ANZ Virtual Internship

September 6, 2021

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[3]:
```
import os
```

[4]:
```
os.getcwd()
```

[4]:
```
'C:\\Users\\godar'
```

[5]:
```
os.chdir('C:\\Users\\godar\\Desktop\\Power Bi\\ANZ')
```

[6]:
```
os.getcwd()
```

[6]:
```
'C:\\Users\\godar\\Desktop\\Power Bi\\ANZ'
```

[7]:
```
Data =pd.read_excel("ANZ synthesised transaction dataset.xlsx")
```

[8]:
```
Data.head()
```

[8]:
```
       status  card_present_flag bpay_biller_code         account currency  \
0  authorized                1.0              NaN  ACC-1598451071      AUD
1  authorized                0.0              NaN  ACC-1598451071      AUD
2  authorized                1.0              NaN  ACC-1222300524      AUD
3  authorized                1.0              NaN  ACC-1037050564      AUD
4  authorized                1.0              NaN  ACC-1598451071      AUD

        long_lat txn_description                            merchant_id  \
0  153.41 -27.95             POS  81c48296-73be-44a7-befa-d053f48ce7cd
1  153.41 -27.95       SALES-POS  830a451c-316e-4a6a-bf25-e37caedca49e
2  151.23 -33.94             POS  835c231d-8cdf-4e96-859d-e9d571760cf0
3  153.10 -27.66       SALES-POS  48514682-c78a-4a88-b0da-2d6302e64673
4  153.41 -27.95       SALES-POS  b4e02c10-0852-4273-b8fd-7b3395e32eb0

   merchant_code first_name  …  age merchant_suburb merchant_state  \
0            NaN      Diana  …   26         Ashmore            QLD
1            NaN      Diana  …   26          Sydney            NSW
```

```
2            NaN    Michael  …   38          Sydney            NSW
3            NaN     Rhonda  …   40         Buderim            QLD
4            NaN      Diana  …   26   Mermaid Beach            QLD

                    extraction amount                   transaction_id  \
0  2018-08-01T01:01:15.000+0000  16.25  a623070bfead4541a6b0fff8a09e706c
1  2018-08-01T01:13:45.000+0000  14.19  13270a2a902145da9db4c951e04b51b9
2  2018-08-01T01:26:15.000+0000   6.42  feb79e7ecd7048a5a36ec889d1a94270
3  2018-08-01T01:38:45.000+0000  40.90  2698170da3704fd981b15e64a006079e
4  2018-08-01T01:51:15.000+0000   3.25  329adf79878c4cf0aeb4188b4691c266

     country     customer_id merchant_long_lat movement
0  Australia  CUS-2487424745     153.38 -27.99    debit
1  Australia  CUS-2487424745     151.21 -33.87    debit
2  Australia  CUS-2142601169     151.21 -33.87    debit
3  Australia  CUS-1614226872     153.05 -26.68    debit
4  Australia  CUS-2487424745     153.44 -28.06    debit

[5 rows x 23 columns]
```

[10]:
```
pd.DataFrame({"columns": Data.columns})
```

[10]:
```
              columns
0              status
1   card_present_flag
2    bpay_biller_code
3             account
4            currency
5            long_lat
6     txn_description
7         merchant_id
8       merchant_code
9          first_name
10            balance
11               date
12             gender
13                age
14    merchant_suburb
15     merchant_state
16         extraction
17             amount
18     transaction_id
19            country
20        customer_id
21  merchant_long_lat
22           movement
```

[11]:
```
print("Data shape" , Data.shape)
```

```
Data shape (12043, 23)
```

[12]:
```
print("Number of unique customer ID's:", Data.customer_id.nunique())
```

```
Number of unique customer ID's: 100
```

[13]:
```
print("Number of rows in dataset", len(Data))
print("Number of unique customer ID's:", Data.customer_id.nunique())
```

```
Number of rows in dataset 12043
Number of unique customer ID's: 100
```

[14]:
```
Data.date.describe()
```

[14]:
```
count                   12043
unique                     91
top       2018-09-28 00:00:00
freq                      174
first     2018-08-01 00:00:00
last      2018-10-31 00:00:00
Name: date, dtype: object
```

[15]:
```
pd.date_range(start ="2018-08-01", end ="2018-10-31").difference(Data.date)
```

[15]:
```
DatetimeIndex(['2018-08-16'], dtype='datetime64[ns]', freq=None)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12043 entries, 0 to 12042
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   status              12043 non-null  object
 1   card_present_flag   7717 non-null   float64
 2   bpay_biller_code    885 non-null    object
 3   account             12043 non-null  object
 4   currency            12043 non-null  object
 5   long_lat            12043 non-null  object
 6   txn_description     12043 non-null  object
 7   merchant_id         7717 non-null   object
 8   merchant_code       883 non-null    float64
 9   first_name          12043 non-null  object
 10  balance             12043 non-null  float64
 11  date                12043 non-null  datetime64[ns]
 12  gender              12043 non-null  object
 13  age                 12043 non-null  int64
 14  merchant_suburb     7717 non-null   object
```

```
 15  merchant_state     7717 non-null   object
 16  extraction         12043 non-null  object
 17  amount             12043 non-null  float64
 18  transaction_id     12043 non-null  object
 19  country            12043 non-null  object
 20  customer_id        12043 non-null  object
 21  merchant_long_lat  7717 non-null   object
 22  movement           12043 non-null  object
dtypes: datetime64[ns](1), float64(4), int64(1), object(17)
memory usage: 2.1+ MB
```

[17]:
```
missing =Data.isnull().sum()
missing=missing[missing>0]
missing_percentange =round(missing/len(Data),3)*100
pd.DataFrame({"Number of missing values": missing,"Percentage":
 ↪missing_percentange }).sort_values(by ="Percentage",

 ↪                  ascending = False)
```

[17]:

	Number of missing values	Percentage
bpay_biller_code	11158	92.7
merchant_code	11160	92.7
card_present_flag	4326	35.9
merchant_id	4326	35.9
merchant_suburb	4326	35.9
merchant_state	4326	35.9
merchant_long_lat	4326	35.9

[18]:
```
Data.describe()
```

[18]:

	card_present_flag	merchant_code	balance	age	\
count	7717.000000	883.0	12043.000000	12043.000000	
mean	0.802644	0.0	14704.195553	30.582330	
std	0.398029	0.0	31503.722652	10.046343	
min	0.000000	0.0	0.240000	18.000000	
25%	1.000000	0.0	3158.585000	22.000000	
50%	1.000000	0.0	6432.010000	28.000000	
75%	1.000000	0.0	12465.945000	38.000000	
max	1.000000	0.0	267128.520000	78.000000	

	amount
count	12043.000000
mean	187.933588
std	592.599934
min	0.100000
25%	16.000000

```
50%       29.000000
75%       53.655000
max     8835.980000
```

1 Exploratory data analysis(EDA)

[19]:
```
Data.status.value_counts(dropna=False)
```

[19]:
```
authorized    7717
posted        4326
Name: status, dtype: int64
```

[20]:
```
Data.card_present_flag.value_counts(dropna=False)
```

[20]:
```
1.0    6194
NaN    4326
0.0    1523
Name: card_present_flag, dtype: int64
```

[21]:
```
Data.currency.value_counts(dropna=False)
```

[21]:
```
AUD    12043
Name: currency, dtype: int64
```

[22]:
```
Data.long_lat.head()
```

[22]:
```
0    153.41 -27.95
1    153.41 -27.95
2    151.23 -33.94
3    153.10 -27.66
4    153.41 -27.95
Name: long_lat, dtype: object
```

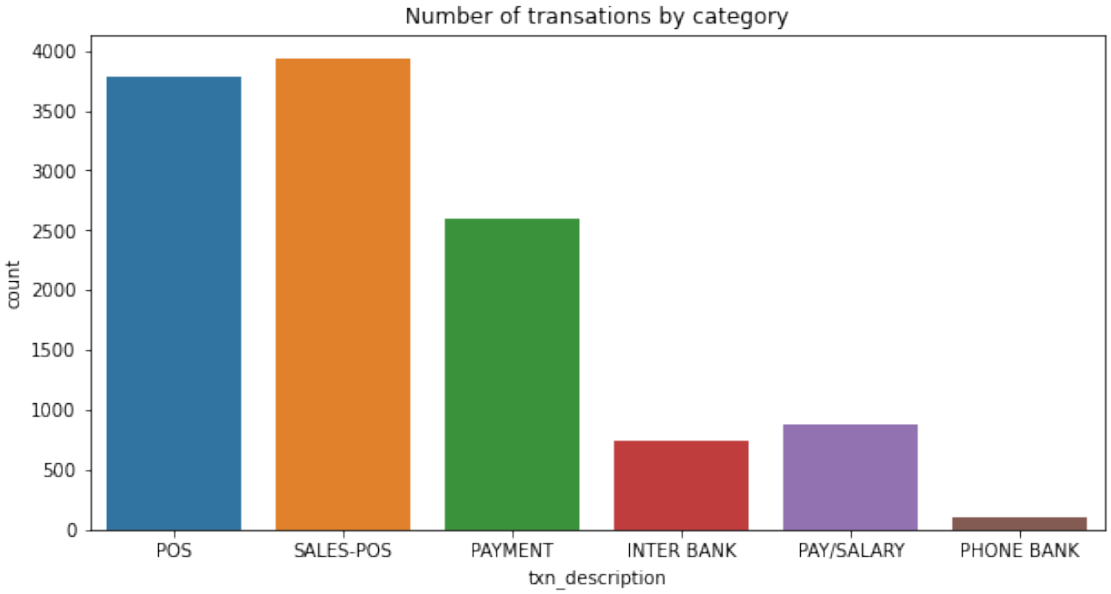
2 Transation description(types of transactions)

[23]:
```
Data.txn_description.value_counts(dropna=False)
```

[23]:
```
SALES-POS     3934
POS           3783
PAYMENT       2600
PAY/SALARY     883
INTER BANK     742
PHONE BANK     101
Name: txn_description, dtype: int64
```

[24]:
```
plt.figure(figsize=(10,5))
sns.countplot(Data.txn_description)
plt.title("Number of transations by category")
```

[24]:
```
Text(0.5, 1.0, 'Number of transations by category')
```



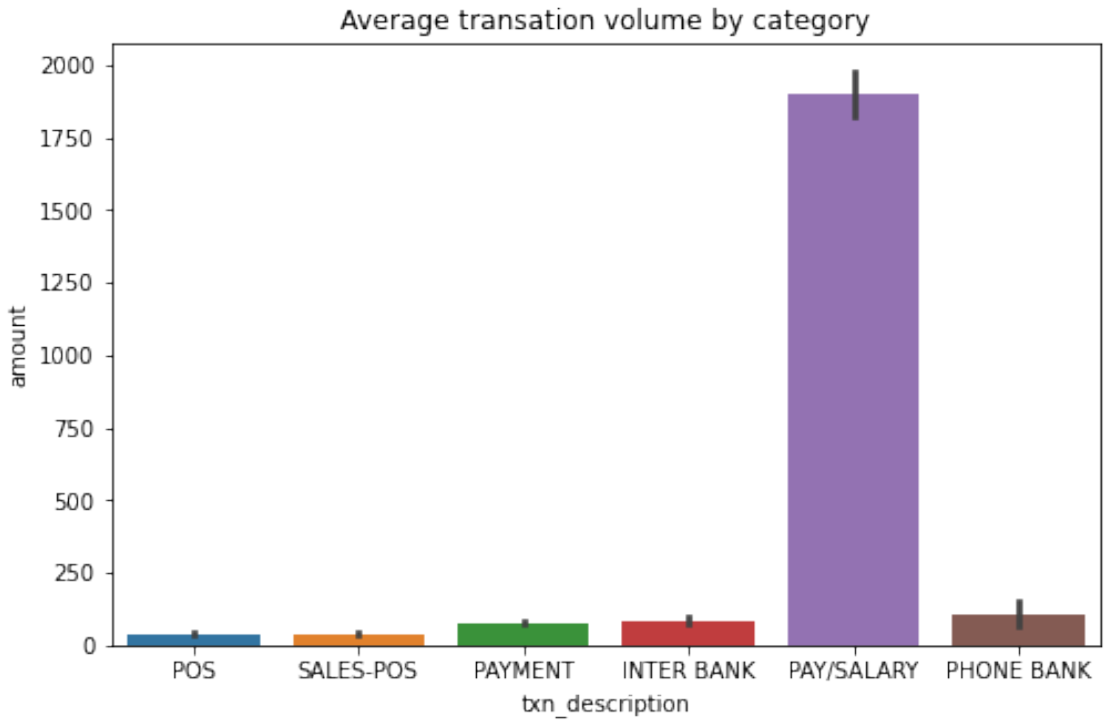
[25]:
```
Data[["txn_description","amount"]].groupby("txn_description", as_index=False).
↪mean().sort_values(by="amount",ascending=False)
```

[25]:

	txn_description	amount
1	PAY/SALARY	1898.728029
3	PHONE BANK	106.099010
0	INTER BANK	86.699461
2	PAYMENT	77.613077
4	POS	40.407412
5	SALES-POS	39.909789

[26]:
```
plt.figure(figsize=(8,5))
sns.barplot(x="txn_description",y="amount",data=Data)
plt.title("Average transation volume by category")
```

[26]:
```
Text(0.5, 1.0, 'Average transation volume by category')
```

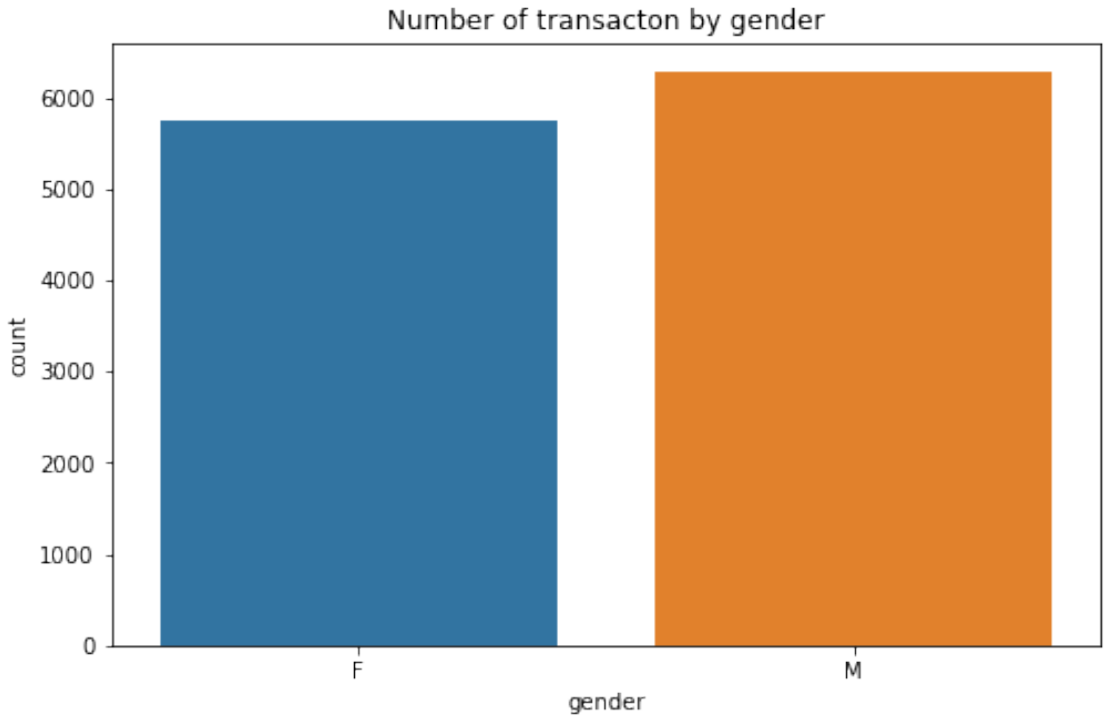


[27]:
```
Data.gender.value_counts(dropna=False)
```

[27]:
```
M    6285
F    5758
Name: gender, dtype: int64
```

[28]:
```
plt.figure(figsize=(8,5))
sns.countplot( Data.gender)
plt.title("Number of transacton by gender")
```

[28]:
```
Text(0.5, 1.0, 'Number of transacton by gender')
```



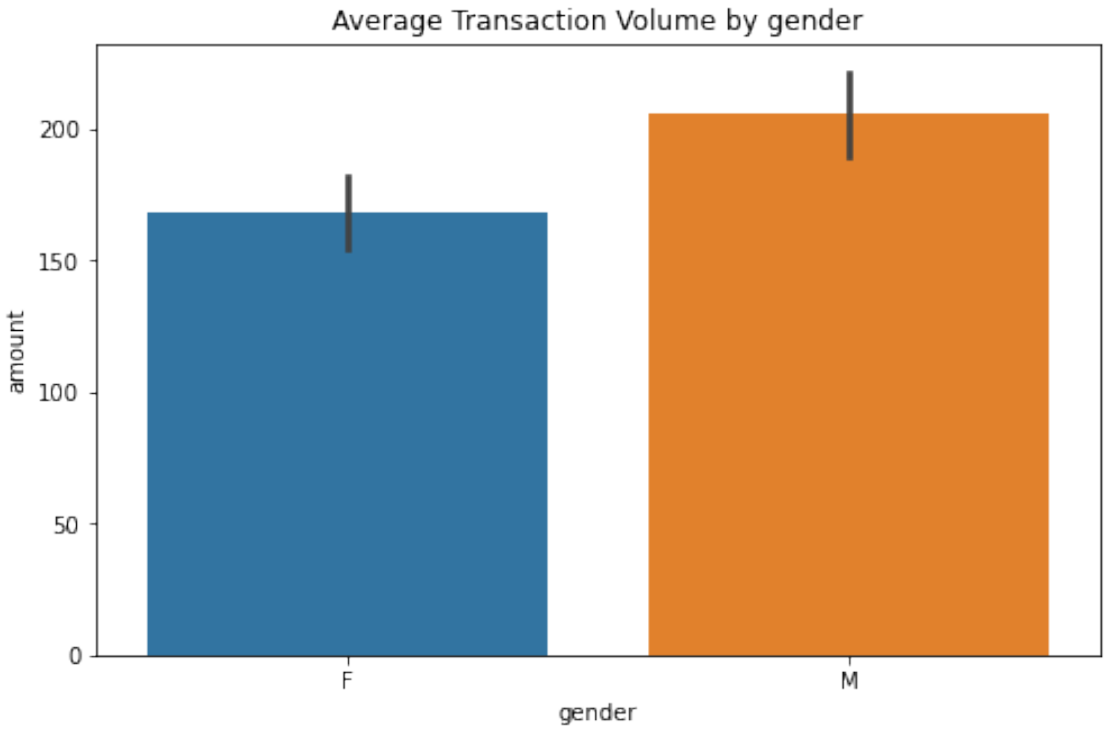
[29]:
```
Data[["gender","amount"]].groupby("gender", as_index=False).mean().
↪sort_values(by="amount",ascending=False)
```

[29]:

	gender	amount
1	M	205.721809
0	F	168.517303

[30]:
```
plt.figure(figsize=(8,5))
sns.barplot(x="gender", y="amount",data=Data)
plt.title("Average Transaction Volume by gender")
```

[30]:
```
Text(0.5, 1.0, 'Average Transaction Volume by gender')
```



[31]:
```
Data.merchant_suburb.value_counts(dropna=False)
```

[31]:
```
NaN              4326
Melbourne         255
Sydney            233
Southport          82
Brisbane City      79
                 ...
Grafton             1
Dowsing Point       1
East Toowoomba      1
Russell             1
Arndell Park        1
Name: merchant_suburb, Length: 1610, dtype: int64
```

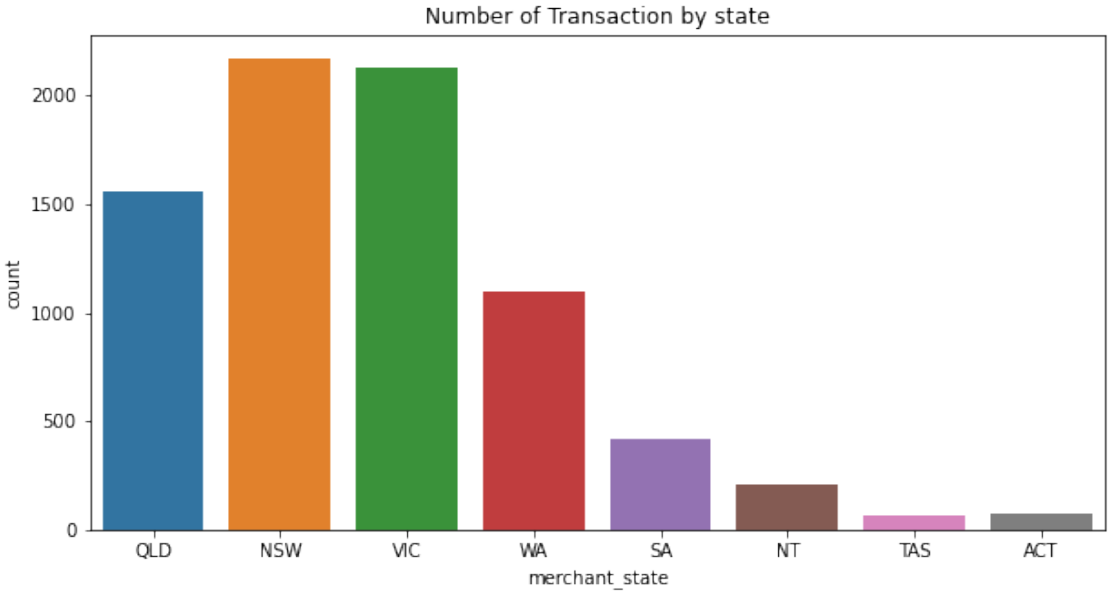
[32]:
```
Data.merchant_state.value_counts(dropna=False)
```

[32]:
```
NaN    4326
NSW    2169
VIC    2131
QLD    1556
WA     1100
SA      415
```

```
NT      205
ACT      73
TAS      68
Name: merchant_state, dtype: int64
```

[33]:
```
plt.figure(figsize=(10,5))
sns.countplot(Data.merchant_state)
plt.title("Number of Transaction by state")
```

[33]:
```
Text(0.5, 1.0, 'Number of Transaction by state')
```



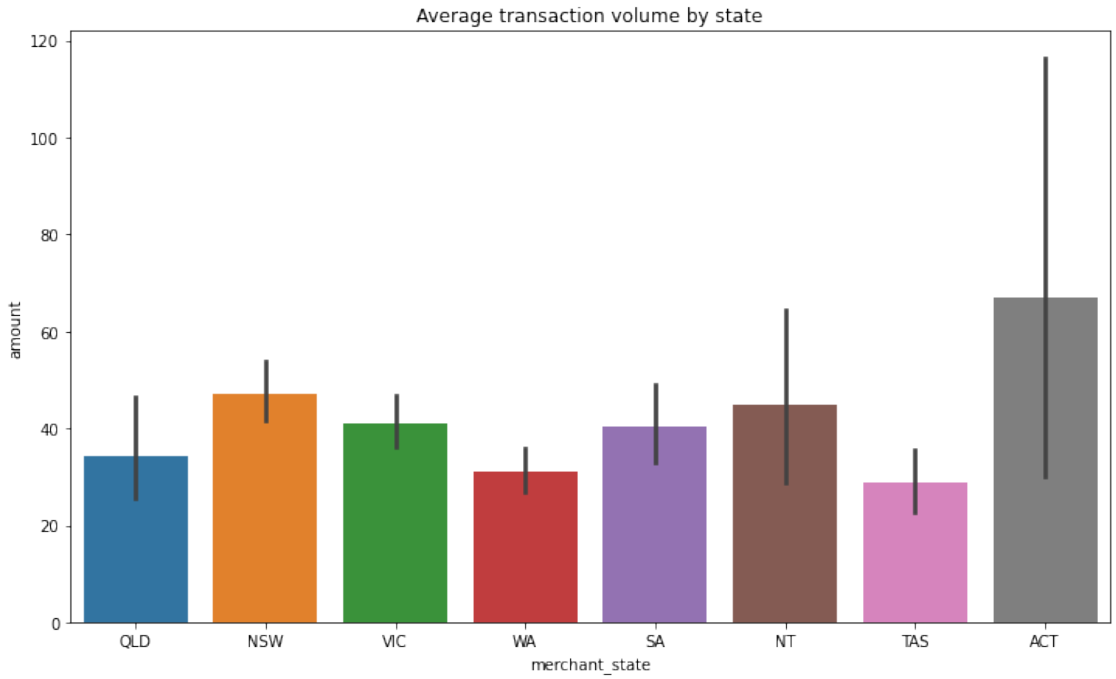
[34]:
```
Data[["merchant_state","amount"]].groupby("merchant_state", as_index=False).
↪mean().sort_values(by="amount",ascending=False)
```

[34]:

	merchant_state	amount
0	ACT	66.803836
1	NSW	47.036316
2	NT	44.726293
6	VIC	41.099953
4	SA	40.425470
3	QLD	34.372397
7	WA	30.901873
5	TAS	28.866618

[35]:
```
plt.figure(figsize=(12,7))
sns.barplot(x ="merchant_state", y="amount", data=Data)
plt.title("Average transaction volume by state")
```

[35]: Text(0.5, 1.0, 'Average transaction volume by state')



[36]:
```
Data.extraction.head()
```

[36]:
```
0    2018-08-01T01:01:15.000+0000
1    2018-08-01T01:13:45.000+0000
2    2018-08-01T01:26:15.000+0000
3    2018-08-01T01:38:45.000+0000
4    2018-08-01T01:51:15.000+0000
Name: extraction, dtype: object
```

[37]:
```
Data.country.value_counts(dropna=False)
```

[37]:
```
Australia    12043
Name: country, dtype: int64
```

[38]:
```
Data.merchant_long_lat.head()
```

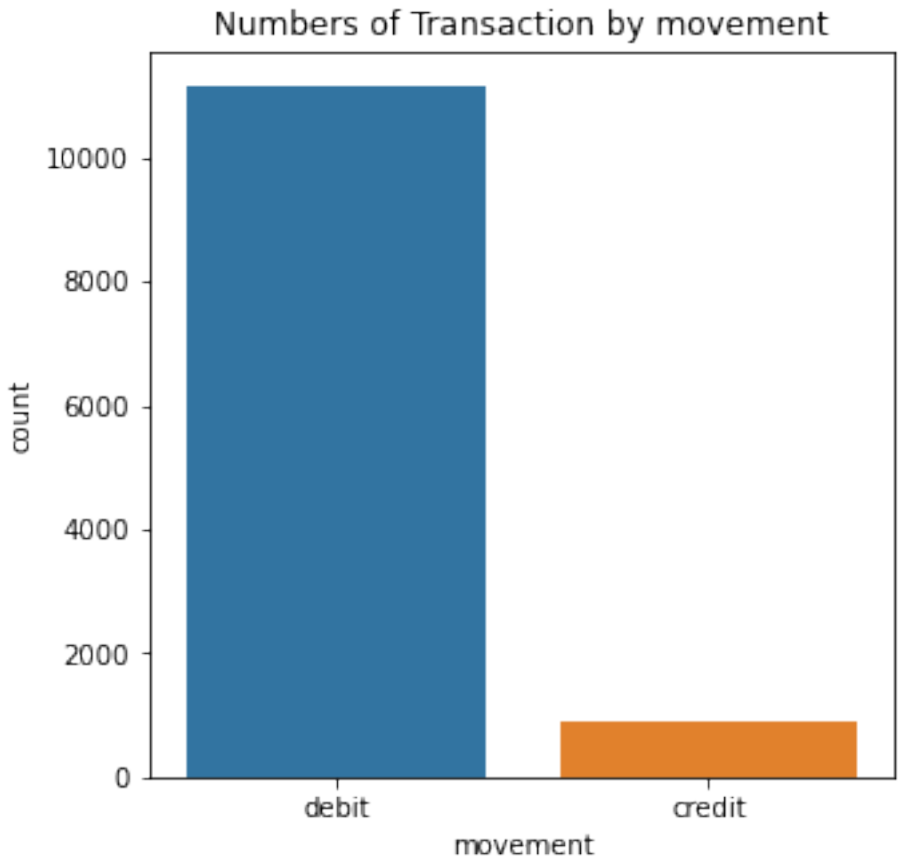
[38]:
```
0    153.38 -27.99
1    151.21 -33.87
2    151.21 -33.87
3    153.05 -26.68
4    153.44 -28.06
Name: merchant_long_lat, dtype: object
```

[39]:
```
Data.movement.value_counts(dropna=False)
```

[39]:
```
debit     11160
credit      883
Name: movement, dtype: int64
```

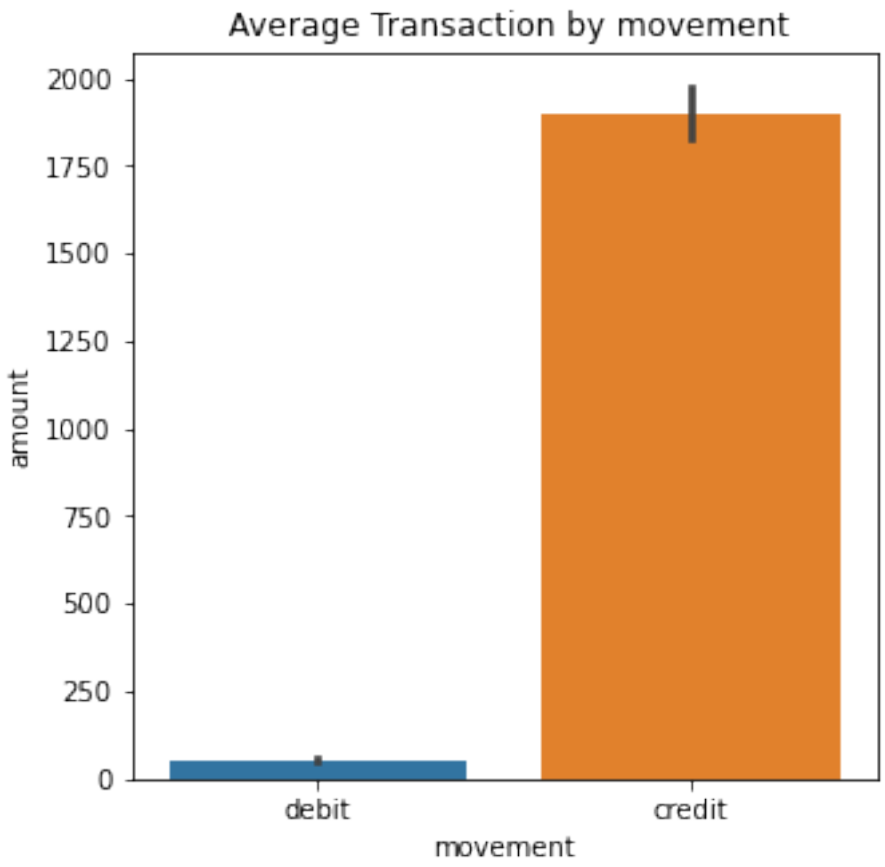
[40]:
```
plt.figure(figsize=(5,5))
sns.countplot("movement",data=Data)
plt.title("Numbers of Transaction by movement")
```

[40]:
```
Text(0.5, 1.0, 'Numbers of Transaction by movement')
```



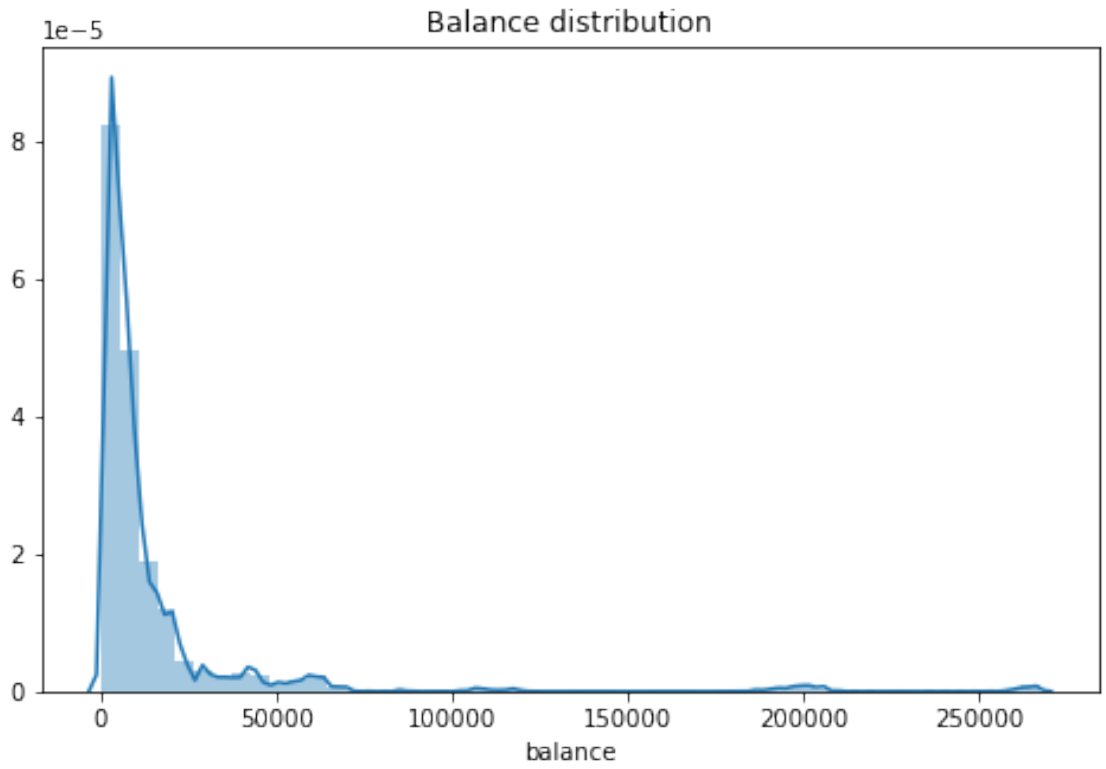
[41]:
```
plt.figure(figsize=(5,5))
sns.barplot(x= "movement", y="amount", data= Data)
plt.title(" Average Transaction by movement")
```

[41]:
```
Text(0.5, 1.0, ' Average Transaction by movement')
```



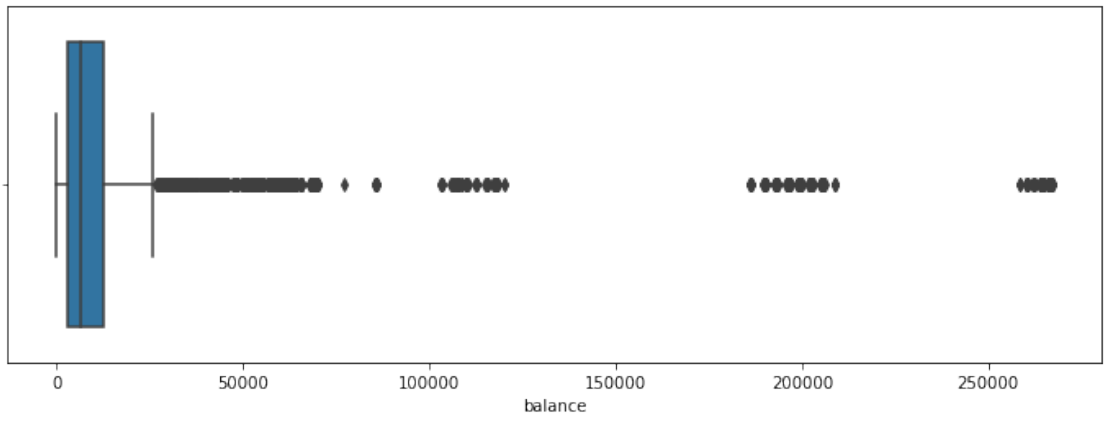
[42]:
```
plt.figure(figsize=(8,5))
sns.distplot(Data.balance)
plt.title("Balance distribution")
```

[42]: Text(0.5, 1.0, 'Balance distribution')



[43]:
```
plt.figure(figsize=(12,4))
sns.boxplot(Data.balance)
```

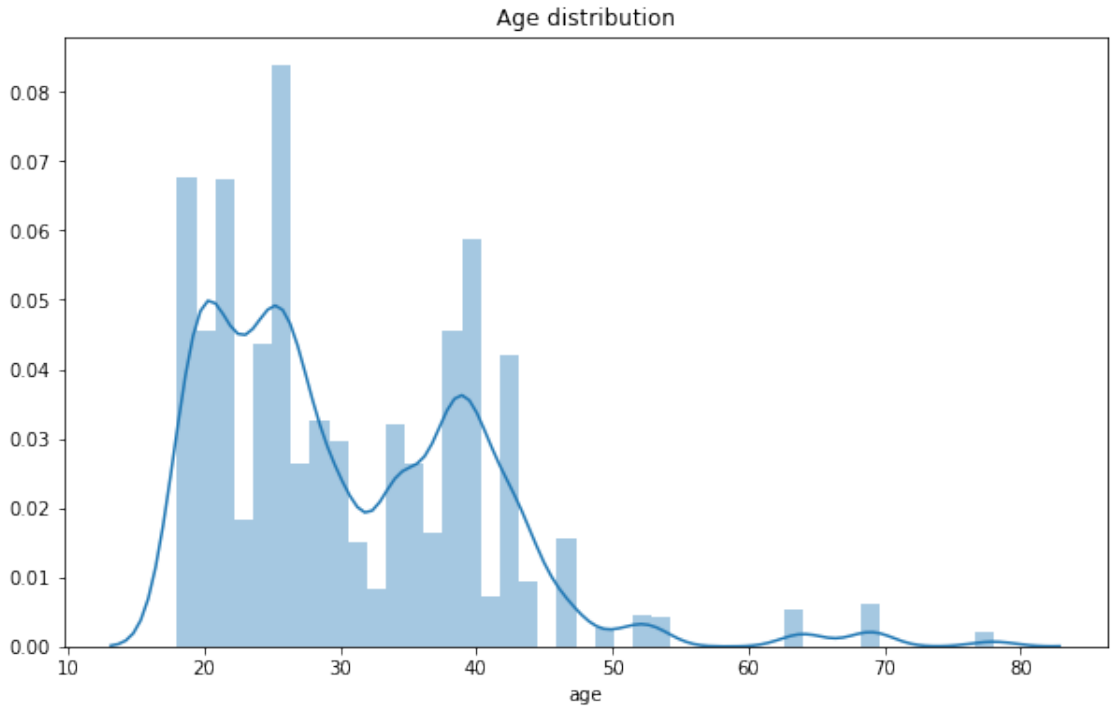
[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1f78ee953d0>



[44]:
```
plt.figure(figsize=(10,6))
sns.distplot(Data.age)
```

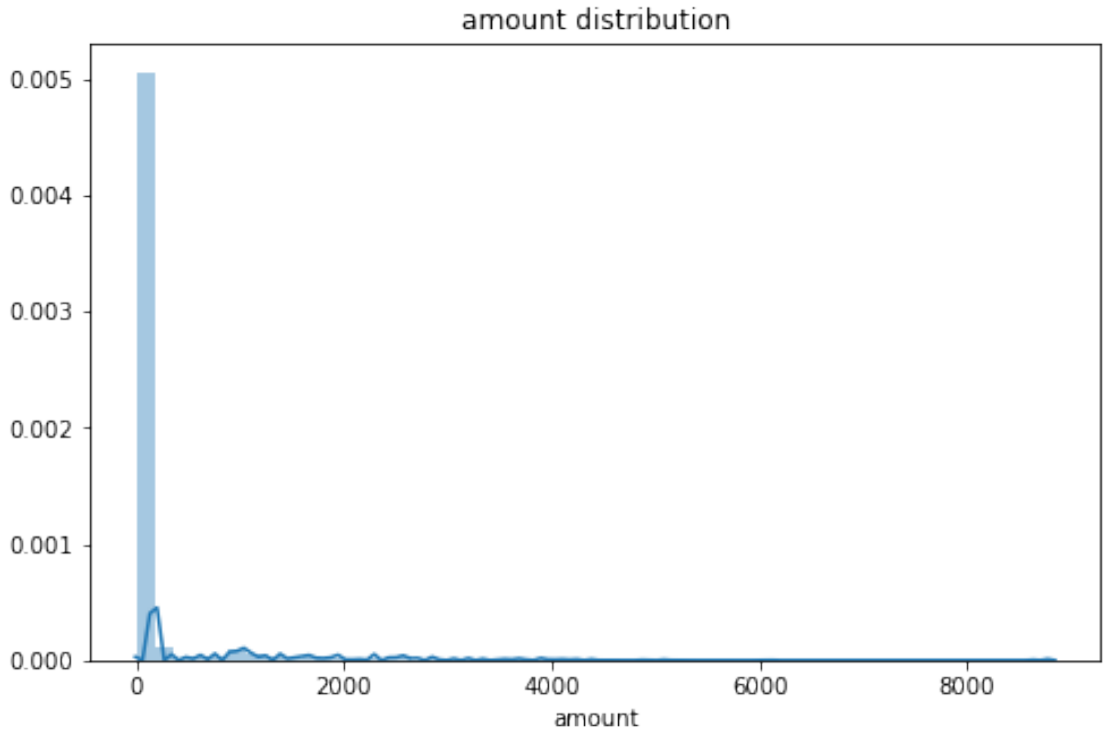
```
plt.title("Age distribution")
```

[44]: Text(0.5, 1.0, 'Age distribution')



[45]:
```
plt.figure(figsize=(8,5))
sns.distplot(Data.amount)
plt.title("amount distribution")
```

[45]: Text(0.5, 1.0, 'amount distribution')



3 Drop unwanted colums

[46]:
```
print("After:",Data.shape)
```

```
After: (12043, 23)
```

[47]:
```
missing =Data.isnull().sum()
missing=missing[missing>0]
missing_percentange =round(missing/len(Data),3)*100
pd.DataFrame({"Number of missing values": missing,"Percentage":
 ↪missing_percentange }).sort_values(by ="Percentage",

 ↪                  ascending = False)
```

[47]:

	Number of missing values	Percentage
bpay_biller_code	11158	92.7
merchant_code	11160	92.7
card_present_flag	4326	35.9
merchant_id	4326	35.9
merchant_suburb	4326	35.9
merchant_state	4326	35.9
merchant_long_lat	4326	35.9

[48]:
```
nonSales = Data.loc[(Data.txn_description != "SALES-POS") | (Data.
 ↪txn_description != "POS"), :]
nonSales.isnull().sum().sort_values(ascending = False)
```

[48]:
```
merchant_code        11160
bpay_biller_code     11158
card_present_flag     4326
merchant_state        4326
merchant_suburb       4326
merchant_id           4326
merchant_long_lat     4326
movement                 0
first_name               0
account                  0
currency                 0
long_lat                 0
txn_description          0
date                     0
balance                  0
gender                   0
age                      0
extraction               0
amount                   0
transaction_id           0
country                  0
customer_id              0
status                   0
dtype: int64
```

[49]:
```
cols = ["card_present_flag", "merchant_state", "merchant_suburb",␣
 ↪"merchant_id", "merchant_long_lat"]
for col in cols:
    Data[col].fillna("n/a", inplace = True)
```

[50]:
```
missing = Data.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(ascending = False)
```

[50]:
```
merchant_code       11160
bpay_biller_code    11158
dtype: int64
```

[51]:
```
Data = Data.drop(["merchant_code", "bpay_biller_code"], axis = 1)
```

[52]:
```
Data.isnull().sum().max()
```

[52]:
```
0
```

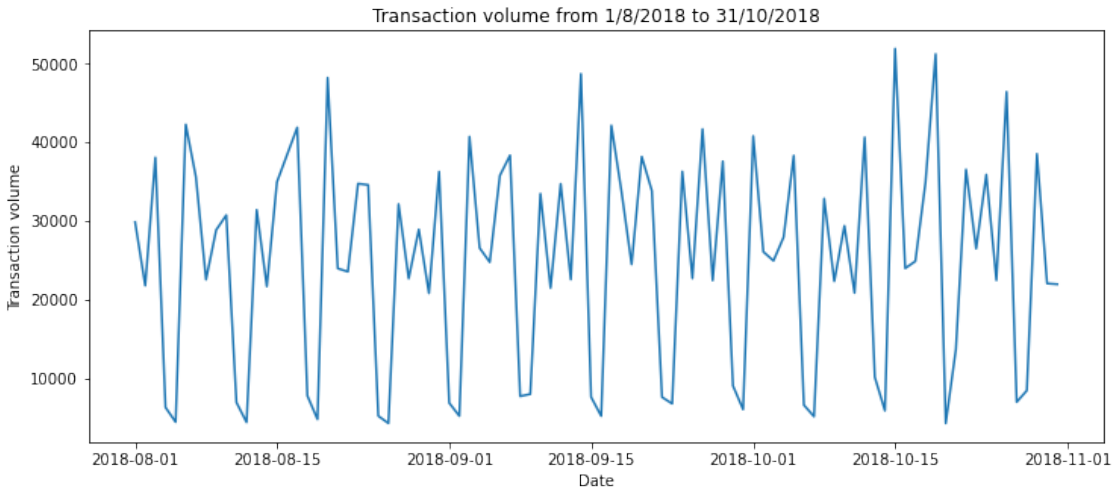
4 Create feature for month, dayofweek and hour

[53]:
```
daily_amount = pd.DataFrame(Data.groupby("date").amount.sum())
daily_amount.head()
```

[53]:
```
              amount
date
2018-08-01  29867.94
2018-08-02  21786.32
2018-08-03  38096.58
2018-08-04   6296.05
2018-08-05   4426.50
```

[54]:
```
fig, ax = plt.subplots(figsize = (12, 5))
ax.plot(daily_amount.index, daily_amount.amount)
plt.title("Transaction volume from 1/8/2018 to 31/10/2018")
plt.xlabel("Date")
plt.ylabel("Transaction volume")
```

[54]:
```
Text(0, 0.5, 'Transaction volume')
```



[55]:
```
Data["month"] = pd.DatetimeIndex(Data.date).month
Data["dayofweek"] = pd.DatetimeIndex(Data.date).dayofweek
Data[["date", "month", "dayofweek"]].head()
```

[55]:
```
        date  month  dayofweek
0 2018-08-01      8          2
1 2018-08-01      8          2
2 2018-08-01      8          2
3 2018-08-01      8          2
```

```
4 2018-08-01      8          2
```

[56]:
```
Data.extraction.head()
```

[56]:
```
0    2018-08-01T01:01:15.000+0000
1    2018-08-01T01:13:45.000+0000
2    2018-08-01T01:26:15.000+0000
3    2018-08-01T01:38:45.000+0000
4    2018-08-01T01:51:15.000+0000
Name: extraction, dtype: object
```

[57]:
```
Data["extraction"] = [timestamp.split("T")[1].split(".")[0] for timestamp in␣
 ↪Data.extraction]
Data.extraction.head()
```

[57]:
```
0    01:01:15
1    01:13:45
2    01:26:15
3    01:38:45
4    01:51:15
Name: extraction, dtype: object
```

[58]:
```
Data["hour"] = [time.split(":")[0] for time in Data.extraction]
Data[["extraction", "hour"]].head()
```

[58]:
```
  extraction hour
0   01:01:15   01
1   01:13:45   01
2   01:26:15   01
3   01:38:45   01
4   01:51:15   01
```

[59]:
```
print("Before: ", Data.hour.dtype)
Data["hour"] = pd.to_numeric(Data.hour)
print("After: ", Data.hour.dtype)
```

```
Before:  object
After:  int64
```

[60]:
```
Data.head()
```

[60]:
```
       status card_present_flag         account currency       long_lat  \
0  authorized                 1  ACC-1598451071      AUD  153.41 -27.95
1  authorized                 0  ACC-1598451071      AUD  153.41 -27.95
2  authorized                 1  ACC-1222300524      AUD  151.23 -33.94
3  authorized                 1  ACC-1037050564      AUD  153.10 -27.66
4  authorized                 1  ACC-1598451071      AUD  153.41 -27.95
```

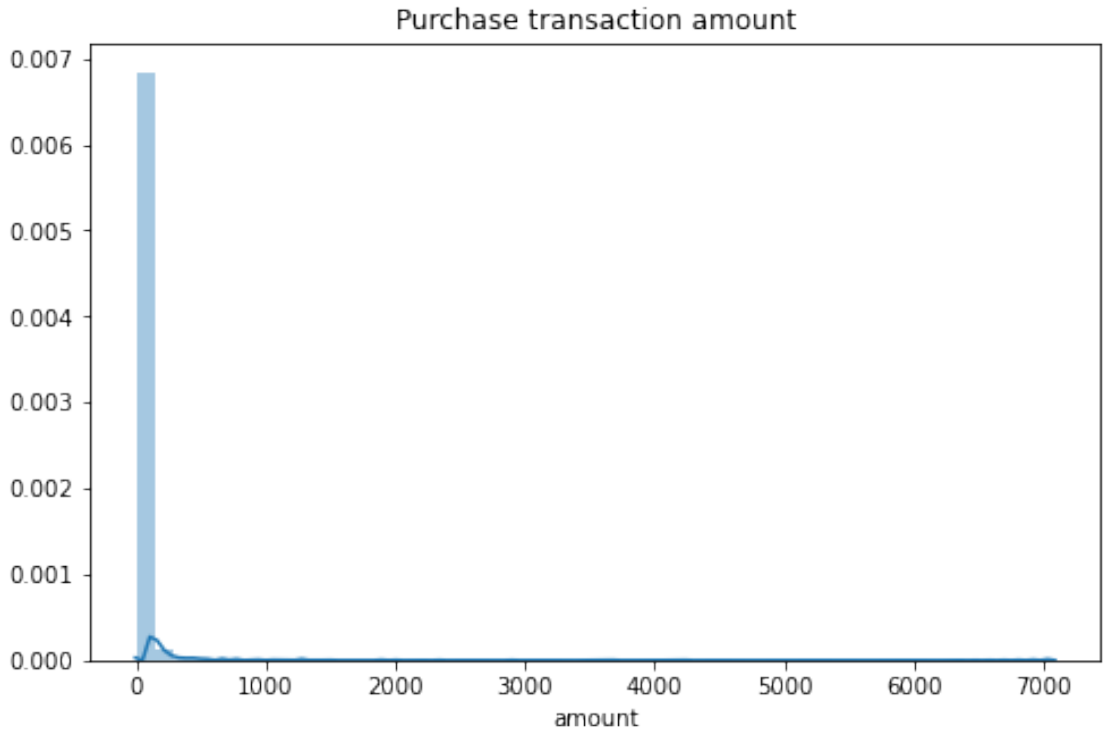
```
  txn_description                             merchant_id first_name  balance  \
0             POS  81c48296-73be-44a7-befa-d053f48ce7cd      Diana    35.39
1       SALES-POS  830a451c-316e-4a6a-bf25-e37caedca49e      Diana    21.20
2             POS  835c231d-8cdf-4e96-859d-e9d571760cf0    Michael     5.71
3       SALES-POS  48514682-c78a-4a88-b0da-2d6302e64673     Rhonda  2117.22
4       SALES-POS  b4e02c10-0852-4273-b8fd-7b3395e32eb0      Diana    17.95

        date  … extraction  amount                    transaction_id  \
0 2018-08-01  …    01:01:15   16.25  a623070bfead4541a6b0fff8a09e706c
1 2018-08-01  …    01:13:45   14.19  13270a2a902145da9db4c951e04b51b9
2 2018-08-01  …    01:26:15    6.42  feb79e7ecd7048a5a36ec889d1a94270
3 2018-08-01  …    01:38:45   40.90  2698170da3704fd981b15e64a006079e
4 2018-08-01  …    01:51:15    3.25  329adf79878c4cf0aeb4188b4691c266

     country     customer_id  merchant_long_lat movement month dayofweek hour
0  Australia  CUS-2487424745      153.38 -27.99    debit     8         2    1
1  Australia  CUS-2487424745      151.21 -33.87    debit     8         2    1
2  Australia  CUS-2142601169      151.21 -33.87    debit     8         2    1
3  Australia  CUS-1614226872      153.05 -26.68    debit     8         2    1
4  Australia  CUS-2487424745      153.44 -28.06    debit     8         2    1

[5 rows x 24 columns]
```

[61]:
```
purchases_amount = Data.loc[(Data.txn_description == "POS") | (Data.
 ↪txn_description == "SALES-POS"), "amount"]
purchases_amount.head()
```

[61]:
```
0    16.25
1    14.19
2     6.42
3    40.90
4     3.25
Name: amount, dtype: float64
```

[62]:
```
plt.figure(figsize = (8, 5))
sns.distplot(purchases_amount)
plt.title("Purchase transaction amount")
```

[62]:
```
Text(0.5, 1.0, 'Purchase transaction amount')
```

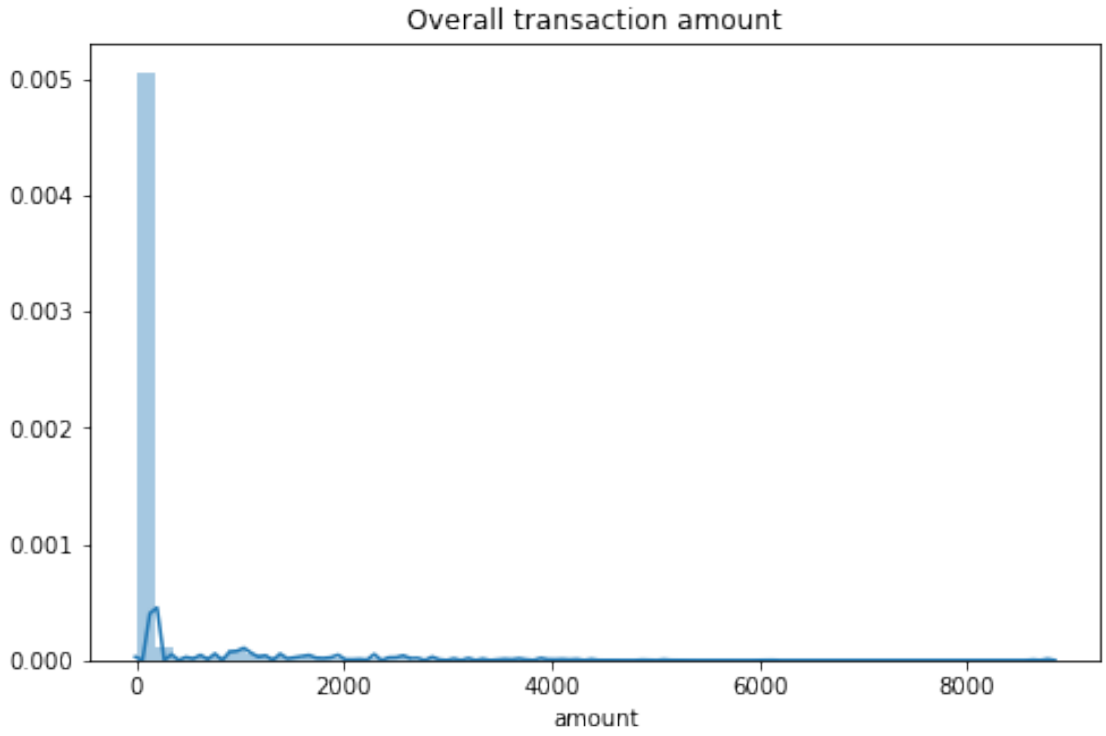


[63]:
```
purchases_amount.describe()
```

[63]:
```
count    7717.000000
mean       40.153732
std       149.833070
min         0.100000
25%        12.080000
50%        19.700000
75%        33.910000
max      7081.090000
Name: amount, dtype: float64
```

[64]:
```
plt.figure(figsize = (8, 5))
sns.distplot(Data.amount)
plt.title("Overall transaction amount")
```

[64]:
```
Text(0.5, 1.0, 'Overall transaction amount')
```



[65]:
```
Data.amount.describe()
```

[65]:
```
count    12043.000000
mean       187.933588
std        592.599934
min          0.100000
25%         16.000000
50%         29.000000
75%         53.655000
max       8835.980000
Name: amount, dtype: float64
```

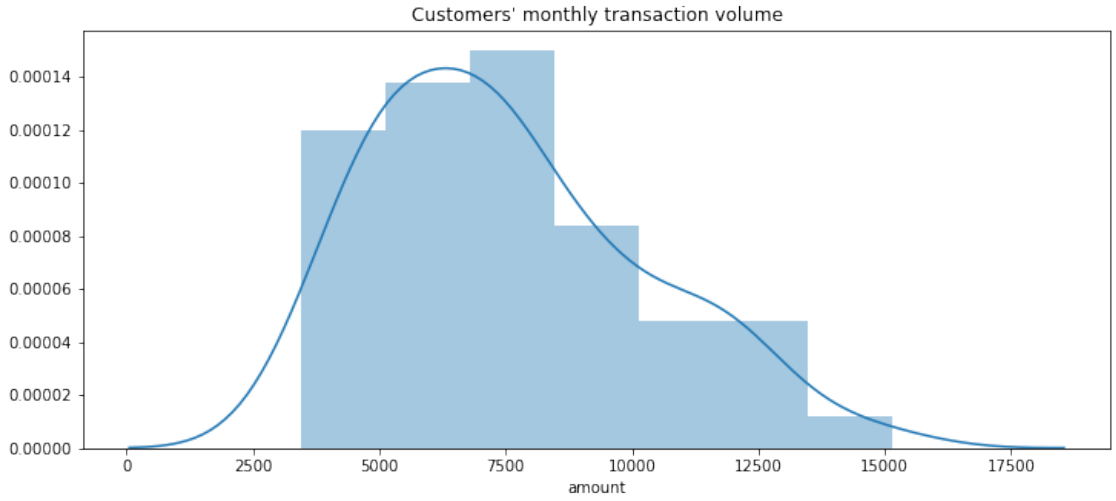
[66]:
```
customer_monthly_volume = pd.DataFrame(Data.groupby("customer_id").amount.sum()␣
↪/ 3)
customer_monthly_volume.head()
```

[66]:

customer_id	amount
CUS-1005756958	5422.990000
CUS-1117979751	11328.123333
CUS-1140341822	5670.200000
CUS-1147642491	9660.273333
CUS-1196156254	12016.906667

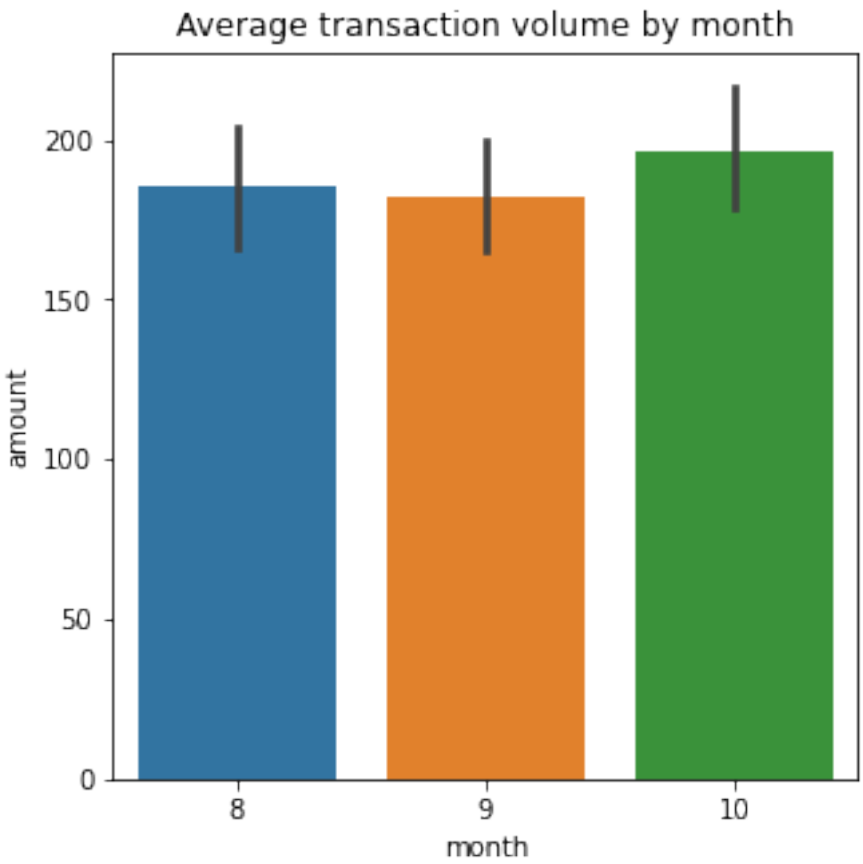
[67]:
```
plt.figure(figsize = (12, 5))
sns.distplot(customer_monthly_volume.amount)
plt.title("Customers' monthly transaction volume")
```

[67]:
```
Text(0.5, 1.0, "Customers' monthly transaction volume")
```



[68]:
```
plt.figure(figsize = (5, 5))
sns.barplot(x = "month", y = "amount", data = Data)
plt.title("Average transaction volume by month")
```

[68]:
```
Text(0.5, 1.0, 'Average transaction volume by month')
```



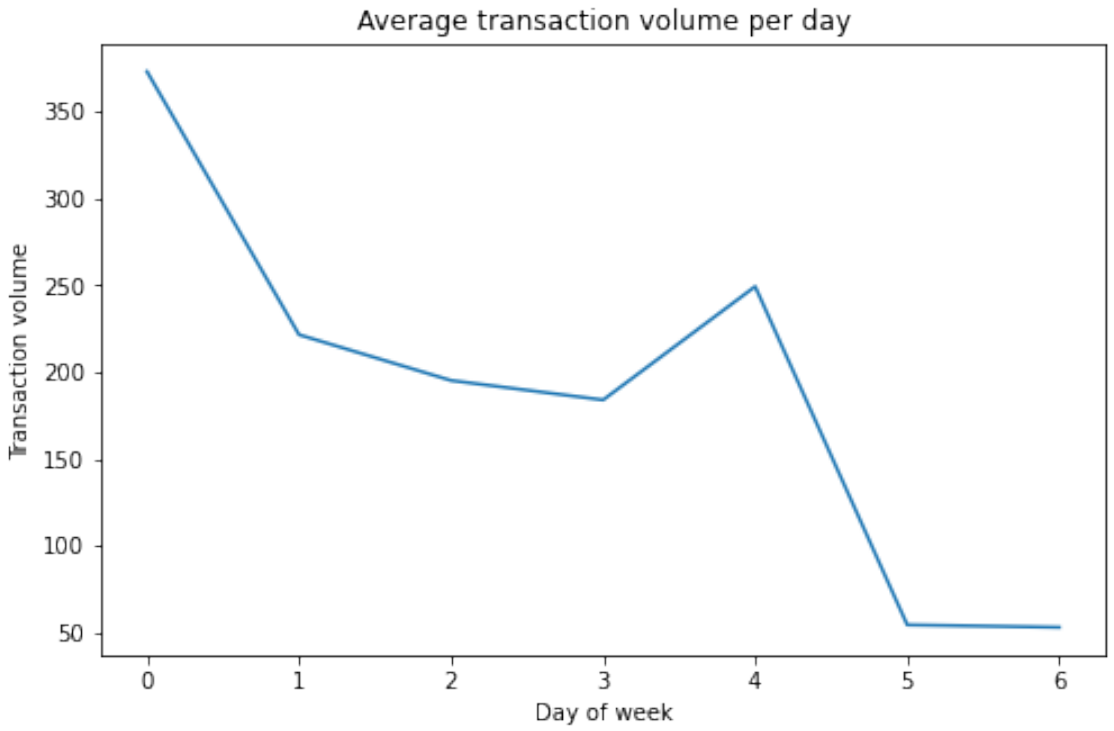
[69]:
```
average_daily_volume = pd.DataFrame(Data.groupby("dayofweek").amount.mean())
average_daily_volume.head()
```

[69]:

dayofweek	amount
0	373.221000
1	221.576456
2	195.215570
3	184.010422
4	249.353517

[70]:
```
fig, ax = plt.subplots(figsize = (8, 5))
ax.plot(average_daily_volume.index, average_daily_volume.amount)
plt.title("Average transaction volume per day")
plt.ylabel("Transaction volume")
plt.xlabel("Day of week")
```

[70]:
```
Text(0.5, 0, 'Day of week')
```



[71]:
```
Data.txn_description.value_counts()
```

[71]:
```
SALES-POS     3934
POS           3783
PAYMENT       2600
PAY/SALARY     883
INTER BANK     742
PHONE BANK     101
Name: txn_description, dtype: int64
```

[72]:
```
Data.loc[Data.txn_description == "PAY/SALARY", "category"] = "Salary"
Data.loc[(Data.txn_description == "SALES-POS") | (Data.txn_description ==␣
↪"POS"), "category"] = "Purchase"
Data.category.fillna("Others", inplace = True)
Data[["txn_description", "category"]].head(10)
```

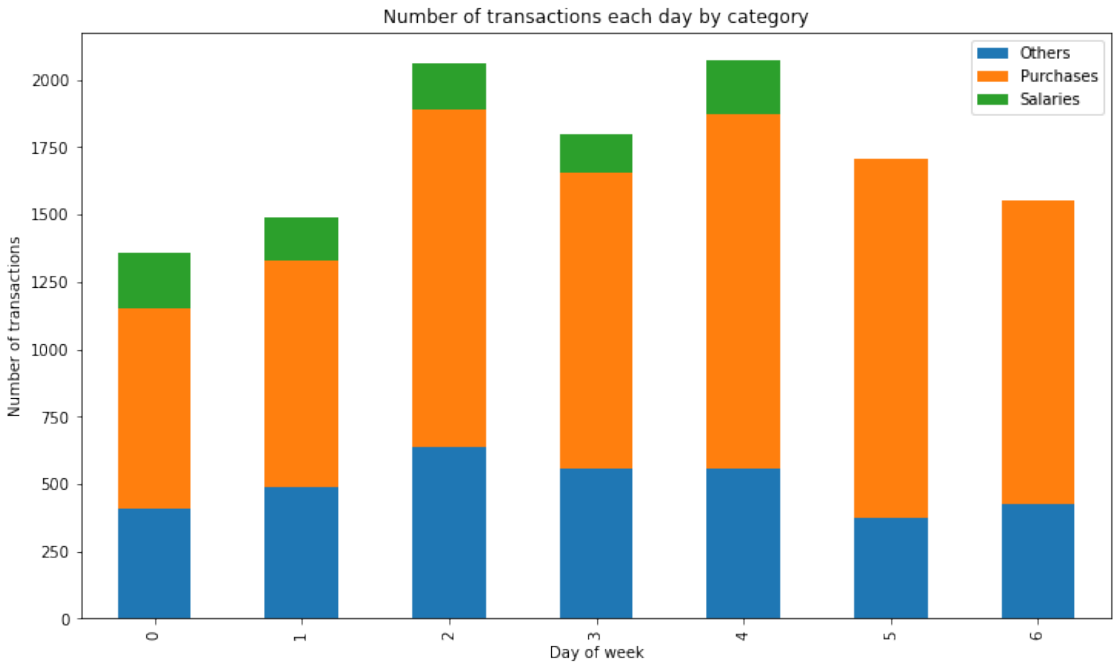
[72]:

	txn_description	category
0	POS	Purchase
1	SALES-POS	Purchase
2	POS	Purchase
3	SALES-POS	Purchase
4	SALES-POS	Purchase
5	PAYMENT	Others

```
6       SALES-POS  Purchase
7             POS  Purchase
8             POS  Purchase
9      INTER BANK    Others
```

[73]:
```
stacked_barplot = pd.DataFrame(Data.groupby(["dayofweek", "category"]).amount.
↪count())
stacked_barplot.unstack().plot(kind = "bar", stacked = True, figsize = (12, 7))
plt.title("Number of transactions each day by category")
plt.legend(["Others", "Purchases", "Salaries"])
plt.ylabel("Number of transactions")
plt.xlabel("Day of week")
```

[73]:
```
Text(0.5, 0, 'Day of week')
```

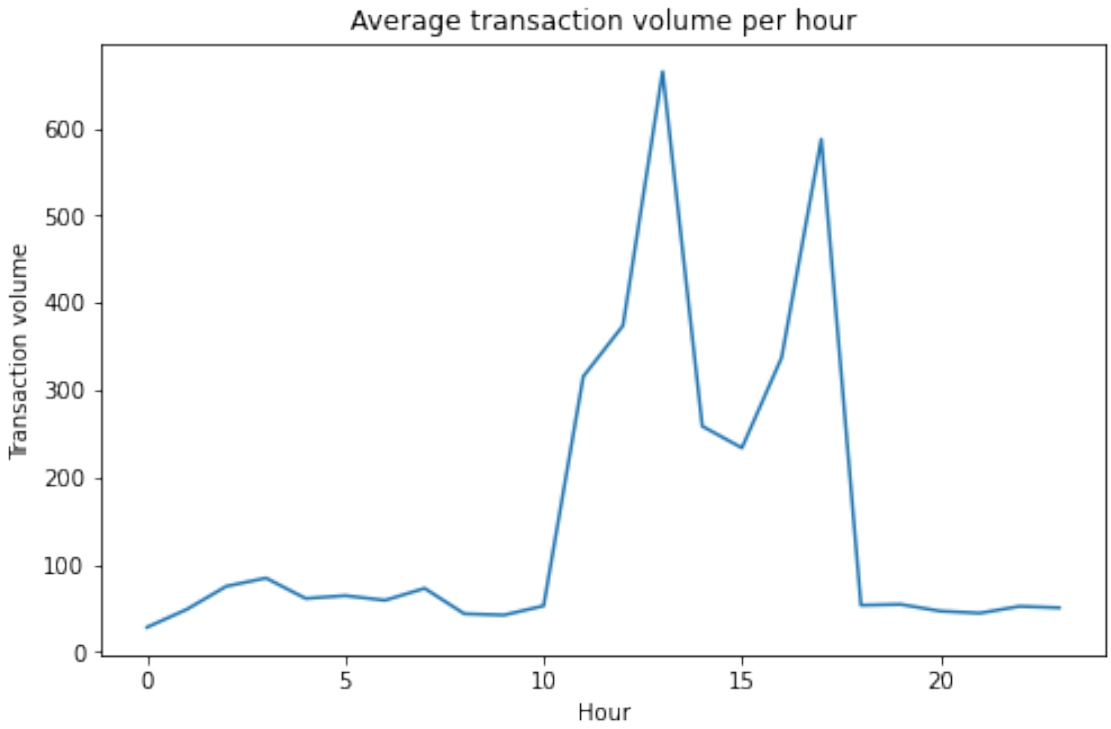


[74]:
```
average_hourly_volume = pd.DataFrame(Data.groupby("hour").amount.mean())
average_hourly_volume.head()
```

[74]:
```
          amount
hour
0      28.274907
1      48.716402
2      75.269764
3      84.725918
4      61.301845
```

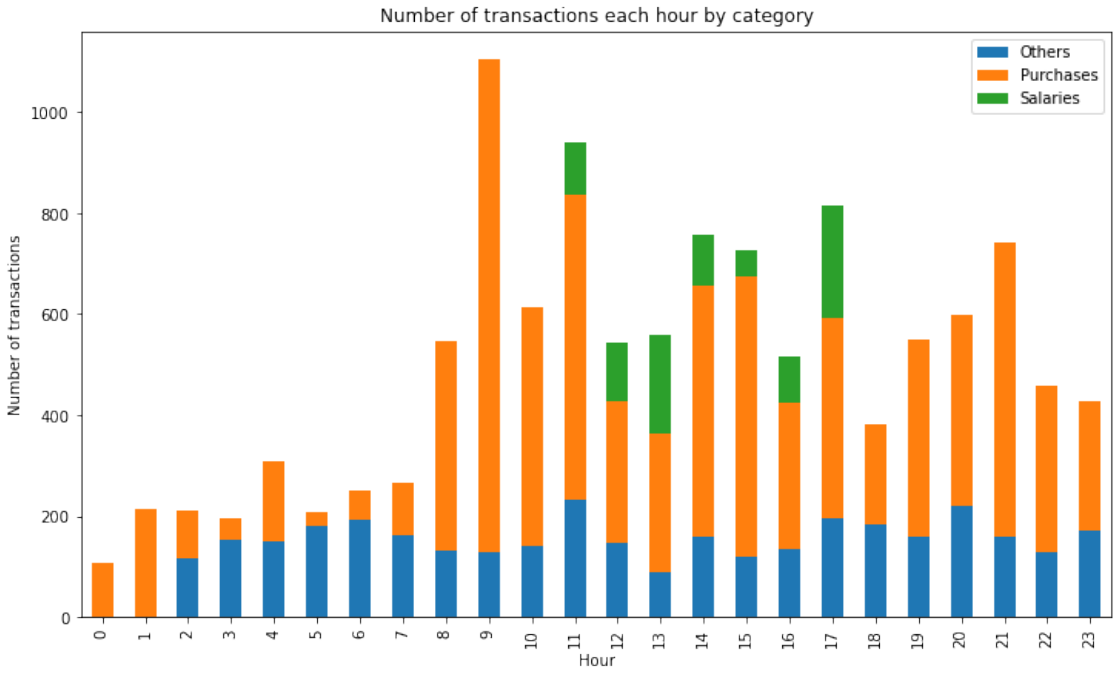
[75]:
```
fig, ax = plt.subplots(figsize = (8, 5))
ax.plot(average_hourly_volume.index, average_hourly_volume.amount)
plt.title("Average transaction volume per hour")
plt.ylabel("Transaction volume")
plt.xlabel("Hour")
```

[75]:
```
Text(0.5, 0, 'Hour')
```



[76]:
```
stacked_barplot = pd.DataFrame(Data.groupby(["hour", "category"]).amount.
↪count())
stacked_barplot.unstack().plot(kind = "bar", stacked = True, figsize = (12, 7))
plt.title("Number of transactions each hour by category")
plt.legend(["Others", "Purchases", "Salaries"])
plt.ylabel("Number of transactions")
plt.xlabel("Hour")
```

[76]:
```
Text(0.5, 0, 'Hour')
```



[77]:
```
Data[["long_lat", "merchant_long_lat"]].head()
```

[77]:

	long_lat	merchant_long_lat
0	153.41 -27.95	153.38 -27.99
1	153.41 -27.95	151.21 -33.87
2	151.23 -33.94	151.21 -33.87
3	153.10 -27.66	153.05 -26.68
4	153.41 -27.95	153.44 -28.06

[78]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statistics import mode

# Machine learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error
```

[81]:
```
pd.DataFrame({"Columns": Data.columns})
```

[81]:

	Columns
0	status
1	card_present_flag
2	account
3	currency
4	long_lat
5	txn_description
6	merchant_id
7	first_name
8	balance
9	date
10	gender
11	age
12	merchant_suburb
13	merchant_state
14	extraction
15	amount
16	transaction_id
17	country
18	customer_id
19	merchant_long_lat
20	movement
21	month
22	dayofweek
23	hour
24	category

[82]:
```
salary_df = pd.DataFrame({"customer_id": Data.customer_id.unique()})
salary_df.head()
```

[82]:

	customer_id
0	CUS-2487424745
1	CUS-2142601169
2	CUS-1614226872
3	CUS-2688605418
4	CUS-4123612273

[83]:
```
example = Data.loc[(Data.customer_id == salary_df.customer_id[0]) & (Data.
↪txn_description == "PAY/SALARY"), ["date", "amount"]].groupby("date",␣
↪as_index = False).sum()
example
```

[83]:

	date	amount
0	2018-08-01	1013.67
1	2018-08-08	1013.67
2	2018-08-15	1013.67
3	2018-08-22	1013.67

```
4  2018-08-29  1013.67
5  2018-09-05  1013.67
6  2018-09-12  1013.67
7  2018-09-19  1013.67
8  2018-09-26  1013.67
9  2018-10-03  1013.67
10 2018-10-10  1013.67
11 2018-10-17  1013.67
12 2018-10-24  1013.67
13 2018-10-31  1013.67
```

[84]:
```
# Loop through all salary payments for each customer
# Assume the salary level is constant for each customer over the observed period
df_freq = []
df_amount = []

for customer in range(len(salary_df)):
    salary = Data.loc[(Data.customer_id == salary_df.customer_id[customer]) &␣
 ↪(Data.txn_description == "PAY/SALARY"), ["date", "amount"]].groupby("date",␣
 ↪as_index = False).sum()
    count = len(salary)
    if count == 0:
        df_amount.append(np.nan)
        df_freq.append(np.nan)
    else:
        days_between_payments = []
        for date in range(len(salary)-1):
            days_between_payments.append((salary.date[date + 1] - salary.
 ↪date[date]).days)
        df_freq.append(max(days_between_payments))
        df_amount.append(mode(salary.amount))

salary_df["salary_freq"] = df_freq
salary_df["salary_amount"] = df_amount
salary_df["annual_salary"] = salary_df["salary_amount"] /␣
 ↪salary_df["salary_freq"] * 365.25
salary_df.head()
```

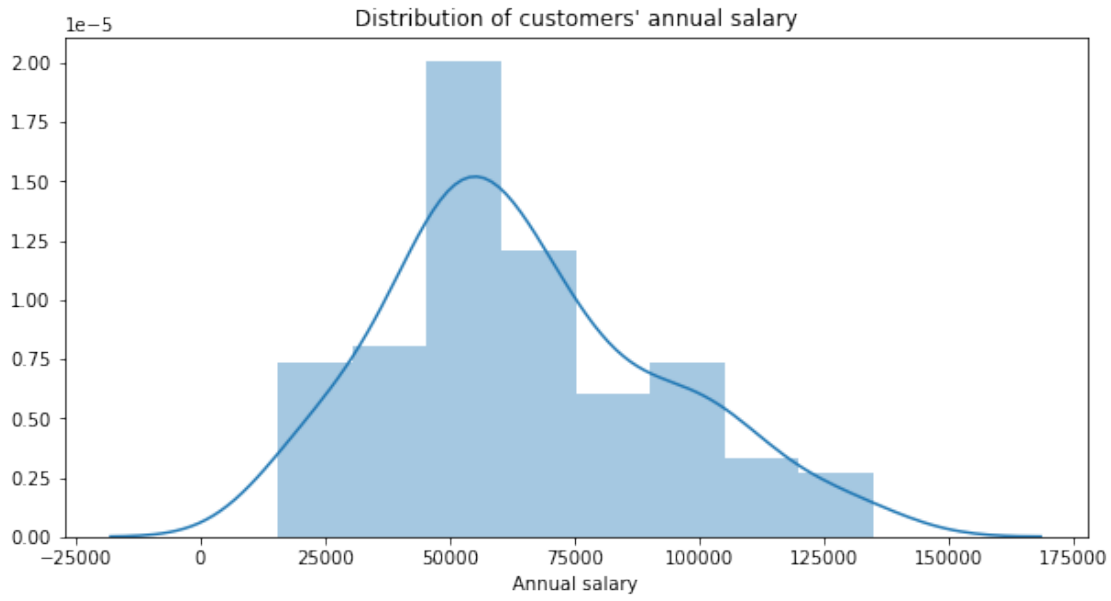
[84]:

	customer_id	salary_freq	salary_amount	annual_salary
0	CUS-2487424745	7	1013.67	52891.852500
1	CUS-2142601169	7	1002.13	52289.711786
2	CUS-1614226872	7	892.09	46547.981786
3	CUS-2688605418	14	2320.30	60534.969643
4	CUS-4123612273	7	1068.04	55728.801429

[85]:
```
# Plot customer's annual salary distribution
plt.figure(figsize = (10, 5))
```

```
sns.distplot(salary_df.annual_salary)
plt.title("Distribution of customers' annual salary")
plt.xlabel("Annual salary")
```

[85]: Text(0.5, 0, 'Annual salary')



[86]:
```
# Unique customer id's
unique_id = Data.customer_id.unique()
len(unique_id)
```

[86]: 100

[87]:
```
unique_id[:5]
```

[87]:
```
array(['CUS-2487424745', 'CUS-2142601169', 'CUS-1614226872',
       'CUS-2688605418', 'CUS-4123612273'], dtype=object)
```

[89]:
```
avg_no_weekly_trans = []
for id_ in unique_id:
    array = Data.loc[Data.customer_id == id_, "date"]
    avg_no_weekly_trans.append(round(len(array)/array.nunique()*7))
avg_no_weekly_trans[:5]
```

[89]: [48, 29, 24, 14, 21]

[90]:
```
max_amount = []
for id_ in unique_id:
```

```
    array = Data.loc[Data.customer_id == id_, "amount"]
    max_amount.append(max(array))
max_amount[:5]
```

[90]: [1452.21, 2349.55, 892.09, 2320.3, 1068.04]

[91]:
```
no_large_trans = []
for id_ in unique_id:
    count = 0
    array = Data.loc[Data.customer_id == id_, "amount"]
    for amount in array:
        if amount > 100:
            count += 1
    no_large_trans.append(count)
no_large_trans[:5]
```

[91]: [22, 23, 22, 25, 32]

[92]:
```
no_days_with_trans = []
for id_ in unique_id:
    array = Data.loc[Data.customer_id == id_, "date"]
    no_days_with_trans.append(array.nunique())
no_days_with_trans[:5]
```

[92]: [85, 74, 76, 63, 44]

[93]:
```
avg_trans_amount = []
for id_ in unique_id:
    array = Data.loc[Data.customer_id == id_, "amount"]
    avg_trans_amount.append(array.mean())
avg_trans_amount[:5]
```

[93]: [45.34877162629756,
 78.20610561056101,
 74.46501930501928,
 159.3041860465116,
 166.50835820895517]

[94]:
```
median_balance = []
for id_ in unique_id:
    array = Data.loc[Data.customer_id == id_, "balance"]
    median_balance.append(array.median())
median_balance[:5]
```

[94]: [1580.4, 1132.66, 3618.5, 5616.63, 6162.45]

[95]:
```
# Assume customers live in the state where most of their transactions occured
state = []
```

```
for id_ in unique_id:
    array = Data.loc[Data.customer_id == id_, "merchant_state"]
    state.append(mode(array))
state[:5]
```

[95]:
```
['QLD', 'NSW', 'QLD', 'NSW', 'VIC']
```

[96]:
```
age = []
for id_ in unique_id:
    array = Data.loc[Data.customer_id == id_, "age"]
    age.append(mode(array))
age[:5]
```

[96]:
```
[26, 38, 40, 20, 43]
```

[97]:
```
gender = []
for id_ in unique_id:
    array = Data.loc[Data.customer_id == id_, "gender"]
    gender.append(mode(array))
gender[:5]
```

[97]:
```
['F', 'M', 'F', 'M', 'F']
```

[98]:
```
# Predictor variables
features_df = pd.DataFrame({"customer_id": unique_id,
                            "avg_no_weekly_trans": avg_no_weekly_trans,
                            "max_amount": max_amount,
                            "no_large_trans": no_large_trans,
                            "avg_trans_amount": avg_trans_amount,
                            "median_balance": median_balance,
                            "state": state,
                            "age": age,
                            "gender": gender})
features_df.head()
```

[98]:

	customer_id	avg_no_weekly_trans	max_amount	no_large_trans	avg_trans_amount	median_balance	state	age	gender
0	CUS-2487424745	48	1452.21	22	45.348772	1580.40	QLD	26	F
1	CUS-2142601169	29	2349.55	23	78.206106	1132.66	NSW	38	M
2	CUS-1614226872	24	892.09	22	74.465019	3618.50	QLD	40	F
3	CUS-2688605418	14	2320.30	25	159.304186	5616.63	NSW	20	M
4	CUS-4123612273	21	1068.04	32	166.508358	6162.45	VIC	43	F

[99]:
```
salary_df.head()
```

[99]:

	customer_id	salary_freq	salary_amount	annual_salary
0	CUS-2487424745	7	1013.67	52891.852500
1	CUS-2142601169	7	1002.13	52289.711786
2	CUS-1614226872	7	892.09	46547.981786
3	CUS-2688605418	14	2320.30	60534.969643
4	CUS-4123612273	7	1068.04	55728.801429

[100]:
```
df = pd.concat([features_df, salary_df.annual_salary], axis = 1)
df.head()
```

[100]:

	customer_id	avg_no_weekly_trans	max_amount	no_large_trans	\
0	CUS-2487424745	48	1452.21	22	
1	CUS-2142601169	29	2349.55	23	
2	CUS-1614226872	24	892.09	22	
3	CUS-2688605418	14	2320.30	25	
4	CUS-4123612273	21	1068.04	32	

	avg_trans_amount	median_balance	state	age	gender	annual_salary
0	45.348772	1580.40	QLD	26	F	52891.852500
1	78.206106	1132.66	NSW	38	M	52289.711786
2	74.465019	3618.50	QLD	40	F	46547.981786
3	159.304186	5616.63	NSW	20	M	60534.969643
4	166.508358	6162.45	VIC	43	F	55728.801429

[101]:
```
df.isnull().sum()
```

[101]:
```
customer_id           0
avg_no_weekly_trans   0
max_amount            0
no_large_trans        0
avg_trans_amount      0
median_balance        0
state                 0
age                   0
gender                0
annual_salary         0
dtype: int64
```

5 Here, we will split 70% of the dataframe into training set, which is used to train our model and 30% of the dataframe into test set, which is used to assess model predictions.

[102]:
```
X = df.drop(["customer_id", "annual_salary"], axis = 1)
Y = df.annual_salary
print("X shape: ", X.shape)
print("Y shape: ", Y.shape)
```

```
X shape:  (100, 8)
Y shape:  (100,)
```

[103]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3,␣
↪random_state = 42)

print("X_train shape: ", X_train.shape)
print("Y_train shape: ", Y_train.shape)
print("X_test shape: ", X_test.shape)
print("Y_test shape: ", Y_test.shape)
```

```
X_train shape:  (70, 8)
Y_train shape:  (70,)
X_test shape:  (30, 8)
Y_test shape:  (30,)
```

[104]:
```
# Crete column transformer
ohe = OneHotEncoder(sparse = False)
scaler = StandardScaler()
column_transform = make_column_transformer((ohe, ["state", "gender"]), (scaler,␣
↪["avg_no_weekly_trans", "max_amount", "no_large_trans", "avg_trans_amount",␣
↪"median_balance", "age"]))
```

6 linear Regression

[106]:
```
# Instantiate model and pipeline
lm = LinearRegression()
lm_pipeline = make_pipeline(column_transform, lm)
```

[107]:
```
# Fit pipeline and make predictions
lm_pipeline.fit(X_train, Y_train)
lm_pred = lm_pipeline.predict(X_test)
```

[108]:
```
# RMSE
print("RMSE: ", round(np.sqrt(mean_squared_error(lm_pred, Y_test))))
```

```
RMSE:  27836.0
```

[109]:
```
# Instantiate model and pipeline
tree = DecisionTreeRegressor()
tree_pipeline = make_pipeline(column_transform, tree)
```

[110]:
```
# Fit pipeline and make predictions
tree_pipeline.fit(X_train, Y_train)
tree_pred = tree_pipeline.predict(X_test)
```

[111]:
```
# RMSE
print("RMSE: ", round(np.sqrt(mean_squared_error(tree_pred, Y_test))))
```

```
RMSE:  22481.0
```

Conclusion The RMSE for both models are over $20,000 and although decision tree performed better than linear regression by having a smaller RMSE, both models still appear to be highly inaccurate. Therefore, it is risky to use them to predict customers' income bracket. More data is required to develop a more reliable model.

Nevertheless, one can invest more time into coming up with more features and selecting the best ones using backward elimination by optimising for a specific metric like AIC, however I doubt the result will be materially different as we only have a very limited amount of data (100 salaries) available.