mybackend\api\views.py

```python
# views.py
from rest_framework import viewsets, status
from rest_framework.response import Response
from rest_framework.decorators import action
from .models import DataPoint
from .serializers import DataPointSerializer
import random
import numpy as np
from datetime import datetime

class DataViewSet(viewsets.ModelViewSet):
    queryset = DataPoint.objects.all()
    serializer_class = DataPointSerializer

    def generate_realistic_data(self):
        """
        Generate realistic-looking data for both scatter and line plots
        """
        num_points = 50  # Number of data points to generate

        # Generate x values (evenly spaced)
        x_values = np.linspace(0, 10, num_points)

        # Generate line data (smooth curve with some noise)
        base_line = 50 + 30 * np.sin(x_values * 0.5) + 20 * np.cos(x_values * 0.3)
        noise = np.random.normal(0, 2, num_points)
        line_values = base_line + noise

        # Generate scatter data (correlated with line but more scattered)
        scatter_noise = np.random.normal(0, 10, num_points)
        scatter_values = base_line + scatter_noise

        # Ensure all values are positive and rounded to 2 decimal places
        line_values = np.maximum(0, line_values)
        scatter_values = np.maximum(0, scatter_values)

        return {
            'x_values': [round(x, 2) for x in x_values],
            'line_values': [round(y, 2) for y in line_values],
            'scatter_values': [round(y, 2) for y in scatter_values]
        }

    # gets invoked on GET http://127.0.0.1:8000/api/data/
    def list(self, request, *args, **kwargs):
        """
        Override the default GET behavior to return generated data
        """
        try:
```

```python
            # Generate new data
            generated_data = self.generate_realistic_data()

            # Format the response
            response_data = {
                'scatter_data': {
                    'x': generated_data['x_values'],
                    'y': generated_data['scatter_values']
                },
                'line_data': {
                    'x': generated_data['x_values'],
                    'y': generated_data['line_values']
                },
                'metadata': {
                    'x_range': {
                        'min_x': min(generated_data['x_values']),
                        'max_x': max(generated_data['x_values'])
                    },
                    'y_range': {
                        'min_y': min(min(generated_data['scatter_values']),
                                     min(generated_data['line_values'])),
                        'max_y': max(max(generated_data['scatter_values']),
                                     max(generated_data['line_values']))
                    },
                    'total_points': len(generated_data['x_values']),
                    'timestamp': datetime.now().isoformat()
                }
            }

            return Response(response_data)

        except Exception as e:
            return Response(
                {'error': str(e)},
                status=status.HTTP_400_BAD_REQUEST
            )

    def get_random_data_variation(self):
        """
        Generate different types of data patterns
        """
        pattern_type = random.choice(['linear', 'exponential', 'sinusoidal', 'random'])
        num_points = 50
        x_values = np.linspace(0, 10, num_points)

        if pattern_type == 'linear':
            slope = random.uniform(0.5, 2.0)
            intercept = random.uniform(0, 30)
            base_values = slope * x_values + intercept

        elif pattern_type == 'exponential':
            base_values = np.exp(x_values * 0.3) + random.uniform(0, 10)

        elif pattern_type == 'sinusoidal':
            frequency = random.uniform(0.3, 0.8)
            amplitude = random.uniform(20, 40)
            base_values = amplitude * np.sin(x_values * frequency) + 50

        else:  # random
            base_values = np.random.uniform(0, 100, num_points)

        return base_values

    # gets invoked on GET http://127.0.0.1:8000/api/data/random_variation/
    @action(detail=False, methods=['GET'])
    def random_variation(self, request):
        """
        Endpoint to get random variations of data patterns
        """
        try:
            base_values = self.get_random_data_variation()
            x_values = np.linspace(0, 10, len(base_values))

            # Add noise to create scatter and line variations
            scatter_noise = np.random.normal(0, 5, len(base_values))
            line_noise = np.random.normal(0, 2, len(base_values))

            scatter_values = base_values + scatter_noise
            line_values = base_values + line_noise

            response_data = {
                'scatter_data': {
                    'x': [round(x, 2) for x in x_values],
                    'y': [round(y, 2) for y in scatter_values]
                },
                'line_data': {
                    'x': [round(x, 2) for x in x_values],
                    'y': [round(y, 2) for y in line_values]
                },
                'metadata': {
                    'pattern_type': 'random variation',
                    'timestamp': datetime.now().isoformat()
                }
            }

            return Response(response_data)

        except Exception as e:
            return Response(
                {'error': str(e)},
```

```python
                status=status.HTTP_400_BAD_REQUEST
            )


"""
#serializers.py
from rest_framework import serializers
from .models import DataPoint

class DataPointSerializer(serializers.ModelSerializer):
    class Meta:
        model = DataPoint
        fields = ['id', 'x_value', 'y_value', 'line_value', 'category', 'timestamp']
####################################################
#models.py
from django.db import models

class DataPoint(models.Model):
    x_value = models.FloatField(help_text="X-axis value for both graphs")
    y_value = models.FloatField(help_text="Y-axis value for scatter plot")
    line_value = models.FloatField(help_text="Y-axis value for line graph")
    category = models.CharField(max_length=100, null=True, blank=True,
                              help_text="Optional category for data grouping")
    timestamp = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['x_value']  # Default ordering by x_value

    def __str__(self):
        return f"DataPoint (x={self.x_value}, scatter_y={self.y_value}, line_y={self.line_value})"
##########################################
#urls.py
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import DataViewSet

router = DefaultRouter()
router.register(r'data', DataViewSet)

#GET http://127.0.0.1:8000/api/data/
#GET http://127.0.0.1:8000/api/data/random_variation/

urlpatterns = [
    path('', include(router.urls)),
]
"""
```