(X_train_reg, y_train_reg)

# Make predictions on the test data
y_pred_reg = svr.predict(X_test_reg)

# Evaluate the regression model using Mean Squared Error
mse = mean_squared_error(y_test_reg, y_pred_reg)
print(f"Regression Mean Squared Error: {mse:.4f}")
```

Slip 10

```
class FacadeDP{
    public static void main(String args[]) throws Exception{
        ShapeMaker s=new ShapeMaker();
        s.drawCircle();
        s.drawRectangle();
        s.drawSquare();
    }
}
```

```
interface Shape{
    void draw();
}

class Rectangle implements Shape{
    public void draw(){
        System.out.println("Shape::Rectangle");
    }
}

class Square implements Shape{
    public void draw(){
        System.out.println("Shape::Square");
    }
}

class Circle implements Shape{
    public void draw(){
        System.out.println("Shape::Circle");
    }
}

class ShapeMaker{
    private Shape circle;
    private Shape rectangle;
    private Shape square;

    public ShapeMaker(){
        circle=new Circle();
        rectangle=new Rectangle();
        square=new Square();
    }

    public void drawCircle(){
        circle.draw();
    }
    public void drawRectangle(){
        rectangle.draw();
    }
    public void drawSquare(){
        square.draw();
    }
}
```

....

```
import pandas as pd

from sklearn.datasets import load_iris

iris = load_iris()


iris.feature_names


iris.target_names


df = pd.DataFrame(iris.data,columns=iris.feature_names)

df.head()


df['target'] = iris.target

df.head()


df[df.target==1].head()


df[df.target==2].head()


df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])

df.head()
```

```
df[45:55]


df0 = df[:50]

df1 = df[50:100]

df2 = df[100:]


import matplotlib.pyplot as plt


plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+')

plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')


plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')

plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')


from sklearn.model_selection import train_test_split


X = df.drop(['target','flower_name'], axis='columns')

y = df.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)

len(X_train)

len(X_test)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train,y_train)

print(knn.score(X_test, y_test))

print(knn.predict([[4.8,3.0,1.5,0.3]]))

from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)

import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(7,5))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

plt.show()

from sklearn.metrics import classification_report



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

Slip 11

```
import java.io.*;
import java.util.*;

class AbstractFactory
{
    public static void main(String args[]) throws Exception
    {
        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
        System.out.println("Enter choice Bank or Loan...");
        System.out.println("Enter 1 for Bank");
        System.out.println("Enter 2 for Loan");
        Scanner sc=new Scanner(System.in);
        int var;
        var=sc.nextInt();
        switch(var)
        {
            case 1:
                AbstractFactories bankFactory=FactoryCreator.getFactory("Bank");
                System.out.println("Enter Bank name:");
                String bNm=br.readLine();
                Bank b1=bankFactory.getBank(bNm);
                System.out.println("Your account for "+b1.getBankName()+" is
created");
                break;
            case 2:
                AbstractFactories loanFactory=FactoryCreator.getFactory("Loan");
                System.out.println("Enter Loan name:");
                String lnm=br.readLine();
                Loan l1=loanFactory.getLoan(lnm);
                System.out.println("Enter the loan amount:");
                double loanAmount=Double.parseDouble(br.readLine());
                System.out.println("Enter interest rate:");
                double rate=Double.parseDouble(br.readLine());
                System.out.println("Enter the number of years:");
                int years=Integer.parseInt(br.readLine());
                l1.getInterestRate(rate);
                l1.calculateLoan(loanAmount,years);
                break;
        }
    }
}

interface Bank{
    String getBankName();
}

class ICICIBank implements Bank{
    private final String Bankname;
    ICICIBank(){
        Bankname="ICICIBank";
    }
    public String getBankName(){
        return Bankname;
    }
}

class AxisBank implements Bank{
    private final String Bankname;
    AxisBank(){
        Bankname="AxisBank";
    }
    public String getBankName(){
        return Bankname;
    }
}

class KotakBank implements Bank{
    private final String Bankname;
    KotakBank(){
        Bankname="KotakBank";
    }
    public String getBankName(){
        return Bankname;
    }
}

abstract class Loan{
    protected Double rate;
    abstract void getInterestRate(Double rate);
    public void calculateLoan(Double loanamt,int year){
        Double EMI;
        int num;
        num=year*12;
        rate = rate /1200;

        EMI=((rate*Math.pow((1+rate),num))/((Math.pow((1+rate),num))-1))*loanamt;
        System.out.println("Your Monthly EMI is "+EMI+" for the amount
"+loanamt+" you have borrowed");
    }
}
class HomeLoan extends Loan{
    public void getInterestRate(Double r){
        rate=r;
    }
}

class CarLoan extends Loan{
    public void getInterestRate(Double r){
        rate=r;
    }
```

```
}

class BikeLoan extends Loan{
    public void getInterestRate(Double r){
        rate=r;
    }
}
abstract class AbstractFactories{
    public abstract Bank getBank(String b);
    public abstract Loan getLoan(String l);
}
class BankFactory extends AbstractFactories{
    public Bank getBank(String b){

        if(b==null){
            return null;
        }
        else if(b.equalsIgnoreCase("ICICIBank")){
            return new ICICIBank();
        }
        else if(b.equalsIgnoreCase("AxisBank")){
            return new AxisBank();
        }
        else if(b.equalsIgnoreCase("KotakBank")){
            return new KotakBank();
        }
        return null;
    }
     public Loan getLoan(String loan)
    {
        return null;
    }
}

class LoanFactory extends AbstractFactories{
    public Bank getBank(String loan)
    {
        return null;
    }
    public Loan getLoan(String l){

        if(l==null){
            return null;
        }
        else if(l.equalsIgnoreCase("HomeLoan")){
            return new HomeLoan();
        }
        else if(l.equalsIgnoreCase("CarLoan")){
            return new CarLoan();
        }
        else if(l.equalsIgnoreCase("BikeLoan")){
            return new BikeLoan();
        }
        return null;
    }

}

class FactoryCreator{
    public static AbstractFactories getFactory(String choice){
        if(choice.equalsIgnoreCase("Bank")){
            return new BankFactory();
        }
        else if(choice.equalsIgnoreCase("Loan")){
            return new LoanFactory();
        }
        return null;
    }
}
```

…………………….

```
import pandas as pd
df = pd.read_csv("./titanic.csv")
df.head()
df.drop(['PassengerId','Name','SibSp','Parch','Ticket','Cabin','Embarked'],axis='column
s',inplace=True)
df.head()

inputs = df.drop('Survived',axis='columns')
target = df.Survived

dummies = pd.get_dummies(inputs.Sex)
print(dummies.head(3))

inputs = pd.concat([inputs,dummies],axis='columns')
inputs.head(3)

inputs.drop(['Sex','male'],axis='columns',inplace=True)
inputs.head(3)

inputs.columns[inputs.isna().any()]

inputs.Age[:10]

inputs.Age = inputs.Age.fillna(inputs.Age.mean())
inputs.head()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(inputs,target,test_size=0.3)

from sklearn.naive_bayes import GaussianNB
model = GaussianNB()

model.fit(X_train,y_train)

model.score(X_test,y_test)

X_test[0:10]

y_test[0:10]
```

```
print(model.predict(X_test[0:10]))

print(model.predict_proba(X_test[:10]))

from sklearn.model_selection import cross_val_score
print(cross_val_score(GaussianNB(),X_train, y_train, cv=5))
```

Slip 12

```
import java.util.Scanner;


abstract class Pizza {
    String name;
    String size;

    void prepare() {
        System.out.println("Preparing " + size + " " + name + " pizza");
    }

    void bake() {
        System.out.println("Baking " + size + " " + name + " pizza");
    }

    void cut() {
        System.out.println("Cutting " + size + " " + name + " pizza");
    }

    void box() {
        System.out.println("Boxing " + size + " " + name + " pizza");
    }
}

class NyStyleCheesePizza extends Pizza {
    NyStyleCheesePizza(String size) {
        name = "NY Style Cheese";
        this.size = size;
    }
}

class ChicagoStyleCheesePizza extends Pizza {
    ChicagoStyleCheesePizza(String size) {
        name = "Chicago Style Cheese";
        this.size = size;
    }
}

// PizzaStore with Factory Method
abstract class PizzaStore {
    abstract Pizza createPizza(String type, String size);

    Pizza orderPizza(String type, String size) {
        Pizza pizza = createPizza(type, size);

        if (pizza != null) {
            pizza.prepare();
            pizza.bake();
            pizza.cut();
            pizza.box();
            return pizza;
        } else {
            System.out.println("Sorry, we don't have that pizza type.");
            return null;
        }
    }
}

class NyPizzaStore extends PizzaStore {
    Pizza createPizza(String type, String size) {
        if (type.equals("cheese")) {
            return new NyStyleCheesePizza(size);
        }
        return null;
    }
}

class ChicagoPizzaStore extends PizzaStore {
    Pizza createPizza(String type, String size) {
        if (type.equals("cheese")) {
            return new ChicagoStyleCheesePizza(size);
        }
        return null;
    }
}

public class PizzaMain {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Welcome to the Pizza Store!");
        System.out.println("Select a store (1 for NY Pizza Store, 2 for Chicago
Pizza Store):");
        int storeChoice = scanner.nextInt();

        PizzaStore pizzaStore;

        if (storeChoice == 1) {
            pizzaStore = new NyPizzaStore();
        } else if (storeChoice == 2) {
            pizzaStore = new ChicagoPizzaStore();
        } else {
            System.out.println("Invalid choice. Exiting.");
            return;
        }

        while (true) {
            System.out.println("Menu:");
            System.out.println("1. Pizza");
            System.out.println("2. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            if (choice == 1) {
                System.out.print("Enter pizza size (small, medium, large): ");
                String size = scanner.next().toLowerCase();
                System.out.print("Enter pizza type (cheese): ");
                String pizzaType = scanner.next().toLowerCase();
                Pizza pizza = pizzaStore.orderPizza(pizzaType, size);
                if (pizza != null) {
                    System.out.println("You ordered a " + pizza.size + " " +
pizza.name + " pizza");
                }
            } else if (choice == 2) {
                break;
            } else {
                System.out.println("Invalid choice. Please try again.");
            }
        }

        scanner.close();
    }
}
```

…………………….

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Create a synthetic dataset with three clusters
n_samples = 300
n_features = 2
n_clusters = 3

X, _ = make_blobs(n_samples=n_samples, n_features=n_features,
centers=n_clusters, random_state=42)

# Apply K-Means clustering
```

```
kmeans = KMeans(n_clusters=n_clusters)
print(kmeans.fit(X))

# Get cluster centers and labels
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_


# Visualize the data and cluster centers
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], marker='x', c='red', s=100,
label='Cluster Centers')
plt.legend()
plt.title('K-Means Clustering')
plt.show()
```

Slip 13

```
import java.io.*;

public class UppercaseToLowercaseConverter {

    public static void main(String[] args) {
        try {
            // Create BufferedReader to read from System.in
            BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

            // Create BufferedWriter to write to System.out
            BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(System.out));

            System.out.println("Enter text (Ctrl+C to exit):");

            String line;
            while ((line = reader.readLine()) != null) {
                // Convert the input line to lowercase
```

```
                    String lowercaseLine = line.toLowerCase();

                    // Write the lowercase line to the output
                    writer.write(lowercaseLine);
                    writer.newLine();
                    writer.flush();
                }

                // Close the reader and writer
                reader.close();
                writer.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

....

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

# Create a synthetic dataset with three clusters
n_samples = 300
n_features = 2
n_clusters = 3

X, _ = make_blobs(n_samples=n_samples, n_features=n_features,
centers=n_clusters, random_state=42)

# Apply Agglomerative Clustering
agg_clustering = AgglomerativeClustering(n_clusters=n_clusters)
agg_labels = agg_clustering.fit_predict(X)
```

```
# Visualize the data and Agglomerative Clustering results
plt.scatter(X[:, 0], X[:, 1], c=agg_labels, cmap='viridis')
plt.title('Agglomerative Clustering')
plt.show()
```