# cme250_hw1_solutions

January 21, 2019

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.linear_model import LinearRegression
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

# 1 Part 2. Applied Exercise

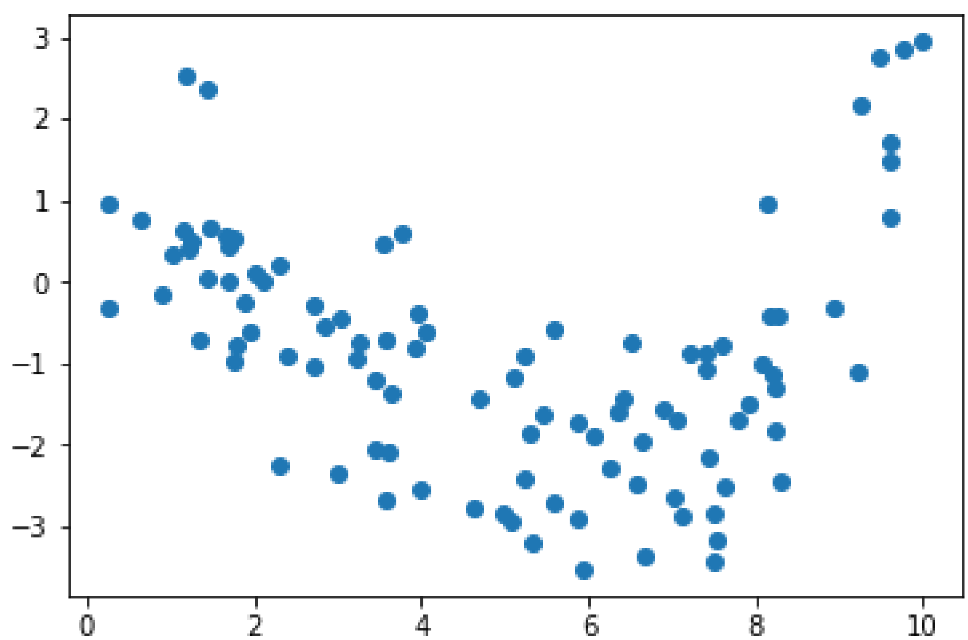
## 1.1 Question 1.

### 1.1.1 (a)

```
In [2]: # generate x values by sampling n observations from the interval [0,10]
        n = 100
        x = np.random.rand(n) * 10
```

```
In [3]: # generate y values by using deterministic relationship between x and y and adding noise
        y = 0.03 * x**3 - 0.3 * x**2 + 0.3 * x + np.random.randn(n)
```

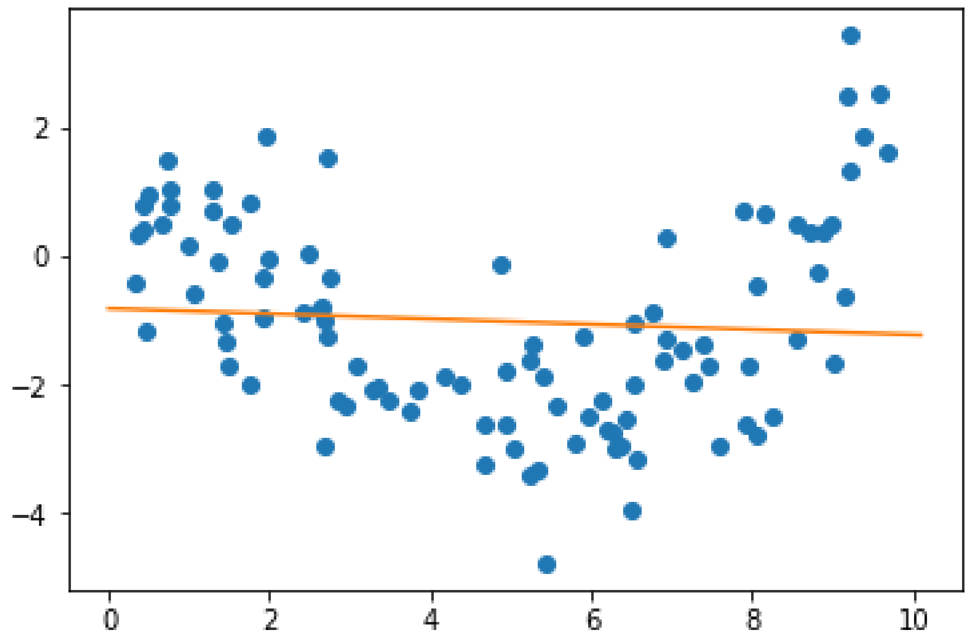
```
In [4]: plt.plot(x, y, 'o')
        plt.show()
```



### 1.1.2 (b)

In [13]:
```
def fit_LinearRegression(n, nruns, nfeats, verbose=True):
    """
    Fit a linear regression model using sklearn for nruns and nfeats.

    Parameters:
    n (int): sample size of random dataset to generate
    nruns (int): number of random datasets to generate and models to fit
    nfeats (int): order of model to fit; e.g. y = beta_0 * x is a 1st order model
    verbose (bool): whether to show plot and print R2 statistics

    Returns:
    R2s (array[float]): an (nruns,)-dimensional array of R2s of model fits
    """

    R2s = np.zeros(nruns)

    for run in range(nruns):
        x = np.random.rand(n) * 10
        y = 0.03 * x**3 - 0.3 * x**2 + 0.3 * x + np.random.randn(n)

        X = np.zeros((n, nfeats))
        for i in range(nfeats):
            X[:,i] = x**(i+1)

        lr = LinearRegression()
        lr.fit(X, y)

        R2s[run] = lr.score(X, y)

        if verbose and run == 0:
            x_line = np.arange(0,10.1,0.01)
            X_line = np.zeros((len(x_line), nfeats))
            for i in range(nfeats):
                X_line[:,i] = x_line**(i+1)
            y_hat = lr.predict(X_line)
            plt.plot(x, y, 'o')
            plt.plot(x_line, y_hat)
            plt.show()

    if verbose:
        print("R2: {:0.2f} +/- {:0.2f}".format(np.mean(R2s), np.std(R2s)))

    return R2s
```

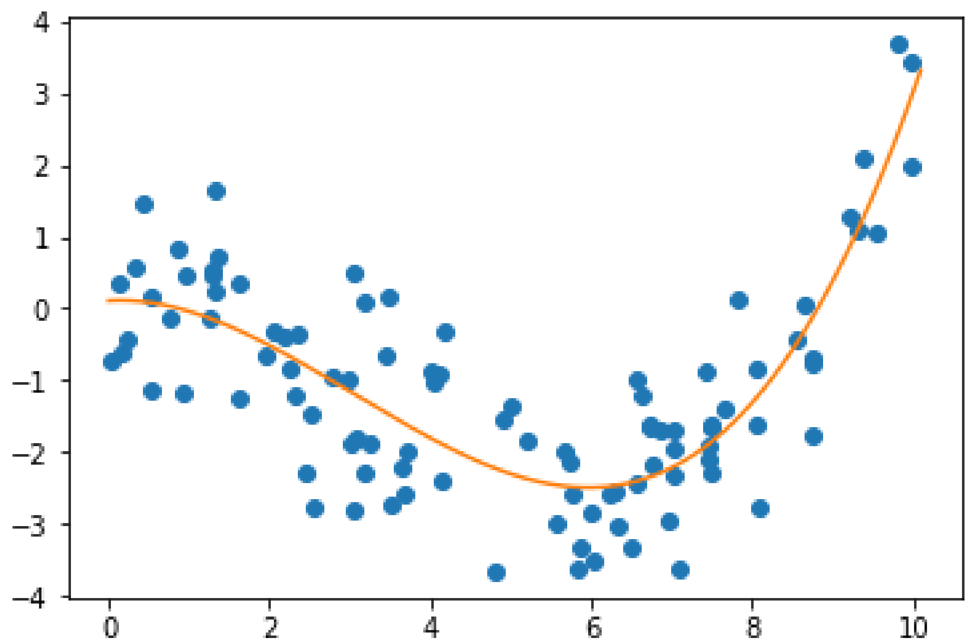
In [15]:
```
_ = fit_LinearRegression(n=100, nruns=10, nfeats=1, verbose=True)
```




```
R2: 0.01 +/- 0.01
```

#### 1.1.3 (c)

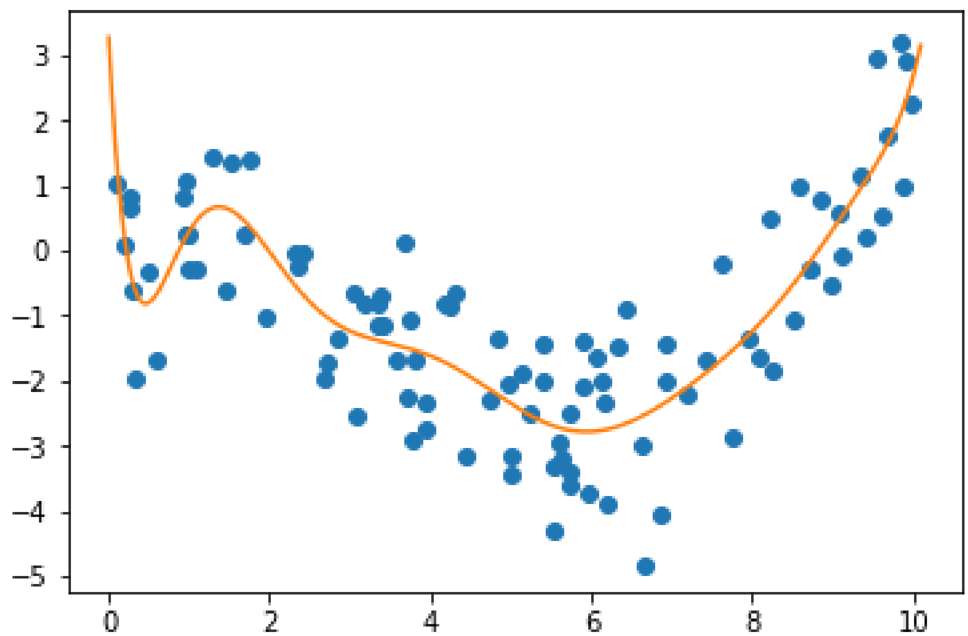
```
In [16]: _ = fit_LinearRegression(n=100, nruns=10, nfeats=3, verbose=True)
```



```
R2: 0.64 +/- 0.05
```

#### 1.1.4 (d)

```
In [17]: _ = fit_LinearRegression(n=100, nruns=10, nfeats=10, verbose=True)
```




```
R2: 0.65 +/- 0.06
```

#### 1.1.5 (e)

The 1st order model in part (b) has the highest bias. It makes a strong assumption about the functional relationship between $y$ and $x$ (linear). It is the least flexible model and is prone to underfitting.

The 10th order model in part (d) has the highest variance. It is the most flexible model, and as such is the most vulnerable to overfitting to noise (i.e. error/epsilon) rather than signal.

Since we know the generating process for $y$ is a cubic function of $x$ plus some irreducible noise, we know that the model from part (c), a linear regression on $x, x^2, x^3$, will generalize best to unseen data.

# 2 Part 3. Young People Survey

## 2.1 Question 2.

#### 2.1.1 (a)

```
In [2]: # use pandas to read in .csv file
        path = '../data/responses.csv'
        df = pd.read_csv(path)
```

```
In [21]: # let's take a look at our dataframe!
         df.head()
```

```
Out[21]:    Music  Slow songs or fast songs  Dance  Folk  Country  Classical music  \
         0    5.0                       3.0    2.0   1.0      2.0              2.0
         1    4.0                       4.0    2.0   1.0      1.0              1.0
         2    5.0                       5.0    2.0   2.0      3.0              4.0
         3    5.0                       3.0    2.0   1.0      1.0              1.0
         4    5.0                       3.0    4.0   3.0      2.0              4.0

            Musical  Pop  Rock  Metal or Hardrock          ...             Age  \
         0      1.0  5.0   5.0                1.0          ...            20.0
         1      2.0  3.0   5.0                4.0          ...            19.0
         2      5.0  3.0   5.0                3.0          ...            20.0
         3      1.0  2.0   2.0                1.0          ...            22.0
         4      3.0  5.0   3.0                1.0          ...            20.0

            Height  Weight  Number of siblings  Gender  Left - right handed  \
         0   163.0    48.0                 1.0  female        right handed
         1   163.0    58.0                 2.0  female        right handed
         2   176.0    67.0                 2.0  female        right handed
         3   172.0    59.0                 1.0  female        right handed
         4   170.0    59.0                 1.0  female        right handed

                          Education  Only child  Village - town  House - block of flats
         0  college/bachelor degree          no         village          block of flats
         1  college/bachelor degree          no            city          block of flats
         2         secondary school          no            city          block of flats
         3  college/bachelor degree         yes            city         house/bungalow
         4         secondary school          no         village         house/bungalow

         [5 rows x 150 columns]
```

```
In [22]: # dataframe dimensions
         df.shape
```

```
Out[22]: (1010, 150)
```

```
In [3]: # drop any rows with NaNs. in this case, default dropna parameters are fine
        df = df.dropna()
        df.shape
```

```
Out[3]: (674, 150)
```

Looks like we lost about 33% of our samples to missing data!

### 2.1.2 (b)

```
In [14]: y = df['Education']
         X = df.loc[:, df.columns != 'Education']
```

### 2.1.3 (c)

```
In [16]: # print columns that are categorical
         for column in X.columns:
             if X[column].dtype == type(object):
                 print("{}: {}".format(column, np.unique(X[column])))
```

```
Smoking: ['current smoker' 'former smoker' 'never smoked' 'tried smoking']
Alcohol: ['drink a lot' 'never' 'social drinker']
Punctuality: ['i am always on time' 'i am often early' 'i am often running late']
Lying: ['everytime it suits me' 'never' 'only to avoid hurting someone'
 'sometimes']
Internet usage: ['few hours a day' 'less than an hour a day' 'most of the day']
Gender: ['female' 'male']
Left - right handed: ['left handed' 'right handed']
Only child: ['no' 'yes']
Village - town: ['city' 'village']
House - block of flats: ['block of flats' 'house/bungalow']
```

Since many of the categorical variables are in fact ordered (e.g. Drinking: Never - Social drinker - Drink a lot), one could make a case for encoding these variables using integers (0, 1, 2) rather than one-hot encoding. However, if you think the difference between e.g. never drinking and social drinking vs. social drinking and drinking a lot is not the same, it makes more sense (at least if we are using a linear model) to use one-hot encoding. Either answer will be accepted here.

### One-hot encoding

In [17]:
```python
# select columns with categorical labels (type=object) and encode them as one-hot values
cat = X.select_dtypes(include=object).columns
X_onehot = pd.get_dummies(X, prefix=cat, columns=cat, drop_first=True)
```

In [18]:
```python
# notice gender, handedness, only child, village, and house are now one-hot
X_onehot.head()
```

Out[18]:
```
   Music  Slow songs or fast songs  Dance  Folk  Country  Classical music  \
0    5.0                       3.0    2.0   1.0      2.0              2.0
1    4.0                       4.0    2.0   1.0      1.0              1.0
2    5.0                       5.0    2.0   2.0      3.0              4.0
4    5.0                       3.0    4.0   3.0      2.0              4.0
5    5.0                       3.0    2.0   3.0      2.0              3.0

   Musical  Pop  Rock  Metal or Hardrock  \
0      1.0  5.0   5.0                1.0
1      2.0  3.0   5.0                4.0
2      5.0  3.0   5.0                3.0
4      3.0  5.0   3.0                1.0
5      3.0  2.0   5.0                5.0

                  ...                     Lying_never  \
0                 ...                               1
1                 ...                               0
2                 ...                               0
4                 ...                               0
5                 ...                               0

   Lying_only to avoid hurting someone  Lying_sometimes  \
0                                    0                0
1                                    0                1
2                                    0                1
4                                    0                0
5                                    1                0
```

```
   Internet usage_less than an hour a day  Internet usage_most of the day  \
0                                       0                               0
1                                       0                               0
2                                       0                               0
4                                       0                               0
5                                       0                               0

   Gender_male  Left - right handed_right handed  Only child_yes  \
0            0                                 1               0
1            0                                 1               0
2            0                                 1               0
4            0                                 1               0
5            1                                 1               0

   Village - town_village  House - block of flats_house/bungalow
0                       1                                      0
1                       0                                      0
2                       0                                      0
4                       1                                      1
5                       0                                      0

[5 rows x 156 columns]
```

In [19]:
```python
# we've created 16 one-hot columns from the 10 previous categorical ones
X_onehot.shape
```

Out[19]: (674, 156)

Note that sklearn can handle a response variable that is categorical without transforming it into a numeric encoding, so we won't do anything to the y vector here.

### Ordered numerical encoding

In [20]:
```python
# first cast categorical columns into category type
X_ordered = X.copy()
X_ordered[cat] = X[cat].astype('category')
```

In [24]:
```python
# currently the categories are not ordered
X_ordered['Lying'].cat.ordered
```

Out[24]: False

In [30]:
```python
# if we encode the categories using default settings, pandas will order them alphabetically
# since alphabetical order is not the logical order, we define the following order manually
ordered_cats = [['never smoked', 'tried smoking', 'former smoker', 'current smoker'],
                ['never', 'social drinker', 'drink a lot'],
                ['i am often running late', 'i am always on time', 'i am often early'],
                ['never', 'only to avoid hurting someone', 'sometimes', 'everytime it suits me
                ['less than an hour a day', 'few hours a day', 'most of the day']]

for i, col in enumerate(['Smoking', 'Alcohol', 'Punctuality', 'Lying', 'Internet usage']):
    X_ordered[col] = X_ordered[col].cat.reorder_categories(ordered_cats[i], ordered=True)
```

In [31]:
```python
# finally, create the correctly ordered codes
for col in cat:
    X_ordered[col] = X_ordered[col].cat.codes
```

**Now our data is ready to be input to a machine learning algorithm! Stay tuned for Homework 2.**