ANZ Data Virtual Internship - Task 1 - Josh Bryden

In [64]:
```
# Imports
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import geopandas as gpd
from shapely.geometry import Point
from geopandas import GeoDataFrame
import plotly_express as px

#Pandas settings
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# seaborn settings
sns.set_style("darkgrid")
```

Data Importing and Exploratory Data Analysis

In [65]:
```
# read in csv file
data = pd.read_csv('ANZ_synthesised_transaction_dataset.csv')
# display head
data.head()
```

Out[65]:

	status	card_present_flag	bpay_biller_code	account	currency	long_lat	txn_descr
0	authorized	1.0	NaN	ACC-1598451071	AUD	153.41 -27.95	
1	authorized	0.0	NaN	ACC-1598451071	AUD	153.41 -27.95	SALES
2	authorized	1.0	NaN	ACC-1222300524	AUD	151.23 -33.94	
3	authorized	1.0	NaN	ACC-1037050564	AUD	153.10 -27.66	SALES
4	authorized	1.0	NaN	ACC-1598451071	AUD	153.41 -27.95	SALES

In [66]:
```
data.shape
```

Out[66]: (12043, 23)

In [67]:
```
data.describe() #Perform basic summary stats on numeric columns
```

Out[67]:

	card_present_flag	merchant_code	balance	age	amount
count	7717.000000	883.0	12043.000000	12043.000000	12043.000000
mean	0.802644	0.0	14704.195553	30.582330	187.933588
std	0.398029	0.0	31503.722652	10.046343	592.599934
min	0.000000	0.0	0.240000	18.000000	0.100000
25%	1.000000	0.0	3158.585000	22.000000	16.000000
50%	1.000000	0.0	6432.010000	28.000000	29.000000
75%	1.000000	0.0	12465.945000	38.000000	53.655000
max	1.000000	0.0	267128.520000	78.000000	8835.980000

Noting above that the pd.describe displays float and integer objects only, this means many other columns are encoded as objects. The below code calls upon df.dtypes to determine the types of values in each column.

In [68]:
```
data.dtypes
```

Out[68]:
```
status                object
card_present_flag    float64
bpay_biller_code      object
account               object
currency              object
long_lat              object
txn_description       object
merchant_id           object
merchant_code        float64
first_name            object
balance              float64
date                  object
gender                object
age                    int64
merchant_suburb       object
merchant_state        object
extraction            object
amount               float64
transaction_id        object
country               object
customer_id           object
merchant_long_lat     object
movement              object
dtype: object
```

In [69]:
```
data.nunique() # Determines the number of unique values per column
```

Out[69]:
```
status                   2
card_present_flag        2
bpay_biller_code         3
account                100
currency                 1
long_lat               100
txn_description          6
merchant_id           5725
merchant_code            1
first_name              80
balance              12006
date                    91
gender                   2
age                     33
merchant_suburb       1609
merchant_state           8
extraction            9442
```

```
amount                4457
transaction_id       12043
country                  1
customer_id            100
merchant_long_lat     2703
movement                 2
dtype: int64
```

The above code shows us that as per the description we do indeed have 100 customers data based off 100 unique values for the accounts column.

In [70]:
```
data.isnull().sum() # counts the number of null values for each column
```

Out[70]:
```
status                   0
card_present_flag     4326
bpay_biller_code     11158
account                  0
currency                 0
long_lat                 0
txn_description          0
merchant_id           4326
merchant_code        11160
first_name               0
balance                  0
date                     0
gender                   0
age                      0
merchant_suburb       4326
merchant_state        4326
extraction               0
amount                   0
transaction_id           0
country                  0
customer_id              0
merchant_long_lat     4326
movement                 0
dtype: int64
```

In [71]:
```
# Check the percentage of missing values per column
print("Percentage of missing values:")
print()
for column in data.columns:
    print(f'Column {column} has'
          f' {100 * sum(data[column].isna()) / len(data):.2f}%'
          f' missing values')
```

```
Percentage of missing values:

Column status has 0.00% missing values
Column card_present_flag has 35.92% missing values
Column bpay_biller_code has 92.65% missing values
Column account has 0.00% missing values
Column currency has 0.00% missing values
Column long_lat has 0.00% missing values
Column txn_description has 0.00% missing values
Column merchant_id has 35.92% missing values
Column merchant_code has 92.67% missing values
Column first_name has 0.00% missing values
Column balance has 0.00% missing values
Column date has 0.00% missing values
Column gender has 0.00% missing values
Column age has 0.00% missing values
Column merchant_suburb has 35.92% missing values
Column merchant_state has 35.92% missing values
Column extraction has 0.00% missing values
Column amount has 0.00% missing values
Column transaction_id has 0.00% missing values
Column country has 0.00% missing values
```

```
Column customer_id has 0.00% missing values
Column merchant_long_lat has 35.92% missing values
Column movement has 0.00% missing values
```

Based off the above code, we have significant numbers of null values in the dataset. However considering this is transactional data:

```
1. Not all payments were via Bpay - hence we have a lack of
values in bpay_biller_code
2. We have 4326 missing values in the card_present_flag,
merchant_id, merchant_suburb, merchant_state and
merchant_long_lat columns. This could be due to the card not
being present at the time of transaction (online or manual
purchases) or for another reason entirely.
3. We are missing a lot of data in the merchant_code column
(~92%). This could be due to most transctions (~92%) are not
Bpay transactions and will hence not have a merchant code. As
such we should remove this column alongside the bpay_biller_code
column.
```

Data Cleaning

In [72]:

```
# assign to clean data frame variable
data_clean = data

# split up lat_long column into lat and long for ease of plotting later
data_clean[['long','lat']] = data_clean['long_lat'].str.split(' ', expand=Tru

# split up merchant long_lat into lat and long for ease of plotting later
data_clean[['merchant_long','merchant_lat']] = data_clean['merchant_long_lat'
#data_clean.head() # Sanity check
```

In [73]:

```
# drop columns missing signifcant amounts of data / unneeded columns:

# 1. merchant_code - missing data (see above)
# 2. currency - all in AUD in this dataset (based off unique values)
# 3. country - all in Australia in this dataset (based off unique values)
# 4. long_lat - not needed after split above
# 5. merchant_long_lat - not needed after split above
data_clean = data_clean.drop(['merchant_code','currency', 'country', 'long_la
```

In [74]:

```
# Change dtypes to numeric for all latitude and longitude columns
data_clean = data_clean.astype({'long':'float64', 'lat':'float64', 'merchant_
```

In [75]:

```
# ensure that the date column is a datetime object
data_clean['date'] = pd.to_datetime(data_clean['date'], format= '%d/%m/%y')
```

In [76]:

```
# extract day of week from date and add to df - represented by number
data_clean['weekday'] = data_clean['date'].dt.dayofweek

# create dictonary of name of days based off pandas dayofweek function
day_of_week_names={0:'Monday', 1:'Tuesday', 2:'Wednesday', 3:'Thursday', 4:'F

# assign and map weekday column to our dictonary of day names
data_clean['weekday'] = data_clean['date'].dt.dayofweek.map(day_of_week_names

# extract the hour from the extraction column - first convert to datetime the
data_clean['time_hour'] = pd.to_datetime(data_clean['extraction'])
data_clean['time_hour'] = data_clean['time_hour'].dt.hour
```

In [77]:
```
# Sort data by date
data_clean.sort_values(by=['date'], inplace=True)
```

In [78]:
```
data_clean.describe()
```

Out[78]:

	card_present_flag	balance	age	amount	long	
count	7717.000000	12043.000000	12043.000000	12043.000000	12043.000000	12043.0
mean	0.802644	14704.195553	30.582330	187.933588	143.648563	-38.1
std	0.398029	31503.722652	10.046343	592.599934	16.669352	54.6
min	0.000000	0.240000	18.000000	0.100000	114.620000	-573.0
25%	1.000000	3158.585000	22.000000	16.000000	138.690000	-37.7
50%	1.000000	6432.010000	28.000000	29.000000	145.230000	-33.8
75%	1.000000	12465.945000	38.000000	53.655000	151.220000	-30.7
max	1.000000	267128.520000	78.000000	8835.980000	255.000000	-12.3

Noting the above that in the 'lat' and 'long' columns we have large values of 255 degrees and -573 degrees respectively. These values for latitude and longitude are not possible.

In [79]:
```
# Investigate the above by locating rows where -573 is the value in the 'lat'
data_clean.loc[data_clean['lat'] == -573].head() # note --> .head() used here
```

Out[79]:

	status	card_present_flag	account	txn_description	merchant_id	first_name	b
99	posted	NaN	ACC-2901672282	PAYMENT	NaN	Daniel	1
51	posted	NaN	ACC-2901672282	PAYMENT	NaN	Daniel	1
47	authorized	0.0	ACC-2901672282	SALES-POS	7ce5471b-363c-46ab-b398-ca517347829a	Daniel	1
392	posted	NaN	ACC-2901672282	PAY/SALARY	NaN	Daniel	4
531	authorized	1.0	ACC-2901672282	POS	e957b3e1-e8d7-49a4-98a9-e5feba8b1a74	Daniel	4

From the above output we can see that our unsual value for longitude of 255 degrees appears to be for the same customer (Daniel) whos latitude value is -573. Whilst the remaining data could be accurate, we will drop these rows as this will throw off our visualisations later on.

In [80]:
```
# assign the indexs of the affending rows
indexnames_daniel_latlong = data_clean[data_clean['lat']==-573].index
# drop rows with the corresponding index from data_clean
data_clean.drop(indexnames_daniel_latlong, inplace=True)
```

In [81]:
```
print('Therefore a loss of {} rows of data.'.format(data.shape[0]-data_clean.
```

```
Therefore a loss of 123 rows of data.
```

Insights into the dataset

In [82]:
```
# Average transaction amount
print('The average transaction amount is ${:.2f} and the median transaction am

print() # creating a space in output
# Average number of transaction per customer over the 3 month period
print('The average number of transaction per person over the 3 month period i

print() # creating a space in output
# Average balance in an ANZ account across the 3 month period
print('The average balance in an ANZ account across the 3 month period is ${:
```

```
The average transaction amount is $187.10 and the median transaction amount is
$28.74

The average number of transaction per person over the 3 month period is 120.40
and the median number of transactions is 109.00

The average balance in an ANZ account across the 3 month period is $14796.44 a
nd the median balance is $6462.94
```

The above output is showing us that the average transaction amount has been greatly affected by outliers, with an average transaction amount of *$187 and a median value of* $29.

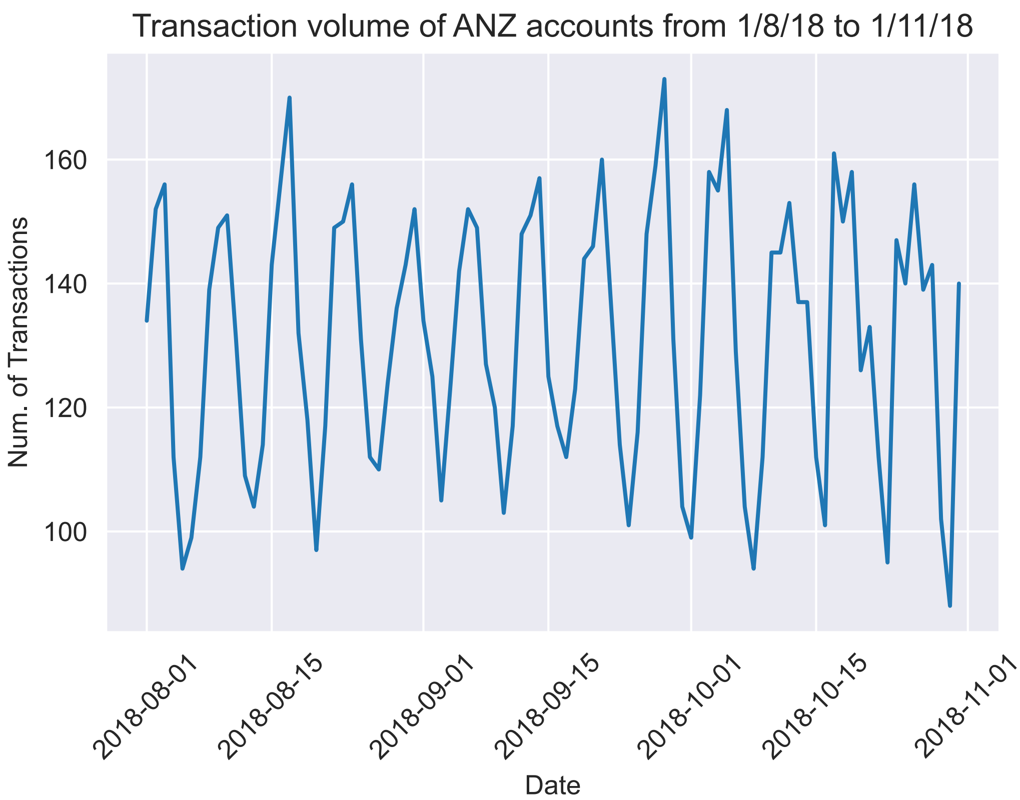
The code above also shows that there is a relatively small difference in the average and median number of transactions per customer across the 3 month period, thus we can conclude that the number of transactions per customer was evenly distributed in this time period.

The average balance in an ANZ account at the time of transaction also differed largely compared to the median balance in an account. The average balance across the time period was *$14707, and the median balance was* $6432, indicating the presence of outliers in the dataset (ie. accounts with large balances).

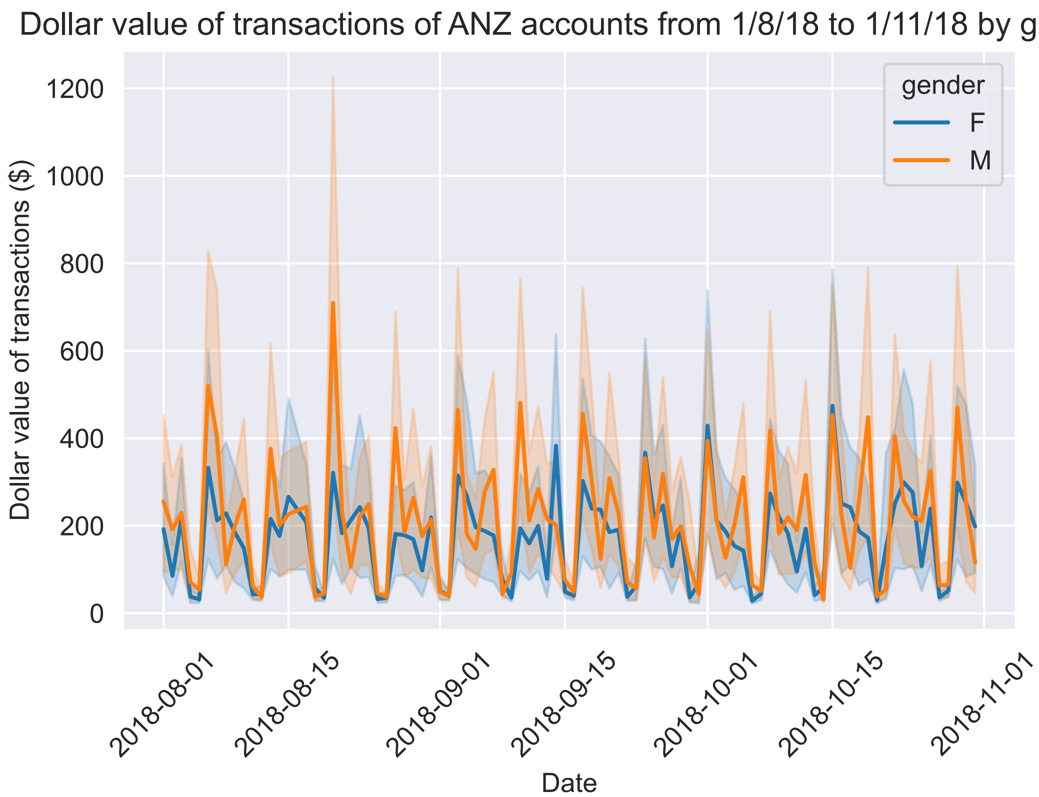
Transaction volume

In [83]:
```
# count the number of transaction per day - store as df
date_transactions_counting = data_clean.groupby('date').count()
```

In [84]:
```
# seaborn lineplot using grouped dataframe of transactions and dates to form
sns.lineplot(date_transactions_counting.index, date_transactions_counting['cu
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Num. of Transactions')
plt.title('Transaction volume of ANZ accounts from 1/8/18 to 1/11/18')
plt.show()
```



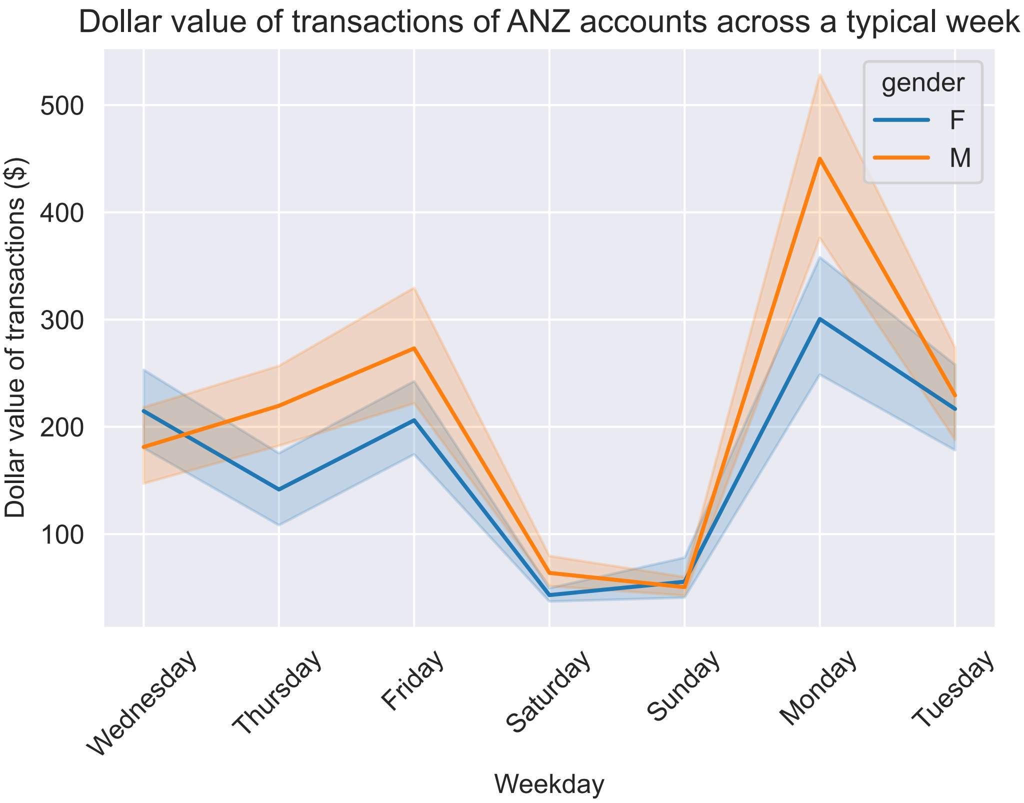
In [85]:
```
sns.lineplot(x='date', y='amount', data=data_clean, hue='gender')
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Dollar value of transactions ($)')
plt.title('Dollar value of transactions of ANZ accounts from 1/8/18 to 1/11/1
plt.show()
```



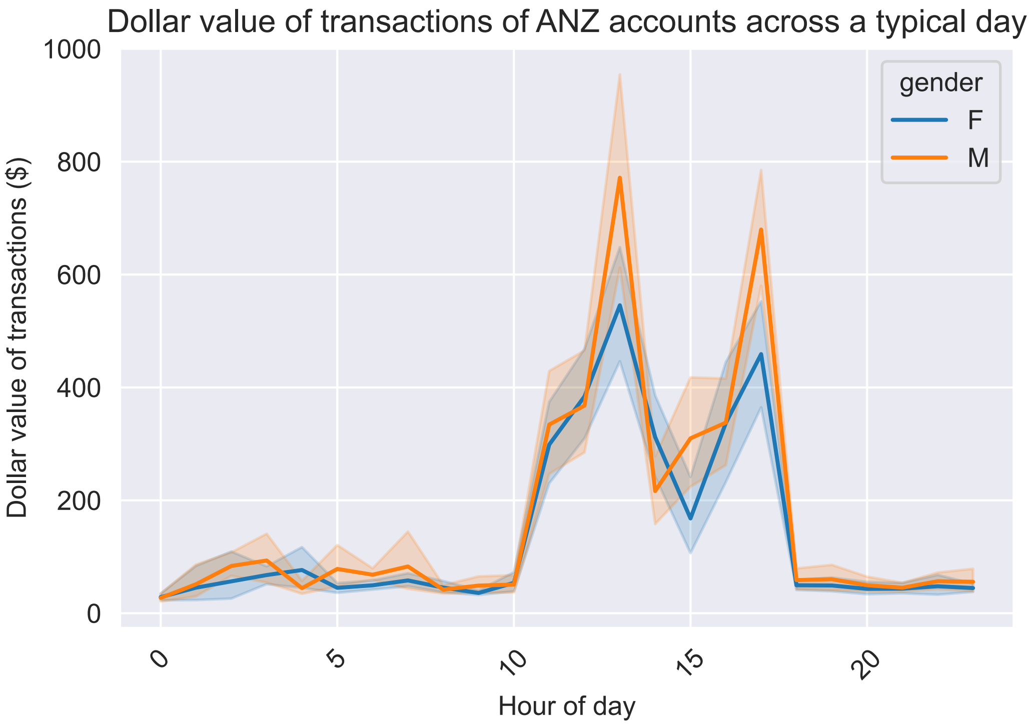
We can see a trend here, lets break this down to examine transactions across a day and a week.

Spending across a typical week and day

In [86]:
```
# sns lineplot
sns.lineplot(x='weekday', y='amount', data=data_clean, hue='gender')
plt.xticks(rotation=45)
plt.xlabel('Weekday')
plt.ylabel('Dollar value of transactions ($)')
plt.title('Dollar value of transactions of ANZ accounts across a typical week
plt.show()
```



In [87]:
```
# sns lineplot
sns.lineplot(x='time_hour', y='amount', data=data_clean, hue='gender')
plt.xticks(rotation=45)
plt.xlabel('Hour of day')
plt.ylabel('Dollar value of transactions ($)')
plt.title('Dollar value of transactions of ANZ accounts across a typical day'
plt.show()
```



Interestingly we can see that transactions for both males and females increase at the begining of the work week (Monday) and then decrease on a Tuesday. Males in the datset tend to then increase spending until Friday before spedning drops on a weekend. Females in the dataset tend to spend less on a Thursday before increasing spending on a Friday. Furthermore we found that most transactions typcially occur for both genders between 10am and 3pm and then 3pm to around 5-7pm.

Transactions by State

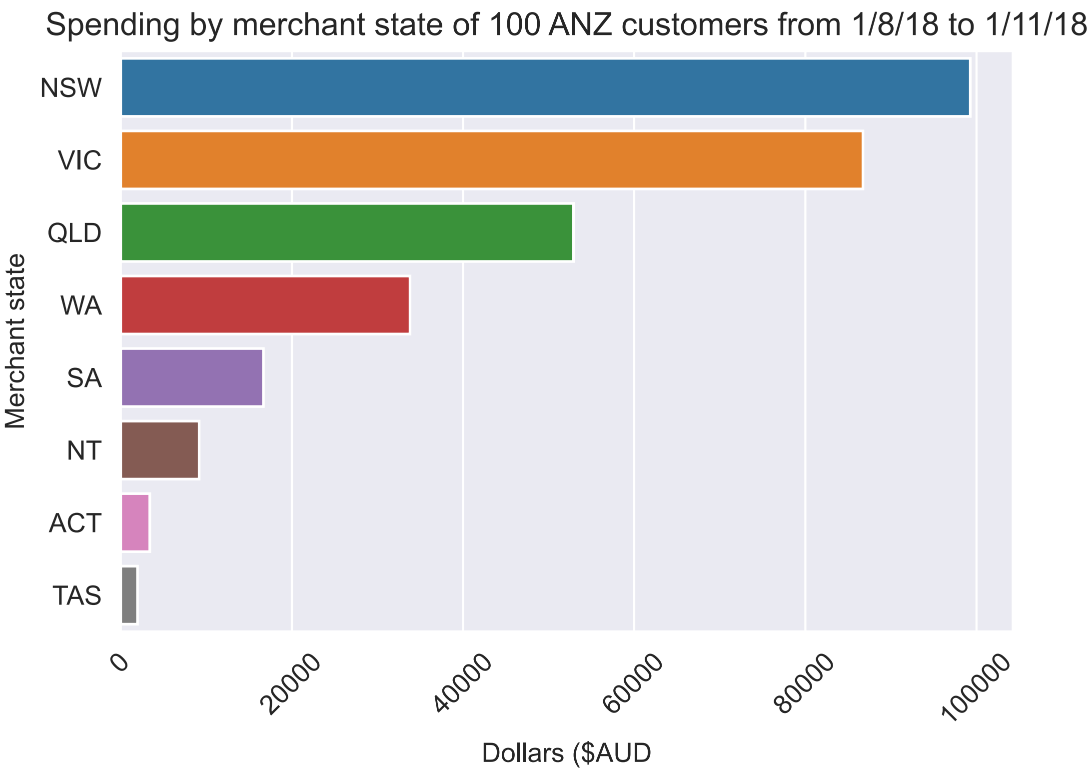
In [88]:
```
# create new df of the sum of transactions grouped by the merchants state
merchant_groupby_state = data_clean.groupby(['merchant_state'])['amount'].sum
# sort by largest first
merchant_groupby_state = merchant_groupby_state.sort_values('amount', ascendi
merchant_groupby_state.head(8) # 8 states and territories
```

Out[88]:

	merchant_state	amount
1	NSW	99272.77
6	VIC	86730.70
3	QLD	52917.30
7	WA	33807.41
4	SA	16673.02
2	NT	9168.89
0	ACT	3395.97
5	TAS	1962.93

In [89]:
```
# visualise the above dataframe
sns.barplot(x='amount', y='merchant_state', data=merchant_groupby_state)
plt.xticks(rotation=45)
plt.xlabel('Dollars ($AUD')
plt.ylabel('Merchant state')
```

```
plt.title('Spending by merchant state of 100 ANZ customers from 1/8/18 to 1/1
plt.show()
```



This is in line with the population rates of each state and territory of Australia.

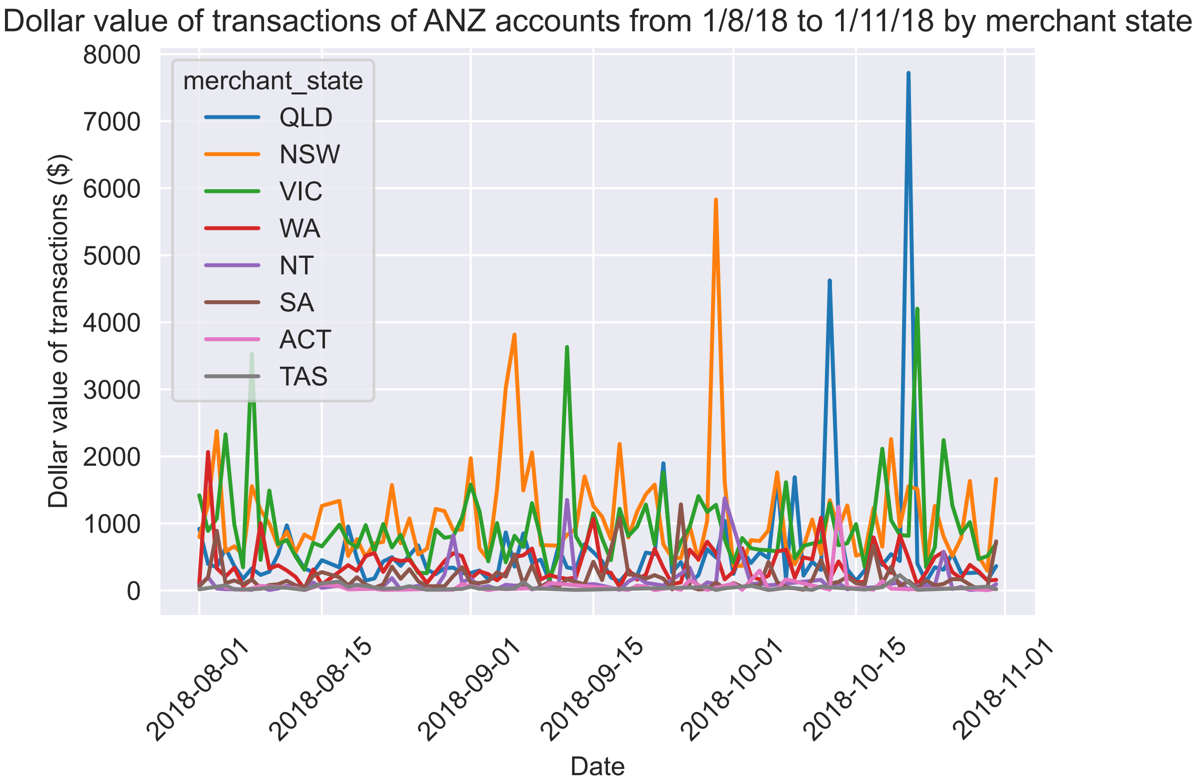
Source:
(https://en.wikipedia.org/wiki/States_and_territories_of_Australia#States_and_territories)

In [90]:
```
# create new df of the sum of transactions by merchant state per date - ie. t
merchant_groupby_state_date = data_clean.groupby(['date','merchant_state'])['
# sort by largest first
merchant_groupby_state_date = merchant_groupby_state_date.sort_values('amount
merchant_groupby_state_date.head()
```

Out[90]:

	date	merchant_state	amount
543	2018-10-21	QLD	7720.32
395	2018-09-29	NSW	5831.31
482	2018-10-12	QLD	4624.66
553	2018-10-22	VIC	4202.75
243	2018-09-06	NSW	3816.98

In [91]:
```
# visualise the above dataframe
sns.lineplot(x='date', y='amount', data=merchant_groupby_state_date, hue='merc
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Dollar value of transactions ($)')
plt.title('Dollar value of transactions of ANZ accounts from 1/8/18 to 1/11/1
plt.show()
```



The above shows that the largest 3 states by population (NSW, VIC, QLD) have the highest spending over time. Notably there was a significant increase in spending in NSW on the 29th of September 2018 and in QLD on the 21st of October 2018

In [92]:
```
# new df of the sum of transactions aggregated by merchant suburb
merchant_groupby_suburb = data_clean.groupby(['merchant_suburb'])['amount'].s
merchant_groupby_suburb = merchant_groupby_suburb.sort_values('amount', ascen
merchant_groupby_suburb.head()
```

Out[92]:

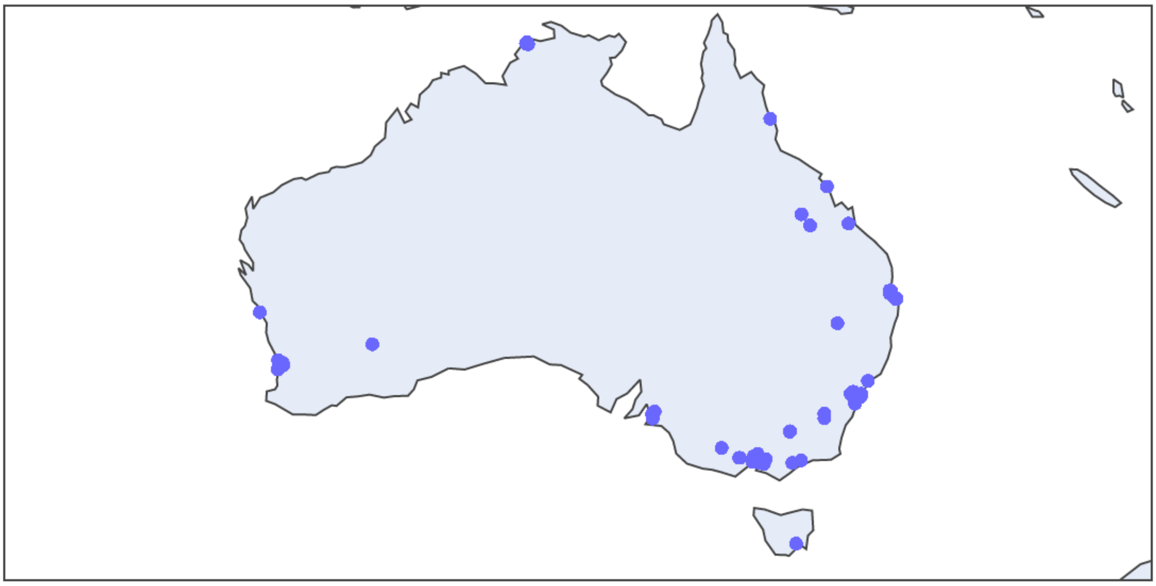
	merchant_suburb	amount
1379	Sydney	20303.55
893	Melbourne	11746.02
1321	South Brisbane	11740.58
880	Mascot	10282.62
978	Mount Gambier	4710.25

The above dataframe shows that the majority of spending on these ANZ accounts is occuring in major CBD's and surrounding suburbs.

Geographical plotting of Transactions

In [93]:
```
# using point from shaply - zip together long and lat columns to create our c
transactions_geographical_lat_long = [Point(xy) for xy in zip(data_clean['lon
# create geo pandas df using above zipped coordinates and link with data_clea
dataframe_geopandas = GeoDataFrame(data_clean, geometry= transactions_geograp
# create plotly express scatter geo plot
fig = px.scatter_geo(dataframe_geopandas, lat=dataframe_geopandas.geometry.y,
fig.update_geos(fitbounds='locations')  # ensures zoom level is on Aus
fig.show()
```

Load image for html output where plotly will not render



Saving dataset to pickle for later analysis

In [94]:
```
data_clean.to_pickle('anz_data_clean.pickle')
```