EXPERIMENT NO 3

MULTIPLE LINEAR REGRESSION BY SCRATCH

Importing Modules

In [1]:
```
# importing the modules
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams

# parameters to make the visual look better
rcParams['figure.figsize'] = (14, 7)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
```

Creating The Model Class

In [2]:
```
class LinearRegression:
    '''
    A class which implements linear regression model with gradient descent.
    '''
    def __init__(self, learning_rate=0.01, n_iterations=10000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights, self.bias = None, None
        self.loss = []

    '''
    function to calculate mean square error
    '''

    def _mean_squared_error(self, y, y_hat):
        '''
        Private method, used to evaluate loss at each iteration.

        :param: y - array, true values
        :param: y_hat - array, predicted values
        :return: float
        '''
        error = 0
        for i in range(len(y)):
            error += (y[i] - y_hat[i])**2
        return error / len(y)

    '''
    function to fit the model
    '''

    def fit(self, X, y):
        '''
        Used to calculate the coefficient of the linear regression model.

        :param X: array, features
        :param y: array, true values
        :return: None
        '''
        # 1. Initialize weights and bias to zeros
        self.weights = np.zeros(X.shape[1])
        self.bias = 0

        # 2. Perform gradient descent
        for i in range(self.n_iterations):
            # Line equation
            y_hat = np.dot(X, self.weights) + self.bias
            loss = self._mean_squared_error(y, y_hat)
            self.loss.append(loss)

            # Calculate derivatives
            partial_w = (1 / X.shape[0]) * (2 * np.dot(X.T, (y_hat - y)))
            partial_d = (1 / X.shape[0]) * (2 * np.sum(y_hat - y))

            # Update the coefficients
            self.weights -= self.learning_rate * partial_w
            self.bias -= self.learning_rate * partial_d

    '''
    function to find prediction of the model
    '''

    def predict(self, X):
        '''
        Makes predictions using the line equation.

        :param X: array, features
        :return: array, predictions
        '''
        return np.dot(X, self.weights) + self.bias
```

Reading the dataset

In [3]:
```
from sklearn.datasets import load_diabetes

data = load_diabetes()
X = data.data
y = data.target
```

Splitting Data Into Train & Test Dataset

In [4]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=42)
```

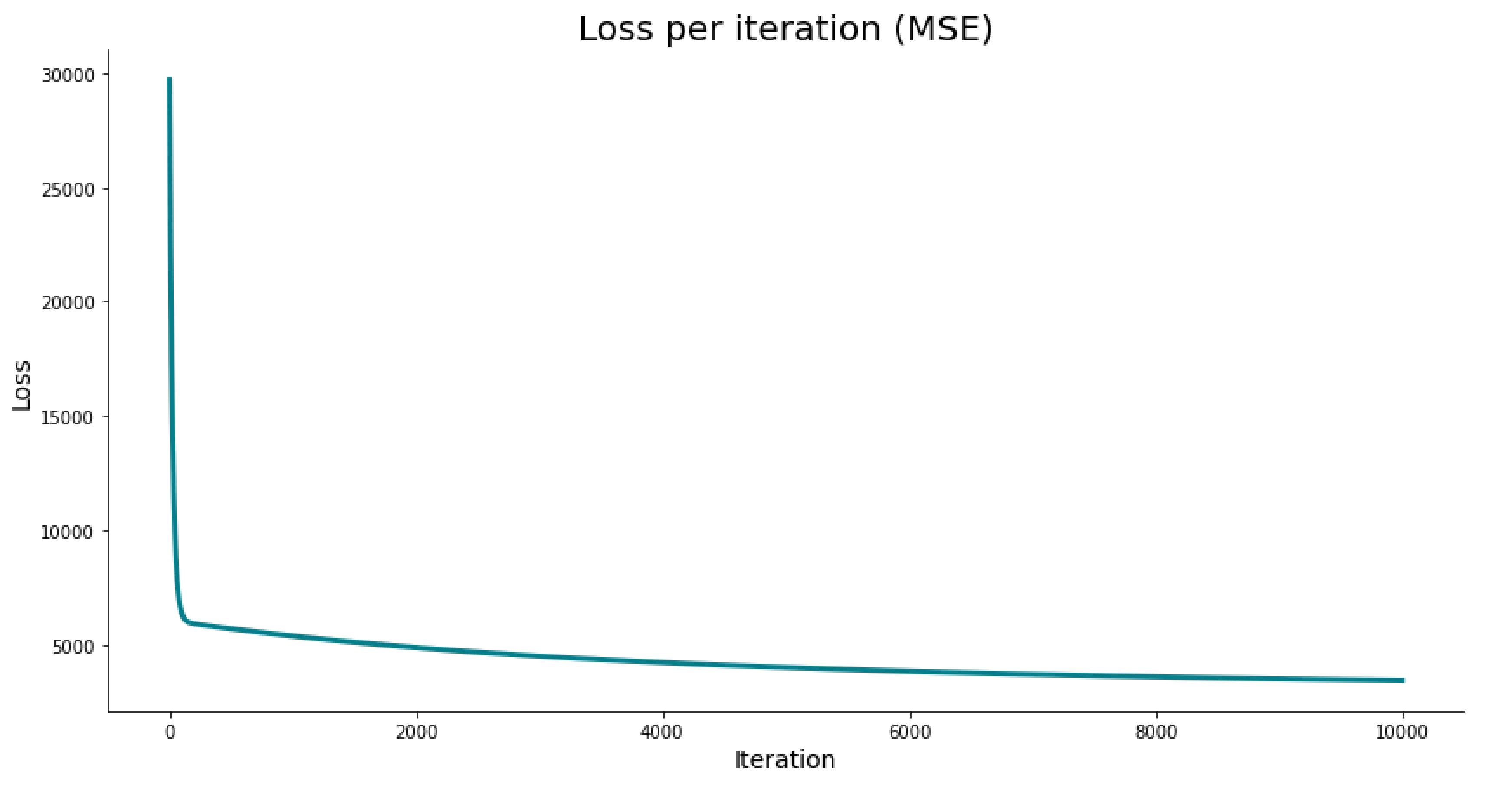
Training the Model

In [5]:
```
model = LinearRegression()
model.fit(X_train, y_train)
preds = model.predict(X_test)

# first few predicted values

preds[:10]
```

Out[5]:
```
array([164.21205411, 159.47509229, 161.03082858, 251.53128339,
       149.27798242, 131.37880168, 219.01979007, 197.83564132,
       114.99909736, 137.90827135])
```

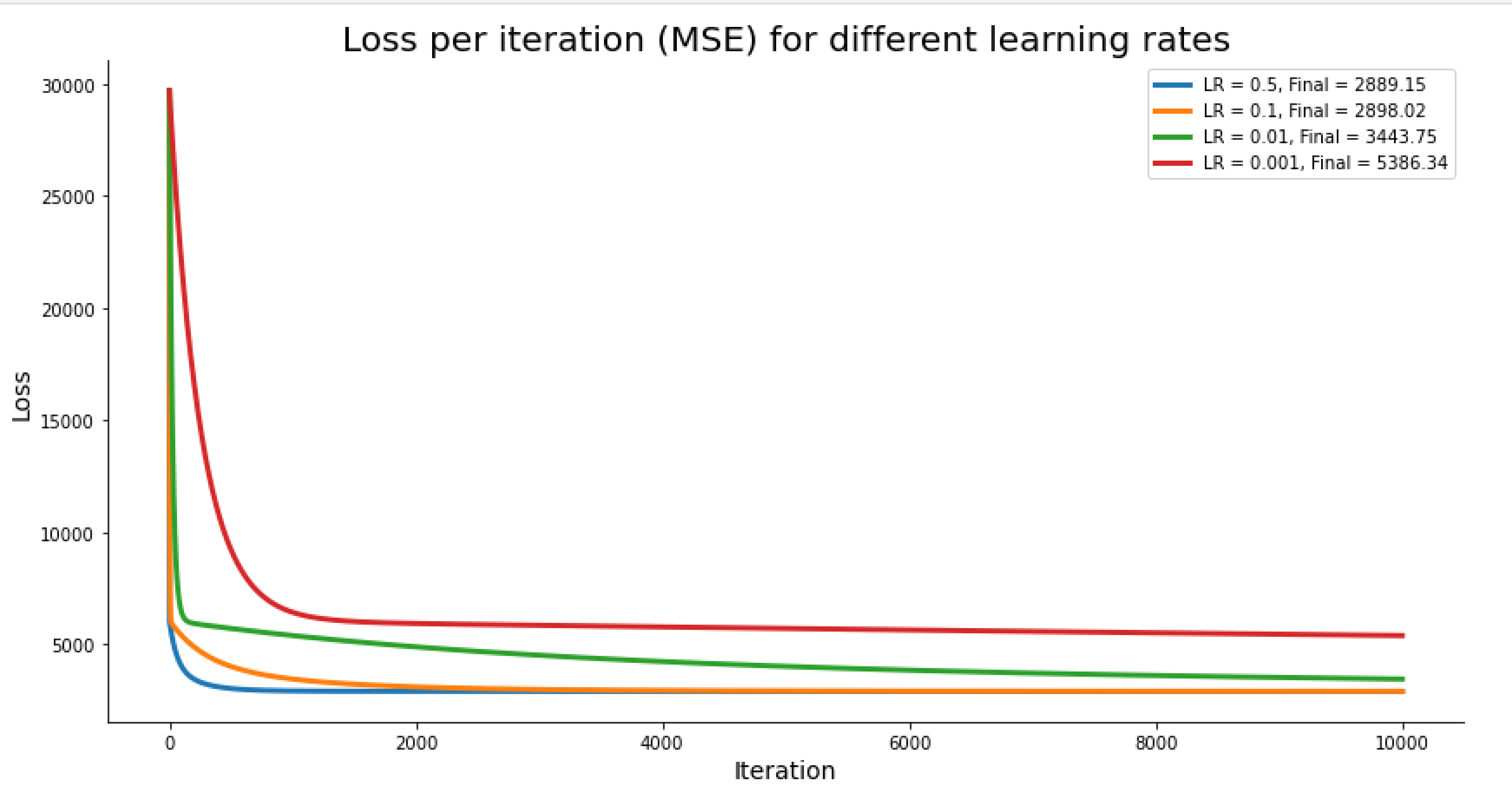
Visualizing of the Loss

In [6]:
```
xs = np.arange(len(model.loss))
ys = model.loss

plt.plot(xs, ys, lw=3, c='#087E8B')
plt.title('Loss per iteration (MSE)', size=20)
plt.xlabel('Iteration', size=14)
plt.ylabel('Loss', size=14)
plt.show()
```



Visualizing the Loss at Diffrent Learning Rates

In [7]:
```
losses = {}
for lr in [0.5, 0.1, 0.01, 0.001]:
    model = LinearRegression(learning_rate=lr)
    model.fit(X_train, y_train)
    losses[f'LR={str(lr)}'] = model.loss

xs = np.arange(len(model.loss))

plt.plot(xs,
         losses['LR=0.5'],
         lw=3,
         label=f"LR = 0.5, Final = {losses['LR=0.5'][-1]:.2f}")
plt.plot(xs,
         losses['LR=0.1'],
         lw=3,
         label=f"LR = 0.1, Final = {losses['LR=0.1'][-1]:.2f}")
plt.plot(xs,
         losses['LR=0.01'],
         lw=3,
         label=f"LR = 0.01, Final = {losses['LR=0.01'][-1]:.2f}")
plt.plot(xs,
         losses['LR=0.001'],
         lw=3,
         label=f"LR = 0.001, Final = {losses['LR=0.001'][-1]:.2f}")
plt.title('Loss per iteration (MSE) for different learning rates', size=20)
plt.xlabel('Iteration', size=14)
plt.ylabel('Loss', size=14)
plt.legend()
plt.show()
```



Scoring the Model

In [8]:
```
# the learning rate of 0.5 is good so we will select it and train the model again and find the Mean Squared Er
model = LinearRegression(learning_rate=0.5)
model.fit(X_train, y_train)
preds = model.predict(X_test)

model._mean_squared_error(y_test, preds)
```

Out[8]:
```
2885.8431887424276
```

-------X-------------X-------------X-------------X-------------X-------------X--
-----------X-------------X-------