



Data-driven Design and Analyses of Structures and Materials (3dasm)

Lecture 2

Miguel A. Bessa | M.A.Bessa@tudelft.nl | Associate Professor

OPTION 1. Run this notebook **locally** in your computer:

1. Confirm that you have the 3dasm conda environment (see Lecture 1).
2. Go to the 3dasm_course folder in your computer and pull the last updates of the **repository**:

```
git pull
```

3. Open command window and load jupyter notebook (it will open in your internet browser):

```
conda activate 3dasm  
jupyter notebook
```

4. Open notebook of this Lecture.

OPTION 2. Use Google's Colab (no installation required, but times out if idle):

1. go to <https://colab.research.google.com>
2. login
3. File > Open notebook
4. click on Github (no need to login or authorize anything)
5. paste the git link: https://github.com/bessagroup/3dasm_course
6. click search and then click on the notebook for this Lecture.

```
In [1]:
```

```
# Basic plotting tools needed in Python.

import matplotlib.pyplot as plt # import plotting tools to create figures
import numpy as np # import numpy to handle a lot of things!

%config InlineBackend.figure_format = "retina" # render higher resolution images in the notebook
plt.style.use("seaborn") # style for plotting that comes from seaborn
plt.rcParams["figure.figsize"] = (8,4) # rescale figure size appropriately for slides
```

Outline for today

- Handling data and handling pandas!
- Application of knowledge gained in Lecture 1
- Understanding the governing model for the car stopping distance problem

Reading material: This notebook

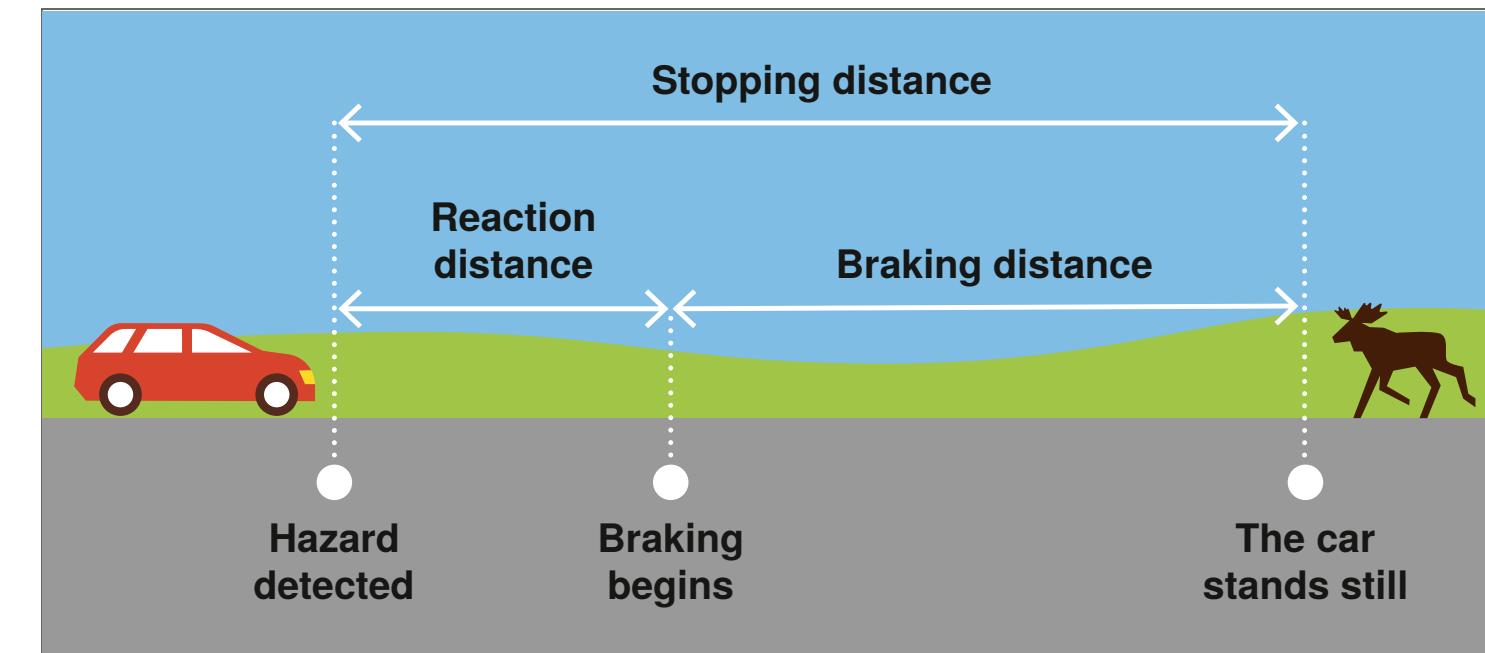
The car stopping distance problem (again!)

Imagine we want to predict y for a given x but that **we had no idea** that this problem is governed by:

$$y = zx + 0.1x^2$$

- y is the **output**: the car stopping distance
- x is the **input**: the car velocity
- z is a hidden variable: an **rv** z representing the driver's reaction time (in seconds)

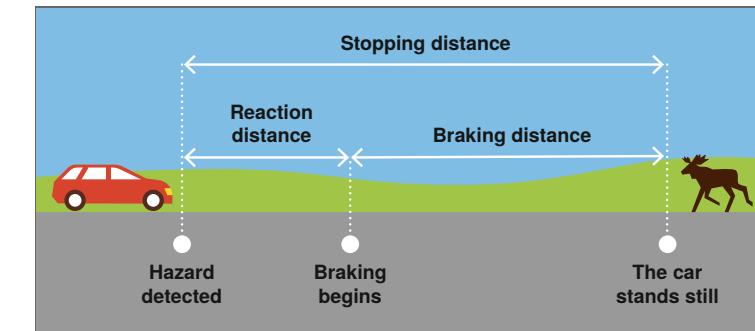
where $z \sim \mathcal{N}(\mu_z = 1.5, \sigma_z^2 = 0.5^2)$



The car stopping distance problem

Instead, you are just provided with the data \mathcal{D} that contains N measurements of different people stopping the car.

How does the data \mathcal{D} look like?



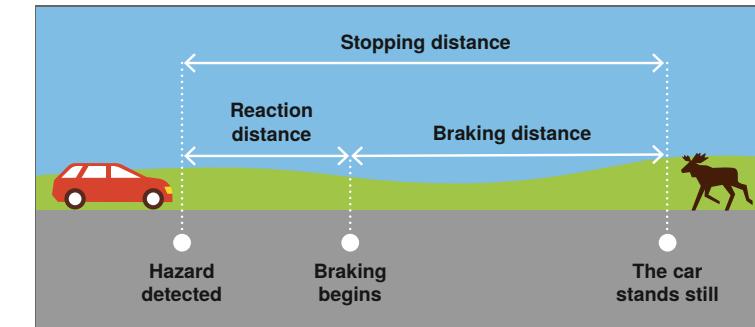
A table with N rows and 2 columns (input $x = \mathcal{D}_x$ and output $y = \mathcal{D}_y$).

| x (m/s) | y (m) |
|---------|-------|
| 15.5 | 21.4 |
| 5.3 | 11.8 |
| 78.0 | 701.6 |
| ... | ... |
| 10.8 | 22.1 |

Note that z is nowhere to be found! We could only measure the velocity of the car and the stopping distance, but we could not measure how quickly each driver reacts to seeing the deer!

Data generation

However, it's more instructive if we *create* the data from the governing equation. This is more interesting than just being provided with the data!



- Usually, in ML literature people don't create the data...
 - Data is collected from somewhere (Internet?) and we don't have a clue about the hidden variables involved in the data generation process (unknown causes that explain the data!).
- **But...** In engineering practice this can be different! Sometimes you *can create data* (e.g. from computer simulations, from conducting your own experiments, etc.)
 - Although, models/measurements are never perfect (errors) and can be stochastic (noise)!

Anyway, let's create the dataset for our problem...

In [2]:

```
from scipy.stats import norm # import the normal dist, as we learned before!
# Define our car stopping distance function
def y(x):
    z = norm.rvs(1.5, 0.5, size=1) # randomly draw 1 sample from the normal dist.
    y = z*x + 0.1*x**2 # compute the stopping distance
    return y

N = 33 # number of points to generate data
Data_x = np.linspace(3, 83, N) # generate a dataset with N points for velocities x between 3 and 83 m/s
print("Let's see the Data_x vector:\n", Data_x)

Data_y = np.zeros_like(Data_x) # create an empty vector for Data_y with same size as Data_x
i = 0 # initialize the counter
for x_i in Data_x:
    Data_y[i] = y(x_i)
    i += 1

print("\nLet's see the Data_y vector:\n", Data_y)
# IMPORTANT NOTE: Every time you run this cell you generate a new Data_y vector because of the z rv!!!
```

Let's see the Data_x vector:

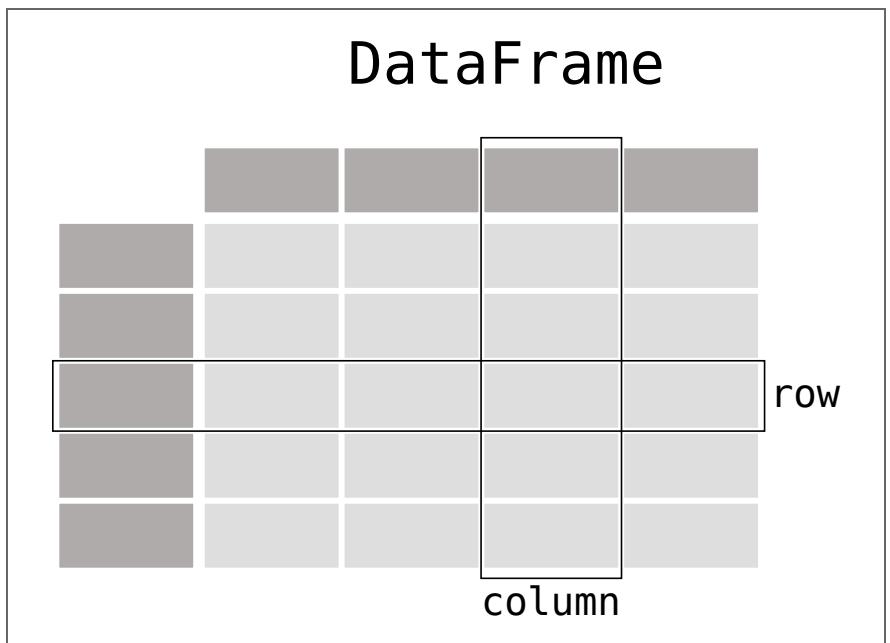
```
[ 3.   5.5   8.  10.5  13.  15.5  18.  20.5  23.  25.5  28.  30.5  33.  35.5
 38.  40.5  43.  45.5  48.  50.5  53.  55.5  58.  60.5  63.  65.5  68.  70.5
 73.  75.5  78.  80.5  83. ]
```

Let's see the Data_y vector:

```
[ 6.6228835  10.3905087  17.96659493  23.42966362  30.89579891
 45.59685527  49.16011657  85.08081813  76.72897378  99.82697749
 109.74056361 123.8578289  153.95711001 179.5269188  187.94020114
 219.33589832 297.37101301 281.08247987 284.48886732 302.39922853
 380.32715662 416.50202633 387.79197732 473.16796893 442.29438313
 518.71985399 587.9409602  651.66160338 631.27248144 713.785457
 711.72178846 789.99906168 740.44892466]
```

Say hello to the pandas!

pandas is an open source library that is very common to handle data in ML
Let's create a pandas dataframe \mathcal{D} that includes both the input \mathcal{D}_x and output \mathcal{D}_y that we created.



In [3]:

```
import pandas as pd # Pandas dataframe (very common in ML)
#
# Create a dictionary with the names of the input and output variables
# and their respective values:
input_dictionary = {
    'x' : Data_x, # vector with input values of x
    'y' : Data_y, # vector with output values of y
}
#
# Then, we create a Pandas data frame:
car_df = pd.DataFrame(input_dictionary)

print(car_df)
```

| | x | y |
|----|------|------------|
| 0 | 3.0 | 6.622883 |
| 1 | 5.5 | 10.390509 |
| 2 | 8.0 | 17.966595 |
| 3 | 10.5 | 23.429664 |
| 4 | 13.0 | 30.895799 |
| 5 | 15.5 | 45.596855 |
| 6 | 18.0 | 49.160117 |
| 7 | 20.5 | 85.080818 |
| 8 | 23.0 | 76.728974 |
| 9 | 25.5 | 99.826977 |
| 10 | 28.0 | 109.740564 |
| 11 | 30.5 | 123.857829 |
| 12 | 33.0 | 153.957110 |
| 13 | 35.5 | 179.526919 |
| 14 | 38.0 | 187.940201 |
| 15 | 40.5 | 219.335898 |
| 16 | 43.0 | 297.371013 |
| 17 | 45.5 | 281.082480 |
| 18 | 48.0 | 284.488867 |
| 19 | 50.5 | 302.399229 |

| | | |
|----|------|------------|
| 20 | 53.0 | 380.327157 |
| 21 | 55.5 | 416.502026 |
| 22 | 58.0 | 387.791977 |
| 23 | 60.5 | 473.167969 |
| 24 | 63.0 | 442.294383 |
| 25 | 65.5 | 518.719854 |
| 26 | 68.0 | 587.940960 |
| 27 | 70.5 | 651.661603 |
| 28 | 73.0 | 631.272481 |
| 29 | 75.5 | 713.785457 |
| 30 | 78.0 | 711.721788 |
| 31 | 80.5 | 789.999062 |
| 32 | 83.0 | 740.448925 |

Say hello to the pandas!

The **pandas** DataFrame we created is quite simple!

Today we won't go through many of the powerful data handling features of pandas.

We will try to introduce things as we need them.

- A basic need: save your data!



```
In [4]:  
# Saving our "car_df" pandas dataframe to the "docs" folder in our computer is quite simple!  
car_df.to_pickle("docs/car_dataframe.pkl") # done!
```

Go check in the "docs" folder if you have created the "car_dataframe.pkl" file!

Load data from someone else

As mentioned, many ML problems start in a different way:

- We are provided with a dataset in some format.
 - For example, someone else conducted the car stopping distance experiment and saved the measurements in a [csv](#) file called "**data_for_car_prob.csv**".
 - Then we receive the file and have to use it in our analysis!

We have that file in our "docs" folder. Go there and try to open it!

- Let's learn how to import this [csv](#) file data into a pandas dataframe.

```
In [5]:
```

```
    car_csv_data = pd.read_csv("docs/data_for_car_prob.csv") # read csv data provided by someone else
print(car_csv_data)
```

| | Unnamed: 0 | x | y |
|----|------------|-----------|------------|
| 0 | 0 | 9.516939 | 29.749036 |
| 1 | 1 | 72.398757 | 642.132203 |
| 2 | 2 | 17.950326 | 36.648484 |
| 3 | 3 | 9.440853 | 18.604106 |
| 4 | 4 | 78.791008 | 769.656168 |
| 5 | 5 | 16.961121 | 57.971010 |
| 6 | 6 | 65.410368 | 559.093313 |
| 7 | 7 | 58.671099 | 463.686613 |
| 8 | 8 | 21.550603 | 92.242676 |
| 9 | 9 | 36.866913 | 197.688573 |
| 10 | 10 | 15.728748 | 56.885233 |
| 11 | 11 | 58.511494 | 388.753795 |
| 12 | 12 | 57.419190 | 399.807488 |
| 13 | 13 | 38.459157 | 213.181519 |
| 14 | 14 | 8.841742 | 20.387384 |
| 15 | 15 | 60.733051 | 516.341724 |
| 16 | 16 | 49.256663 | 307.931956 |
| 17 | 17 | 35.895121 | 181.123049 |
| 18 | 18 | 79.195652 | 750.178284 |
| 19 | 19 | 69.156669 | 553.153541 |
| 20 | 20 | 77.634896 | 746.031880 |
| 21 | 21 | 9.254011 | 20.810698 |
| 22 | 22 | 15.451468 | 39.872527 |
| 23 | 23 | 14.438247 | 42.118771 |
| 24 | 24 | 13.410999 | 44.775122 |
| 25 | 25 | 53.747057 | 375.013937 |

| | | | |
|----|----|-----------|------------|
| 26 | 26 | 10.283719 | 19.438868 |
| 27 | 27 | 82.005477 | 742.336845 |
| 28 | 28 | 81.805562 | 706.620282 |
| 29 | 29 | 51.837742 | 345.212876 |
| 30 | 30 | 20.283785 | 65.303165 |
| 31 | 31 | 28.359647 | 155.185137 |
| 32 | 32 | 74.993715 | 676.628982 |
| 33 | 33 | 21.827564 | 81.150935 |
| 34 | 34 | 70.519111 | 700.520033 |
| 35 | 35 | 74.208532 | 622.453560 |
| 36 | 36 | 14.518958 | 40.927570 |
| 37 | 37 | 13.357644 | 39.770922 |
| 38 | 38 | 75.346253 | 707.973754 |
| 39 | 39 | 44.923956 | 251.300805 |
| 40 | 40 | 26.801159 | 124.098654 |
| 41 | 41 | 29.906265 | 118.100900 |
| 42 | 42 | 40.226356 | 215.082100 |
| 43 | 43 | 66.282662 | 537.845048 |
| 44 | 44 | 47.342777 | 308.558833 |
| 45 | 45 | 3.087674 | 5.947997 |
| 46 | 46 | 21.254611 | 101.295276 |
| 47 | 47 | 46.939484 | 345.778352 |
| 48 | 48 | 38.875692 | 219.095582 |
| 49 | 49 | 76.705452 | 742.720134 |

We see that one of the columns is redundant!

Let's delete that one

```
In [6]:
```

```
car_prob_df = car_csv_data[['x','y']] # pandas has a fast way to select the
# columns of interest (in this case 'x' and 'y')
print(car_prob_df)
```

| | x | y |
|----|-----------|------------|
| 0 | 9.516939 | 29.749036 |
| 1 | 72.398757 | 642.132203 |
| 2 | 17.950326 | 36.648484 |
| 3 | 9.440853 | 18.604106 |
| 4 | 78.791008 | 769.656168 |
| 5 | 16.961121 | 57.971010 |
| 6 | 65.410368 | 559.093313 |
| 7 | 58.671099 | 463.686613 |
| 8 | 21.550603 | 92.242676 |
| 9 | 36.866913 | 197.688573 |
| 10 | 15.728748 | 56.885233 |
| 11 | 58.511494 | 388.753795 |
| 12 | 57.419190 | 399.807488 |
| 13 | 38.459157 | 213.181519 |
| 14 | 8.841742 | 20.387384 |
| 15 | 60.733051 | 516.341724 |
| 16 | 49.256663 | 307.931956 |
| 17 | 35.895121 | 181.123049 |
| 18 | 79.195652 | 750.178284 |
| 19 | 69.156669 | 553.153541 |
| 20 | 77.634896 | 746.031880 |
| 21 | 9.254011 | 20.810698 |
| 22 | 15.451468 | 39.872527 |
| 23 | 14.438247 | 42.118771 |
| 24 | 13.410999 | 44.775122 |

| | | |
|----|-----------|------------|
| 25 | 53.747057 | 375.013937 |
| 26 | 10.283719 | 19.438868 |
| 27 | 82.005477 | 742.336845 |
| 28 | 81.805562 | 706.620282 |
| 29 | 51.837742 | 345.212876 |
| 30 | 20.283785 | 65.303165 |
| 31 | 28.359647 | 155.185137 |
| 32 | 74.993715 | 676.628982 |
| 33 | 21.827564 | 81.150935 |
| 34 | 70.519111 | 700.520033 |
| 35 | 74.208532 | 622.453560 |
| 36 | 14.518958 | 40.927570 |
| 37 | 13.357644 | 39.770922 |
| 38 | 75.346253 | 707.973754 |
| 39 | 44.923956 | 251.300805 |
| 40 | 26.801159 | 124.098654 |
| 41 | 29.906265 | 118.100900 |
| 42 | 40.226356 | 215.082100 |
| 43 | 66.282662 | 537.845048 |
| 44 | 47.342777 | 308.558833 |
| 45 | 3.087674 | 5.947997 |
| 46 | 21.254611 | 101.295276 |
| 47 | 46.939484 | 345.778352 |
| 48 | 38.875692 | 219.095582 |
| 49 | 76.705452 | 742.720134 |

This looks better.

In [7]:

```
Data_x = car_prob_df['x'].values # select the input VALUES from your dataframe into Data_x
Data_y = car_prob_df['y'].values # select the output VALUES from your dataframe into Data_y
print("Data_x is:\n",Data_x)
print("\nData_y is:\n",Data_y)
```

Data_x is:

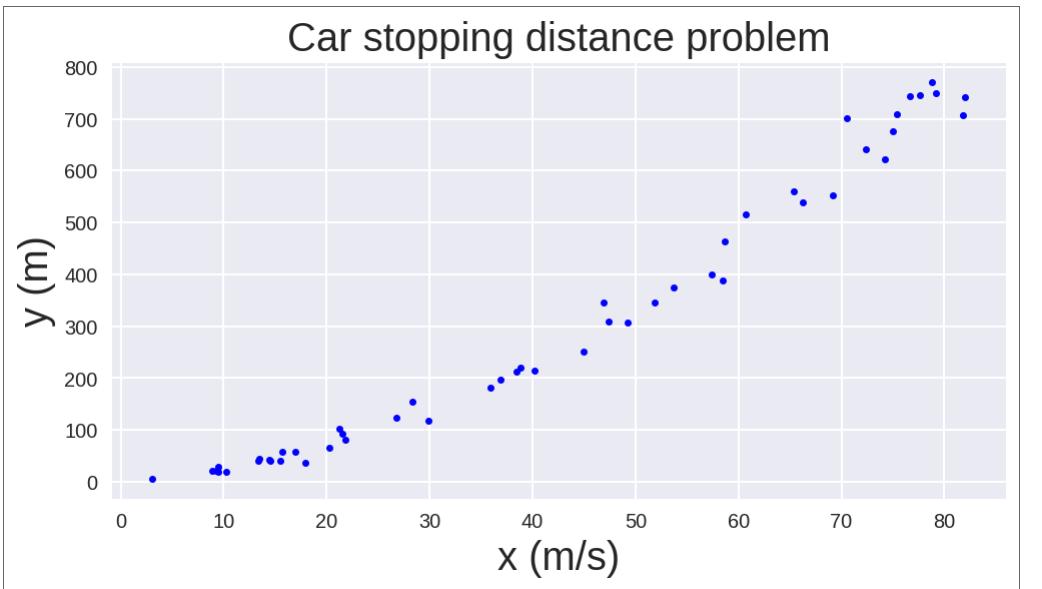
```
[ 9.51693942 72.39875748 17.95032583  9.44085299 78.79100778 16.96112056
 65.4103675 58.67109927 21.55060313 36.86691294 15.72874781 58.51149357
 57.41918959 38.45915667  8.84174221 60.73305107 49.25666345 35.89512052
 79.19565172 69.15666925 77.63489641  9.25401128 15.45146824 14.43824684
 13.41099874 53.74705712 10.28371886 82.00547705 81.80556249 51.8377421
 20.28378484 28.35964692 74.99371524 21.82756352 70.51911096 74.20853195
 14.51895792 13.35764354 75.34625316 44.92395642 26.80115926 29.90626522
 40.22635624 66.28266205 47.34277718  3.08767411 21.25461134 46.93948443
 38.87569199 76.70545196]
```

Data_y is:

```
[ 29.74903647 642.13220315  36.64848446 18.60410602 769.65616843
 57.97101034 559.09331318 463.68661322 92.24267632 197.68857288
 56.88523327 388.75379474 399.80748803 213.18151905 20.38738432
 516.34172363 307.93195589 181.12304936 750.17828361 553.15354059
 746.03187971 20.81069833 39.87252654 42.11877078 44.77512244
 375.01393668 19.43886782 742.33684483 706.62028237 345.21287569
 65.30316533 155.18513747 676.62898211 81.15093549 700.52003305
 622.45356019 40.92757044 39.77092163 707.97375405 251.30080489
 124.09865438 118.10089977 215.08209978 537.84504756 308.55883254
 5.94799685 101.29527607 345.77835213 219.09558165 742.72013356]
```

In [8]:

```
fig_car_data, ax_car_data = plt.subplots() # create a plot
ax_car_data.plot(Data_x, Data_y, 'b.')
ax_car_data.set_xlabel("x (m/s)", fontsize=20) # create x-axis label with font size 20
ax_car_data.set_ylabel("y (m)", fontsize=20) # create y-axis label with font size 20
ax_car_data.set_title("Car stopping distance problem", fontsize=20); # create title with font size 20
```



In-class Exercise 1

Plot the function $x \sin(x)$ in the domain $x \in [0, 10]$.

Here's a little reminder of the steps to do this:

1. Define the function $f(x) = x \sin(x)$.
2. Create a vector "x_data" of 50 points that are uniformly spaced between 0 and 10.
3. Calculate the values of $f(x) = x \sin(x)$ for each of the 50 points of vector "x_data" and save them in "y_data".
4. Plot the function from the 50 points that you defined, label the x-axis as "x" and the y axis as "y", and include a title "Exercise 1" in the plot.
5. Create a pandas dataframe called "my_df" with the data, and save it in a pickle file called "exercise1_data.pkl" that needs to be in the "docs" folder of this lecture.

```
In [9]:
```

```
# Exercise 1.  
# Write your code here:  
# until here.
```

In-class Exercise 2

Derive the expected value and variance for the governing model for the car stopping distance problem:

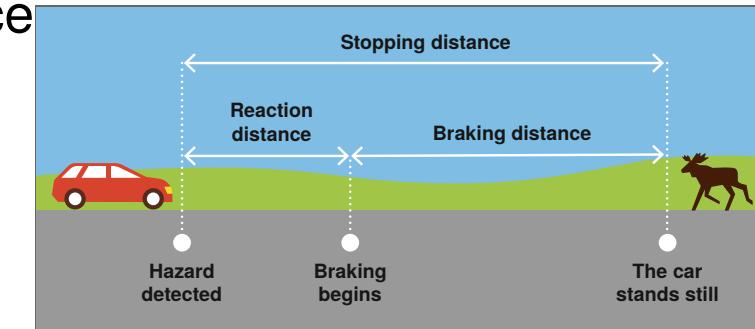
$$\mathbb{E}[y] = ?$$

$$\mathbb{V}[y] = ?$$

(In case you haven't seen enough of the problem statement 😊) recall that:

$$y = zx + 0.1x^2$$

where $z \sim \mathcal{N}(\mu_z = 1.5, \sigma_z^2 = 0.5^2)$

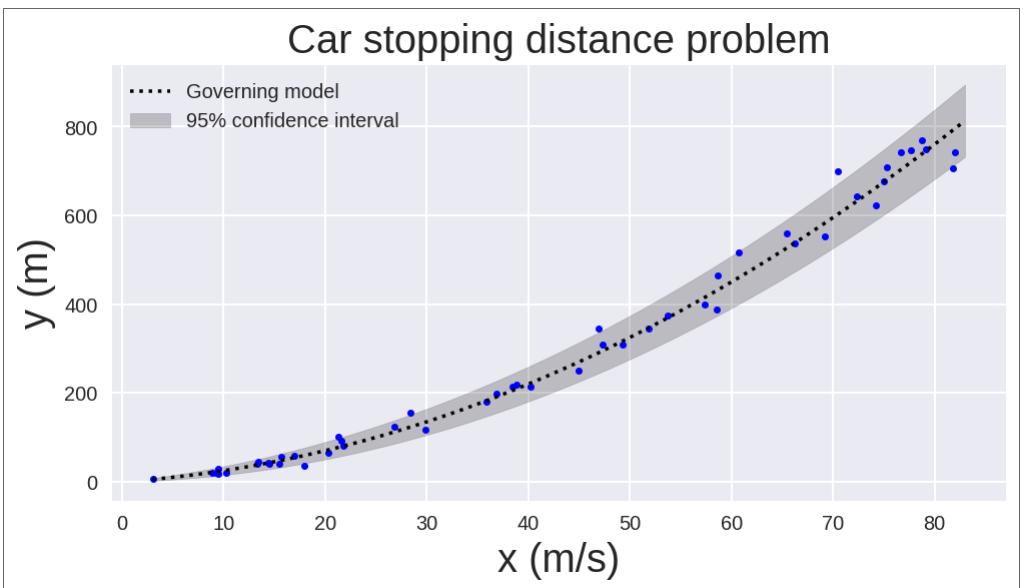


In [10]:

```
# Let's plot the model (we pretend for a moment that we know it!)
real_x = np.linspace(3, 83, 1000)
real_mu_y = 1.5*real_x + 0.1*(real_x**2) # Recall: E[z*x+0.1*x^2] = E[z]*E[x]+0.1*E[x^2]
real_sigma_y = np.sqrt( 0.5**2*real_x**2 ) # Recall: V[z*x+x^2] = V[z*x]+V[x^2]=...=sigma_z^2*mu_x^2

ax_car_data.plot(real_x, real_mu_y, 'k:', label="Governing model")
ax_car_data.fill_between(real_x, real_mu_y - 1.9600 * real_sigma_y,
                        real_mu_y + 1.9600 * real_sigma_y,
                        color='k', alpha=0.2,
                        label='95% confidence interval') # See book Sections 4.7.4 and 4.7.5 for a discussion on CI vs credible interval
ax_car_data.legend()
fig_car_data
```

Out[10]:



Question: If we didn't know the governing model, what should ML do using the data (blue dots)?

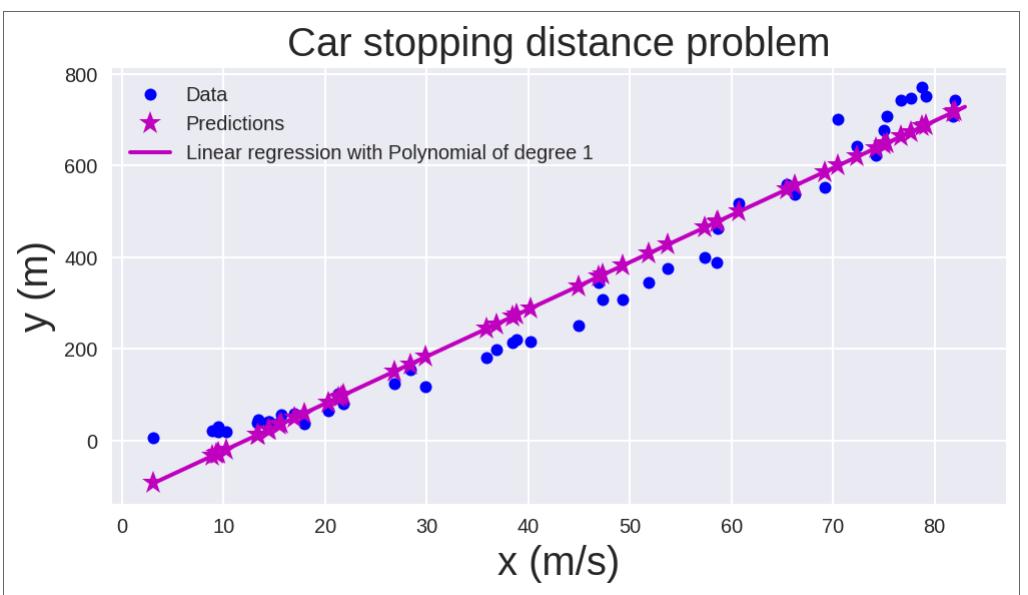
ML (supervised learning regression) goal

1. ML should find the mean response μ_y from the data $\mathcal{D} = \{\mathcal{D}_x, \mathcal{D}_y\}$ (data: blue dots).
1. Of course we will always make a prediction error (we just see the data, and we don't know the model or complete reality).
1. Ideally, ML should also tell us the confidence we have on the predictions we are making.
 - This is what probabilistic ML does!
1. **However**, even if we don't estimate our confidence (like most ML), having a probabilistic perspective over ML is advantageous because our unknowns can be modeled as [rv's](#).
 - This helps us understand where the models come from and that the world is probabilistic.

In the next classes we will understand the simplest ML supervised learning regression models. This will allow us to make simple predictions, so that later we make some serious predictions! Here's a teaser for the next lectures about linear regression:

```
In [12]:  
# Code to generate this figure is hidden during presentation (it's shown as notes).  
# (We will learn how to do this later...)  
fig_poly # show figure
```

Out[12]:



See you next class

Have fun!