Assignment 1 Report

1. The four sections below account for four requirements respectively.
2. Items begin with solid dot as well as their sub-items are my answer, such as :

- Here goes the answers

  Also my answer

3. The detailed implementations, including optimization and variance of models, are in `hist.py kde.py knn.py` , while `source.py` displays several thorough figures for each requirement by importing the three and duplicating partial codes.
4. All figures in the report can be plotted by running :

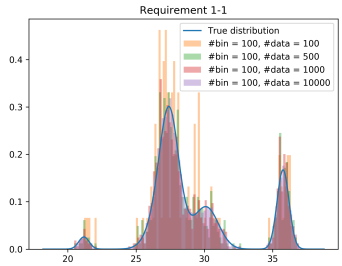
```
python3 hist.py
python3 kde.py
python3 knn.py
```

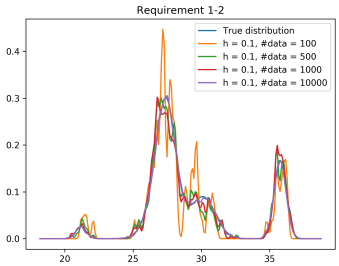
The number of data

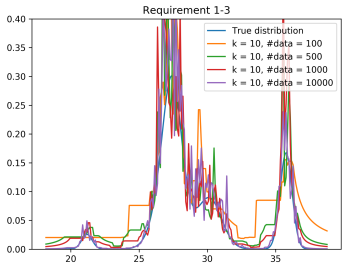
> According to the **law of large numbers**, the average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed.

- It is quite clear that the number of training data could influence our models, though could vary between different models.

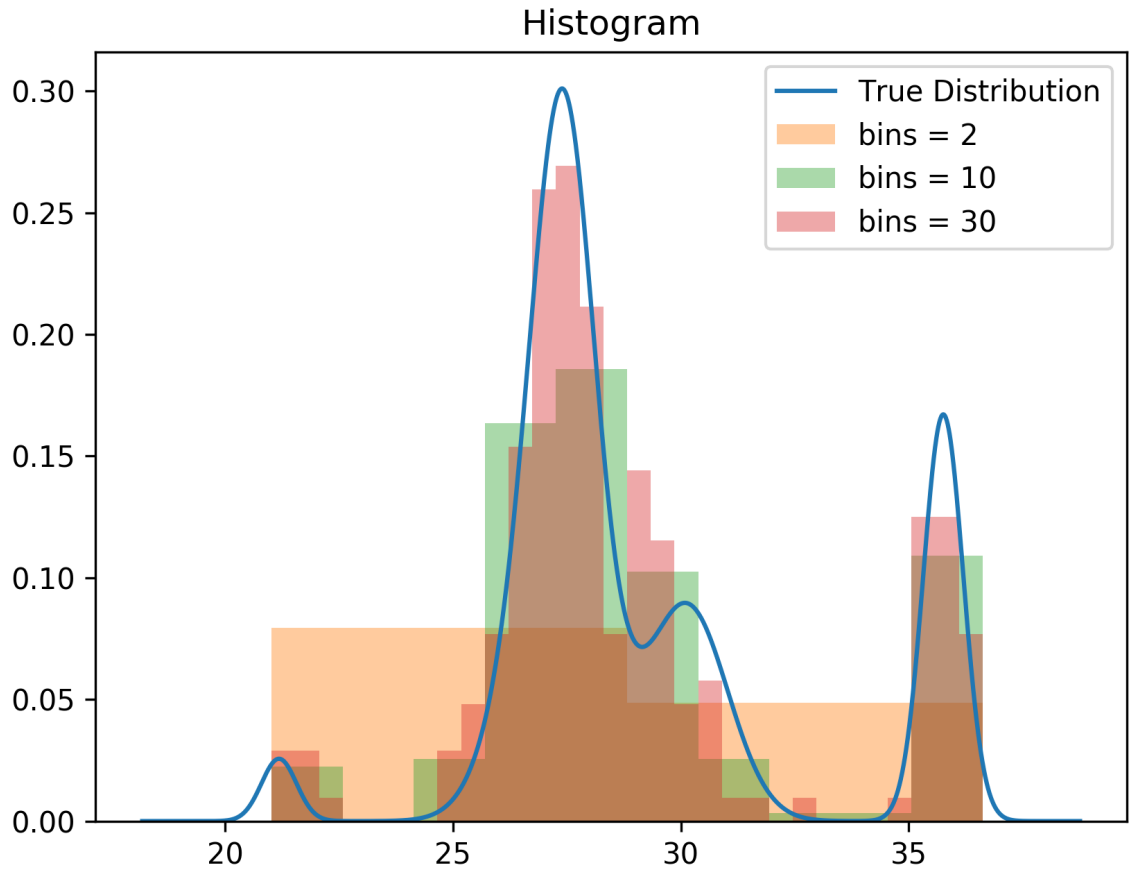
  As the number of data grows, our estimation improves since the distribution we get is closer to the true distribution. **Thus, when the dataset is sufficiently large, we can somehow get a well-trained model.**
- For Histogram, it always behaves pretty well if the dataset is large enough.
- For KDE, the model is also well-improved given large dataset.
- For KNN, however, it seems that the model is mostly dependent on `K` since even a large number of data can not smooth the curve well.







Histogram Method

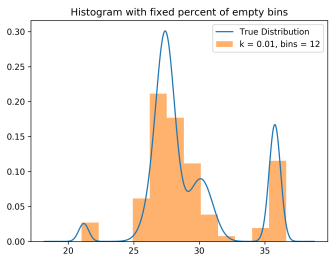


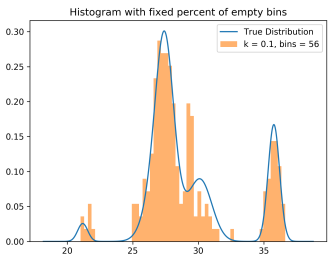
- How to choose the number of bins ?

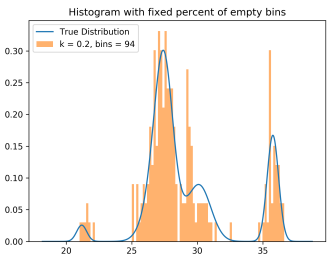
  From the figure above, we soon find that given more bins, it's more likely for some bins to be empty, which may denotes overfitting.

  Therefore, a more generalized model can be obtained by introducing `k`, governing not the number of bins, but **the percent of empty bins**, which more **straightforward** and easier to control.

  Also notice that there may be several partitions whose number of bins are the same. In this case, one simple optimization here is to choose the largest one to avoid underfitting.







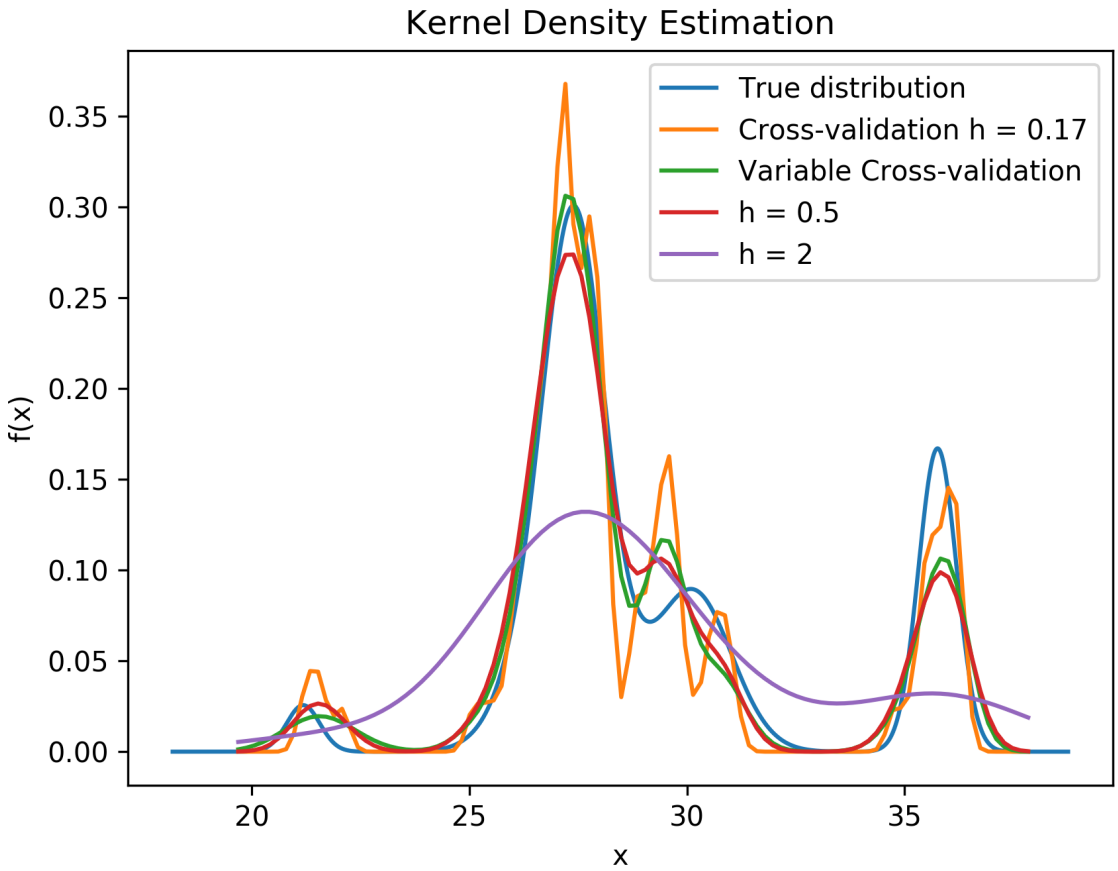
Kernel Density Estimation

- How to choose `h` ?

  Well, Not too small, not too large...

Intuitively, just small enough not to get a bad curve. However, simple ways such as Cross-validation can be used to guess for a better one, which is near **0.24** in my case.

- You can also run `python3 kde.py` to get the same figure, slightly different each time you plot it due to Cross-validation.



Here are two optimization methods.

1. Cross-validation (CV) **Automatically** choose an optimal bandwidth for KDE using the cross-validation method. Detail implementation of this method can be easily found in many textbooks on Machine Learning.

   Since dataset are randomly divided into `s` parts, it will no always output the same h. In practice, this basic but useful technique gives me h roughly in range **between 0.20 and 0.28**. It just fits well compared with the true distribution.

2. Variable KDE Based on **KNN** and **Cross-validation**, I come up with an algorithm that combines both and behaves well. The mechanism is specified as follows.

   1. Based on CV, we first get an almost optimal `h0`
   2. From KNN we have `P(xi) = K/(N*Vi)`, where `K` is a hyper-parameter, `N` is the number of the training data and `Vi` is the smallest volume to contain precisely `K` data points, estimating at point `xi`.
   3. Then we should determine how each `Vi` smooth h around our previous `h0` . For simplicity, I just use Sigmoid function centered at `(m, h0)` , where `m` is the mean of `Vi` .

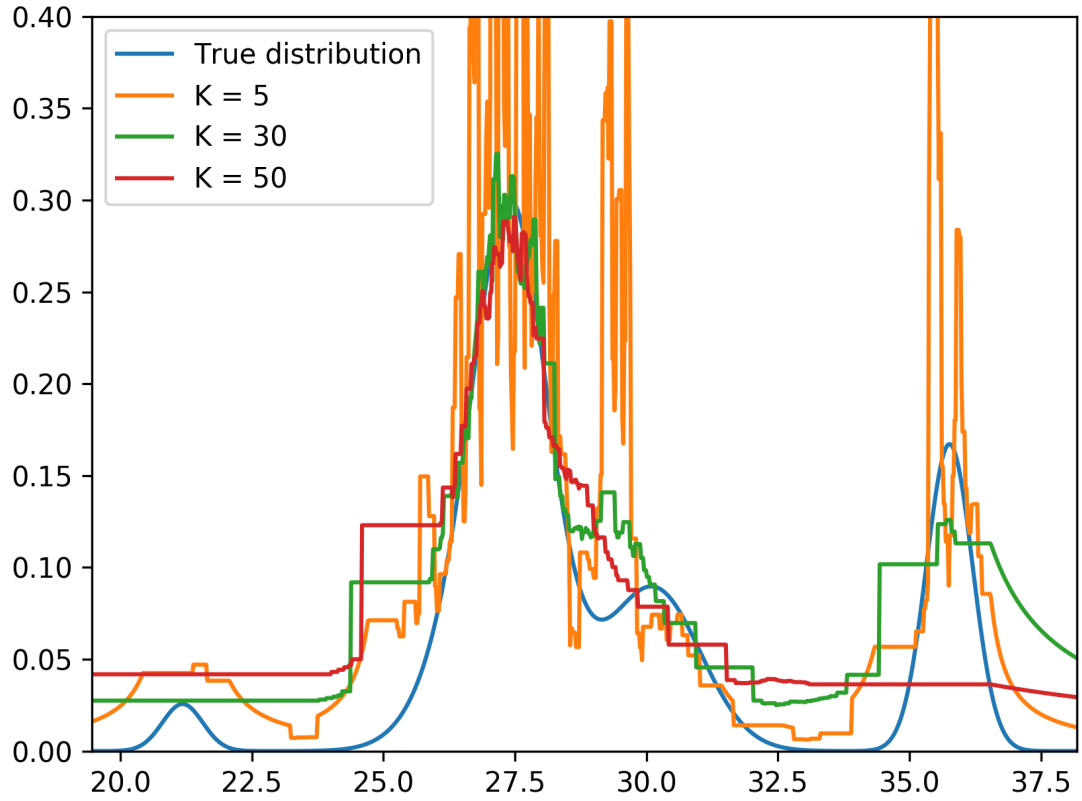
   It has such advantages as listed below :

   - **More Stability** It seems that after smooth, the curve becomes **less sensitive to noise and outliers**

- **More Smooth** Since now the bandwidth (h) can vary with the distribution. Bandwidths are smoothly tuned by a sigmoid function centered at `h0`, which is derived by KNN

K-Nearest Neighbors

- Here are fancy "curves" with K = 5, 30, 50.



- Conclusion can be drawn that it is not exactly sort of PDF, by either the formula `P(xi) = K/(N*Vi)` or by the figure above.
  - Specifically, let `L` be the set of left most k data points, such that

    `L = {x1, ..., xk}, x1 < x2 <...< xk`

    We can easily proof that integral over `(-∞, x1)` yields `+∞` .
    $\int_{-\infty}^{x_0} p(x)dx = \int_{-\infty}^{x_0} \frac{K}{NV(x)}dx = \int_{-\infty}^{x_0} \frac{K}{N\times 2(x_k-x)}dx = \frac{K}{2N}\int_{-\infty}^{x_0} \frac{1}{x_k-x}dx = \frac{K}{2N}\int_{x_k-x_0}^{+\infty} \frac{1}{x}dx = +\infty$
  - Cases in higher dimensions are the same.