Module 5 exercises: Regression

Dataset characteristics

California Housing dataset characteristics:

- Number of Instances: 20640
- Number of Attributes: 8 numeric, predictive attributes and the target
- Attribute Information:
  - MedInc median income in block group
  - HouseAge median house age in block group
  - AveRooms average number of rooms per household
  - AveBedrms average number of bedrooms per household
  - Population block group population
  - AveOccup average number of household members
  - Latitude block group latitude
  - Longitude block group longitude

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average number of rooms and bedrooms in this dataset are provided per household, these columns may take surpinsingly large values for block groups with few households and many empty houses, such as vacation resorts.

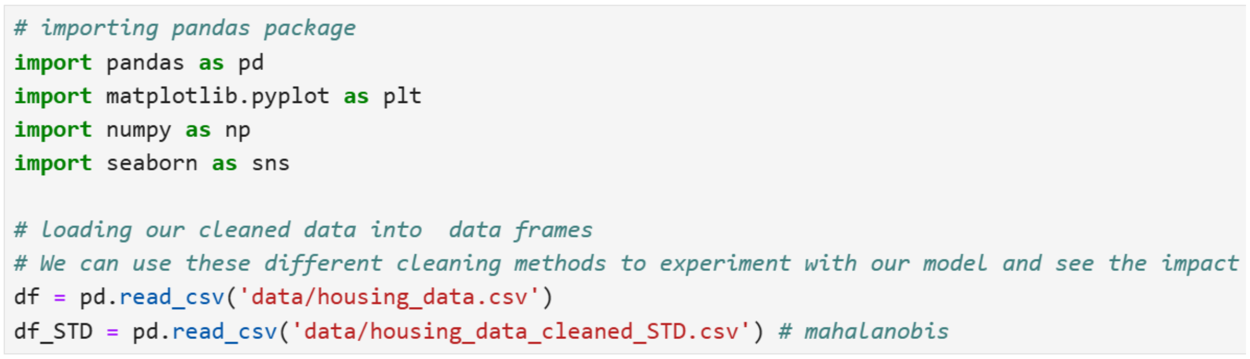
The goal of this notebook is to build a Linear Regression Model on the California Housing Dataset. We will be trying to predict the MedianPrice per house in each block. Denoted as the target column. This price is in $100,000s

We have two datasets available:

- `housing_data.csv` is the original dataset
- `housing_data_cleaned_STD.csv` is the same dataset but having had 114 rows removed with missing AveRooms. It has also had all points above or below 3 standard deviations removed.

Exercise 1 – Setup

1. Import the following packages:
    i. Pandas
    ii. matplotlib
    iii. numpy
    iv. seaborn
2. Load the following data file into a `DataFrame`:
    i. `housing_data.csv`

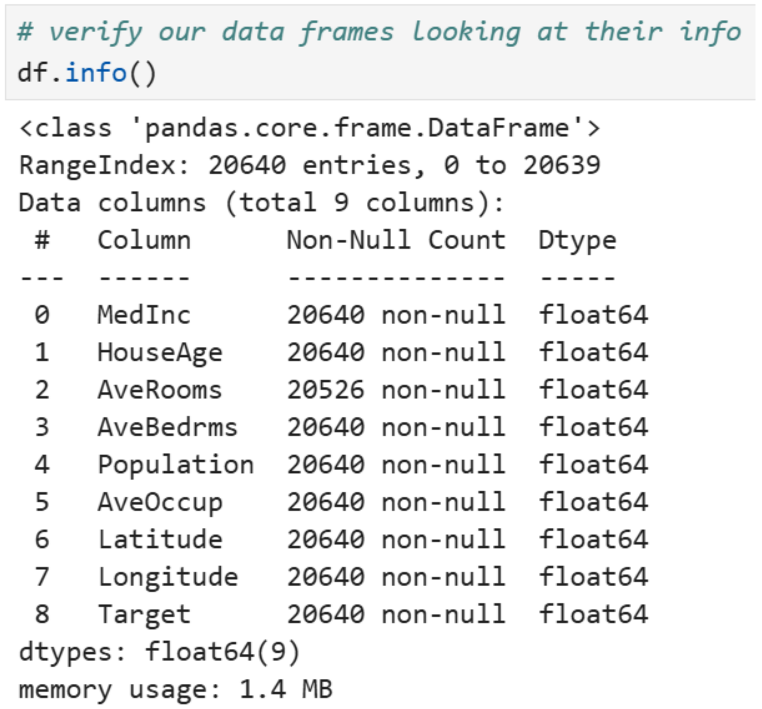


```python
# importing pandas package
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# loading our cleaned data into  data frames
# We can use these different cleaning methods to experiment with our model and see the impact
df = pd.read_csv('data/housing_data.csv')
df_STD = pd.read_csv('data/housing_data_cleaned_STD.csv') # mahalanobis
```

Exercise 2 – EDA

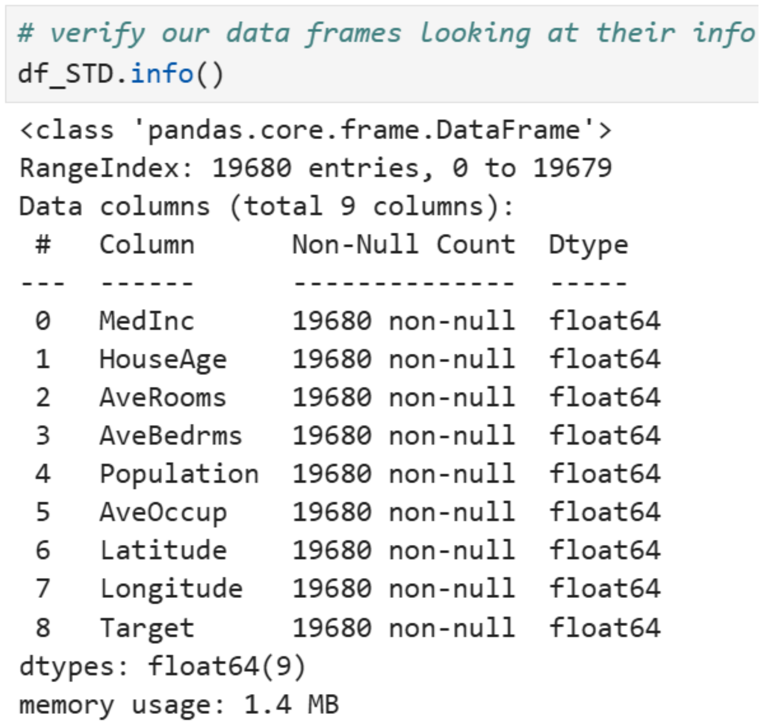
Familiarise yourself with the dataset. Seek to understand:

- the types of data.
- distributions of features.
- obvious patterns.
- errors, nulls, outliers.



```python
# verify our data frames looking at their info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20526 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   Target      20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```



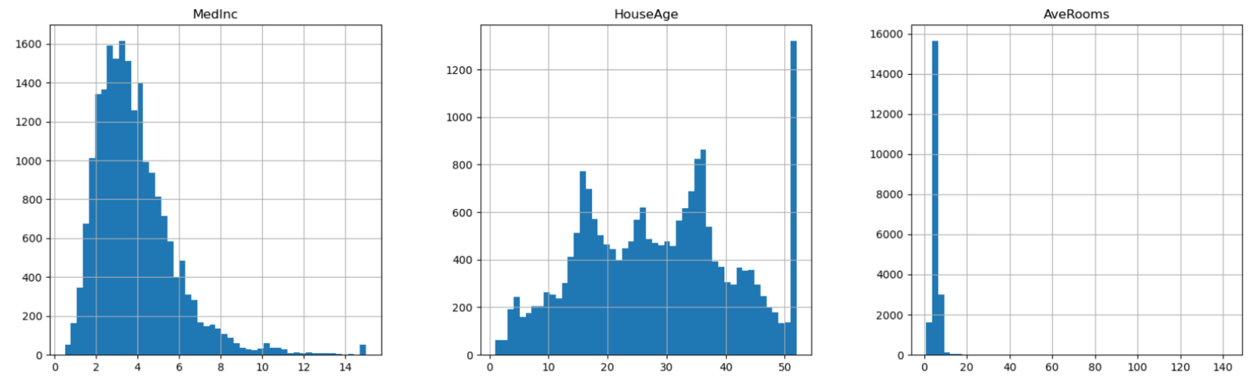
```python
# verify our data frames looking at their info
df_STD.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19680 entries, 0 to 19679
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      19680 non-null  float64
 1   HouseAge    19680 non-null  float64
 2   AveRooms    19680 non-null  float64
 3   AveBedrms   19680 non-null  float64
 4   Population  19680 non-null  float64
 5   AveOccup    19680 non-null  float64
 6   Latitude    19680 non-null  float64
 7   Longitude   19680 non-null  float64
 8   Target      19680 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

```
#viewing the head of the df
df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
# viewing the distributions
df.hist(bins = 50, figsize=(20,20));
```



Etc.

Exercise 3 – Assumptions

1. Select the features you would like to use to build a regression model.
2. Determine whether any assumtions of linear regression would be violated by using those features.

Linear regression assumes that each variableis linearly correlated with the target variable

- We need to investigate this

No multi-colinearity, i.e none of the features are highly correlated with each other, makes regression bad

- We need to investigate this also

Independence

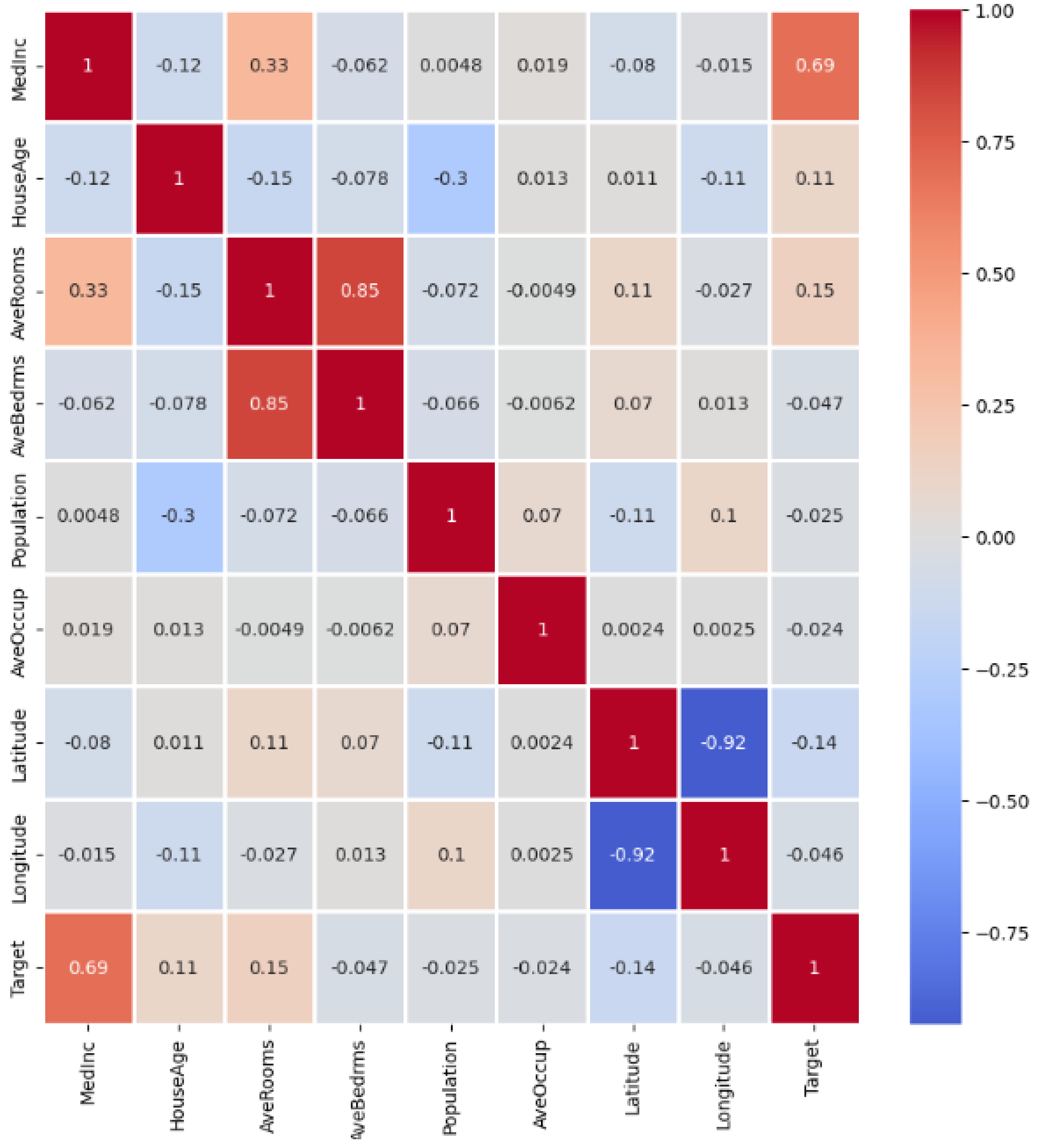
- We will assume this to be the case
- (can use the Durbin-Watson test, to asses, but we will not be covering this)

There are some other assumptions that we can test for after fitting the model

- Homoscedasticity, by using Residual vs Predicted value plots
- Multivariate Normality, using Q-Q plots
- (There are more formal tests for Multivariate Normality but we will not be covering those, namely Shapiro-Wilk, Kolmogorov-Smironov, Jarque-Barre, or D'Agostino-Pearson)

```
# plotting scatter of all columns against eachother
pd.plotting.scatter_matrix(df,figsize = (20,20));
```

```
# Creating an axes on which to draw the heatmap
fig, ax =  plt.subplots(figsize = (10,10))

# Note: there is an exercise sheet on Matplot lib which you may find useful in understanding about figures and axes

# Plotting the heatmap on the axes
sns.heatmap(df.corr(method='pearson'),
            annot=True,
            center=0.0,
            linewidths=0.8,
            ax=ax,
            cmap='coolwarm');
```



3. How might you mitigate these problems?

   Exclude or combine colinear features

Exercise 4 – Feature engineering

1. Inspect the data. Are there any features that could be created by combining existing ones together?
2. Try to construct a novel feature from existing ones. Bedroom ratio may be predictive, so we choose to incorporate it.

```
df["BedroomRatio"] = df["AveBedrms"]/df["AveRooms"]
```

Exercise 5 – Preprocessing

1. So we can evaluate our model, we split our data into a training and testing dataset. Why do we do this?

   To understand how well the model learns the problem i.e. generalises to unseen data.

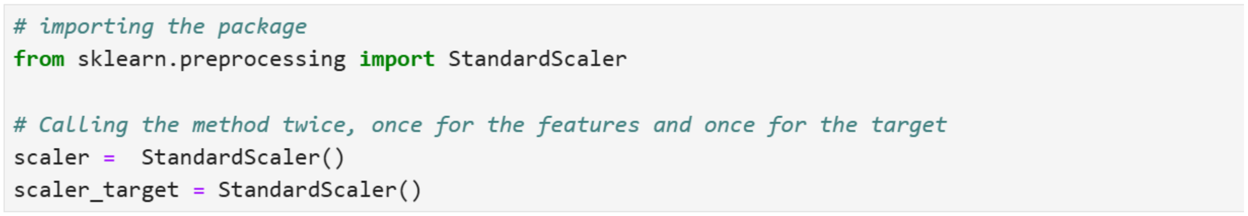
2. Use scikit-learn's `train_test_split` function, found in the `preprocessing` module, to do this. Set `random_state` to 42 and `test_size` to 0.30 You should end up with 4 sets of data after doing so: `X_train`, `X_test`, `y_train`, `y_test`.

```
# Importing package
from sklearn.model_selection import train_test_split

X = df[['MedInc',
        'HouseAge',
        'AveRooms',
        'AveBedrms',
        'Population',
        'AveOccup',
        'Latitude',
        'Longitude',
        'BedroomRatio']]

y = df['Target']

#Splitting the data into test and train sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

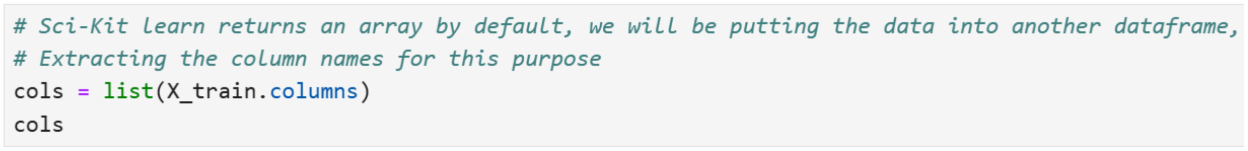
3. We then make the decicision to scale the data. Why might we choose to scale the data in a regression modelling project?

   To aid result interpretation, remove unhelpful intercept, or obtain consistent behaviour from regularised models.

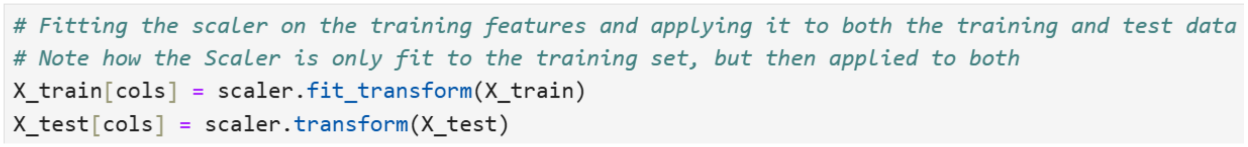
4. Create, fit, and apply two StandardScalers, also found in preprocessing, to `X_train` and `y_train`.



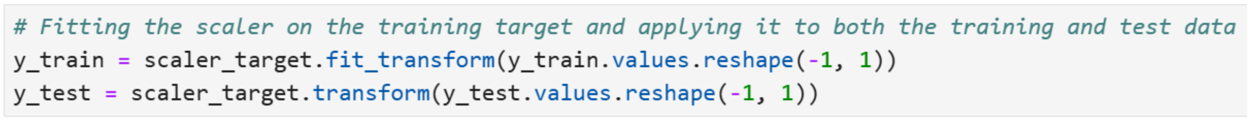
```
# importing the package
from sklearn.preprocessing import StandardScaler

# Calling the method twice, once for the features and once for the target
scaler =  StandardScaler()
scaler_target = StandardScaler()
```



```
# Sci-Kit learn returns an array by default, we will be putting the data into another dataframe,
# Extracting the column names for this purpose
cols = list(X_train.columns)
cols
```



```
# Fitting the scaler on the training features and applying it to both the training and test data
# Note how the Scaler is only fit to the training set, but then applied to both
X_train[cols] = scaler.fit_transform(X_train)
X_test[cols] = scaler.transform(X_test)
```



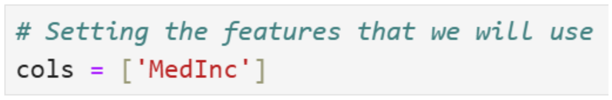
```
# Fitting the scaler on the training target and applying it to both the training and test data
y_train = scaler_target.fit_transform(y_train.values.reshape(-1, 1))
y_test = scaler_target.transform(y_test.values.reshape(-1, 1))
```

5. Apply the one used for X to `X_test` and the one usef for y to `y_test`. Why might we do this?

   We don't want any information from the test data to be passed to the model inadvertently. The mean and std used to scale test must therefore be derived from train.

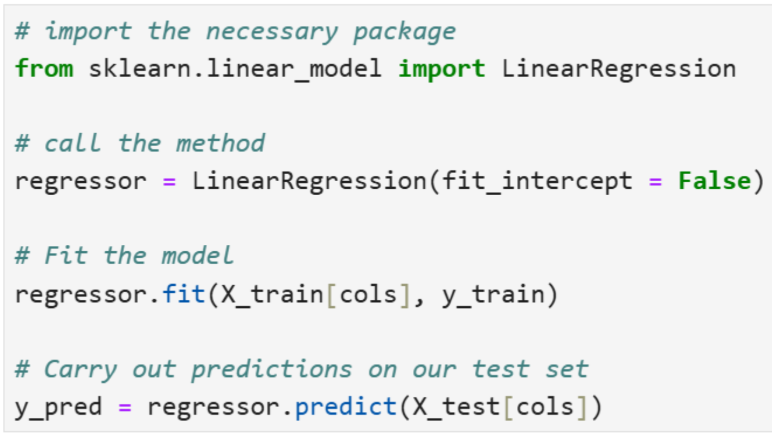
Exercise 6 – Building your first model

1. After preprocessing the dataset, we can fit our regression model. Import the `LinearRegression` model from scikit-learn's `linear_model` module.



```
# Setting the features that we will use
cols = ['MedInc']
```

2. Create a `LinearRegression` model, setting `fit_intercept=False`. Then use the `fit` method to train the model on `X_train` and `y_train`.
3. Generate predictions from the model for `X_test` using the `predict` method, storing them in the variable `y_pred`.



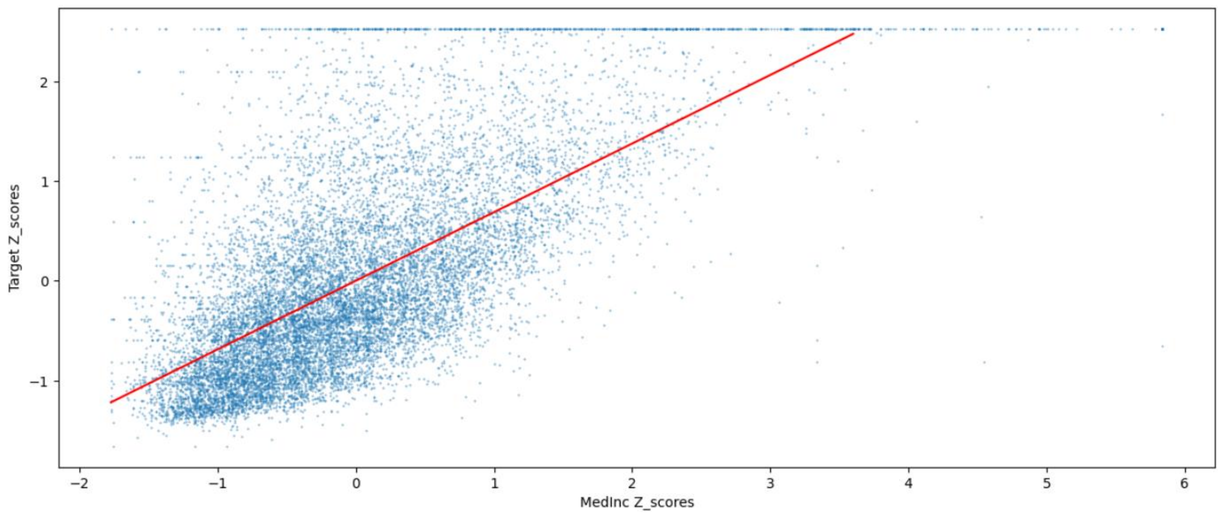
```
# import the necessary package
from sklearn.linear_model import LinearRegression

# call the method
regressor = LinearRegression(fit_intercept = False)

# Fit the model
regressor.fit(X_train[cols], y_train)

# Carry out predictions on our test set
y_pred = regressor.predict(X_test[cols])
```

Exercise 7 - Plotting your model

Run the below code to plot your model against the training data. How well do you think it fits?

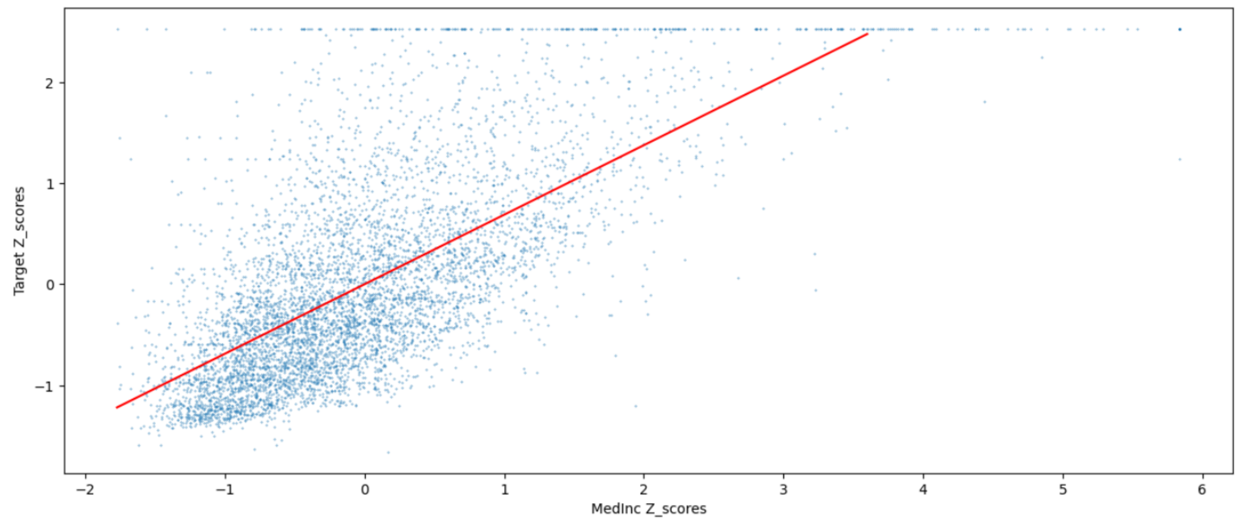
Note: This code only works when `cols=['MedInc']`, and the model has been trained when that was the case. It will need to be adapted if other features have been used.

```
plt.figure(figsize=(15, 6))
# Plotting MedInc against the Target
plt.scatter(X_train[cols], y_train, s=0.1)

# Selecting x values to input into the model
# Started at the min value of MedInc that we have,
# the upper limit was found by experimenitng so as not to exceed the y value
# in this case the linear model is very poor at predicting anything over a st
andardised x value of over 3
x = np.linspace(X_train[cols].min(), 3.6, 100)

# Using the gradient and intercept from the model to plot
# The intercept is zero but it is added for completeness
# y = m              *x+ c
y = (regressor.coef_)*x+(regressor.intercept_)

# # Plotting the linear model
plt.plot(x, y, '-r')

# # Adding labels to each axis
plt.xlabel('MedInc Z_scores')
plt.ylabel('Target Z_scores')

# calling the plot
plt.show()
```



Run the below code to plot your model against the training data. How well do you think it fits?

```
plt.figure(figsize=(15, 6))
# Plotting MedInc against the Target
plt.scatter(X_train[cols], y_train, s=0.1)

# Selecting x values to input into the model
# Started at the min value of MedInc that we have,
# the upper limit was found by experimenitng so as not to exceed the y value
# in this case the linear model is very poor at predicting anything over a st
andardised x value of over 3
x = np.linspace(X_train[cols].min(), 3.6, 100)

# Using the gradient and intercept from the model to plot
# The intercept is zero but it is added for completeness
# y = m              *x+ c
y = (regressor.coef_)*x+(regressor.intercept_)

# # Plotting the linear model
plt.plot(x, y, '-r')

# # Adding labels to each axis
plt.xlabel('MedInc Z_scores')
plt.ylabel('Target Z_scores')

# calling the plot
plt.show()
```



Exercise 8 – Evaluating your model

Print the following error metrics for your model. To make them more interpretable, you may wish to invert the scaling of the target variable using `.inverse_transform`:

- `mean_absolute_error`
- `mean_squared_error`
- root mean squared error
- `r2_score`

```
# first we need to reverse the transformation so we have meanginful results
# calling inverse_transform on our scaler
y_true = scaler_target.inverse_transform(y_test) # <- observed y values
y_pred_inverse = scaler_target.inverse_transform(y_pred) # <- predicted y
```

```python
# Import the necessary packages
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Calculating the errors
mae = mean_absolute_error(y_true, y_pred_inverse)
mse = mean_squared_error(y_true, y_pred_inverse)
rmse = np.sqrt(mse)

# Calculting the R^2
r2 = r2_score(y_true, y_pred_inverse)
```

```python
# Viewing these results
print('MAE = ', mae.round(3))
print('MSE = ', mse)
print('RMSE = ', rmse)
print('R^2 = ', r2)
```

```
MAE =  0.623
MSE =  0.6917979868048499
RMSE =  0.8317439430526982
R^2 =  0.47293192589970245
```

How well does the model perform? Would you use it?

Not accurate in most cases,

```
On average, our predictions for Median House Price are off by $62315.593
```

Exercise 9 – Next steps

What would you do to improve the performance of the model you built?

Lots of room for improvement by trying different models, engineering new features etc.

Extension task

You can try rebuilding the model using the dataset which has had outliers removed to see the difference in model performance, or you could decide which points are outliers yourself and model that data.

Write a brief summary of what impact the removal of outliers has had on the model. Would the model be better or worse at predicting the median price of certain types of houses?

Has its ability to predict one kind of house increased? Has another decreased?