Branch: master ▾ ibmadvanceddatascience / Capstone.ipynb

Find file Copy path

arvindrajan92 Add files via upload c326cad on Sep 16

1 contributor

2381 lines (2380 sloc) 130 KB

Exploring Data Sources and Use Cases

The dataset used in this IBM Data Science Capstone Project is obtained from THE ICONIC (https://github.com/theiconic/datascientist), one of the largest online fashion retailer in Australia. The file is part of their Data Science Challenge assigned to their potential candidates for the role of data scientist.

OBJECTIVE: To 'infer' a customer's gender based on the amazingly rich user behavioural data, which will allow us to better tailor our site and offerings to their needs.

The dataframe is extracted from the 'data.json' file as pandas dataframe and the head is displayed

In [1]:
```
import pandas as pd
df_data = pd.read_json('data.json')
df_data.head()
```

Out[1]:

	afterpay_payments	android_orders	apple_payments	average_discount_onoffer	average_discount_used	cancels	cc_
0	0	0	0	0.3364	3584.4818	0	1
1	0	0	0	0.1404	1404.0966	0	0
2	1	0	0	0.1851	1899.7270	2	1
3	0	0	0	0.0000	3875.6715	0	1
4	0	0	0	0.0000	0.0000	0	1

5 rows × 43 columns

The dataframe is examined using the describe function

In [2]:
```
pd.set_option('display.max_columns', 500) # altering maximum columns that can be displayed in order to exa
mine all columns
df_data.describe()
```

Out[2]:

	afterpay_payments	android_orders	apple_payments	average_discount_onoffer	average_discount_used	cancels
count	46279.000000	46279.000000	46279.000000	46279.000000	46279.000000	46279.0(
mean	0.053437	0.042935	0.000562	0.190271	2357.381799	0.05309
std	0.224905	0.535762	0.023696	0.190814	2033.075229	2.16983
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000(
25%	0.000000	0.000000	0.000000	0.000000	83.883000	0.00000(
50%	0.000000	0.000000	0.000000	0.150000	2122.648100	0.00000(
75%	0.000000	0.000000	0.000000	0.314300	3829.882950	0.00000(
max	1.000000	33.000000	1.000000	1.000000	10000.000000	460.000(

From the original dataset, the following three issues have been identified:

1. Some entries in the `days_since_last_order` were represented in hours instead of days; hence making the number larger than `days_since_first_order`, which is not sensible.
2. Many entries of `coupon_discount_applied` were filled with NaNs. This could be a mistake in entry, e.g., a dash sign.
3. All the entries under `average_discount_used` were in multiples of 10,000, i.e., in the range of 0 to 10,000. Just like `average_discount_onoffer`, `average_discount_used` should also be showing the discount rate.

To further examine the relationship between the features, a correlation metrix is visualised

In [3]:
```
# Import seaborn, numpy and pyplot packages
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(20,10))

# Generate a mask for the upper triangle
mask = np.zeros_like(df_data.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(df_data.corr(), mask=mask, linewidths=.1)
```

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x1e52b285e10>

From the correlation matrix visualisation above, it can be seen that mspt_items, mapp_items, and mftw_items are highly correlated to male_items, which is an important observation for clustering in the upcoming task

ETL (Extract Transform Load)

The customer IDs are made as the index values of the dataframe

In [4]:
```
# Making customer_id as the index to the dataframe
df_data.set_index('customer_id', inplace=True)
df_data.head()
```

Out[4]:

	afterpay_payments	android_orders	apple_payments	average_discount_onoffer	av
customer_id					
64f7d7dd7a59bba7168cc9c960a5c60e	0	0	0	0.3364	3
fa7c64efd5c037ff2abcce571f9c1712	0	0	0	0.1404	1
18923c9361f27583d2320951435e4888	1	0	0	0.1851	1
aa21f31def4edbdcead818afcdfc4d32	0	0	0	0.0000	3
668c6aac52ff54d4828ad379cdb38e7d	0	0	0	0.0000	0.

Data Cleansing and Feature Engineering

From the issues identified above, the following three changes have been made in the process of cleaning the dataset:

1. Some entries in the `days_since_last_order` were represented in hours instead of days; hence making the number larger than `days_since_first_order`, which is not sensible. For such values, they were **divided by 24**.
2. Many entries of `coupon_discount_applied` were filled with NaNs. Based on the assumption that they could be a mistake in entry, e.g., a dash sign, they are **replaced with 0s**.
3. All the entries under `average_discount_used` were in multiples of 10,000, i.e., in the range of 0 to 10,000. Just like `average_discount_onoffer`, `average_discount_used` should also be showing the discount rate. Accordingly, `average_discount_used` was **adjusted to be in the range of 0 to 1**.

Additionally, for convenience during further analysis, strings 'Y' and 'N' from label `is_newsletter_subscriber` have been **replaced with 1s and 0s** respectively.

In [5]:
```
# Create a copy of the dataframe df_customer named df_customer_clean to store the corrected columns of the
 original dataframe
df_data_clean = df_data.copy()

# correction to average_discount_used by changing to rate
df_data_clean['average_discount_used'] = df_data['average_discount_used']/df_data['average_discount_used']
.max()

# correction to coupon_discount_applied to fill in NaN with 0
df_data_clean['coupon_discount_applied'] = df_data['coupon_discount_applied'].fillna(0)

# correction to is_newsletter_subscriber to 1 and 0 for Y and N respectively and store in new dataframe df
_customer_clean
df_data_clean['is_newsletter_subscriber'] = df_data.is_newsletter_subscriber.replace(('Y','N'), (1,0))

# create function to correct the days_since_last_order column
def correct_days_since_last_order(first_order,last_order):
    for i, value in enumerate(last_order):
        if value > first_order[i]:
            last_order[i] = value/24
    return last_order

# correction to the days_since_last_order column
df_data_clean['days_since_last_order'] = correct_days_since_last_order(df_data['days_since_first_order'].v
alues,df_data['days_since_last_order'].values)

# describe the data to check if the corrections are made
df_data_clean[['average_discount_used','coupon_discount_applied','is_newsletter_subscriber','days_since_la
st_order']].describe()
```

Out[5]:

	average_discount_used	coupon_discount_applied	is_newsletter_subscriber	days_since_last_order
count	46279.000000	46279.000000	46279.000000	46279.000000
mean	0.235738	135.939460	0.408501	1059.453986
std	0.203308	743.982289	0.491562	679.425687
min	0.000000	0.000000	0.000000	1.000000

25%	0.008388	0.000000	0.000000	360.000000
50%	0.212265	0.000000	0.000000	1111.000000
75%	0.382988	32.720000	1.000000	1735.000000
max	1.000000	33332.260000	1.000000	2160.000000

In [6]:
```
# Scaling the chosen features of the data before training the autoencoder
from sklearn import preprocessing
df_data_clean_scaled = preprocessing.scale(df_data_clean[['mspt_items','male_items','mapp_items','mftw_ite
ms']])
```

Model Definition

Clustering using deep-learning and hierarchical clustering

In [7]:
```
# Import models and packages for deep-learning
import keras.backend as K

from keras.models import Model
from keras.layers import Dense, Input, Dropout
from keras.engine.topology import Layer
```

```
C:\Users\Arvind\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second a
rgument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.fl
oat64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [8]:
```
# Split the data up in train and validation sets to 70:30
from sklearn.model_selection import train_test_split
train_x, val_x = train_test_split(df_data_clean_scaled, test_size=0.3, random_state=32)
```

In [9]:
```
# Building the autoencoder
input_shape = train_x.shape[1]
input_data = Input(shape=(input_shape,))

# "encoded" is the encoded representation of the input
encoded = Dense(32, activation='relu')(input_data)
encoded = Dense(128, activation='relu')(encoded)
encoded = Dense(128, activation='relu')(encoded)
encoded = Dense(2, activation='sigmoid')(encoded)

# "decoded" is the lossy reconstruction of the input
decoded = Dense(128, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(32, activation='relu')(decoded)
decoded = Dense(input_shape)(decoded)

# This model maps an input to its reconstruction
autoencoder = Model(input_data, decoded)
```

In [10]:
```
# Prints out the summary of autoencoder
autoencoder.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 4)                 0         
_________________________________________________________________
dense_1 (Dense)              (None, 32)                160       
_________________________________________________________________
dense_2 (Dense)              (None, 128)               4224      
_________________________________________________________________
dense_3 (Dense)              (None, 128)               16512     
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 258       
_________________________________________________________________
dense_5 (Dense)              (None, 128)               384       
_________________________________________________________________
dense_6 (Dense)              (None, 128)               16512     
_________________________________________________________________
dense_7 (Dense)              (None, 32)                4128      
_________________________________________________________________
dense_8 (Dense)              (None, 4)                 132       
=================================================================
Total params: 42,310
Trainable params: 42,310
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
# This model defined here maps an input to its encoded representation
encoder = Model(input_data, encoded)
```

In [12]:
```
# Compiles the autoencoder with optimizer and loss functions
autoencoder.compile(optimizer='adam', loss='mse')
```

Model Training

In [13]:
```
# Training the autoencoder
train_autoenc = autoencoder.fit(train_x, train_x, epochs=100, batch_size=128, validation_data=(val_x, val_
x))
```

```
Train on 32395 samples, validate on 13884 samples
Epoch 1/100
32395/32395 [==============================] - 2s 60us/step - loss: 0.5094 - val_loss: 0.3650
Epoch 2/100
32395/32395 [==============================] - 1s 36us/step - loss: 0.3078 - val_loss: 0.2753
Epoch 3/100
32395/32395 [==============================] - 1s 38us/step - loss: 0.2844 - val_loss: 0.2688
Epoch 4/100
32395/32395 [==============================] - 1s 40us/step - loss: 0.2607 - val_loss: 0.2438
Epoch 5/100
32395/32395 [==============================] - 1s 38us/step - loss: 0.2258 - val_loss: 0.2291
Epoch 6/100
32395/32395 [==============================] - 1s 40us/step - loss: 0.2590 - val_loss: 0.2167
Epoch 7/100
32395/32395 [==============================] - 1s 42us/step - loss: 0.2196 - val_loss: 0.2062
Epoch 8/100
32395/32395 [==============================] - 1s 42us/step - loss: 0.1965 - val_loss: 0.1637
Epoch 9/100
32395/32395 [==============================] - 1s 39us/step - loss: 0.1707 - val_loss: 0.1526
Epoch 10/100
32395/32395 [==============================] - 1s 40us/step - loss: 0.1430 - val_loss: 0.1072
Epoch 11/100
32395/32395 [==============================] - 1s 38us/step - loss: 0.1196 - val_loss: 0.1028
Epoch 12/100
32395/32395 [==============================] - 1s 36us/step - loss: 0.1138 - val_loss: 0.0997
Epoch 13/100
32395/32395 [==============================] - 1s 30us/step - loss: 0.0963 - val_loss: 0.0895
Epoch 14/100
32395/32395 [==============================] - 1s 30us/step - loss: 0.1135 - val_loss: 0.0903
Epoch 15/100
32395/32395 [==============================] - 1s 30us/step - loss: 0.0997 - val_loss: 0.0846
Epoch 16/100
32395/32395 [==============================] - 1s 28us/step - loss: 0.1096 - val_loss: 0.0823
Epoch 17/100
32395/32395 [==============================] - 1s 29us/step - loss: 0.1000 - val_loss: 0.0720
Epoch 18/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.1009 - val_loss: 0.0810
Epoch 19/100
32395/32395 [==============================] - 1s 36us/step - loss: 0.0827 - val_loss: 0.0736
Epoch 20/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0936 - val_loss: 0.1029
Epoch 21/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0876 - val_loss: 0.0721
Epoch 22/100
32395/32395 [==============================] - 1s 36us/step - loss: 0.0984 - val_loss: 0.0888
Epoch 23/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.1140 - val_loss: 0.0746
Epoch 24/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0948 - val_loss: 0.0776
Epoch 25/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0773 - val_loss: 0.0667
Epoch 26/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0762 - val_loss: 0.0751
Epoch 27/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0774 - val_loss: 0.0832
Epoch 28/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0830 - val_loss: 0.0725
Epoch 29/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0731 - val_loss: 0.0638
Epoch 30/100
32395/32395 [==============================] - 1s 36us/step - loss: 0.0764 - val_loss: 0.1042
Epoch 31/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0927 - val_loss: 0.1026
Epoch 32/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0784 - val_loss: 0.0782
Epoch 33/100
32395/32395 [==============================] - 1s 36us/step - loss: 0.0662 - val_loss: 0.0767
Epoch 34/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0688 - val_loss: 0.0648
Epoch 35/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0742 - val_loss: 0.0689
Epoch 36/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0713 - val_loss: 0.0665
```

```
Epoch 37/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0612 - val_loss: 0.0716
Epoch 38/100
32395/32395 [==============================] - 1s 36us/step - loss: 0.0669 - val_loss: 0.0661
Epoch 39/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0613 - val_loss: 0.0619
Epoch 40/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0616 - val_loss: 0.0675
Epoch 41/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0699 - val_loss: 0.0628
Epoch 42/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.1003 - val_loss: 0.1165
Epoch 43/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0976 - val_loss: 0.0682
Epoch 44/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0800 - val_loss: 0.0721
Epoch 45/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0725 - val_loss: 0.0673
Epoch 46/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0627 - val_loss: 0.0609
Epoch 47/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0578 - val_loss: 0.0617
Epoch 48/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0570 - val_loss: 0.0547
Epoch 49/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0585 - val_loss: 0.0539
Epoch 50/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0586 - val_loss: 0.0536
Epoch 51/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0537 - val_loss: 0.0556
Epoch 52/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0599 - val_loss: 0.0731
Epoch 53/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0716 - val_loss: 0.0556
Epoch 54/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0586 - val_loss: 0.0576
Epoch 55/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0545 - val_loss: 0.0555
Epoch 56/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0550 - val_loss: 0.0653
Epoch 57/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0632 - val_loss: 0.0563
Epoch 58/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0554 - val_loss: 0.0523
Epoch 59/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0480 - val_loss: 0.0499
Epoch 60/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0468 - val_loss: 0.0433
Epoch 61/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0526 - val_loss: 0.0470
Epoch 62/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0861 - val_loss: 0.0567
Epoch 63/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0619 - val_loss: 0.0508
Epoch 64/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0514 - val_loss: 0.0473
Epoch 65/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0460 - val_loss: 0.0445
Epoch 66/100
32395/32395 [==============================] - 1s 34us/step - loss: 0.0632 - val_loss: 0.0584
Epoch 67/100
32395/32395 [==============================] - 1s 31us/step - loss: 0.0790 - val_loss: 0.1245
Epoch 68/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0894 - val_loss: 0.0725
Epoch 69/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.0793 - val_loss: 0.0694
Epoch 70/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0630 - val_loss: 0.0679
Epoch 71/100
32395/32395 [==============================] - 1s 33us/step - loss: 0.1149 - val_loss: 0.1085
Epoch 72/100
32395/32395 [==============================] - 1s 31us/step - loss: 0.0819 - val_loss: 0.0776
Epoch 73/100
32395/32395 [==============================] - 1s 30us/step - loss: 0.0606 - val_loss: 0.0763
Epoch 74/100
32395/32395 [==============================] - 1s 31us/step - loss: 0.0556 - val_loss: 0.0701
Epoch 75/100
32395/32395 [==============================] - 1s 31us/step - loss: 0.0535 - val_loss: 0.0747
Epoch 76/100
32395/32395 [==============================] - 1s 31us/step - loss: 0.0546 - val_loss: 0.0535
Epoch 77/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.1018 - val_loss: 0.0790
Epoch 78/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0629 - val_loss: 0.0860
Epoch 79/100
32395/32395 [==============================] - 1s 38us/step - loss: 0.0661 - val_loss: 0.0775
```

```
Epoch 80/100
32395/32395 [==============================] - 1s 38us/step - loss: 0.0582 - val_loss: 0.0546
Epoch 81/100
32395/32395 [==============================] - 1s 32us/step - loss: 0.0508 - val_loss: 0.0579
Epoch 82/100
32395/32395 [==============================] - 1s 42us/step - loss: 0.0466 - val_loss: 0.0461
Epoch 83/100
32395/32395 [==============================] - 1s 40us/step - loss: 0.0452 - val_loss: 0.0552
Epoch 84/100
32395/32395 [==============================] - 1s 45us/step - loss: 0.0456 - val_loss: 0.0462
Epoch 85/100
32395/32395 [==============================] - 2s 47us/step - loss: 0.0460 - val_loss: 0.0442
Epoch 86/100
32395/32395 [==============================] - 2s 53us/step - loss: 0.0476 - val_loss: 0.0429
Epoch 87/100
32395/32395 [==============================] - 2s 57us/step - loss: 0.0540 - val_loss: 0.0556
Epoch 88/100
32395/32395 [==============================] - 1s 44us/step - loss: 0.0499 - val_loss: 0.0815
Epoch 89/100
32395/32395 [==============================] - 2s 48us/step - loss: 0.0488 - val_loss: 0.0487
Epoch 90/100
32395/32395 [==============================] - 1s 41us/step - loss: 0.0538 - val_loss: 0.0541
Epoch 91/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0525 - val_loss: 0.0696
Epoch 92/100
32395/32395 [==============================] - 1s 38us/step - loss: 0.0485 - val_loss: 0.0555
Epoch 93/100
32395/32395 [==============================] - 1s 42us/step - loss: 0.0447 - val_loss: 0.0433
Epoch 94/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0434 - val_loss: 0.0415
Epoch 95/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0419 - val_loss: 0.0485
Epoch 96/100
32395/32395 [==============================] - 1s 37us/step - loss: 0.0488 - val_loss: 0.0579
Epoch 97/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0496 - val_loss: 0.0582
Epoch 98/100
32395/32395 [==============================] - 1s 39us/step - loss: 0.0603 - val_loss: 0.0531
Epoch 99/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0528 - val_loss: 0.0476
Epoch 100/100
32395/32395 [==============================] - 1s 35us/step - loss: 0.0579 - val_loss: 0.0631
```

In [14]:
```
# Prediction from the autoencoder
pred_enco = encoder.predict(df_data_clean_scaled)
```

In [15]:
```
# Performing hierarchical clusterring
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
gen_pred_hcluster = cluster.fit_predict(pred_enco)
```

In [16]:
```
# Inserting the inferred gender into a new dataframe
df_data_clean_gender = df_data_clean.copy()
df_data_clean_gender['inferred_gender'] = gen_pred_hcluster
```

Model Evaluation

In [17]:
```
from sklearn import metrics
metrics.calinski_harabaz_score(df_data_clean[['mspt_items','male_items','mapp_items','mftw_items']], gen_p
red_hcluster)
```

Out[17]:
```
9343.063124545648
```

In [18]:
```
metrics.silhouette_score(df_data_clean[['mspt_items','male_items','mapp_items','mftw_items']], gen_pred_hc
luster, metric='euclidean')
```

Out[18]:
```
0.568629045434286
```

In [19]:
```
# Examining the gender distribution
df_data_clean_gender['inferred_gender'].value_counts()
```

Out[19]:
```
1    34711
0    11568
Name: inferred_gender, dtype: int64
```

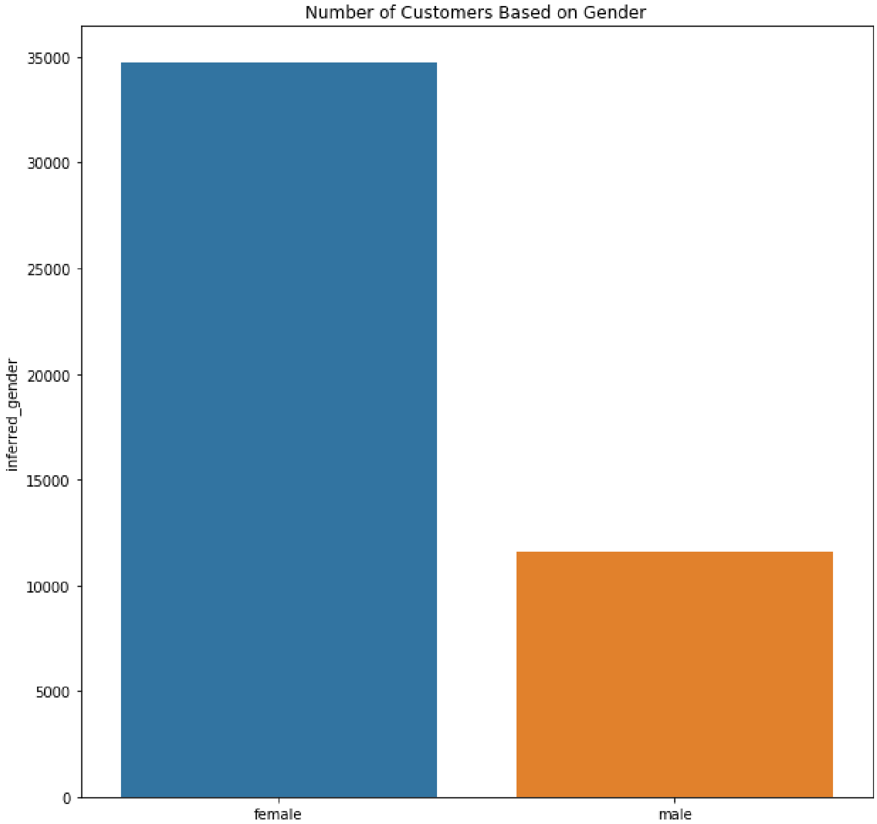
In [20]:
```
# Examining the sum of features attributed to the respective genders
# Based on the observation here and the distribution above, 0 is MALE and 1 is FEMALE
df_data_clean_gender.groupby('inferred_gender').sum()
```

Out[20]:

	afterpay_payments	android_orders	apple_payments	average_discount_onoffer	average_discount_usec
inferred_gender					

0	870	1142	16	2287.6031	2917.831180
1	1603	845	10	6517.9595	7991.896046

In [21]:
```
plt.subplots(1, sharex=True, figsize=(10,10))
sns.barplot(['female','male'],df_data_clean_gender['inferred_gender'].value_counts())
plt.title('Number of Customers Based on Gender')
plt.show()
```



In [22]:
```
f, ax = plt.subplots(2, sharex=True, figsize=(10,10))
sns.barplot(['male','female'], df_data_clean_gender.groupby('inferred_gender').male_items.sum(),ax=ax[0])
ax[0].set_title('Number of Purchases')
sns.barplot(['male','female'], df_data_clean_gender.groupby('inferred_gender').female_items.sum(),ax=ax[1
])
plt.show()
```