Assignment 5 - Kaggle Competition and Unsupervised Learning

Nick Carroll

Netid: nc230

Note: this assignment falls under collaboration Mode 2: Individual Assignment – Collaboration Permitted. Please refer to the syllabus for additional information.

Instructions for all assignments can be found here, and is also linked to from the course syllabus.

Total points in the assignment add up to 90; an additional 10 points are allocated to presentation quality.

Learning objectives

Through completing this assignment you will be able to...

1. Apply the full supervised machine learning pipeline of preprocessing, model selection, model performance evaluation and comparison, and model application to a real-world scale dataset
2. Apply clustering techniques to a variety of datasets with diverse distributional properties, gaining an understanding of their strengths and weaknesses and how to tune model parameters
3. Apply PCA and t-SNE for performing dimensionality reduction and data visualization

1

[40 points] Kaggle Classification Competition

You've learned a great deal about supervised learning and now it's time to bring together all that you've learned. You will be competing in a Kaggle Competition along with the rest of the class! Your goal is to predict hotel reservation cancellations based on a number of potentially related factors such as lead time on the booking, time of year, type of room, special requests made, number of children, etc. While you will be asked to take certain steps along the way to your submission, you're encouraged to try creative solutions to this problem and your choices are wide open for you to make your decisions on how to best make the predictions.

IMPORTANT: Follow the link posted on Ed to register for the competition

You can view the public leaderboard anytime here

The Data. The dataset is provided as `a5_q1.pkl` which is a pickle file format, which allows you to load the data directly using the code below; the data can be downloaded from the Kaggle competition website. A data dictionary for the project can be found here and the original paper that describes the dataset can be found here. When you load the data, 5 matrices are provided `X_train_original`, `y_train`, and `X_test_original`, which are the original, unprocessed features and labels for the training set and the test features (the test labels are not provided - that's what you're predicting). Additionally, `X_train_ohe` and `X_test_ohe` are provided which are one-hot-encoded (OHE) versions of the data. The OHE versions OHE processed every categorical variable. This is provided for convenience if you find it helpful, but you're welcome to reprocess the original data other ways if your prefer.

Scoring. You will need to achieve a minimum acceptable level of performance to demonstrate proficiency with using these supervised learning techniques. Beyond that, it's an open competition and scoring in the top three places of the *private leaderboard* will result in **5 bonus points in this assignment** (and the pride of the class!). Note: the Kaggle leaderboard has a public and private component. The public component is viewable throughout the competition, but the private leaderboard is revealed at the end. When you make a submission, you immediately see your submission on the public leaderboard, but that only represents scoring on a fraction of the total collection of test data, the rest remains hidden until the end of the competition to prevent overfitting to the test data through repeated submissions. You will be be allowed to hand-select two eligible submissions for private score, or by default your best two public scoring submissions will be selected for private scoring.

Requirements:

(a) Explore your data. Review and understand your data. Look at it; read up on what the features represent; think through the application domain; visualize statistics from the paper data to understand any key relationships. **There is no output required for this question**, but you are encouraged to explore the data personally before going further.

(b) Preprocess your data. Preprocess your data so it's ready for use for classification and describe what you did and why you did it. Preprocessing may include: normalizing data, handling missing or erroneous values, separating out a validation dataset, preparing categorical variables through one-hot-encoding, etc. To make one step in this process easier, you're provided with a one-hot-encoded version of the data already.

- Comment on each type of preprocessing that you apply and both how and why you apply it.

(c) Select, train, and compare models. Fit at least 5 models to the data. Some of these can be experiments with different hyperparameter-tuned versions of the same model, although all 5 should not be the same type of model. There are no constraints on the types of models, but you're encouraged to explore examples we've discussed in class including:

1. Logistic regression
2. K-nearest neighbors
3. Random Forests

4. Neural networks
5. Support Vector Machines
6. Ensembles of models (e.g. model bagging, boosting, or stacking). `Scikit-learn` offers a number of tools for assisting with this including those for bagging, boosting, and stacking. You're also welcome to explore options beyond the `sklean` universe; for example, some of you may have heard of XGBoost which is a very fast implementation of gradient boosted decision trees that also allows for parallelization.

When selecting models, be aware that some models may take far longer than others to train. Monitor your output and plan your time accordingly.

Assess the classification performance AND computational efficiency of the models you selected:

- Plot the ROC curves and PR curves for your models in two plots: one of ROC curves and one of PR curves. For each of these two plots, compare the performance of the models you selected above and trained on the training data, evaluating them on the validation data. Be sure to plot the line representing random guessing on each plot. You should plot all of the model's ROC curves on a single plot and the PR curves on a single plot. One of the models should also be your BEST performing submission on the Kaggle public leaderboard (see below). In the legends of each, include the area under the curve for each model (limit to 3 significant figures). For the ROC curve, this is the AUC; for the PR curve, this is the average precision (AP).
- As you train and validate each model time how long it takes to train and validate in each case and create a plot that shows both the training and prediction time for each model included in the ROC and PR curves.
- Describe:
  - Your process of model selection and hyperparameter tuning
  - Which model performed best and your process for identifying/selecting it

(d) Apply your model "in practice". Make *at least* 5 submissions of different model results to the competition (more submissions are encouraged and you can submit up to 5 per day!). These do not need to be the same that you report on above, but you should select your *most competitive* models.

- Produce submissions by applying your model on the test data.
- Be sure to RETRAIN YOUR MODEL ON ALL LABELED TRAINING AND VALIDATION DATA before making your predictions on the test data for submission. This will help to maximize your performance on the test data.
- In order to get full credit on this problem you must achieve an AUC on the Kaggle public leaderboard above the "Benchmark" score on the public leaderboard.

Guidance:

1. **Preprocessing**. You may need to preprocess the data for some of these models to perform well (scaling inputs or reducing dimensionality). Some of this preprocessing may differ from model to model to achieve the best performance. A helpful tool for creating such preprocessing and model fitting pipelines is the sklearn `pipeline` module which lets you group a series of processing steps together.
2. **Hyperparameters**. Hyperparameters may need to be tuned for some of the model you use. You may want to perform hyperparameter tuning for some of the models. If you experiment with different hyperparameters that include many model runs, you may want to apply them to a small subsample of your overall data before running it on the larger training set to be time efficient (if you do, just make sure to ensure your selected subset is representative of the rest of your data).
3. **Validation data**. You're encouraged to create your own validation dataset for comparing model performance; without this, there's a significant likelihood of overfitting to the data. A common choice of the split is 80% training, 20% validation. Before you make your final predictions on the test data, be sure to retrain your model on the entire dataset.
4. **Training time**. This is a larger dataset than you've worked with previously in this class, so training times may be higher that what you've experienced in the past. Plan ahead and get your model pipeline working early so you can experiment with the models you use for this problem and have time to let them run.

Starter code

Below is some code for (1) loading the data and (2) once you have predictions in the form of confidence scores for those classifiers, to produce submission files for Kaggle.

In [ ]:
```
import pandas as pd
import numpy as np
import pickle

################################
# Load the data
################################
data = pickle.load( open( "./data/a5_q1.pkl", "rb" ) )

y_train = data['y_train']
X_train_original = data['X_train'] # Original dataset
X_train_ohe = data['X_train_ohe']  # One-hot-encoded dataset

X_test_original = data['X_test']
X_test_ohe = data['X_test_ohe']

################################
# Produce submission
################################

def create_submission(confidence_scores, save_path):
    '''Creates an output file of submissions for Kaggle

    Parameters
    ----------
    confidence_scores : list or numpy array
        Confidence scores (from predict_proba methods from classifiers) or
        binary predictions (only recommended in cases when predict_proba is
        not available)
```

```
    save_path : string
        File path for where to save the submission file.

    Example:
    create_submission(my_confidence_scores, './data/submission.csv')

    '''
    import pandas as pd

    submission = pd.DataFrame({"score":confidence_scores})
    submission.to_csv(save_path, index_label="id")
```

ANSWER

Preliminary Exploration of Data

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
X_train_original.head()
```

Out[ ]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nig
0	Resort Hotel	342	2015	July	27	1	0	
2	Resort Hotel	7	2015	July	27	1	0	
3	Resort Hotel	13	2015	July	27	1	0	
4	Resort Hotel	14	2015	July	27	1	0	
5	Resort Hotel	14	2015	July	27	1	0	

5 rows × 29 columns

In [ ]:
```
print(f"The shape of the original training dataset is {X_train_original.shape}, and the shape of the one-hot-encoded training data
```

```
The shape of the original training dataset is (95512, 29), and the shape of the one-hot-encoded training dataset is (95512, 940).
```

In [ ]:
```
pd.DataFrame({'Mean': X_train_original.mean(), 'Standard Deviation': X_train_original.std()})
```

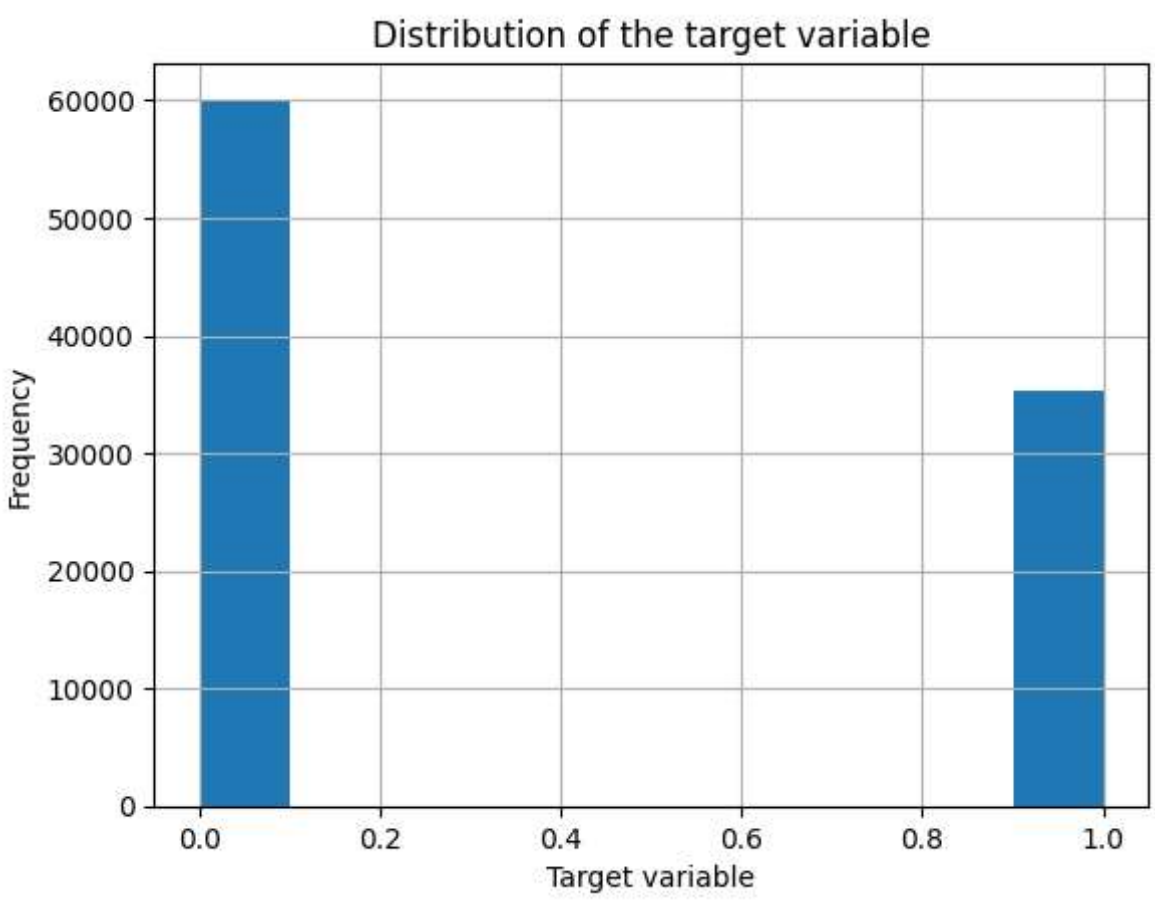
Out[ ]:

	Mean	Standard Deviation
lead_time	103.849768	106.722804
arrival_date_year	2016.157205	0.707470
arrival_date_week_number	27.152902	13.611204
arrival_date_day_of_month	15.823038	8.786777
stays_in_weekend_nights	0.928491	0.999940
stays_in_week_nights	2.503288	1.918017
adults	1.855746	0.596925
children	0.103696	0.397763
babies	0.007748	0.093348
is_repeated_guest	0.031598	0.174929
previous_cancellations	0.087235	0.844491
previous_bookings_not_canceled	0.140035	1.532968
booking_changes	0.220621	0.653900
agent	86.893778	110.839209
company	188.237117	131.459182
days_in_waiting_list	2.316348	17.651287
adr	101.679313	50.906371
required_car_parking_spaces	0.062673	0.246274
total_of_special_requests	0.571310	0.792712

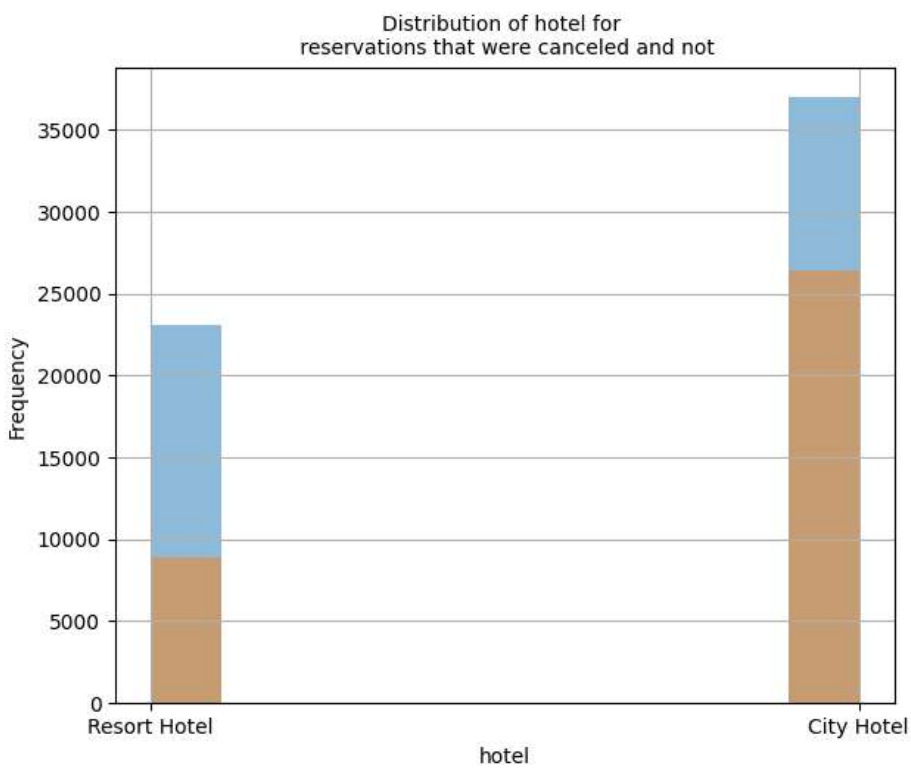
Plotting of Data

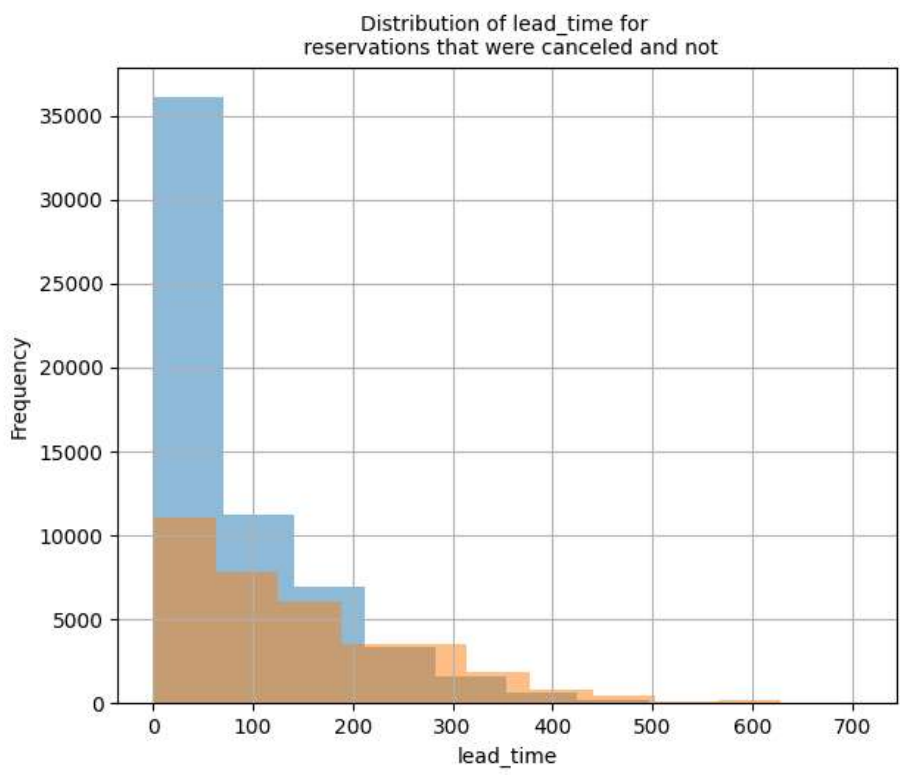
In [ ]:
```
import matplotlib.pyplot as plt

y_train.hist()
plt.title("Distribution of the target variable")
plt.xlabel("Target variable")
plt.ylabel("Frequency")
```

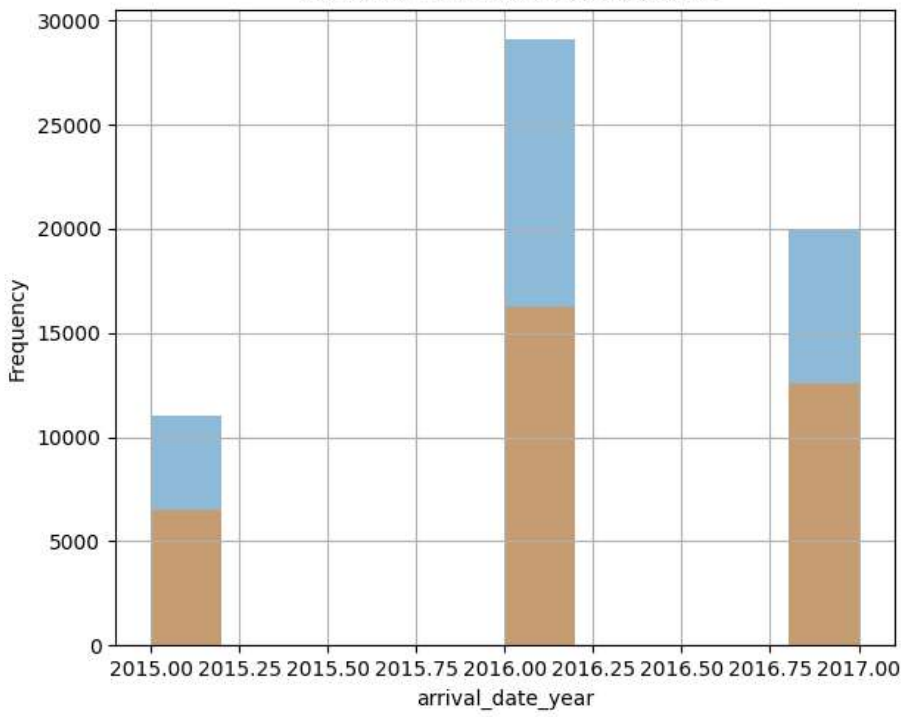
Out[ ]:
```
Text(0, 0.5, 'Frequency')
```

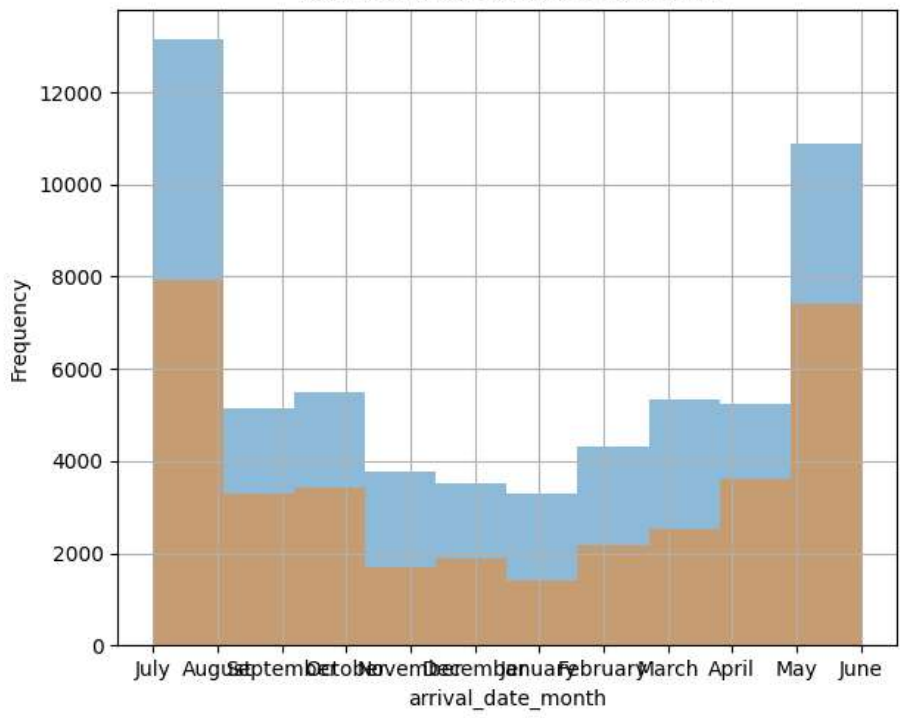


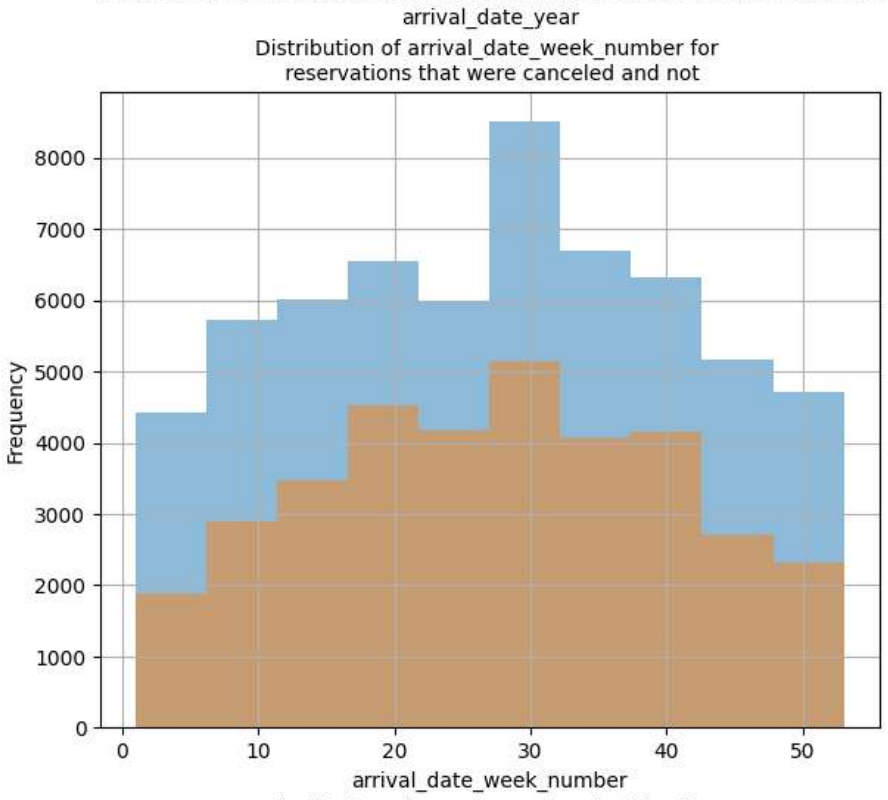
In [ ]:
```
fig, axs = plt.subplots((X_train_original.shape[1] + 1) // 2, 2, figsize = (15, 100))

concatData = pd.concat((X_train_original, y_train), axis = 1)

for i, eachColumn in enumerate(X_train_original.columns):
    for is_canceled in range(2):
        # axs[i // 2][i % 2].hist(concatData.query(f'is_canceled == {is_canceled}')[eachColumn], alpha = 0.5, label = f'is_cancele
        concatData.query(f'is_canceled == {is_canceled}')[eachColumn].hist(ax = axs[i // 2][i % 2], alpha = 0.5, label = f'is_canc
        axs[i // 2][i % 2].set_title(f"Distribution of {eachColumn} for \n reservations that were canceled and not",  fontsize = 1
        axs[i // 2][i % 2].set_xlabel(eachColumn, fontsize = 10)
        axs[i // 2][i % 2].set_ylabel("Frequency", fontsize = 10)
```

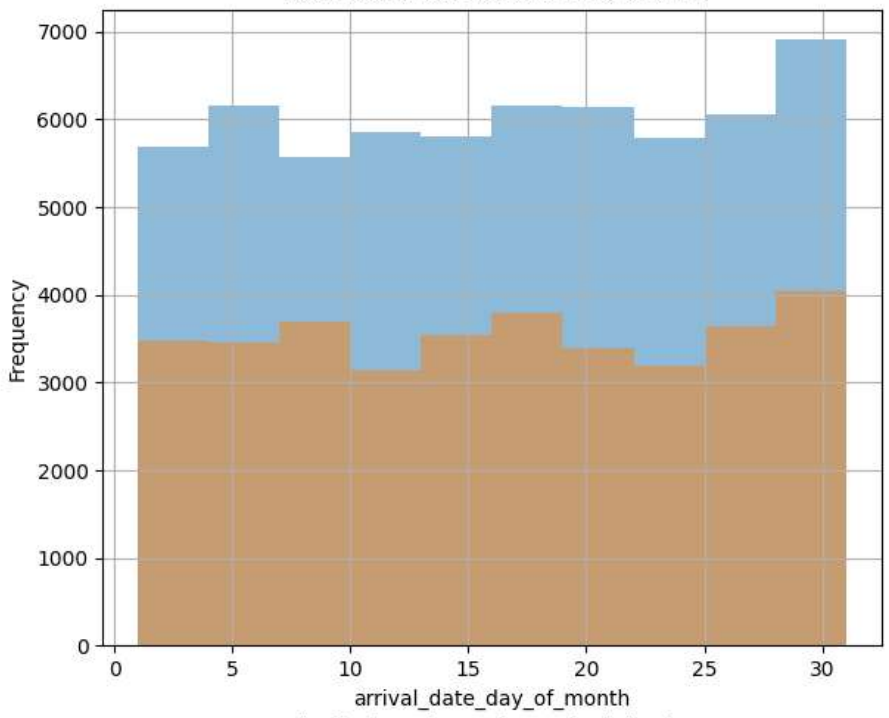

Distribution of hotel for
reservations that were canceled and not
Frequency
0
5000
10000
15000
20000
25000
30000
35000
Resort Hotel
City Hotel
hotel

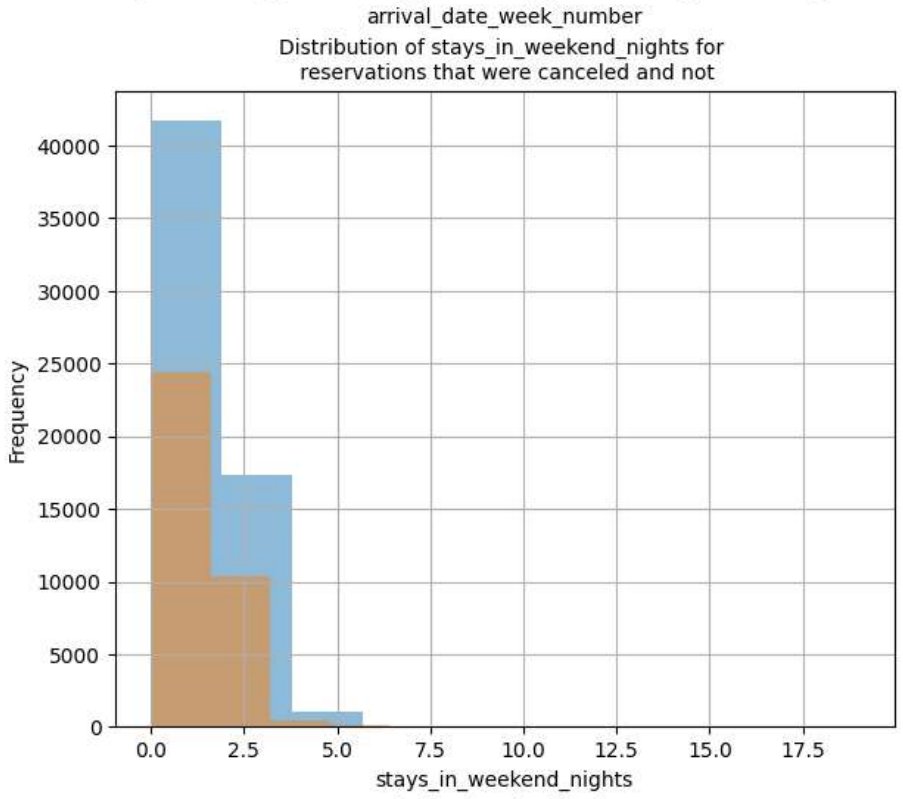

Distribution of lead_time for
reservations that were canceled and not
Frequency
0
5000
10000
15000
20000
25000
30000
35000
0
100
200
300
400
500
600
700
lead_time

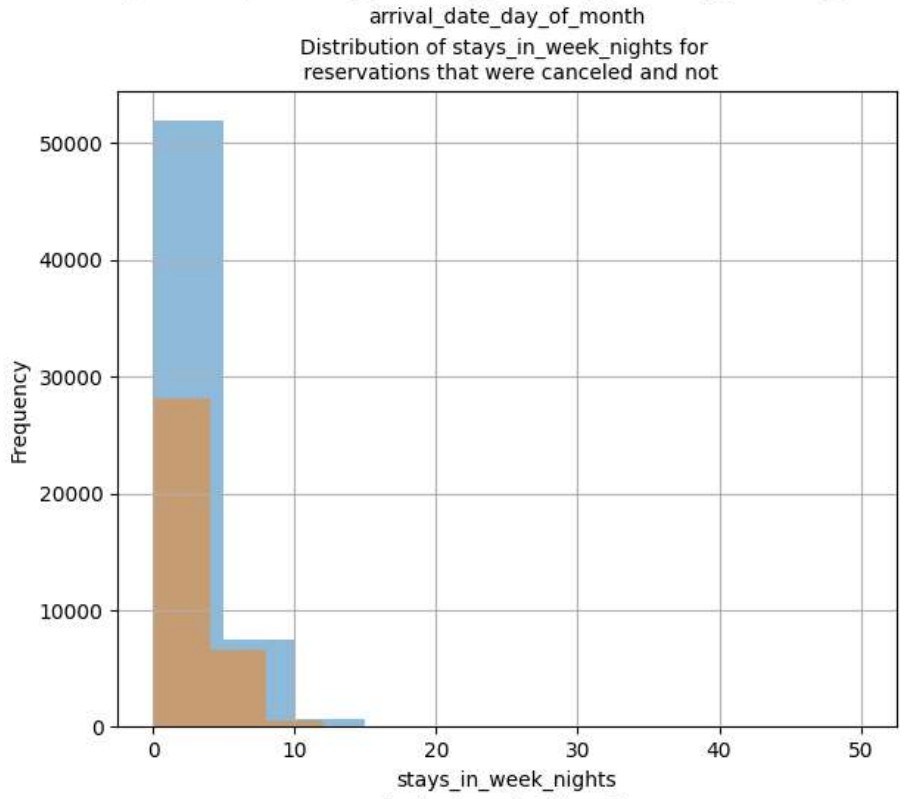

Distribution of arrival_date_year for
reservations that were canceled and not
Frequency
0
5000
10000
15000
20000
25000
30000
2015.00
2015.25
2015.50
2015.75
2016.00
2016.25
2016.50
2016.75
2017.00
arrival_date_year


Distribution of arrival_date_month for
reservations that were canceled and not
Frequency
0
2000
4000
6000
8000
10000
12000
July
August
September
October
November
December
January
February
March
April
May
June
arrival_date_month


Distribution of arrival_date_week_number for
reservations that were canceled and not
Frequency
0
1000
2000
3000
4000
5000
6000
7000
8000
0
10
20
30
40
50
arrival_date_week_number

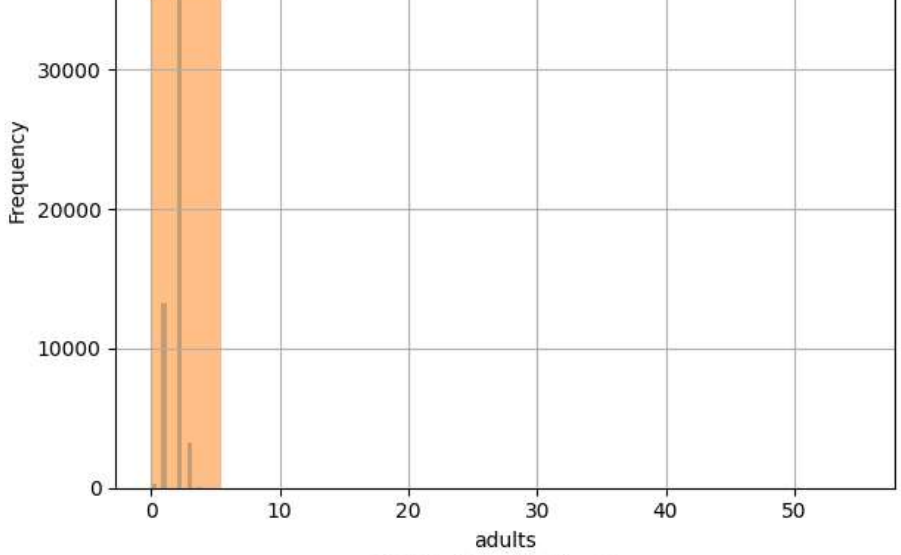

Distribution of arrival_date_day_of_month for
reservations that were canceled and not
Frequency
0
1000
2000
3000
4000
5000
6000
7000
0
5
10
15
20
25
30
arrival_date_day_of_month

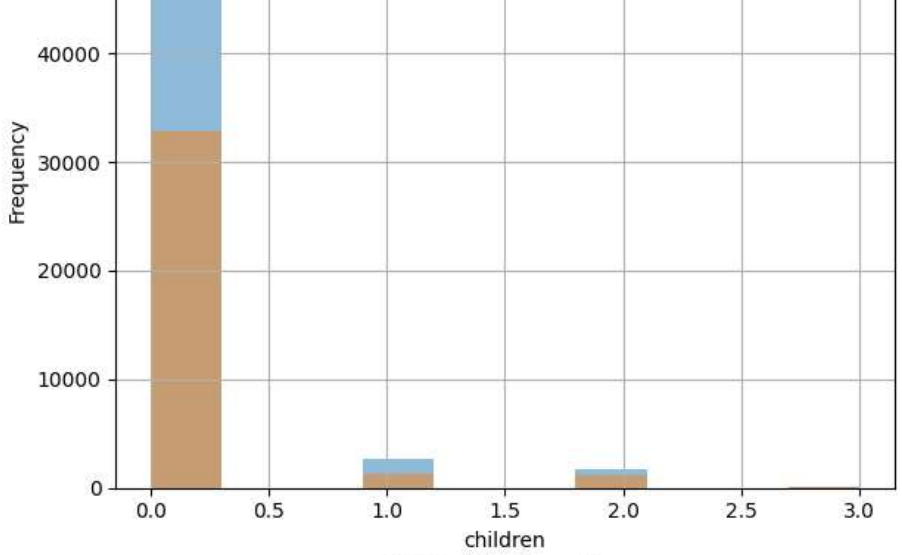

Distribution of stays_in_weekend_nights for
reservations that were canceled and not
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
stays_in_weekend_nights

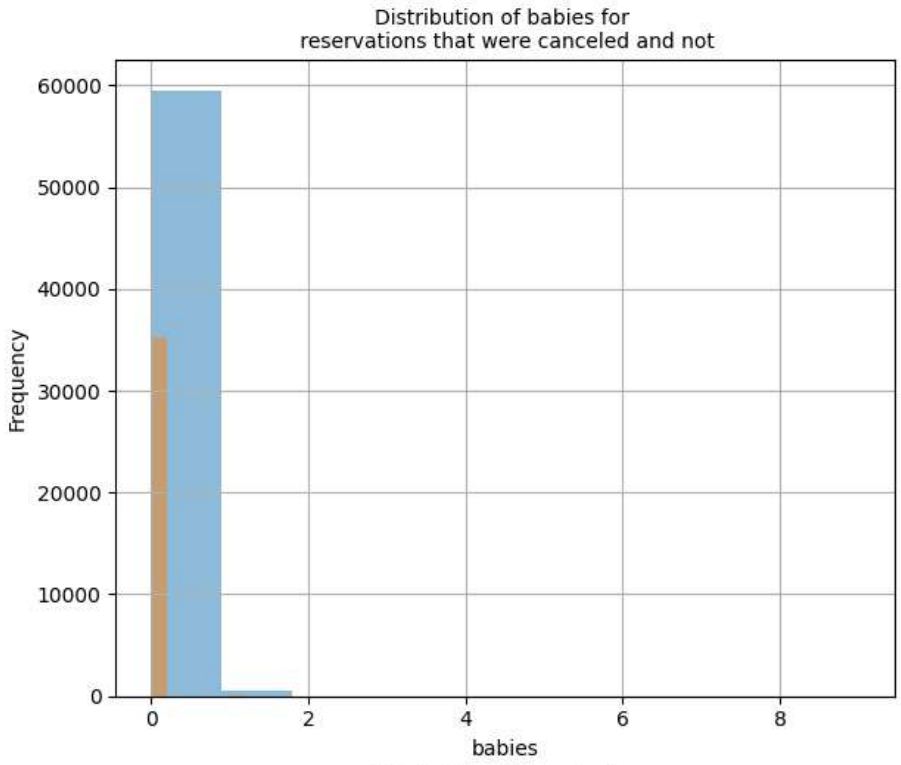

Distribution of stays_in_week_nights for
reservations that were canceled and not
Frequency
0
10000
20000
30000
40000
50000
0
10
20
30
40
50
stays_in_week_nights

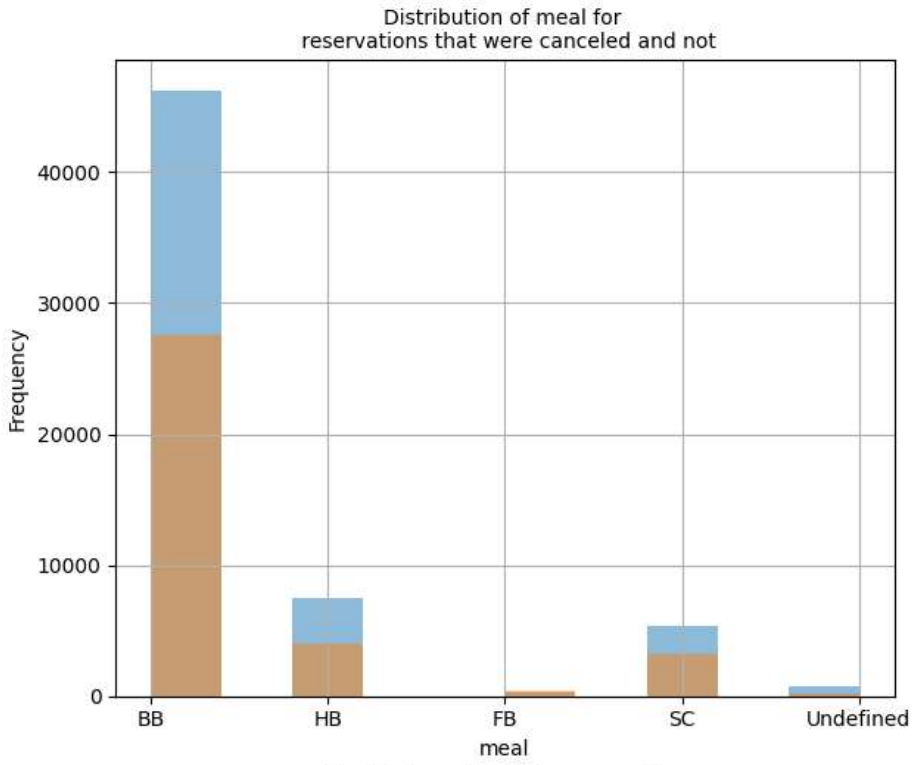
Distribution of adults for
reservations that were canceled and not
40000

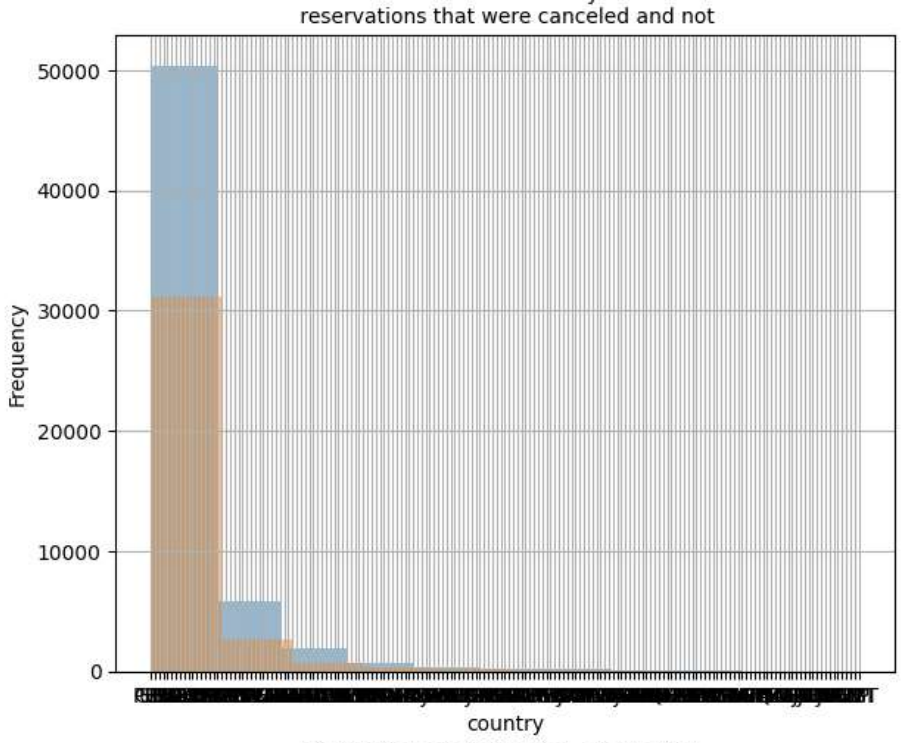
Distribution of children for
reservations that were canceled and not
50000

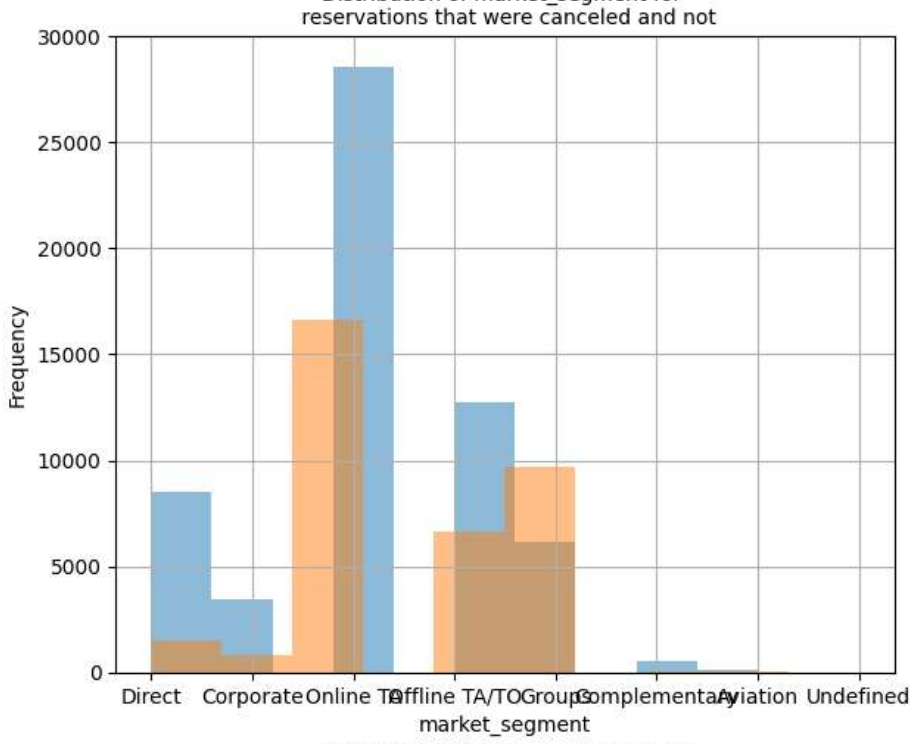

Frequency
adults

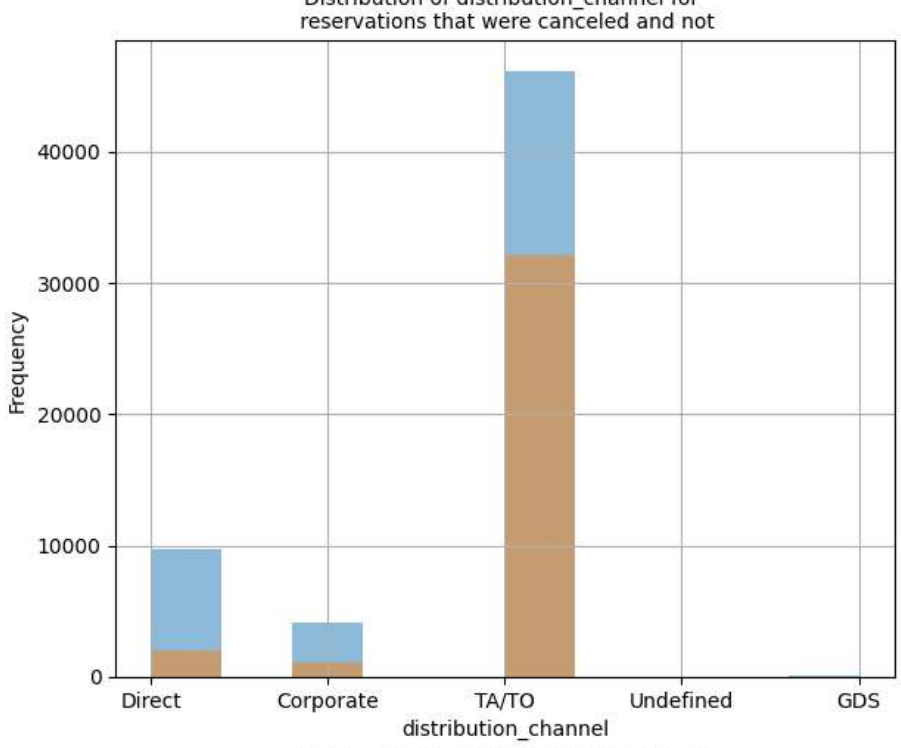

Frequency
children

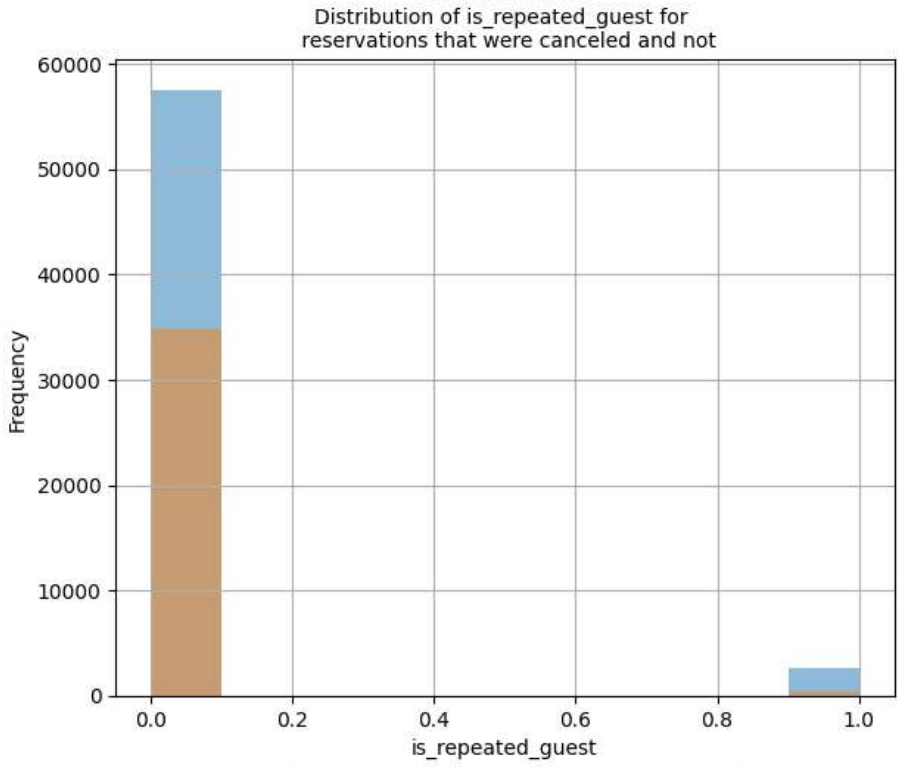

Distribution of babies for
reservations that were canceled and not
Frequency
babies


Distribution of meal for
reservations that were canceled and not
Frequency
BB
HB
FB
SC
Undefined
meal


Distribution of country for
reservations that were canceled and not
Frequency
country

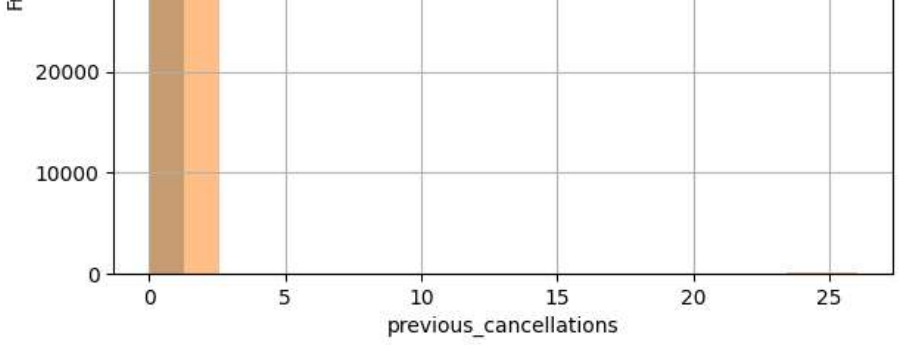

Distribution of market_segment for
reservations that were canceled and not
Frequency
Direct
Corporate
Online TA
Offline TA/TO
Groups
Complementary
Aviation
Undefined
market_segment


Distribution of distribution_channel for
reservations that were canceled and not
Frequency
Direct
Corporate
TA/TO
Undefined
GDS
distribution_channel

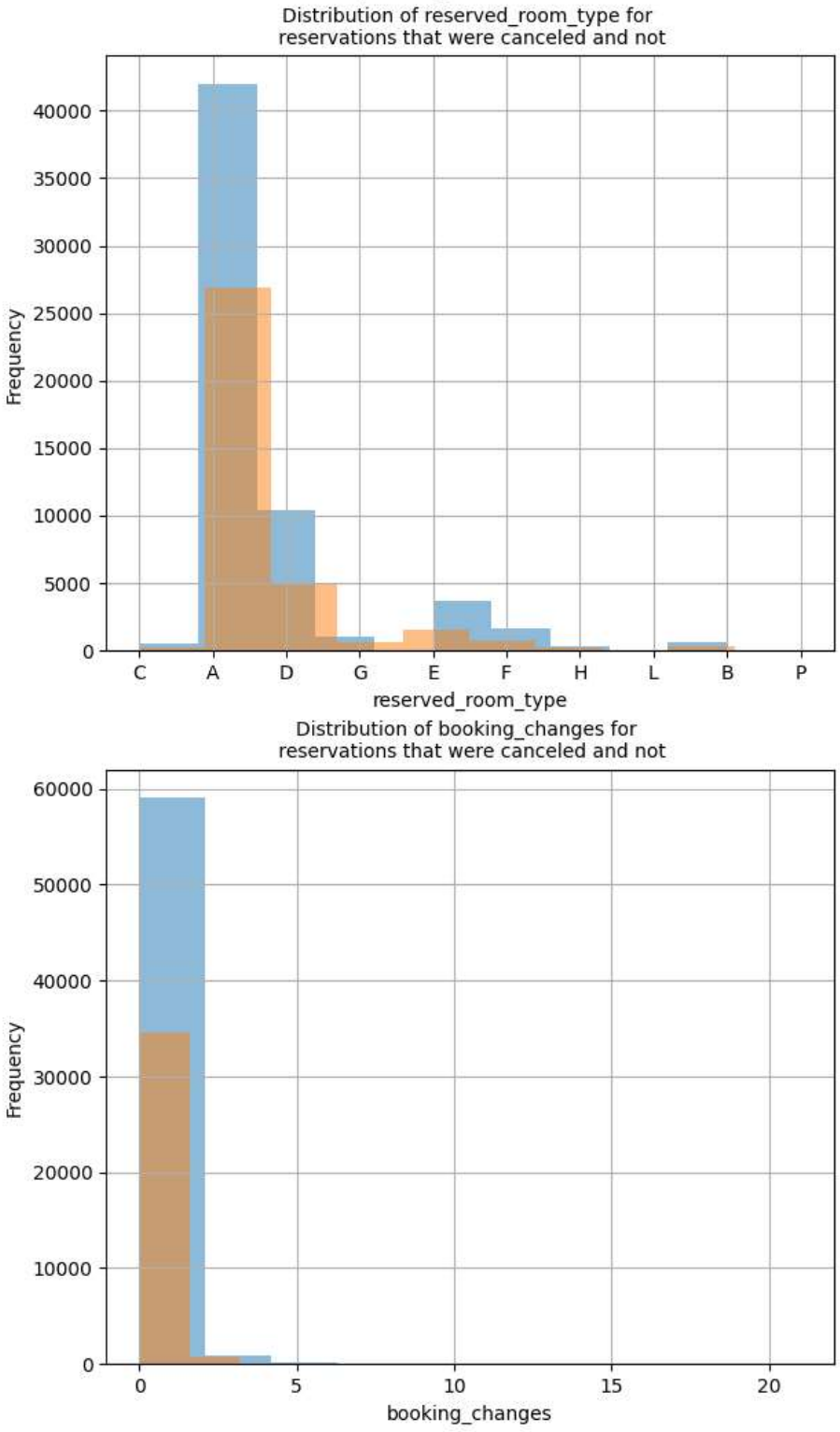

Distribution of is_repeated_guest for
reservations that were canceled and not
Frequency
is_repeated_guest

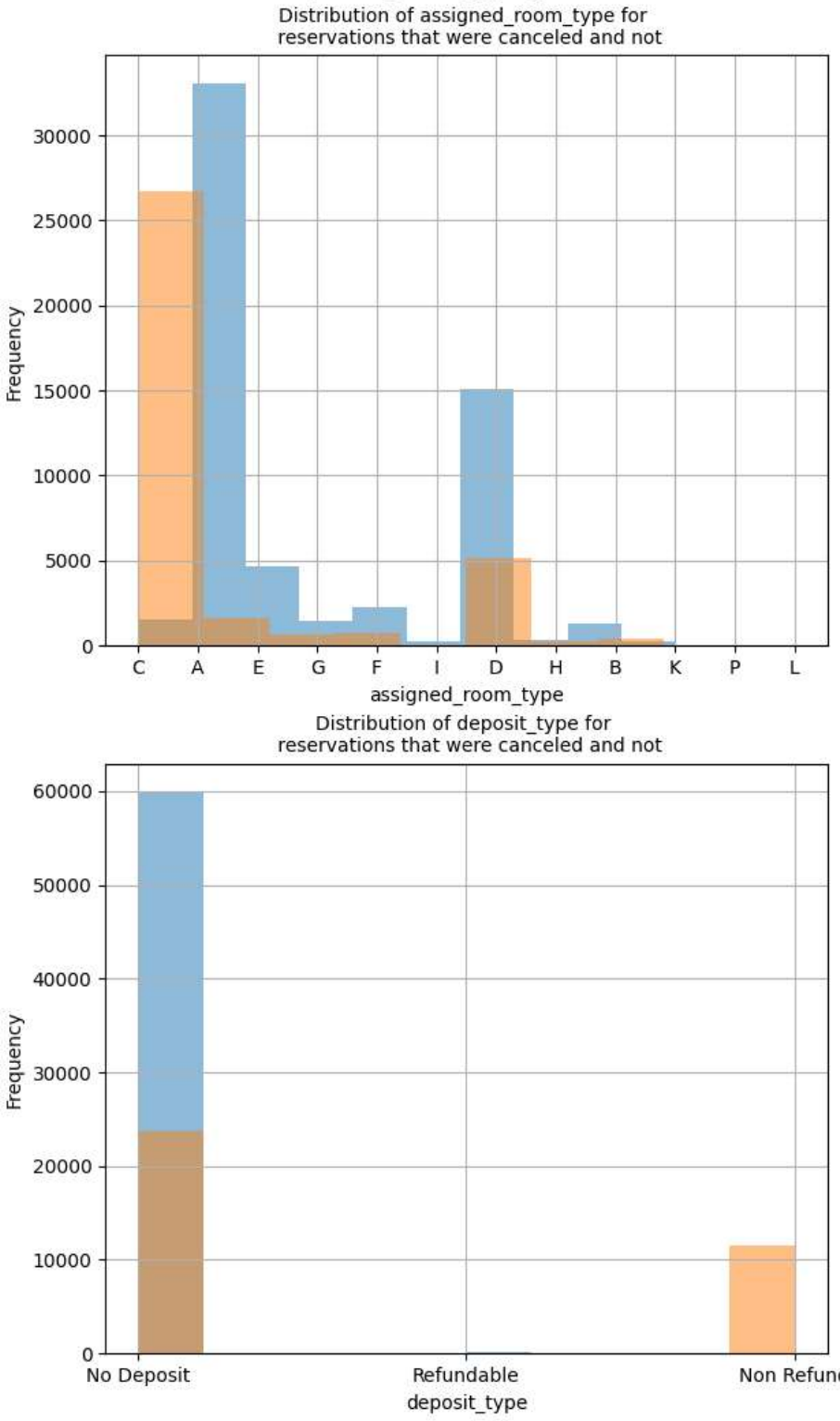
Distribution of previous_cancellations for
reservations that were canceled and not
Frequency

Distribution of previous_bookings_not_canceled for
reservations that were canceled and not
Frequency


Frequency
0
10000
20000
0
5
10
15
20
25
previous_cancellations

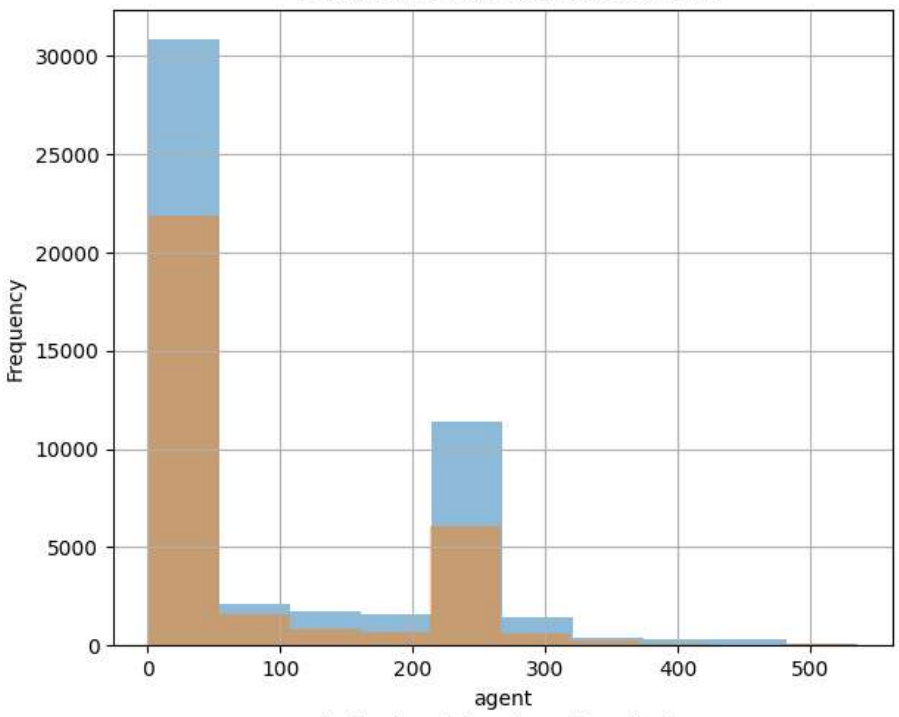

Frequency
0
10000
20000
0
10
20
30
40
50
60
70
previous_bookings_not_canceled

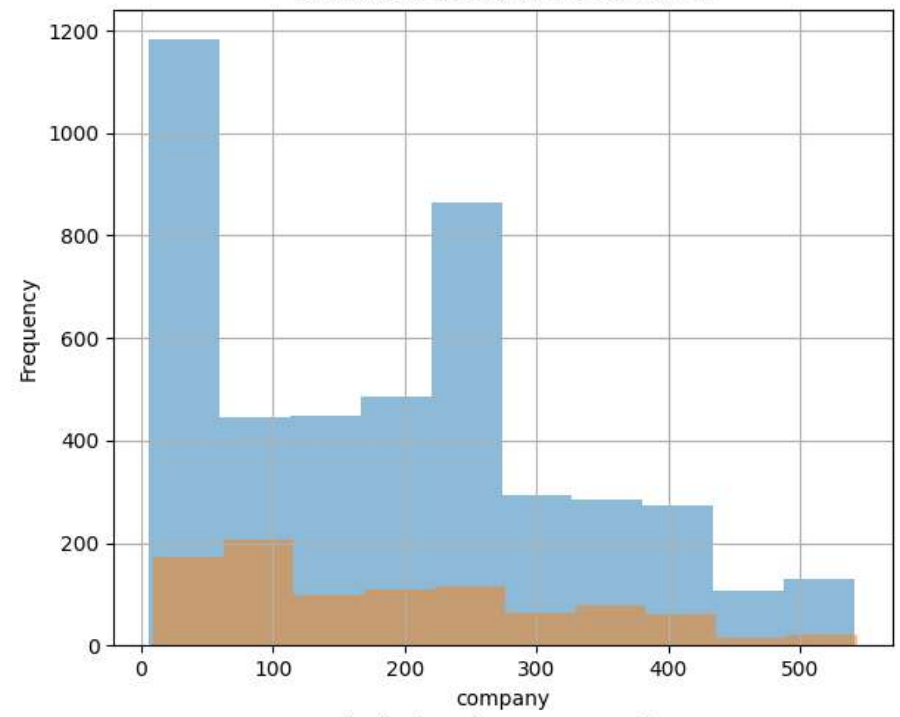

Distribution of reserved_room_type for
reservations that were canceled and not
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000
C
A
D
G
E
F
H
L
B
P
reserved_room_type

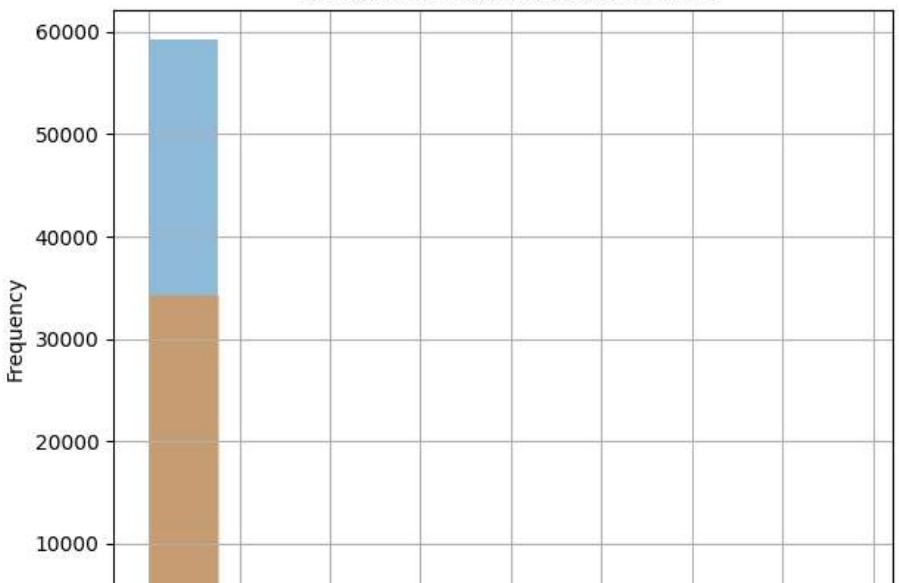

Distribution of assigned_room_type for
reservations that were canceled and not
Frequency
0
5000
10000
15000
20000
25000
30000
C
A
E
G
F
I
D
H
B
K
P
L
assigned_room_type

Distribution of booking_changes for
reservations that were canceled and not
Frequency
0
10000
20000
30000
40000
50000
60000
0
5
10
15
20
booking_changes

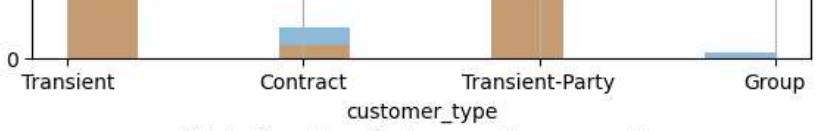
Distribution of deposit_type for
reservations that were canceled and not
Frequency
0
10000
20000
30000
40000
50000
60000
No Deposit
Refundable
Non Refund
deposit_type

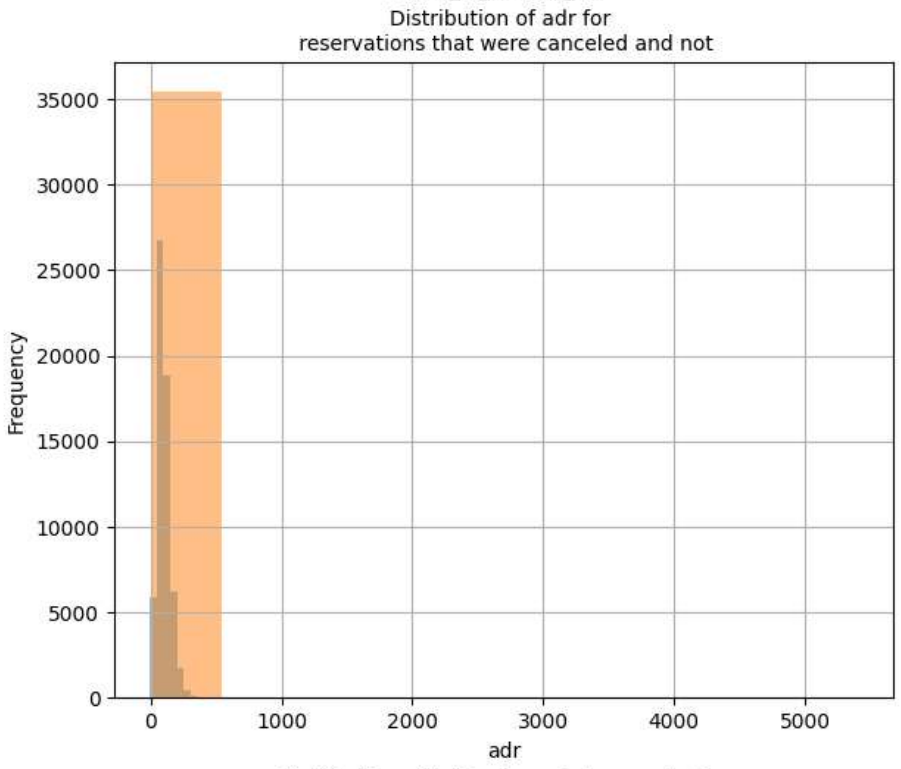

Distribution of agent for
reservations that were canceled and not
Frequency
0
5000
10000
15000
20000
25000
30000
0
100
200
300
400
500
agent

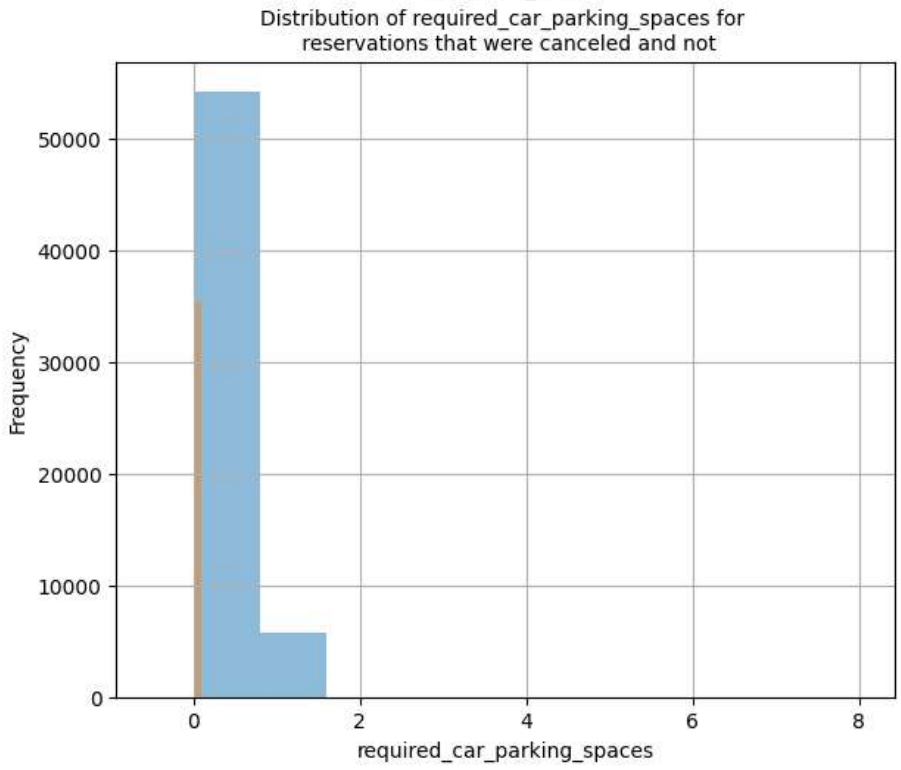

Distribution of company for
reservations that were canceled and not
Frequency
0
200
400
600
800
1000
1200
0
100
200
300
400
500
company

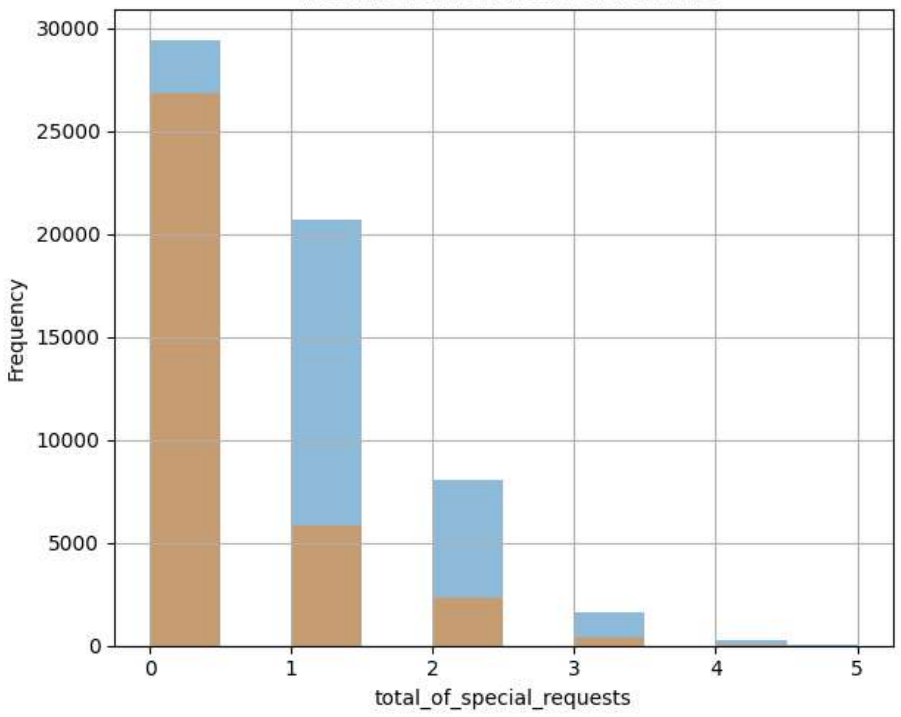

Distribution of days_in_waiting_list for
reservations that were canceled and not
Frequency
0
10000
20000
30000
40000
50000
60000

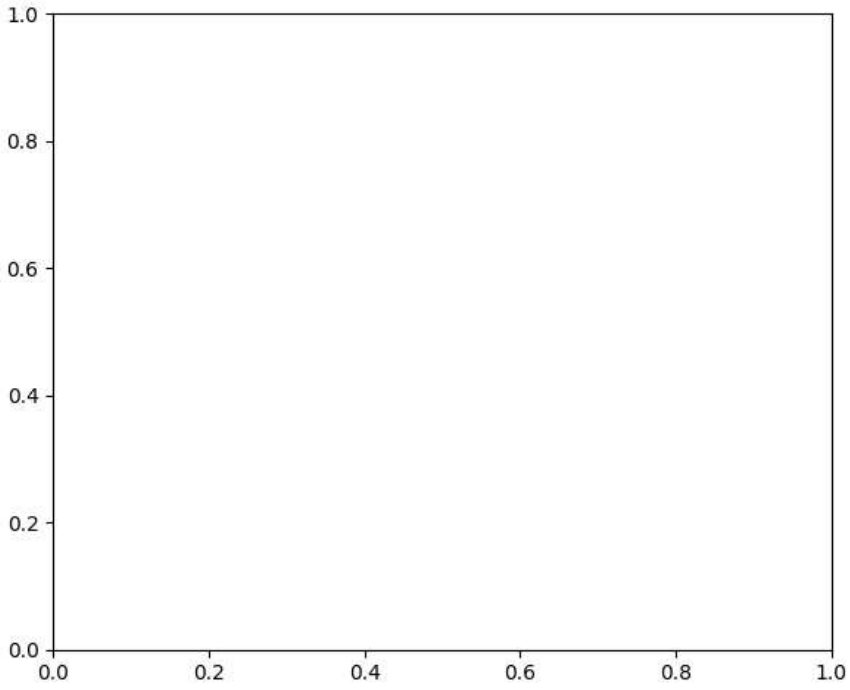

Distribution of customer_type for
reservations that were canceled and not
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000











Reviewing Missing Values

In [ ]:
```
X_train_ohe.isna().sum()[X_train_ohe.isna().sum() != 0]
```

Out[ ]:
```
children    2
dtype: int64
```

Imputing missing children values with the mean

In [ ]:
```
filledData = X_train_ohe.fillna(data['X_train'].loc[:, 'children'].mean())
```

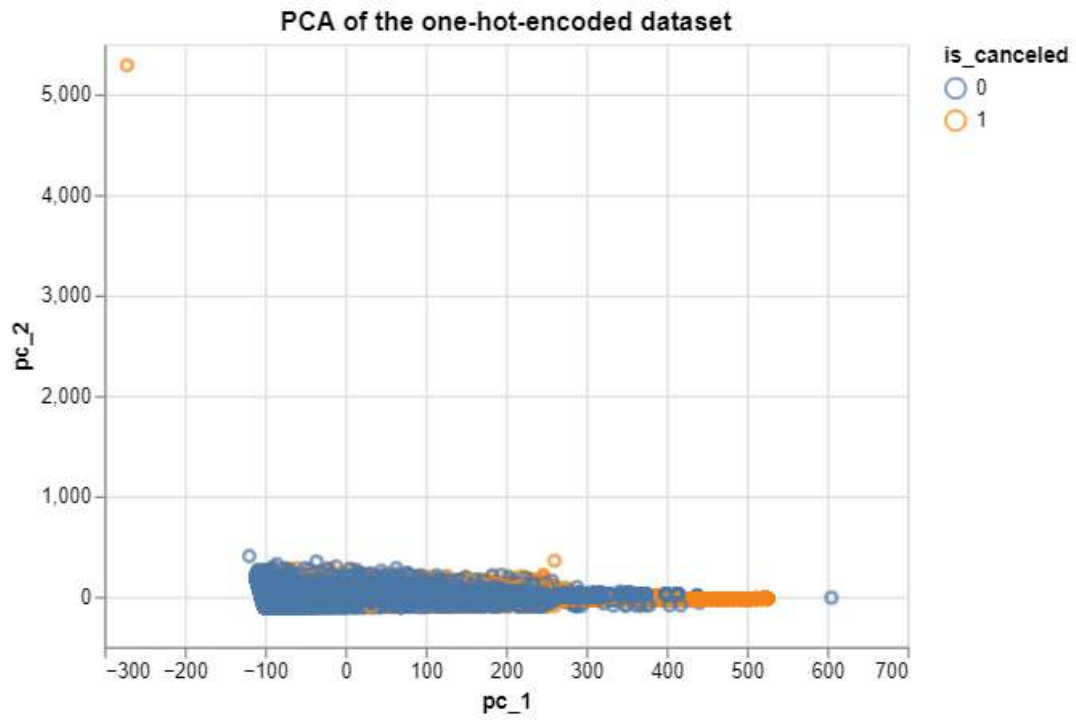
Creating a Normalized Data Set

In [ ]:
```
normalizedData = (filledData - filledData.mean()) / filledData.std().replace(0, 1)

assert normalizedData['lead_time'].equals((filledData['lead_time'] - filledData['lead_time'].mean()) / filledData['lead_time'].std
assert normalizedData['customer_type_Transient'].equals((filledData['customer_type_Transient'] - filledData['customer_type_Transie
assert normalizedData.isna().sum().sum() == 0
```

Visualizing Data along Principal Components

In [ ]:
```
from sklearn.decomposition import PCA
import altair as alt

alt.data_transformers.disable_max_rows()

pca = PCA(n_components=2)
pca_X = pca.fit_transform(filledData)

alt.Chart(
    pd.concat(
        (
            pd.DataFrame(
                pca_X, columns=["pc_1", "pc_2"], index=filledData.index
            ),
            y_train,
        ),
        axis=1,
    ), title = "PCA of the one-hot-encoded dataset"
).mark_point().encode(x="pc_1", y="pc_2", color="is_canceled:N")
```

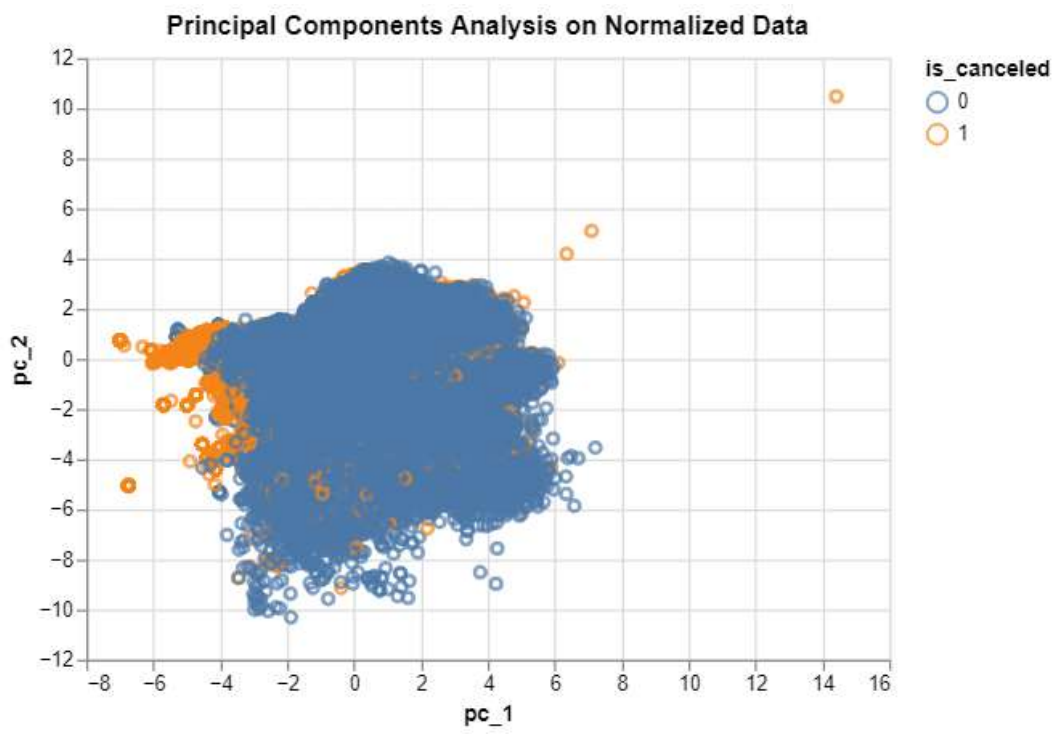
Out[ ]:



In [ ]:

```
norm_pca = PCA(n_components=2)
norm_pca_X = norm_pca.fit_transform(normalizedData)

alt.Chart(
    pd.concat(
        (
            pd.DataFrame(
                norm_pca_X, columns=["pc_1", "pc_2"], index=normalizedData.index
            ),
            y_train,
        ),
        axis=1,
    ), title = "Principal Components Analysis on Normalized Data"
).mark_point().encode(x="pc_1", y="pc_2", color="is_canceled:N")
```

Out[ ]:



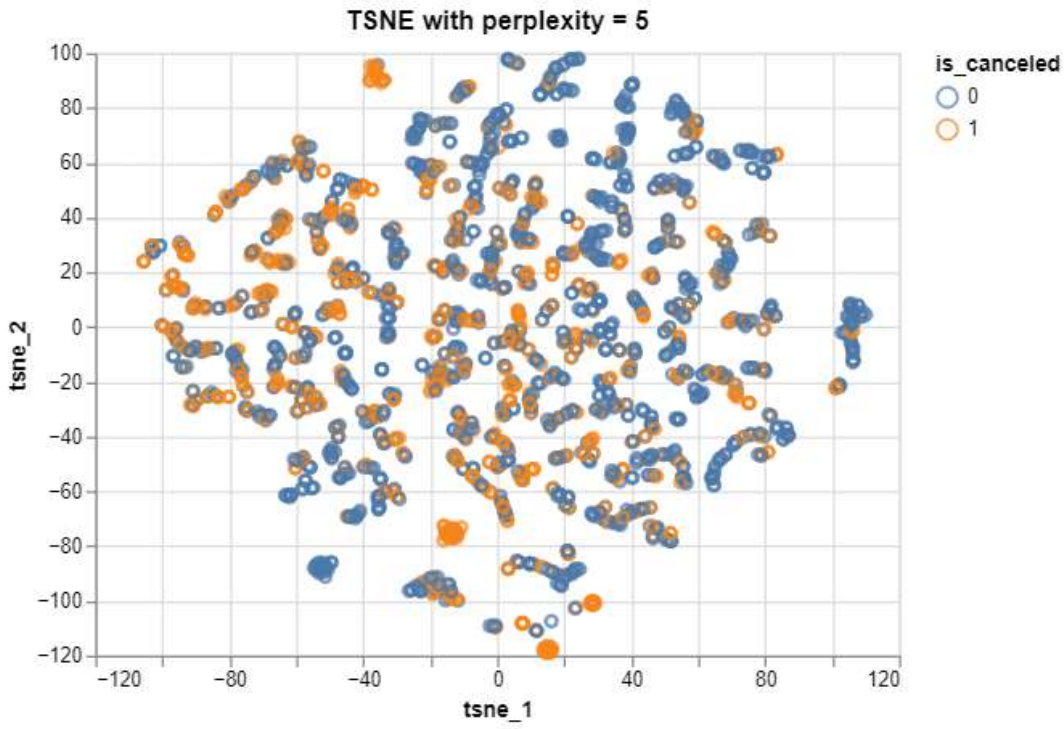
Visualizing Data with t-SNE

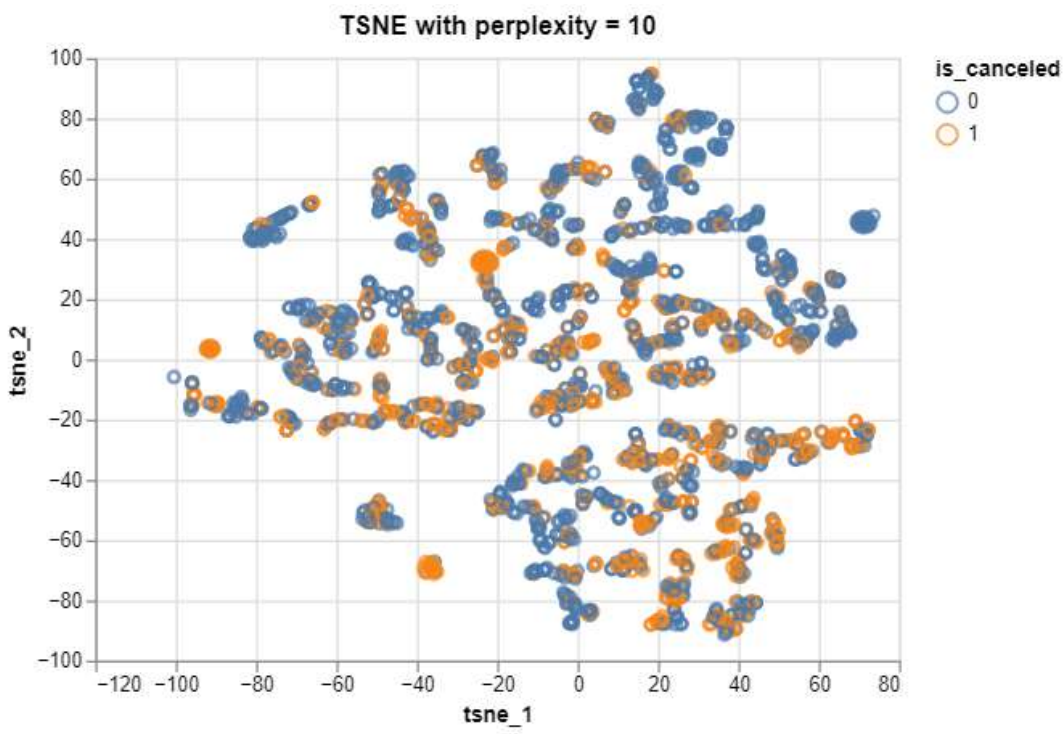
In [ ]:

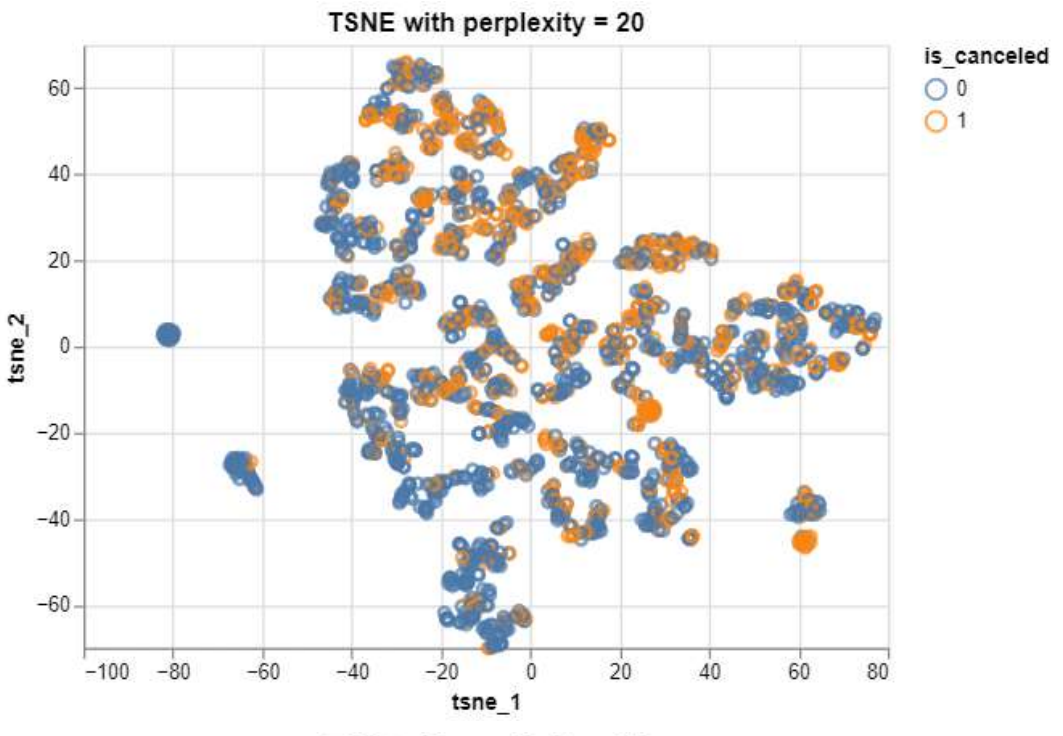
```
from sklearn.manifold import TSNE

charts = []
for my_perplexity in (5, 10, 20, 50, 100, 200, 500, 1000):
    tsne = TSNE(n_components = 2, perplexity=my_perplexity)
    tsne_X = tsne.fit_transform(filledData[:2000])


    charts.append(alt.Chart(
        pd.concat(
            (
                pd.DataFrame(
                    tsne_X, columns=["tsne_1", "tsne_2"]
                ),
                y_train.reset_index(drop = True)[:2000],
            ),
            axis=1,
        ), title = f"TSNE with perplexity = {my_perplexity}"
    ).mark_point().encode(x="tsne_1", y="tsne_2", color="is_canceled:N"))
```

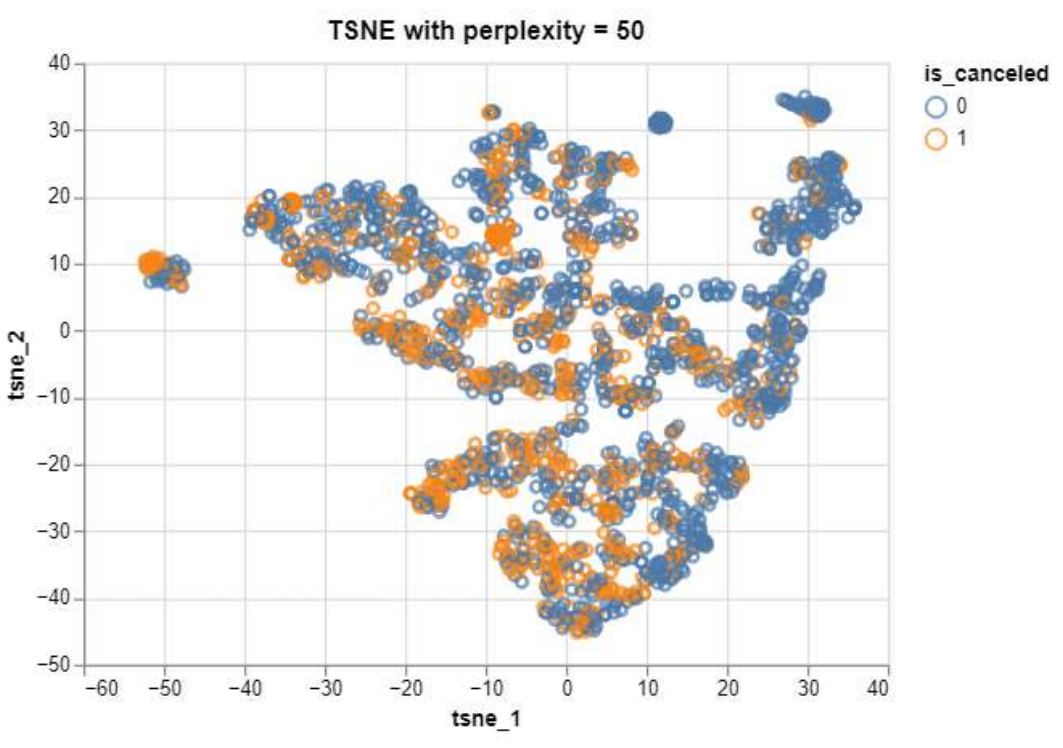
In [ ]:

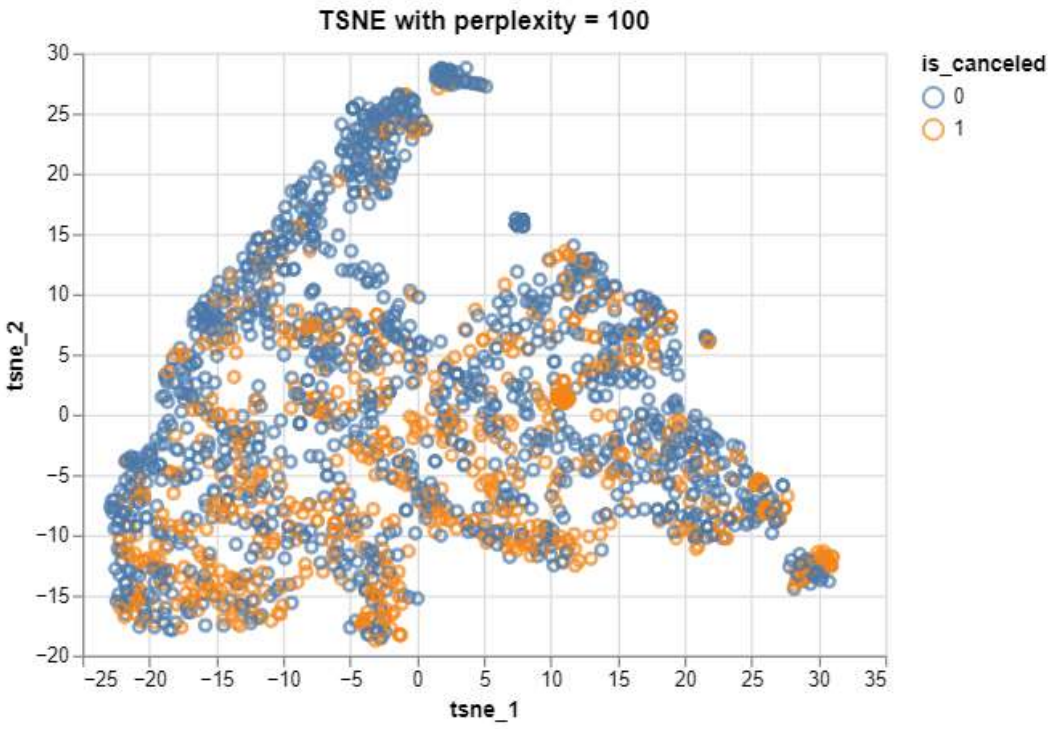
```
for eachChart in charts:
    display(eachChart)
```

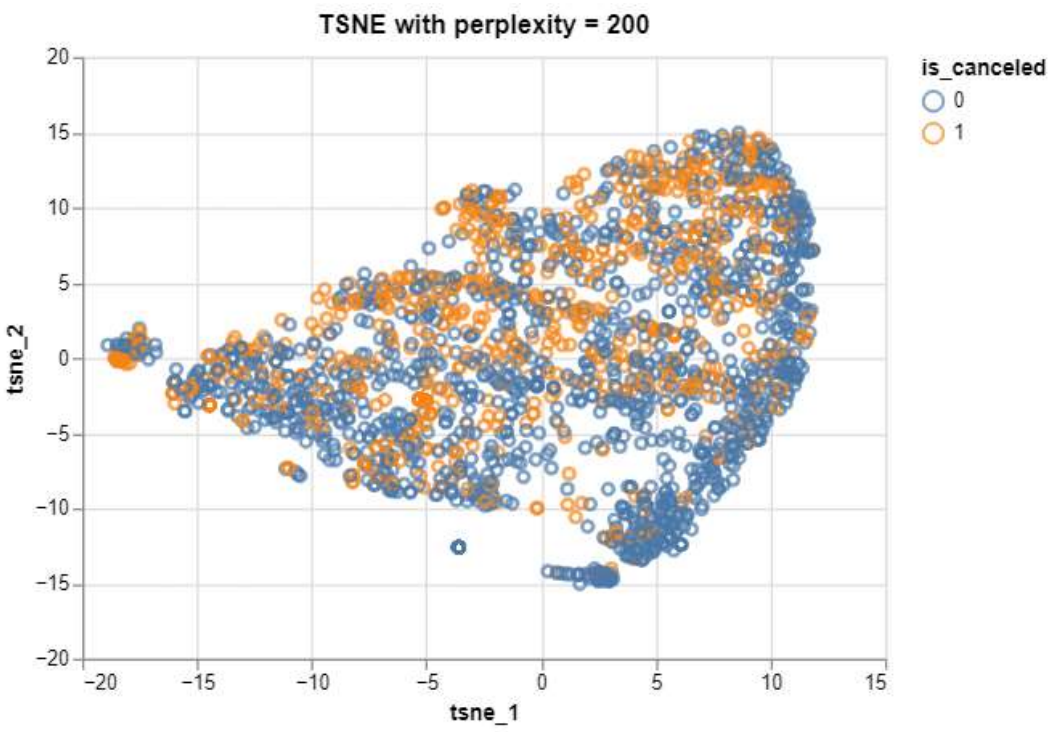

TSNE with perplexity = 5
tsne_2
tsne_1
is_canceled
0
1

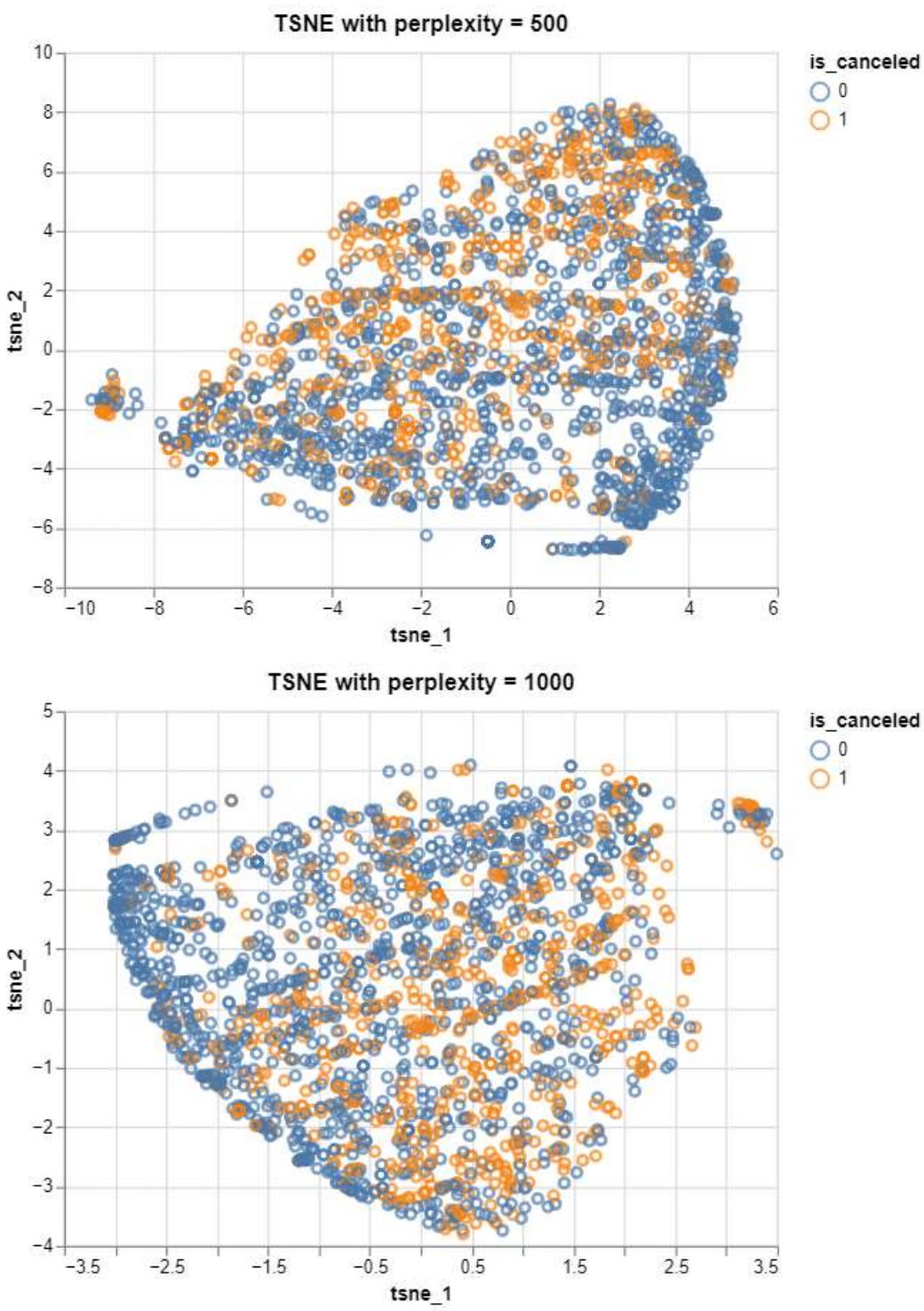

TSNE with perplexity = 10
tsne_2
tsne_1
is_canceled
0
1


TSNE with perplexity = 20
tsne_2
tsne_1
is_canceled
0
1


TSNE with perplexity = 50
tsne_2
tsne_1
is_canceled
0
1



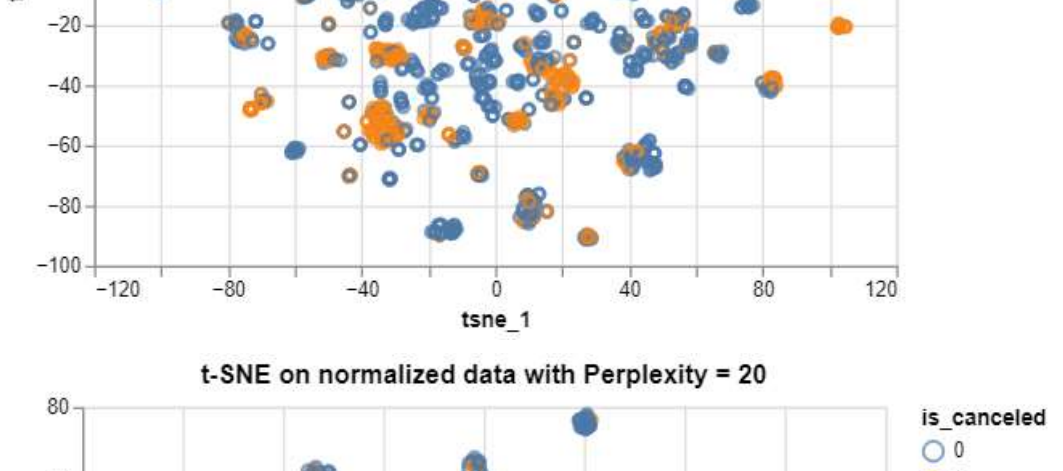


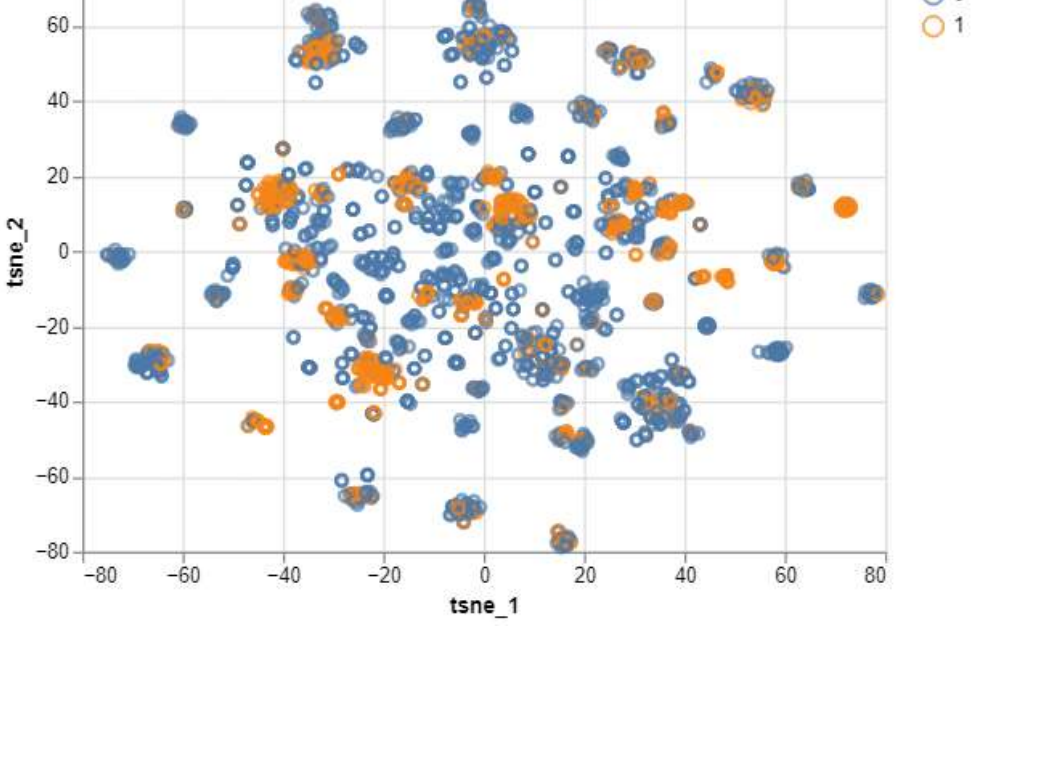


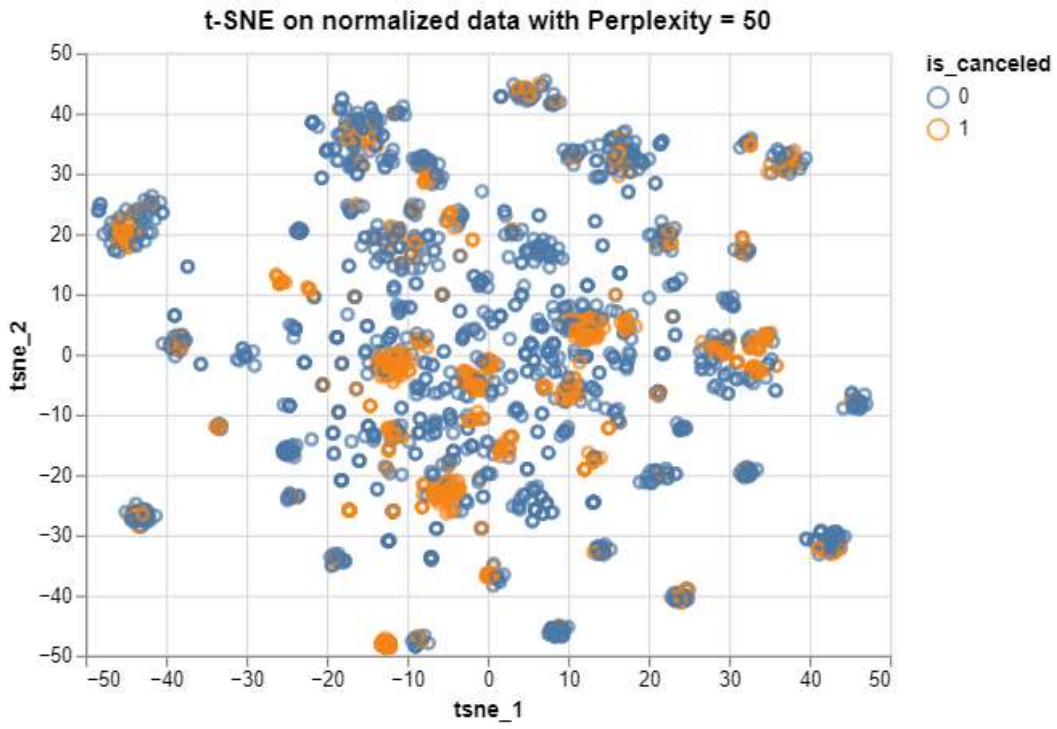
In [ ]:
```
norm_charts = []
for my_perplexity in (5, 10, 20, 50, 100, 200, 500, 1000):
    norm_tsne = TSNE(n_components = 2, perplexity=my_perplexity)
    norm_tsne_X = norm_tsne.fit_transform(normalizedData[:2000])


    norm_charts.append(alt.Chart(
```

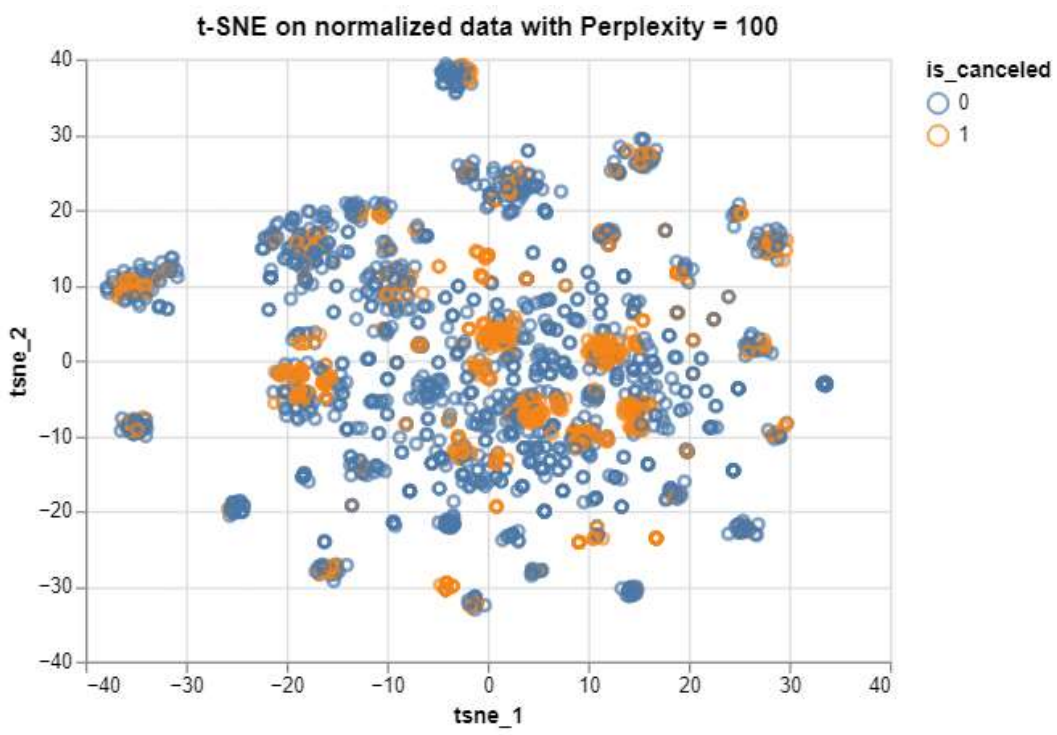
```
        pd.concat(
            (
                pd.DataFrame(
                    norm_tsne_X, columns=["tsne_1", "tsne_2"]
                ),
                y_train.reset_index(drop = True)[:2000],
            ),
            axis=1,
        ), title = f"t-SNE on normalized data with Perplexity = {my_perplexity}"
    ).mark_point().encode(x="tsne_1", y="tsne_2", color="is_canceled:N"))
```

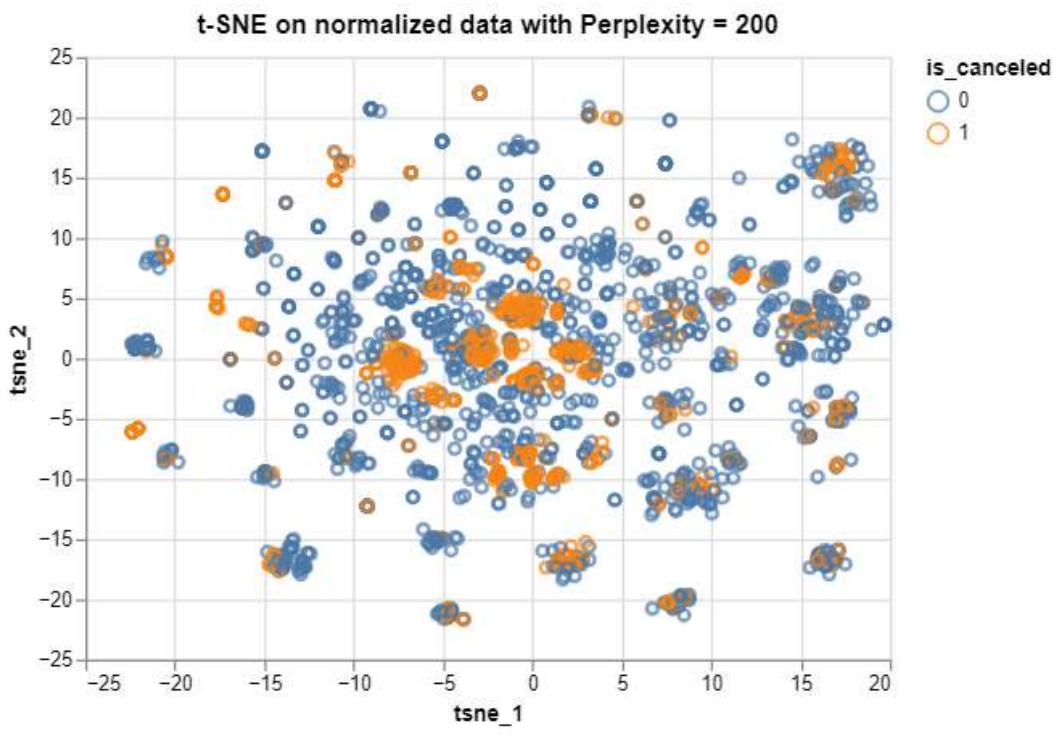
In [ ]:
```
for normChart in norm_charts:
    display(normChart)
```

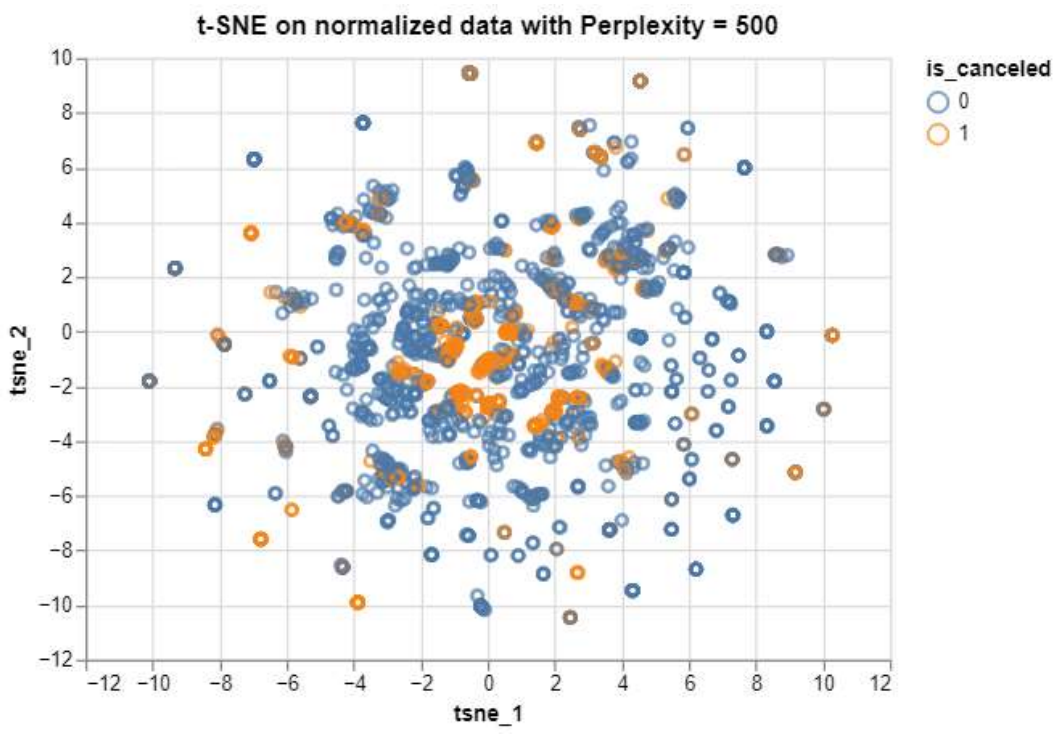


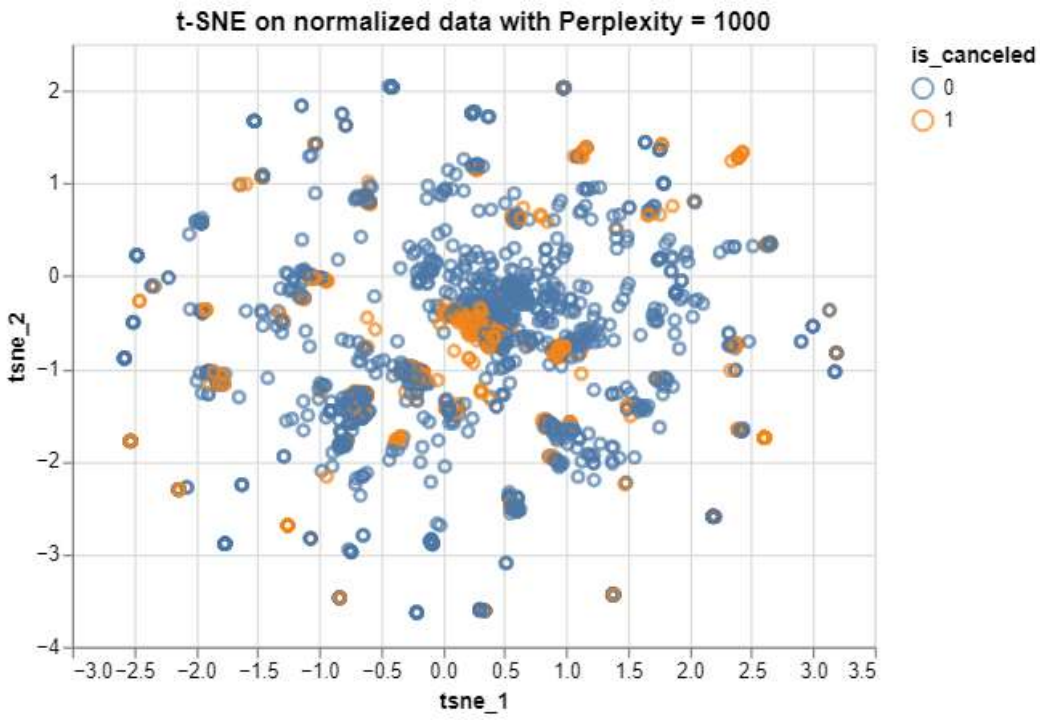



t-SNE on normalized data with Perplexity = 50
tsne_2
tsne_1
is_canceled
0
1


t-SNE on normalized data with Perplexity = 100
tsne_2
tsne_1
is_canceled
0
1


t-SNE on normalized data with Perplexity = 200
tsne_2
tsne_1
is_canceled
0
1


t-SNE on normalized data with Perplexity = 500
tsne_2
tsne_1
is_canceled
0
1



Setting up Validation set

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train_split, y_val = train_test_split(filledData, y_train, test_size = 0.2, random_state = 42)
X_train_norm, X_val_norm, y_train_norm, y_val_norm = train_test_split(normalizedData, y_train, test_size = 0.2, random_state = 42)
```

Modeling

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score

rf = RandomForestClassifier()
rf.fit(X_train, y_train_split)
rf_accuracy = rf.score(X_val, y_val)
rf_roc = roc_auc_score(y_val, rf.predict_proba(X_val)[:, 1])

print(f"The validation accuracy of the random forest model is {rf_accuracy * 100: .2f}%, and the ROC AUC score on the validation s
```

```
The validation accuracy of the random forest model is  89.02%, and the ROC AUC score on the validation set is  95.61%.
```

In [ ]:
```
norm_rf = RandomForestClassifier(n_estimators=200)
norm_rf.fit(X_train_norm, y_train_norm)
norm_rf_accuracy = norm_rf.score(X_val_norm, y_val_norm)
norm_rf_roc = roc_auc_score(y_val_norm, norm_rf.predict_proba(X_val_norm)[:, 1])

print(f"The validation accuracy of the random forest model on the normalized data is {norm_rf_accuracy * 100: .2f}%, and the ROC A
```

```
The validation accuracy of the random forest model on the normalized data is  88.98%, and the ROC AUC score on the validation set
is  95.71%.
```

In [ ]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV, PredefinedSplit
from scipy.stats import loguniform
import numpy as np

mlp = MLPClassifier(hidden_layer_sizes=(100, 100))
# mlp = MLPClassifier()

# # Specify model hyperparameter search space
# param_dist = {
#     "learning_rate_init": loguniform(1e-4, 1e-2),
#     "alpha": loguniform(1e-7, 10),
#     "batch_size": [50, 100, 200],
#     "hidden_layer_sizes": [(10, 10), (30, 30), (50, 50), (100,), (100, 100)],
# }

# # For RandomSeachCV, we will need to combine training and validation sets then
# #  specify which portion is training and which is validation
# # Also, for the final performance evaluation, train on all of the training AND validation data
# X_train_plus_val = np.concatenate((X_train[:5000], X_val), axis=0)
# y_train_plus_val = np.concatenate((y_train[:5000], y_val), axis=0)

# # Create a predefined train/test split for RandomSearchCV (to be used later)
# validation_fold = np.concatenate((-1 * np.ones(len(y_train[:5000])), np.zeros(len(y_val))))
# train_val_split = PredefinedSplit(validation_fold)

# # run randomized search
# n_iter_search = 50
# random_search = RandomizedSearchCV(
#     mlp,
#     param_distributions=param_dist,
#     n_iter=n_iter_search,
#     n_jobs=-1,
#     cv=train_val_split,
# )

# random_search.fit(X_train_plus_val, y_train_plus_val)
# mlp.set_params(**random_search.best_params_)
mlp.fit(X_train[:10_000], y_train_split[:10_000])
mlp_accuracy = mlp.score(X_val, y_val)
mlp_roc = roc_auc_score(y_val, mlp.predict_proba(X_val)[:, 1])
```

```
print(f"The validation accuracy of the MLP model is {mlp_accuracy * 100: .2f}%, and the ROC AUC score on the validation set is {ml
```

```
The validation accuracy of the MLP model is  83.18%, and the ROC AUC score on the validation set is  91.08%.
```

In [ ]:
```
norm_mlp = MLPClassifier(hidden_layer_sizes=(300, 300), max_iter=1000)
# norm_mlp = MLPClassifier()

# # Specify model hyperparameter search space
# param_dist = {
#     "learning_rate_init": loguniform(1e-4, 1e-2),
#     "alpha": loguniform(1e-7, 10),
#     "batch_size": [50, 100, 200],
#     "hidden_layer_sizes": [(10, 10), (30, 30), (50, 50), (100,), (100, 100)],
# }

# # For RandomSeachCV, we will need to combine training and validation sets then
# #  specify which portion is training and which is validation
# # Also, for the final performance evaluation, train on all of the training AND validation data
# X_train_plus_val = np.concatenate((X_train_norm[:5000], X_val_norm), axis=0)
# y_train_plus_val = np.concatenate((y_train_norm[:5000], y_val_norm), axis=0)

# # Create a predefined train/test split for RandomSearchCV (to be used later)
# validation_fold = np.concatenate((-1 * np.ones(len(y_train_norm[:5000])), np.zeros(len(y_val))))
# train_val_split = PredefinedSplit(validation_fold)

# # run randomized search
# n_iter_search = 50
# random_search = RandomizedSearchCV(
#     norm_mlp,
#     param_distributions=param_dist,
#     n_iter=n_iter_search,
#     n_jobs=-1,
#     cv=train_val_split,
# )

# random_search.fit(X_train_plus_val, y_train_plus_val)
# norm_mlp.set_params(**random_search.best_params_)
norm_mlp.fit(X_train_norm[:30_000], y_train_norm[:30_000])
norm_mlp_accuracy = norm_mlp.score(X_val_norm, y_val_norm)
norm_mlp_roc = roc_auc_score(y_val_norm, norm_mlp.predict_proba(X_val_norm)[:, 1])

print(f"The validation accuracy of the MLP model is {norm_mlp_accuracy * 100: .2f}%, and the ROC AUC score on the validation set i
```

```
The validation accuracy of the MLP model is  85.56%, and the ROC AUC score on the validation set is  92.79%.
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver = 'saga', penalty = 'l1', C = 0.1, max_iter=200)
lr.fit(X_train_norm, y_train_norm)
lr_accuracy = lr.score(X_val_norm, y_val_norm)
lr_roc = roc_auc_score(y_val_norm, lr.predict_proba(X_val_norm)[:, 1])

print(f"The validation accuracy of the logistic regression model on the normalized data is {lr_accuracy * 100: .2f}%, and the ROC
```

```
The validation accuracy of the logistic regression model on the normalized data is  83.54%, and the ROC AUC score on the validati
on set is  91.26%.
```

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_norm, y_train_norm)
knn_accuracy = knn.score(X_val_norm, y_val_norm)
knn_roc = roc_auc_score(y_val_norm, knn.predict_proba(X_val_norm)[:, 1])

print(f"The validation accuracy of the KNN model on the normalized data is {knn_accuracy * 100: .2f}%, and the ROC AUC score on th
```

```
The validation accuracy of the KNN model on the normalized data is  82.02%, and the ROC AUC score on the validation set is  87.8
9%.
```

In [ ]:
```
filledTest = X_test_ohe.fillna(data['X_train'].loc[:, 'children'].mean())

normalizedTest = (filledTest - filledData.mean()) / filledData.std().replace(0, 1)

assert normalizedTest['lead_time'].equals((filledTest['lead_time'] - filledData['lead_time'].mean()) / filledData['lead_time'].std
assert normalizedTest['customer_type_Transient'].equals((filledTest['customer_type_Transient'] - filledData['customer_type_Transie
assert normalizedTest.isna().sum().sum() == 0
```

In [ ]:
```
create_submission(norm_rf.predict_proba(normalizedTest)[:, 1], 'submission3.csv')
```

In [ ]:
```
norm_pca_n = PCA(n_components=X_train_norm.shape[1])
norm_pca_X_n = norm_pca_n.fit_transform(X_train_norm)
norm_pca_val_n = norm_pca_n.transform(X_val_norm)

norm_pca_rf = RandomForestClassifier(n_estimators=200)
norm_pca_rf.fit(norm_pca_X_n, y_train_norm)
norm_pca_rf_accuracy = norm_pca_rf.score(norm_pca_val_n, y_val_norm)
norm_pca_rf_roc = roc_auc_score(y_val_norm, norm_pca_rf.predict_proba(norm_pca_val_n)[:, 1])

print(f"The validation accuracy of the random forest model on the normalized data is {norm_pca_rf_accuracy * 100: .2f}%, and the R
```

```
The validation accuracy of the random forest model on the normalized data is  87.20%, and the ROC AUC score on the validation set
is  94.35%.
```

2

[25 points] Clustering

Clustering can be used to reveal structure between samples of data and assign group membership to similar groups of samples. This exercise will provide you with experience applying clustering algorithms and comparing these techniques on various datasets to experience the pros and cons of these approaches when the structure of the data being clustered varies. For this exercise, we'll explore clustering in two dimensions to make the results more tangible, but in practice these approaches can be applied to any number of dimensions.

Note: For each set of plots across the five datasets, please create subplots within a single figure (for example, when applying DBSCAN - please show the clusters resulting from DBSCAN as a single figure with one subplot for each dataset). This will make comparison easier.

(a) Run K-means and choose the number of clusters. Five datasets are provided for you below and the code to load them below.

- Scatterplot each dataset
- For each dataset run the k-means algorithm for values of k ranging from 1 to 10 and for each plot the "elbow curve" where you plot dissimilarity in each case. Here, you can measure dissimilarity using the within-cluster sum-of-squares, which in sklean is known as "inertia" and can be accessed through the `inertia_` attribute of a fit KMeans class instance.
- For each dataset, where is the elbow in the curve of within-cluster sum-of-squares and why? Is the elbow always clearly visible? When it's not clear, you will have to use your judgment in terms of selecting a reasonable number of clusters for the data. *There are also other metrics you can use to explore to measure the quality of cluster fit (but do not have to for this assignment) including the silhouette score, the Calinski-Harabasz index, and the Davies-Bouldin, to name a few within sklearn alone. However, assessing the quality of fit without "preferred" cluster assignments to compare against (that is, in a truly unsupervised manner) is challenging because measuring cluster fit quality is typically poorly-defined and doesn't generalize across all types of inter- and intra-cluster variation.*
- Plot your clustered data (different color for each cluster assignment) for your best k-means fit determined from both the elbow curve and your judgment for each dataset and your inspection of the dataset.

(b) Apply DBSCAN. Vary the `eps` and `min_samples` parameters to get as close as you can to having the same number of clusters as your choices with K-means. In this case, the black points are points that were not assigned to clusters.

(c) Apply Spectral Clustering. Select the same number of clusters as selected by k-means.

(d) Comment on the strengths and weaknesses of each approach. In particular, mention:

- Which technique worked "best" and "worst" (as defined by matching how human intuition would cluster the data) on each dataset?
- How much effort was required to get good clustering for each method (how much parameter tuning needed to be done)?

Note: For these clustering plots in this question, do NOT include legends indicating cluster assignment; instead, just make sure the cluster assignments are clear from the plot (e.g. different colors for each cluster)

Code is provided below for loading the datasets and for making plots with the clusters as distinct colors

In [ ]:
```
################################
# Load the data
################################
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons

# Create / load the datasets:
n_samples = 1500
X0, _ = make_blobs(n_samples=n_samples, centers=2, n_features=2, random_state=0)
X1, _ = make_blobs(n_samples=n_samples, centers=5, n_features=2, random_state=0)

random_state = 170
X, y = make_blobs(n_samples=n_samples, random_state=random_state, cluster_std=1.3)
transformation = [[0.6, -0.6], [-0.2, 0.8]]
X2 = np.dot(X, transformation)
X3, _ = make_blobs(n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5], random_state=random_state)
X4, _ = make_moons(n_samples=n_samples, noise=.12)

X = [X0, X1, X2, X3, X4]
# The datasets are X[i], where i ranges from 0 to 4
```

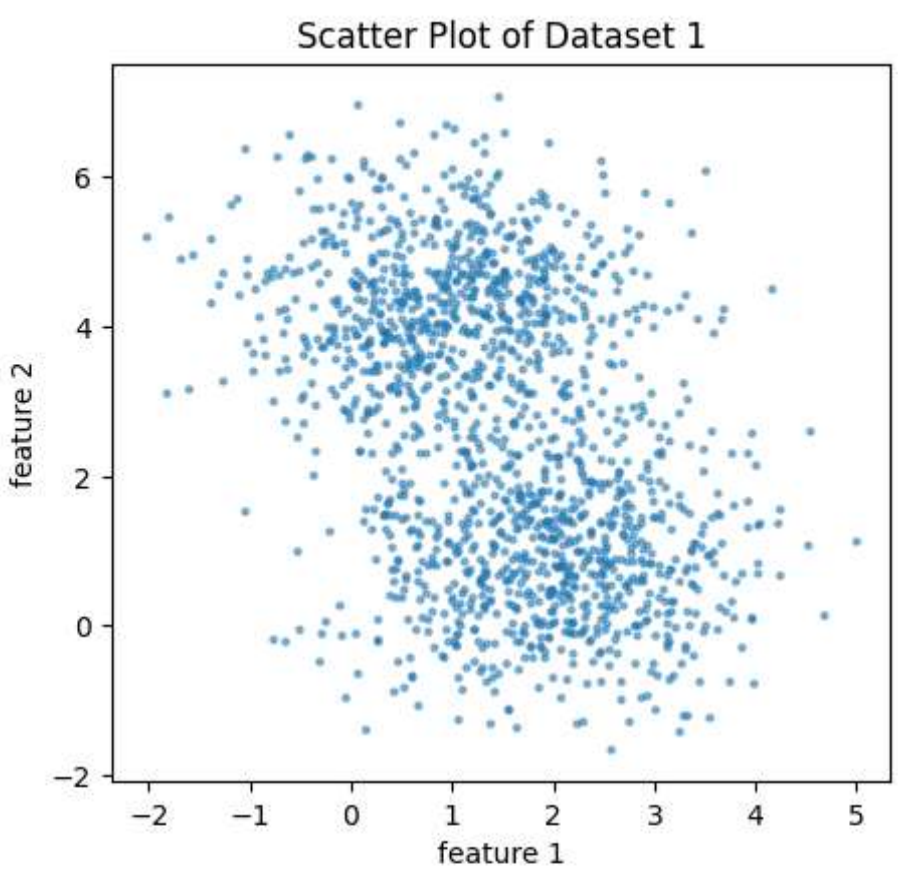
In [ ]:
```
################################
# Code to plot clusters
################################
def plot_cluster(ax, data, cluster_assignments):
    '''Plot two-dimensional data clusters

    Parameters
    ----------
    ax : matplotlib axis
        Axis to plot on
    data : list or numpy array of size [N x 2]
        Clustered data
    cluster_assignments : list or numpy array [N]
        Cluster assignments for each point in data

    '''
    clusters = np.unique(cluster_assignments)
    n_clusters = len(clusters)
    for ca in clusters:
        kwargs = {}
        if ca == -1:
            # if samples are not assigned to a cluster (have a cluster assignment of -1, color them gray)
            kwargs = {'color':'gray'}
```

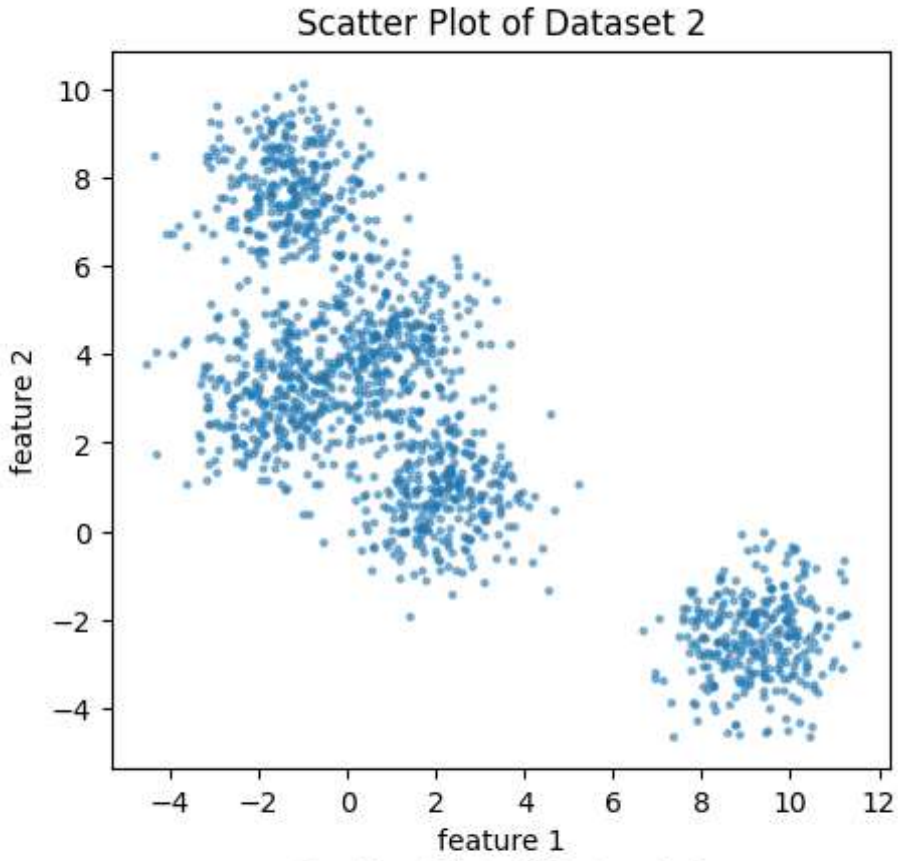
```
            n_clusters = n_clusters - 1
        ax.scatter(data[cluster_assignments==ca, 0], data[cluster_assignments==ca, 1],s=5,alpha=0.5, **kwargs)
        ax.set_xlabel('feature 1')
        ax.set_ylabel('feature 2')
        ax.set_title(f'No. Clusters = {n_clusters}')
        ax.axis('equal')
```

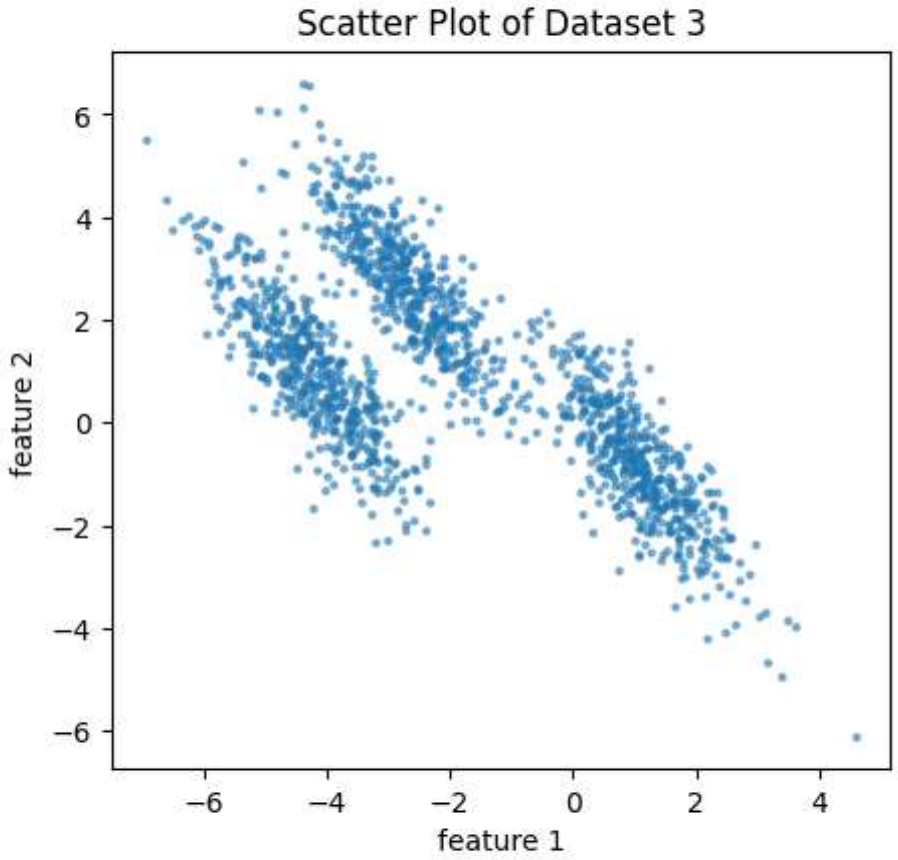
ANSWER

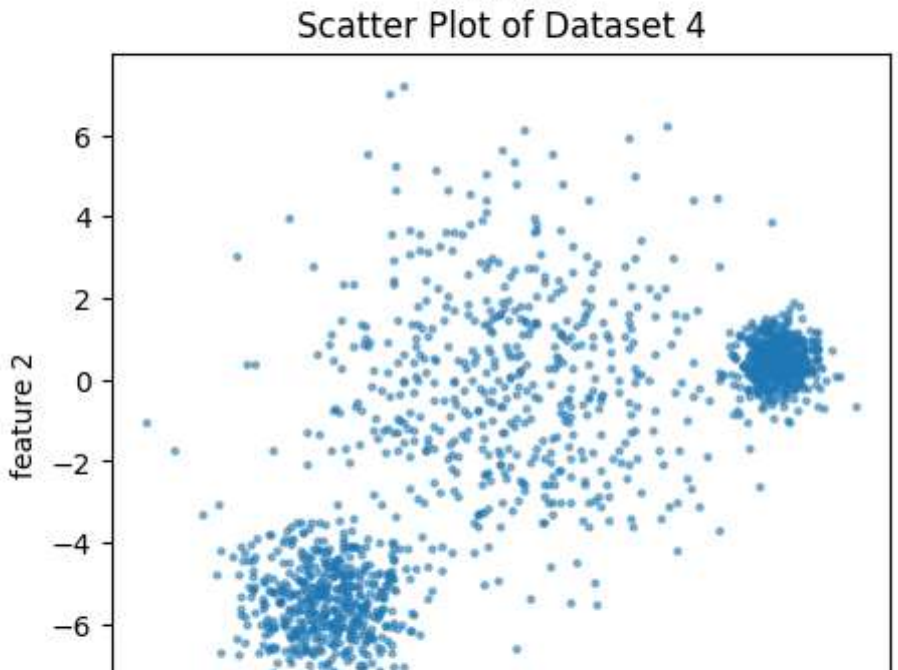
(a) Plotting the Datasets

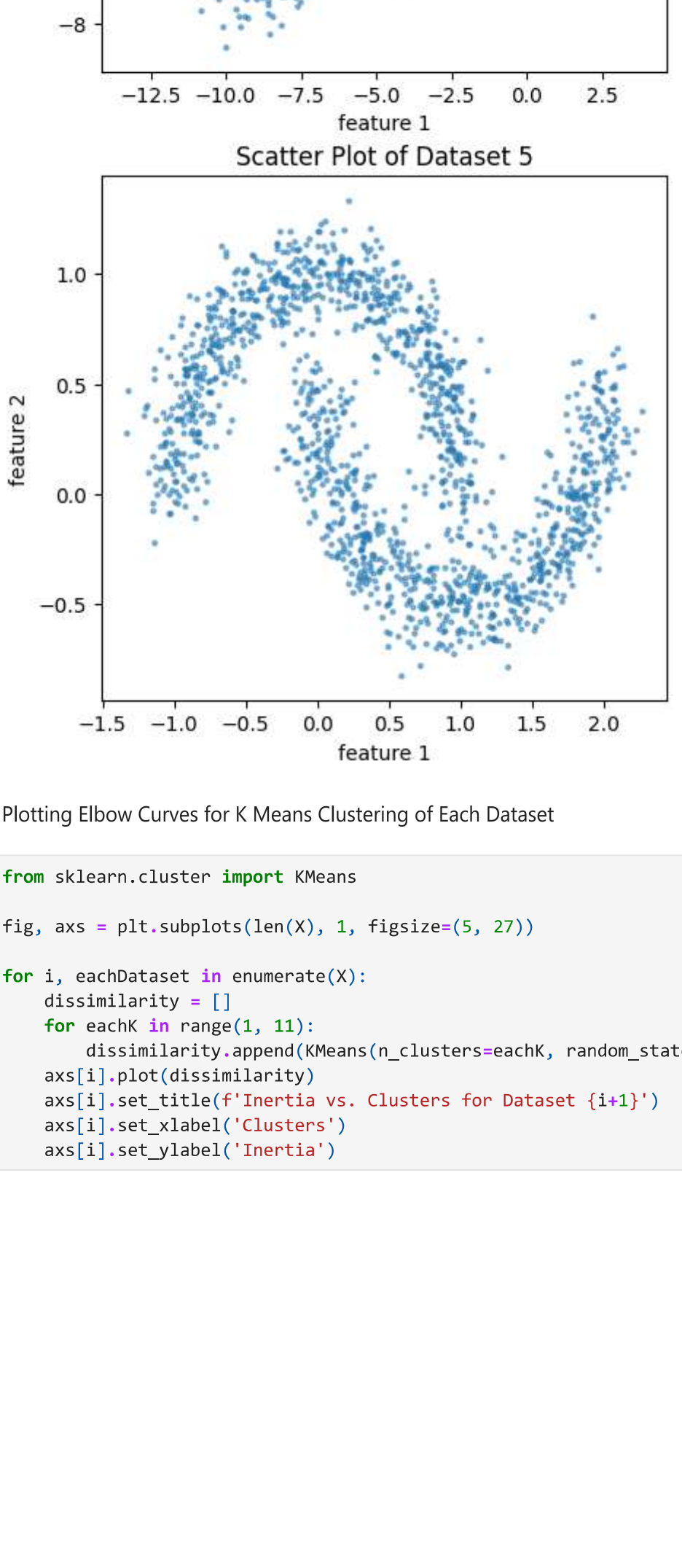
In [ ]:
```
fig, axs = plt.subplots(len(X), 1, figsize=(5, 27))

for i, eachDataset in enumerate(X):
    axs[i].scatter(eachDataset[:, 0], eachDataset[:, 1], s=5, alpha=0.5)
    axs[i].set_title(f'Scatter Plot of Dataset {i+1}')
    axs[i].set_xlabel('feature 1')
    axs[i].set_ylabel('feature 2')
```


Scatter Plot of Dataset 1
feature 2
feature 1
6
4
2
0
−2
−2
−1
0
1
2
3
4
5


Scatter Plot of Dataset 2
feature 2
feature 1
10
8
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8
10
12


Scatter Plot of Dataset 3
feature 2
feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4

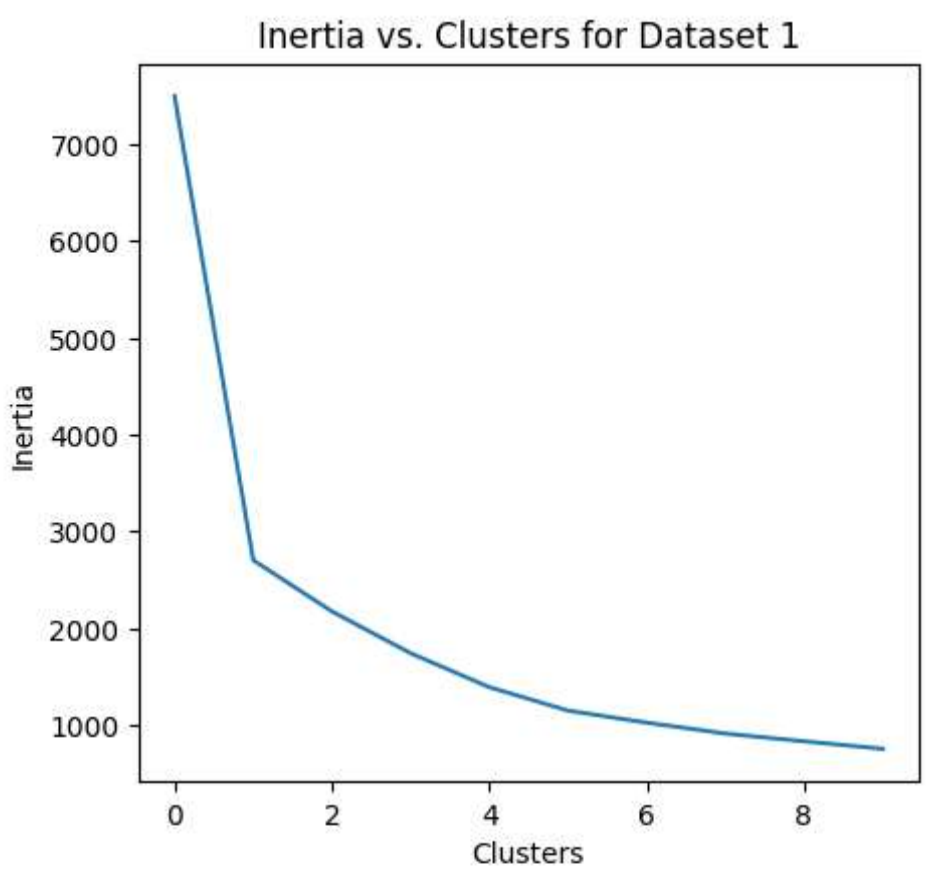

Scatter Plot of Dataset 4
feature 2
6
4
2
0
−2
−4
−6

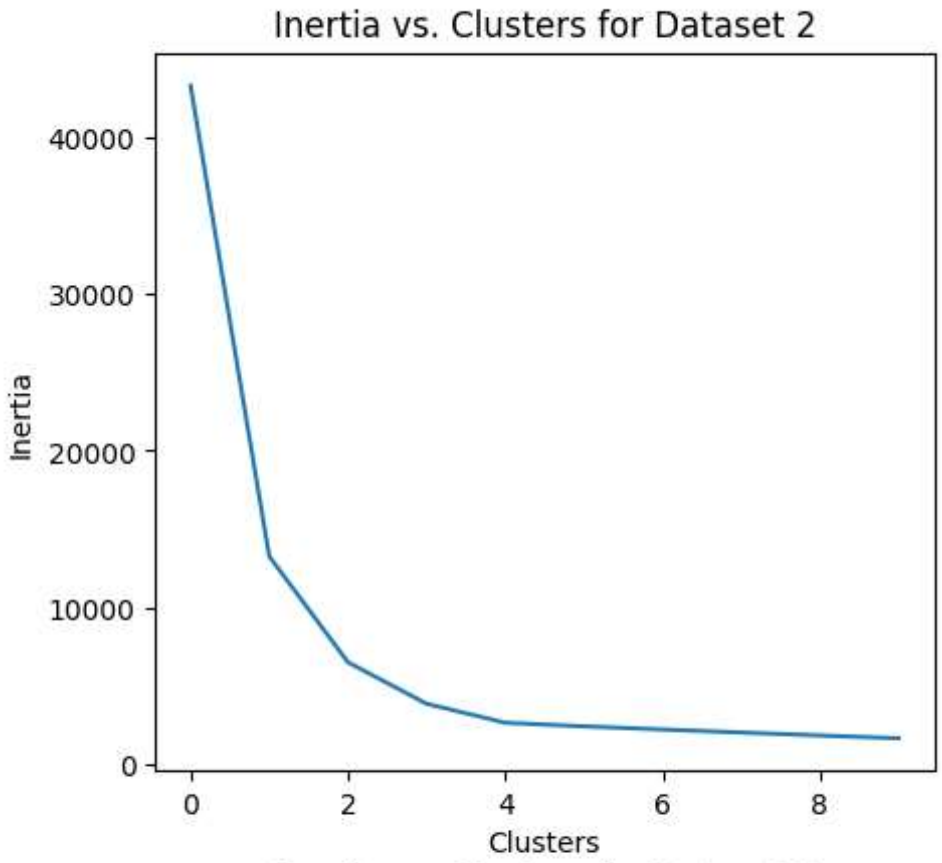


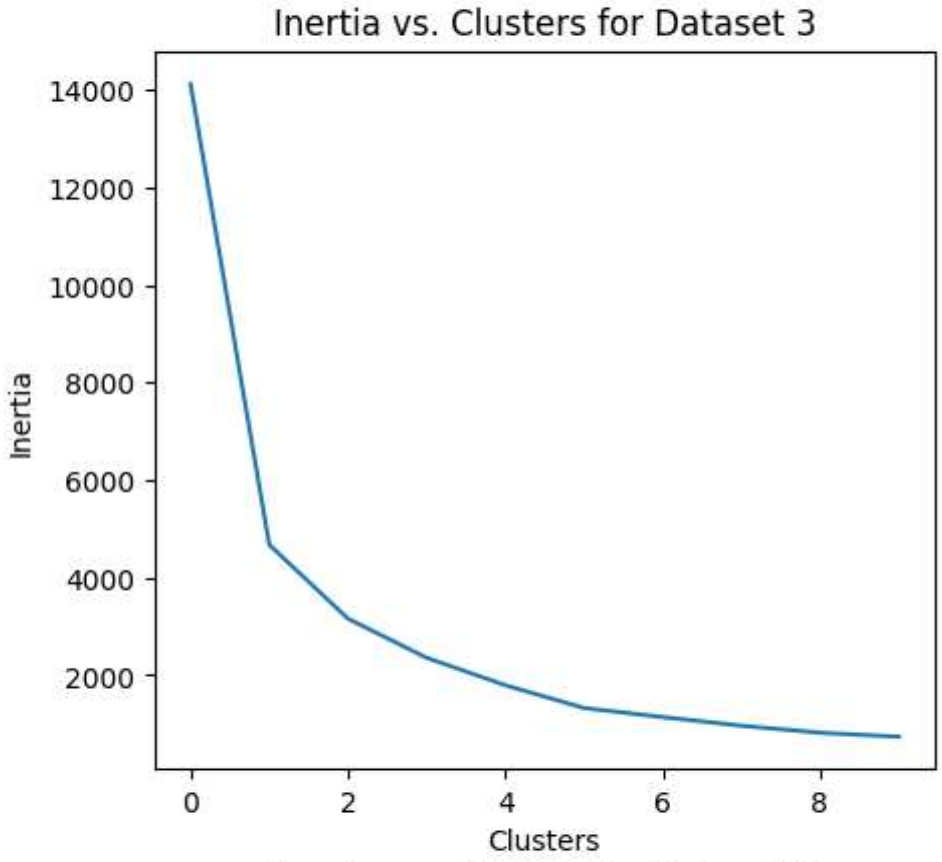
Plotting Elbow Curves for K Means Clustering of Each Dataset

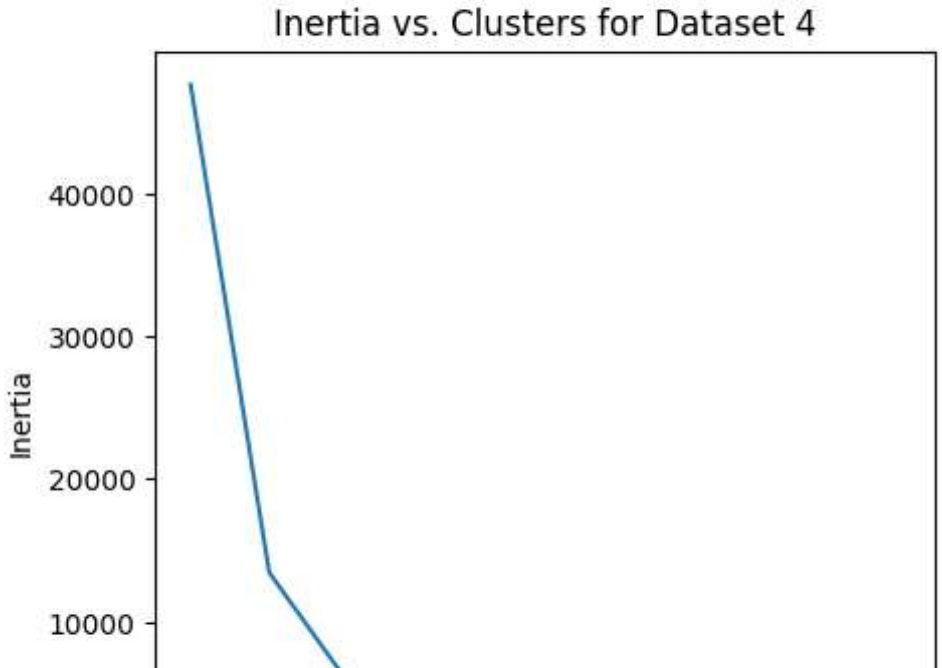
In [ ]:

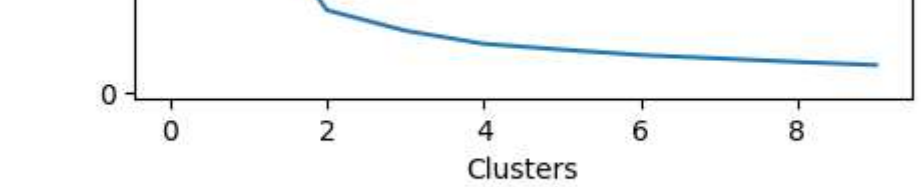
```
from sklearn.cluster import KMeans

fig, axs = plt.subplots(len(X), 1, figsize=(5, 27))

for i, eachDataset in enumerate(X):
    dissimilarity = []
    for eachK in range(1, 11):
        dissimilarity.append(KMeans(n_clusters=eachK, random_state=0).fit(eachDataset).inertia_)
    axs[i].plot(dissimilarity)
    axs[i].set_title(f'Inertia vs. Clusters for Dataset {i+1}')
    axs[i].set_xlabel('Clusters')
    axs[i].set_ylabel('Inertia')
```

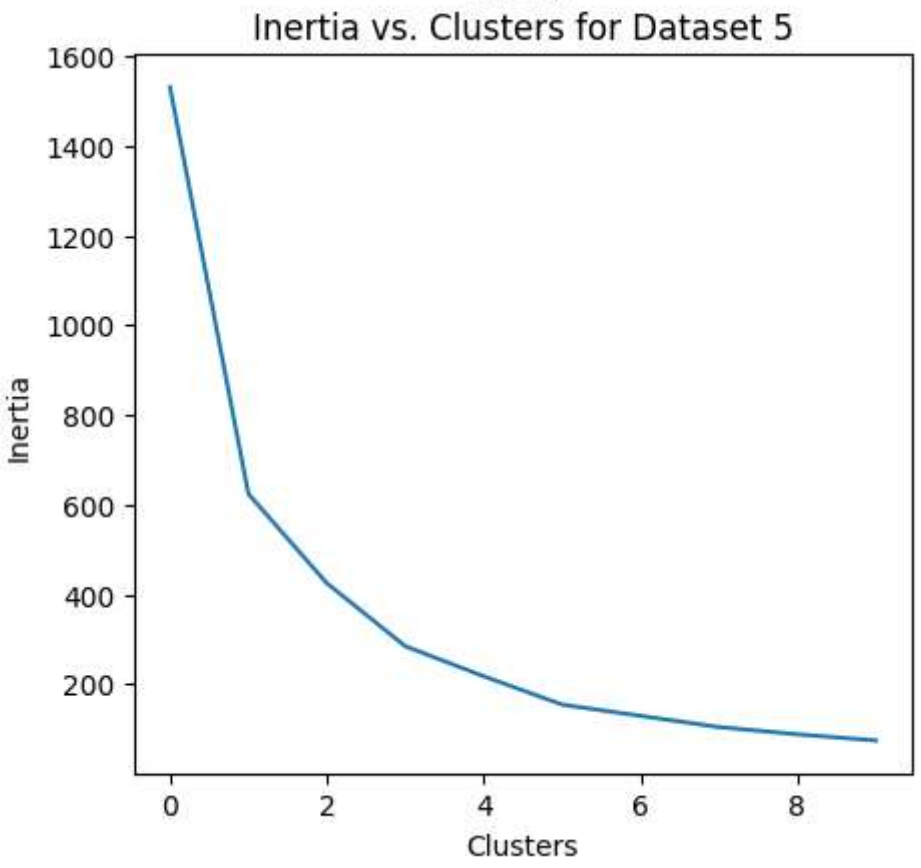

Inertia vs. Clusters for Dataset 1
Inertia
7000
6000
5000
4000
3000
2000
1000
0
2
4
6
8
Clusters


Inertia vs. Clusters for Dataset 2
Inertia
40000
30000
20000
10000
0
0
2
4
6
8
Clusters


Inertia vs. Clusters for Dataset 3
Inertia
14000
12000
10000
8000
6000
4000
2000
0
2
4
6
8
Clusters


Inertia vs. Clusters for Dataset 4
Inertia
40000
30000
20000
10000



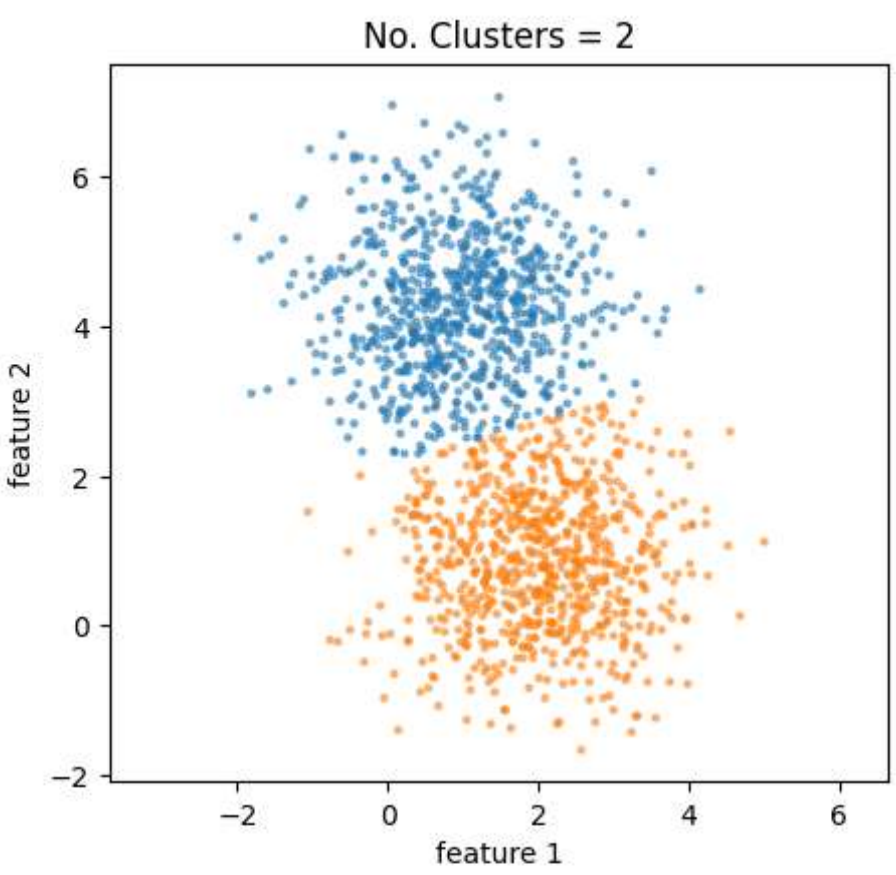


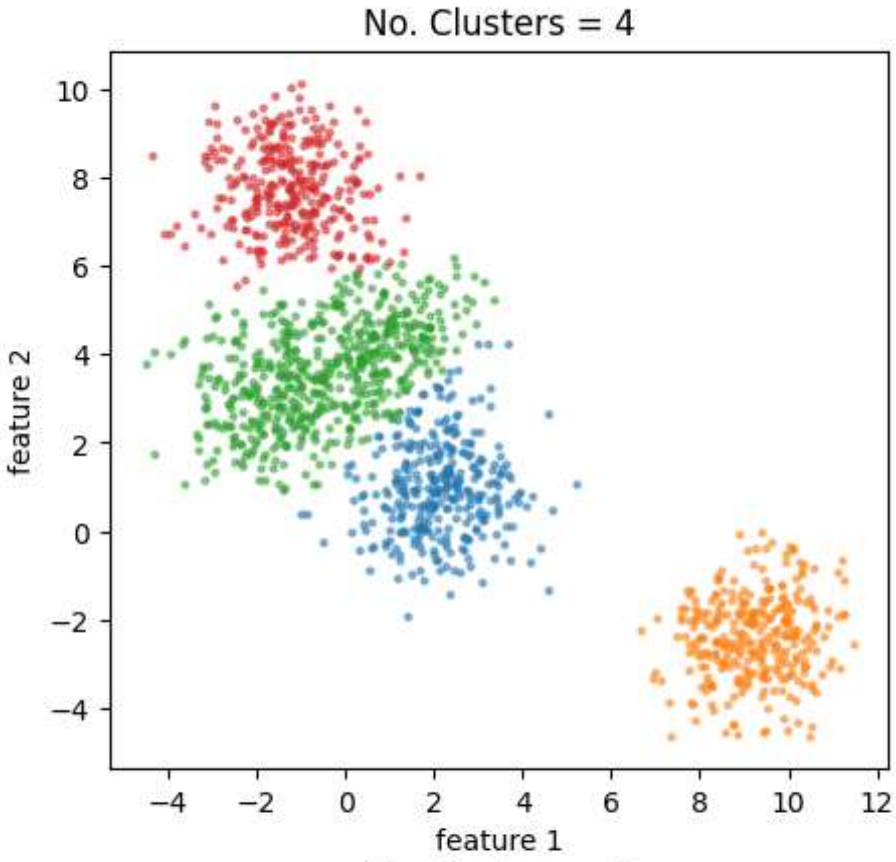
The selected number of clusters for each dataset is:

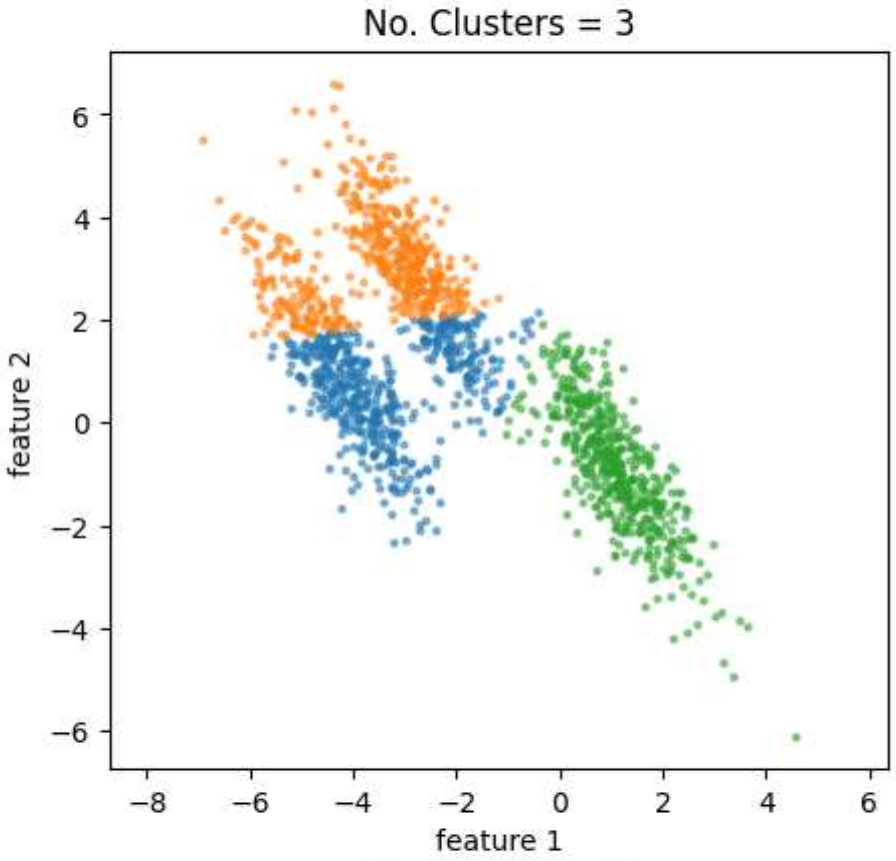
Dataset 1: 2
Dataset 2: 4
Dataset 3: 3
Dataset 4: 3
Dataset 5: 2

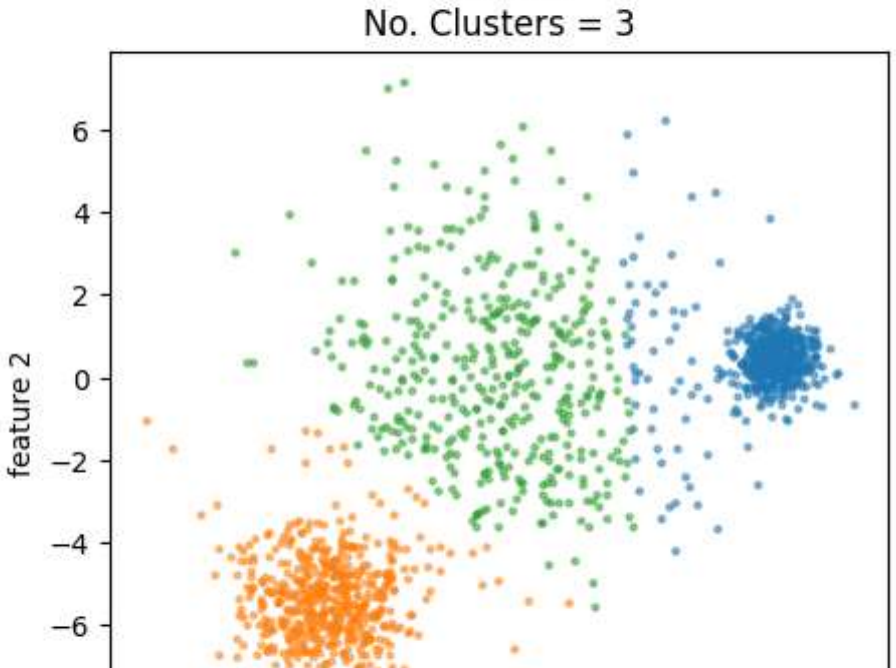
Plotting the Clusters

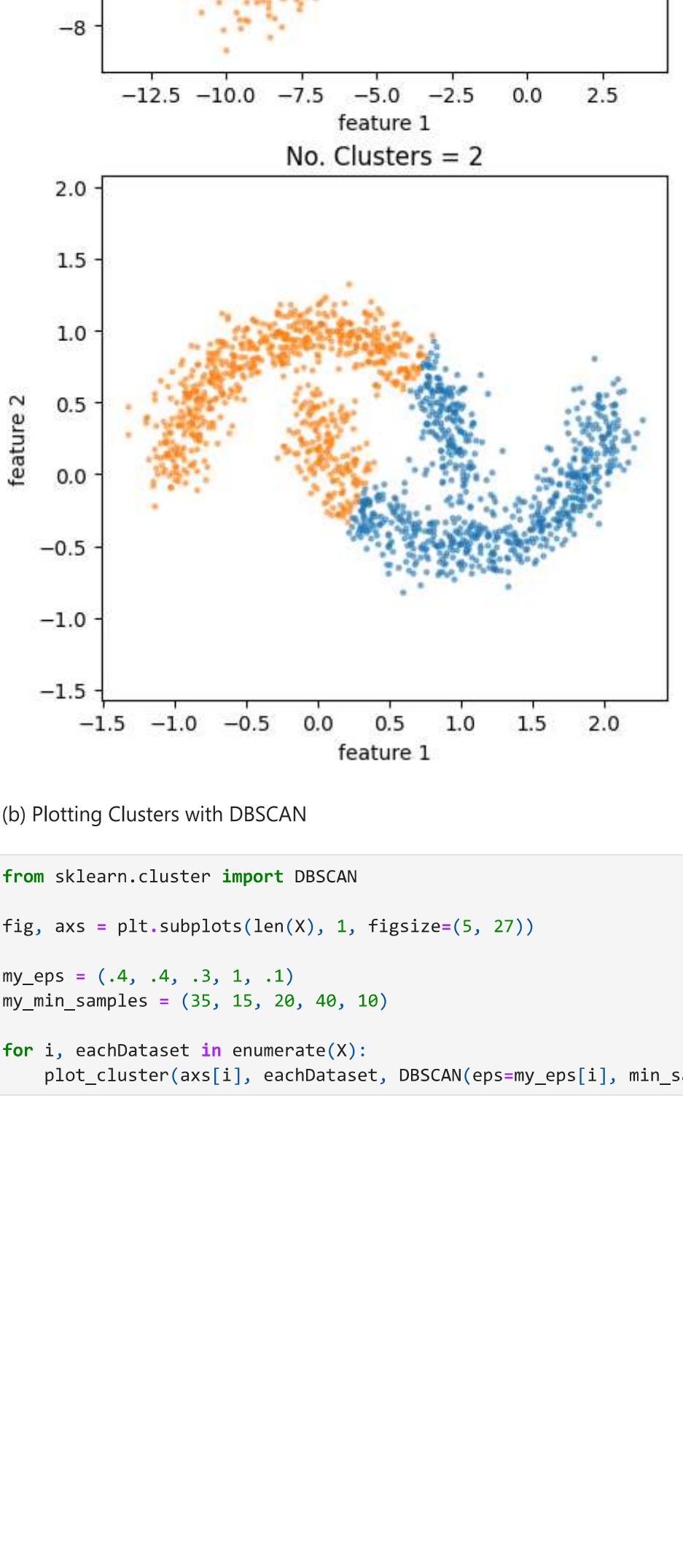
In [ ]:
```
fig, axs = plt.subplots(len(X), 1, figsize=(5, 27))

clusters = (2, 4, 3, 3, 2)

for i, eachDataset in enumerate(X):
    plot_cluster(axs[i], eachDataset, KMeans(n_clusters=clusters[i], random_state=0).fit_predict(eachDataset))
```


No. Clusters = 2
feature 2
feature 1
6
4
2
0
−2
−2
0
2
4
6


No. Clusters = 4
feature 2
feature 1
10
8
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8
10
12


No. Clusters = 3
feature 2
feature 1
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

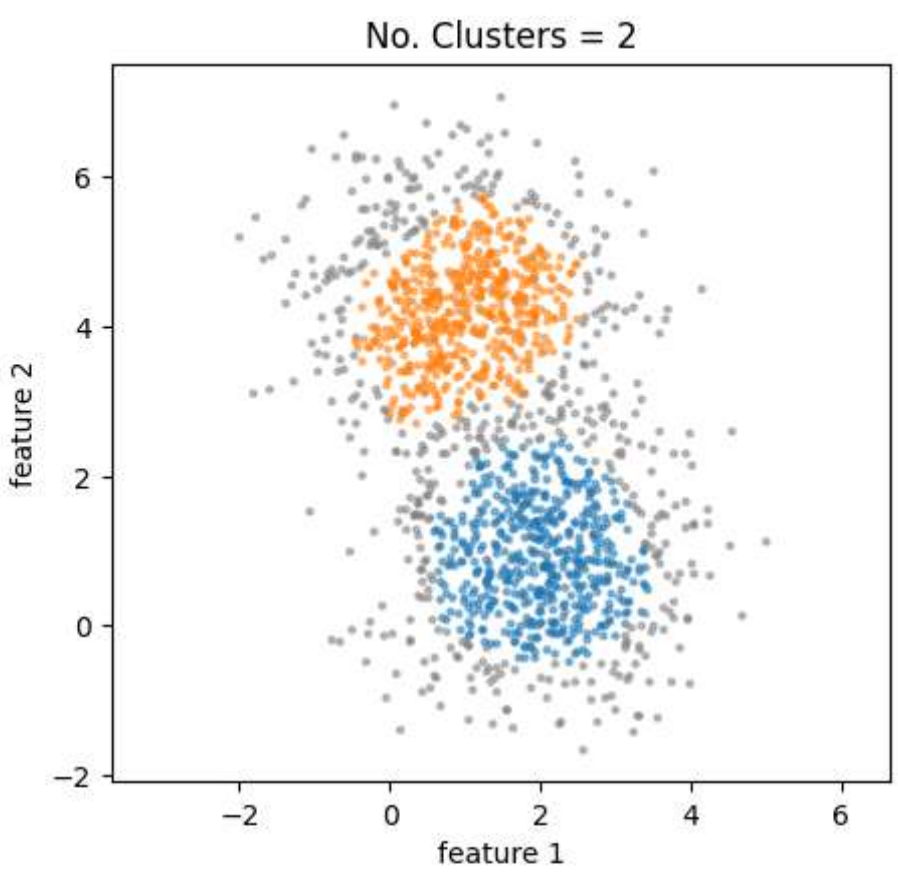

No. Clusters = 3
feature 2
6
4
2
0
−2
−4
−6

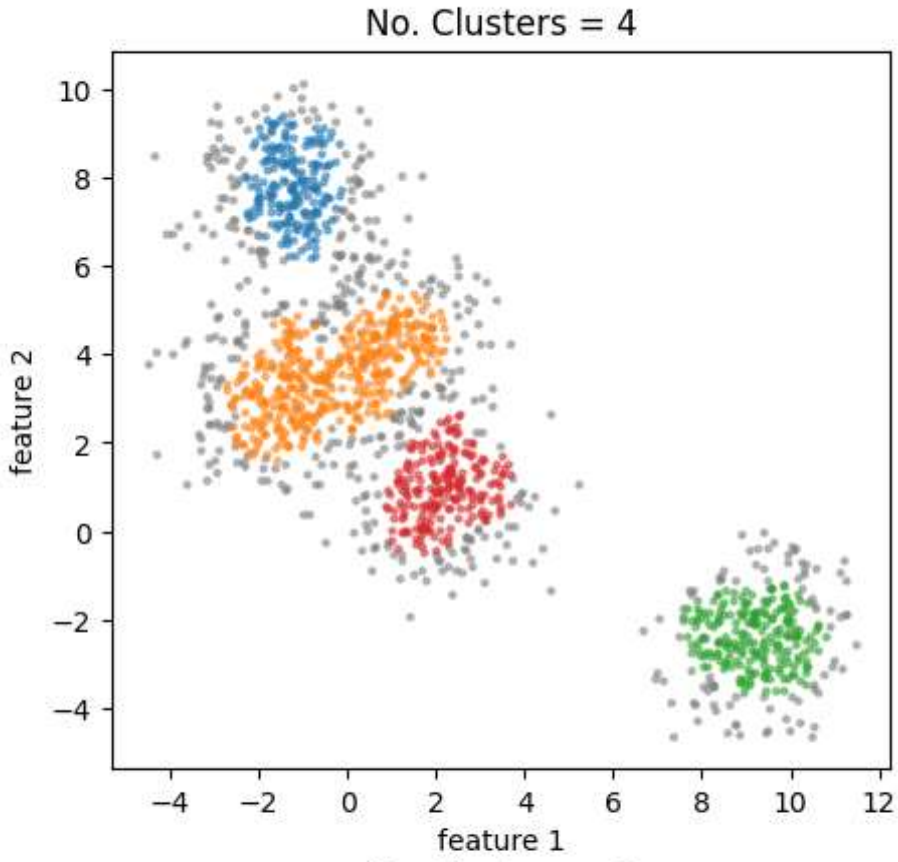


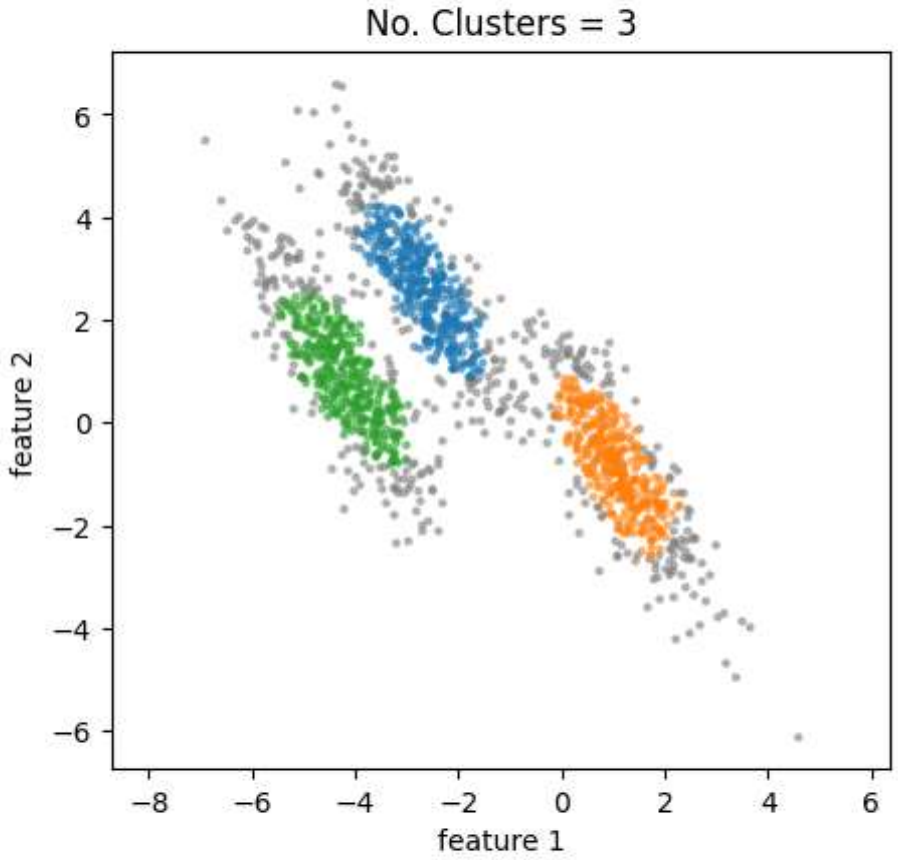
(b) Plotting Clusters with DBSCAN

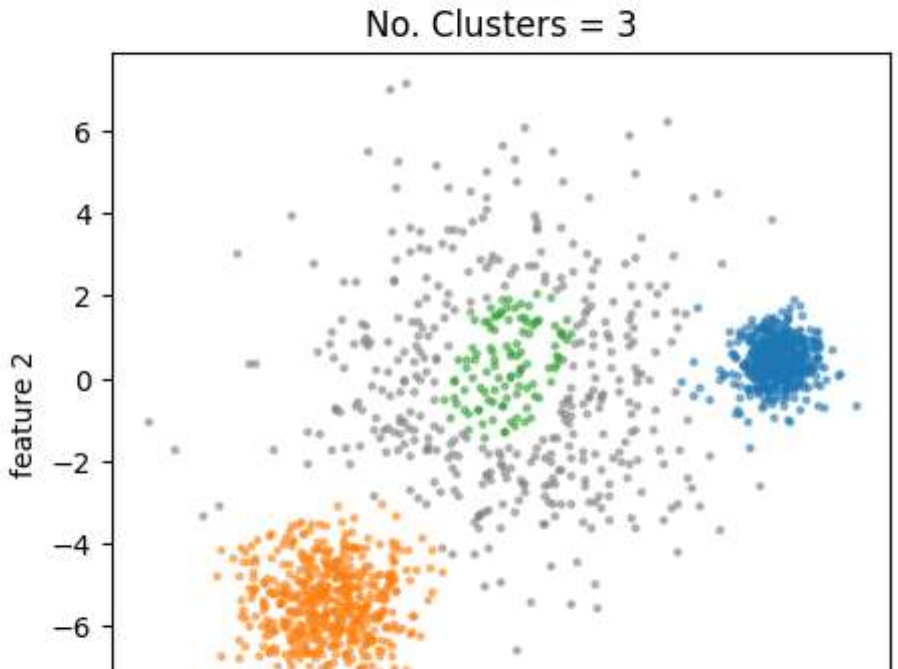
In [ ]:

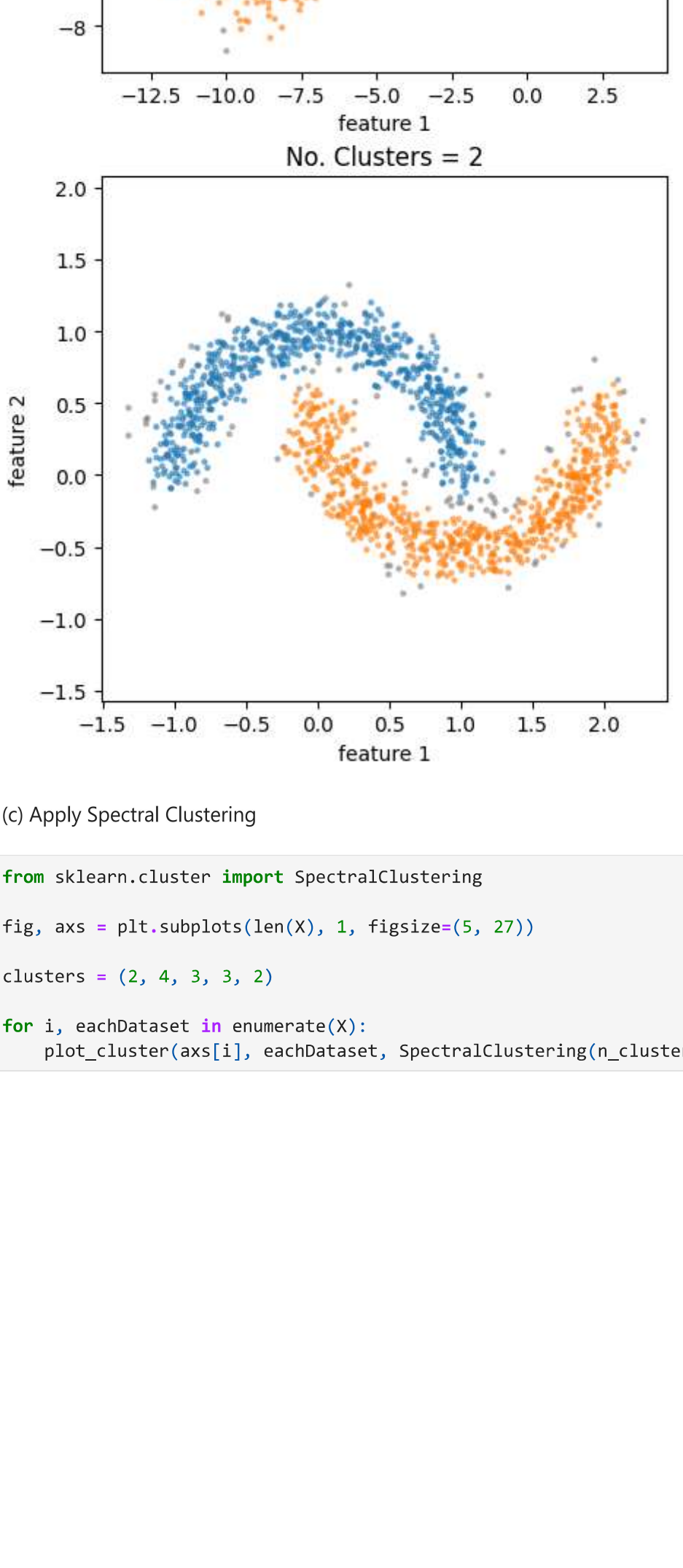
```
from sklearn.cluster import DBSCAN

fig, axs = plt.subplots(len(X), 1, figsize=(5, 27))

my_eps = (.4, .4, .3, 1, .1)
my_min_samples = (35, 15, 20, 40, 10)

for i, eachDataset in enumerate(X):
    plot_cluster(axs[i], eachDataset, DBSCAN(eps=my_eps[i], min_samples = my_min_samples[i]).fit_predict(eachDataset))
```


No. Clusters = 2
feature 2
feature 1
6
4
2
0
−2
−2
0
2
4
6


No. Clusters = 4
feature 2
feature 1
10
8
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8
10
12


No. Clusters = 3
feature 2
feature 1
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

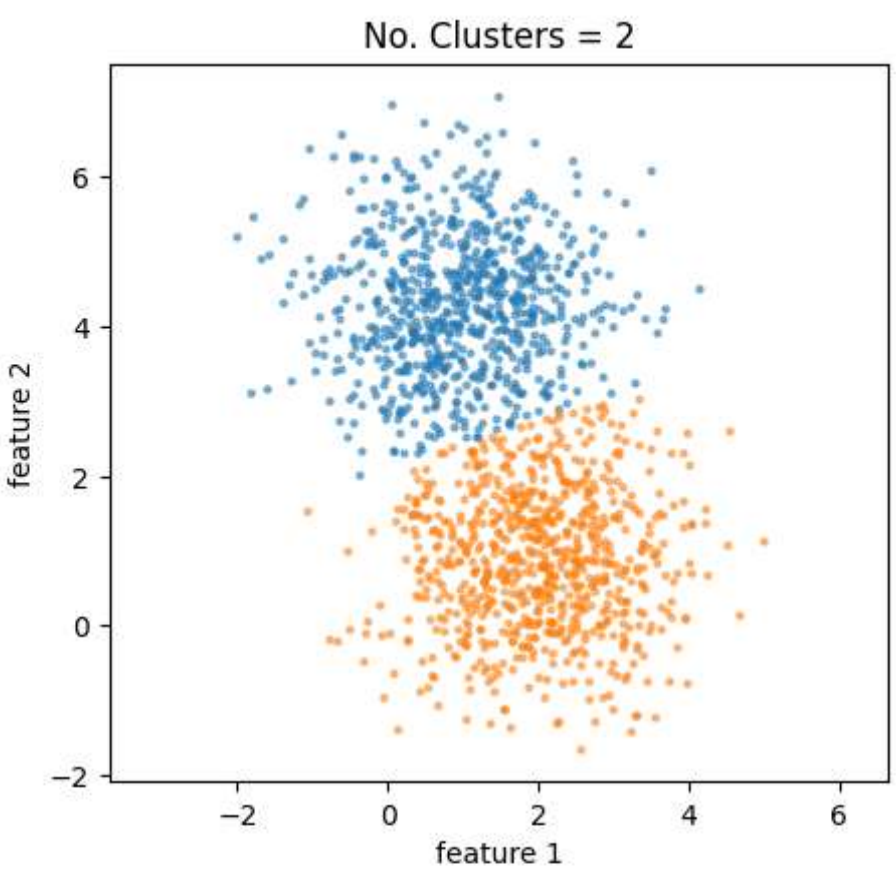

No. Clusters = 3
feature 2
6
4
2
0
−2
−4
−6

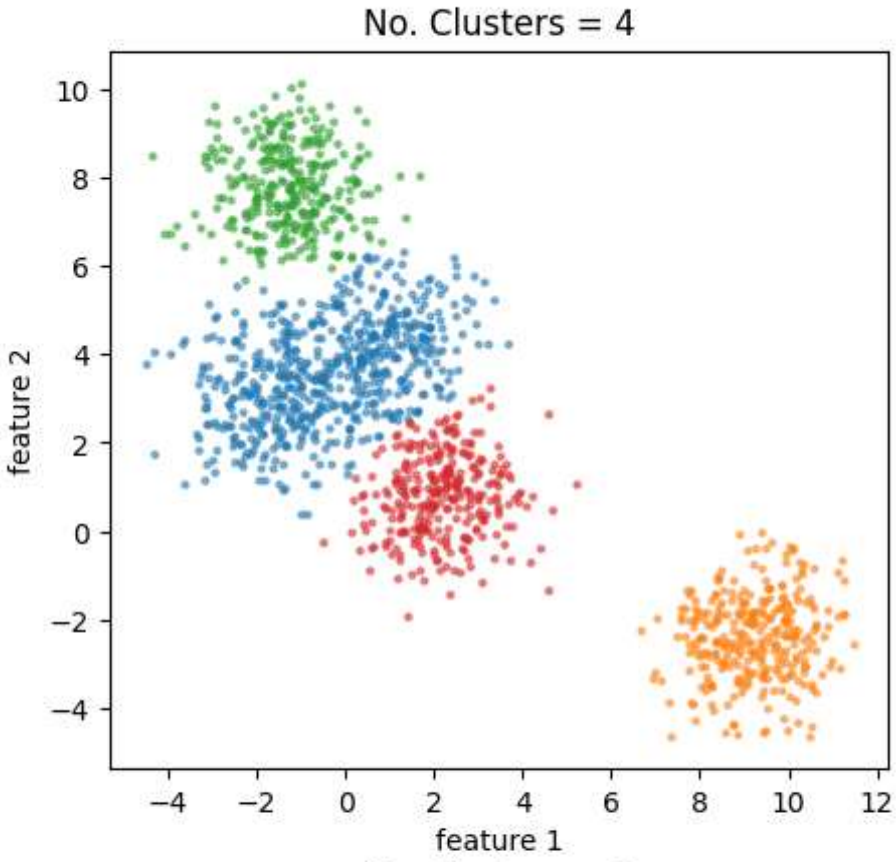


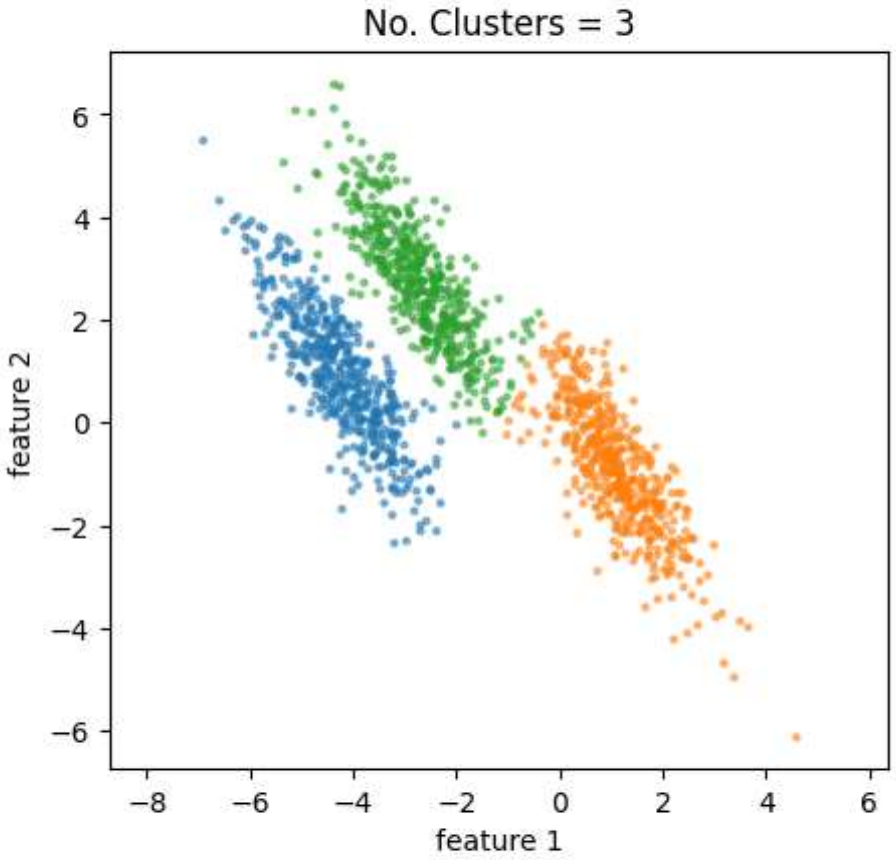
(c) Apply Spectral Clustering

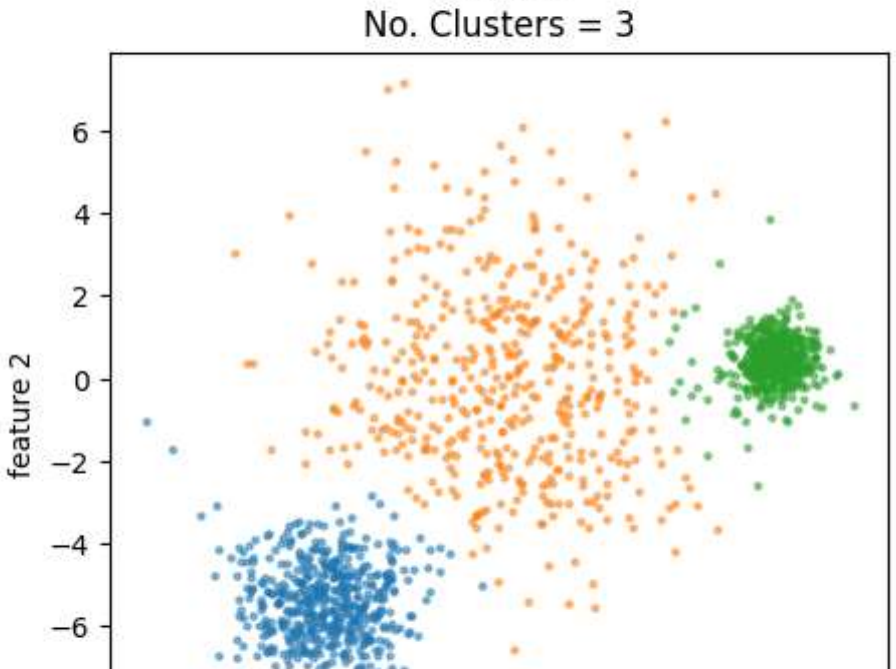
In [ ]:

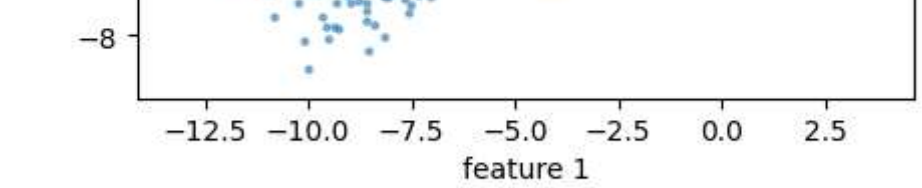
```
from sklearn.cluster import SpectralClustering

fig, axs = plt.subplots(len(X), 1, figsize=(5, 27))

clusters = (2, 4, 3, 3, 2)

for i, eachDataset in enumerate(X):
    plot_cluster(axs[i], eachDataset, SpectralClustering(n_clusters=clusters[i], random_state=0).fit_predict(eachDataset))
```

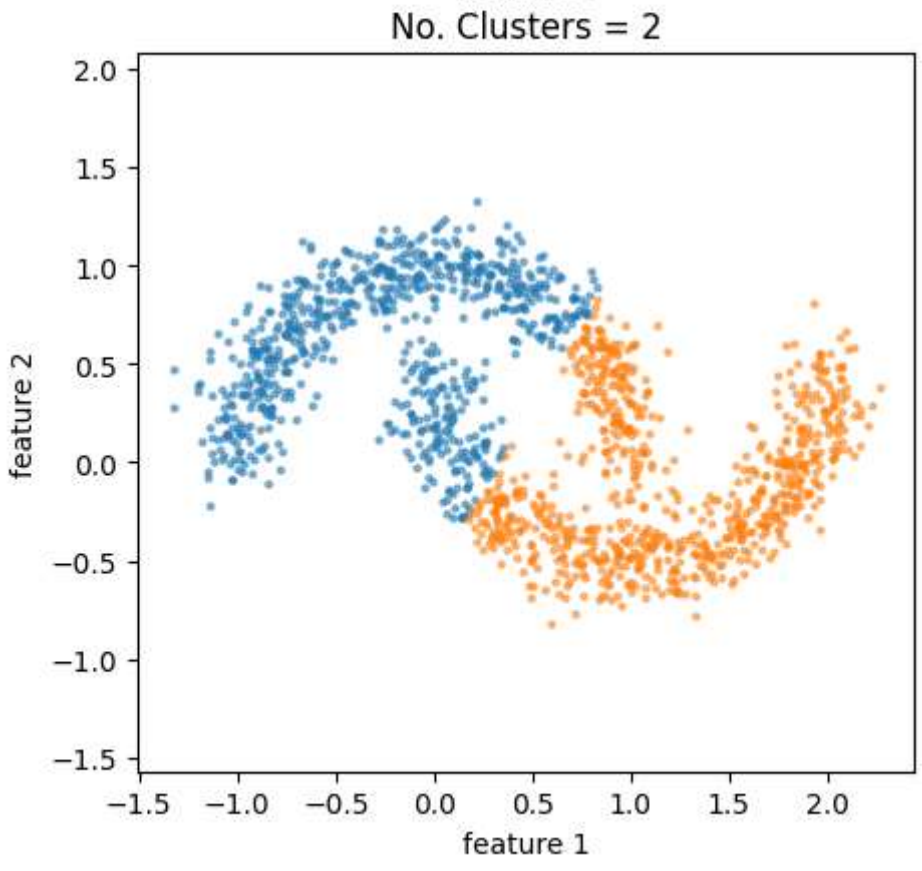

No. Clusters = 2
feature 2
feature 1
6
4
2
0
−2
−2
0
2
4
6


No. Clusters = 4
feature 2
feature 1
10
8
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8
10
12


No. Clusters = 3
feature 2
feature 1
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


No. Clusters = 3
feature 2
6
4
2
0
−2
−4
−6





(d) Reviewing the Clustering Techniques

For dataset 1, both K Means and Spectral Clustering worked equally well, but K Means is computationally less expensive. DBSCAN performed worse, leaving many points out of an expected cluster, and required substantially more hyperparameter tuning. K Means and Spectral Clustering required much less hyperparameter tuning.

For dataset2, both K Means and Spectral Clustering worked equally well, but K Means is computationally less expensive. DBSCAN performed worse, leaving many points out of an expected cluster, and required substantially more hyperparameter tuning. K Means and Spectral Clustering required much less hyperparameter tuning.
For dataset 3, Spectral Clustering performed best, and K Means performed worst. DBSCAN performed better than K Means, but again, left many points out of an expected cluster, and required substantially more hyperparameter tuning. K Means and Spectral Clustering required much less hyperparameter tuning.
For dataset 4, both K Means and Spectral Clustering worked equally well, but K Means is computationally less expensive. DBSCAN performed worse, leaving many points out of an expected cluster, and had a difficult time clustering the less dense cluster altogether. Additionally, DBSCAN required substantially more hyperparameter tuning. K Means and Spectral Clustering required much less hyperparameter tuning.
For dataset 1, both K Means and Spectral Clustering performed worst. DBSCAN performed best, and hardly left any points out of an expected cluster. DBSCAN required substantially more hyperparameter tuning. K Means and Spectral Clustering required much less hyperparameter tuning.

3

[25 points] Dimensionality reduction and visualization of digits with PCA and t-SNE

(a) Reduce the dimensionality of the data with PCA for data visualization. Load the `scikit-learn` digits dataset (code provided to do this below). Apply PCA and reduce the data (with the associated cluster labels 0-9) into a 2-dimensional space. Plot the data with labels in this two dimensional space (labels can be colors, shapes, or using the actual numbers to represent the data - definitely include a legend in your plot).

(b) Create a plot showing the cumulative fraction of variance explained as you incorporate from 1 through all D principal components of the data (where D is the dimensionality of the data).

- What fraction of variance in the data is UNEXPLAINED by the first two principal components of the data?
- Briefly comment on how this may impact how well-clustered the data are.

You can use the `explained_variance_` *attribute of the PCA module in* `scikit-learn` *to assist with this question*

(c) Reduce the dimensionality of the data with t-SNE for data visualization. T-distributed stochastic neighborhood embedding (t-SNE) is a nonlinear dimensionality reduction technique that is particularly adept at embedding the data into lower 2 or 3 dimensional spaces. Apply t-SNE using the `scikit-learn` implementation to the digits dataset and plot it in 2-dimensions (with associated cluster labels 0-9). You may need to adjust the parameters to get acceptable performance. You can read more about how to use t-SNE effectively here.

(d) Briefy compare/contrast the performance of these two techniques.

- Which seemed to cluster the data best and why?
- Notice that while t-SNE has a `fit` method and a `fit_transform` method, these methods are actually identical, and there is no `transform` method. Why is this? What implications does this imply for using this method?

Note: Remember that you typically will not have labels available in most problems.

Code is provided for loading the data below.

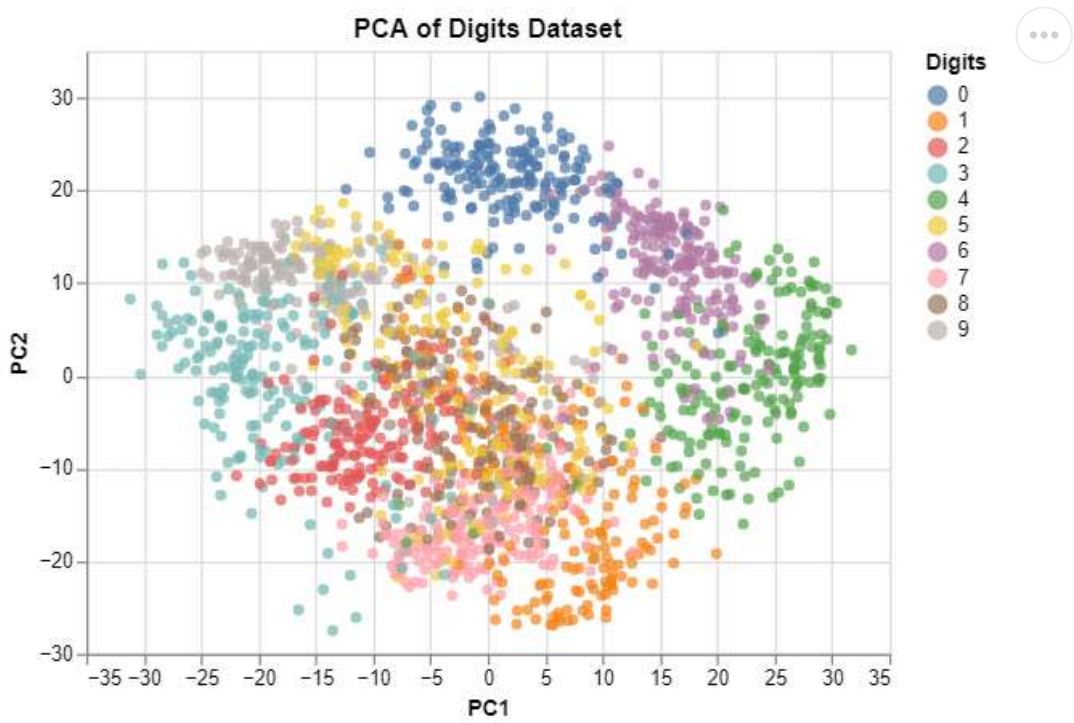
In [ ]:
```
################################
# Load the data
################################
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Load dataset
digits = datasets.load_digits()
n_sample = digits.target.shape[0]
n_feature = digits.images.shape[1] * digits.images.shape[2]
X_digits = np.zeros((n_sample, n_feature))
for i in range(n_sample):
    X_digits[i, :] = digits.images[i, :, :].flatten()
y_digits = digits.target
```

ANSWER

(a) Plotting Digits with Principal Component Analysis

In [ ]:
```
import altair as alt
import pandas as pd

alt.Chart(
    pd.DataFrame(
        np.hstack(
            (PCA(n_components=2).fit_transform(X_digits), y_digits.reshape(-1, 1))
        ),
        columns=["PC1", "PC2", "Digits"],
    ), title = "PCA of Digits Dataset"
).mark_circle().encode(x="PC1", y="PC2", color="Digits:N")
```

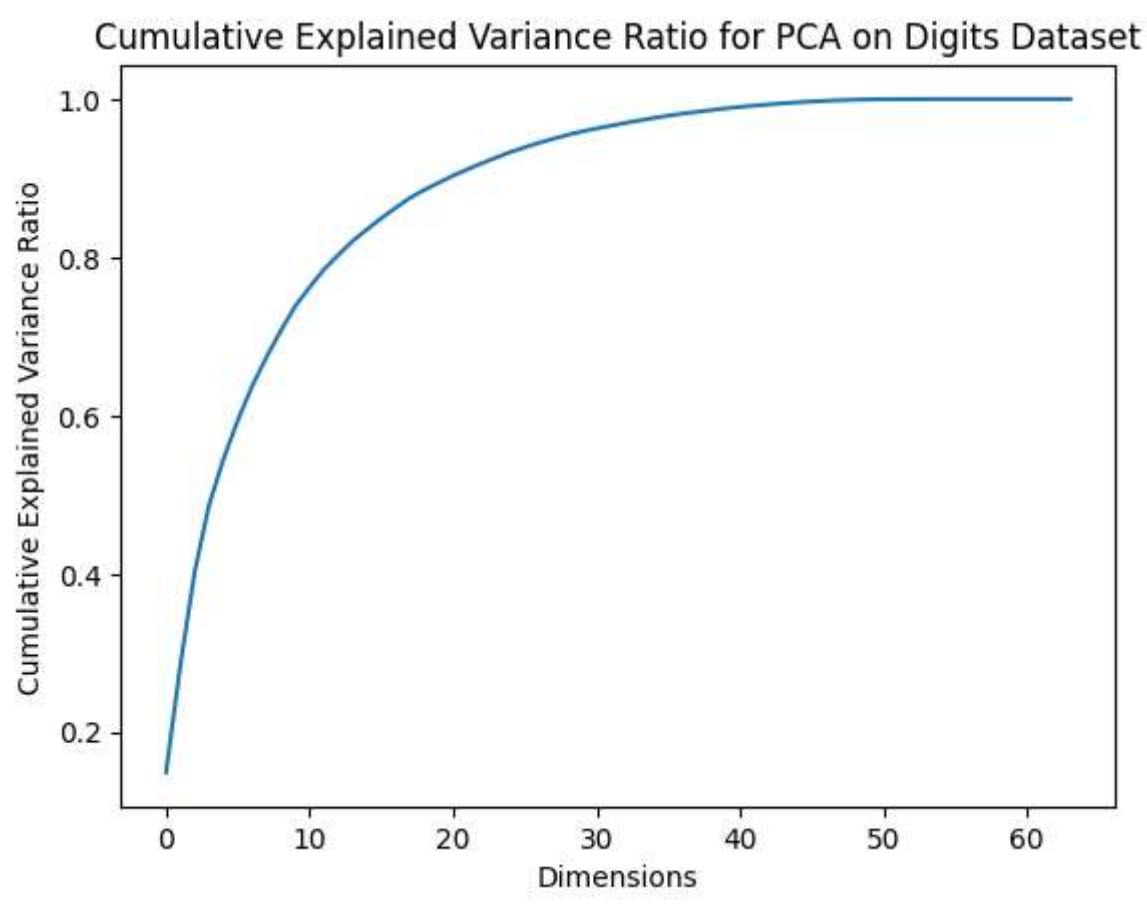
Out[ ]:



(b) Plotting proportion of variance explained by dimensions with PCA

In [ ]:
```
explainedVarianceRatio = PCA(n_components=64).fit(X_digits).explained_variance_ratio_
plt.plot(explainedVarianceRatio.cumsum())
plt.title('Cumulative Explained Variance Ratio for PCA on Digits Dataset')
plt.xlabel('Dimensions')
plt.ylabel('Cumulative Explained Variance Ratio')
```

Out[ ]: Text(0, 0.5, 'Cumulative Explained Variance Ratio')



In [ ]:
```
print(f"The proportion of the variance explained by the first two principal components is {explainedVarianceRatio[:2].sum() *100:.
```

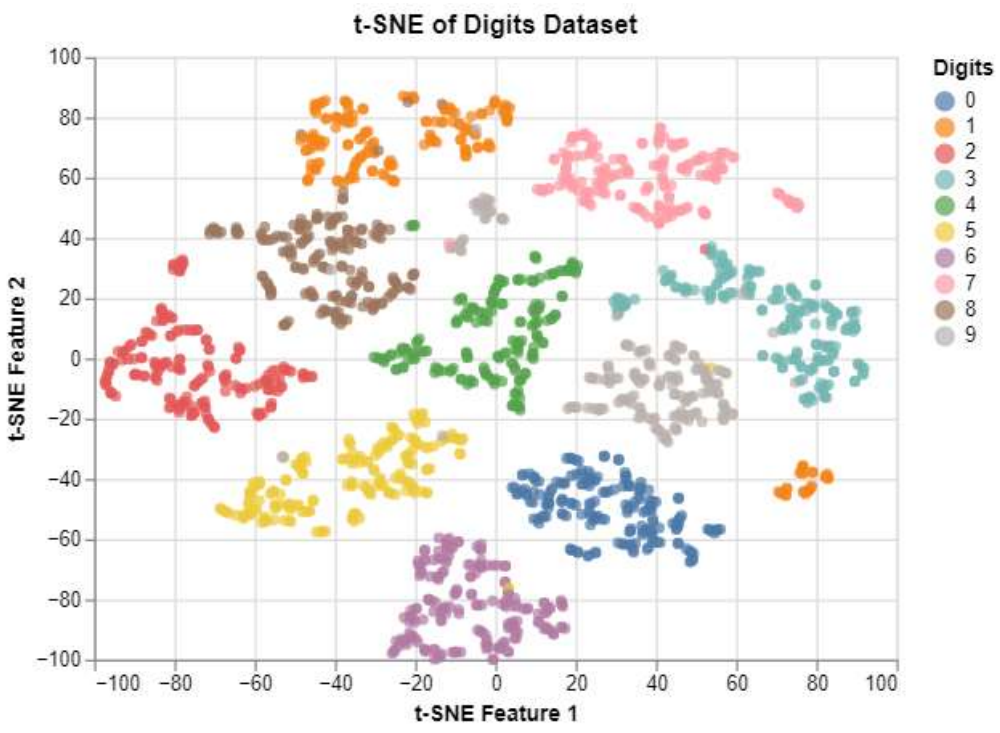
```
The proportion of the variance explained by the first two principal components is 28.51 %.
```

Since only a small portion of the variance is explained by the first two principal components, clustering in two dimensions will under-represent how well the data clusters.

(c) Plotting Clusters with t-SNE

In [ ]:
```
alt.Chart(
    pd.DataFrame(
        np.hstack(
            (TSNE(n_components=2, perplexity = 5).fit_transform(X_digits), y_digits.reshape(-1, 1))
        ),
        columns=["t-SNE Feature 1", "t-SNE Feature 2", "Digits"],
    ), title = "t-SNE of Digits Dataset"
).mark_circle().encode(x="t-SNE Feature 1", y="t-SNE Feature 2", color="Digits:N")
```

Out[ ]:



(d) Comparing PCA with t-SNE

t-SNE appears to perform much better than PCA on this dataset, because it manages to separate the clusters much better. This is due to the fact that t-SNE is a non-linear dimensionality reduction, while PCA is a linear dimensionality reduction, and the features of the digits are non-linear (because it is image data).

t-SNE does not have a transform function, because it's non-linear transform requires all of the data to complete its dimensionality reduction. This means that if you had a validation set, or any other set, that the t-SNE was not fit on, but that you wanted to visualize in that space, you could not do this, because it would need to be fit with that data from the start.