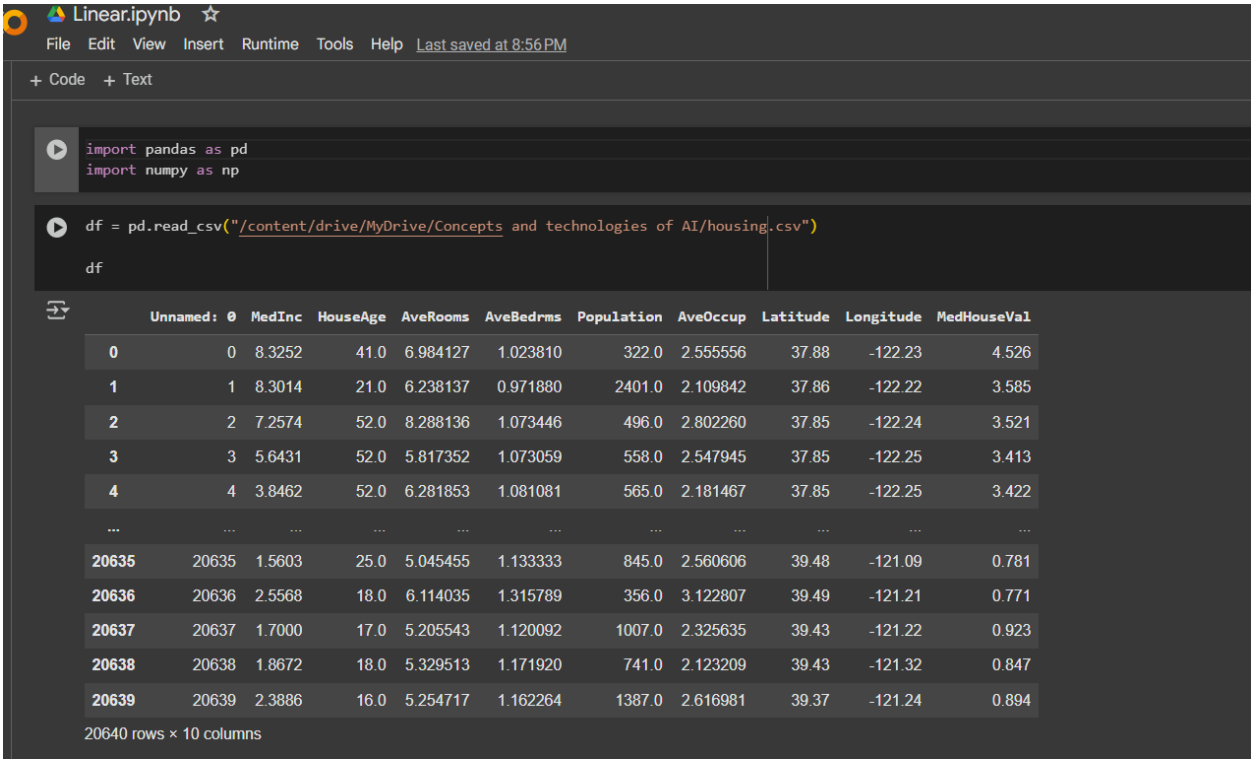
Linear regression:-



	Unnamed: 0	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
...	...	...	...	...	...	...	...	...	...	...
20635	20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09	0.781
20636	20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21	0.771
20637	20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22	0.923
20638	20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32	0.847
20639	20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24	0.894

20640 rows × 10 columns

```
[ ] HouseAge = df['HouseAge'].to_numpy()
    HouseAge
```

```
array([41., 21., 52., ..., 17., 18., 16.])
```

```
[ ] AveRooms = df['AveRooms'].to_numpy()
    AveRooms
```

```
array([6.98412698, 6.23813708, 8.28813559, ..., 5.20554273, 5.32951289,
       5.25471698])
```

```
[ ] AveBedrms = df['AveBedrms'].to_numpy()
    AveBedrms
```

```
array([1.02380952, 0.97188049, 1.07344633, ..., 1.12009238, 1.17191977,
       1.16226415])
```

```
[ ] x0 = np.ones(len(HouseAge))

    x0
```

```
array([1., 1., 1., ..., 1., 1., 1.])
```

```
[ ] X2 = np.array([x0, HouseAge, AveRooms, AveBedrms]).T

    W = np.array([0, 0, 0, 0])

    Y2 = np.array(AveBedrms)
```

```
[ ] Y2
```

```
array([1.02380952, 0.97188049, 1.07344633, ..., 1.12009238, 1.17191977,
       1.16226415])
```

```
[ ] X2T = np.array([x0, HouseAge, AveRooms, AveBedrms]).T

    X2T
```

```
array([[ 1.        , 41.        ,  6.98412698,  1.02380952],
       [ 1.        , 21.        ,  6.23813708,  0.97188049],
       [ 1.        , 52.        ,  8.28813559,  1.07344633],
       ...,
       [ 1.        , 17.        ,  5.20554273,  1.12009238],
       [ 1.        , 18.        ,  5.32951289,  1.17191977],
       [ 1.        , 16.        ,  5.25471698,  1.16226415]])
```

```
[ ] def cost_function(X, Y, W):


      m = len(Y)

      J = np.sum((X.dot(W) - Y)** 2) / (2 * m)

      return J
```

```
# Test case
X_test = np.array([[1, 2], [3, 4], [5, 6]])
Y_test = np.array([3, 7, 11])
W_test = np.array([1, 1])

cost = cost_function(X_test, Y_test, W_test)


if cost == 0:
    print("Proceed Further")
else:
    print("Something went wrong: Reimplement the cost function")
    print("Cost function output:", cost)
```

```
Proceed Further
```

```
[ ] initial_cost = cost_function(X2, Y2, W)
    print(initial_cost)
```

```
0.713638501290639
```

```python
def gradient_descent(X, Y, B, alpha, iterations):
    cost_history = [0] * iterations
    m = len(Y)

    for iteration in range(iterations):

        Y_pred = X.dot(B)

        loss = Y_pred - Y

        dw = (X.T.dot(loss) ) / (m)

        W_update = W - alpha * dw

        cost = cost_function(X, Y, W_update)
        cost_history[iteration] = cost

    return W_update, cost_history
```

```python
[ ] alpha = 0.0001
    new_weights, cost_history = gradient_descent(X2, Y2, W, alpha, 100000)


    print(new_weights)


    print(cost_history)
```

```
[0.00010967 0.00309445 0.00069477 0.00014273]
```

```
[ ]  def rmse(Y, Y_pred):

       rmse = np.sqrt(sum((Y-Y_pred)**2)/len(Y))
       return rmse
```

```
     def r2(Y, Y_pred):

       mean_y = np.mean(Y)
       ss_tot = sum((Y - mean_y) ** 2)
       ss_res = sum((Y - Y_pred) ** 2)
       r2 = 1 - (ss_res / ss_tot)
       return r2
```

```
     Y_pred = X2.dot(new_weights)

     print(rmse(Y2, Y_pred))
     print(r2(Y2, Y_pred))
```

```
1.1115518239589113
-4.501576511152731
```

Logistic Regression:-

```
[26] from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
     import pandas as pd
```

```
df = pd.read_csv('/content/drive/MyDrive/Concepts and technologies of AI/Houseprice.csv')
df.head()
```

	HouseAge	HouseFloor	HouseArea	HousePrice
0	52	2	112.945574	543917.179841
1	93	1	174.312126	817740.124828
2	15	4	125.219577	387992.503019
3	72	4	121.210124	240840.742388
4	61	4	59.221737	277273.386525

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
[28] X = df.drop('HouseFloor', axis=1)
     y = df['HouseFloor']
```

```
[29]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[30]
model = LogisticRegression(max_iter=1000) # Increased max_iter to ensure convergence
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

LogisticRegression

LogisticRegression(max_iter=1000)

```
y_pred = model.predict(X_test)
```

```
[32]
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMe
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMe
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMe
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
```

```
Accuracy: 0.25
Confusion Matrix:
[[4 0 0 3 0]
 [3 0 2 1 0]
 [2 0 0 1 0]
 [1 0 0 1 0]
 [1 0 0 1 0]]
Classification Report:
              precision    recall  f1-score   support

           1       0.36      0.57      0.44         7
           2       0.00      0.00      0.00         6
           3       0.00      0.00      0.00         3
           4       0.14      0.50      0.22         2
           5       0.00      0.00      0.00         2

    accuracy                           0.25        20
   macro avg       0.10      0.21      0.13        20
weighted avg       0.14      0.25      0.18        20
```