CSOC_Assign1

May 31, 2025

[2]:
```
#Data Uploading
import pandas as pd
import matplotlib as plt

df = pd.read_csv("D:\\NIKHIL\\Sample Data\\housing.csv")
df.head()
```

[2]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
0	-122.23	37.88	41.0	880.0	129.0	
1	-122.22	37.86	21.0	7099.0	1106.0	
2	-122.24	37.85	52.0	1467.0	190.0	
3	-122.25	37.85	52.0	1274.0	235.0	
4	-122.25	37.85	52.0	1627.0	280.0	

	population	households	median_income	median_house_value	ocean_proximity
0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	496.0	177.0	7.2574	352100.0	NEAR BAY
3	558.0	219.0	5.6431	341300.0	NEAR BAY
4	565.0	259.0	3.8462	342200.0	NEAR BAY

[3]:
```
#Data Processing
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
```

```
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

[5]:
```
df['total_bedrooms'] = df['total_bedrooms'].fillna(df['total_rooms']*0.2026264)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20640 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

[6]:
```
df = df.sample(frac=1).reset_index(drop=True)
```

[7]:
```
df.head()
```

[7]:

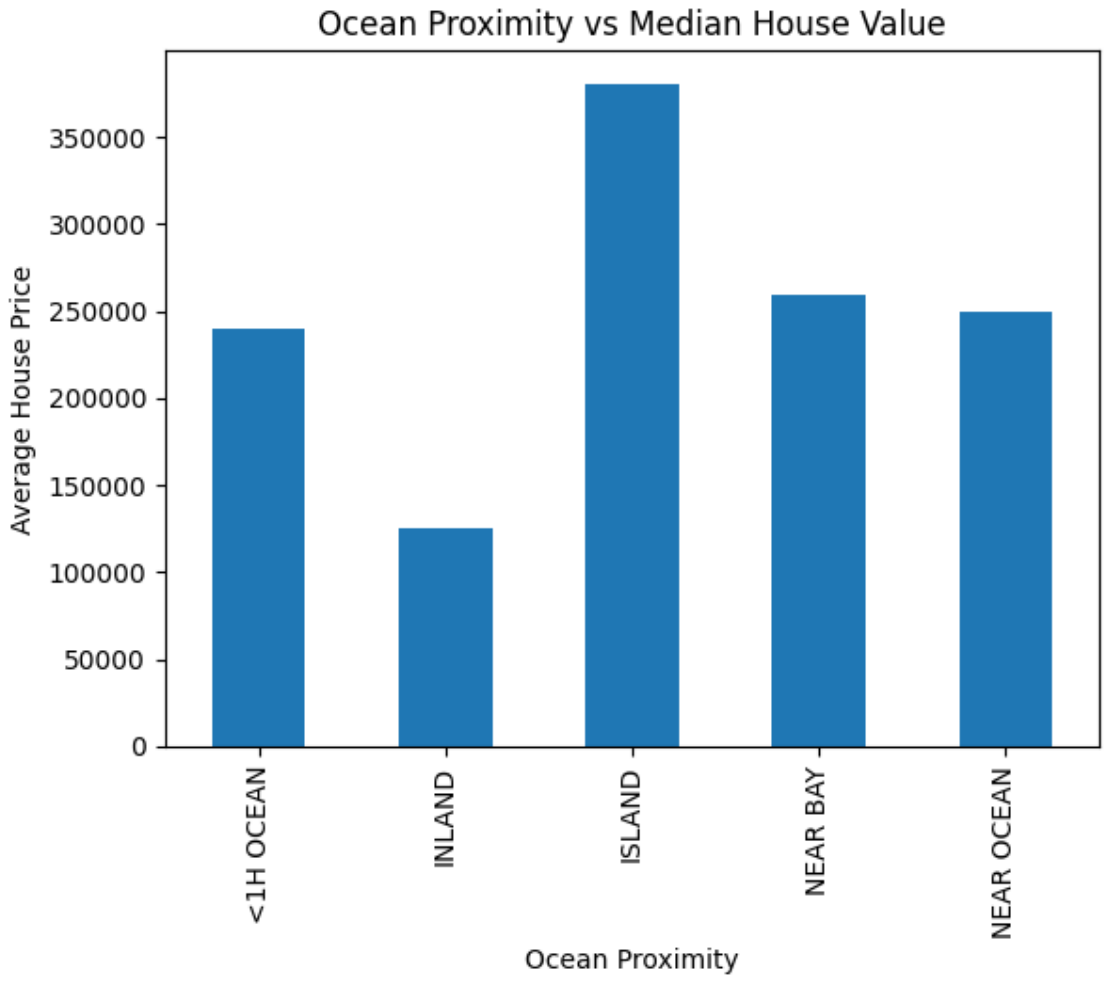
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-121.53	38.48	5.0	27870.0	5027.0	11935.0	4855.0	4.8811	212200.0	INLAND
1	-118.09	34.07	31.0	1054.0	252.0	1032.0	258.0	2.3424	188500.0	<1H OCEAN
2	-118.40	33.98	36.0	2526.0	452.0	996.0	441.0	5.6110	456600.0	<1H OCEAN
3	-118.27	34.06	45.0	564.0	353.0	1172.0	319.0	1.4940	187500.0	<1H OCEAN
4	-122.28	37.84	52.0	729.0	160.0	395.0	155.0	1.6875	132000.0	NEAR BAY

[8]:
```
#Data Processing Ends
```

[9]:
```
#Data Analysis and Plotting
```

[10]:
```
ax = df.groupby('ocean_proximity')['median_house_value'].mean().plot(kind='bar')
ax.set_xlabel('Ocean Proximity')
ax.set_ylabel('Average House Price')
ax.set_title('Ocean Proximity vs Median House Value')
```

[10]: Text(0.5, 1.0, 'Ocean Proximity vs Median House Value')



[11]:
```
ax =df.plot(kind='scatter', x='median_income', y='median_house_value', alpha=0.
↪4)
ax.set_title('Median Income vs House Price')
ax.set_xlabel('Median Income')
ax.set_ylabel('House Price')
```

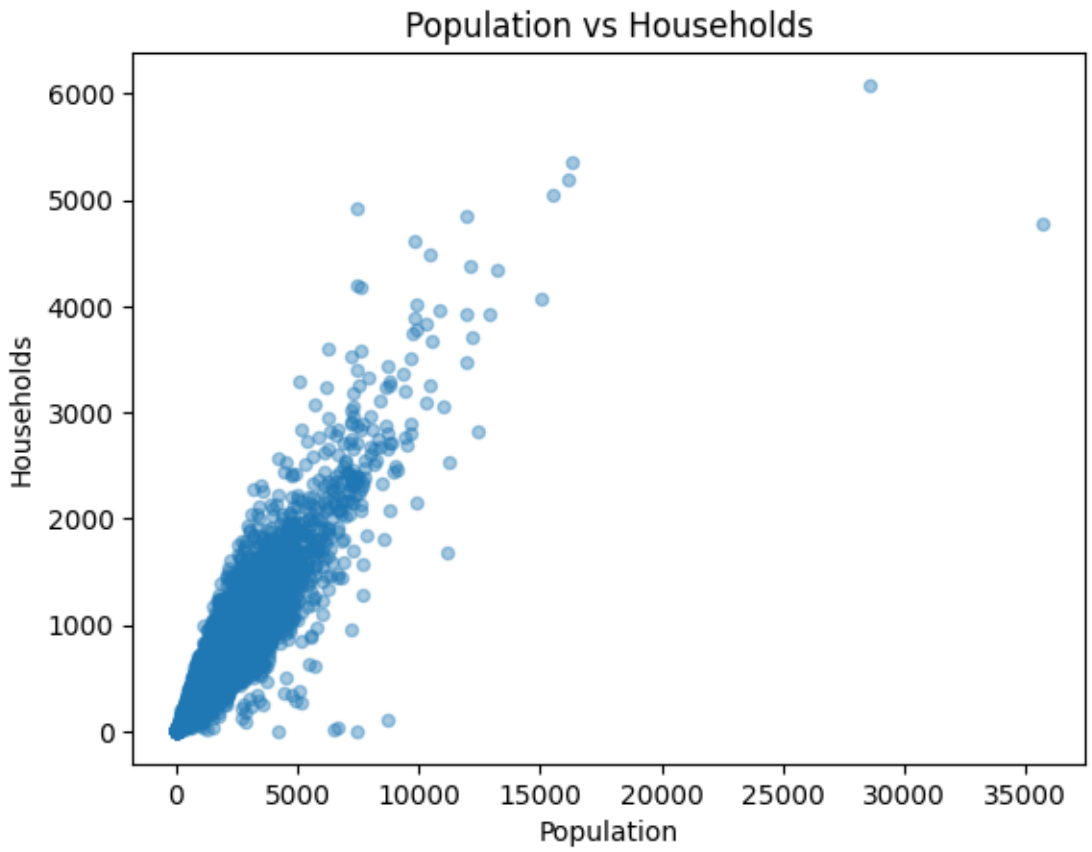
[11]: Text(0, 0.5, 'House Price')

[12]:
```
ax = df.plot(kind='scatter', x='total_rooms', y='median_house_value', alpha=0.4)
ax.set_title('Total Rooms vs House Price')
ax.set_xlabel('Total Rooms')
ax.set_ylabel('House Price')
```

[12]:
```
Text(0, 0.5, 'House Price')
```

[13]:
```
ax = df.plot(kind='scatter', x='population', y='households', alpha=0.4)
ax.set_title('Population vs Households')
ax.set_xlabel('Population')
ax.set_ylabel('Households')
```

[13]:
```
Text(0, 0.5, 'Households')
```

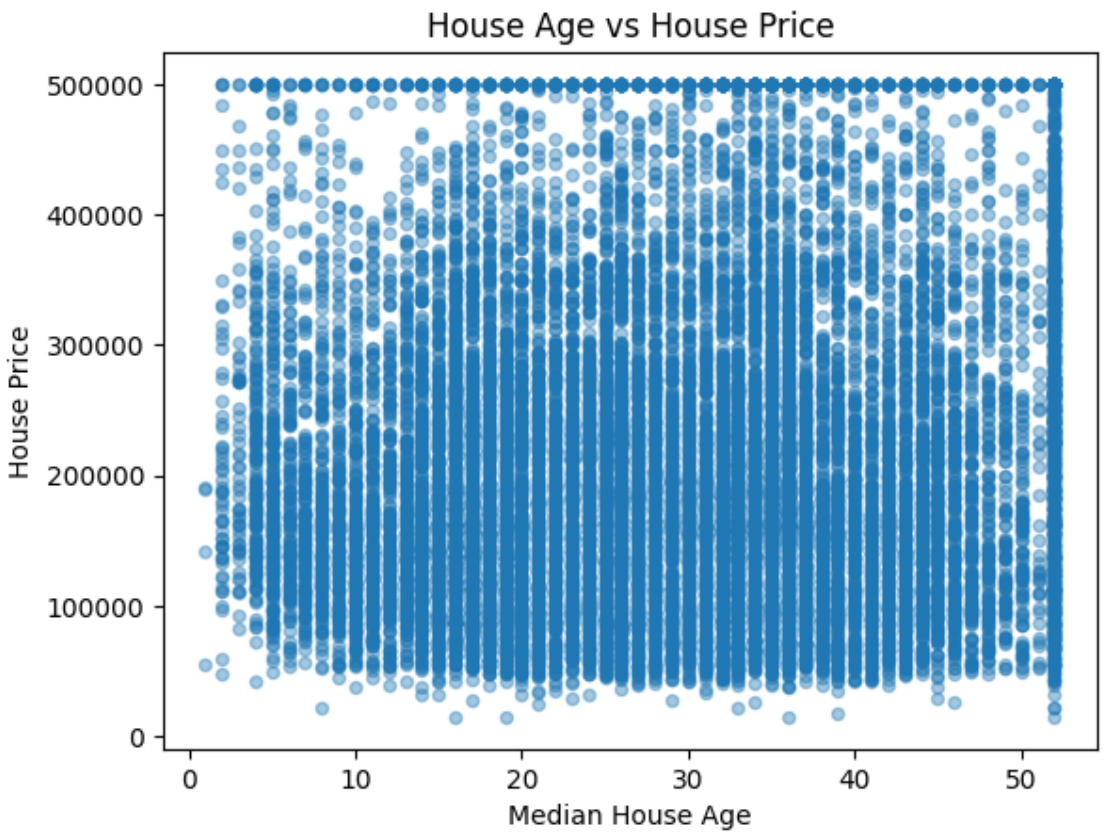


[14]:
```
ax = df.plot(kind='scatter', x='households', y='median_house_value', alpha=0.4)
ax.set_title('Households vs House Price')
ax.set_xlabel('Households')
ax.set_ylabel('House Price')
```

[14]:
```
Text(0, 0.5, 'House Price')
```

[15]:
```
ax = df.plot(kind='scatter', x='housing_median_age', y='median_house_value',␣
↪alpha=0.4)
ax.set_title('House Age vs House Price')
ax.set_xlabel('Median House Age')
ax.set_ylabel('House Price')
```

[15]:
```
Text(0, 0.5, 'House Price')
```



[18]:
```
df.head()
```

[18]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-121.53	38.48	5.0	27870.0	5027.0	11935.0	4855.0	4.8811	212200.0	INLAND
1	-118.09	34.07	31.0	1054.0	252.0	1032.0	258.0	2.3424	188500.0	<1H OCEAN
2	-118.40	33.98	36.0	2526.0	452.0	996.0	441.0	5.6110	456600.0	<1H OCEAN
3	-118.27	34.06	45.0	564.0	353.0	1172.0	319.0	1.4940	187500.0	<1H OCEAN
4	-122.28	37.84	52.0	729.0	160.0	395.0	155.0	1.6875	132000.0	NEAR BAY

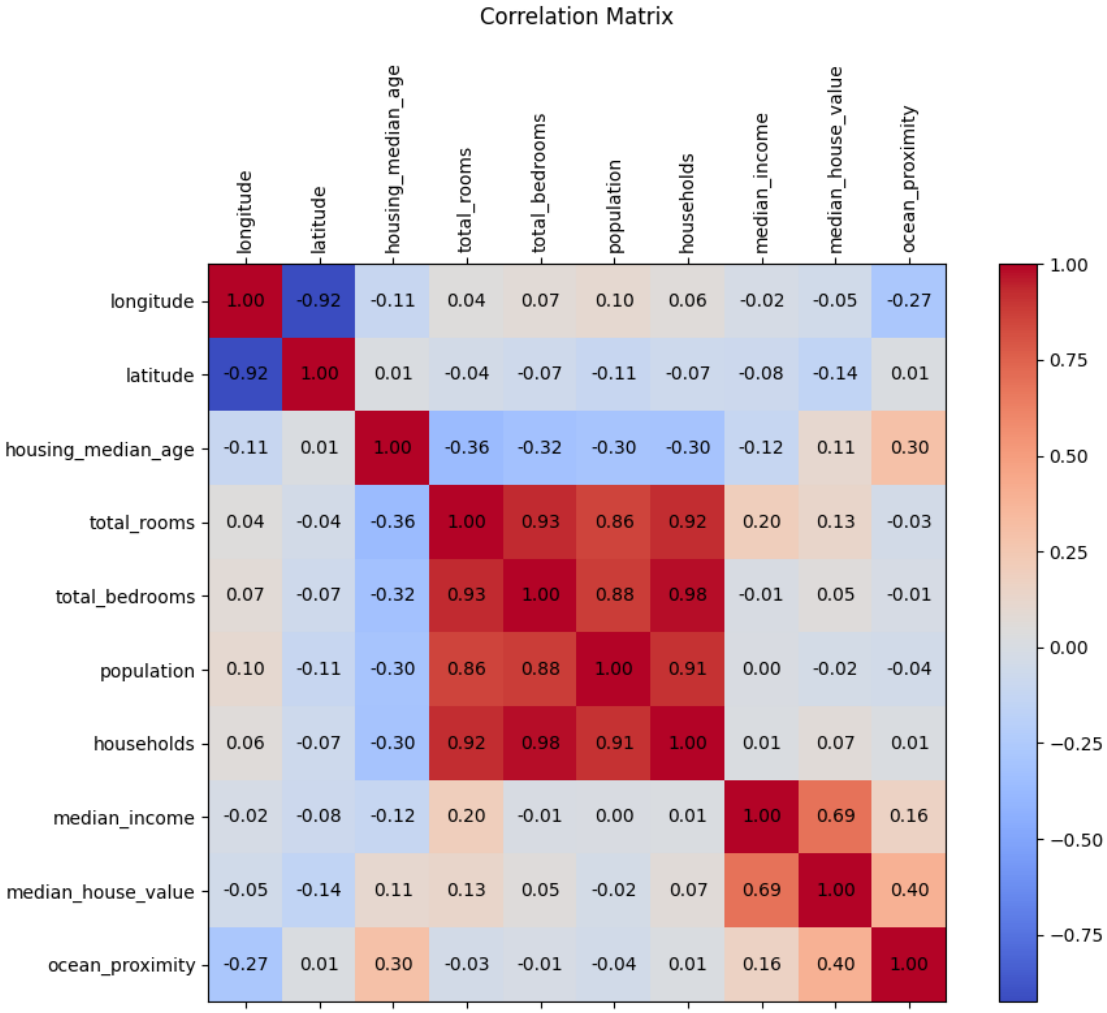
[16]:
```
#splitting Data
total_rows = len(df)
split_index = int(0.8*total_rows)
train_df = df.iloc[:split_index]
```

```
test_df = df.iloc[split_index:]
print(len(train_df))
print(len(test_df))
```

```
16512
4128
```

[19]:
```
#Converting Ocean Proximity into calculatable values (int)
proximity_mapping = {
    'INLAND': 1,
    '<1H OCEAN': 2,
    'NEAR OCEAN': 3,
    'NEAR BAY': 4,
    'ISLAND': 5
}
df['ocean_proximity'] = df['ocean_proximity'].map(proximity_mapping).
 ↪astype('float64')
```

[22]:
```
import matplotlib.pyplot as plt

# Compute correlation matrix
corr = df.corr()

# Plotting the heatmap
fig, ax = plt.subplots(figsize=(10, 8))
cax = ax.matshow(corr, cmap='coolwarm')

# Axis labels
ax.set_xticks(range(len(corr.columns)))
ax.set_yticks(range(len(corr.columns)))
ax.set_xticklabels(corr.columns, rotation=90)
ax.set_yticklabels(corr.columns)

# Annotate with correlation values
for i in range(len(corr.columns)):
    for j in range(len(corr.columns)):
        ax.text(j, i, f"{corr.iloc[i, j]:.2f}", va='center', ha='center',␣
 ↪color='black')

# Colorbar and title
fig.colorbar(cax)
plt.title("Correlation Matrix", pad=20)
plt.tight_layout()
plt.show()
```



[ ]:
```
# Drop 'longitude', 'latitude', and 'population' columns
df = df.drop(columns=['longitude', 'latitude', 'population'])
```

[16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20640 non-null  float64
 5   population          20640 non-null  float64
```

```
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

[18]:
```
df.head()
```

[18]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-121.51	38.52	30.0	3236.0	588.0	1167.0	569.0	4.0972	181400.0	1.0
1	-122.61	38.24	18.0	2933.0	481.0	1279.0	443.0	5.0849	188500.0	2.0
2	-122.32	37.99	24.0	4865.0	968.0	2315.0	893.0	4.2852	173500.0	4.0
3	-119.40	35.06	21.0	2213.0	458.0	1250.0	440.0	2.9187	52100.0	1.0
4	-119.98	38.92	27.0	2682.0	606.0	1010.0	399.0	3.1500	86900.0	1.0

[ ]:
```

```

[20]:
```
#Model Deployment
```

[21]:
```
#Defining Features and Target Variable
x = df[['longitude', 'latitude', 'housing_median_age', 'total_rooms',
        'total_bedrooms', 'population', 'households', 'median_income',
        'ocean_proximity']]

y = df['median_house_value']
```

[22]:
```
#Assigning Random weights and bias
import random
bias = random.uniform(-1,1)
weights = {w : random.uniform(-1,1) for w in x.columns}
print(bias)
print(weights)
```

```
0.10546328029578489
{'longitude': -0.47570320332531524, 'latitude': -0.8395618521860477,
'housing_median_age': 0.8635204422246869, 'total_rooms': 0.7897893465939934,
'total_bedrooms': -0.8332640780598413, 'population': 0.5905296497373969,
'households': -0.8462605632418341, 'median_income': 0.05956736235891391,
'ocean_proximity': 0.2731985580092846}
```

[23]:
```
#Defining Prediction Function
def predict(x_row, weights, bias):
    return sum(x_row[f] * weights[f] for f in x.columns) + bias
```

[24]:
```
#Defining Cost Function
def cost_function(x,y,weights,bias):
    total_error = 0
    for i in range(len(x)):
        cost = y.iloc[i]-(sum(x.iloc[i][f]*weights[f] for f in x.columns) +␣
 ↪bias)
        total_error += cost ** 2
    return total_error/(len(x))
```

[25]:
```
#Defining Gradient Descent Function
def new_weights(x, y, weights, bias, learnrate=0.00000005):
    n = len(x)

    weight_grads = {f: 0 for f in x.columns}
    bias_grad = 0

    for i in range(n):
        # Prediction: ŷ = w1*x1 + w2*x2 + ... + wn*xn + bias
        y_pred = sum(x.iloc[i][f] * weights[f] for f in x.columns) + bias
        error = y_pred - y.iloc[i]

        for f in x.columns:
            weight_grads[f] += (2/n) * error * x.iloc[i][f]
        bias_grad += (2/n) * error

    for f in x.columns:
        weights[f] -= learnrate * weight_grads[f]
    bias -= learnrate * bias_grad

    return bias, weights
```

[30]:
```
#Defining Training Loops
def train(x, y, weights, bias, lr=0.05, epochs=100):
    for epoch in range(epochs):
        for i in range(len(x)):
            y_pred = predict(x.iloc[i], weights, bias)
            error = y_pred - y.iloc[i]

            for f in x.columns:
                weights[f] -= lr * error * x.iloc[i][f]b
            bias -= lr * error
    return weights, bias
```

[ ]:
```python
# Combined Training Function with Batch Gradient Descent
def train(x, y, weights, bias, learnrate=0.00000005, epochs=100):
    n = len(x)

    for epoch in range(epochs):
        weight_grads = {f: 0 for f in x.columns}
        bias_grad = 0

        # Calculate gradients for the whole batch
        for i in range(n):
            y_pred = sum(x.iloc[i][f] * weights[f] for f in x.columns) + bias
            error = y_pred - y.iloc[i]

            for f in x.columns:
                weight_grads[f] += (2/n) * error * x.iloc[i][f]
            bias_grad += (2/n) * error

        # Update weights and bias
        for f in x.columns:
            weights[f] -= learnrate * weight_grads[f]
        bias -= learnrate * bias_grad

    return weights, bias
```

[31]:
```python
# x = (x - x.mean()) / x.std()
```

[32]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20640 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  float64
dtypes: float64(10)
memory usage: 1.6 MB
```

[ ]:
```python
weights, bias = train(x, y, weights, bias)
```

[31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20640 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  float64
dtypes: float64(10)
memory usage: 1.6 MB
```

[32]:
```
df.head()
```

[32]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
0	-119.61	34.43	16.0	2665.0	391.0	
1	-119.79	36.55	32.0	1393.0	276.0	
2	-118.63	34.21	31.0	3952.0	647.0	
3	-121.36	38.15	42.0	2051.0	334.0	
4	-118.00	33.77	28.0	2401.0	503.0	

	population	households	median_income	median_house_value	ocean_proximity
0	794.0	311.0	9.0267	500001.0	2.0
1	999.0	245.0	2.0216	76800.0	1.0
2	1762.0	588.0	5.5709	244800.0	2.0
3	878.0	318.0	4.3553	185700.0	1.0
4	1155.0	456.0	3.5139	211700.0	2.0

[33]:
```
# Explicitly exclude any known non-numeric column like 'ocean_proximity'
exclude_columns = ['median_house_value', 'ocean_proximity']
feature_columns = [col for col in df.columns if col not in exclude_columns and␣
 ↪df[col].dtype in ['int64', 'float64']]

# Extract test features and labels
test_x = test_df[feature_columns].apply(pd.to_numeric, errors='coerce')  #␣
 ↪Convert all to numeric safely
y_test = test_df['median_house_value']

# Updated predict function
def predict(x_row, weights, bias):
```

```
    return sum(x_row[col] * weights[col] for col in x_row.index) + bias

# Final Testing
predictions = [predict(test_x.iloc[i], weights, bias) for i in␣
 ↪range(len(test_x))]

# Calculate test error (Mean Squared Error)
test_error = sum((predictions[i] - y_test.iloc[i])**2 for i in␣
 ↪range(len(y_test))) / len(y_test)
print("Test MSE:", test_error)
```

```
Test MSE: 76870083298715.58
```

[34]:
```
mean_y = sum(y_test) / len(y_test)
ss_total = sum((y - mean_y) ** 2 for y in y_test)
ss_res = sum((predictions[i] - y_test.iloc[i])**2 for i in range(len(y_test)))
r2_score = 1 - (ss_res / ss_total)
print("R² Score:", r2_score)
```

```
R² Score: -5932.602677077416
```

[35]:
```
mean_y = sum(y_test) / len(y_test)
ss_total = sum((y - mean_y) ** 2 for y in y_test)
ss_res = sum((predictions[i] - y_test.iloc[i])**2 for i in range(len(y_test)))
r2_score = 1 - (ss_res / ss_total)
print("R² Score:", r2_score)
```

```
R² Score: -5932.602677077416
```

[36]:
```
mae = sum(abs(predictions[i] - y_test.iloc[i]) for i in range(len(y_test))) /␣
 ↪len(y_test)
print("MAE:", mae)
```

```
MAE: 6646243.186668597
```