1.

$$\underbrace{E_x[E(g(x), f(x))]}_{①} = \frac{1}{N}\sum_{i=1}^{N} E(g(x_i), y_i)$$

N: # of training samples
f: unknown function
g: learnable function
E: cost function
y_i: label associated with input x^i

①: Expected value of cost between g(x) and f(x) for every x in the dataset.

②: Approximate expectation by computing a mean over the errors assigning equal weights to each sample.

①: Is it okay to give an equal weight to the cost associated with training sample?

No. Assigning each sample with equal weights is not reasonable since we may find that for some sample points, they are selected with more probability while some other sample points are selected with lower probability. Thus, the weight should be different.

② Given that we established that not every data x is equally likely, is taking the sum of all per-example costs and dividing by N reasonable?

No. Since not all data is equally likely, we might get g(x) that is partially following the actual trend of f(x). For the region which g(x) could possibly predict way off the actual trend, dividing by N is not reasonalb since we won't be capturing the gap within g(x) and f(x) within the region that we "missed". So the result may show a small expected value while the actual result is way bigger.

③ Should we weigh each per example cost differently, depending on how likely each x is?

Yes. In this case, we can have a more identical distributed g(x) and then we'll have a more accurate Expected value of cost.

2. Consider the following model: $Y_i = 5 + 0.5X_i + \varepsilon_i$

$\varepsilon_i \overset{iid}{\sim} N(0,1)$

① What is $E[Y|X=0]$, $E(Y|X=-2)$ and $Var[Y|X]$?

$E(Y|X=0) = 5$ $\qquad$ $E(Y|X=-2) = 4$

$$Var(Y|X) = E((Y - E(Y|X))^2 | X)$$

$$= E(Y^2 + (E(Y|X))^2 - 2 \cdot Y \cdot E(Y|X) \mid X)$$

$$= E(Y^2|X) + E((E(Y|X))^2 | X) - 2E(Y \cdot E(Y|X) | X)$$

$$= E(Y^2|X) + (E(Y|X))^2 E(1|X) - 2E(Y|X)E(Y|X)$$

$$= E(Y^2|X) - E(Y|X)^2$$

$$= 1$$

② What is the probability of $Y > 5$, given $X = 2$?

$$P(Y>5 | X=2)$$
$$= 1 - P(\varepsilon < -1)$$
$$= 1 - 0.1587$$
$$= 0.8413$$

③ If X has a mean of zero and variance of 10, what are $E[Y]$. and $var[Y]$?

$\mu_X = 0$ $\qquad$ $\sigma_X^2 = 10$

$E[Y] = 5$

$$VAR(Y) = Var(5 + 0.5X + \varepsilon)$$
$$= Var(0.5X + \varepsilon) = Var(0.5X) + var(\varepsilon)$$
$$= 0.25\,Var(X) + var(\varepsilon)$$
$$= 0.25(10) + 1 = 3.5$$

④ What is $Cov(X, Y)$?

$$Cov(X,Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

$$= E[XY - XE[Y] - E[X]Y + E[X]E[Y]]$$

$$= E[XY] - E[X\,E[Y]] - E[E[X]\,Y] + E[E[X]\,E[Y]]$$

$$= E[XY] - E[Y]E[X]$$

$$= E[X(5+0.5X)] - 5E[X]$$

$$= E[5X + 0.5X^2] - 5E[X]$$

$$= E[5X] + E[0.5X^2] - 5E[X]$$

$$= 0.5E[X^2]$$

$$= 0.5 \int_{-\infty}^{\infty} x^2 f(x)\,dx$$

$$= 0.5 \int_{-\infty}^{\infty} x^2 (5 + 0.5x)\,dx$$

= Divergent

3. Least Square Regression

Linear Regression Model: $y = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_k x_k + \varepsilon$

$\varepsilon \sim N(0, \sigma^2)$

- Label $\hat{y}$ is drawn from a Gaussian with mean $\theta^T x$ and variance σ^2.

① Given a set of N observations, provide the closed form solution for an ordinary Least Squares estimate $\hat{\theta}$ for the model parameters θ.

Estimating with MLE

$$\begin{aligned}
\mathbf{w} &= \operatorname*{argmax}_{\mathbf{w}} P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n | \mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}} \prod_{i=1}^{n} P(y_i, \mathbf{x}_i | \mathbf{w}) && \text{Because data points are independently sampled.} \\
&= \operatorname*{argmax}_{\mathbf{w}} \prod_{i=1}^{n} P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i | \mathbf{w}) && \text{Chain rule of probability.} \\
&= \operatorname*{argmax}_{\mathbf{w}} \prod_{i=1}^{n} P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i) && \mathbf{x}_i \text{ is independent of } \mathbf{w}\text{, we only model } P(y_i|\mathbf{x}) \\
&= \operatorname*{argmax}_{\mathbf{w}} \prod_{i=1}^{n} P(y_i | \mathbf{x}_i, \mathbf{w}) && P(\mathbf{x}_i) \text{ is a constant - can be dropped} \\
&= \operatorname*{argmax}_{\mathbf{w}} \sum_{i=1}^{n} \log[P(y_i | \mathbf{x}_i, \mathbf{w})] && \text{log is a monotonic function} \\
&= \operatorname*{argmax}_{\mathbf{w}} \sum_{i=1}^{n} \left[\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(e^{-\frac{(\mathbf{x}_i^\top \mathbf{w} - y_i)^2}{2\sigma^2}}\right)\right] && \text{Plugging in probability distribution} \\
&= \operatorname*{argmax}_{\mathbf{w}} -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 && \text{First term is a constant, and } \log(e^z) = z \\
&= \operatorname*{argmin}_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 && \text{Always minimize; } \frac{1}{n} \text{ makes the loss interpretable (average squared error).}
\end{aligned}$$

We are minimizing a *loss function*, $l(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)^2$. This particular loss function is also known as the squared loss or Ordinary Least Squares (OLS). OLS can be optimized with gradient descent, Newton's method, or in closed form.

Closed Form: $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}^\top$ where $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ and $\mathbf{y} = [y_1, \dots, y_n]$.

OR

0

Essentially we're trying to minimize $\epsilon^T\epsilon$ with respect to β.

So we're trying to get

$$\frac{d\epsilon^T\epsilon}{d\beta} = 0$$

$$\frac{d(y - X\beta)^T(y - X\beta)}{d\beta} = 0$$

$$\frac{d}{d\beta}(-2y^T(X\beta) + \beta^T X^T X\beta) = 0$$

$$2(X^T X)\beta = 2X^T y$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

As required, assuming $X^T X$ is invertible.

- OLS method, we assume $Var(\epsilon_i \mid X_i) = \sigma^2$ where σ is constant

- However, when $Var(\epsilon_i \mid X_i) = f(X_i) \neq \sigma^2$, the error term for each X_i has a weight W_i corresponding to it. This is called weighted Least Square Regression.

② Provide a closed form weighted least squares estimate $\hat{\theta}$ for the model parameters θ.

The method of ordinary least squares assumes that there is constant variance in the errors (which is called **homoscedasticity**). The method of **weighted least squares** can be used when the ordinary least squares assumption of constant variance in the errors is violated (which is called **heteroscedasticity**). The model under consideration is

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon^*,$$

where ϵ^* is assumed to be (multivariate) normally distributed with mean vector **0** and nonconstant variance-covariance matrix

$$\begin{pmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{pmatrix}$$

If we define the reciprocal of each variance, σ_i^2, as the weight, $w_i = 1/\sigma_i^2$, then let matrix **W** be a diagonal matrix containing these weights:

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_n \end{pmatrix}$$

The weighted least squares estimate is then

$$\begin{aligned} \hat{\beta}_{WLS} &= \arg\min_{\beta} \sum_{i=1}^{n} \epsilon_i^{*2} \\ &= (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{Y} \end{aligned}$$

4.

Logistic Function: $p(y) = \dfrac{e^{w^T x}}{1 + e^{w^T x}}$

Linear Function: $y = \vec{w}^T x$

Linear Regression: Assumes the presence of a linear relationship between independent and dependent variables. Predict value based on the independent variable

Logistic Regression: Uses the value of independent variable to predict the category of dependent variable

Homework 2: Linear Regression

The is the coding potion of Homework 2. The homework is aimed at testing the ability to deal with a real-world dataset and use linear regression on it.

In [100…

```
import numpy as np
import pandas as pd

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

Load Dataset

Loading the California Housing dataset using sklearn.

In [101…

```
# Load dataset
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

Part 1 : Analyse the dataset

In [102…

```
# Put the dataset along with the target variable in a pandas dataframe
data = pd.DataFrame(housing.data, columns=housing.feature_names)
# Add target to data
data['target'] = housing['target']
data.head()
```

Out[102]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.:
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.:
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.:
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.:
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.:

Part 1a : Check for missing values in the dataset

The dataset might have missing values represented by a `NaN`. Check if the dataset has such missing values.

In [103…

```
# Check for missing values
def is_null(dataframe):
    """
    This function takes as input a pandas dataframe and outputs whether t
    dataframe has missing values. Missing values can be detected by check
    for the presence of None or NaN. inf or -inf must also be treated as

    Input:
        dataframe: Pandas dataframe
    Output:
        Return True is there are missing value in the dataframe. If not,
    """
    # YOUR CODE HERE
    list_featurenames=[]
    for i in dataframe.columns.values:
        list_featurenames.append(i)
    count=0
    result=0
    while (count < len(list_featurenames)):
          marks_list = dataframe[list_featurenames[count]].tolist()
          if ("None" in marks_list) or ("NaN" in marks_list) or ("inf" in
              result+=1
          else:
              result+=0
          count+=1
    if(result !=0):
        return True
    else:
        return False
    raise NotImplementedError()
```

In [104…

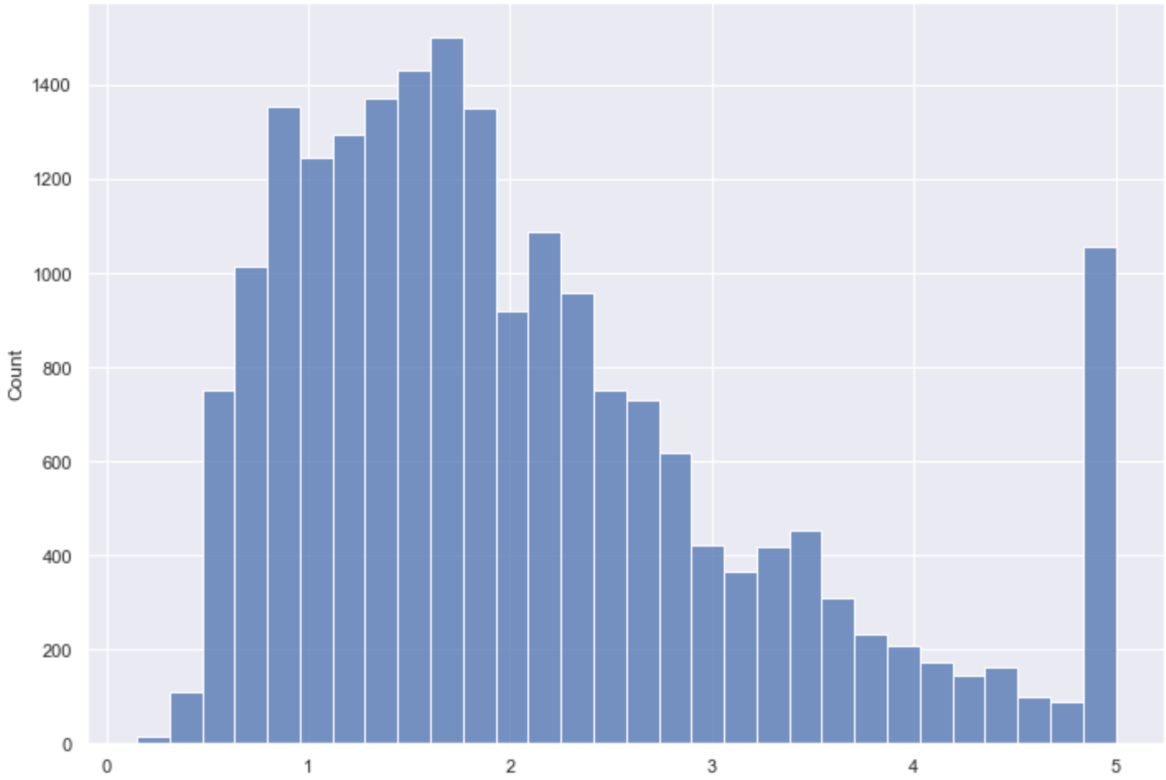
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

# This dataset has no null values; you can run this cell as a sanity chec
print(f"The data has{'' if is_null(data) else ' no'} missing values.")
assert not is_null(data)
```

```
The data has no missing values.
```

Part 1b: Studying the distribution of the target variable

Plot the histogram of the target variable over a fixed number of bins (say, 30).

Example histogram output:



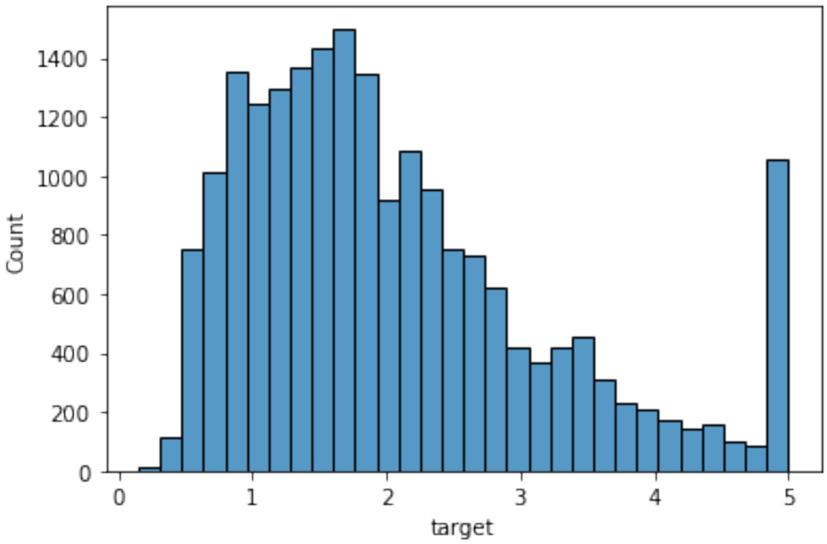
Hint: Use the histogram plotting function available in Seaborn in Matplotlib.

In [105…

```
# Plot histogram of target variable

# YOUR CODE HERE
sns.histplot(data=data, x="target",bins=30)
```

Out[105]:

```
<AxesSubplot:xlabel='target', ylabel='Count'>
```



Part 1c: Plotting the correlation matrix

Given the dataset stored in the `data` variable, plot the correlation matrix for the dataset. The dataset has 9 variables (8 features and one target variable) and thus, the correlation matrix must have a size of `9x9`.

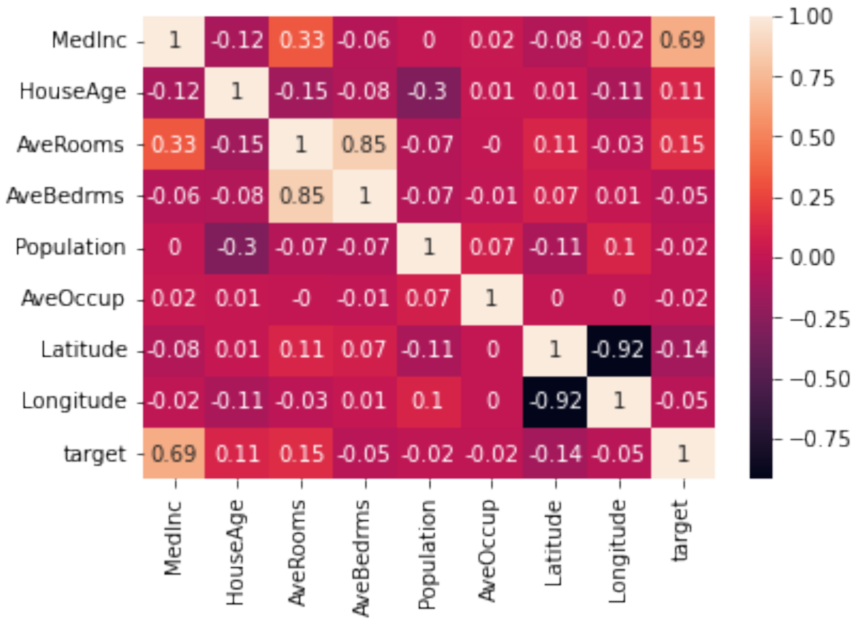
Hint: You may use the correlation matrix computation of a dataset provided by the `pandas` library.

Link: What is a correlation matrix?

In [106…

```
# Correlation matrix
def get_correlation_matrix(dataframe):
    """
    Given a pandas dataframe, obtain the correlation matrix
    computing the correlation between the entities in the dataset.

    Input:
        dataframe: Pandas dataframe
    Output:
        Return the correlation matrix as a pandas dataframe, rounded off
    """
    # YOUR CODE HERE
    corrMatrix = dataframe.corr().round(2)
    return corrMatrix
    raise NotImplementedError()

# Plot the correlation matrix
correlation_matrix = get_correlation_matrix(data)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

Out[106]: <AxesSubplot:>



In [107…

```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

# You can check your output against the expected correlation matrix below
ground_truth = np.array([
    [1.0, -0.12, 0.33, -0.06, 0.0, 0.02, -0.08, -0.02, 0.69],
    [-0.12, 1.0, -0.15, -0.08, -0.3, 0.01, 0.01, -0.11, 0.11],
    [0.33, -0.15, 1.0, 0.85, -0.07, 0.0, 0.11, -0.03, 0.15],
    [-0.06, -0.08, 0.85, 1.0, -0.07, -0.01, 0.07, 0.01, -0.05],
    [0.0, -0.3, -0.07, -0.07, 1.0, 0.07, -0.11, 0.1, -0.02],
    [0.02, 0.01, 0.0, -0.01, 0.07, 1.0, 0.0, 0.0, -0.02],
    [-0.08, 0.01, 0.11, 0.07, -0.11, 0.0, 1.0, -0.92, -0.14],
    [-0.02, -0.11, -0.03, 0.01, 0.1, 0.0, -0.92, 1.0, -0.05],
    [0.69, 0.11, 0.15, -0.05, -0.02, -0.02, -0.14, -0.05, 1.0],
])
assert np.allclose(ground_truth, get_correlation_matrix(data).to_numpy(),
```

Part 1d: Extracting relevant variables

Based on the correlation matrix obtained in the previous part, identify the top-4 most relevant features from the dataset for predicting the target variable.

MedInc, AveRooms, Latitude, HouseAge,

Part 2: Data Manipulation

This section is focused on arranging the dataset in a format suitable for training the linear regression model.

Part 2a: Normalize the dataset

Find the mean and standard deviation corresponding to each feature and target variable in the dataset. Use the values of the mean and standard deviation to normalize the dataset.

In [108…

```
features = np.concatenate([data[name].to_numpy()[:, None] for name in hou
target = housing['target']

# Normalize data

def normalize(features, target):
    # YOUR CODE HERE
    feature_mean = np.mean(features, axis=0);
    target_mean = np.mean(target);

    std_feature = np.std(features,axis=0);
    std_target = np.std(target);

    normalized_feature = (features - feature_mean)/std_feature;
    normalized_target = (target - target_mean)/std_target;

    return normalized_feature, normalized_target




features_normalized, target_normalized = normalize(features, target)
print(features_normalized.shape, target_normalized.shape)
```

```
(20640, 8) (20640,)
```

In [109…

```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.
assert all(np.abs(features_normalized.mean(axis=0)) < 1e-2), "Mean should
assert all(np.abs(features_normalized.std(axis=0) - 1) < 1e-2), "Standard
assert np.abs(target_normalized.mean(axis=0)) < 1e-2, "Mean should be clo
assert np.abs(target_normalized.std(axis=0) - 1) < 1e-2, "Standard deviat
```

Part 2b: Train-Test Split

Use the train-test split function from `sklearn` and execute a 80-20 train-test split of the dataset.

In [110…

```
# YOUR CODE HERE
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(features_normalized,
```

In [111…

```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

# Sanity checking:
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(16512, 8)
(4128, 8)
(16512,)
(4128,)
```

Part 3: Linear Regression

In this part, a linear regression model is used to fit the dataset loaded and normalized above.

Part 3a: Code for Linear Regression

Implement a closed-form solution for ordinary least squares linear regression in `MyLinearRegression`, and print out the RMSE and R^2 between the ground truth and the model prediction.

In [112…

```
class MyLinearRegression:
    def __init__(self):
        self.theta = None

    def fit(self, X, Y):
        # Given X and Y, compute theta using the closed-form solution for
        # YOUR CODE HERE
         self.theta = np.linalg.inv(X.T @ X) @ X.T @ Y
         # normal equation
         # theta_best = (X.T * X)^(-1) * X.T * y



    def predict(self, X):
        # Predict Y for a given X
        # YOUR CODE HERE
        return X @ self.theta
```

In [113…

```
# Train the model on (X_train, Y_train) using Linear Regression
my_model = MyLinearRegression()
my_model.fit(X_train, Y_train)
```

In [114…
```
from sklearn.metrics import mean_squared_error, r2_score

# Compute train RMSE using (X_train, Y_train)
y_train_predict = my_model.predict(X_train)
train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
train_r2 = r2_score(Y_train, y_train_predict)
print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(train_rmse))
print('R2 score is {}'.format(train_r2))
print("\n")

# Compute test RMSE using (X_test, Y_test)
y_test_predict = my_model.predict(X_test)
test_rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
test_r2 = r2_score(Y_test, y_test_predict)
print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(test_rmse))
print('R2 score is {}'.format(test_r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.6274618681586667
R2 score is 0.605104535189154


The model performance for testing set
--------------------------------------
RMSE is 0.627902271715941
R2 score is 0.61032171028735
```

Part 3b: Compare with LinearRegression from sklearn.linear_model

Use LinearRegression from the `sklearn` package to fit the dataset and compare the results obtained with your own implementaion of Linear Regression.

The linear regressor should be named `model` for the cells below to run properly.

In [115…
```
# YOUR CODE HERE
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=False)
model.fit(X_train, Y_train)
```

Out[115]:
```
LinearRegression(fit_intercept=False)
```

In [116...

```
# model evaluation for training set
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict
sklearn_train_r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_train_rmse))
print('R2 score is {}'.format(sklearn_train_r2))
print("\n")

# model evaluation for testing set
y_test_predict = model.predict(X_test)
sklearn_test_rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
sklearn_test_r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_test_rmse))
print('R2 score is {}'.format(sklearn_test_r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.6274618681586667
R2 score is 0.605104535189154


The model performance for testing set
--------------------------------------
RMSE is 0.627902271715941
R2 score is 0.6103217102873499
```

Part 3c: Analysis Linear Regression Performance

In this section, provide the observed difference in performance along with an explanation of the following:

- Difference between training between unnormalized and normalized data.
- Difference between training on all features versus training on the top-5 most relevant features in the dataset.
- Difference between (1) training on all features (unnormalized), (2) training on top-4 unnormalized features, and (3) training on top-4 normalized features.

Write your answer below.

1. Difference between training between unormalized data and normalized data

In [117…
```
#unormalized data:
features = np.concatenate([data[name].to_numpy()[:, None] for name in hou
target = housing['target']
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(features, target, tes
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=False)
model.fit(X_train, Y_train)

y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict
sklearn_train_r2 = r2_score(Y_train, y_train_predict)

print("The model performance for unormalized training set ")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_train_rmse))
print('R2 score is {}'.format(sklearn_train_r2))
print("\n")

y_test_predict = model.predict(X_test)
sklearn_test_rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
sklearn_test_r2 = r2_score(Y_test, y_test_predict)

print("The model performance for unormalized testing set")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_test_rmse))
print('R2 score is {}'.format(sklearn_test_r2))
```

```
The model performance for unormalized training set
--------------------------------------
RMSE is 0.7797615813170762
R2 score is 0.5471380488445526


The model performance for unormalized testing set
--------------------------------------
RMSE is 0.7679962114768092
R2 score is 0.5416367242891427
```

notice that the normalized results are The model performance for training set -------------------------------------- RMSE is 0.6317291557308954 R2 score is 0.6029200869553969 Thus we can see that for the unormalized training set, RMSE is larger and R2 score is lower. Lower RMSE means a better fit, thus the normalized data is a better fit.

1. Difference between training on all features vs training on the top 5 most relevant features in the dataset.

In [118...

```
features = np.concatenate([data[name].to_numpy()[:, None] for name in ['M
target = housing['target']
features_normalized, target_normalized = normalize(features, target)
X_train, X_test, Y_train, Y_test = train_test_split(features_normalized,
model = LinearRegression(fit_intercept=False)
model.fit(X_train, Y_train)

# model evaluation for training set
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict
sklearn_train_r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_train_rmse))
print('R2 score is {}'.format(sklearn_train_r2))
print("\n")

# model evaluation for testing set
y_test_predict = model.predict(X_test)
sklearn_test_rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
sklearn_test_r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_test_rmse))
print('R2 score is {}'.format(sklearn_test_r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.6716294887222827
R2 score is 0.5468732701874435


The model performance for testing set
--------------------------------------
RMSE is 0.7015397835433674
R2 score is 0.5163663278027684
```

notice that the training on all features is

The model performance for training set

RMSE is 0.6317291557308954 R2 score is 0.6029200869553969

Here we see that the training on top 5 relevant features have a higher RMSE.

3. Difference between (1) training on all features (unnormalized), (2) training on top-4 unnormalized features, and (3) training on top-4 normalized features.

In [119…

```
#1 training on all features(unormalized)
#The model performance for unormalized training set
#--------------------------------------
#RMSE is 0.7762058370256328
#R2 score is 0.5466384223187377


#The model performance for unormalized testing set
#--------------------------------------
#RMSE is 0.7820769711201272
#R2 score is 0.5441477573183839



#2.training on top-4 unormalized features
features = np.concatenate([data[name].to_numpy()[:, None] for name in ['M
target = housing['target']
#features_normalized, target_normalized = normalize(features, target)
X_train, X_test, Y_train, Y_test = train_test_split(features, target, tes
model = LinearRegression(fit_intercept=False)
model.fit(X_train, Y_train)

# model evaluation for training set
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict
sklearn_train_r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set on unormalized top 4")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_train_rmse))
print('R2 score is {}'.format(sklearn_train_r2))
print("\n")

# model evaluation for testing set
y_test_predict = model.predict(X_test)
sklearn_test_rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
sklearn_test_r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set on unormalized top 4")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_test_rmse))
print('R2 score is {}'.format(sklearn_test_r2))
```

```
The model performance for training set on unormalized top 4
--------------------------------------
RMSE is 0.8099595270729483
R2 score is 0.5094564505381836


The model performance for testing set on unormalized top 4
--------------------------------------
RMSE is 0.7896908931461009
R2 score is 0.523323206456513
```

In [120…

```
#3.Training on top 4 normalize
features = np.concatenate([data[name].to_numpy()[:, None] for name in ['M
target = housing['target']
features_normalized, target_normalized = normalize(features, target)
X_train, X_test, Y_train, Y_test = train_test_split(features_normalized,
model = LinearRegression(fit_intercept=False)
model.fit(X_train, Y_train)

# model evaluation for training set
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict
sklearn_train_r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set on normalized top 4")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_train_rmse))
print('R2 score is {}'.format(sklearn_train_r2))
print("\n")

# model evaluation for testing set
y_test_predict = model.predict(X_test)
sklearn_test_rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
sklearn_test_r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set on normalized top 4")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_test_rmse))
print('R2 score is {}'.format(sklearn_test_r2))
```

```
The model performance for training set on normalized top 4
--------------------------------------
RMSE is 0.6918283242712713
R2 score is 0.5241016932078433


The model performance for testing set on normalized top 4
--------------------------------------
RMSE is 0.7038972671429633
R2 score is 0.49285515384303724
```

As we can see, the RMSE on normalized top 4 is the lowest which indicates that it is the best model among these three.

In [ ]: