We want to predict both the house price and house rental price based on size (in square feet) and number of bedrooms.

In [96]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

In [97]:
```
dic= {"size(sq ft) x1":[1500,2000,2500,1800],
      "Bedrooms x2" :[3,4,4,3],
      "Price($)": [300000,400000,450000,350000],
      "Rent($)": [1500,1800,2200,1600] }

data=pd.DataFrame(dic)
data
```

Out[97]:

	size(sq ft) x1	Bedrooms x2	Price($)	Rent($)
0	1500	3	300000	1500
1	2000	4	400000	1800
2	2500	4	450000	2200
3	1800	3	350000	1600

In [98]:
```python

X = data[["size(sq ft) x1", "Bedrooms x2"]].values
Y = data[["Price($)", "Rent($)"]].values
print(X)

print("///////////////////")

print(Y)
print('????????????????????????????')
Y1= data["Price($)"]
print(Y1)
Y2=data["Rent($)"]
print(Y2)
```

```
[[1500    3]
 [2000    4]
 [2500    4]
 [1800    3]]
///////////////////
[[300000   1500]
 [400000   1800]
 [450000   2200]
 [350000   1600]]
????????????????????????????
0    300000
1    400000
2    450000
3    350000
Name: Price($), dtype: int64
0    1500
1    1800
2    2200
3    1600
Name: Rent($), dtype: int64
```

In [99]:

```python

ones_col = np.ones((4, 1))

X1 = np.hstack((ones_col, X))

print("Column of Ones:")
print(ones_col)

np.set_printoptions(precision=2)
print("Matrix for size and bedroom:")
print(X1)

```

```
Column of Ones:
[[1.]
 [1.]
 [1.]
 [1.]]
Matrix for size and bedroom:
[[1.0e+00 1.5e+03 3.0e+00]
 [1.0e+00 2.0e+03 4.0e+00]
 [1.0e+00 2.5e+03 4.0e+00]
 [1.0e+00 1.8e+03 3.0e+00]]
```

In [100]:

```python
X1_T= X1.T
print(X1_T)
```

```
[[1.0e+00 1.0e+00 1.0e+00 1.0e+00]
 [1.5e+03 2.0e+03 2.5e+03 1.8e+03]
 [3.0e+00 4.0e+00 4.0e+00 3.0e+00]]
```

In [17]:
```
#  this is basically X1^T*X1 store in all
all= X1_T.dot(X1)
print(all)

inverse_all= np.linalg.inv(all)

print("The final result is (X1^T*X1)^-1")
print(inverse_all)

mul_X1trans_Y1= X1_T@Y1
print(mul_X1trans_Y1)

result_price= inverse_all@mul_X1trans_Y1
print("Final coefficent")
print(result_price)
```

```
[[4.00e+00 7.80e+03 1.40e+01]
 [7.80e+03 1.57e+07 2.79e+04]
 [1.40e+01 2.79e+04 5.00e+01]]
The final result is (X1^T*X1)^-1
[[ 1.26e+01  8.82e-04 -4.03e+00]
 [ 8.82e-04  5.88e-06 -3.53e-03]
 [-4.03e+00 -3.53e-03  3.12e+00]]
[1.50e+06 3.00e+09 5.35e+06]
Final coefficent
[42647.06   117.65 29411.76]
```

In [18]:
```
# This is coefficent for price
print("This is coefficents for price")
b01,b11,b21= result_price
print("b01:",b01)
print("b11:",b11)
print("b21:",b21)
```

```
This is coefficents for price
b01: 42647.058823533356
b11: 117.64705882353155
b21: 29411.76470588334
```

In [19]:
```
mul_X1trans_Y2= X1_T@Y2
print(mul_X1trans_Y2)
```

```
[7.10e+03 1.42e+07 2.53e+04]
```

In [20]:
```
result_rent=inverse_all@mul_X1trans_Y2
print('This is the coefficent for result_rent')
print(result_rent)
```

```
This is the coefficent for result_rent
[301.47   0.68  44.12]
```

In [21]:

```
# This is coefficent for rent
print("This is coefficents for price")
b02,b12,b22= result_rent
print("b02:",b02)
print("b12:",b12)
print("b22:",b22)
```

```
This is coefficents for price
b02: 301.47058823530097
b12: 0.6764705882353041
b22: 44.11764705882524
```

In [22]:

```
# now just checking checking the future price and rent using formula (y=
#where x1 is  size(sq ft) and x2 is bedroom

# can you do this?
#it is because coefficent and intercept are find for both price and rent

# lets suppose  size(sq ft)=3000 and bedrooms= 6
size_sq_ft=3000
bedrooms= 6
```

In [24]:

```
y= b02+b12*size_sq_ft+b22*bedrooms
print("The rent of the house (size=3000) and bedrooms(6) is:")
print(y)
```

```
The rent of the house (size=3000) and bedrooms(6) is:
2595.588235294165
```

In [25]:

```
y= b01+b11*size_sq_ft+b21*bedrooms
print("The price of the house (size=3000) and bedrooms(6) is:")
print(y)
```

```
The price of the house (size=3000) and bedrooms(6) is:
572058.8235294281
```

In [26]:

```

# this is checking how correct work, getting values from already existing
size_sq_ft=2500
bedrooms=4
y= b02+b12*size_sq_ft+b22*bedrooms
print("The rent of the house (size=3000) and bedrooms(6) is:")
print(y)
```

```
The rent of the house (size=3000) and bedrooms(6) is:
2169.1176470588625
```

In [78]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math

dic= {"size(sq ft) x1":[1500,2000,2500,1800],
      "Bedrooms x2" :[3,4,4,3],
      "Price($)": [300000,400000,450000,350000],
      "Rent($)": [1500,1800,2200,1600] }

data=pd.DataFrame(dic)
data

X = data[["size(sq ft) x1", "Bedrooms x2"]].values
Y = data[["Price($)", "Rent($)"]].values
print(X)

print("///////////////////")

print(Y)
print('????????????????????????????')
Y1= data["Price($)"]
print(Y1)
Y2=data["Rent($)"]
print(Y2)

ones_col = np.ones((4, 1))

X1 = np.hstack((ones_col, X))

print("Column of Ones:")
print(ones_col)

np.set_printoptions(precision=2)
print("Matrix for size and bedroom:")
print(X1)

X1_T= X1.T
print(X1_T)

#  this is basically X1^T*X1 store in all
all= X1_T.dot(X1)
print(all)

inverse_all= np.linalg.inv(all)

print("The final result is (X1^T*X1)^-1")
print(inverse_all)

mul_X1trans_Y1= X1_T@Y1
print(mul_X1trans_Y1)



result_price= inverse_all@mul_X1trans_Y1
print("Final coefficent")
```

```
print(result_price)


mul_X1trans_Y2= X1_T@Y2
print(mul_X1trans_Y2)

result_rent=inverse_all@mul_X1trans_Y2
print('This is the coefficent for result_rent')
print(result_rent)

# This is coefficent for price
print("This is coefficents for price")
b01,b11,b21= result_price
print("b01:",b01)
print("b11:",b11)
print("b21:",b21)

mul_X1trans_Y2= X1_T@Y2
print(mul_X1trans_Y2)

# now just checking checking the future price and rent using formula (y=
#where x1 is  size(sq ft) and x2 is bedroom

# can you do this?
#it is because coefficent and intercept are find for both price and rent

# lets suppose  size(sq ft)=3000 and bedrooms= 6
size_sq_ft=3000
bedrooms= 6

y= b02+b12*size_sq_ft+b22*bedrooms
print("The rent of the house (size=3000) and bedrooms(6) is:")
print(y)


y= b01+b11*size_sq_ft+b21*bedrooms
print("The price of the house (size=3000) and bedrooms(6) is:")
print(y)

# this is checking how correct work, getting values from already existing
size_sq_ft=2500
bedrooms=4
y= b02+b12*size_sq_ft+b22*bedrooms
print("The rent of the house (size=2500) and bedrooms(4) is:")
print(y)
```

```
[[1500    3]
 [2000    4]
 [2500    4]
 [1800    3]]
//////////////////
[[300000   1500]
 [400000   1800]
 [450000   2200]
 [350000   1600]]
????????????????????????????
0    300000
1    400000
2    450000
3    350000
Name: Price($), dtype: int64
0    1500
1    1800
2    2200
3    1600
Name: Rent($), dtype: int64
Column of Ones:
[[1.]
 [1.]
 [1.]
 [1.]]
Matrix for size and bedroom:
[[1.0e+00 1.5e+03 3.0e+00]
 [1.0e+00 2.0e+03 4.0e+00]
 [1.0e+00 2.5e+03 4.0e+00]
 [1.0e+00 1.8e+03 3.0e+00]]
[[1.0e+00 1.0e+00 1.0e+00 1.0e+00]
 [1.5e+03 2.0e+03 2.5e+03 1.8e+03]
 [3.0e+00 4.0e+00 4.0e+00 3.0e+00]]
[[4.00e+00 7.80e+03 1.40e+01]
 [7.80e+03 1.57e+07 2.79e+04]
 [1.40e+01 2.79e+04 5.00e+01]]
The final result is (X1^T*X1)^-1
[[ 1.26e+01  8.82e-04 -4.03e+00]
 [ 8.82e-04  5.88e-06 -3.53e-03]
 [-4.03e+00 -3.53e-03  3.12e+00]]
[1.50e+06 3.00e+09 5.35e+06]
Final coefficent
[42647.06   117.65 29411.76]
[7.10e+03 1.42e+07 2.53e+04]
This is the coefficent for result_rent
[301.47   0.68  44.12]
This is coefficents for price
b01: 42647.058823533356
b11: 117.64705882353155
b21: 29411.76470588334
[7.10e+03 1.42e+07 2.53e+04]
The rent of the house (size=3000) and bedrooms(6) is:
2595.588235294165
The price of the house (size=3000) and bedrooms(6) is:
572058.8235294281
```

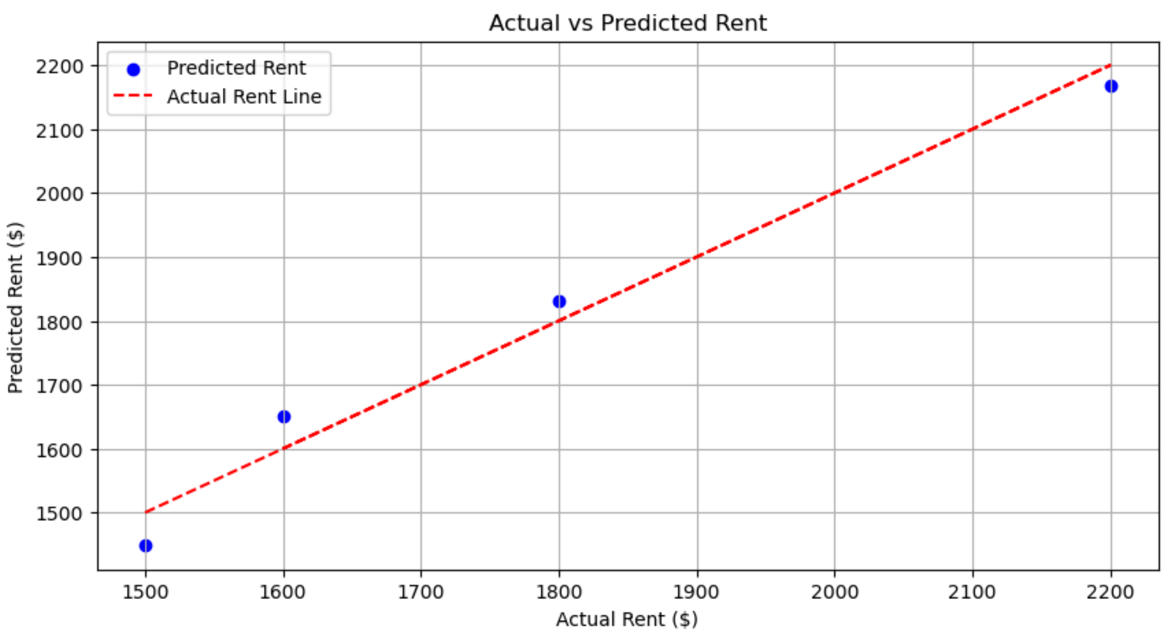
```
The rent of the house (size=2500) and bedrooms(4) is:
2169.1176470588625
```

In [180]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


dic = {
    "size(sq ft) x1": [1500, 2000, 2500, 1800],
    "Bedrooms x2": [3, 4, 4, 3],
    "Price($)": [300000, 400000, 450000, 350000],
    "Rent($)": [1500, 1800, 2200, 1600]
}


data = pd.DataFrame(dic)

# Separating Independent and Dependent Variables
X = data[["size(sq ft) x1", "Bedrooms x2"]].values
Y1 = data["Price($)"].values
Y2 = data["Rent($)"].values

# Adding Ones Column to X
ones_col = np.ones((X.shape[0], 1))
X1 = np.hstack((ones_col, X))

#(X^T * X)^-1
X1_T = X1.T
all_matrix = X1_T.dot(X1)
inverse_all = np.linalg.inv(all_matrix)

# Coefficients for Price Prediction
mul_X1trans_Y1 = X1_T @ Y1
result_price = inverse_all @ mul_X1trans_Y1
b01, b11, b21 = result_price

# Coefficients for Rent Prediction
mul_X1trans_Y2 = X1_T @ Y2
result_rent = inverse_all @ mul_X1trans_Y2
b02, b12, b22 = result_rent

# Function to Predict Price and Rent
def predict(size, bedrooms):
    price = b01 + b11 * size + b21 * bedrooms
    rent = b02 + b12 * size + b22 * bedrooms
    return price, rent

# Predicting for size=3000 and bedrooms=6
size_sq_ft = 3000
bedrooms = 6
price_pred, rent_pred = predict(size_sq_ft, bedrooms)

print(f"The predicted price of the house (size=3000, bedrooms=6) is: ${pr
print(f"The predicted rent of the house (size=3000, bedrooms=6) is: ${ren

# Plotting Predicted vs Actual for Rent
plt.figure(figsize=(10, 5))
predicted_rent = [predict(row[0], row[1])[1] for row in X]
plt.scatter(Y2, predicted_rent, color='blue', label='Predicted Rent')
```

```python
plt.plot(Y2, Y2, color='red', linestyle='--', label='Actual Rent Line')
plt.xlabel("Actual Rent ($)")
plt.ylabel("Predicted Rent ($)")
plt.title("Actual vs Predicted Rent")
plt.legend()
plt.grid(True)
plt.show()

# Plotting Predicted vs Actual for Price
plt.figure(figsize=(10, 5))
predicted_price = [predict(row[0], row[1])[0] for row in X]
plt.scatter(Y1, predicted_price, color='green', label='Predicted Price')
plt.plot(Y1, Y1, color='red', linestyle='--', label='Actual Price Line')
plt.xlabel("Actual Price ($)")
plt.ylabel("Predicted Price ($)")
plt.title("Actual vs Predicted Price")
plt.legend()
plt.grid(True)
plt.show()

```

```
The predicted price of the house (size=3000, bedrooms=6) is: $572058.82
The predicted rent of the house (size=3000, bedrooms=6) is: $2595.59
```



checking the students performance index based on hours studied and Previous Scores using maltivariate Linear Regression Algorithm

In [102]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math
import sklearn

```

In [103]:
```
data=pd.read_csv('Student_Performance.csv')
data
```

Out[103]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0
2	8	51	Yes	7	2	45.0
3	5	52	Yes	5	2	36.0
4	7	75	No	8	5	66.0
...	...	...	...	...	...	...
9995	1	49	Yes	4	2	23.0
9996	7	64	Yes	8	5	58.0
9997	6	83	Yes	8	5	74.0
9998	9	97	Yes	7	0	95.0
9999	7	74	No	8	1	64.0

10000 rows × 6 columns

In [104]:
```
data.shape
```

Out[104]:
```
(10000, 6)
```

In [105]:
```
data.count()
```

Out[105]:
```
Hours Studied                       10000
Previous Scores                     10000
Extracurricular Activities          10000
Sleep Hours                         10000
Sample Question Papers Practiced    10000
Performance Index                   10000
dtype: int64
```

In [106]:
```
data.isnull().sum()
```

Out[106]:
```
Hours Studied                       0
Previous Scores                     0
Extracurricular Activities          0
Sleep Hours                         0
Sample Question Papers Practiced    0
Performance Index                   0
dtype: int64
```

In [107]:
```
data.duplicated().sum()
```

Out[107]: 127

In [108]:
```
dataset=data.drop_duplicates()
dataset
```

Out[108]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0
2	8	51	Yes	7	2	45.0
3	5	52	Yes	5	2	36.0
4	7	75	No	8	5	66.0
...	...	...	...	...	...	...
9995	1	49	Yes	4	2	23.0
9996	7	64	Yes	8	5	58.0
9997	6	83	Yes	8	5	74.0
9998	9	97	Yes	7	0	95.0
9999	7	74	No	8	1	64.0

9873 rows × 6 columns

In [109]:
```
dataset.shape
```

Out[109]: (9873, 6)

In [110]:
```
dataset.describe()
```

Out[110]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index
count	9873.000000	9873.000000	9873.000000	9873.000000	9873.000000
mean	4.992100	69.441102	6.531652	4.583004	55.216651
std	2.589081	17.325601	1.697683	2.867202	19.208570
min	1.000000	40.000000	4.000000	0.000000	10.000000
25%	3.000000	54.000000	5.000000	2.000000	40.000000
50%	5.000000	69.000000	7.000000	5.000000	55.000000
75%	7.000000	85.000000	8.000000	7.000000	70.000000
max	9.000000	99.000000	9.000000	9.000000	100.000000

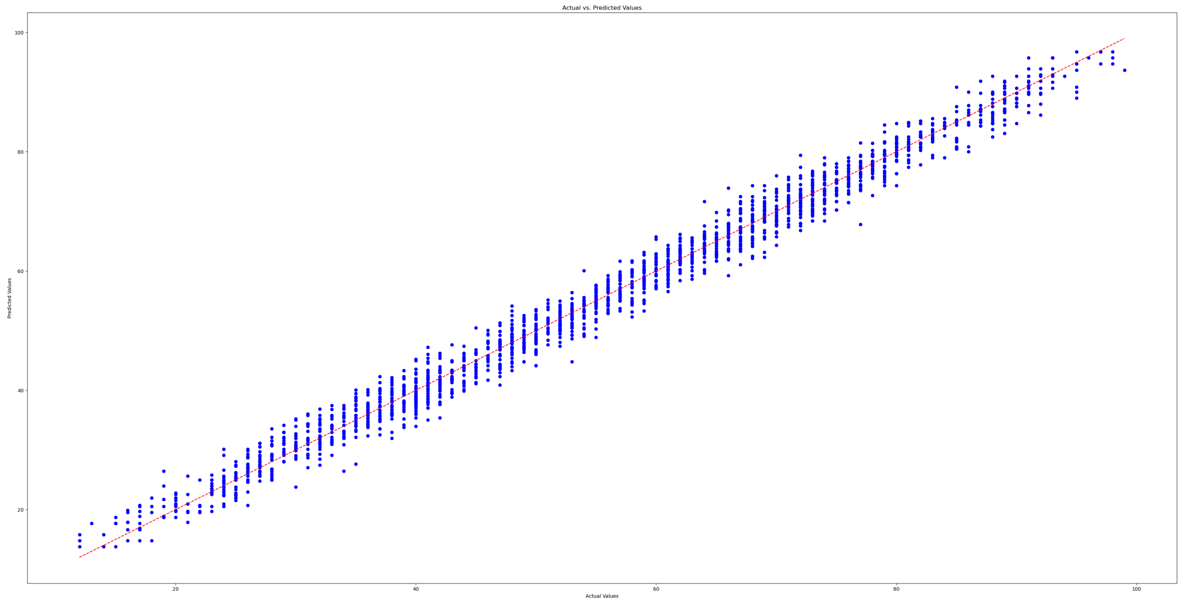
In [111]:
```
dataset.nunique()
```

Out[111]:
```
Hours Studied                        9
Previous Scores                     60
Extracurricular Activities           2
Sleep Hours                          6
Sample Question Papers Practiced    10
Performance Index                   91
dtype: int64
```

In [112]:

```
X = data[['Hours Studied', 'Previous Scores']].values
y = data['Performance Index'].values


X = np.hstack((np.ones((X.shape[0], 1)), X))


split_ratio = 0.8
split_index = int(split_ratio * len(X))
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]


# Normal Equation: beta = (X^T * X)^-1 * X^T * y
beta = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train


y_pred = X_test @ beta


mse = np.mean((y_test - y_pred) ** 2)

ss_total = np.sum((y_test - np.mean(y_test)) ** 2)
ss_residual = np.sum((y_test - y_pred) ** 2)
r2 = 1 - (ss_residual / ss_total)

# Printing evaluation metrics
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Step 8: Visualizing the results
plt.figure(figsize=(42,21))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='r
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values')
plt.show()

# Printing the coefficients
print(f'Intercept: {beta[0]}')
print(f'Coefficients: {beta[1:]}')

```

```
Mean Squared Error: 5.383627302493887
R-squared: 0.9852579816509904
```




```
Intercept: -29.82632662203914
Coefficients: [2.86 1.02]
```

In [ ]:

```
 
```