Q)Write a Java Program to implement Abstract Factory Pattern for Shape interface.

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

// Concrete implementation of Circle
class Circle implements Shape {
	@Override
	public void draw() {
	System.out.println("Drawing Circle");
	}
}

// Concrete implementation of Square
class Square implements Shape {
	@Override
	public void draw() {
	System.out.println("Drawing Square");
	}
}

// Concrete implementation of Rectangle
class Rectangle implements Shape {
	@Override
	public void draw() {
	System.out.println("Drawing Rectangle");
	}
}

// Concrete implementation of Triangle
class Triangle implements Shape {
	@Override
	public void draw() {
	System.out.println("Drawing Triangle");
	}
}

// Abstract Factory interface
interface ShapeFactory {
	Shape createCircle();
	Shape createSquare();
	Shape createRectangle();
```

```
        Shape createTriangle();
}

// Concrete Factory 1
class ConcreteFactory1 implements ShapeFactory {
        @Override
        public Shape createCircle() {
        return new Circle();
        }

        @Override
        public Shape createSquare() {
        return new Square();
        }

        @Override
        public Shape createRectangle() {
        return new Rectangle();
        }

        @Override
        public Shape createTriangle() {
        return new Triangle();
        }
}

// Concrete Factory 2
class ConcreteFactory2 implements ShapeFactory {
        @Override
        public Shape createCircle() {
        return new Circle();
        }

        @Override
        public Shape createSquare() {
        return new Square();
        }

        @Override
        public Shape createRectangle() {
        return new Rectangle();
        }

        @Override
```

```
        public Shape createTriangle() {
        return new Triangle();
        }
}

// Client class using the Abstract Factory
public class Client {
        public static void main(String[] args) {
        // Create a factory (you can switch between ConcreteFactory1 and ConcreteFactory2)
        ShapeFactory factory = new ConcreteFactory1();

        // Use the factory to create shapes
        Shape circle = factory.createCircle();
        Shape square = factory.createSquare();
        Shape rectangle = factory.createRectangle();
        Shape triangle = factory.createTriangle();

        // Draw the shapes
        circle.draw();
        square.draw();
        rectangle.draw();
        triangle.draw();
        }
}
```

Q) Write a python program to implement Multiple Linear Regression for a given dataset.

```
# Import necessary libraries
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Generate or load your dataset
# For the purpose of this example, let's generate a random dataset
np.random.seed(42)
num_samples = 100
X1 = np.random.rand(num_samples) * 10
X2 = np.random.rand(num_samples) * 5
```

```
noise = np.random.randn(num_samples) * 2
y = 3 * X1 + 2 * X2 + 5 + noise

# Create a DataFrame
data = pd.DataFrame({'X1': X1, 'X2': X2, 'y': y})

# Split the dataset into training and testing sets
X = data[['X1', 'X2']]
y = data['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model on the training set
model.fit(X_train, y_train)

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

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Print the coefficients and intercept
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)

# Visualization (optional)
plt.scatter(X_test['X1'], y_test, color='black', label='Actual')
plt.scatter(X_test['X1'], y_pred, color='red', label='Predicted')
plt.xlabel('X1')
plt.ylabel('y')
plt.legend()
plt.show()

feature1,feature2,target
2.0,3.5,7.1
3.5,5.1,9.3
4.2,6.0,11.1
5.1,6.5,12.5

import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load the dataset
dataset = pd.read_csv('dataset.csv')

# Separate features and target variable
X = dataset[['feature1', 'feature2']]
y = dataset['target']
```

Q) Write node js application that transfer a file as an attachment on web and enables browser to prompt the user to download file using express js.

```
var express = require('express');
var app = express();
var PORT = 3000;
app.get('/', function(req, res){
res.download('hello.txt');
res.sendFile(“hello.txt”) // res.setHeader('Content-Type', 'application/octet-stream');
});
app.listen(PORT, function(err){
if (err) console.log(err);
console.log("Server listening on PORT", PORT);
});
```

Q)Write a JAVA Program to implement built-in support (java.util.Observable) Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(), getPressure()

```
import java.util.Observable;
import java.util.Observer;

// WeatherData class representing the weather station
class WeatherData extends Observable {
        private float temperature;
        private float humidity;
        private float pressure;

        // Constructor
        public WeatherData() {
        // Initialization
```

```
        this.temperature = 0.0f;
        this.humidity = 0.0f;
        this.pressure = 0.0f;
        }

        // Method to notify observers about the change in measurements
        public void measurementsChanged() {
        setChanged(); // Marks the object as changed
        notifyObservers(); // Notifies all registered observers
        }

        // Setter method to update measurements
        public void setMeasurements(float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged(); // Notify observers after measurements are updated
        }

        // Getter methods for temperature, humidity, and pressure
        public float getTemperature() {
        return temperature;
        }

        public float getHumidity() {
        return humidity;
        }

        public float getPressure() {
        return pressure;
        }
}

// DisplayElement interface representing the display elements
interface DisplayElement {
        void display();
}

// CurrentConditionsDisplay class representing a display element
class CurrentConditionsDisplay implements Observer, DisplayElement {
        private float temperature;
        private float humidity;
        private Observable observable;
```

```
        // Constructor
        public CurrentConditionsDisplay(Observable observable) {
        this.observable = observable;
        observable.addObserver(this); // Register as an observer
        }

        // Update method called when the observable object is updated
        @Override
        public void update(Observable obs, Object arg) {
        if (obs instanceof WeatherData) {
        WeatherData weatherData = (WeatherData) obs;
        this.temperature = weatherData.getTemperature();
        this.humidity = weatherData.getHumidity();
        display(); // Update and display
        }
        }

        // Display method to show the current conditions
        @Override
        public void display() {
        System.out.println("Current conditions: " + temperature + "F degrees and " + humidity +
"% humidity");
        }
}

// Main class to test the weather station
public class Main {
        public static void main(String[] args) {
        // Create a weather data object
        WeatherData weatherData = new WeatherData();

        // Create display elements and register them as observers
        CurrentConditionsDisplay currentConditionsDisplay = new
CurrentConditionsDisplay(weatherData);

        // Simulate measurements changes
        weatherData.setMeasurements(80, 65, 30.4f);
        weatherData.setMeasurements(82, 70, 29.2f);
        weatherData.setMeasurements(78, 90, 29.2f);
        }
}
```

Q) Write a python program to implement Polynomial Linear Regression for given dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Generate or load your dataset
# For the purpose of this example, let's generate a random dataset
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 0.5 * X**2 - 3 * X + 2 + np.random.randn(100, 1) * 2
dataset_path = 'your_dataset.csv'
df = pd.read_csv(dataset_path)

# Assuming your dataset has two columns 'X' and 'y'
X = df[['X']].values
y = df['y'].values

# 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 PolynomialFeatures to transform input features
degree = 2  # Choose the degree of the polynomial
poly_features = PolynomialFeatures(degree=degree, include_bias=False)
X_train_poly = poly_features.fit_transform(X_train)

# Create and train the Polynomial Linear Regression model
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)

# Make predictions on the training set
y_train_pred = poly_reg.predict(X_train_poly)

# Evaluate the model on the training set
mse_train = mean_squared_error(y_train, y_train_pred)
print(f'Mean Squared Error on Training Set: {mse_train}')

# Visualize the Polynomial Linear Regression fit
X_range = np.linspace(0, 10, 100).reshape(-1, 1)
X_range_poly = poly_features.transform(X_range)
y_range_pred = poly_reg.predict(X_range_poly)
```

```
plt.scatter(X, y, label='Actual Data')
plt.plot(X_range, y_range_pred, color='red', label=f'Polynomial Regression (Degree {degree})')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Polynomial Linear Regression')
plt.show()
```

Q) Create your Django app in which after running the server, you should see on the browser, the text “Hello! I am learning Django”, which you defined in the index view.

```
pip install django
```

Certainly! Here's a step-by-step guide to create a simple Django app with an "Hello! I am learning Django" message in the index view:

1. **Create a Django Project:**

   Open a terminal and run the following commands to create a new Django project:

   ```bash
   django-admin startproject mydjangoapp
   cd mydjangoapp
   ```

2. **Create a Django App:**

   Inside your project directory (`mydjangoapp`), run the following command to create a new app:

   ```bash
   python manage.py startapp myapp
   ```

3. **Define a View:**

   Open the `myapp/views.py` file and define a simple view:

```python
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello! I am learning Django")
```

4. **Create URLs Configuration:**

   Create a `urls.py` file inside the `myapp` directory and define the URL patterns:

```python
from django.urls import path
from .views import index

urlpatterns = [
    path('', index, name='index'),
]
```

5. **Include App URLs in Project URLs:**

   Open the `mydjangoapp/urls.py` file and include the URLs of your app:

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]
```

6. **Run the Development Server:**

   Run the following command to start the Django development server:

```bash
python manage.py runserver
```

   Visit `http://127.0.0.1:8000/` in your web browser, and you should see the "Hello! I am learning Django" message.

That's it! You've created a simple Django app with a single view. Feel free to explore and expand your app as you continue learning Django.