```
import os

for root, dirs, files in os.walk('/content/ratings_Electronics 3.csv'):
    for file in files:
        if file.endswith('.csv'):
            print(os.path.join(root, file))
```

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

```
!pip install scikit-surprise
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 kB 32.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.10.1)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp310-cp310-linux_x86_64.whl size=3095447 sha256=3fca
  Stored in directory: /root/.cache/pip/wheels/a5/ca/a8/4e28def53797fdc4363ca4af740db15a9c2f1595ebc51fb445
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.3
```

```
from collections import defaultdict
from surprise import accuracy
from sklearn.metrics import mean_absolute_error
from surprise.model_selection import train_test_split
```

```
df = pd.read_csv("/content/ratings_Electronics 3.csv", names=["userId", "productId", "rating", "timestamp"])
df.head()
```

	userId	productId	rating	timestamp
0	AKM1MP6P0OYPR	0132793040	5.0	1365811200
1	A2CX7LUOHB2NDG	0321732944	5.0	1341100800
2	A2NWSAGRHCP8N5	0439886341	1.0	1367193600
3	A2WNBOD3WNDNKT	0439886341	3.0	1374451200
4	A1GI0U4ZRJA8WN	0439886341	1.0	1334707200

```
rows_count, columns_count = df.shape
df.dtypes
```

```
userId        object
productId     object
rating       float64
timestamp    float64
dtype: object
```

```
unique_userId = df['userId'].nunique()
unique_productId = df['productId'].nunique()
```

```
#checking null values
df.apply(lambda x : sum(x.isnull()))
```

```
userId       0
productId    0
rating       0
timestamp    1
dtype: int64
```

```
df_t = df.describe().T
df_t
```

	count	mean	std	min	25%	50%	75%	max
rating	927109.0	3.977244e+00	1.398075e+00	1.0	3.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00

```
#Check rating
df['rating'].value_counts()
```

```
5.0    501415
4.0    184366
1.0    113162
3.0     73306
2.0     54860
Name: rating, dtype: int64
```

```
r_count = pd.DataFrame(df['rating'].value_counts()).reset_index()
r_count.columns = ['Labels', 'Ratings']
r_count
```

	Labels	Ratings
0	5.0	501415
1	4.0	184366
2	1.0	113162
3	3.0	73306
4	2.0	54860

```
df_c = df.copy()
```

```
users_counts = df_c['userId'].value_counts().rename('users_counts')
users_data   = df_c.merge(users_counts.to_frame(),
                          left_on='userId',
                          right_index=True)
```

```
df2 = users_data[users_data.users_counts >= 10]
df2.head()
```

	userId	productId	rating	timestamp	users_counts
118	AT09WGFUM934H	0594481813	3.0	1.377907e+09	13
77282	AT09WGFUM934H	B00005105L	5.0	1.377475e+09	13
160734	AT09WGFUM934H	B000068O1M	5.0	1.384906e+09	13
160973	AT09WGFUM934H	B000068O34	5.0	1.384733e+09	13
162248	AT09WGFUM934H	B000068O4J	5.0	1.387238e+09	13

```
df.corr()
```

```
<ipython-input-55-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a
  df.corr()
```

	rating	timestamp
rating	1.000000	0.100472
timestamp	0.100472	1.000000

```
product_rating_counts = df2['productId'].value_counts().rename('product_rating_counts')
product_rating_data   = df2.merge(product_rating_counts.to_frame(),
                                  left_on='productId',
                                right_index=True)
```

```
product_rating_data = product_rating_data[product_rating_data.product_rating_counts >= 50]
product_rating_data.head()
```

```
                    userId  productId  rating   timestamp  users_counts  product_rating_counts
electronics_df = product_rating_data.copy()

data = electronics_df.drop(['userId', 'productId'], axis=1)
     330836  A3QQGFI3CTV9SX  B0000BZLTF     5.0  1.211933e+09            34                     60
data.head()
```

	rating	timestamp	users_counts	product_rating_counts
330872	5.0	1.327277e+09	25	60
330777	5.0	1.282003e+09	15	60
330836	5.0	1.211933e+09	34	60
330305	5.0	1.397434e+09	10	60
330714	4.0	1.384906e+09	11	60

Result Evaluation

```
k = 5

from surprise import Reader
reader = Reader(rating_scale=(1, 5))

from surprise import Dataset
Research_data = Dataset.load_from_df(data[['users_counts', 'product_rating_counts', 'rating']], reader)

trainset, testset = train_test_split(Research_data, test_size=0.30, random_state=7)

X = data.drop('timestamp', axis=1)  # replace 'target_variable' with the name of your target variable column
y = data['rating']

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

```
y_pred = lr.predict(X_test)

# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred)
print('Mean squared error: {:.2f}'.format(mse))

# Evaluate the model using accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy score of Rating: {:.2f}'.format(accuracy))

# Evaluate the model using precision
precision = precision_score(y_test, y_pred, average = 'micro')
print('precision score of Rating: {:.2f}'.format(precision))

# Evaluate the model using precision
r2 = r2_score(y_test, y_pred)
print('r2 score of Rating: {:.2f}'.format(r2))
```

```
Mean squared error: 0.00
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-81-66ee93d6b615> in <cell line: 6>()
      4
      5 # Evaluate the model using accuracy
----> 6 accuracy = accuracy_score(y_test, y_pred)
      7 print('Accuracy score of Rating: {:.2f}'.format(accuracy))
      8

                                  2 frames
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py in _check_targets(y_true, y_pred)
     93
     94     if len(y_type) > 1:
---> 95         raise ValueError(
     96             "Classification metrics can't handle a mix of {0} and {1} targets".format(
     97                 type_true, type_pred

ValueError: Classification metrics can't handle a mix of multiclass and continuous targets
```

```
from sklearn.metrics import accuracy_score, precision_score, r2_score
from sklearn.linear_model import LogisticRegression
```

```
# Train the model
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converg
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()
```

```
# Make predictions on the testing set
y_pred = lr.predict(X_test)
```

```
# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred)
print('Mean squared error of rating: {:.2f}'.format(mse))
```

```
Mean squared error of rating: 0.14
```

```
# Evaluate the model using accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy score of Rating: {:.2f}'.format(accuracy))
```

```
Accuracy score of Rating: 0.86
```

```
# Evaluate the model using precision
precision = precision_score(y_test, y_pred, average = 'micro')
print('precision score of Rating: {:.2f}'.format(precision))
```

```
precision score of Rating: 0.86
```

```
# Evaluate the model using precision
r2 = r2_score(y_test, y_pred)
print('r2 score of Rating: {:.2f}'.format(r2))
```

```
r2 score of Rating: 0.86
```