HW4_Solution

February 3, 2024

0.1 Question 1

0.1.1 Dataset

`homework4_file1.csv`

0.1.2 Data Description

The dataset contains records of merchant transactions, each with a unique merchant identifier, time of transaction, and amount in cents.

0.1.3 Objective

Analyze merchant transaction data to understand business growth and health. Preprocess the dataset for future merchant transactions and generate specific features for each merchant.

0.1.4 Task

Generate the following features for each unique merchant: - **trans_amount_avg:** Average transaction amount for each merchant. - **trans_amount_volume:** Total transaction amount for each merchant. - **trans_frequency:** Total count of transactions for each merchant. - **trans_recency:** Recency of the last transaction (in days from 1/1/2035). - **avg_time_btwn_trans:** Average time between transactions (in hours). - **avg_trans_growth_rate:** Average growth rate in transaction amounts.

0.1.5 Data Dimension

The dataset is N by 3, where N is the number of records.

0.1.6 Final Deliverables

- Shape of the new dataset.
- The top five rows of the new dataset using `new_dataset.head()`.
- Descriptive statistics of the new dataset.

[2]:
```
import pandas as pd
import numpy as np
```

[3]:
```
data1 = pd.read_csv('homework4_file1.csv')
```

[5]:
```
data1.shape
```

[5]: (100000, 3)

[7]:
```
data1.head()
```

[7]:
```
     merchant                 time  amount_usd_in_cents
0  d087d4c321  2034-12-11 22:16:41                 5059
1  fe1cb2e840  2034-08-13 21:11:59                12743
2  878047f4b9  2033-06-05 21:15:00                 7601
3  3932608d23  2034-04-28 19:55:01                 5790
4  84a09b4188  2034-07-26 04:37:05                 6153
```

[8]:
```
#Count unique merchant
data1['merchant'].nunique()
```

[8]: 7902

[9]:
```
#transform to datetime format
data1['time'] = data1['time'].apply(pd.to_datetime)
```

[10]:
```
#sort time, group by merchant, and aggregate into a list
data1 = data1.sort_values(by='time')
data1_agg = data1.groupby('merchant').agg(list).reset_index()
```

[11]:
```
data1_agg.head()
```

[11]:
```
     merchant                                              time  \
0  00057d4302         [2033-05-30 02:54:34, 2033-05-30 04:20:31]
1  000ed1585f  [2033-05-08 15:51:43, 2033-05-19 13:15:51, 203…
2  000f8c3297  [2033-12-20 17:57:20, 2034-01-26 15:10:54, 203…
3  0020aefbd9                              [2034-05-30 21:55:06]
4  0026f256ac                              [2033-09-15 01:17:32]

                                 amount_usd_in_cents
0                                       [1156, 1279]
1                       [32004, 35784, 21932, 22481]
2  [7374, 14489, 15047, 4623, 3826, 3643, 4155, 7…
3                                             [3589]
4                                            [34880]
```

[13]:
```
data1_agg.shape
```

[13]: (7902, 3)

[14]:
```
#function that calculates the average time between transactions for each␣
 ↪merchant
def avg_time_btwn_trans(y):
    if len(y) == 1:
```

```
        return 0
    else:
        td = np.mean([z1 - z0  for z0, z1 in zip(y,y[1:])])
        td_hrs = (td.days)*24 + (td.seconds)/(60*60)
        avg_time_btwn_trans = np.round(td_hrs, 2)
        return avg_time_btwn_trans
```

[15]:
```
#function that calculates the average transaction growth rate for each merchant
def avg_trans_growth_rate(y):
    if len(y) == 1:
        return 0
    else:
        grw = np.mean([(z1 - z0)/z0  for z0, z1 in zip(y,y[1:])])
        avg_trans_growth_rate = np.round(grw, 2)
        return avg_trans_growth_rate
```

[18]:
```
data1['time'].max()
```

[18]:
```
Timestamp('2034-12-31 07:55:58')
```

[19]:
```
def get_aggregated(x):
    x['trans_amount_avg'] = x['amount_usd_in_cents'].apply(lambda y: np.
 ↪round(np.mean(y), 2))
    x['trans_amount_volume'] = x['amount_usd_in_cents'].apply(lambda y: np.
 ↪round(np.sum(y),2))
    x['trans_frequency'] = x['amount_usd_in_cents'].apply(lambda y: len(y))
    x['trans_recency'] = x['time'].apply(lambda t: (np.datetime64('2035-01-01')␣
 ↪- np.max(t)).days + 1)
    x['avg_time_btwn_trans'] = x['time'].apply(lambda t: avg_time_btwn_trans(t))
    x['avg_trans_growth_rate'] = x['amount_usd_in_cents'].apply(lambda y:␣
 ↪avg_trans_growth_rate(y))
    return x
```

[20]:
```
new_dataset = get_aggregated(data1_agg)
```

[21]:
```
new_dataset.head()
```

[21]:
```
     merchant                                                time  \
0  00057d4302         [2033-05-30 02:54:34, 2033-05-30 04:20:31]
1  000ed1585f  [2033-05-08 15:51:43, 2033-05-19 13:15:51, 203…
2  000f8c3297  [2033-12-20 17:57:20, 2034-01-26 15:10:54, 203…
3  0020aefbd9                              [2034-05-30 21:55:06]
4  0026f256ac                              [2033-09-15 01:17:32]

                         amount_usd_in_cents  trans_amount_avg  \
0                               [1156, 1279]           1217.50
1                 [32004, 35784, 21932, 22481]         28050.25
```

```
2  [7374, 14489, 15047, 4623, 3826, 3643, 4155, 7…          6635.56
3                                             [3589]          3589.00
4                                            [34880]         34880.00

   trans_amount_volume  trans_frequency  trans_recency  avg_time_btwn_trans  \
0                 2435                2            581                 1.43
1               112201                4            175              3424.03
2               106169               16             59               508.47
3                 3589                1            216                 0.00
4                34880                1            473                 0.00

   avg_trans_growth_rate
0                   0.11
1                  -0.08
2                   0.23
3                   0.00
4                   0.00
```

[22]:
```
new_dataset.describe()
```

[22]:
```
       trans_amount_avg  trans_amount_volume  trans_frequency  trans_recency  \
count      7.902000e+03         7.902000e+03      7902.000000    7902.000000
mean       3.073318e+04         1.963547e+05        12.655024     170.320299
std        1.417803e+05         6.000438e+05        46.531552     180.309019
min        2.090000e+02         2.090000e+02         1.000000       1.000000
25%        4.846177e+03         1.025200e+04         1.000000      26.000000
50%        9.053630e+03         3.484000e+04         3.000000      98.000000
75%        2.114705e+04         1.388630e+05         8.000000     265.000000
max        1.038551e+07         1.549983e+07      1673.000000     727.000000

       avg_time_btwn_trans  avg_trans_growth_rate
count          7902.000000            7902.000000
mean            749.494185               1.011835
std            1461.800362               9.954018
min               0.000000              -1.000000
25%               0.000000               0.000000
50%             170.445000               0.030000
75%             841.512500               0.560000
max           15327.180000             606.650000
```

0.2 Question 2

0.2.1 Datasets Provided

- `sales_data.csv`
- `product_info.csv`

0.2.2 `sales_data.csv`

Contains transaction records with columns: `- TransactionID - ProductID - Date - Quantity - Price`

0.2.3 `product_info.csv`

Contains product details with columns: `- ProductID - ProductName - Category`

0.2.4 Tasks

Your task involves multiple steps of data manipulation using Pandas and NumPy to extract insights from these datasets.

0.2.5 1. Data Loading and Merging

- Load both datasets using Pandas.
- Merge them into a single DataFrame on `ProductID`.

0.2.6 2. Data Cleaning

- Check for and handle any missing values in the merged dataset.
- Convert the `Date` column to a DateTime object.

0.2.7 3. Data Analysis using Slicing and Indexing

- Create a new column `TotalSale`, calculated as `Quantity` * `Price`.
- Using slicing, create a subset DataFrame containing only transactions from the last quarter of the year (October, November, December).
- Using Boolean indexing, find all transactions for a specific `Category` (e.g., `Electronics`).
- Extract all transactions where the `TotalSale` is above the 75th percentile of the `TotalSale` column using NumPy functions.

0.2.8 4. Advanced Indexing

- Using `loc` and `iloc`, perform the following:
  - Select all rows for `ProductID` 101 and columns `ProductName` and `TotalSale`.
  - Select every 10th row from the merged dataset and only the columns `Date` and `Category`.

0.2.9 5. Grouping and Aggregation

- Group the data by `Category` and calculate the total and average `TotalSale` for each category.

0.2.10 6. Time-Series Analysis

- Resample the data on a monthly basis and calculate the total `Quantity` sold per month.

0.2.11 Final Deliverables

- Provide the code for each step.
- Include comments explaining your approach.

- Display the first 5 rows of the DataFrame after each major step.

[23]:
```
# Load datasets
sales_data = pd.read_csv('sales_data.csv')
product_info = pd.read_csv('product_info.csv')
```

[24]:
```
# Merge datasets on ProductID
merged_data = pd.merge(sales_data, product_info, on='ProductID')
```

[25]:
```
sales_data.head()
```

[25]:

	TransactionID	ProductID	Date	Quantity	Price
0	1	136	2023-03-13	8	245.288680
1	2	121	2023-06-09	2	355.603776
2	3	179	2023-04-18	7	25.393345
3	4	142	2023-09-03	10	260.758110
4	5	101	2023-06-21	1	212.490775

[27]:
```
product_info.head()
```

[27]:

	ProductID	ProductName	Category
0	100	not	Clothing
1	101	ready	Clothing
2	102	fill	Books
3	103	avoid	Clothing
4	104	beyond	Toys

[28]:
```
merged_data.head()
```

[28]:

	TransactionID	ProductID	Date	Quantity	Price	ProductName	\
0	1	136	2023-03-13	8	245.288680	pull	
1	92	136	2023-07-02	6	21.266893	pull	
2	260	136	2023-04-15	2	356.242853	pull	
3	411	136	2023-08-21	2	91.071146	pull	
4	479	136	2023-03-02	10	331.557053	pull	

	Category
0	Toys
1	Toys
2	Toys
3	Toys
4	Toys

[35]:
```
print("Sales Data Shape:", sales_data.shape)
print("Product Info Shape:", product_info.shape)
print("Merged Data Shape:", merged_data.shape)
```

```
Sales Data Shape: (10000, 5)
```

```
Product Info Shape: (100, 3)
Merged Data Shape: (10000, 7)
```

[36]:
```python
# Check for missing values
print(merged_data.isnull().sum())
```

```
TransactionID    0
ProductID        0
Date             0
Quantity         0
Price            0
ProductName      0
Category         0
dtype: int64
```

[37]:
```python
merged_data['Date'] = pd.to_datetime(merged_data['Date'])
```

[47]:
```python
# Create 'TotalSale' column
merged_data['TotalSale'] = merged_data['Quantity'] * merged_data['Price']

# Subset for last quarter of the year
last_quarter_data = merged_data[merged_data['Date'].dt.month.isin([10, 11, 12])]

# Boolean indexing for a specific category, e.g., 'Electronics'
electronics_data = merged_data[merged_data['Category'] == 'Electronics']

# Transactions above the 75th percentile of 'TotalSale'
percentile_75 = np.percentile(merged_data['TotalSale'], 75)
high_value_sales_75 = merged_data[merged_data['TotalSale'] > percentile_75].
 ↪reset_index(drop = True)
```

[48]:
```python
high_value_sales_75.head()
```

[48]:
```
   TransactionID  ProductID       Date  Quantity       Price ProductName  \
0            479        136 2023-03-02        10  331.557053        pull
1            692        136 2023-09-28         8  494.070419        pull
2            879        136 2023-10-31        10  499.344566        pull
3            996        136 2024-01-14        10  444.985596        pull
4           1309        136 2023-03-31        10  423.493947        pull

  Category    TotalSale
0     Toys  3315.570534
1     Toys  3952.563354
2     Toys  4993.445658
3     Toys  4449.855959
4     Toys  4234.939470
```

[43]:
```
high_value_sales_75.shape
```

[43]:
```
(2500, 8)
```

[49]:
```
# Selecting specific rows and columns using loc and iloc
productID_101_data = merged_data.loc[merged_data['ProductID'] == 101,␣
↪['ProductName', 'TotalSale']]
every_10th_row = merged_data.iloc[::10, merged_data.columns.
↪get_indexer(['Date', 'Category'])]
```

[51]:
```
productID_101_data.shape
```

[51]:
```
(98, 2)
```

[52]:
```
productID_101_data.head()
```

[52]:

	ProductName	TotalSale
394	ready	212.490775
395	ready	1331.007870
396	ready	3311.017493
397	ready	1565.745895
398	ready	74.588211

[53]:
```
every_10th_row.shape
```

[53]:
```
(1000, 2)
```

[54]:
```
every_10th_row.head()
```

[54]:

	Date	Category
0	2023-03-13	Toys
10	2023-10-31	Toys
20	2023-04-21	Toys
30	2023-05-29	Toys
40	2023-12-18	Toys

[55]:
```
# Group by 'Category' and calculate total and average 'TotalSale'
grouped_data = merged_data.groupby('Category')['TotalSale'].agg(['sum',␣
↪'mean']).reset_index()

print(grouped_data.shape)
grouped_data.head()
```

```
(5, 3)
```

[55]:

	Category	sum	mean
0	Books	2.756942e+06	1405.169284

```
1         Clothing  2.547137e+06  1339.893113
2      Electronics  2.151251e+06  1468.430950
3  Home Appliances  3.339347e+06  1414.378361
4             Toys  3.320096e+06  1436.649185
```

[59]:
```
# Resample data on a monthly basis and calculate total 'Quantity', returning a␣
↪DataFrame
monthly_sales = merged_data.resample('M', on='Date').agg({'Quantity': 'sum'}).
↪reset_index()
```

[60]:
```
print(monthly_sales.shape)
monthly_sales.head()
```

```
(13, 2)
```

[60]:
```
        Date  Quantity
0 2023-01-31       902
1 2023-02-28      4175
2 2023-03-31      4874
3 2023-04-30      4375
4 2023-05-31      4851
```

0.3 Question 3

Zillow's marketplace offers a data-driven home valuation platform utilized by a diverse range of users including home buyers, sellers, renters, homeowners, real estate agents, mortgage providers, property managers, and landlords. The machine learning and data science team at Zillow employs various tools for predicting home valuations, such as Zestimate (Zillow Estimate), Zestimate Forecast, Zillow Home Value Index, Rent Zestimate, Zillow Rent Index, and the Pricing Tool.

0.3.1 Assignment Overview:

You are provided with a dataset named `zillow_feature_sample.csv`, containing various features relevant to Zillow's marketplace. Accompanying the dataset is a data dictionary titled `zillow_data_dictionary.xlsx`, which details the description of each column.

0.3.2 Tasks:

1. Develop a Missing Data Strategy: - Assess the `zillow_feature_sample.csv` dataset and devise a comprehensive strategy to handle missing data.

2. Quantitative Analysis of Missing Data: - Calculate and report the percentage of missing data in each feature of the dataset. - Analyze and infer the potential mechanism of missing data (e.g., Missing Completely at Random, Missing at Random, Missing Not at Random).

3. Imputation Strategy: - Propose and justify an imputation strategy for the missing values in the dataset. Your rationale should be data-driven and well-explained.

4. Open-Ended Exploration: - This question is open-ended, allowing you to explore other relevant aspects of the dataset. Conduct additional analyses or apply data processing techniques as appropriate.

0.3.3 Submission Guidelines:

- Document your analysis and findings in a clear and structured format.
- Ensure that your submission is thorough and well-reasoned.

[62]:
```
#!pip install ydata-profiling
```

[64]:
```
from ydata_profiling import ProfileReport
```

[65]:
```
zillow_data = pd.read_csv("zillow_feature_sample.csv")
```

[67]:
```
zillow_data.shape
```

[67]:
```
(10000, 58)
```

[68]:
```
zillow_data.head()
```

[68]:
```
   parcelid  airconditioningtypeid  architecturalstyletypeid  basementsqft  \
0  12833975                    NaN                       NaN           NaN
1  11070096                    1.0                       NaN           NaN
2  12752672                    1.0                       NaN           NaN
3  11338563                    NaN                       NaN           NaN
4  17098704                    NaN                       NaN           NaN

   bathroomcnt  bedroomcnt  buildingclasstypeid  buildingqualitytypeid  \
0          3.0         4.0                  NaN                    6.0
1          4.0         4.0                  NaN                    7.0
2          2.0         3.0                  NaN                    6.0
3          3.0         4.0                  NaN                    7.0
4          0.0         3.0                  NaN                    NaN

   calculatedbathnbr  decktypeid  …  numberofstories  fireplaceflag  \
0                3.0         NaN  …              NaN            NaN
1                4.0         NaN  …              NaN            NaN
2                2.0         NaN  …              NaN            NaN
3                3.0         NaN  …              NaN            NaN
4                NaN         NaN  …              1.0            NaN

   structuretaxvaluedollarcnt  taxvaluedollarcnt  assessmentyear  \
0                    155403.0           304592.0          2016.0
1                    493070.0           821783.0          2016.0
2                    126695.0           247962.0          2016.0
3                    130500.0           308900.0          2016.0
4                    142271.0           223101.0          2016.0

   landtaxvaluedollarcnt  taxamount  taxdelinquencyflag  taxdelinquencyyear  \
```

```
0              149189.0    3708.29                NaN                 NaN
1              328713.0   10087.59                NaN                 NaN
2              121267.0    3377.86                NaN                 NaN
3              178400.0    3578.92                NaN                 NaN
4               80830.0    2564.86                NaN                 NaN

   censustractandblock
0         6.037409e+13
1         6.037108e+13
2         6.037504e+13
3         6.037920e+13
4         6.111000e+13

[5 rows x 58 columns]
```

[70]:

```
zillow_profile = ProfileReport(zillow_data, title="Zillow Profiling Report")
zillow_profile.to_file("zillow_profile_report.html")
```

```
/Users/wale/anaconda3/lib/python3.10/site-
packages/ydata_profiling/profile_report.py:354: UserWarning: Try running
command: 'pip install --upgrade Pillow' to avoid ValueError
  warnings.warn(
Summarize dataset:  92%|
                                             |
60/65 [00:00<00:00, 49.32it/s, Calculate auto
correlation]/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
```

```
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
/Users/wale/anaconda3/lib/python3.10/site-
packages/scipy/stats/_stats_py.py:4921: ConstantInputWarning: An input array is
```

```
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(warn_msg))
Summarize dataset: 100%|

    | 1364/1364 [01:14<00:00, 18.28it/s, Completed]
Generate report structure: 100%|

             | 1/1 [00:06<00:00,  6.21s/it]
Render HTML: 100%|

             | 1/1 [00:10<00:00, 10.95s/it]
Export report to file: 100%|

             | 1/1 [00:00<00:00, 11.84it/s]
```

[ ]:

```

```