In [ ]:
```
import pandas as pd
import numpy as np
```

Outline

1. Exploratory Data Analysis (EDA)

In [ ]:
```
df = pd.read_csv('data/housing.csv')
print("{} : \n {} \n".format('total houses (rows)', len(df)))
display(df.head(2))
```

```
total houses (rows) :
 20640
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_valu
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500

In [ ]:
```
near_ocean= df['ocean_proximity'] == '<1H OCEAN'
inland = df['ocean_proximity'] == 'INLAND'

houses = df[near_ocean | inland]
del houses['ocean_proximity']

houses.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_
701	-121.97	37.64	32.0	1283.0	194.0	485.0	171.0	6.0574	43
830	-121.99	37.61	9.0	3666.0	711.0	2341.0	703.0	4.6458	21
859	-121.97	37.57	21.0	4342.0	783.0	2172.0	789.0	4.6146	24
860	-121.96	37.58	15.0	3575.0	597.0	1777.0	559.0	5.7192	28
861	-121.98	37.58	20.0	4126.0	1031.0	2079.0	975.0	3.6832	21

- **Question 1**

There's one feature with missing values. What is it?

- `total_rooms`
- **`total_bedrooms`**
- `population`
- `households`

In [ ]:
```
# Missing 157 values in total_bedrooms
houses.isna().sum()
```

Out[ ]:
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        157
population              0
households              0
median_income           0
median_house_value      0
dtype: int64
```

- **Question 2**

What's the median (50% percentile) for variable `'population'`?

- 995
- 1095

- **1195**
- 1295

In [ ]:
```
print('Population median = ' , houses['population'].median())
```

```
Population median =  1195.0
```

2. Data Splitting

Prepare and split the dataset

- Shuffle the dataset (the filtered one you created above), use seed 42.
- Split your data in train/val/test sets, with $60\%/20\%/20\%$ distribution.
- Apply the log transformation to the **median_house_value** variable using the `np.log1p()` function.

In [ ]:
```
# Distributions of data
n = len(houses)
n_train = int(0.6*n)
n_val = int(0.2*n)
#n_test = int(0.2*n)

# Shuffle data
np.random.seed(42)
idx = np.arange(n)
np.random.shuffle(idx)
houses_shuffled = houses.iloc[idx]

# Split data
X_train = houses_shuffled.iloc[:n_train].copy()
X_val = houses_shuffled.iloc[n_train:n_train + n_val].copy()
X_test = houses_shuffled.iloc[n_train+n_val:].copy()

# Apply log transformation
Y_train = np.log1p( X_train['median_house_value'] ).values
Y_val = np.log1p( X_val['median_house_value'] ).values
Y_test = np.log1p( X_test['median_house_value']).values

# To avoid accidentally using the target variable
del X_train['median_house_value']
del X_val['median_house_value']
del X_test['median_house_value']
```

3. Linear Regression

Consider a hypothetical dataset with multiple features $X_1, \cdots, X_d$ and a target variable Y as shown:

$$\left(\begin{array}{c|cccc|c} & X_0 & X_1 & \cdots & X_d & Y \\ \hline x_1 & 1 & x_{11} & \cdots & x_{1d} & y_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n & 1 & x_{n1} & \cdots & x_{nd} & y_n \end{array}\right).$$

Here, each row vector $\mathbf{x}_i = (1, x_{i1}, \ldots, x_{id})$ represents an instance of the dataset with $d+1$ values. The first column, X_0, is the intercept term and is set to 1 for all instances. The dataset is separated in a feature matrix $\mathbf{X}$ and a target vector $\mathbf{Y}$:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{pmatrix} \quad \text{and} \quad \mathbf{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

The linear regression model for multiple features $X_1, \ldots, X_d$ can be represented as:

$$\hat{\mathbf{Y}} = \mathbf{X}\mathbf{w}$$

Here, $\mathbf{w} = (w_0, \ldots, w_d)^T$ denotes the column vector of weights that the model seeks to learn for optimal regression. $\hat{\mathbf{Y}}$ represents the predicted values, and $\mathbf{Y}$ stands for the true values (or target values). The error function commonly used is the sum of squared errors (SSE):

$$\text{SSE} = \sum_{i}^{n} ||\epsilon_i||^2 = \epsilon\epsilon^T$$

where $\epsilon = \hat{\mathbf{Y}} - \mathbf{Y}$.

Rearranging the error function gives:

$$
\begin{aligned}
\text{SSE} &= (\hat{\mathbf{Y}} - \mathbf{Y})^T(\hat{\mathbf{Y}} - \mathbf{Y}) \\
&= \mathbf{Y}^T\mathbf{Y} - 2\mathbf{Y}^T\hat{\mathbf{Y}} + \hat{\mathbf{Y}}^T\hat{\mathbf{Y}} \\
&= \mathbf{Y}^T\mathbf{Y} - 2\mathbf{Y}^T(\mathbf{X}\mathbf{w}) + (\mathbf{X}\mathbf{w})^T(\mathbf{X}\mathbf{w}) \\
&= \mathbf{Y}^T\mathbf{Y} - 2\mathbf{w}^T(\mathbf{X}^T\mathbf{Y}) + \mathbf{w}^T(\mathbf{X}^T\mathbf{X})\mathbf{w}
\end{aligned}
$$

The goal is to minimize this error to optimize the ideal weights. Instead of using gradient descent, we take the derivative of the SSE with respect to the weights, set it equal to zero, and find a local minimum of the SSE function. This leads to the following optimal weights for the optimization problem:

$$
\begin{aligned}
\frac{\partial(\text{SSE})}{\partial\mathbf{w}} &= -2\mathbf{X}^T\mathbf{Y} + (\mathbf{X}^T\mathbf{X})\mathbf{w} + \mathbf{w}^T(\mathbf{X}^T\mathbf{X}) \\
&= -2\mathbf{X}^T\mathbf{Y} + 2(\mathbf{X}^T\mathbf{X})\mathbf{w} = 0
\end{aligned}
$$

Solving for the optimal weights, we get the following analytical solution:

$$
\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}
$$

In summary, for a given dataset with multiple features, we can represent the linear regression model as a product of weights and input features. The objective is to minimize the sum of squared errors (SSE) to find the optimal weights for the model. By taking the derivative of the SSE with respect to the weights and setting it to zero, we can determine the optimal weights for the optimization problem using the above equation.

- **Question 3**

- We need to deal with missing values for the column from Q1.
- We have two options: fill it with 0 or with the mean of this variable.
- Try both options. For each, train a linear regression model without regularization using the code from the lessons.
- For computing the mean, use the training only!
- Use the validation dataset to evaluate the models and compare the RMSE of each option.
- Round the RMSE scores to 2 decimal digits using `round(score, 2)`

Which option gives better RMSE?

- With 0
- With mean
- **Both are equally good**

In [ ]:
```
# filling missing values with zeros and mean
def handle_nan(df, feature, fillnan_with):
    df_copy = df.copy()
    if fillnan_with == 'mean':
        df_copy[feature].fillna(value = df_copy[feature].mean(), inplace=True)
    elif fillnan_with == 'zero':
        df_copy[feature].fillna(value = 0, inplace=True)

    return df_copy.values

# Root mean squared error
def rmse(y, y_pred):
    error = y_pred - y
    mse = (error ** 2).mean()
    rmse = np.sqrt(mse)
    return rmse
```

In [ ]:
```
def linear_regression(X, y):
    # adding ones in the dataset X
    X_0 = np.ones(X.shape[0])
    X = np.column_stack([X_0, X])

    XTX = X.T.dot(X)
    XTX_inverse = np.linalg.inv(XTX)
    w = XTX_inverse.dot(X.T).dot(y)

    Y_pred = X.dot(w)
    return Y_pred
```

In [ ]:
```
# filling missing values
X_train_zeros = handle_nan(X_train, 'total_bedrooms', 'zero')
X_val_zeros = handle_nan(X_val, 'total_bedrooms', 'zero')

X_train_mean = handle_nan(X_train, 'total_bedrooms', 'mean')
X_val_mean = handle_nan(X_val, 'total_bedrooms', 'mean')

# For training set
Y_pred = linear_regression(X_train_zeros, Y_train)
rmse_train_zeros = rmse(Y_train, Y_pred)
```

```
print('RMSE for train set with zeros: ',round(rmse_train_zeros, 2) )

Y_pred = linear_regression(X_train_mean, Y_train)
rmse_train_mean= rmse(Y_train, Y_pred)
print('RMSE for train set with mean: ',round(rmse_train_mean, 2) )

print('\n')
# For validation set
Y_pred = linear_regression(X_val_zeros, Y_val)
rmse_val_zeros = rmse(Y_val, Y_pred)
print('RMSE for validation set with zeros: ', round(rmse_train_mean, 2) )

Y_pred = linear_regression(X_val_mean, Y_val)
rmse_val_mean = rmse(Y_val, Y_pred)
print('RMSE for validation set with mean: ',round(rmse_train_mean, 2) )
```

```
RMSE for train set with zeros:  0.34
RMSE for train set with mean:  0.34


RMSE for validation set with zeros:  0.34
RMSE for validation set with mean:  0.34
```

4. Ridge Regression (L_2)

If two or more columns of a matrix are not orthogonal to each other (i.e., they are correlated), it implies that the matrix is not invertible (singular matrix). Consequently, the optimal weights equation:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

becomes problematic, as the inverse matrix $(\mathbf{X}^T\mathbf{X})$ is either hard to compute accurately or non-existent. This can result in large or unstable estimates of the regression coefficients, which can lead to poor model performance.

Regularization aims to ensure the existence of the inverse by forcing the matrix to be invertible, controlling the weights of the model so that they behave correctly and do not become too large.

Instead of simple minimizing the squared residual error $||\mathbf{Y} - \hat{\mathbf{Y}}||^2$, we add a regularization term involving the squared norm of the weights vector $||\mathbf{w}||^2$:

$$L(\mathbf{w}) = ||\mathbf{Y} - \hat{\mathbf{Y}}||^2 + \alpha||\mathbf{w}||^2$$

The goal is to minimize $L(\mathbf{w})$. To achieve this, we take the derivative of $L(\mathbf{w})$ with respect to the weights, set it equal to zero, and find a local minimum of the function. This results in the following optimal weights:

$$\frac{dL(\mathbf{w})}{d\mathbf{w}} = -2\mathbf{X}^T\mathbf{Y} + 2(\mathbf{X}^T\mathbf{X})\mathbf{w} + 2\alpha\mathbf{w} = 0$$

therefore, the optimal solution is

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

where $\mathbf{I}$ is the identity matrix. The matrix $(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})$ is always invertible for $\alpha > 0$.

The reason behind this is that $\mathbf{X}^T\mathbf{X}$ is always symmetric and positive semi-definite, meaning all its eigenvalues are non-negative. However, this does not guarantee invertibility, as eigenvalues can still be zero, resulting in a singular matrix.

When adding the two matrices, $(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})$, the resulting matrix's eigenvalues are the sums of the corresponding eigenvalues of the original matrices. Since the eigenvalues of $\mathbf{X}^T\mathbf{X}$ are non-negative and those of $\alpha\mathbf{I}$ are positive, the eigenvalues of $(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})$ are strictly positive. A symmetric matrix with strictly positive eigenvalues is positive definite, which are always invertible as none of their eigenvalues are equal to zero. Consequently, the matrix $(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})$ is always invertible for $\alpha > 0$.

In [ ]:
```
def ridge_regression(X, y, r = 0.0):
    # adding ones in the dataset X
    X_0 = np.ones(X.shape[0])
    X = np.column_stack([X_0, X])

    XTX = X.T.dot(X)
    # add regularization term rI
    I = np.eye(XTX.shape[0])
    XTX_inverse = np.linalg.inv(XTX + r*I)
    w = XTX_inverse.dot(X.T).dot(y)

    Y_pred = X.dot(w)
    return Y_pred, w
```

- **Question 4**

- Now let's train a regularized linear regression.

- For this question, fill the NAs with 0.
- Try different values of `r` from this list: `[0, 0.000001, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10]`.
- Use RMSE to evaluate the model on the validation dataset.
- Round the RMSE scores to 2 decimal digits.

Which `r` gives the best RMSE? If there are multiple options, select the smallest `r`.

- **0**
- 0.000001
- 0.001
- 0.0001

In [ ]:
```
# filling missing values with zeros
X_train = handle_nan(X_train, 'total_bedrooms', 'zero')
X_val = handle_nan(X_val, 'total_bedrooms', 'zero')


for r in [0, 0.000001, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10]:
    Y_pred, _ = ridge_regression(X_val, Y_val, r=r)
    rmse_val = round( rmse(Y_val, Y_pred), 5)
    print('%06s %0.5f' % (r, rmse_val))
```

```
     0 0.34000
  1e-06 0.34000
0.0001 0.34000
 0.001 0.34000
  0.01 0.34003
   0.1 0.34128
     1 0.34616
     5 0.34770
    10 0.34794
```

If multiple options yield similar results, it is common practice to select the smallest regularization term. In this case, $r = 0$, suggesting that regularization is not necessary.

- **Question 5**
- We used seed 42 for splitting the data. Let's find out how selecting the seed influences our score.
- Try different seed values: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$.
- For each seed, do the train/validation/test split with $60\%/20\%/20\%$ distribution.
- Fill the missing values with 0 and train a model without regularization.
- For each seed, evaluate the model on the validation dataset and collect the RMSE scores.
- What's the standard deviation of all the scores? To compute the standard deviation, use np.std.
- Round the result to 3 decimal digits (`round(std, 3)`)

What's the value of std?

- 0.5
- 0.05
- **0.005**
- 0.0005

In [ ]:
```
def split_data(df, target_column, train_size = 0.6,
               val_size = 0.2, seed = 42, log_transform = True):

    if train_size + val_size >= 1.0:
        raise ValueError("Value larger then 1")

    n = len(df)
    n_train = int(train_size*n)
    n_val = int(val_size*n)

    # Shuffle data
    np.random.seed(seed)
    idx = np.arange(n)
    np.random.shuffle(idx)
    df_shuffled = df.iloc[idx]

    # Split data
    X_train = df_shuffled.iloc[:n_train].copy()
    X_val = df_shuffled.iloc[n_train:n_train + n_val].copy()
    X_test = df_shuffled.iloc[n_train + n_val:].copy()
```

```
    if log_transform:
        Y_train = np.log1p(X_train[target_column]).values
        Y_val = np.log1p(X_val[target_column]).values
        Y_test = np.log1p(X_test[target_column]).values
    else:
        Y_train = X_train[target_column].values
        Y_val = X_val[target_column].values
        Y_test = X_test[target_column].values

    del X_train[target_column]
    del X_val[target_column]
    del X_test[target_column]

    # Fill missing values with zeros
    X_train = handle_nan(X_train, 'total_bedrooms', 'zero')
    X_val = handle_nan(X_val, 'total_bedrooms', 'zero')
    X_test = handle_nan(X_test, 'total_bedrooms', 'zero')

    X = {'train':X_train, 'val':X_val, 'test':X_test}
    Y = {'train': Y_train,'val':Y_val,'test': Y_test}

    return X,Y
```

In [ ]:
```
display(houses.head(2))

seeds = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

errors = []
for seed in seeds:
    X,Y = split_data(df = houses, target_column= 'median_house_value', seed = seed)
    Y_pred = linear_regression(X['val'], Y['val'])
    error = rmse(Y['val'], Y_pred)

    print('%10s' %seed, round( error, 3) )
    errors.append( error )

print('Std =', round(np.std(errors), 3))
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_v
701	-121.97	37.64	32.0	1283.0	194.0	485.0	171.0	6.0574	4310
830	-121.99	37.61	9.0	3666.0	711.0	2341.0	703.0	4.6458	2170

```
         0 0.337
         1 0.337
         2 0.337
         3 0.331
         4 0.337
         5 0.342
         6 0.334
         7 0.344
         8 0.35
         9 0.334
Std = 0.005
```

Standard deviation shows how different the values are. If it's low, then all values are approximately the same. If it's high, the values are different. If standard deviation of scores is low, then our model is stable.

- **Question 6**
- Split the dataset like previously, use seed 9.
- Combine train and validation datasets.
- Fill the missing values with 0 and train a model with $r = 0.001$.

What's the RMSE on the test dataset?

- 0.13
- 0.23
- **0.33**
- 0.43

In [ ]:
```
# Split data and fill missing values with zeros
X,Y = split_data(df = houses, target_column= 'median_house_value', seed = 9)

# Combine train and validation
X_train = np.concatenate([ X['train'], X['val']])
Y_train = np.concatenate([ Y['train'], Y['val']])

# Train model on train and validation and use in test set
```

```
_, w = ridge_regression(X_train, Y_train, r = 0.001)

# Regression with the trained weights
Y_pred = w[0] + X['test'].dot(w[1:])


print('RMSE on test set = ', round( rmse(Y['test'], Y_pred), 2))
```

```
RMSE on test set =  0.33
```