CSC 2515: Introduction to Machine Learning

Homework 2

By: Alexis Bruneau

Collaborated with: Ramy ElMallah, Anton Korikov

Presented to Dr.Amir-massoud Farahmand

November 4th, 2022

Table of Content

Question 1 Bias and Variance Decomposition for the l_2-regularized Mean Estimator – 25 pts

Consider a r.v Y with an unknown distribution p. This random variable has an (unknown) mean $\mu = E[Y]$ and variance $\sigma^2 = Var[Y] = E[(Y-\mu)^2]$. Consider dataset $D = \{Y_1, \dots, Y_n\}$ with independently sampled $Y_i \sim p$.

a) [3pt] Show that the sample average estimator $\boldsymbol{h_{avg} = \frac{1}{n}\sum_{i=1}^{n} Y_i}$ is the solution of the following optimization problem:

$$\boldsymbol{h_{avg}(D) \leftarrow argmin_{m \epsilon R} \frac{1}{n}\sum_{i=1}^{n} |Y_i - m|^2}$$

Recall that we have the following bias-variance decomposition for the mean estimator h(D):

$$\boldsymbol{|E_D[h(D) - \mu]|^2 + E_D[|h(D) - E_D[h(D)]|^2]}$$

Since the value of $h_{avg}(D)$ is the minimum argument of the function mentioned above, the following can be said to find h_{avg}:

We assume the function is a convex function and we set the derivative to be equal 0

$$\frac{d}{dm} = \frac{1}{n}\sum_{i=1}^{n} |Y_i - m|^2 = 0$$

$$= -\frac{2}{n}\sum_{i=1}^{n} (Y_i - m) = 0$$

$$-\frac{2}{n}\sum_{i=1}^{n} Y_i + \frac{2}{n}(mn) = 0$$

$$2m = \frac{2}{n}\sum_{i=1}^{n} Y_i$$

We get that m is equal to the following:

$$m = \frac{1}{n}\sum_{i=1}^{n} Y_i = \boldsymbol{h_{avg}}$$

b) [2pts] Compute the bias and variance of $h_{avg}(D)$

Knowing the bias decomposition for the mean estimator $h(D)$, we can calculate the bias and variance with the following equations.

$$E\left[\, h_{avg}\right] = \frac{1}{n}E[Y] = \frac{1}{n} * m$$

Bias:

$$bias = \ |E_D[h(D)] - \mu|^2 = |E_D[h(D)] - \mu|^2$$

$$= |E[\frac{1}{n}\sum_{i=1}^{n} Y_i] - \mu\,|^2$$

$$= |\frac{1}{n}\sum_{i=1}^{n} E[Y_i] - \mu|^2$$

$$bias = \left|\frac{1}{n}\sum_{i=1}^{n} \mu - \mu\right|^2 = 0$$

Variance:

$$Variance = E_D[|h(D) - E_D[h(D)]|^2]$$

$$= E\left[\left|\frac{1}{n}\sum_{i=1}^{n} Y_i - E\left[\frac{1}{n}\sum_{i=1}^{n} Y_i\right]\right|^2\right]$$

$$= E\left[\left|\frac{1}{n}\sum_{i=1}^{n} (Y_i - \mu)^2\right|^2\right]$$

$$= \frac{1}{n^2}\sum_{i=1}^{n} E[(Y_i - \mu)^2] = \frac{1}{n^2} n\sigma^2 = \frac{\sigma^2}{n}$$

Hence, the Bias is = 0, and the Variance is equal to $\frac{\sigma^2}{n}$.

c) [5pt] Consider the l_2-regularized mean estimator $h_\lambda(D)$ defined for any $\lambda \geq 0$.

$$h_\lambda(D) \leftarrow argmin_{m \epsilon R} \frac{1}{n} \sum_{i=1}^{n} |Y_i - m|^2 + \lambda |m|^2$$

Provide an explicit formula for the estimator $h_\lambda(D)$ in terms of the sample average and λ.

Using the same logic as in a), the formula for the estimator $h_\lambda(D)$ can be found with the following equations.

Assuming the function is convex, the minimum value can be found by setting $\frac{d}{dm} = 0$

$$\frac{d}{dm} = \frac{d}{dm}\left(\frac{1}{n}\sum_{i=1}^{n} |Y_i - m|^2 + \lambda |m|^2\right) = 0$$

$$= -\frac{2}{n}\sum_{i=1}^{n} |Y_i - m| + 2\lambda m = 0$$

$$= -\frac{2}{n}\sum_{i=1}^{n} Y_i + \frac{2mn}{n} + 2\lambda m = 0$$

$$2m + 2\lambda m = \frac{2}{n}\sum_{i=1}^{n} Y_i$$

$$2m(1 + \lambda) = \frac{2}{n}\sum_{i=1}^{n} Y_i$$

Hence, we get the following formula for $h_\lambda(D)$:

$$m = \frac{1}{(1+\lambda)n}\sum_{i=1}^{n} Y_i = h_\lambda(D)$$

d) [6pt] Compute the bias and variance of this regularized estimator

The bias and variance of this regularized estimator can be found using the following formulas.

Bias:

$$Bias = |E_D[h_\lambda(D)] - \mu]|^2$$

$$E_D[h_\lambda(D)] = E\left[\frac{1}{(1+\lambda)} \cdot Y\right] = \frac{1}{(1+\lambda)} E[Y] = \frac{\mu}{(1+\lambda)}$$

$$Bias = \left|\frac{\mu}{(1+\lambda)} - \mu\right|^2$$

$$Bias = \left|-\frac{\mu\lambda}{(1+\lambda)}\right|^2 = \frac{\mu^2\lambda^2}{(1+\lambda)^2}$$

$$Variance = E_D[|h_\lambda(D) - E_D[h_\lambda(D)]|^2]$$

$$= E_D\left[\left|\left(\frac{1}{(1+\lambda)n}\sum_{i=1}^{n} Y_i\right) - E_d\left[\frac{1}{(1+\lambda)n}\sum_{i=1}^{n} Y_i\right]\right|^2\right]$$

$$= E_D\left[\left|\frac{1}{(1+\lambda)n}\sum_{i=1}^{n}(Y_i - \mu)\right|^2\right]$$

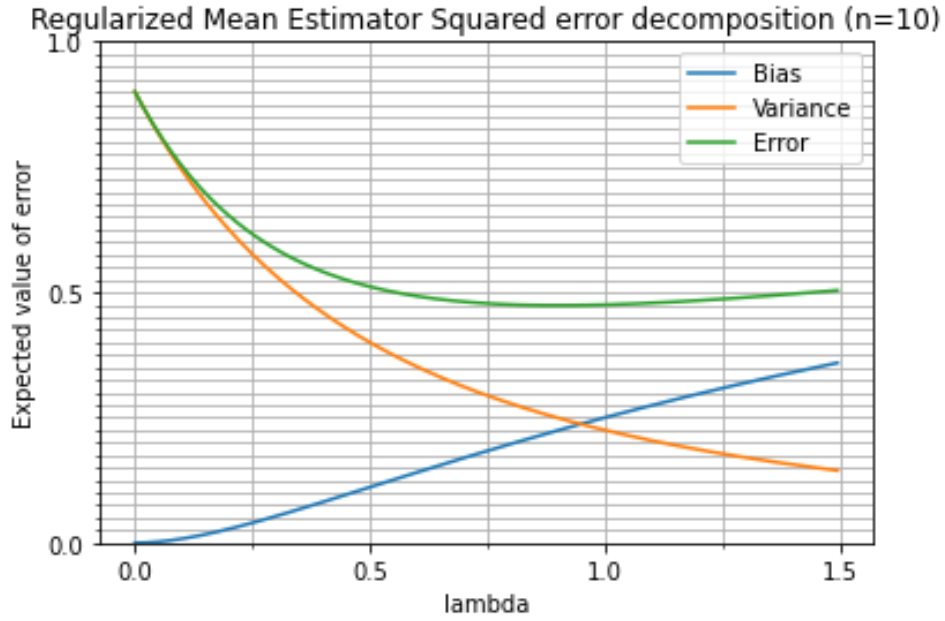
$$= \frac{1}{(1+\lambda)^2 n^2}\sum_{i=1}^{n} E_D[(Y_i - \mu)^2]$$

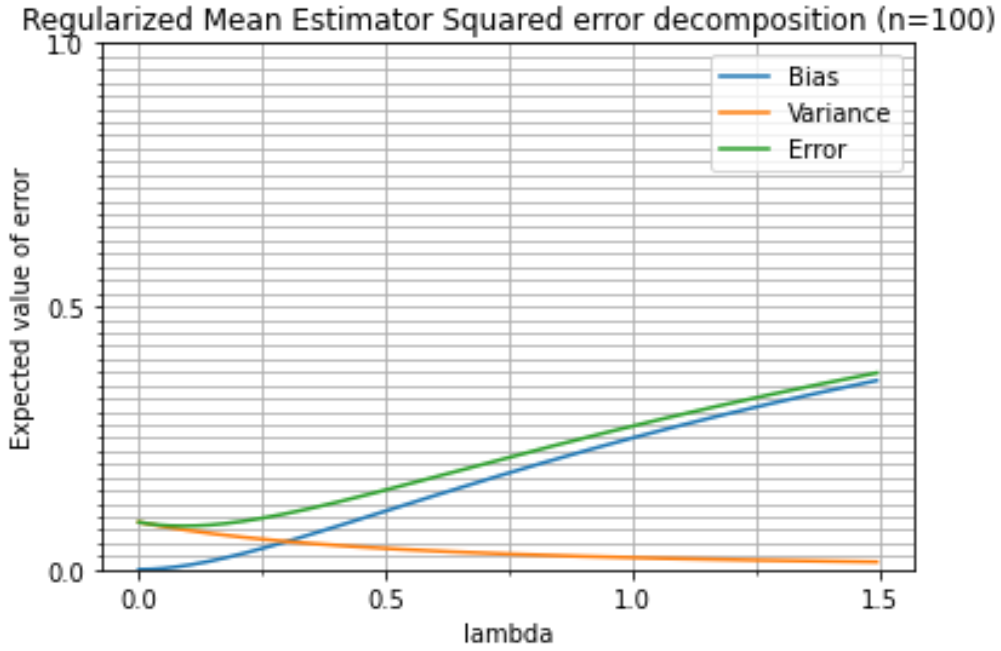
$$= \frac{1}{(1+\lambda)^2 n^2} * n * \sigma^2 = \frac{1}{(1+\lambda)^2 n}\sigma^2$$

Hence, for the regularized estimator, the bias is $\frac{\mu^2\lambda^2}{(1+\lambda)^2}$ and the variance is $\frac{1}{(1+\lambda)^2 n}\sigma^2$.

e) [5pt] Visualize $E_D[|h_\lambda(D) - \mu|^2]$, the bias, and the variance terms for a range of λ. As a starting point, you can choose $\mu = 1, \sigma^2 = 9,$ and $n = 10$.

The following is a visualization for the bias, decomposition, and total error for a range of λ when $\mu = 1, \sigma^2 = 9, n = 10$ and $\mu = 1, \sigma^2 = 9, n = 100$





f) [4pt] Explain the visualization from the previous part, in terms of the effect of bias and variance on the expected squared error.

From the figures presented in f), we can observe the following. As the number of sample n increases, the bias stays the same. This is expected as the bias equation has no relation to n.

$$\text{bias} = \frac{\mu^2\lambda^2}{(1+\lambda)^2}$$

We can also see that using the same values for u, and σ but increasing the value of n decreased the variance and the total error. This is also expected as n is in the numerator of the variance equation:

$$variance = \frac{1}{(1+\lambda)^2 n}\sigma^2$$

And the error is the sum of the bias and variance. It can also be seen that when lambda is 0, the bias is equal to 0 and increases as the value of lambda increases. On the opposite end, when lambda increases, the variance decreases. With a larger number of samples, where the expected variance is lower, a smaller value of lambda will achieve the smallest expected squared error. Using the visualization, it can also be seen when the model if the model is overfitted or underfitted. If we're on the left side of the minimum value for error, we're underfitting the model, and if we're on the right, we're over fitting it.

Question 2 Different Flavours of Regression – 20pts.

a) *[5pt] Suppose that we are given a dataset of $\{Z_1, \dots, Z_n\}$ all independently drawn from a Poisson distribution with unknown parameter λ. Derive the maximum likelihood estimate of the λ. Show the individual steps and note where you use the data assumptions (identically and independently distributed). Hint: Use the standard approach to derive the MLE via $argmax_\lambda log \prod p(Z_i; \lambda)$*

The probability mass function of a random variable Z with Poisson distribution with the rate $\lambda \epsilon \{0, \infty\}$ is:

$$P\{Z = k; \lambda\} = p(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Using the information given, the likelihood function $L(\lambda)$ can be written as:

$$L(\lambda) = \prod_{i=1}^{n} p(Z_i; \lambda) = \prod_{i=1}^{n} \left\{ \frac{\lambda^k e^{-\lambda}}{k!} \right\}$$

$$L(\lambda) = \frac{\lambda^{k_1 + \cdots + k_n}}{k_1! \dots k_n!} e^{n\lambda}, k \epsilon \{0,1, \dots, \infty\}, \lambda > 0$$

For the likelihood function, **considering iid** sample for $k_1, k_2, \dots, k_n$ from Poisson variable,

$$L(\lambda | k_1, k_2, \dots, k_n) = P(Z = k_1 | \lambda) P(Z = k_2 | \lambda) \dots P(Z = k_n | \lambda)$$

$$L(\lambda | k_1, k_2, \dots, k_n) = e^{-\lambda} \frac{\lambda^{k_1}}{k_1!} \dots e^{-\lambda} \frac{\lambda^{k_n}}{k_n!} = e^{-n\lambda} \frac{\lambda^{k_1 + k_2 + \cdots + k_n}}{k_1! \, k_2! \dots k_n!}$$

Taking the log-likelihood:

$$L(\lambda | k_1, k_2, \dots, k_n) = \ln \left(\prod_{i=1}^{n} \frac{\lambda^{k_i} e^{-\lambda}}{k_i!} \right)$$

$$L(\lambda | k_1, k_2, \dots, k_n) = \sum_{i}^{n} \ln \left(\frac{\lambda^{k_i} e^{-\lambda}}{k_i!} \right)$$

$$L(\lambda | k_1, k_2, \dots, k_n) = \sum_{i=1}^{n} [k_j \ln(\lambda) - \lambda - \ln(k_j!)]$$

$$L(\lambda | k_1, k_2, \dots, k_n) = -n\lambda + \ln(\lambda) \sum_{i=1}^{n} k_j - \sum_{i=1}^{n} \ln(k_j!)$$

Calculating the derivative of the natural log likelihood function with respect to λ gives the following:

$$\frac{d \, L(\lambda | k_1, k_2, \dots, k_n)}{d\lambda} = \frac{d}{d\lambda} (-n\lambda + \ln(\lambda) \sum_{i=1}^{n} k_i - \sum_{i=1}^{n} \ln(k_i!))$$

$$= -n + \frac{1}{\lambda} \sum_{i=1}^{n} k_i - 0$$

$$= -n + \frac{\sum_{i=1}^{n} k_i}{\lambda}$$

Setting the derivative equal to zero to solve λ gives the following:

$$\frac{d\, L(\lambda|k_1, k_2, \dots, k_n)}{d\lambda} = -n + \frac{\sum_{i=1}^{n} k_i}{\lambda} = 0$$

$$\boldsymbol{\lambda = \frac{\sum_{i=1}^{n} k_i}{n}}$$

b) The Poisson regression model is based on considering the λ parameter of the Poisson distribution a function of x and a weight w, which should be determined. The relation is specified by

$$log\lambda(x; w) = w^T x$$

Here the logarithm of the rate has a linear model, as opposed to the rate itself. This ensures that the rate is always non-negative. This rate function leads to the following statistical model for target $y \epsilon \{0,1, \dots\}$ *conditioned on the input x:*

$$p(y|x; w) = \frac{e^{y w^T x} \cdot e^{-e^{w^T x}}}{y!}$$

Derive the maximum likelihood function of this model given a dataset consisting of i.i.d input and response variables {($(x^1, y^1), \dots, (x^N, y^N)$*)}. Note that the resulting estimator does not have a closed form solution, so you only have to simplify the resulting loss function as far as possible*

Given the Poisson distribution's probability mass function $p(y|x; w)$ and the response variables *{(*$(x^1, y^1), \dots, (x^N, y^N)$*)},* the probability of attaining this particular set of data can be written as:

$$p(y_1, \dots, y_n|x_1, \dots, x_n; w) = \prod_{i=1}^{n} \frac{e^{y_i w^T x_i} e^{-e^{w^T x_i}}}{y_i!}$$

For the method of maximum likelihood, we want to find the set of parameters w that makes this probability as big as possible. Setting the equation above as a likelihood function in terms of w does that:

$$L(w|X, Y) = \prod_{i=1}^{n} \frac{e^{y_i w^T x_i} e^{-e^{w^T x_i}}}{y_i!}$$

Using the log-likelihood gives the following:

$$l(w|X, Y) = logL(w|X, Y)$$

$$l(w|X, Y) = \ln\left(\prod_{i=1}^{n} \frac{e^{y_i w^T x_i} e^{-e^{w^T x_i}}}{y_i!}\right)$$

$$= \sum_{i=1}^{n} ln(\frac{e^{y_i w^T x_i} e^{-e^{w^T x_i}}}{y_i!})$$

$$l(w|X,Y) = \sum_{i=1}^{n} (y_i w^T x_i) - e^{w^T x_i} - \ln(y_i!)$$

Deriving $l(w|X,Y)$ w.r.t w gives the following (the term $\ln(y_i)$ is dropped immediately as there is no terms w):

$$\frac{dl(w|X,Y)}{dw} = \frac{d\left(\sum_{i=1}^{n} (y_i w^T x_i) - e^{w^T x_i}\right)}{dw}$$

$$= \sum_{i=1}^{n} y_i x_i - x_i e^{w^T x_i}$$

To find the maximum, we solve the equations $\frac{dl(w|X,Y)}{dw} = 0$ which can be written as the following:

$$\sum_{i=1}^{n} y_i x_i - x_i e^{w^T x_i} = 0$$

This does not have a closed-form solution. It has the same form as a logistic regression, and it would be impossible to directly solve for w. However, the negative log-likelihood is a convex function. Using gradient descent would find the optimal value of w.

c) *[5pt] The weighted least squares cost uses positive weights $a_1, \dots, a_N$ per datapoint to construct a parameter estimate of the form:*

$$w^* \leftarrow \underset{w}{\operatorname{argmin}} \frac{1}{2}\sum_{i=1}^{N} a_i^{(i)}\left(y^{(i)} - w^T x^{(i)}\right)^2$$

Show that the solution to this optimization problem is given by the formula

$$w^* = (X^T A X)^{-1} X^T A y$$

Where X is the design matrix and A is a diagonal matrix where $A_{ii} = a^{(i)}$

We know that the direct solution for w that minimizes the regularized weighted least squares loss is the following:

$$L = \frac{1}{2}\sum_{i=1}^{N} a_i^{(i)}\left(y^{(i)} - w^T x^{(i)}\right)^2$$

Deriving L w.r.t w is given by:

$$\frac{dL}{dw} = \frac{1}{2}\sum_{i=1}^{N} a_i^{(i)} * \left(2x^{(i)}\right) * \left(y^{(i)} - w^T x^{(i)}\right)$$

$$\frac{dL}{dw} = \sum_{i=1}^{N} a_i^{(i)}\left(x^{(i)}\right)\left(y^{(i)}\right) - \sum_{i=1}^{N} a_i^{(i)}\left(x^{(i)}\right)\left(w^T x^{(i)}\right)$$

For the first term of the equation, it can be written as such:

$$\sum_{i=1}^{N} a_i^{(i)}\left(x^{(i)}\right)\left(y^{(i)}\right) = X^T A y$$

Where:

$$X = N\ x\ D$$

$$A = N\ x\ N$$

$$y = N\ X\ 1$$

$$X^T = \begin{bmatrix} x_{11} & x_{21} & .. & .. & x_{N1} \\ x_{12} & x_{22} & .. & .. & x_{N2} \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ x_{1D} & .. & .. & .. & x_{ND} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & .. & .. & .. & 0 \\ .. & a_{22} & .. & .. & .. \\ .. & .. & a_{33} & .. & .. \\ .. & .. & .. & .. & .. \\ 0 & .. & .. & .. & a_{NN} \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ .. \\ .. \\ y_N \end{bmatrix}$$

The result of $X^T Ay$ gives a matrix of size D X 1

$$X^T Ay = \sum_{i=1}^{N} a_i^{(i)} \left(x^{(i)}\right)\left(y^{(i)}\right) = a^1 x^1 y^1 + a^2 x^2 y^2 + \ldots + a^N x^N y^N$$

Similarly, for the second term can be written in Matrix form as such:

$$\sum_{i=1}^{N} a_i^{(i)} \left(x^{(i)}\right)\left(w^T x^{(i)}\right) = X^T AXw$$

Where:

$$X = N \; x \; D$$

$$A = N \; x \; N$$

$$w = D \; X \; 1$$

$$X^T = \begin{bmatrix} x_{11} & x_{21} & .. & .. & x_{N1} \\ x_{12} & x_{22} & .. & .. & x_{N2} \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ x_{1D} & .. & .. & .. & x_{ND} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & .. & .. & .. & 0 \\ .. & a_{22} & .. & .. & .. \\ .. & .. & a_{33} & .. & .. \\ .. & .. & .. & .. & .. \\ 0 & .. & .. & .. & a_{NN} \end{bmatrix}$$

$$X = \begin{bmatrix} x_{11} & x_{12} & .. & .. & x_{1D} \\ x_{21} & x_{22} & .. & .. & x_{2D} \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ X_{N1} & .. & .. & .. & x_{ND} \end{bmatrix}$$

$$w = \begin{bmatrix} w_1 \\ w_2 \\ .. \\ .. \\ w_D \end{bmatrix}$$

The result of $X^T AXw$ is of dimension D x 1:

$$X^T AXw = \sum_{i=1}^{N} a_i^{(i)} \left(x^{(i)}\right)\left(w^T x^{(i)}\right) = x_{11} a_{11} w \cdot x^1 + x_{12} a_{22} w \cdot x^2 + \cdots + x_{1N} a_{NN} w \cdot x^N$$

It is to note that the terms $w \cdot x^1$, $w \cdot x^2$… are scalars as w is of size D x 1 and x^i is one row of the matrix X of size D X 1.

Hence, we get that the function L can be written as the following:

$$L = X^T Ay - X^T AXw$$

Setting $\frac{dL}{dw} = 0$, we get the following:

$$\frac{dL}{dw} = 0 = X^T Ay - X^T AXw$$

$$X^T AXw = X^T Ay$$

$$w = (X^T AXw)^{-1} X^T Ay$$

d) *[5pt] Locally weighted least squares combines ideas from k-NN and linear regression. For each query x, we first compute distance-based weights for each training example*

$$a^i(x) = \frac{e^{-\frac{\|x-x^i\|^2}{2\tau^2}}}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}}$$

For some temperature parameter $\tau > 0$ We then construct a local solution

$$w^*(x) \leftarrow \underset{w}{argmin} \frac{1}{2}\sum_{i=1}^{N} a^i(x)\left(y^i - w^T x^i\right)^2$$

The prediction then takes the form $\hat{y} = w^*(x)^T x$. How does this algorithm behave as the temperature $\tau \to 0$? What happens as $\tau \to \infty$? What is the disadvantage of this approach compared to the least squares regression in terms of computational complexity?

As $\tau \to \infty$ **,** the value of $a^i = \frac{1}{N}$ and can be calculated with the following expression:

$$\lim_{\tau\to\infty} a^i(x) = \lim_{\tau\to\infty} \left(\frac{e^{-\frac{\|x-x^i\|^2}{2\tau^2}}}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}} \right)$$

$$= \lim_{\tau\to\infty} \left(\frac{\left(\frac{1}{e^{-\frac{\|x-x^i\|^2}{2\tau^2}}}\right)}{\left(\frac{1}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}}\right)} \right) = \frac{\left(\frac{1}{e^{-\frac{\|x-x^i\|^2}{2\infty^2}}}\right)}{\left(\frac{1}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{2\infty^2}}}\right)} = \frac{\frac{1}{e^0}}{\frac{1}{\sum_{j=1}^{N} e^0}}$$

$$\lim_{\tau\to\infty} a^i(x) = \frac{1}{\sum_{j=1}^{N} 1} = \frac{1}{N}$$

From this, as $\tau \to \infty$**,** all residuals get an equal weighting and the algorithm acts like a linear regression.

It is a bit more complicated to calculate the algorithm when $\tau \to 0$**.**

$$\lim_{\tau\to 0} a^i(x) = \lim_{\tau\to 0} \left(\frac{e^{-\frac{\|x-x^i\|^2}{2\tau^2}}}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}} \right)$$

$$= \lim_{\boldsymbol{\tau \to 0}} \left(\frac{\left(\frac{1}{e^{-\frac{\|x-x^i\|^2}{2\tau^2}}} \right)}{\left(\frac{1}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}} \right)} \right) = \frac{\left(\frac{1}{e^{-\frac{\|x-x^i\|^2}{0}}} \right)}{\left(\frac{1}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{0}}} \right)} = \frac{\frac{1}{e^{-\infty}}}{\frac{1}{\sum_{j=1}^{N} e^{-\infty}}}$$

It is hard to conclude something with this approach. However, it is possible to calculate the value of a^i for the 2 following cases, case 1 $\|x - x^i\|$ = 0 or case 2 $\|x - x^i\| > 0$

For the case when $\|x - x^i\|$ = 0, we can say the following:

$$\left(\frac{1}{e^{-\frac{\|x-x^i\|^2}{2\tau^2}}} \right) = \left(\frac{1}{e^{-\frac{0}{2\tau^2}}} \right) = 1$$

Hence, we can say the following for case 1:

$$\left(\frac{e^{-\frac{\|x-x^i\|^2}{2\tau^2}}}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}} \right) = \frac{1}{1 + \sum_{j \neq i}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}}$$

We get the following for case 1 when $\tau \to 0$:

$$\lim_{\tau \to 0} \left(\frac{1}{1 + \sum_{j \neq i}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}} \right) = \frac{1}{1 + \sum_{j \neq i}^{N} e^{-\frac{\|x-x^j\|^2}{2(0)^2}}} = \frac{1}{1 + \sum_{j \neq i}^{N} e^{-\infty}} = 1$$

$$a^i = 1$$

For case 2 $\|x - x^i\| > 0$, we can say the following:

$$\frac{\left(\frac{1}{e^{-\frac{\|positive\ number\|^2}{2\tau^2}}} \right)}{\left(\frac{1}{\sum_{j=1}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}} \right)} = \frac{\left(\frac{1}{e^{-\frac{\|positive\ number\|^2}{2\tau^2}}} \right)}{\left(\frac{1}{e^{-\frac{\|positive\ number\|^2}{2\tau^2}}} \right) + \sum_{j \neq 1}^{N} e^{-\frac{\|x-x^j\|^2}{2\tau^2}}}$$

We get the following for case 2 when $\tau \to 0$:

$$a^{i\neq j} = \frac{\left(\frac{1}{e^{-\frac{\|positive\ number\|^2}{2(0)^2}}}\right)}{\left(\frac{1}{e^{-\frac{\|positive\ number\|^2}{2(0)^2}}}\right) + \sum_{j\neq 1}^{N} e^{-\frac{\|x-x^j\|^2}{2(0)^2}}} = \frac{one\ large\ number}{one\ large\ number + \sum_{j\neq 1}^{N} large\ number}$$

$$a^{i\neq j} \approx 0$$

From the equation above, we can say that as the results of $\|x - x^i\|$ gets bigger, the closer the value of $a^{i\neq j}$ will be to 0. We showed that when $\tau \to 0$ the possible values for $a^i(x)$ are the following:

$$a^i = 1\ when\ \|x - x^i\| = 0$$

$$a^{i\neq j} \approx 0\ when\ \|x - x^i\| \gg 0$$

When $\tau \to 0$, the weight $a^i(x)$ only relies on the nearest neighbour. The locally weighted squared regression algorithm is more computationally demanding compared to the least squared regression. It requires optimization to fit the weights at run time for each query test point. Also, it needs to calculate the distance to all points in the training data at run-time for each query point. The least squares regression only fits the weights on the training data set once. It stores them and multiplies each query test point at run time. It is to note that there's not training for LWR like KNN. However, storing the full training set is then used at inference time to do the fitting.

Question 3 Implementing Regression Methods in Python

3.1 Initial Data Analysis [15pts]

a)[0pt] Load the Capital Bike sharing Dataset from the downloaded hour.csv file as a pandas dataframe.

This was accomplished using the following code:

```
[ ] uploaded = files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Bike-Sharing-Dataset.zip to Bike-Sharing-Dataset (1).zip
```

```
[ ] !unzip Bike-Sharing-Dataset.zip
```

```
Archive:  Bike-Sharing-Dataset.zip
replace Readme.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: Readme.txt
replace day.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: day.csv
replace hour.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: hour.csv
```

```
[ ] data = pd.read_csv('hour.csv')
    data
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0000	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0000	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0000	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0000	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0000	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	17375	2012-12-31	1	1	12	19	0	1	1	2	0.26	0.2576	0.60	0.1642	11	108	119
17375	17376	2012-12-31	1	1	12	20	0	1	1	2	0.26	0.2576	0.60	0.1642	8	81	89
17376	17377	2012-12-31	1	1	12	21	0	1	1	1	0.26	0.2576	0.60	0.1642	7	83	90
17377	17378	2012-12-31	1	1	12	22	0	1	1	1	0.26	0.2727	0.56	0.1343	13	48	61
17378	17379	2012-12-31	1	1	12	23	0	1	1	1	0.26	0.2727	0.65	0.1343	12	37	49

17379 rows × 17 columns

b)[2pt] Describe and summarize the data in terms of number of data points, dimensions, and used data types.

The summary of the data set is summarized as follows:

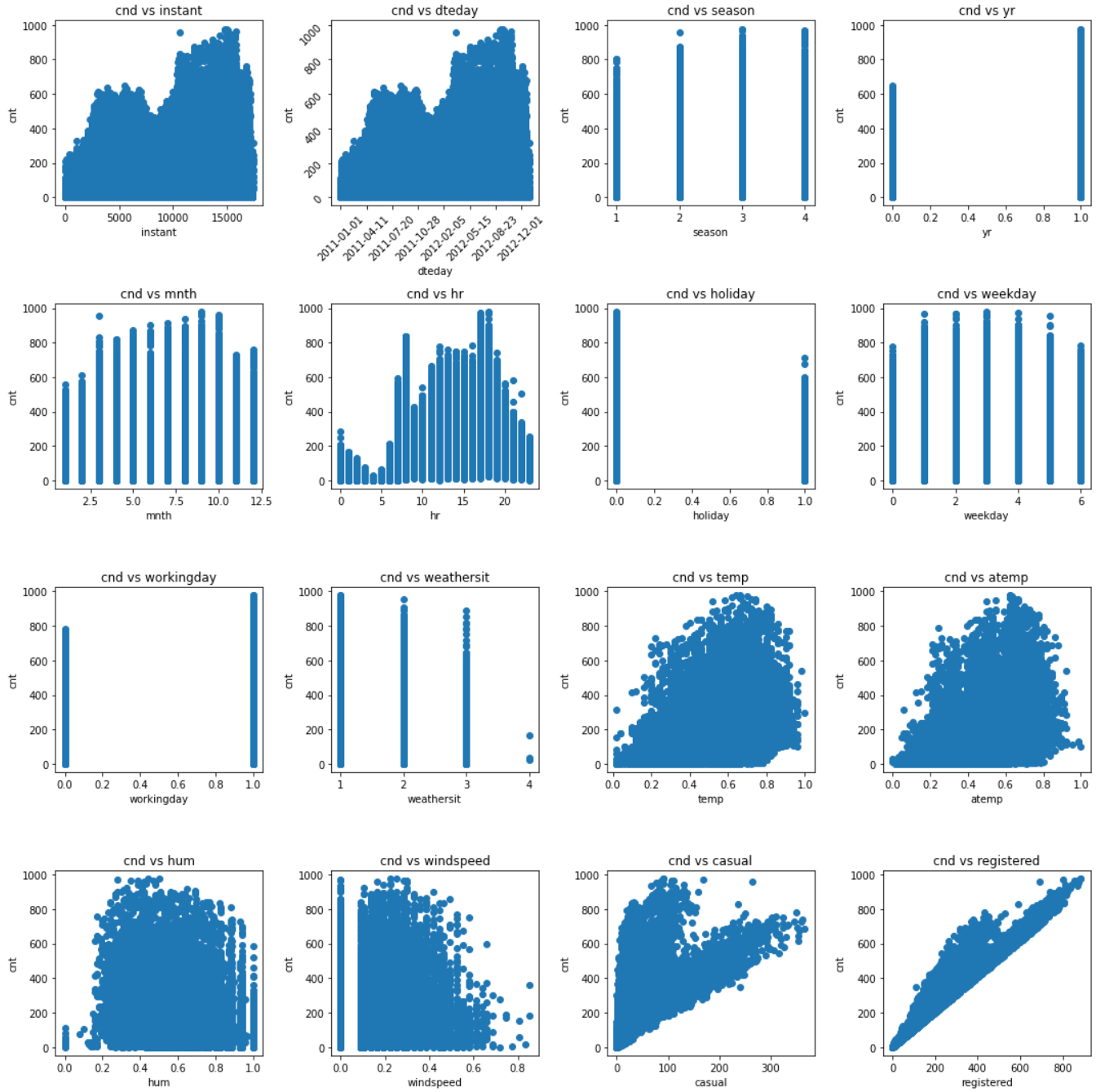
The data has 17379 samples with 16 features (some are dependent). Each sample has the following features:

- dteday : date - string
- season : season (1:winter, 2:spring, 3:summer, 4:fall) - integer
- yr : year (0: 2011, 1:2012) - integer
- mnth : month (1 to 12) - integer
- hr : hour (0 to 23) - integer

- holiday : weather day is holiday or not (0 or 1) - integer
- weekday : day of the week - integer
- workingday : if day is neither weekend nor holiday is 1, otherwise is 0 - integer
- weathersit : - integer
- 1: Clear, Few clouds, Partly cloudy, Partly cloudy
- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
- 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp : Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale) - float
- atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-16, t_max=+50 (only in hourly scale) - float
- hum: Normalized humidity. The values are divided to 100 (max) - float
- windspeed: Normalized wind speed. The values are divided to 67 (max) - float
- casual: count of casual users - integer
- registered: count of registered users - integer
- cnt: count of total rental bikes including both casual and registered – integer

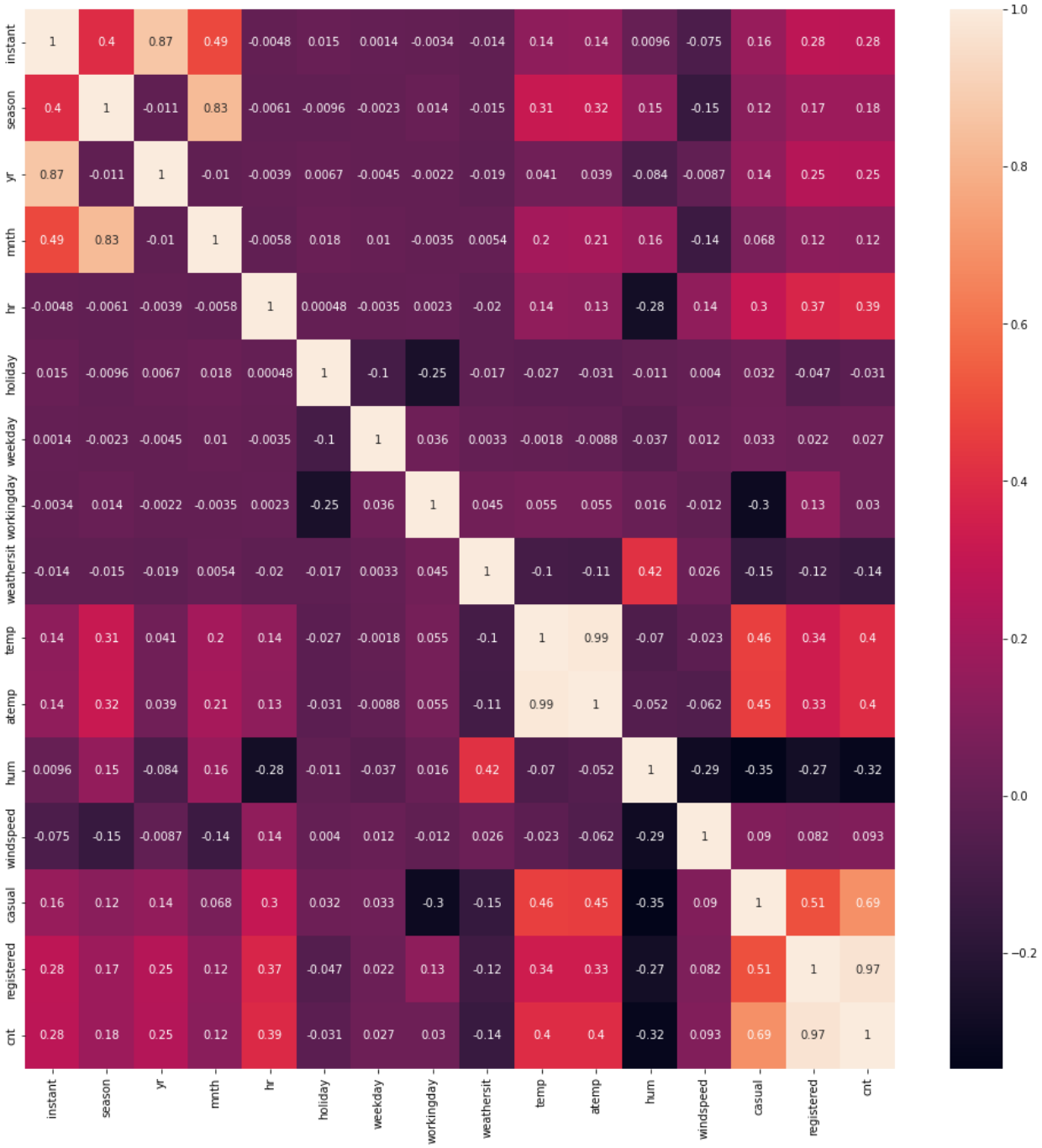
c)Present a single grid containing plots for each feature against the target. Choose the appropriate axis for dependent vs independent variables.

Underneath is the grid containing a plot for each feature. Each plot is scatter plots of the target vs the feature. It is to note that most of dteday labels were removed for clarity.



d)[5pt] Perform a correlation analysis on the data and plot the correlation matrix as a colored image. State which feature is the most positively, most negatively, and least correlated with the target column cnt.

Underneath is the correlation matrix that was obtained. The feature that is the most positively correlated to cnt is the feature ‘registered’ with a correlation of 0.97 to cnt. The most negatively correlated feature was hum with a correlation of -0.32 to cnt. Finally, the least correlated feature was the weekday with a correlation of 0.027 to cnt.



e)[1pt] Drop the following columns from the dataframe: instant, atemp, registered,casual,dteday.

The columns were dropped from the dataframe using the following line:

```
[ ]  data = data.drop(columns=["instant", "atemp", "registered", "casual", "dteday"])
```

f)[2pt] Shuffle the dataframe's rows using sklearn.utils.shuffle with random state 0. Split the data into a training set and a test set on index 10000.

The following image illustrates the code used to shuffle the dataframe with random state 0 and split the data into training and test set based on index 10000

```
[ ]  data = sklearn.utils.shuffle(data, random_state=0)
```

```
[ ]  train_data = data[0:10000]
     test_data = data[10000:]
```

3.2 Regression implementation – 40 pts

a)[5pts] Implement the ordinary least squares regression algorithm as discussed in class. You are free to implement the closed form solution.

The ordinary least squares regression algorithm (close form) was implemented in the function ordinary_least_squrares_regression. The code can be seen underneath:

```
[80] def ordinary_least_squares_regression(X, y, use_solver=False):
       X = np.insert(X.to_numpy(), 0, values=1, axis=1) # add one column for bias
       y = y.to_numpy()
       a = np.dot(X.T, X)
       b = np.dot(X.T, y)
       if use_solver:
         wb = scipy.linalg.solve(a, b)
       else:
         wb = np.dot(np.linalg.inv(a), b)
       return wb[0], wb[1:]
```

```
X_train = train_data.drop(columns=["cnt"])
y_train = train_data["cnt"]
X_test = test_data.drop(columns=["cnt"])
y_test = test_data["cnt"]
```

b)[3pts] Fit the ordinary least squares regression model to the pre-processed data. Report the coefficient of determination, also known as the R^2 score, of your model.
The R^2 score was calculated using the r2_score function. The R^2 score for the training data was 0.3952, and the score on the Test Data was 0.3769. We observed a low score for both. Later, we discuss that some features were categorical which reduces the accuracy. The code for this section is shown underneath.

```
[82] b, w = ordinary_least_squares_regression(X_train, y_train, use_solver=True)
```

```
[83] print(f"R2 on Training Data is {r2_score(y_train, np.dot(X_train.to_numpy(), w)+b):.4f}")
     print(f"R2 on Test Data is {r2_score(y_test, np.dot(X_test.to_numpy(), w)+b):.4f}")

     R2 on Training Data is 0.3952
     R2 on Test Data is 0.3769
```

c)[5pt] You will find that the fit of the model is not very good. This is in part due to the fact that the dataset contains categorical input variables. So far, your implementation uses these categorical features as continuous inputs, which leads to not so good performance. Instead, it is advised to explicitly encode these input variables as categorical variables. Recall that one way of dealing with categorical variables is to replace them with 1-hot-encoded vectors. The following columns in the dataframe are known to be categorical: season, mnth, hr, weekday, weathersit. Substitute these features with 1-hot-encoded vectors in the dataframe. Use this dataframe for all upcoming questions. Make sure that you split the dataframe as described in Question 3.1 (f)

Using the function get_dummies, the features 'season', 'mnth', 'hr', 'weekday', and 'weathersit' have been replaced with 1-hot-encoded vectors. The code to accomplish this is shown underneath.

```
#switch categorical features to one-hot encoding
X_train_new = pd.get_dummies(data=X_train, columns=["season", "mnth", "hr", "weekday", "weathersit"])
X_test_new = pd.get_dummies(data=X_test, columns=["season", "mnth", "hr", "weekday", "weathersit"])
```

d)[2pt] Re-fit the model with the new data and report the updated R^2 score.
After hot-encoding the categorical features, the model was re-fitted. The R^2 increased for the training and testing data. The R^2 value on the training data is 0.6879 and on the testing data 0.6818. The code for this question is shown underneath.

```
[ ]  b2, w2 = ordinary_least_squares_regression(X_train_new, y_train, use_solver=True)
     print(f"R2 on Training Data is {r2_score(y_train, np.dot(X_train_new.to_numpy(), w2)+b2):.4f}")
     print(f"R2 on Test Data is {r2_score(y_test, np.dot(X_test_new.to_numpy(), w2)+b2):.4f}")

     R2 on Training Data is 0.6879
     R2 on Test Data is 0.6818
```

e)[5pt] Implement the locally weighted regression algorithm as described in Question 2.
The locally weighted regression algorithm was implemented and is shown in the picture underneath.

```
[86] def calc_A(X, x, tau):
       a = [math.exp(-(np.linalg.norm(x-x_i)**2)/(2*tau**2)) for x_i in X]
       b = sum([math.exp(-(np.linalg.norm(x-x_j)**2)/(2*tau**2)) for x_j in X])
       return np.diag([a_i/b for a_i in a])

     def locally_weighted_regression(X, y, x, tau=1, use_solver=True, ignore_small_a=False):
       X = np.insert(X.to_numpy(), 0, values=1, axis=1) # add one column for bias
       y = y.to_numpy()
       A = calc_A(X, x, tau)
       if ignore_small_a:
         A[np.abs(A) < np.finfo(float).eps*100] = 0
       a = np.dot(np.dot(X.T, A), X)
       b = np.dot(np.dot(X.T, A), y)
       if use_solver:
         wb = scipy.linalg.solve(a, b)
       else:
         wb = np.dot(np.linalg.inv(a), b)
       return wb #includes bias in the beginning
```

f)[5pt] Fit he locally weighted regression model to the data with $\tau = 1$. Report the R^2 score of your model. Verify whether and describe how the expected behaviour for $\tau \to 0$ and $\tau \to \infty$ as described in your answer to Question 2 holds.
The results for this question used 200 samples from the full test set. The following image is the code used to call the locally weighted regression algorithm and calculate the R^2 value.

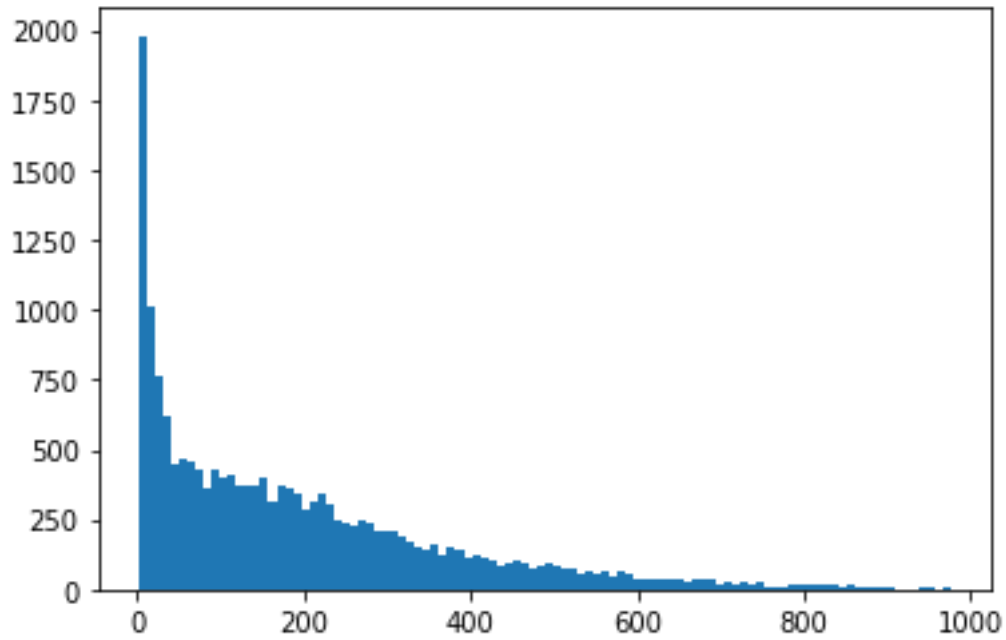
```
[88] # Get the weights and make a prediction for each sample in the test set
     MAX_NUMBER = 200
     y_preds = []
     import time
     s = time.time()
     for x in np.insert(X_test_new.head(MAX_NUMBER).to_numpy(), 0, values=1, axis=1):
       wb = locally_weighted_regression(X_train_new, y_train, x)
       y_pred = np.dot(x, wb)
       print('.', end=' ')
       y_preds.append(y_pred)
     print(f"R2 on Test Data is {r2_score(y_test.head(MAX_NUMBER), y_preds):.4f}")
     print(time.time()-s)
```

The results were the following when $\tau = 1$. The R^2 value increased and is 0.8401. The second simulation was made when $\tau \to 0$, for the simulation, τ was set too 0.001. For this test, the R^2 value was 0.9014. Finally, when $\tau \to \infty$, τ was set too 99999, the R^2 was 0.6879.

g)[pt] Plot a histogram of the target variable. What distribution does the target follow?

The figure underneath illustrates the histogram of the target variable and the code to generate this graph. From the histogram, we can see that there is a right skewness (positive skeweness). The frequency distribution is very asymmetrical

Tau = 1



h)[5pt] Implement the Poisson regression algorithm as describe in Question 2. Since the maximum likelihood estimate is not solvable in closed form, you will need to implement gradient descent. You may use autograd to compute the gradient of the loss function but implement the gradient descent procedure by hand as presented in the tutorial.

The following code is the code used to calculate the Poisson regression algorithm:

Q3 h) Implement the Poisson regression algorithm

[48]
```
import autograd.numpy as np
import autograd

def poissonNegLogLiklihood (w,X,y):
  #w is numpy float vector of dimension len(X[0])
  negLogL = -np.mean(np.dot(X,w)*y - np.exp(np.dot(X,w)))

  return negLogL

#gradient w.r.t  w
#NOTE: df_dx has same arguments as poissonNegLogLiklihood: (w,X,y)
df_dw = autograd.grad(poissonNegLogLiklihood, 0)
```

[49]
```
#implement gradient descent
#------------------------------
X = X_train_new
y = y_train

X = np.insert(X.to_numpy(), 0, values=1.0, axis=1) # add one column for bias
y = y.to_numpy()

# hyperparameters
LEARNING_RATE = 0.001

#initialize w
INITIAL_w = np.asarray([0.0001]*X.shape[1])
w = np.copy(INITIAL_w)

#number of iterations
N_ITER = 50000
```

i)[3pt] Fit the Poisson model to the data. Report the fraction of explained Tweedie deviance also known as the D score, of you model. The D score between the correct target label y and the predicted target label $\hat{y}$ can be calculated as:

$$D(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n} 2(y_i \log\left(\frac{y_i}{\hat{y}_i}\right) + \hat{y}_i - y_i)$$

The following image shows the code used for question i. We got a D score of 80.1%.

Q3 i)

```
[50] for i in range(N_ITER):
         delta = -LEARNING_RATE*df_dw(w,X,y)  # compute gradient times learning rate
         w += delta  # update params
```

```
[51] #predicted value of y^i is \lambda = exp(w^T X[i])
     X = X_test_new
     X = np.insert(X.to_numpy(), 0, values=1.0, axis=1)
     y_pred_poisson = [np.exp(np.dot(w.T,X[i])) for i in range(len(y_test))]

     D_score = d2_tweedie_score(y_test, y_pred_poisson, power=1)
```

```
D_score
```

```
0.8014495612708745
```

Thus, roughly 80% of the tweedy deviance is explained by the model.

j)[5pt] For Linear Regression and Poisson Regression, report the final weights. Which are the most and least significant features in each model? Justify an answer. If the feature is categorical, report both the feature name, as well as the 1-hot index. Write a small explanation why visualizing weight contribution is not meaningful for the locally weighted linear regression.

The picture underneath illustrates the weights obtained for linear regression and poison regression. From the results, we conclude that the most significant feature of the linear regression is **mnth_7** with a coefficient value of -9237 which correspond to the month of July. The least significant feature was windspeed with a coefficient value of -28. Those were the maximum and minimum values obtained for the weights.

For the Poisson Regression, the two most significant features were hour ‘hr_4’ and ‘hr_5’ with coefficient values of -2.175 and -2.793 respectively which correspond to times between 4-5Am and 5-6AM. The least significant feature was the weather ‘weathersit_4’ with a coefficient value of 0.035. Those were the maximum and minimum values obtained for the weights.

For this exercise, visualizing weight for the locally weighted linear regression would not be meaningful since for this algorithm, it only learns weights that are good locally. It requires optimization to fit the weights at run time for each query. Hence, it wouldn’t make sense visualizing all the weights for each query points.

Linear Regression

Final weights:

```
print(np.array(w_ols))
w_ols_abs = abs(np.array(w_ols))
print("MAX", X_train_new.columns[np.argmax(w_ols_abs)-1], max(w_ols_abs))
print("MIN", X_train_new.columns[np.argmin(w_ols_abs)-1], min(w_ols_abs))
```

```
[-1030.96454885    83.95015855  -481.27919214  -448.63355803
   258.88030375   -88.76921633   -28.82720772  4835.23461396
  4878.28311753  4868.46224251  4902.54826255 -9212.06295502
 -9211.19286831 -9205.14172498 -9214.59279014 -9204.26359885
 -9222.36912301 -9237.03181298 -9218.68758567 -9188.68417263
 -9203.48690172 -9225.30099674 -9218.76010782  6809.73157543
  6791.54117174  6782.15371602  6773.93164589  6770.91494452
  6788.16843782  6846.32773943  6983.66763132  7112.74437381
  6971.71760629  6916.11984881  6942.70675405  6975.4210751
  6971.59409369  6951.39565069  6971.31027107  7027.31382045
  7185.30699813  7153.52152484  7039.90038997  6961.08708512
  6913.34543806  6878.78321578  6836.75724844   418.54107335
   876.19515652   875.33129242   881.25976712   877.7476554
   880.57883026   429.25163269 -1898.20712621 -1909.44306178
 -1961.91500566 -1981.07858026]
MAX mnth_7 9237.031812979487
MIN windspeed 28.827207723687458
```

Poisson Regression

Final weights:

```
[ ]  print(w)
     print("MAX", X_train_new.columns[np.argmax(abs(w))-1], max(abs(w)))
     print("MIN", X_train_new.columns[np.argmin(abs(w))-1], min(abs(w)))
```

```
[ 1.93804509  0.46505851  0.28754496  0.46968569  1.14043159 -0.21654684
 -0.11213129  0.22571599  0.52923952  0.48882584  0.69456374  0.05084188
  0.15851651  0.22324384  0.18798777  0.22203052  0.15688732  0.10052232
  0.17199245  0.29500886  0.19465482  0.08868867  0.08877012 -0.66503848
 -1.12429854 -1.5393485  -2.17533397 -2.79320919 -1.6164983  -0.28128309
  0.76081076  1.21988425  0.70995698  0.43659659  0.58716574  0.73438538
  0.7236487   0.64212001  0.72787505  0.92527535  1.33740595  1.2748264
  0.97923054  0.68039228  0.42663015  0.16498463 -0.19583361  0.56418927
  0.1368542   0.13520642  0.15871753  0.15452773  0.17222476  0.61692517
  0.84202877  0.78011702  0.35196607 -0.03576677]
MAX hr_4 2.7932091859680477
MIN weathersit_4 0.035766768442616
```