```
"Gradient Decent and Multiple Linear Regression"
import numpy as np
import os
import pandas as pd
import seaborn as sns
from scipy.stats import entropy
import matplotlib.pyplot as plt
from tqdm import tqdm
%matplotlib inline
    # Your plotting code here
```

```
"Define the task"
df = pd.read_csv('/content/drive/MyDrive/housing.csv')
```

```
df.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefar
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	

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

```
df.columns
```

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
```

```
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

```
df = df[['area','price']]
```

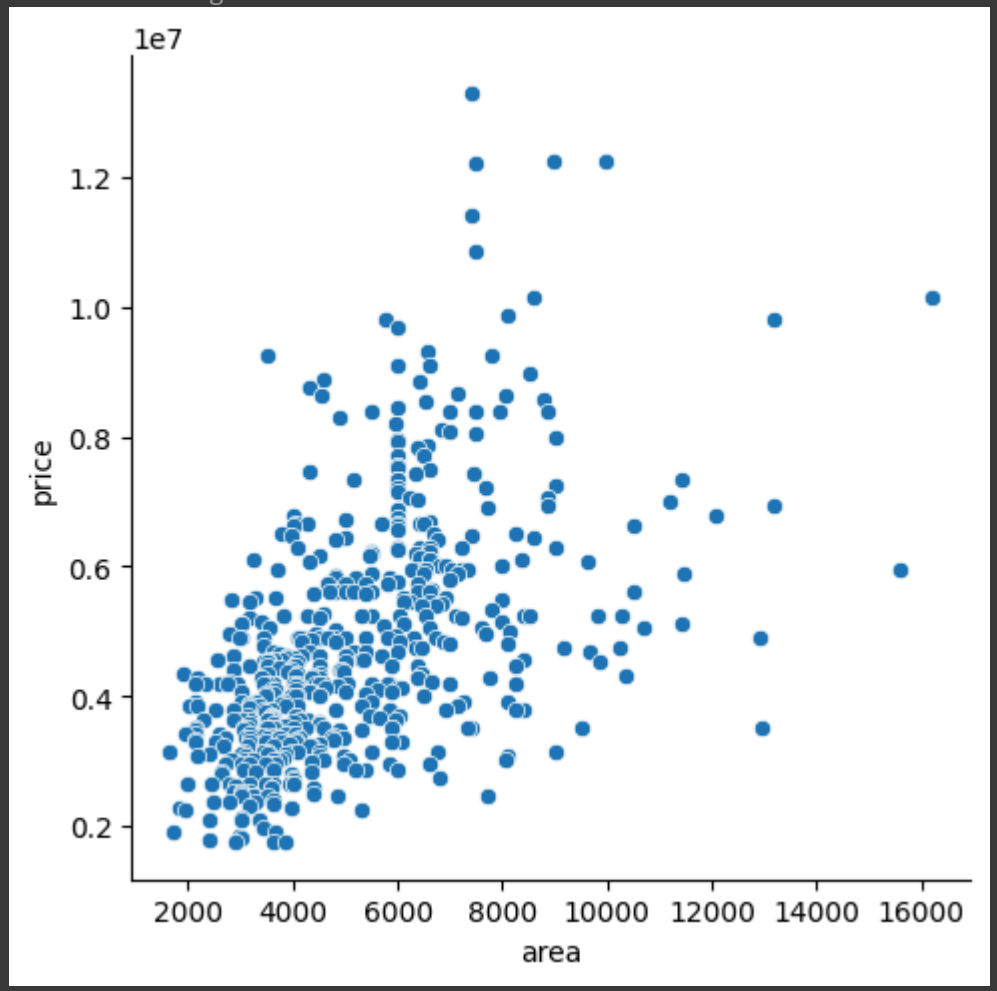
```
df.head()
```

	area	price
0	7420	13300000
1	8960	12250000
2	9960	12250000
3	7500	12215000
4	7420	11410000

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

```
sns.relplot(
    data = df,
    x = 'area',
    y = 'price',
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7e49e5130c90>
```



Data Pipeline

- Standardize the data

```
mean = np.array(df.mean())
print(mean)
```

```
std = np.array(df.std())
print(std)
df = (df - mean)/std
df.head()
```

```
[   5150.5412844  4766729.24770642]
[   2170.14102251 1870439.61565739]
```

	area	price
0	1.045766	4.562174
1	1.755397	4.000809
2	2.216196	4.000809
3	1.082630	3.982096
4	1.045766	3.551716

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

Machine Learning Algorithm

```
def get_house_price(X,w,b):
  y_pred = w * X + b
  return y_pred
```

```
"Initialize the parameters of the most"
w = np.random.randint(100,200)
b = np.random.randint(100,200)
print(w,b)
```

```
184 130
```

```
df['price_pred_random'] = get_house_price(df['area'],w,b)
df.head()
```

	area	price	price_pred_random
0	1.045766	4.562174	322.420861
1	1.755397	4.000809	452.993020
2	2.216196	4.000809	537.780137
3	1.082630	3.982096	329.203830
4	1.045766	3.551716	322.420861

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

```
"Zero-shot Learning: Learning without explicit training. "
```

```
'Zero-shot Learning: Learning without explicit training. '
```

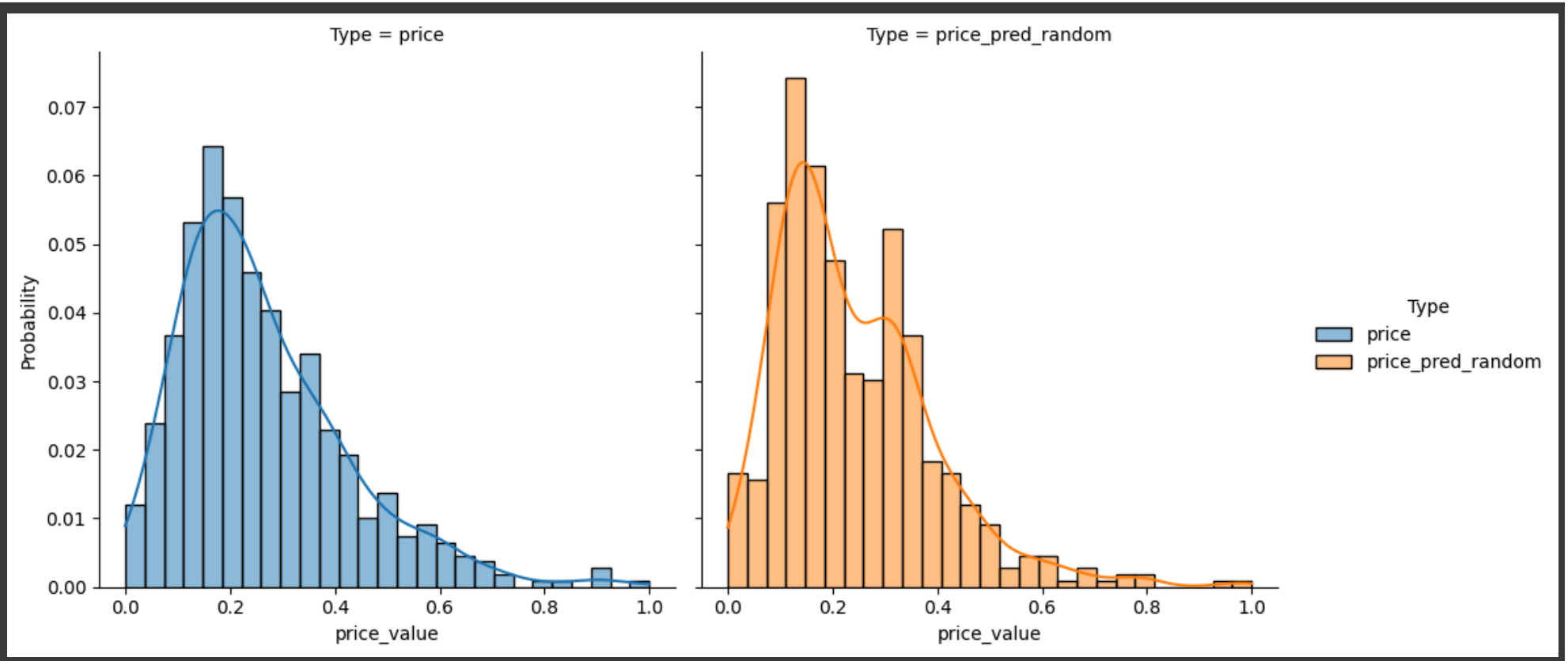
```
__ = df.melt(
    value_vars = ["price","price_pred_random"],
    var_name = "Type",
    value_name = "price_value"
 )
__['price_value'] = __.groupby('Type')['price_value'].transform(
    lambda x: (x-x.min())/(x.max()-x.min())
)
__.head()
```

	Type	price_value
0	price	1.000000
1	price	0.909091
2	price	0.909091
3	price	0.906061
4	price	0.836364

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

Compare actual price and price prediction random values.

```
sns.displot(
    data=__,              # Your DataFrame
    x='price_value',      # Variable to plot on X-axis
    hue='Type',           # Different colors for categories
    kind='hist',          # Plot histogram
    stat='probability',   # Normalize bars so total area = 1
    kde=True,             # Overlay Kernel Density Estimation curve
    col='Type',           # Create separate subplot for each category in 'Type'
    fill=True             # Fill under KDE curve
)
plt.show()
```



Calculate KL Divergence Before learning.

```
price_hist,__ = np.histogram(
    df['price'],
    bins = 50,
    density = True
)
```

```
price_pred_hist,__= np.histogram(
    df['price_pred_random'],
    bins = 50,
    density = True
)
kl_divergence = entropy(price_hist + 1e-10,price_pred_hist + 1e-10)
print(f"KL Divergence is: {kl_divergence}")
```

```
KL Divergence is: 0.3609980742619633
```

Cost Functions

ML Learning Algorithm:

1. Maximize likelihood between true distribution and predicted distribution Alternatively, minimizing the kl divergence Alternatively, minimizing loss/cost function

- We want to maximize likelihood between true distribution and predicted distribution
- We want to minimize dissimilarity between true distribution and predicted distribution
- We consider a function that would minimize the kl divergence if it itself minimizes.
- This function called cost function.
- cost function is often task specific
- We will use Mean Square Error as our cost function.
- y = 10, yhat = 12, e1 = (y-yhat)^2 =4
- y = 5, yhat = 6, e2 = (5-6)^2 = 1
- y = 2, yhat = 2, e3=0,
- e = (e1+e2+e3)/3 = (4+1+0)/3 = 5/3

Calculate loss function value.

```
def cost_function(x,y_true,w,b):
  y_pred = get_house_price(y_true,w,b)
  mse = np.mean((y_true - y_pred)**2) /2
  return mse

X = df['area']
y_true = df['price']

loss = cost_function(X, y_true, w,b)
print(f"Loss: {loss}")
```

```
Loss: 25163.776146788954
```

```
loss1 = cost_function(X,y_true,w =100, b=125)
print(f"Loss: {loss1}")
loss2 = cost_function(X,y_true,w =50, b=50)
print(f"Loss: {loss2}")
loss3 = cost_function(X,y_true,w =10, b=10)
print(f"Loss: {loss3}")
```

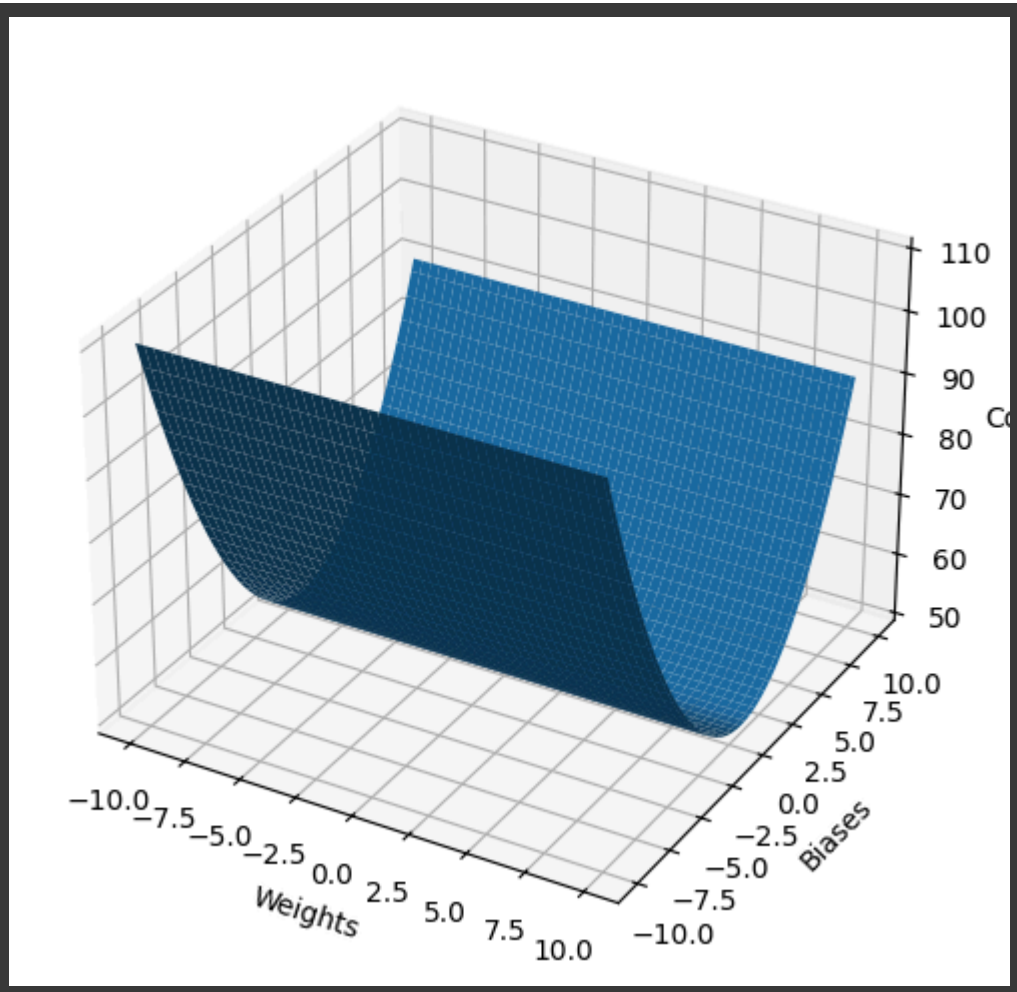
```
Loss: 12704.008256880723
Loss: 2448.2972477064195
Loss: 90.42568807339438
```

```
weights = np.linspace(-10,10,100)
biases = np.linspace(-10,10,100)
weights_mesh, biases_mesh = np.meshgrid(weights,biases)
losses_mesh = []

for w in tqdm(np.ravel(weights_mesh[0])):
  for b in np.ravel(biases_mesh[0]):
```

```
    loss = cost_function(X,y_true,w,b)
    losses_mesh.append(loss)

losses_mesh = np.array(losses_mesh)
losses_mesh = losses_mesh.reshape(weights_mesh.shape)
```

```
100%|██████████| 100/100 [00:03<00:00, 26.66it/s]
```

```
fig = plt.figure(figsize = (6,6))
ax = fig.add_subplot(111, projection = '3d')
ax.plot_surface(weights_mesh,biases_mesh,losses_mesh)
ax.set_xlabel('Weights')
ax.set_ylabel('Biases')
ax.set_zlabel('Cost')
plt.show()
```



Gradient Decent Algorithm

- Compute Gradient
- Update Gradient

```
def compute_gradient(X,y_true,w,b):
  delta = 1e-9
```

```
  cost1 = cost_function(X,y_true,w,b)
  cost2 = cost_function(X,y_true,w+delta, b)
  cost3 = cost_function(X,y_true, w, b+delta)

  dw = (cost2 - cost1)/delta
  db = (cost3 - cost1)/delta
  return dw,db
```

```
print(w,b)
```

```
10.0 -10.0
```

```
loss = cost_function(X,y_true,w,b)
print(loss)
```

```
90.42568807339437
```

```
dw , db = compute_gradient(X,y_true,w,b)
print(dw,db)
```

```
8.983491284197953 -10.000007932831068
```

- Let dw is 3.5, w = w - dw
- let dw is -3.5, w = w + dw = w - (-dw)
- Learning Rate 0.0001 to 0.00001

```
learning_rate = 0.001
w = w - learning_rate *dw
b = b - db * learning_rate
```

```
loss = cost_function(X,y_true,w,b)
print(loss)
```

```
90.24507520133808
```

```
for epoch in range(10000):
    loss = cost_function(X, y_true, w, b)   # 1 Compute cost
    dw, db = compute_gradient(X, y_true, w, b)  # 2 Compute gradients

    # 3 Update parameters
    w = w - learning_rate * dw
    b = b - learning_rate * db

    # 4 Track progress
    if epoch % 1000 == 0:
        print(loss)
```

```
90.24507520133808
12.22119934697334
1.6550289690764202
0.22412941129328492
0.030352435234231888
0.004110453073821865
0.0005566565231571593
7.538524699699955e-05
1.020908632974978e-05
1.3825754768553913e-06
```

```
print(w,b)
df.head()
```

```
1.0004136822306313 -0.00045128212097576885
```

	area	price	price_pred_random
0	1.045766	4.562174	322.420861
1	1.755397	4.000809	452.993020
2	2.216196	4.000809	537.780137
3	1.082630	3.982096	329.203830
4	1.045766	3.551716	322.420861

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

```
df["price_pred_learned"] = get_house_price(df['price'],w,b)
df.head()
```

	area	price	price_pred_random	price_pred_learned
0	1.045766	4.562174	322.420861	4.563610
1	1.755397	4.000809	452.993020	4.002012
2	2.216196	4.000809	537.780137	4.002012
3	1.082630	3.982096	329.203830	3.983292
4	1.045766	3.551716	322.420861	3.552734

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

```
__ = df.melt(
    value_vars = ["price","price_pred_learned"],
    var_name = "Type",
    value_name = "price_value"
 )
```

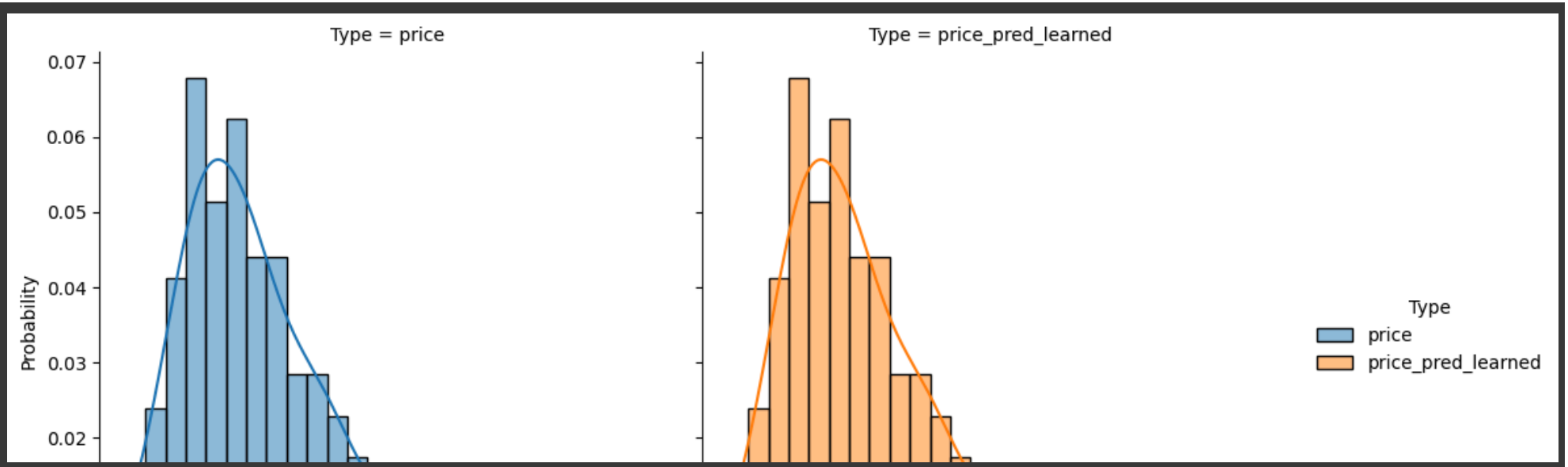
```
__['price_value'] = __.groupby('Type')['price_value'].transform(
    lambda x: (x-x.min())/(x.max()-x.min())
)
__.head()
```

	Type	price_value
0	price	1.000000
1	price	0.909091
2	price	0.909091
3	price	0.906061
4	price	0.836364

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

Compare actual price and Price pread learned. But there is no differents.

```
sns.displot(
    data=__,              # Your DataFrame
    x='price_value',      # Variable to plot on X-axis
    hue='Type',           # Different colors for categories
    kind='hist',          # Plot histogram
    stat='probability',   # Normalize bars so total area = 1
    kde=True,             # Overlay Kernel Density Estimation curve
    col='Type',           # Create separate subplot for each category in 'Type'
    fill=True             # Fill under KDE curve
)
plt.show()
```


Type = price
Type = price_pred_learned
Probability
0.07
0.06
0.05
0.04
0.03
0.02
Type
price
price_pred_learned