Univariable Linear Regression

Load the Data and Libraries:

In [1]:

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline
```

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
plt.rcParams['figure.figsize']=(12,8)
```

In [3]:

```
data=pd.read_csv("bike_sharing_data.txt")
data.head()
```

Out[3]:

	Population	Profit
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233

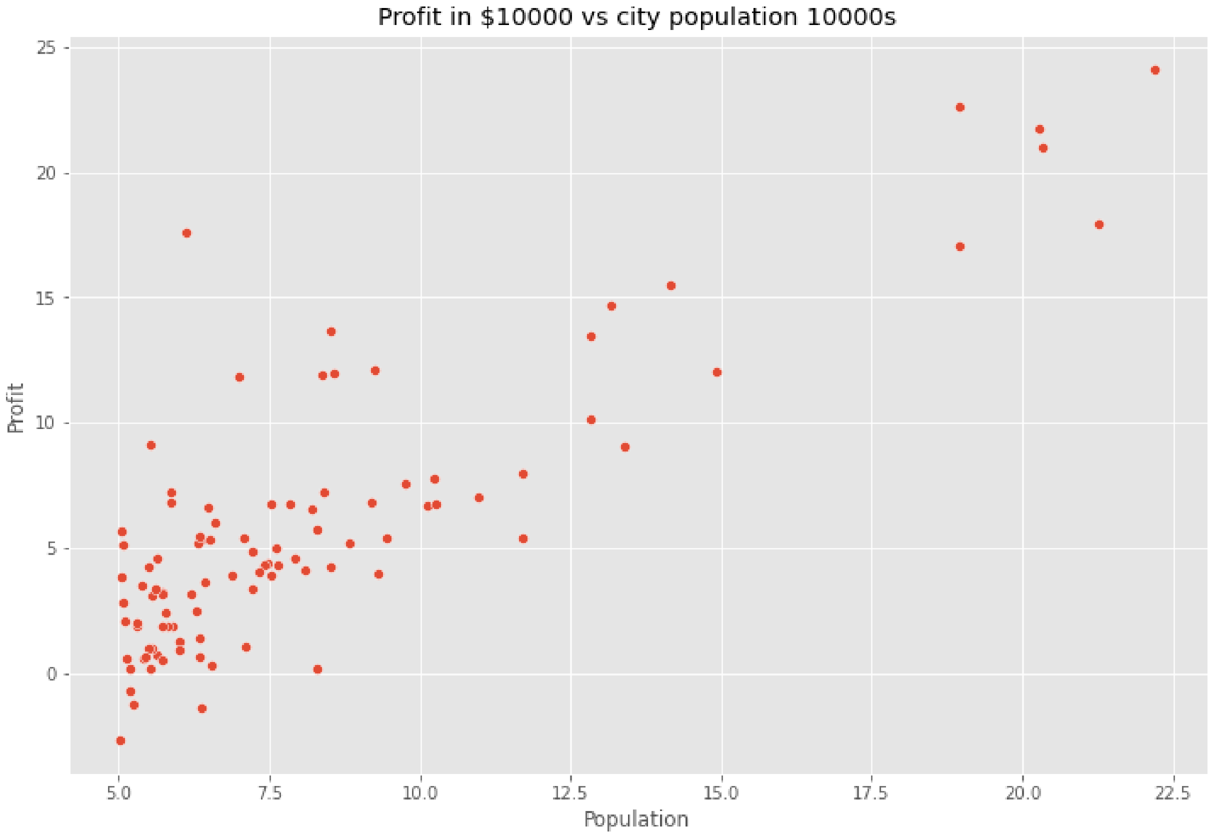
Vizulaize the data:

In [4]:

```
ax=sns.scatterplot(x="Population", y="Profit",data=data)
ax.set_title("Profit in $10000 vs city population 10000s")
```

Out[4]:

```
Text(0.5, 1.0, 'Profit in $10000 vs city population 10000s')
```



In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97 entries, 0 to 96
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Population  97 non-null     float64
 1   Profit      97 non-null     float64
dtypes: float64(2)
memory usage: 1.6 KB
```

Compute the Cost $J(\theta)$

The objective of linear regression is to minimize the cost function

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

where $h_\theta(x)$ is the hypothesis and given by the linear model

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

In [6]:

```
def cost_function(x,y,theta):
    m=len(y)
    y_pred=x.dot(theta)
    error=(y_pred-y)**2
    return 1/(2*m)*np.sum(error)
m=data.Population.values.size
x=np.append(np.ones((m,1)),data.Population.values.reshape(m,1),axis=1)
y=data.Profit.values.reshape(m,1)
```

```
theta=np.zeros((2,1))
cost_function(x,y,theta)
```

Out[6]:

```
32.072733877455676
```

Gradient Descent

Minimize the cost function $J(\theta)$ by updating the below equation and repeat unitil convergence

$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ (simultaneously update θ_j for all j).

In [7]:

```
def gradient_descent(x, y, theta, alpha, iterations):
    m = len(y)
    costs = []
    for i in range(iterations):
        y_pred = x.dot(theta)
        error = np.dot(x.transpose(), (y_pred - y))
        theta -= alpha * 1/m * error
        costs.append(cost_function(x, y, theta))
    return theta, costs
```

In [8]:

```
theta, costs = gradient_descent(x, y, theta, alpha=0.01, iterations=2000)

print("h(x) = {} + {}x1".format(str(round(theta[0, 0], 2)),
                                str(round(theta[1, 0], 2))))
```

```
h(x) = -3.79 + 1.18x1
```

Visualising the Cost Function $J(\theta)$

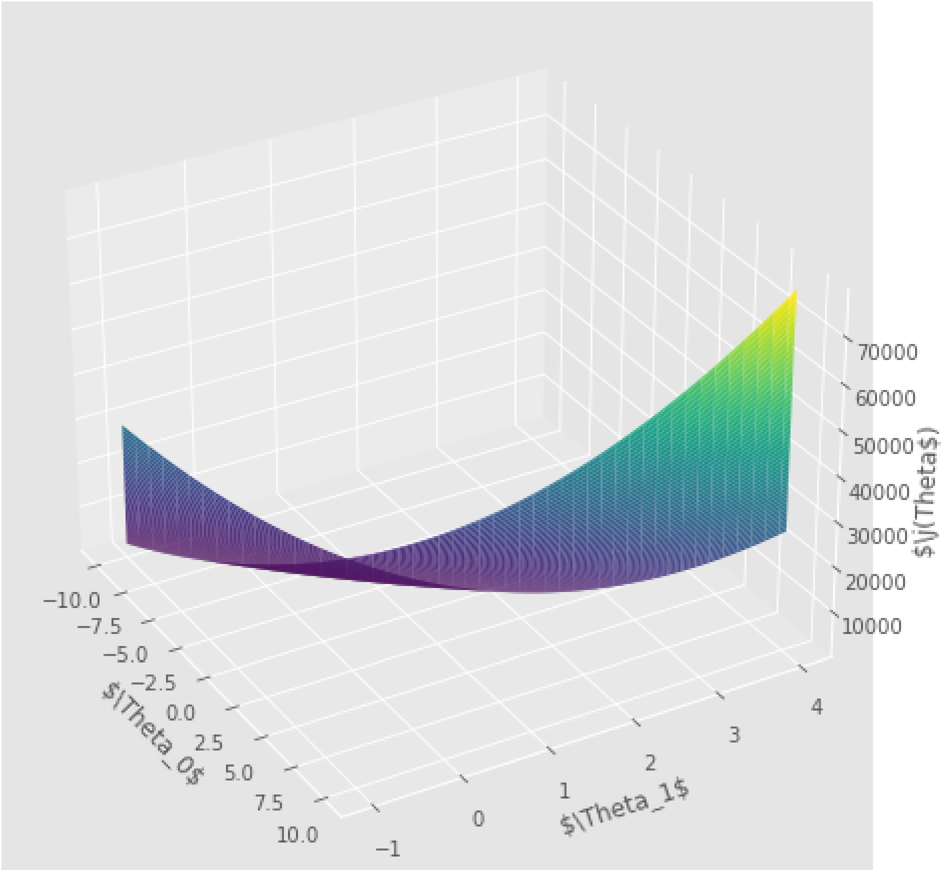
In [9]:

```
from mpl_toolkits.mplot3d import Axes3D
theta_0=np.linspace(-10,10,100)
theta_1=np.linspace(-1,4,100)
cost_value=np.zeros((len(theta_0),len(theta_1)))
for i in range(len(theta_0)):
    for j in range(len(theta_1)):
        t=np.array([theta_0[i],theta_1[j]])
        cost_value[i,j]=cost_function(x,y,t)
```

In [16]:

```
fig=plt.figure(figsize=(12,8))
ax=fig.gca(projection='3d')
surf=ax.plot_surface(theta_0,theta_1,cost_value,cmap='viridis')
plt.xlabel('$\Theta_0\$')
plt.ylabel('$\Theta_1\$')
ax.set_zlabel('$\j(Theta\$)')
ax.view_init(30,330)
plt.show()
```

```
C:\Users\KIIT\AppData\Local\Temp/ipykernel_44012/3237581826.py:2: MatplotlibDeprecat
ionWarning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. S
tarting two minor releases later, gca() will take no keyword arguments. The gca() fu
nction should only be used to get the current axes, or if no axes exist, create new
axes with default keyword arguments. To create a new axes with non-default argument
s, use plt.axes() or plt.subplot().
  ax=fig.gca(projection='3d')
```



Task 7: Plotting the Convergence

Plot $J(\theta)$ against the number of iterations of gradient descent:

In [18]:
```
plt.plot(costs)
plt.xlabel('Iterations')
plt.ylabel('$j(\Theta)$')
plt.title('Value of the cost function over iteration of the gradient descent')
```

Out[18]:
```
Text(0.5, 1.0, 'Value of the cost function over iteration of the gradient descent')
```

Training Data with Linear Regression Fit

In [24]:

```
theta=np.squeeze(theta)
ax=sns.scatterplot(x="Population", y="Profit",data=data)
x_value= [x for x in range(5,25)]
y_value=[(x*theta[1]+ theta[0]) for x in x_value]
sns.lineplot(x_value,y_value)
plt.xlabel("Population in 10000s")
plt.ylabel("Profit in $10000s")
plt.title("Linear Regression Fit")
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only vali
d positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

Inference using the optimized θ values

$h_\theta(x) = \theta^T x$

In [28]:
```
def predict(x,theta):
    y_pred=np.dot(theta.transpose(),x)
    return y_pred
```

In [36]:
```
y_pred_1=predict(np.array([1,5.5]),theta)*10000
print("For a Population of 55000 people,the model predicts the profit of $"+str(roun
```

```
For a Population of 55000 people,the model predicts the profit of $27141.0
```

In [32]:
```
y_pred_2=predict(np.array([1, 9.3]),theta)*10000
print("For a Population of 93000 people,the model predict a profit of $"+str(round(y
```

```
For a Population of 73000 people,the model predict a profit of $48421.0
```

In [ ]:
```

```