data_cleaning

March 1, 2024

[ ]:
```
# imports
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import os
```

1 Datasets

For our project, we will use four different datasets and combine them into a single dataset (joining on the timestamp). The dataset that we found on Kaggle has 9 seperate folders each containing combinations of different runs and drivers. We plan to save at least one of these folders for testing and will use the rest for training and validation. Below we describe the datasets in each of the folders:

GPS: The following columns are in the GPS dataset: 'timestamp', 'latitude', 'longitude', 'elevation', 'accuracy','bearing', 'speed_meters_per_second', 'provider', 'battery','distance_meters', 'elapsed_time_seconds'. We will drop the following rows from the GPS dataset: 'ageofdgpsdata', "dgpsid", "activity", "annotation", 'hdop', 'vdop', 'pdop', "satellites", "geoidheight". From the GPS dataset, the only missing values are several missing values from bearing at the beginning of the dataset. We drop these rows.

Left and Right GPS: The following columns are in the Left and Right GPS: 'timestamp', 'acc_x_dash', 'acc_y_dash', 'acc_z_dash', 'acc_x_above', 'acc_y_above', 'acc_z_above', 'acc_x_below', 'acc_y_below','acc_z_below', 'gyro_x_dash', 'gyro_y_dash', 'gyro_z_dash', 'gyro_x_above', 'gyro_y_above', 'gyro_z_above', 'gyro_x_below','gyro_y_below', 'gyro_z_below', 'temp_dash', 'temp_above', 'temp_below','timestamp_gps', 'latitude', 'longitude', 'speed'. The gyro columns contain rotational data from a gyroscope. We drop the following columns as they do not contribute to our modeling: "temp_dash", "temp_above", "temp_below". There are no missing values.

Dataset Labels: The following columns are in the labels dataset: 'paved_road', 'unpaved_road', 'dirt_road', 'cobblestone_road', 'asphalt_road', 'no_speed_bump', 'speed_bump_asphalt', 'speed_bump_cobblestone', 'good_road_left', 'regular_road_left','bad_road_left', 'good_road_right', 'regular_road_right','bad_road_right'. From this dataset, we create a new dataframe and a score for quality. The following columns are in our new dataframe: 'road', 'condition', 'bumps', 'quality'. There are no missing values.

[ ]:
```
#load
pvss = os.listdir("data/")
```

```
gps = pd.read_csv("data/"+pvss[0])
left_gps = pd.read_csv("data/"+pvss[3])
right_gps = pd.read_csv("data/"+pvss[1])
df_labels = pd.read_csv("data/"+pvss[2])
```

2 Data Cleaning Functions

[ ]:
```
def clean_gps(df):
    """
    Clean the gps data by dropping bad rows and columns

    Parameters
    ----------
    df : pd.DataFrame
        The gps data

    Returns
    -------
    pd.DataFrame
        The cleaned gps data
    """
    bad_cols = ['ageofdgpsdata', "dgpsid", "activity", "annotation"]
    useless_cols = ['hdop', 'vdop', 'pdop', "satellites", "geoidheight"]

    for col in bad_cols + useless_cols:
        if col in df.columns:
            df = df.drop(columns=[col])
    df = df.dropna()
    return df
```

[ ]:
```
def clean_acc(dirty_df):
    """
    Clean the accelerometer data by dropping bad rows and columns

    Parameters
    ----------
    left : pd.DataFrame
        The left accelerometer data
    right : pd.DataFrame
        The right accelerometer data

    Returns
    -------
    pd.DataFrame
        The cleaned accelerometer data
    """
```

```
    useless_cols = ["temp_dash", "temp_above", "temp_below"]
    for col in useless_cols:
        if col in dirty_df.columns:
            dirty_df = dirty_df.drop(columns=[col])

    for col in dirty_df.columns:
        if "mag" in col:
            dirty_df = dirty_df.drop(columns=[col])

    # remove the word "suspect" from the columns
    dirty_df.columns = dirty_df.columns.str.replace("_suspension", "")
    dirty_df.columns = dirty_df.columns.str.replace("dashboard", "dash")

    dirty_df = dirty_df.dropna()

    return dirty_df
```

[ ]:
```
def combine_data(dfs):
    """
    Combine the data into one dataframe by merging on the timestamp column

    Parameters
    ----------
    dfs : list
        A list of dataframes to combine

    Returns
    -------
    pd.DataFrame
        The combined dataframe
    """
    df = dfs[0]
    for i in range(1, len(dfs)):
        # merge the dataframes on timestamp
        df = pd.merge(df, dfs[i], on="timestamp")
    return df
```

[ ]:
```
# Function to convert OHE to label
def ohe_to_label(df_in, classes, df_out, class_name):
    conditions = []
    for r in classes:
        conditions.append(df_in[r] == 1)
    df_out[class_name] = np.select(conditions, classes)
    return df_out
```

3 Labels

[ ]:
```
label_names = pd.DataFrame(columns = ['road', 'condition','quality_right',␣
 ↪'quality_left','bumps'])

road_classes = ['dirt_road', 'cobblestone_road', 'asphalt_road']
quality_left_classes = ['good_road_left', 'regular_road_left', 'bad_road_left']
quality_right_classes = ['good_road_right', 'regular_road_right',␣
 ↪'bad_road_right']
condition_classes = ['paved_road', 'unpaved_road']
bump_classes=['no_speed_bump', 'speed_bump_asphalt', 'speed_bump_cobblestone']

# Convert from one-hot encoding to single label encoding
label_names = ohe_to_label(df_labels, road_classes, label_names, 'road')
label_names = ohe_to_label(df_labels, quality_right_classes, label_names,␣
 ↪'quality_right')
label_names = ohe_to_label(df_labels, quality_left_classes, label_names,␣
 ↪'quality_left')
label_names = ohe_to_label(df_labels, condition_classes, label_names,␣
 ↪'condition')
label_names = ohe_to_label(df_labels, bump_classes, label_names, 'bumps')

# Convert road quality labels to numeric values
label_names = label_names.replace({'quality_right' : { 'good_road_right' : 2,␣
 ↪'regular_road_right' : 1, 'bad_road_right' : 0 }})
label_names = label_names.replace({'quality_left' : { 'good_road_left' : 2,␣
 ↪'regular_road_left' : 1, 'bad_road_left' : 0 }})
label_names['quality'] = label_names.loc[: , "quality_right":"quality_left"].
 ↪mean(axis=1)
label_names = label_names.drop(columns = ["quality_right","quality_left"], axis␣
 ↪= 1)

label_names.head()
```

[ ]:

	road	condition	bumps	quality
0	asphalt_road	paved_road	no_speed_bump	2.0
1	asphalt_road	paved_road	no_speed_bump	2.0
2	asphalt_road	paved_road	no_speed_bump	2.0
3	asphalt_road	paved_road	no_speed_bump	2.0
4	asphalt_road	paved_road	no_speed_bump	2.0

[ ]:
```
label_names.columns
```

[ ]:
```
Index(['road', 'condition', 'bumps', 'quality'], dtype='object')
```

4 GPS

Something we did not expect to see was the distance_meters and elapsed_time_seconds, we are still unsure of what these columns mean and do not plan to use them.

[ ]:
```
gps = clean_gps(gps)
gps.head()
```

[ ]:
```
       timestamp   latitude  longitude   elevation  accuracy    bearing  \
0   1.577219e+09 -27.717812 -51.098895  948.770836      24.0  159.73294
10  1.577219e+09 -27.717840 -51.098877  986.167056       4.0  315.85168
11  1.577219e+09 -27.717840 -51.098877  985.918529       4.0  316.12387
12  1.577219e+09 -27.717840 -51.098876  985.829575       4.0  316.15497
13  1.577219e+09 -27.717839 -51.098873  985.567538       4.0  315.31592

    speed_meters_per_second provider  battery  distance_meters  \
0                  0.053275      gps       87         0.000000
10                 0.101402      gps       86         0.306061
11                 0.056578      gps       86         0.022915
12                 0.033049      gps       86         0.038035
13                 0.014423      gps       86         0.357210

    elapsed_time_seconds
0                    0.0
10                   1.0
11                   1.0
12                   1.0
13                   2.0
```

[ ]:
```
gps.describe()
```

[ ]:
```
          timestamp     latitude    longitude    elevation     accuracy  \
count  1.458000e+03  1458.000000  1458.000000  1458.000000  1458.000000
mean   1.577219e+09   -27.694939   -51.119457   925.209557     4.087791
std    4.277255e+02     0.011650     0.011296    40.538447     0.644262
min    1.577219e+09   -27.717845   -51.132691   874.835101     4.000000
25%    1.577219e+09   -27.701519   -51.128972   888.995484     4.000000
50%    1.577219e+09   -27.689817   -51.124745   908.203685     4.000000
75%    1.577220e+09   -27.687075   -51.109971   961.046558     4.000000
max    1.577220e+09   -27.681820   -51.098860   995.974683    24.000000

           bearing  speed_meters_per_second      battery  distance_meters  \
count  1458.000000              1458.000000  1458.000000      1458.000000
mean    213.629077                 9.343766    82.995885         9.476842
std      95.640711                 7.810864     1.274849         8.119713
min       1.006545                 0.002526    81.000000         0.000000
25%     138.297247                 4.334018    82.000000         4.356211
50%     197.380600                 6.554155    83.000000         6.578976
```

```
75%     317.124672                15.515677    84.000000        16.425724
max     359.790700                26.874480    87.000000        82.384473

       elapsed_time_seconds
count           1458.000000
mean               1.034294
std                0.240517
min                0.000000
25%                1.000000
50%                1.000000
75%                1.000000
max                6.000000
```

5 Left and Right GPS

[ ]:
```
left_gps = clean_acc(left_gps)
right_gps = clean_acc(right_gps)
left_gps.head()
```

[ ]:
```
      timestamp  acc_x_dash  acc_y_dash  acc_z_dash  acc_x_above  acc_y_above  \
0  1.577219e+09    0.365116    0.167893    9.793961     0.327626     0.172733
1  1.577219e+09    0.392649    0.176273    9.771216     0.381496     0.189492
2  1.577219e+09    0.409408    0.181062    9.732909     0.283333     0.182310
3  1.577219e+09    0.371101    0.164302    9.749668     0.314458     0.230194
4  1.577219e+09    0.390255    0.159514    9.869378     0.344385     0.202660

   acc_z_above  acc_x_below  acc_y_below  acc_z_below  …  gyro_x_above  \
0     9.781861     0.024797     0.172611     9.793824  …      0.138446
1     9.699261     0.024797     0.194158     9.842905  …      0.168963
2     9.807000     0.003249     0.227677     9.888395  …     -0.136213
3     9.739963     0.005643     0.172611     9.871635  …     -0.075177
4     9.762708     0.005643     0.200144     9.860862  …      0.062152

   gyro_y_above  gyro_z_above  gyro_x_below  gyro_y_below  gyro_z_below  \
0      0.159659     -0.072572     -0.041780      0.167302     -0.078110
1      0.068106      0.095274      0.019255      0.304631      0.150771
2      0.159659      0.156310     -0.377473     -0.122615      0.028701
3      0.037589      0.064757      0.049773     -0.183650      0.059219
4      0.022330      0.003722      0.141325     -0.046321      0.013442

   timestamp_gps   latitude  longitude     speed
0   1.577219e+09 -27.717841 -51.098865  0.009128
1   1.577219e+09 -27.717841 -51.098865  0.009128
2   1.577219e+09 -27.717841 -51.098865  0.009128
3   1.577219e+09 -27.717841 -51.098865  0.009128
4   1.577219e+09 -27.717841 -51.098865  0.009128
```

[5 rows x 23 columns]

[ ]:
```
left_gps.describe()
```

[ ]:

	timestamp	acc_x_dash	acc_y_dash	acc_z_dash	\
count	1.440360e+05	144036.000000	144036.000000	144036.000000	
mean	1.577220e+09	-0.171308	0.015106	9.719331	
std	4.157976e+02	1.399775	1.940585	1.831907	
min	1.577219e+09	-10.735600	-13.446734	-6.418376	
25%	1.577219e+09	-0.912191	-0.829292	8.904515	
50%	1.577220e+09	-0.102951	0.090082	9.754456	
75%	1.577220e+09	0.543484	0.839467	10.522995	
max	1.577220e+09	9.033323	14.258967	24.665544	

	acc_x_above	acc_y_above	acc_z_above	acc_x_below	\
count	144036.000000	144036.000000	144036.000000	144036.000000	
mean	-0.101238	0.025862	9.802781	-0.283538	
std	1.290272	1.465464	2.079183	1.876397	
min	-8.855334	-9.940375	-8.353019	-16.605328	
25%	-0.786875	-0.693968	8.805027	-1.124420	
50%	-0.044672	0.091330	9.779467	-0.205046	
75%	0.568243	0.724596	10.779046	0.564690	
max	8.924007	13.259439	24.333819	14.990952	

	acc_y_below	acc_z_below	…	gyro_x_below	gyro_y_below	\
count	144036.000000	144036.000000	…	144036.000000	144036.000000	
mean	0.107041	9.786143	…	0.008825	-0.020688	
std	5.205432	6.165539	…	9.085235	29.477635	
min	-58.305765	-56.387895	…	-69.011507	-207.764217	
25%	-1.694867	7.221254	…	-4.344759	-14.496395	
50%	0.180990	9.804598	…	-0.011262	-0.015804	
75%	1.903618	12.356817	…	4.444304	14.098576	
max	69.228569	65.335711	…	122.944060	313.964299	

	gyro_z_below	temp_dash	temp_above	temp_below	\
count	144036.000000	144036.000000	144036.000000	144036.000000	
mean	0.121456	35.906430	36.402741	32.718136	
std	11.714743	1.065259	2.331418	0.930687	
min	-127.565293	33.364094	32.261868	30.249109	
25%	-4.518418	35.089316	34.178782	31.926408	
50%	0.059219	35.856082	36.479079	32.980711	
75%	4.728408	36.814539	38.300147	33.507862	
max	147.367568	38.156378	40.600443	34.705933	

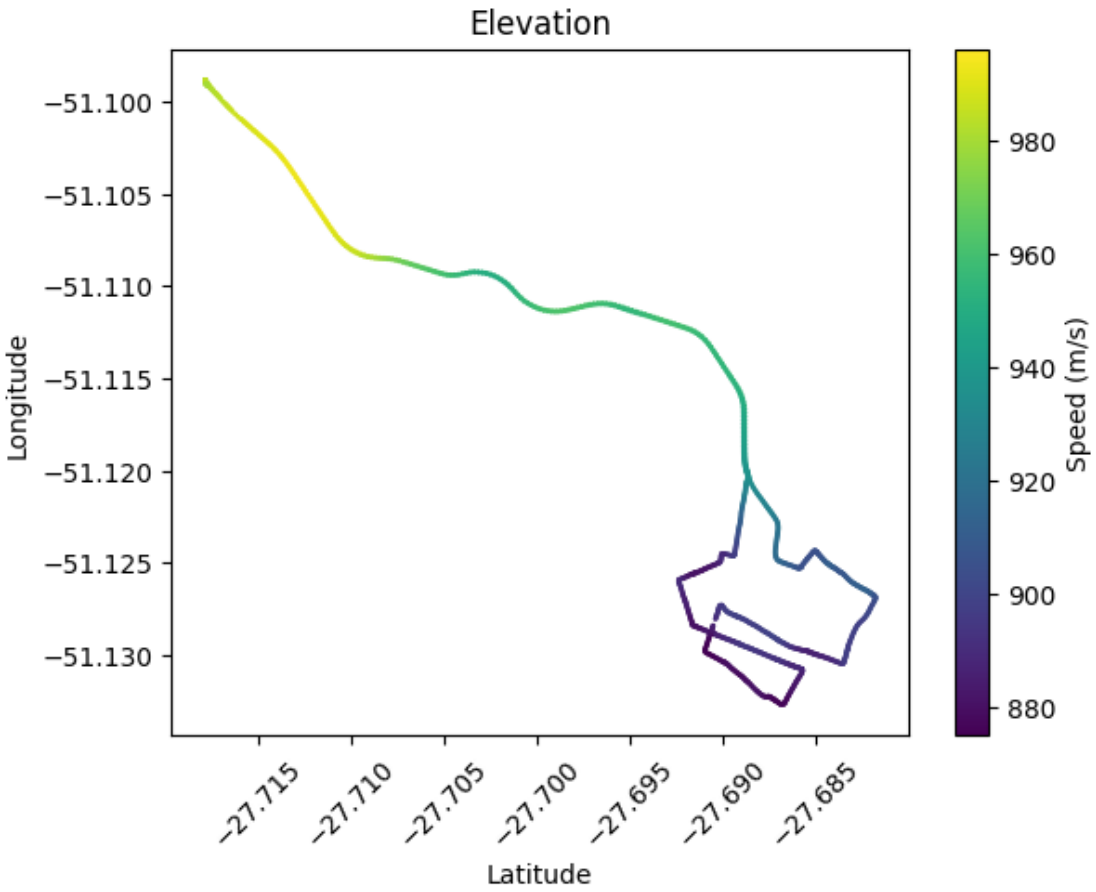
	timestamp_gps	latitude	longitude	speed
count	1.440360e+05	144036.000000	144036.000000	144036.000000

```
mean    1.577220e+09    -27.694558    -51.119821      9.556871
std     4.156529e+02      0.011344      0.011055      7.746386
min     1.577219e+09    -27.717842    -51.132691      0.002526
25%     1.577219e+09    -27.700008    -51.129011      4.508887
50%     1.577220e+09    -27.689739    -51.124897      6.618945
75%     1.577220e+09    -27.687054    -51.110978     16.647470
max     1.577220e+09    -27.681820    -51.098863     26.874480

[8 rows x 26 columns]
```

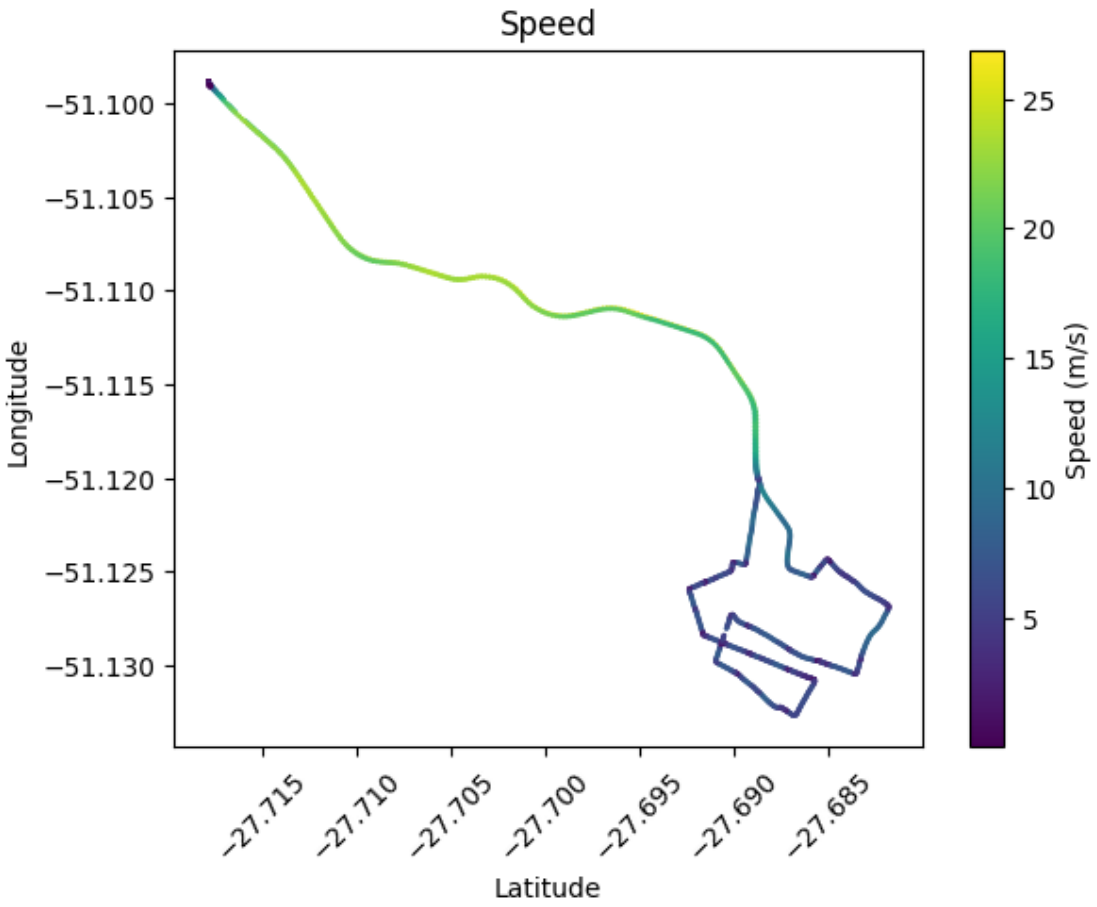
6 Plot Visualizations

6.1 Path

[ ]:
```
# plot path
plt.scatter(gps['latitude'], gps['longitude'], c=gps['elevation'],␣
 ↪cmap='viridis', s=1)
plt.colorbar(label='Speed (m/s)')  # Add color bar with label
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.xticks(rotation=45)
plt.title('Elevation')
plt.show()
```



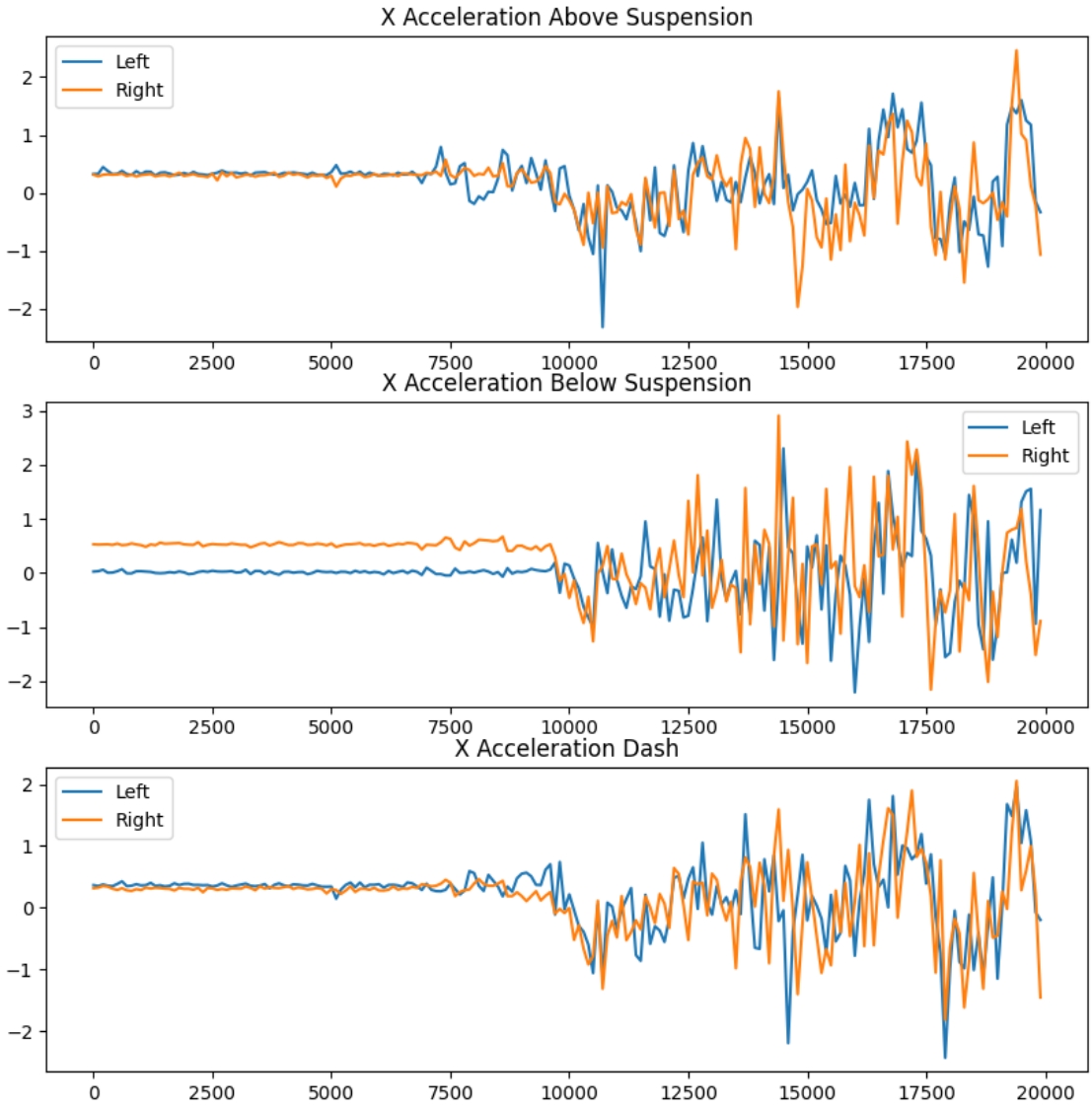
[ ]:
```
plt.scatter(gps['latitude'], gps['longitude'],␣
  ↪c=gps['speed_meters_per_second'], cmap='viridis', s=1)
plt.colorbar(label='Speed (m/s)')  # Add color bar with label
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.xticks(rotation=45)
plt.title('Speed')
plt.show()
```



6.2 Acceleration

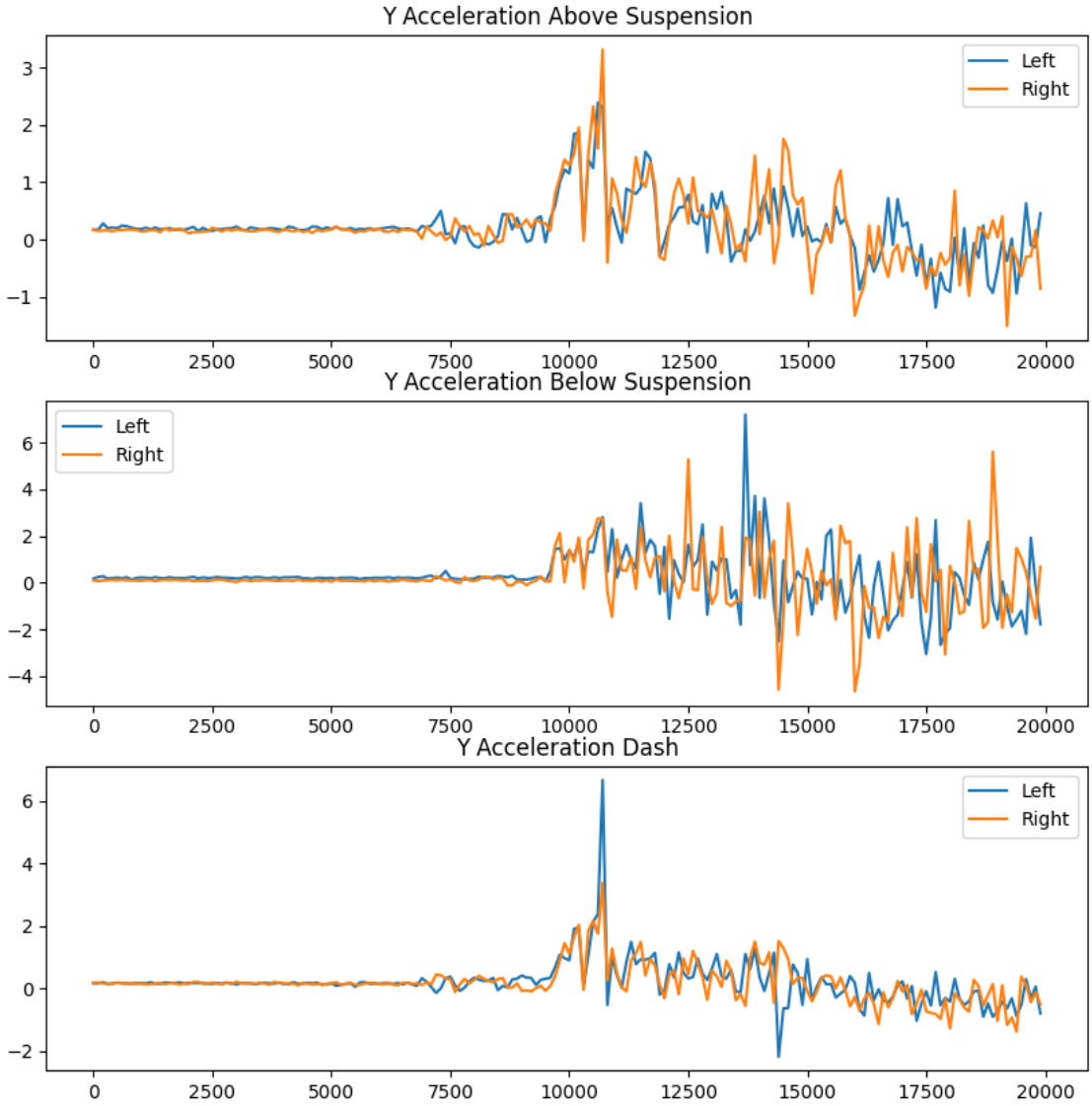
[ ]:
```
# plot x acceleration
plt.figure(figsize=(10, 10))
sep = 100
end = 20000
plt.subplot(3, 1, 1)
plt.plot(left_gps["acc_x_above"][:end][::sep])
plt.plot(right_gps["acc_x_above"][:end][::sep])
plt.title("X Acceleration Above Suspension")
plt.legend(["Left", "Right"])

plt.subplot(3, 1, 2)
plt.plot(left_gps["acc_x_below"][:end][::sep])
plt.plot(right_gps["acc_x_below"][:end][::sep])
plt.title("X Acceleration Below Suspension")
plt.legend(["Left", "Right"])
```

```
plt.subplot(3, 1, 3)
plt.plot(left_gps["acc_x_dash"][:end][::sep])
plt.plot(right_gps["acc_x_dash"][:end][::sep])
plt.title("X Acceleration Dash")
plt.legend(["Left", "Right"])
plt.show()
```



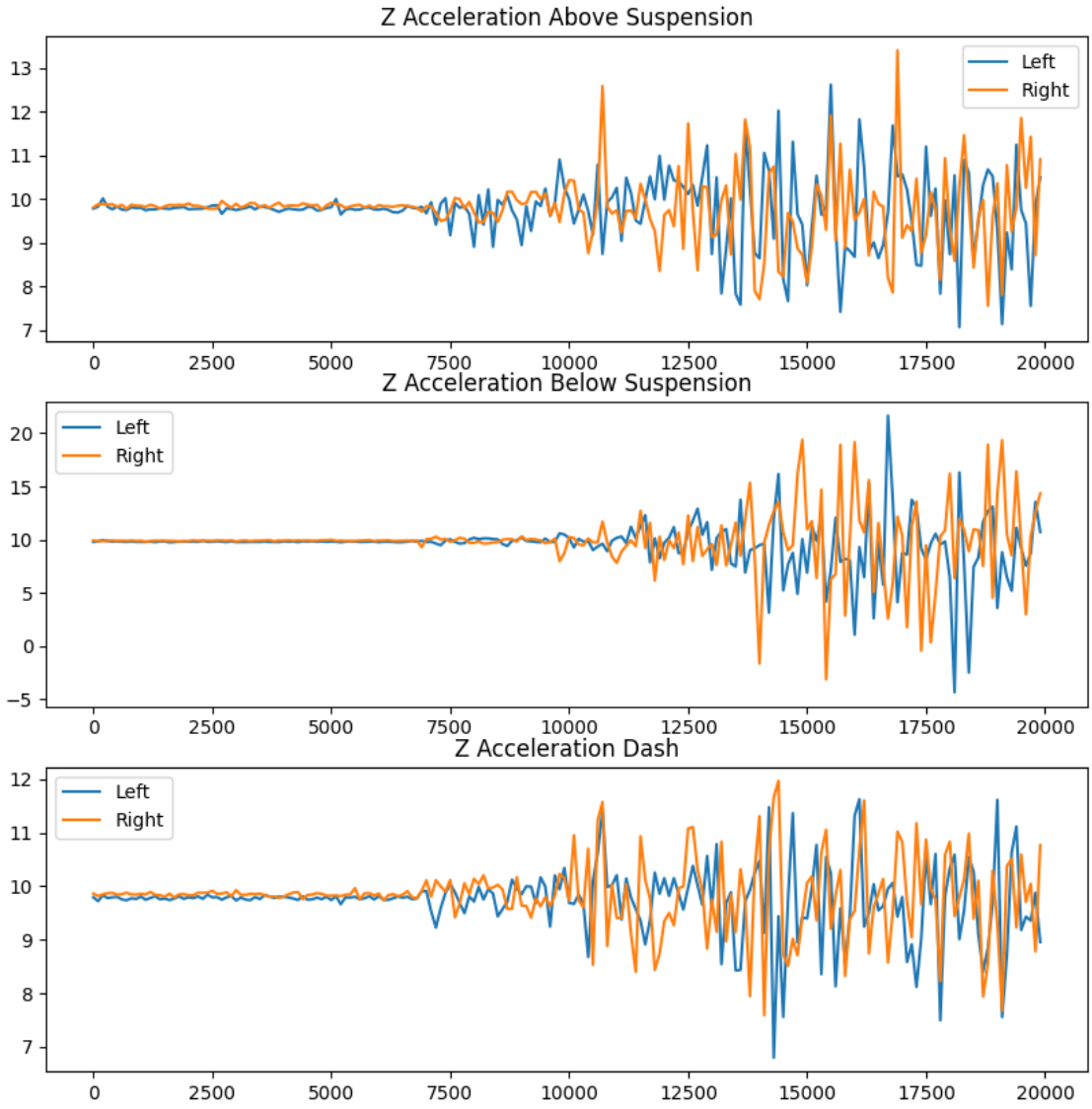
[ ]:
```
# plot y acceleration
plt.figure(figsize=(10, 10))
sep = 100
end = 20000
plt.subplot(3, 1, 1)
```

```
plt.plot(left_gps["acc_y_above"][:end][::sep])
plt.plot(right_gps["acc_y_above"][:end][::sep])
plt.title("Y Acceleration Above Suspension")
plt.legend(["Left", "Right"])

plt.subplot(3, 1, 2)
plt.plot(left_gps["acc_y_below"][:end][::sep])
plt.plot(right_gps["acc_y_below"][:end][::sep])
plt.title("Y Acceleration Below Suspension")
plt.legend(["Left", "Right"])

plt.subplot(3, 1, 3)
plt.plot(left_gps["acc_y_dash"][:end][::sep])
plt.plot(right_gps["acc_y_dash"][:end][::sep])
plt.title("Y Acceleration Dash")
plt.legend(["Left", "Right"])
plt.show()
```



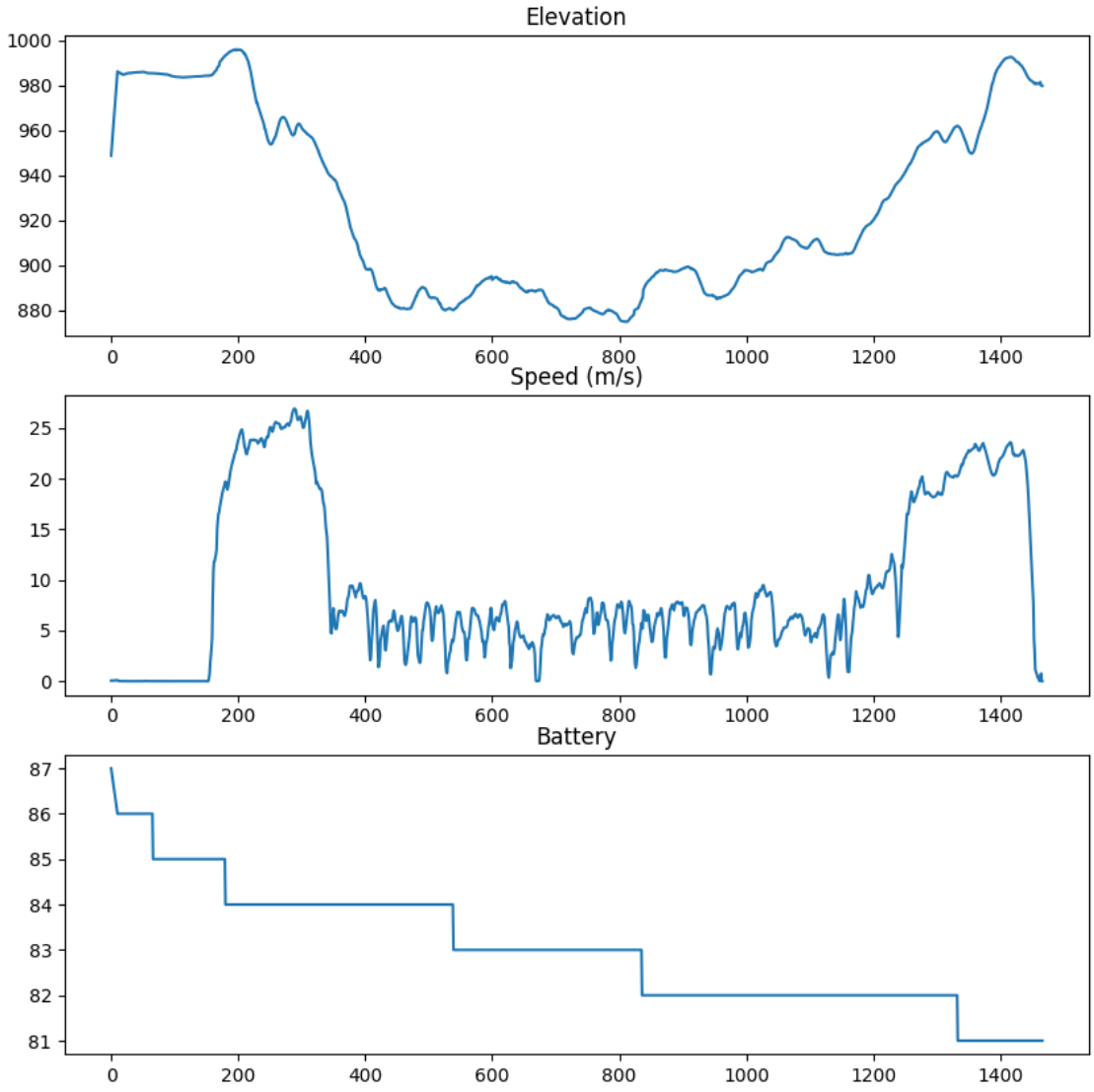
[ ]:
```
# plot y acceleration
plt.figure(figsize=(10, 10))
sep = 100
end = 20000
plt.subplot(3, 1, 1)
plt.plot(left_gps["acc_z_above"][:end][::sep])
plt.plot(right_gps["acc_z_above"][:end][::sep])
plt.title("Z Acceleration Above Suspension")
plt.legend(["Left", "Right"])

plt.subplot(3, 1, 2)
plt.plot(left_gps["acc_z_below"][:end][::sep])
```

```
plt.plot(right_gps["acc_z_below"][:end][::sep])
plt.title("Z Acceleration Below Suspension")
plt.legend(["Left", "Right"])

plt.subplot(3, 1, 3)
plt.plot(left_gps["acc_z_dash"][:end][::sep])
plt.plot(right_gps["acc_z_dash"][:end][::sep])
plt.title("Z Acceleration Dash")
plt.legend(["Left", "Right"])
plt.show()
```



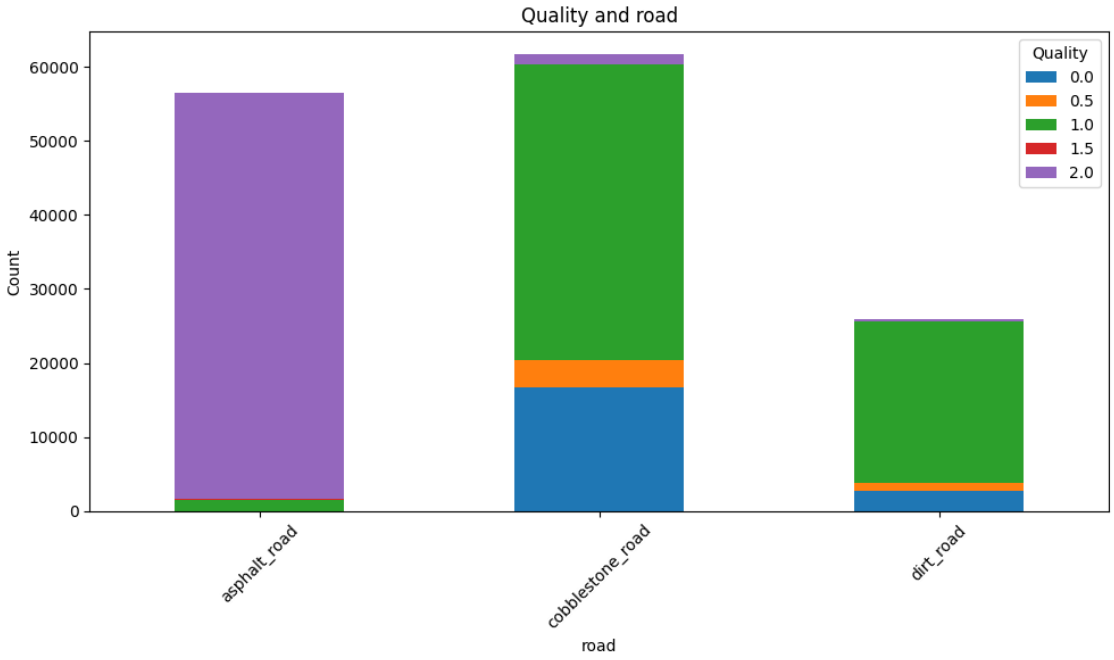
6.3 GPS Data

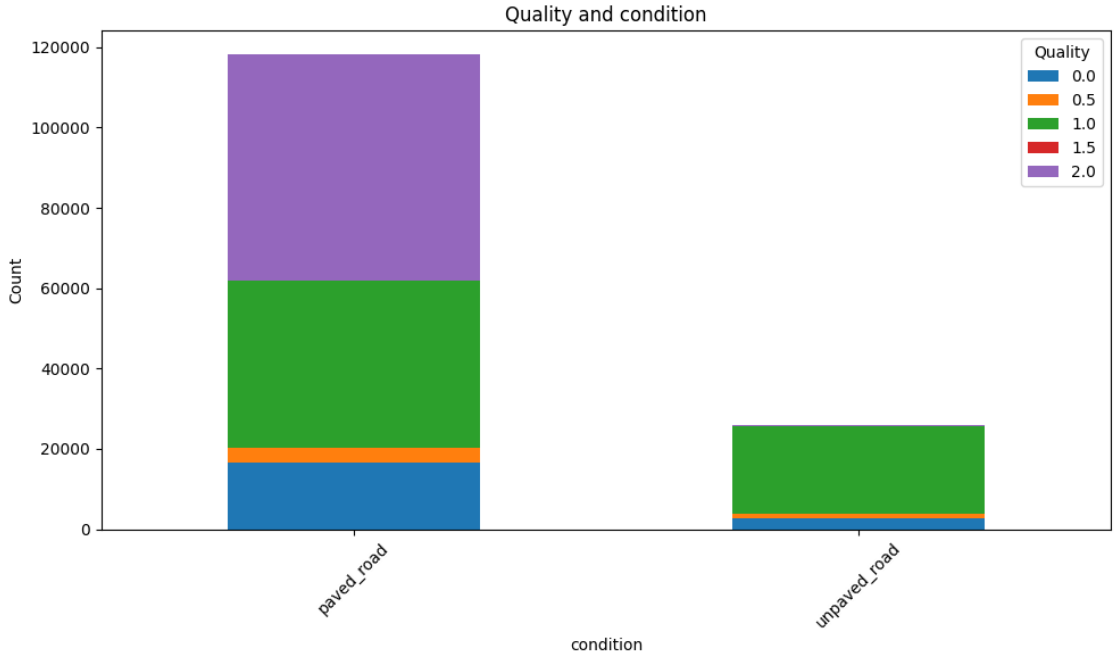
[ ]:
```
#set plot
plt.figure(figsize=(10, 10))
sep = 100
end = 20000

#plot elevation
plt.subplot(3, 1, 1)
plt.plot(gps["elevation"])
plt.title("Elevation")

#plot speed
plt.subplot(3, 1, 2)
plt.plot(gps["speed_meters_per_second"])
plt.title("Speed (m/s)")

#plot battery
plt.subplot(3, 1, 3)
plt.plot(gps["battery"])
plt.title("Battery")
plt.show()
```

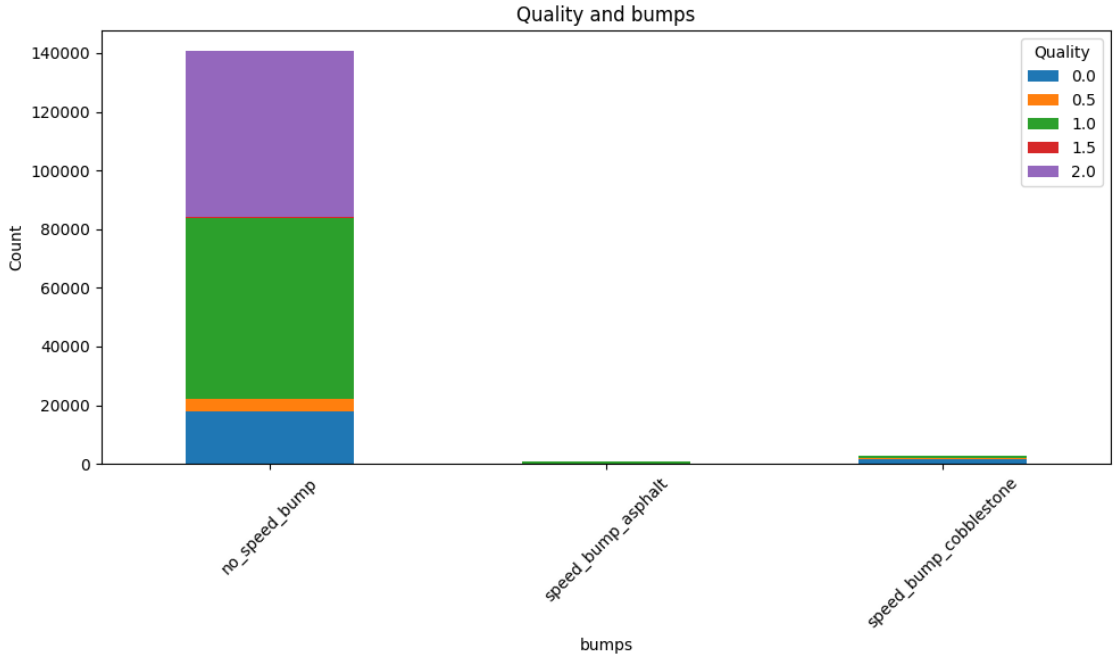


[ ]:
```
def plot_bar_chart(df, variable):
    # Group by 'quality' and the specified variable, and count occurrences
    grouped_data = df.groupby(['quality', variable]).size().
 ↪reset_index(name='count')

    # Plotting
    fig, ax = plt.subplots(figsize=(10, 6))
    grouped_data.pivot(index=variable, columns='quality', values='count').
 ↪plot(kind='bar', stacked=True, ax=ax)

    plt.title('Quality and '+variable)
    plt.xlabel(variable)
    plt.ylabel('Count')
```

```
    plt.xticks(rotation=45)
    plt.legend(title='Quality')
    plt.tight_layout()
    plt.show()



plot_bar_chart(label_names, 'road')
plot_bar_chart(label_names, 'condition')
plot_bar_chart(label_names, 'bumps')
```





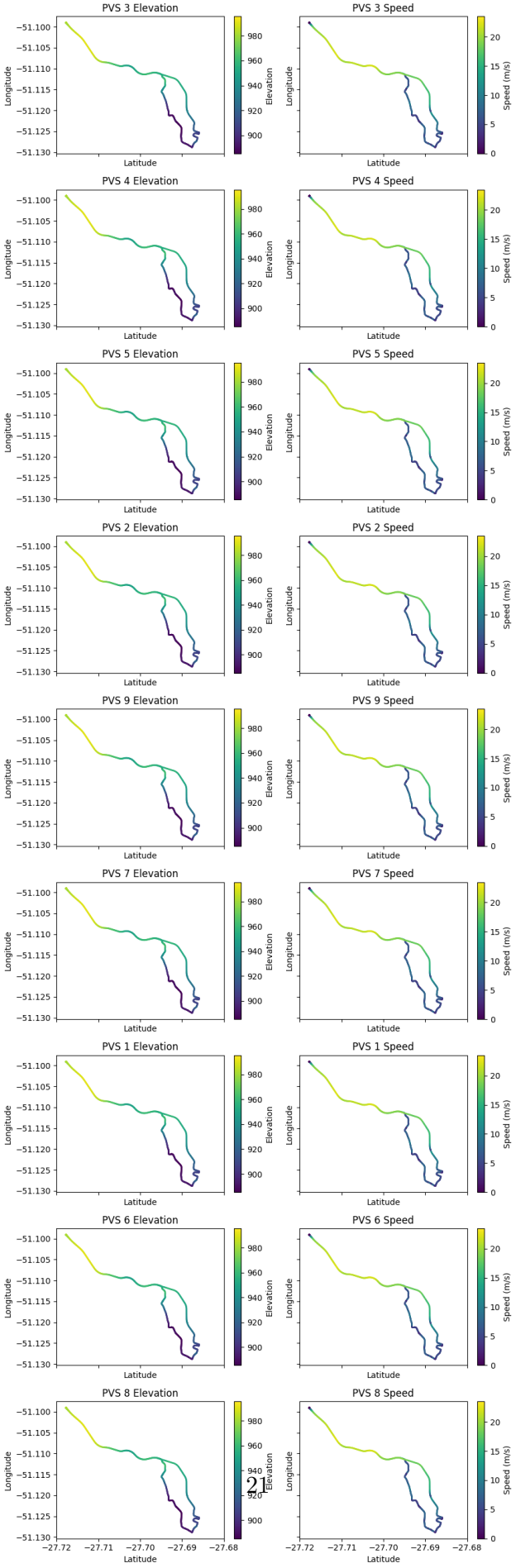


7 Subplots

Comparing the elevation, speed, and course of all 9 different drives.

[ ]:
```
path = '.data/'

# get a list of all the files in the directory
pvss = os.listdir(path)

# initialize subplots
fig, ax = plt.subplots(9,2, figsize=(9,27), sharey=True, sharex=True)
i = j = 0

csvs = []
for pvs in pvss:
    files = os.listdir(os.path.join(path, pvs))
    for file in files:
        if file.endswith(".csv"):
            csvs.append(os.path.join(path, pvs, file))

    # assign labels
    gps = pd.read_csv(csvs[0])
    left_gps = pd.read_csv(csvs[1])
    right_gps = pd.read_csv(csvs[2])
    df_labels = pd.read_csv(csvs[3])

    # plot path with elevation
    sc1 = ax[i,j].scatter(gps['latitude'], gps['longitude'],␣
 ↪c=gps['elevation'], cmap='viridis', s=1)
    cbar1 = plt.colorbar(sc1, ax=ax[i,j])
    cbar1.set_label('Elevation')
    ax[i,j].set_xlabel('Latitude')
    ax[i,j].set_ylabel('Longitude')
    ax[i,j].set_xticks(ax[i,j].get_xticks(), rotation=45)
    ax[i,j].set_title(f'{pvs} Elevation')
    j += 1

    # plot plath with speed
    sc2 = ax[i,j].scatter(gps['latitude'], gps['longitude'],␣
 ↪c=gps['speed_meters_per_second'], cmap='viridis', s=1)
    cbar2 = plt.colorbar(sc2, ax=ax[i,j])
    cbar2.set_label('Speed (m/s)')
    ax[i,j].set_xlabel('Latitude')
    ax[i,j].set_ylabel('Longitude')
    ax[i,j].set_xticks(ax[i,j].get_xticks(), rotation=45)
    ax[i,j].set_title(f'{pvs} Speed')

    i += 1
    j = 0

plt.tight_layout()
```

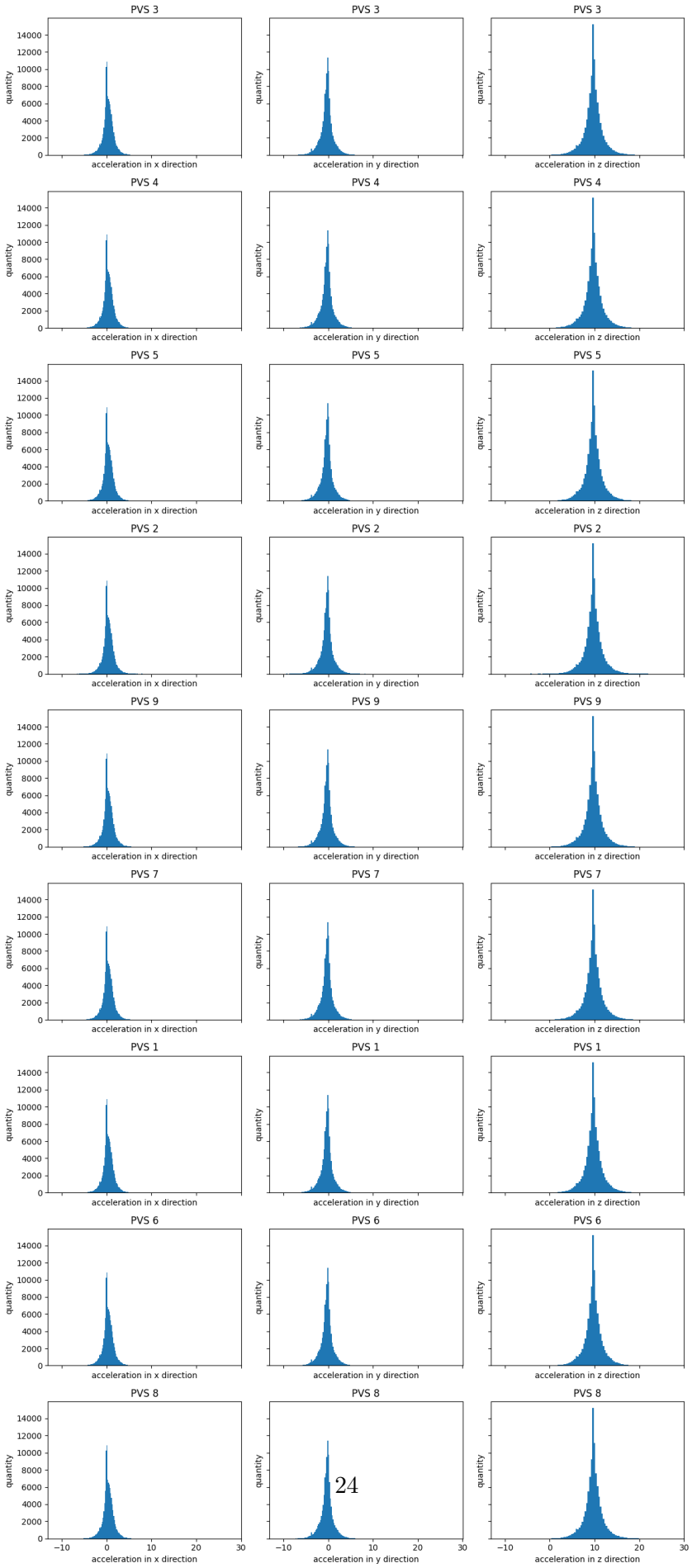
```
plt.show()
```


PVS 3 Elevation
PVS 3 Speed
PVS 4 Elevation
PVS 4 Speed
PVS 5 Elevation
PVS 5 Speed
PVS 2 Elevation
PVS 2 Speed
PVS 9 Elevation
PVS 9 Speed
PVS 7 Elevation
PVS 7 Speed
PVS 1 Elevation
PVS 1 Speed
PVS 6 Elevation
PVS 6 Speed
PVS 8 Elevation
PVS 8 Speed
Longitude
Latitude
Elevation
Speed (m/s)
−51.100
−51.105
−51.110
−51.115
−51.120
−51.125
−51.130
980
960
940
920
900
20
15
10
5
0
−27.72
−27.71
−27.70
−27.69
−27.68

A histogram of acceleration data in the x, y, and z direction of all 9 different drives.

[ ]:
```
path = '.data/'

# get a list of all the files in the directory
pvss = os.listdir(path)

# initialize subplots
fig, ax = plt.subplots(9,3, figsize=(12,27), sharey=True, sharex=True)
i = j = 0

csvs = []
for pvs in pvss:
    files = os.listdir(os.path.join(path, pvs))
    for file in files:
        if file.endswith(".csv"):
            csvs.append(os.path.join(path, pvs, file))

    # assign labels
    gps = pd.read_csv(csvs[0])
    left_gps = pd.read_csv(csvs[1])
    right_gps = pd.read_csv(csvs[2])
    df_labels = pd.read_csv(csvs[3])

    left_gps = clean_acc(left_gps)
    right_gps = clean_acc(right_gps)

    # plot acc in x label
    ax[i,j].hist(left_gps["acc_x_above"], bins=100)
    ax[i,j].set_xlabel('acceleration in x direction')
    ax[i,j].set_ylabel('quantity')
    ax[i,j].set_title(f'{pvs}')
    j += 1

    # plot acc in x label
    ax[i,j].hist(left_gps["acc_y_above"], bins=100)
    ax[i,j].set_xlabel('acceleration in y direction')
    ax[i,j].set_ylabel('quantity')
    ax[i,j].set_title(f'{pvs}')
    j += 1

    # plot acc in x label
    ax[i,j].hist(left_gps["acc_z_above"], bins=100)
    ax[i,j].set_xlabel('acceleration in z direction')
    ax[i,j].set_ylabel('quantity')
```

```
    ax[i,j].set_title(f'{pvs}')

    i += 1
    j = 0

plt.tight_layout()
plt.show()
```


PVS 3
PVS 3
PVS 3
PVS 4
PVS 4
PVS 4
PVS 5
PVS 5
PVS 5
PVS 2
PVS 2
PVS 2
PVS 9
PVS 9
PVS 9
PVS 7
PVS 7
PVS 7
PVS 1
PVS 1
PVS 1
PVS 6
PVS 6
PVS 6
PVS 8
PVS 8
PVS 8
quantity
0
2000
4000
6000
8000
10000
12000
14000
acceleration in x direction
acceleration in y direction
acceleration in z direction
−10
0
10
20
30

[ ]: