Technical Document

Team 2-D

Import packages and data

In [87]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA

pd.set_option('display.max_columns', None)
```

In [58]:

```
flight = pd.read_csv('SunCountry.csv')
flight.head()
```

```
c:\users\naphat\appdata\local\programs\python\python39\lib\site-packages\IPython\core\interactiveshell.py:3553: DtypeWarning: Column
s (12,23) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[58]:

	PNRLocatorID	TicketNum	CouponSeqNbr	ServiceStartCity	ServiceEndCity	PNRCreateDate	ServiceStartDate	PaxName	
0	AAABJK	3377365159634	2	JFK	MSP	2013-11-23	2013-12-13	BRUMSA	4252554D4241434B446964204
1	AAABJK	3377365159634	1	MSP	JFK	2013-11-23	2013-12-08	BRUMSA	4252554D4241434B446964204
2	AAABMK	3372107381942	2	MSP	SFO	2014-02-04	2014-02-23	EILDRY	45494C4445525344696420493
3	AAABMK	3372107381942	1	SFO	MSP	2014-02-04	2014-02-20	EILDRY	45494C4445525344696420493
4	AAABTP	3372107470782	1	MCO	MSP	2014-03-13	2014-04-23	SKELMA	534B454C544F4E44696420493

Data cleaning

In [59]:

```
# Drop duplicate transactions
flight.drop_duplicates(inplace=True)
flight
```

Out[59]:

	PNRLocatorID	TicketNum	CouponSeqNbr	ServiceStartCity	ServiceEndCity	PNRCreateDate	ServiceStartDate	PaxName	
0	AAABJK	3377365159634	2	JFK	MSP	2013-11-23	2013-12-13	BRUMSA	4252554D4241434B446
1	AAABJK	3377365159634	1	MSP	JFK	2013-11-23	2013-12-08	BRUMSA	4252554D4241434B446
2	AAABMK	3372107381942	2	MSP	SFO	2014-02-04	2014-02-23	EILDRY	45494C4445525344696
3	AAABMK	3372107381942	1	SFO	MSP	2014-02-04	2014-02-20	EILDRY	45494C4445525344696
4	AAABTP	3372107470782	1	MCO	MSP	2014-03-13	2014-04-23	SKELMA	534B454C544F4E44696
...	...	...	...	...	...	...	...	...	
3435383	ZZZDRU	3372107838142	1	MSP	SEA	2014-08-23	2014-12-17	VAN WA	56414E2042494E53424
3435384	ZZZDTU	3372106802007	2	SFO	MSP	2013-05-04	2013-07-14	STONDO	53544F4E45446964204
3435385	ZZZDTU	3372106802007	1	MSP	SFO	2013-05-04	2013-07-10	STONDO	53544F4E45446964204
3435386	ZZZNBD	3372106947027	1	MSP	SFO	2013-07-21	2013-08-22	CHORE	43484F44696420493F7
3435387	ZZZYTJ	3372107725754	1	MSP	RSW	2014-07-07	2014-09-28	LIDSRI	4C494453544F4E45446

3292519 rows × 26 columns

In [60]:
```
# Display percent of missing values in each column
flight.isnull().sum() * 100 / len(flight)
```

Out[60]:
```
PNRLocatorID             0.000000
TicketNum                0.000000
CouponSeqNbr             0.000000
ServiceStartCity         0.000000
ServiceEndCity           0.000000
PNRCreateDate            0.000000
ServiceStartDate         0.000000
PaxName                  0.000000
EncryptedName            0.000000
GenderCode               0.897064
birthdateid              0.897064
Age                      0.897064
PostalCode              79.520088
BkdClassOfService        0.000000
TrvldClassOfService      0.000000
BookingChannel           0.000000
BaseFareAmt              0.000000
TotalDocAmt              0.000000
UFlyRewardsNumber       79.398813
UflyMemberStatus        79.398813
CardHolder              79.398813
BookedProduct           65.726485
EnrollDate              79.398813
MarketingFlightNbr       0.000000
MarketingAirlineCode     0.000000
StopoverCode            49.985163
dtype: float64
```

In [61]:
```
# Keep only rows where airlinecode is 'SY' (from SunCountry)
flight = flight[flight['MarketingAirlineCode'] == 'SY']
```

In [62]:
```
# Drop Postalcode (too many missing values)
# Drop UFlyRewardsNumber(we can use UflyMemberStatus to see if they are a member)
# Drop PaxName (it is not a unique identifier of each customer)
flight = flight.drop(['PostalCode','UFlyRewardsNumber','PaxName'],axis=1)
```

In [63]:
```
# Keep only rows where 0 <= age <= 120 and not missing
indexAge = flight[(flight['Age'] < 0) | (flight['Age'] > 120) | (flight['Age'].isna())].index
flight.drop(indexAge , inplace=True)
```

In [64]:
```
# Transform UflyMemberStatus into two columns Elite (1,0) and Standard (1,0)
flight['EliteMember'] = np.where(flight['UflyMemberStatus'] == 'Elite', 1, 0)
flight['StandardMember'] = np.where(flight['UflyMemberStatus'] == 'Standard', 1, 0)
flight = flight.drop(['UflyMemberStatus'],axis=1)
```

In [65]:
```
# Transform CardHolder into 1 if transactions is with card holder, otherwise 0
flight['CardHolder'] = np.where(flight['CardHolder'] == True, 1, 0)
```

In [66]:
```
# Transform BookedProduct into Discount column, 1 if used discount, otherwise 0
flight['Discount'] = np.where(flight['BookedProduct'].isnull() == 0, 1, 0)
flight = flight.drop(['BookedProduct'],axis=1)
```

In [67]:
```
# Transform Gender into (1,0) if Female 1, Male 0
flight['GenderCode'] = np.where(flight['GenderCode'] == 'F', 1, 0)
```

In [68]:
```
# Transfrom StopoverCode into two Layover (1,0) and Over24hrs (1,0)
flight['Layover'] = np.where(flight['StopoverCode'].isnull() == 0, 1, 0)
flight['Over24hrs'] = np.where(flight['Layover'] == 'X', 1, 0)
flight = flight.drop(['StopoverCode'],axis=1)
```

In [69]:
```
# Create a new column DaysBookedOut = ServiceStartDate - PNRCreateDate
flight['ServiceStartDate']= pd.to_datetime(flight['ServiceStartDate'])
flight['PNRCreateDate']= pd.to_datetime(flight['PNRCreateDate'])
flight['DaysBookedOut'] = flight['ServiceStartDate'] - flight['PNRCreateDate']
```

In [70]:
```
# Create a new column ServiceMonth = month of ServiceStartDate
flight['ServiceMonth'] = pd.DatetimeIndex(flight['ServiceStartDate']).month
```

In [71]:
```
# Create a new column Exchange if TotalDocAmt - BaseFareAmt is negative, it is an exchange ticket which is 1, otherwise 0
flight['Exchange'] = np.where(flight['TotalDocAmt']- flight['BaseFareAmt'] < 0, 1, 0)
```

In [72]:
```
# Create new column EnrolltoServiceDays = ServiceStartDate - EnrollDate
flight['EnrollDate']= pd.to_datetime(flight['EnrollDate'])
flight['EnrolltoServiceDays'] = flight['ServiceStartDate'] - flight['EnrollDate']
```

In [73]:
```
#  Transform channel using one-hot encoding
keep_channel = flight.BookingChannel.value_counts().index.values[:5]
flight.loc[~flight.BookingChannel.isin(keep_channel), 'BookingChannel'] = 'Airport booking'

booking_channel_df = pd.get_dummies(flight.BookingChannel)
flight = flight.join(booking_channel_df).drop(columns='BookingChannel')

# Transform seat class and add 'Upgrade' and 'Downgrade'
seat_class_df = pd.get_dummies(flight[['BkdClassOfService', 'TrvldClassOfService']], prefix=['Booked', 'Traveled'], prefix_sep=" ")
flight = flight.join(seat_class_df).drop(columns=['BkdClassOfService', 'TrvldClassOfService'])

flight['Upgrade'] = (flight['Booked Coach'] > flight['Traveled Coach']).astype(int)
flight['Downgrade'] = (flight['Traveled Coach'] > flight['Booked Coach']).astype(int)

# Fix some column name discrepancies
flight.columns = ["".join(col.split()) for col in flight.columns]

# Add CustomerId column
# Here as people might have the same name and surname, we comebine EncryptedName and birthdateid
# to create a unique identifier for each customer
flight['CustomerId'] = flight.EncryptedName + flight.birthdateid.astype('str')
flight = flight.drop(columns=['EncryptedName', 'birthdateid'])
flight.columns
```

Out[73]:
```
Index(['PNRLocatorID', 'TicketNum', 'CouponSeqNbr', 'ServiceStartCity',
       'ServiceEndCity', 'PNRCreateDate', 'ServiceStartDate', 'GenderCode',
       'Age', 'BaseFareAmt', 'TotalDocAmt', 'CardHolder', 'EnrollDate',
       'MarketingFlightNbr', 'MarketingAirlineCode', 'EliteMember',
       'StandardMember', 'Discount', 'Layover', 'Over24hrs', 'DaysBookedOut',
       'ServiceMonth', 'Exchange', 'EnrolltoServiceDays', 'Airportbooking',
       'OutsideBooking', 'ReservationsBooking', 'SCAWebsiteBooking',
       'SYVacation', 'TourOperatorPortal', 'BookedCoach',
       'BookedDiscountFirstClass', 'BookedFirstClass', 'TraveledCoach',
       'TraveledDiscountFirstClass', 'TraveledFirstClass', 'Upgrade',
       'Downgrade', 'CustomerId'],
      dtype='object')
```

In [56]:
```
# Remove outliers using BaseFareAmount

# Get 1st and 3rd quatile of BaseFareAmount
q1, q3 = flight.BaseFareAmt.quantile([0.25, 0.75]).values

# Cutoff for outliers (we use 3 times of interquatile range)
cutoff = q3 + 3 * (q3 - q1)

flight_cleaned = flight[~((flight.BaseFareAmt > cutoff) & (flight.BookedCoach == 1))]
flight_cleaned = flight.loc[~((flight.BaseFareAmt == 0) & (flight.Discount == 0))]

transaction = flight_cleaned.copy()
```

In [28]:
```
transaction.head()
```

Out[28]:

	PNRLocatorID	TicketNum	CouponSeqNbr	ServiceStartCity	ServiceEndCity	PNRCreateDate	ServiceStartDate	GenderCode	Age	BaseFareAmt	...	T
0	AAABJK	3377365159634	2	JFK	MSP	2013-11-23	2013-12-13	1	66.0	234.20	...	
1	AAABJK	3377365159634	1	MSP	JFK	2013-11-23	2013-12-08	1	66.0	234.20	...	
2	AAABMK	3372107381942	2	MSP	SFO	2014-02-04	2014-02-23	0	37.0	293.96	...	
3	AAABMK	3372107381942	1	SFO	MSP	2014-02-04	2014-02-20	0	37.0	293.96	...	
4	AAABTP	3372107470782	1	MCO	MSP	2014-03-13	2014-04-23	1	69.0	112.56	...	

5 rows × 39 columns

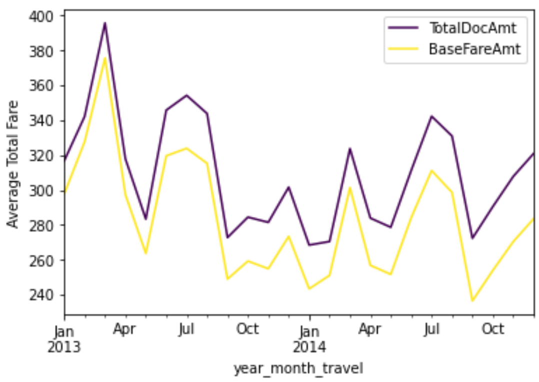
Preliminary Analysis

In [74]:

```python
# Create a copy of transaction dataframe so that it does not affect the original datafframe

df = transaction.copy()

df['year_month_travel'] = pd.to_datetime(df.ServiceStartDate.dt.strftime('%Y-%m'))

df.groupby('year_month_travel')[['TotalDocAmt', 'BaseFareAmt']].mean().plot(ylabel='Average Total Fare', colormap='viridis')
```

Out[74]:

```
<AxesSubplot:xlabel='year_month_travel', ylabel='Average Total Fare'>
```



Here we find that the trends of base fare amount and total fare amount are similar over the period. However, the gap between the 2 changes over time.

Since there are a lot of unknown factors that affect total fare amount e.g. tax, we will rely on base fare amount instead.

In [75]:

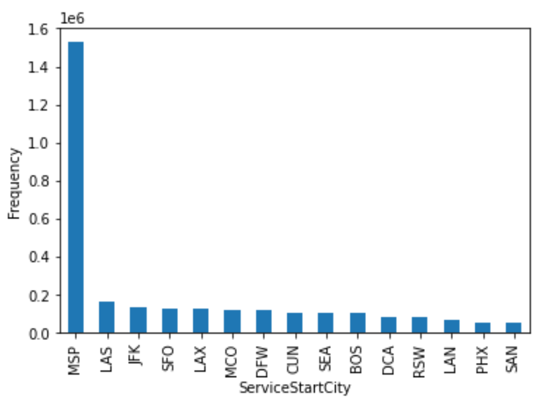
```python
transaction.drop(columns=['TotalDocAmt'], inplace=True)
```

In [204…

```python
# Plot the frequency of service start airport
freq_depart = transaction.groupby('ServiceStartCity')['CustomerId'].count().sort_values(ascending=False)
freq_depart.iloc[:15].plot(kind='bar', ylabel='Frequency')
```

Out[204…

```
<AxesSubplot:xlabel='ServiceStartCity', ylabel='Frequency'>
```



Data aggregation

We also aggregate the transaction-level data to get customer-level data for company the analysis between 2 different granularity.

In [ ]:

```python
# Create a grouping function
def group_func(df):

    member = ((~df.EnrollDate.isna()).sum() > 0).astype(int)

    top_origin = df[df.CouponSeqNbr == 1].ServiceStartCity.mode().values[0] if df[df.CouponSeqNbr == 1].shape[0] != 0 else np.NaN

    temp_df = pd.DataFrame({
    'customer_id': [df.CustomerId],
    'num_tickets': [df.TicketNum.count()],
    'num_PNR': [df.PNRLocatorID.nunique()],
    'top_start_location': [top_origin],
    'avg_book_length': [(df.ServiceStartDate - df.PNRCreateDate).mean().days],
    'gender': [df.GenderCode.mode().values[0]],
    'avg_age': [np.round(df.Age.mean())],
    'is_member': [member],
    'is_elite_member': [df.EliteMember.max() if member else 0],
    'is_standard_member': [df.StandardMember.max() if member else 0],
```

```
    'member_length': [(pd.Timestamp('2014-12-31') - df.EnrollDate.mean()).days if member == 1 else 0],
    'avg_base_fare': [df[df.BaseFareAmt != 0].BaseFareAmt.mean()],
    'avg_discount': [df.Discount.mean()],
    'avg_layover': [df.Layover.mean()],
    'avg_layover_24': df.Over24hrs.mean(),
    'avg_exchange': [df.Exchange.mean()],
    'cardholder': [(df.CardHolder.sum() > 0).astype(int)]}
    )

    sum_df = df.loc[:, 'Airportbooking':'Downgrade'].mean().to_frame().T

    return temp_df.join(sum_df)
```

In [81]:

```
# Perform aggregation. Note that this could take 2-3 hours to run if use the entire data set

# group_df = flight_cleaned.groupby('CustomerId').progress_apply(group_func)
# temp = group_df.reset_index().drop(columns=['level_1', 'customer_id'])

# Or use pre-aggregated data instead
customer = pd.read_csv('groupby_customer.csv', index_col=0)
```

Clustering analysis

Let's define some utility functions

In [134…

```
def plot_sse(X, from_k = 1, to_k = 10):
    kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(X)
                    for k in range(from_k, to_k + 1)]
    inertias = [model.inertia_ for model in kmeans_per_k]

    plt.figure(figsize=(8, 3.5))
    plt.plot(range(1, to_k + 1), inertias, "bo-")
    plt.xlabel("$k$", fontsize=14)
    plt.ylabel("SSE", fontsize=14)
    plt.show()

def plot_silhouette_score_knn(X, from_k = 2, to_k = 10):
    models = [KMeans(n_clusters=k, random_state=42).fit(X)
                    for k in range(from_k, to_k + 1)]
    # Here # of cluster of the models must start from 2
    s_scores = [silhouette_score(X, model.labels_) for model in models]
    k_list = range(2, len(models) + 2)

    plt.figure(figsize=(8, 3.5))
    plt.plot(k_list, s_scores, "ro-")
    plt.xlabel("$k$", fontsize=14)
    plt.ylabel("silhouette_score", fontsize=14)
    plt.show()

def plot_silhouette_score_gmm(X, from_k = 2, to_k = 10):
    models = [GaussianMixture(n_components = k)
                    for k in range(from_k, to_k + 1)]
    # Here # of cluster of the models must start from 2
    s_scores = [silhouette_score(X, model.fit_predict(X)) for model in models]
    k_list = range(from_k, len(models) + 2)

    plt.figure(figsize=(8, 3.5))
    plt.plot(k_list, s_scores, "ro-")
    plt.xlabel("$k$", fontsize=14)
    plt.ylabel("silhouette_score", fontsize=14)
    plt.show()
```

Since the customer-level dataset is large, we will take samples from the dataset instead to perform all the calculation faster.

In [172…

```
customer_percent_sample_size = 2  # Percent of data to sample for customer data

customer_sample = customer.sample(
    n=customer.shape[0] // (100 // customer_percent_sample_size),
    random_state=1)

print(f"Customer sample size = {customer_sample.shape[0]} rows")
```

```
Customer sample size = 30493 rows
```

K-means clustering (customer level)

We first works on customer level dataset wih K-means clustering

In [174…

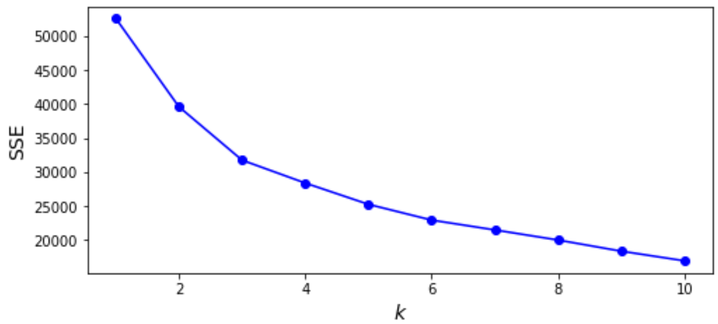
```
# Drop categorical column
customer_sample.drop(columns=['top_start_location'], inplace=True)
```

In [175…

```
# Normalize the dataset
customer_scaler = MinMaxScaler()
customer_scaler.fit(customer_sample.values)

X = customer_scaler.transform(customer_sample)
```

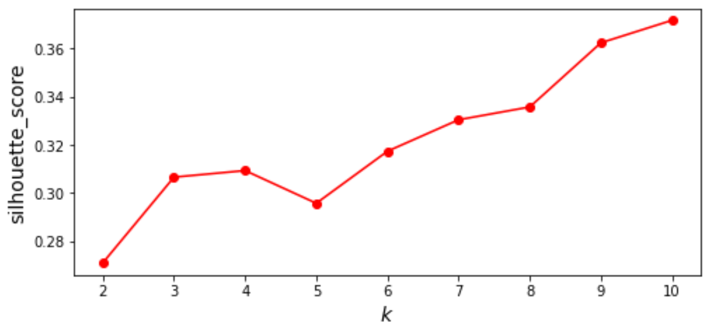
In [105…

```
plot_sse(X)
```



In [101…

```
plot_silhouette_score_knn(X)
```



From the SSE and silhouette score plots, we choose k=3 as it shows a large increase in a silhouette score as well as a large drop of SSE from the previous k. Using higher k is not showing important improvement and will also make it harder to interprete.

In [120…

```
# Run K-means with k=3

k = 3

kmean = KMeans(n_clusters=k)
kmean.fit(X)

# Display cluster centroid

_
centroids = customer_scaler.inverse_transform(kmean.cluster_centers_)
customer_cluster_df = pd.DataFrame(centroids, columns=customer_sample.columns)
customer_cluster_df.round(4)
```

Out[120…

	num_tickets	num_PNR	avg_book_length	gender	avg_age	is_member	is_elite_member	is_standard_member	member_length	avg_base_fare	avg_discou
0	1.9997	1.1075	53.3885	0.5052	37.9814	0.0002	0.0002	0.0000	0.0770	277.6814	0.194
1	1.9771	1.1474	58.2936	0.5386	39.2836	0.0000	0.0000	0.0000	-0.0000	266.1775	0.444
2	2.8080	1.6365	66.2244	0.5328	44.0768	1.0000	0.0045	0.9957	1241.1428	296.3824	0.377

In [141…

```
# Ratio of total members of each cluster
pd.Series(kmean.labels_).value_counts() / kmean.labels_.shape[0]
```

Out[141…

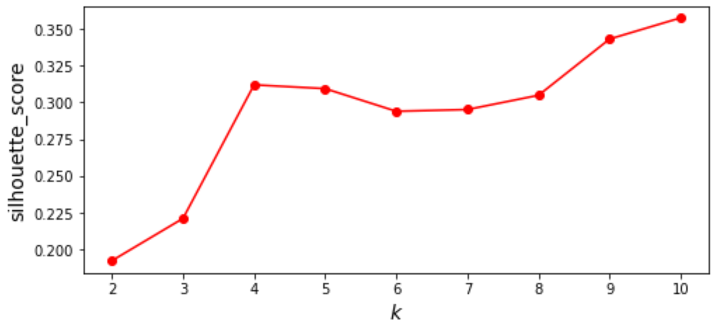
```
1    0.411799
0    0.411373
2    0.176827
dtype: float64
```

Gaussian Mixture Model (customer level)

We try using GMM to do clustering analysis and compare the result with kmeans. We start with plotting

In [136...

```
plot_silhouette_score_gmm(X)
```



In [198...

```
# Run GMM clustering with k = 4

num_clusters = 4  # Explore this

cls = GaussianMixture(n_components = num_clusters, random_state=10)
cls_assignment = cls.fit_predict(X)


# Show cluster centroids
gm_cust_df = customer_sample.copy()

gm_cust_df['cls'] = cls_assignment
gm_cust_df.groupby('cls').mean().round(4)
```

Out[198...

cls	num_tickets	num_PNR	avg_book_length	gender	avg_age	is_member	is_elite_member	is_standard_member	member_length	avg_base_fare	avg_disco
0	2.2420	1.2152	58.1581	0.5220	38.3175	0.0	0.0000	0.0000	0.0000	217.8825	0.9
1	1.8920	1.1106	60.3024	0.5370	39.3205	0.0	0.0000	0.0000	0.0000	286.6933	0.3
2	1.9422	1.0963	50.8821	0.5090	38.1965	0.0	0.0000	0.0000	0.0000	285.7849	0.0
3	2.8159	1.6413	66.2067	0.5326	44.0769	1.0	0.0048	0.9954	1240.8617	296.4463	0.3

In [199...

```
# Ratio of total members of each cluster
gm_cust_df['cls'].value_counts() / gm_cust_df['cls'].shape[0]
```

Out[199...

```
2    0.350244
1    0.300659
3    0.176893
0    0.172203
Name: cls, dtype: float64
```

Kmeans clustering (transaction level)

Since the transactional-level dataset is large, we will take samples from the dataset instead to perform all the calculation faster.

In [145...

```
transaction_percent_sample_size = 1  # Percent of data to sample for transaction data

transaction_sample = transaction.sample(
    n=transaction.shape[0] // (100 // transaction_percent_sample_size),
    random_state=1)

print(f"Transaction sample size = {transaction_sample.shape[0]} rows")
```

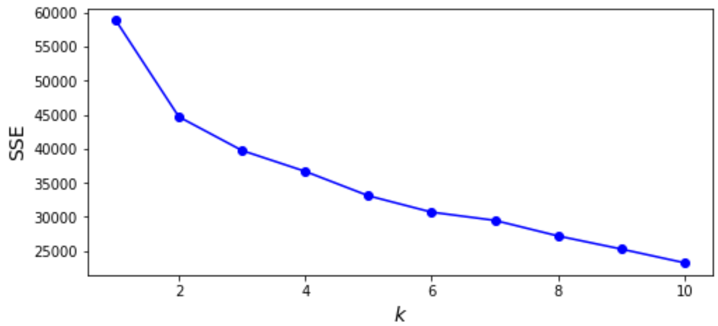
```
Transaction sample size = 32524 rows
```

In [146...

```
# Get only numerical columns and drop irrelevant columns such as date
cols = ['GenderCode','Age','CardHolder','EliteMember', 'StandardMember', 'Discount',
        'Layover', 'Exchange', 'Airportbooking', 'OutsideBooking', 'ReservationsBooking',
        'SCAWebsiteBooking','SYVacation', 'TourOperatorPortal','BookedCoach','BookedDiscountFirstClass',
        'BookedFirstClass', 'TraveledCoach','TraveledDiscountFirstClass', 'TraveledFirstClass', 'Upgrade']

transaction_sample_filtered = transaction_sample[cols]


# Normalize the dataset
transaction_scaler = MinMaxScaler()
transaction_scaler.fit(transaction_sample_filtered.values)

X = transaction_scaler.transform(transaction_sample_filtered)
```

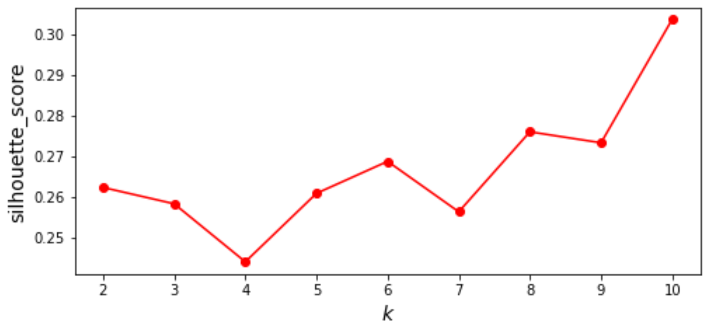
In [147...

```
plot_sse(X)
```



In [148...

```
plot_silhouette_score_knn(X)
```



Here 3 seems to be the best choice for clustering as the silhouette_score goes down significanly after this and SSE also goes down only linearly. Although 2 seems like a better choice but it might be too few for a number of clusters in this scenario

In [152...

```
# Run K-means with k=3

k = 3

kmean = KMeans(n_clusters=k)
kmean.fit(X)

# Display cluster centroid

_
centroids = transaction_scaler.inverse_transform(kmean.cluster_centers_)
transaction_cluster_df = pd.DataFrame(centroids, columns=transaction_sample_filtered.columns)
transaction_cluster_df.round(4)
```

Out[152...

	GenderCode	Age	CardHolder	EliteMember	StandardMember	Discount	Layover	Exchange	Airportbooking	OutsideBooking	ReservationsBooking
0	0.5052	38.3718	0.0028	0.0018	0.0958	0.2085	0.6050	0.0921	0.0007	0.9777	0.0160
1	0.5364	42.1710	0.0158	0.0045	0.3057	-0.0000	0.4001	0.0017	0.0041	-0.0000	0.0642
2	0.5202	41.8076	0.0197	0.0081	0.2835	0.9994	0.4091	0.0024	0.0105	0.0015	0.0899

In [153...

```
# Ratio of total members of each cluster
pd.Series(kmean.labels_).value_counts() / kmean.labels_.shape[0]
```

Out[153...

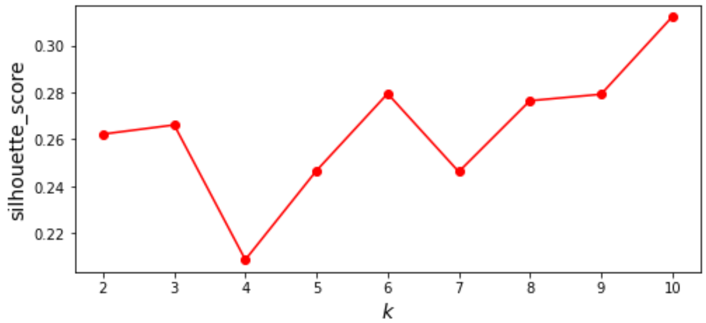
```
0    0.453757
1    0.304145
2    0.242098
dtype: float64
```

Gaussian Mixture Model (transaction level)

We start with choosing k from plotting silhouette_score.

In [154...

```
plot_silhouette_score_gmm(X)
```



k=3 is a good choice here because of the drop of the scores after this. Althouh there are higher k that could yield higher silhouette_score, there are too many clusters to be practical.

In [161…

```
# X_all = MinMaxScaler().fit_transform(transaction[cols].values)
X_all = transaction[cols].values
```

In [159…

```
# Run GMM clustering with k = 3

num_clusters = 3  # Explore this

cls = GaussianMixture(n_components = num_clusters)
cls_assignment = cls.fit_predict(X_all)


# Show cluster centroids
gm_df = transaction[cols].copy()

gm_df['cls'] = cls_assignment
gm_df.groupby('cls').mean().round(4)
```

Out[159…

cls	GenderCode	Age	CardHolder	EliteMember	StandardMember	Discount	Layover	Exchange	Airportbooking	OutsideBooking	ReservationsBooking
0	0.5071	38.8248	0.0041	0.0023	0.1115	0.0000	0.5759	0.0137	0.0036	0.9120	0.0709
1	0.5319	41.9937	0.0184	0.0071	0.3185	0.3644	0.4084	0.0019	0.0000	0.0000	0.0000
2	0.5249	38.9741	0.0055	0.0018	0.1168	0.9995	0.5581	0.2001	0.0118	0.5262	0.1253

In [162…

```
# Run GMM clustering with k = 3

num_clusters = 3  # Explore this

cls = GaussianMixture(n_components = num_clusters)
cls_assignment = cls.fit_predict(X_all)


# Show cluster centroids
gm_df = transaction[cols].copy()

gm_df['cls'] = cls_assignment
gm_df.groupby('cls').mean().round(4)
```

Out[162…

cls	GenderCode	Age	CardHolder	EliteMember	StandardMember	Discount	Layover	Exchange	Airportbooking	OutsideBooking	ReservationsBooking
0	0.4560	49.4144	0.1595	0.0647	0.4853	0.5708	0.4283	0.0138	0.0524	0.2067	0.0907
1	0.5261	42.1116	0.0000	0.0000	0.1518	0.7716	0.4800	0.0212	0.0000	0.1202	0.0000
2	0.5258	39.3365	0.0000	0.0000	0.1862	0.2786	0.5067	0.0463	0.0000	0.4930	0.0512

In [165…

```
gm_df['cls'].value_counts()
```

Out[165…

```
2    2762115
1     273476
0     216875
Name: cls, dtype: int64
```

Choosen model

The clustering results from transaction level do not show a distinct seperation between each cluster. However, the results from customer level analysis shows a clear difference between clusters.

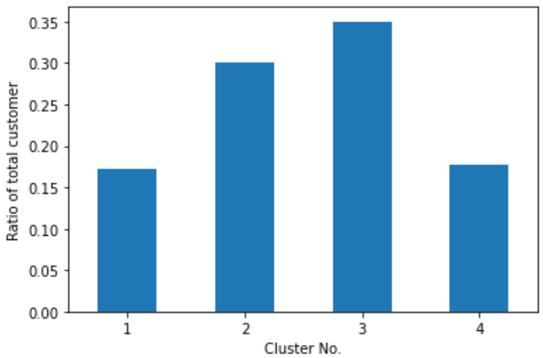
From the customer level analysis, the GMM model with k=4 gives the best result based on silhouette_score increase. It also returns a clustering result that is practical for business implementation.

In [200…

```
cluster_ratio = (gm_cust_df['cls'].value_counts() / gm_cust_df['cls'].shape[0]).sort_index()
cluster_ratio.index = [1,2,3,4]
cluster_ratio.plot(kind='bar', rot=0, xlabel='Cluster No.', ylabel='Ratio of total customer')
```

Out[200…

```
<AxesSubplot:xlabel='Cluster No.', ylabel='Ratio of total customer'>
```



In [201…

```
gm_cust_df.groupby('cls').mean().round(4)
```

Out[201…

cls	num_tickets	num_PNR	avg_book_length	gender	avg_age	is_member	is_elite_member	is_standard_member	member_length	avg_base_fare	avg_disco
0	2.2420	1.2152	58.1581	0.5220	38.3175	0.0	0.0000	0.0000	0.0000	217.8825	0.9
1	1.8920	1.1106	60.3024	0.5370	39.3205	0.0	0.0000	0.0000	0.0000	286.6933	0.3
2	1.9422	1.0963	50.8821	0.5090	38.1965	0.0	0.0000	0.0000	0.0000	285.7849	0.0
3	2.8159	1.6413	66.2067	0.5326	44.0769	1.0	0.0048	0.9954	1240.8617	296.4463	0.3

In [202…

```
gm_cust_df.groupby('cls').mean().round(4).to_csv('final_cluster.csv')
```

We will discuss more about how to utilize this clustering result in the presentation.