Research: The Internet of Things (Cars)

Role: Writing a Synthetic Data Generation Tool

We wanted to simulate vehicles traveling on roads at different velocities and also undergoing different circumstances such as icy roads, speed bumps etc. At the same time, we wanted to connect the roads through sensors that will provide the surrounding conditions.

Task 1:
Calculating geodesic distances and predicting the next coordinate.

```
EARTH_RADIUS = 6378.1   # Radius of earth
def predict_next_coordinate(current_coordinate, distance, direction):
        R = EARTH_RADIUS
        bearing = math.radians(direction)
        lat_1 = math.radians(current_coordinate[0])
        lon_1 = math.radians(current_coordinate[1])
        lat_2 = math.asin(math.sin(lat_1)*math.cos(distance/R) + math.cos(lat_1)*math.sin(distance/R)*math.cos(bearing))
        lon_2 = lon_1 + math.atan2(math.sin(bearing)*math.sin(distance/R)*math.cos(lat_1), math.cos(distance/R)-math.sin(lat_1)*math.sin(lat_2))
        final_lat = math.degrees(lat_2)
        final_lon = math.degrees(lon_2)
        return (final_lat, final_lon)
#end_def
def distance_between_two_coordinates(source_coordinate, destination_coordinate):
        phi1 = math.radians(90.0 - source_coordinate[0])
        phi2 = math.radians(90.0 - destination_coordinate[0])
        theta1 = math.radians(source_coordinate[1])
        theta2 = math.radians(destination_coordinate[1])
        cosine = (math.sin(phi1)*math.sin(phi2)*math.cos(theta1 - theta2) +
           math.cos(phi1)*math.cos(phi2))
```

```
        arc = math.acos(cosine)
        standardized_arc = EARTH_RADIUS * arc
        return standardized_arc
#end_def
```

The first method is used to predict the coordinate that is x distance away from the current coordinate in a specific direction (that we call bearing). The second method is used to find the distance between two given coordinates. These two methods are the backbone of the data generation tool as they relate to actual co-ordinates on the earth. Also, this will be done with libraries in the future.

Task 2:

Making a straight line path by predicting coordinates.

```
SEED_COORDINATE = (40.12009,-88.247681)      # Coordinate
SEED_VELOCITY = 10                           # 10 miles/hour
SEED_TIME = 0.0833333                        # 5 minutes converted to hour
SEED_DIRECTION = 90                          # 90 degrees

def get_random_coordinates(num_coordinates):
        current_velocity = SEED_VELOCITY
        current_coordinate = SEED_COORDINATE
        for i in range(num_coordinates):
                random_error = random.random()
                random_range = random.randint(0, 5)
                velocity_change = random_range*random_error
                current_velocity = current_velocity + velocity_change
                distance_travelled = current_velocity * SEED_TIME
                next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
                current_coordinate = next_coordinate
                csv = str(next_coordinate[0]) + ',' + str(next_coordinate[1])
                print(csv)
#end_def

```

The above method provides N coordinates (specified by the user) in a straight direction (SEED_DIRECTION). The method allows some variance in speed but only in the positive direction. Therefore, the velocity will only rise over time. This was a major drawback and we needed to randomize the direction as well as it would ensure rising and falling velocities.

Task 3:

Making a straight line path by predicting coordinates using negative and positive accelerations.

```
def get_random_coordinates_positive_acceleration(num_coordinates):
        current_velocity = SEED_VELOCITY
        current_coordinate = SEED_COORDINATE
        for i in range(num_coordinates):
                random_error = random.random()
                random_range = random.randint(0, 5)
                random_direction = random.randint(0, 1)
                velocity_change = random_range*random_error
                if random_direction == 0:
                  current_velocity = current_velocity + velocity_change
                else:
                  current_velocity = current_velocity - velocity_change
                distance_travelled = current_velocity * SEED_TIME
                next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
                current_coordinate = next_coordinate
                csv = str(next_coordinate[0]) + ',' + str(next_coordinate[1])
                print(csv)
#end_def

```

To allow both negative and positive accelerations, a random_direction parameter was used to flip the acceleration sign. The data generated was random enough but still lacked some things such as it could make the velocity also negative. i.e: car is traveling in reverse direction. Although this was random enough, it was realistic for the car to switch directions along the same path alternatively.

Task 4.1:

Improve the current method not worrying about direction changes so that it could write to Kafka and also generate data for N cars along the same line

```
def randomize_conditions(max_range):
        random_error = random.random()
        random_range = random.randint(0, max_range)
        velocity_change = random_range*random_error
        random_direction = random.randint(0, 1)
        random_dict = {}
        random_dict['velocity_change'] = velocity_change
        random_dict['direction'] = random_direction
        return random_dict
#end_def

def get_n_car_motion_for_random_coordinates(num_coordinates, num_cars):
        result = []
        current_velocity = SEED_VELOCITY
        current_coordinate = SEED_COORDINATE
        time_stamp = datetime.datetime.now()
        first_stamp = time_stamp
        unique_id = uuid.uuid4()
        first_car_data = []

        for i in range(num_coordinates):
                random_error = random.random()
                random_range = random.randint(0, 5)
                velocity_change = random_range*random_error
                random_direction = random.randint(0, 1)
                if random_direction == 0:
                        current_velocity = current_velocity - velocity_change
                elif random_direction == 1:
                        current_velocity = current_velocity + velocity_change
                distance_travelled = current_velocity * SEED_TIME
                next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
```

```
                current_coordinate = next_coordinate
                json_object = {}
                json_object['car_id'] = str(unique_id)
                json_object['latitude'] = current_coordinate[0]
                json_object['longitude'] = current_coordinate[1]
                json_object['time_stamp'] = str(time_stamp)
                first_car_data.append(json_object)
                time_stamp = time_stamp + datetime.timedelta(0, 300)
        result.append(first_car_data)
        for i in range(num_cars-1):
                secondary_car_data = get_car_motion_for_coordinates(first_car_data, first_stamp)
                result.append(secondary_car_data)
        return result
#end_def

def get_car_motion_for_coordinates(data, first_stamp):
        json_data = []
        temp_object = data[0]
        time_stamp = first_stamp
        current_velocity = SEED_VELOCITY
        unique_id = uuid.uuid4()
        for item in data:
                distance = geo.distance_between_two_coordinates((temp_object['latitude'], temp_object['longitude']),(item['latitude'], item['longitude']))        # stays constant
                random_dict = randomize_conditions(6)
                if random_dict['direction'] == 0:
                        current_velocity = current_velocity - random_dict['velocity_change']
                elif random_dict['direction'] == 1:
                        current_velocity = current_velocity + random_dict['velocity_change']
                time_taken = distance/current_velocity
                temp_object['car_id'] = str(unique_id)
```

```
                temp_object['latitude'] = item['latitude']
                temp_object['longitude'] = item['longitude']
                time_stamp = time_stamp + datetime.timedelta(0, time_ta
ken)
                temp_object['time_stamp'] = str(time_stamp)
                json_data.append(temp_object)
                temp_object = item
        return json_data
#end_def
```

Here the first method is used randomize conditions and provide a dictionary of conditions each time it is called. Given there are N coordinates for M random cars. The first method generates N random coordinates for one car in the beginning and feeds those N coordinates to the other method that then generates random data for M-1 cars along the same path. The motivation for using the same coordinates and time difference was to simulate different velocities in different cars. Other methods were used to write to files and the Kafka Server.

Task 4.2:

```
import json
from pprint import pprint
def json_write_to_file(data, filename, display=False):
        if display is True:
                pprint(data)
        with open(filename, 'w+') as newfile:
                json.dump(data, newfile)
        return
#end_def
def json_read_from_file(filename, display=False):
        with open(filename) as data_file:
                data = json.load(data_file)
        if display is True:
                pprint(data)
        return data
```

Methods to print data and write to file.

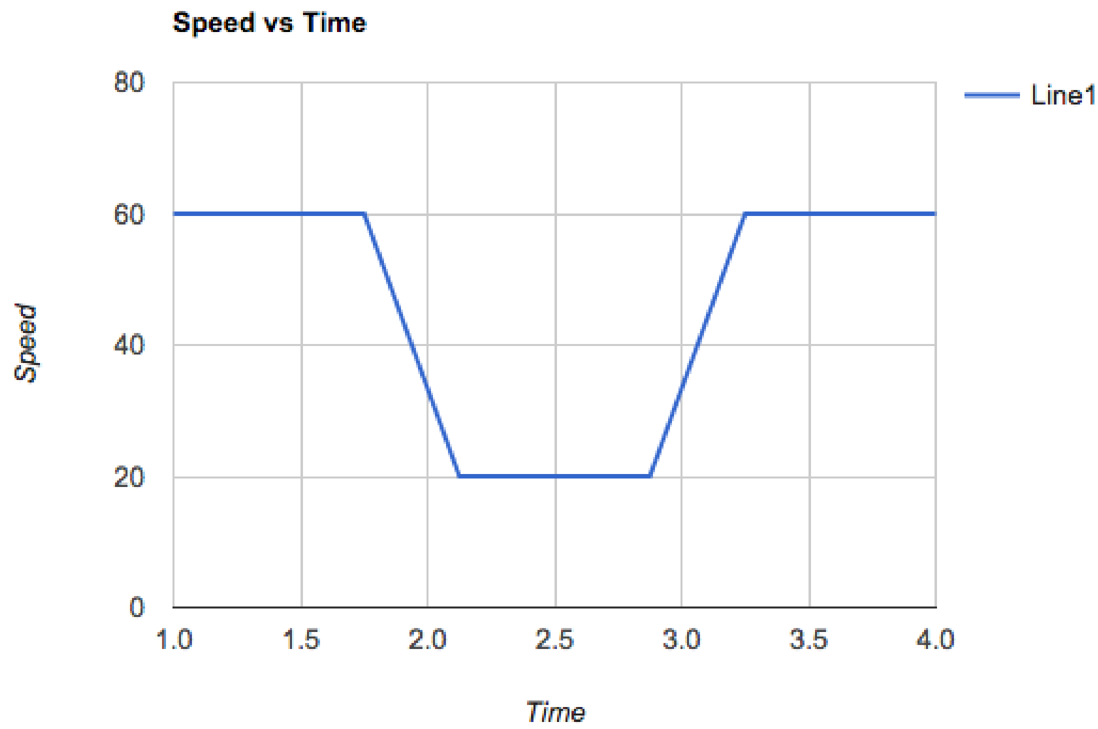
```
from kafka import SimpleProducer, KafkaClient
```

```python

SEED_BROKER = '141.142.236.172:9092'
def generate_and_populate_rect_data_for_n_cars(data_points, num_cars):
    res = rect.get_n_car_motion_for_random_coordinates(data_points, num_cars)
    text_file_name = 'data_for_'+str(num_cars)+'_cars.json'
    fio.json_write_to_file(res, text_file_name)
    # Kafka
    kafka = KafkaClient(SEED_BROKER)
    producer = SimpleProducer(kafka, async=True)
    for item in res:
         for obj in item:
             message_str = obj['car_id'] +','+ str(obj['latitude'])+','+str(obj['longitude'])+','+ obj['time_stamp']
             encoded = message_str.encode()
             producer.send_messages(b'mytopic', encoded)

```

This was a wrapper method that called the data generation method and got the data. It later streamed all of that data to Kafka in the necessary format. The call made is asynchronous and therefore the data is streamed much faster.

Task 5:

Generate data with a speed dip indicating general bad conditions of the road.
The first simple approach was to generate data that looked something like this:



This graph indicates an immediate speed bump and other than that will have constant velocity. It was an ideal scenario for testing but not very realistic.

```python
def get_random_coordinates_sudden_dip(num_coordinates):
        result = []
        current_coordinate = SEED_COORDINATE
        time_stamp = datetime.datetime.now()
        first_stamp = time_stamp
        unique_id = uuid.uuid4()
        first_car_data = []

        for i in range(num_coordinates/3):
                distance_travelled = SEED_VELOCITY * SEED_TIME
                next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
                current_coordinate = next_coordinate
                json_object = {}
                json_object['car_id'] = str(unique_id)
                json_object['latitude'] = current_coordinate[0]
                json_object['longitude'] = current_coordinate[1]
```

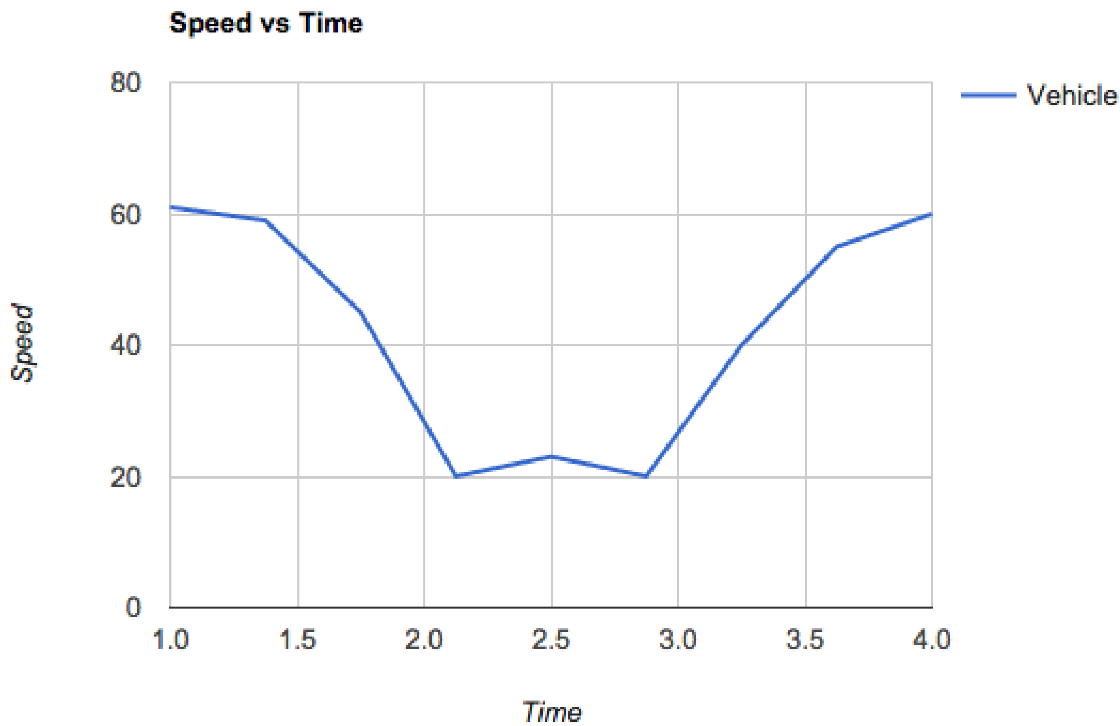
```
                json_object['time_stamp'] = str(time_stamp)
                first_car_data.append(json_object)
                time_stamp = time_stamp + datetime.timedelta(0, 300)

        for i in range(num_coordinates/3):
                distance_travelled = (SEED_VELOCITY-10) * SEED_TIME
                next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
                current_coordinate = next_coordinate
                json_object = {}
                json_object['car_id'] = str(unique_id)
                json_object['latitude'] = current_coordinate[0]
                json_object['longitude'] = current_coordinate[1]
                json_object['time_stamp'] = str(time_stamp)
                first_car_data.append(json_object)
                time_stamp = time_stamp + datetime.timedelta(0, 300)

        for i in range(num_coordinates/3):
                distance_travelled = SEED_VELOCITY * SEED_TIME
                next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
                current_coordinate = next_coordinate
                json_object = {}
                json_object['car_id'] = str(unique_id)
                json_object['latitude'] = current_coordinate[0]
                json_object['longitude'] = current_coordinate[1]
                json_object['time_stamp'] = str(time_stamp)
                first_car_data.append(json_object)
                time_stamp = time_stamp + datetime.timedelta(0, 300)
        result.append(first_car_data)
        return result
```

The above method was used to simulate this data. Since there was no varying velocities there wasn't any drawbacks about the velocity going negative.

Task 6:

Generate data with a speed dip indicating general bad conditions of the road. (varying velocities)

The first simple approach was to generate data that looked something like this:



The above graph shows varying velocity undergoing a sudden dip in speed and then getting back to the range it had before. This graph is more realistic as speeds keep on varying and it shows how there could be a bad patch of road in between a highway.

```
def get_random_coordinates_sudden_dip_varying(num_coordinates):
        result = []
        current_coordinate = SEED_COORDINATE
        time_stamp = datetime.datetime.now()
        first_stamp = time_stamp
        unique_id = uuid.uuid4()
        first_car_data = []
        current_velocity = SEED_VELOCITY

        for i in range(num_coordinates/3):
                distance_travelled = current_velocity * SEED_TIME
                next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
```

```
        current_coordinate = next_coordinate
        json_object = {}
        json_object['car_id'] = str(unique_id)
        json_object['latitude'] = current_coordinate[0]
        json_object['longitude'] = current_coordinate[1]
        json_object['time_stamp'] = str(time_stamp)
        first_car_data.append(json_object)
        time_stamp = time_stamp + datetime.timedelta(0, 300)
        random_dict = randomize_conditions(2)
        print(i, current_velocity)
        if random_dict['direction'] == 1:
            current_velocity = current_velocity + random_dict['velocity_change']
        if random_dict['direction'] == 0:
            if current_velocity + random_dict['velocity_change'] > 10:
                current_velocity = current_velocity - random_dict['velocity_change']
    current_velocity = current_velocity - 8
    for i in range(num_coordinates/3):
        distance_travelled = current_velocity * SEED_TIME
        next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
        current_coordinate = next_coordinate
        json_object = {}
        json_object['car_id'] = str(unique_id)
        json_object['latitude'] = current_coordinate[0]
        json_object['longitude'] = current_coordinate[1]
        json_object['time_stamp'] = str(time_stamp)
        first_car_data.append(json_object)
        time_stamp = time_stamp + datetime.timedelta(0, 300)
        random_dict = randomize_conditions(2)
        print(i, current_velocity)
        if random_dict['direction'] == 1:
            current_velocity = current_velocity + random_di
```

```
ct['velocity_change']
                if random_dict['direction'] == 0:
                        if current_velocity + random_dict['velocity_change'] > 6:
                                current_velocity = current_velocity - random_dict['velocity_change']

        current_velocity = current_velocity + 8
        for i in range(num_coordinates/3):
                distance_travelled = current_velocity * SEED_TIME
                next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
                current_coordinate = next_coordinate
                json_object = {}
                json_object['car_id'] = str(unique_id)
                json_object['latitude'] = current_coordinate[0]
                json_object['longitude'] = current_coordinate[1]
                json_object['time_stamp'] = str(time_stamp)
                first_car_data.append(json_object)
                time_stamp = time_stamp + datetime.timedelta(0, 300)
                random_dict = randomize_conditions(2)
                print(i, current_velocity)
                if random_dict['direction'] == 1:
                        current_velocity = current_velocity + random_dict['velocity_change']
                if random_dict['direction'] == 0:
                        if current_velocity + random_dict['velocity_change'] > 10:
                                current_velocity = current_velocity - random_dict['velocity_change']
        result.append(first_car_data)
        return result

```

The above method was used to generate the data. This was a much more realistic vision since velocity barriers were added in the method to account for velocities going in the negative phase.

Hence, the barrier such as 10mph will act as the lower threshold of the speed. We could even add a higher threshold that could be the speed limit for the road but it really is not needed for now.

Task 7:

Simulating Accidents (Still in progress)

```
def get_random_coordinates_with_accident(num_coordinates):
        current_velocity = SEED_VELOCITY
        current_coordinate = SEED_COORDINATE
        accident = random.randint(1, num_coordinates-1)
        has_crashed = False

        for x in xrange(1,num_coordinates):
            random_dict = randomize_conditions(5)
            if has_crashed:
               has_crashed = False
               current_velocity = SEED_VELOCITY_CRASHED
            if x == accident:
               has_crashed = True
               current_velocity = 0
               csv = str(current_coordinate[0]) + ',' + str(current_coordinate[1]) + ', Accident'
               print(csv)
               continue
             if random_dict['direction'] == 0:
               current_velocity = current_velocity - random_dict['velocity_change']
             elif random_dict['direction'] == 1:
               current_velocity = current_velocity + random_dict['velocity_change']
               distance_travelled = current_velocity * SEED_TIME
              next_coordinate = geo.predict_next_coordinate(current_coordinate, distance_travelled, SEED_DIRECTION)
              csv = str(current_coordinate[0]) + ',' + str(current_coordinate[1]) + ' : ' + str(distance_travelled)
              print(csv)
#end_def
```

The above method is for simulating accidents. (In Progress)

Added weather data

Need to take other factors such as temperature, snow and wind speed into account. In the table below 0 represents a decrease in that quantity and 1 represents an increase in the quantity. The last two cases of the truth table namely, increase in temperature and snow and increase in temperature, snow and wind speed yield not applicable since the assumption is that increase in snow will not increase the temperature. If this case happens it would bring no change to the speed of the vehicle. However, certain thresholds were kept to ensure that minute changes are not considered. These changes played a role in determining the change in the vehicle speeds.

Temperature	Snow	Wind Speed	Change in Vehicle Speed
0	0	0	No change
0	0	1	Slower by 1x
0	1	0	Slower by 2x
0	1	1	Slower by 3x
1	0	0	No change
1	0	1	Slower by 1x
1	1	0/1	Not Applicable

```
def get_speed_change(current_velocity, weather, previous_condition):
        temperature = difference(weather['tmin'], previous_condition['tmin'])
        snow = difference(weather['snow'], previous_condition['snow'])
        windspeed = difference(weather['wind'], previous_condition['windspeed'])
        final_velocity = current_velocity
        if windspeed == 1:
                final_velocity = final_velocity - (0.1*current_velocity)

        if snow == 1:
                final_velocity = final_velocity - (0.2*current_velocity)
        return final_velocity
```

Turns (Left and Right)

We needed to incorporate turns in our data sets so that we could harness more real life situations. The 90 degree left and right turns were included as the starting point. As of now, all the cars along the road turn as it is being assumed the there was a dead-end at the end of the road. There is a need to generate data such that only a fraction of the cars take the turn and others keep on continuing on the same road.

The vehicles can randomly turn as there is a random variable introduced for turns as well. The model has a forty percent probability of deviating from the straight line right now. The theta was changed accordingly to 180 and 0.

The U-Turn was added by changing the theta to 270 as we started with 90 degrees as our seed directions. The limitation about turns is detecting accidents while turning. We need to maintain a set of current points for all the cars and check while turning if any two cars are colliding.