Assignment 5 - Kaggle Competition and Unsupervised Learning

Daniela Jiménez Lara

Netid: dj216

Names of students you worked with on this assignment: Barbara Flores, Shaila Güereca

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.

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 at the Kaggle website (see the Ed post).

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 (see Ed Discussions for the link). 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 [ ]:

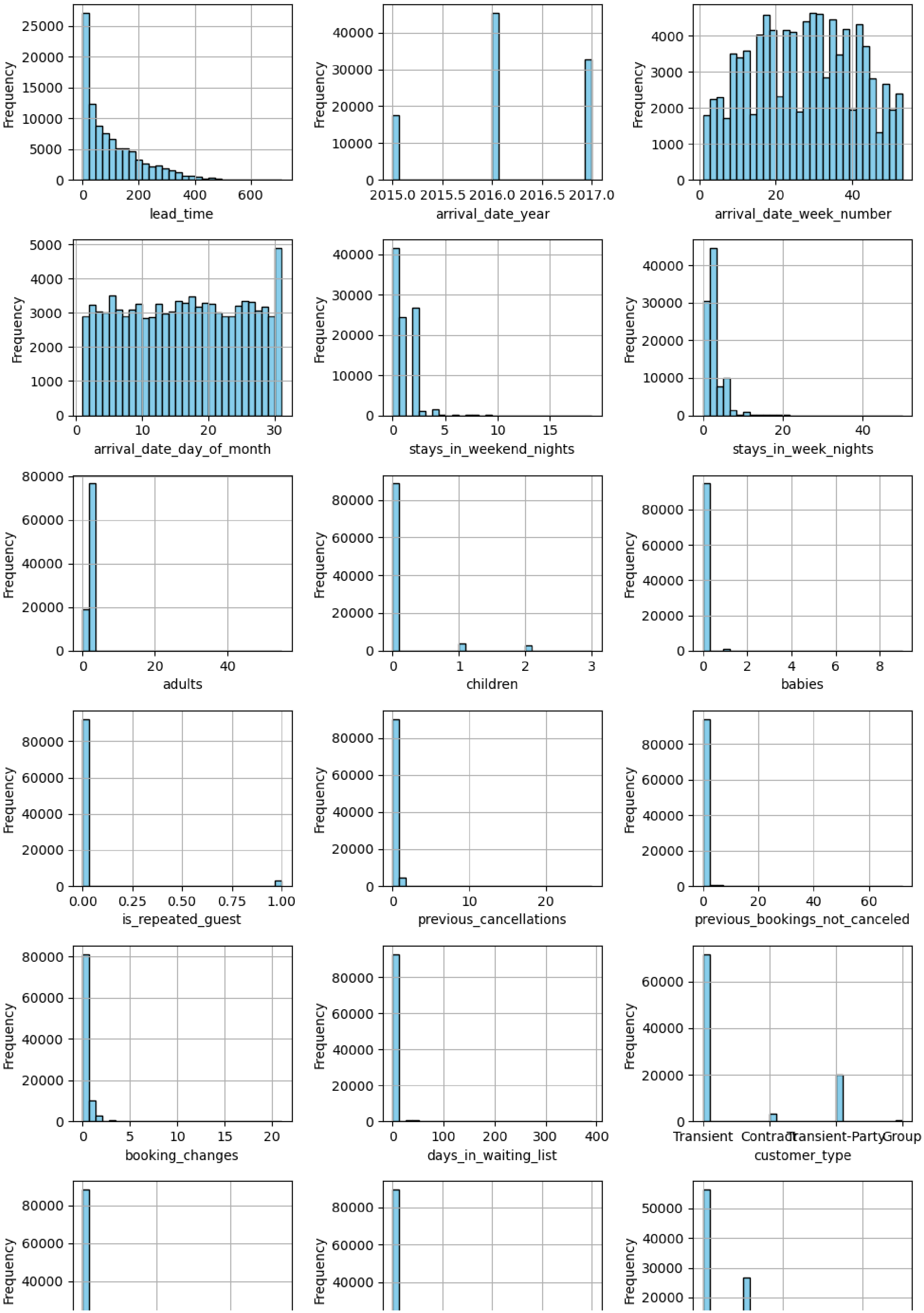
```python
import pandas as pd
import numpy as np
import pickle

################################
# Load the data
################################
data = pd.read_pickle("./data/a5_q1.pkl")

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")
```

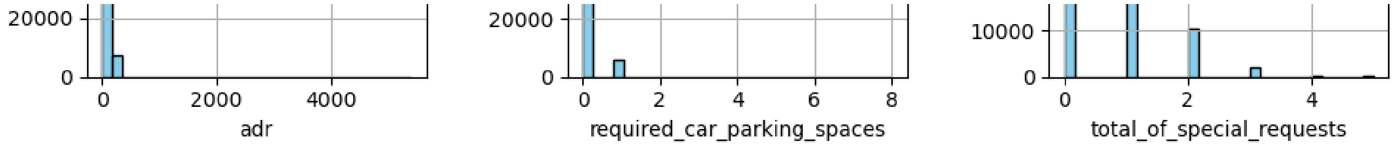
ANSWER

a)

In [ ]:

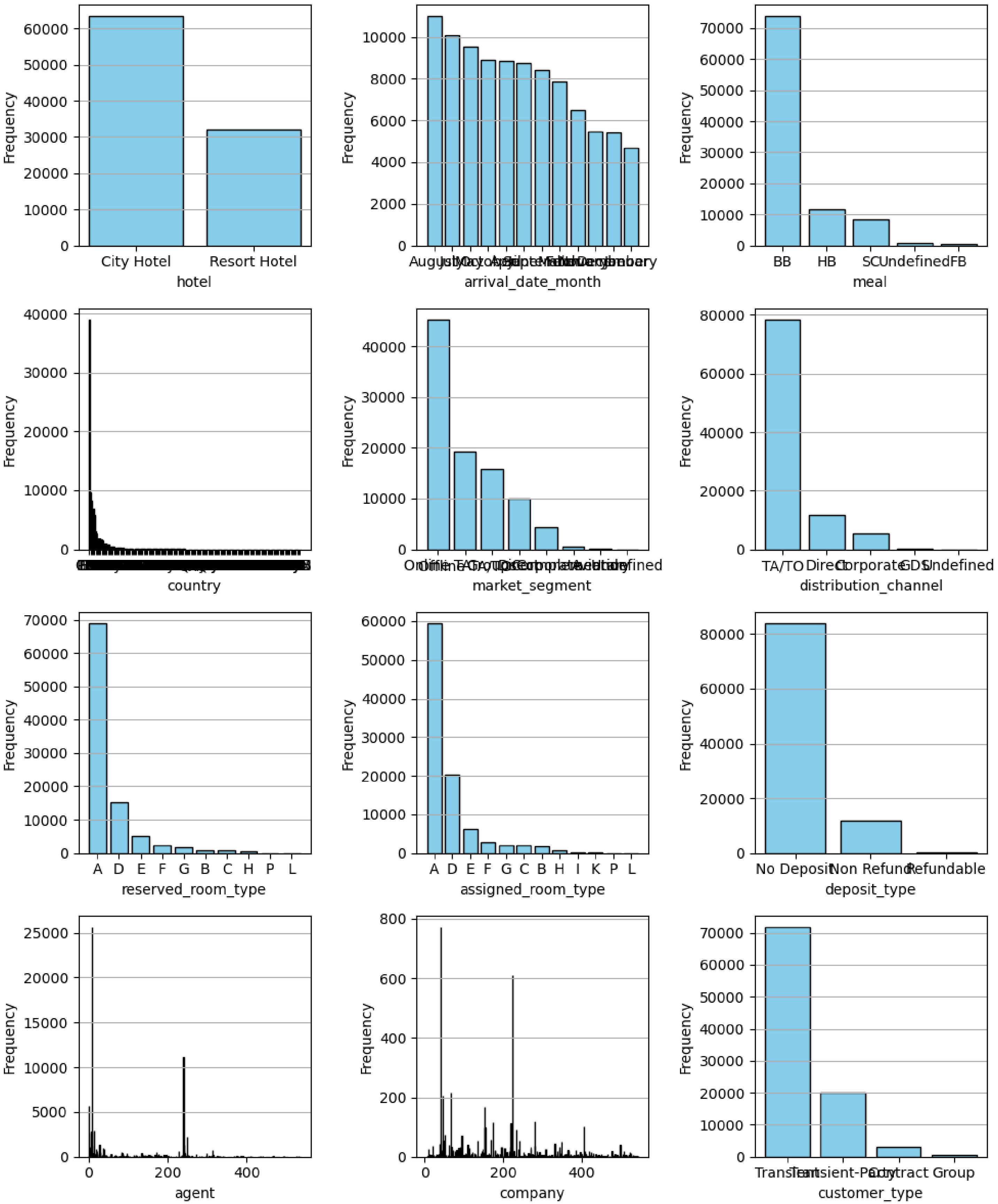
```
import matplotlib.pyplot as plt
import pandas as pd

num_vars = [
    "lead_time",
    "arrival_date_year",
    "arrival_date_week_number",
    "arrival_date_day_of_month",
    "stays_in_weekend_nights",
    "stays_in_week_nights",
    "adults",
    "children",
    "babies",
    "is_repeated_guest",
    "previous_cancellations",
    "previous_bookings_not_canceled",
    "booking_changes",
    "days_in_waiting_list",
    "customer_type",
    "adr",
    "required_car_parking_spaces",
    "total_of_special_requests",
]

char_vars = [
    "hotel",
    "arrival_date_month",
    "meal",
    "country",
    "market_segment",
    "distribution_channel",
    "reserved_room_type",
    "assigned_room_type",
    "deposit_type",
    "agent",
    "company",
    "customer_type",
]


# Create a grid of histograms
num_rows = 6
num_cols = 3
fig, axs = plt.subplots(num_rows, num_cols, figsize=(10, 15))

axs = axs.flatten()

for i, column_name in enumerate(num_vars):
    ax = axs[i]
    ax.hist(X_train_original[column_name], bins=30, color="skyblue", edgecolor="black")
    ax.set_xlabel(column_name)
    ax.set_ylabel("Frequency")
    ax.grid(True)

plt.tight_layout()
plt.show()
```


Frequency
lead_time
0
200
400
600
5000
10000
15000
20000
25000
arrival_date_year
2015.0
2015.5
2016.0
2016.5
2017.0
10000
20000
30000
40000
arrival_date_week_number
20
40
1000
2000
3000
4000
arrival_date_day_of_month
10
20
30
5000
stays_in_weekend_nights
5
10
15
stays_in_week_nights
adults
80000
60000
children
1
2
3
babies
4
6
8
is_repeated_guest
0.00
0.25
0.50
0.75
1.00
previous_cancellations
previous_bookings_not_canceled
booking_changes
days_in_waiting_list
100
300
customer_type
Transient
Contract
Transient-Party
Group
50000



In [ ]:

```
num_rows = 4
num_cols = 3

fig, axs = plt.subplots(num_rows, num_cols, figsize=(10, 12))
axs = axs.flatten()

for i, column_name in enumerate(char_vars):
    ax = axs[i]
    values = X_train_original[column_name].value_counts()
    ax.bar(values.index, values.values, color="skyblue", edgecolor="black")
    ax.set_xlabel(column_name)
    ax.set_ylabel("Frequency")
    ax.grid(axis="y")

# Adjust layout
plt.tight_layout()
plt.show()
```



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

```
<class 'pandas.core.frame.DataFrame'>
Index: 95512 entries, 0 to 119389
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   hotel                           95512 non-null  object 
 1   lead_time                       95512 non-null  int64  
 2   arrival_date_year               95512 non-null  int64  
 3   arrival_date_month              95512 non-null  object 
 4   arrival_date_week_number        95512 non-null  int64  
 5   arrival_date_day_of_month       95512 non-null  int64  
 6   stays_in_weekend_nights         95512 non-null  int64  
 7   stays_in_week_nights            95512 non-null  int64  
 8   adults                          95512 non-null  int64  
 9   children                        95510 non-null  float64
 10  babies                          95512 non-null  int64  
 11  meal                            95512 non-null  object 
 12  country                         95117 non-null  object 
 13  market_segment                  95512 non-null  object 
 14  distribution_channel            95512 non-null  object 
 15  is_repeated_guest               95512 non-null  int64  
 16  previous_cancellations          95512 non-null  int64  
 17  previous_bookings_not_canceled  95512 non-null  int64  
 18  reserved_room_type              95512 non-null  object 
 19  assigned_room_type              95512 non-null  object 
 20  booking_changes                 95512 non-null  int64  
 21  deposit_type                    95512 non-null  object 
 22  agent                           82431 non-null  float64
 23  company                         5453 non-null   float64
 24  days_in_waiting_list            95512 non-null  int64  
 25  customer_type                   95512 non-null  object 
 26  adr                             95512 non-null  float64
 27  required_car_parking_spaces     95512 non-null  int64  
 28  total_of_special_requests       95512 non-null  int64  
dtypes: float64(4), int64(15), object(10)
memory usage: 21.9+ MB
```

(c) Select, train, and compare models.

Logistic Regression

In [ ]:

```python
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# from sklearn.metrics import log_loss

data = pd.read_pickle("./data/a5_q1.pkl")


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"]


X_train_ohe = X_train_ohe.fillna(0)
X_test_ohe = X_test_ohe.fillna(0)

X_train_80, X_train_20, y_train_80, y_train_20 = train_test_split(
    X_train_ohe, y_train, test_size=0.2, random_state=42
)
```

In [ ]:

```python
from sklearn.model_selection import RandomizedSearchCV
import warnings

warnings.simplefilter("ignore")
param_dist = {
    "C": np.logspace(-4, 4, 20),
    "penalty": ["l1", "l2"],
    "solver": ["liblinear", "saga"],
}
lgr_rc = LogisticRegression()
# Perform randomized search cross-validation
random_search = RandomizedSearchCV(
    estimator=lgr_rc, param_distributions=param_dist, n_iter=10, cv=5
)
random_search.fit(X_train_80, y_train_80)

# Print the best hyperparameters
print("Best hyperparameters:", random_search.best_params_)
```

```
Best hyperparameters: {'solver': 'liblinear', 'penalty': 'l1', 'C': 78.47599703514607}
```

In [ ]:

```
from sklearn.metrics import roc_curve, roc_auc_score

# Logistic regression
from sklearn.linear_model import LogisticRegression

# .fillna

lgr = LogisticRegression(
    random_state=0,
    solver="liblinear",
    penalty="l1",
    C=78.475,
    max_iter=2500,
).fit(X_train_80, y_train_80)

y_prob_lgr = lgr.predict_proba(X_train_20)[:, 1]
y_pred_lgr = lgr.predict(X_train_20)[:, 1]

fpr_lgr, tpr_lgr, th_lgr = roc_curve(y_train_20, y_prob_lgr)
auc_score_lgr = roc_auc_score(y_train_20, y_prob_lgr)


print("AUC_score validation", auc_score_lgr)
```

```
AUC_score validation 0.9118886040298004
```

In [ ]:

```
print("Run time on train: 1s")
print("Run time on validation: 1.5s")

run_time_train = {"lgr": 1}
run_time_validation = {"lgr": 1.5}
```

```
Run time on train: 1s
Run time on validation: 1.5s
```

Random Forest

In [ ]:

```
# random forrest with feature selecction
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

rfo = RandomForestClassifier(n_estimators=2000)

sfm = SelectFromModel(rfo, threshold=0.000100)
sfm.fit(X_train_80, y_train_80)

# Transform the data to select features
X_train_80_fs = sfm.transform(X_train_80)
X_train_20_fs = sfm.transform(X_train_20)
```

In [ ]:

```
from sklearn.metrics import roc_curve, roc_auc_score

rf_fs = RandomForestClassifier(n_estimators=2000)
rf_fs.fit(X_train_80_fs, y_train_80)

y_prob_rf_fs = rf_fs.predict_proba(X_train_20_fs)[:, 1]
y_pred_rf_fs = rf_fs.predict(X_train_20_fs)

fpr_rf_fs, tpr_rf_fs, th_rf_fs = roc_curve(y_train_20, y_prob_rf_fs)
```

```
auc_score_rf_fs = roc_auc_score(y_train_20, y_prob_rf_fs)

fpr_rf_fs, tpr_rf_fs, th_rf_fs = roc_curve(y_train_20, y_prob_rf_fs)

auc_score_rf_fs = roc_auc_score(y_train_20, y_prob_rf_fs)
print("AUC_score validation RF with feature selection", auc_score_rf_fs)
```

```
AUC_score validation RF with feature selection 0.9593597495866963
```

In [ ]:

```
print("Run time on train: 2m 46s")
print("Run time validation: 10 m 12 s")

run_time_train["RF"] = 166
run_time_validation["RF"] = 612
```

```
Run time on train: 2m 46s
Run time validation: 10 m 12 s
```

K-nearest neighbors

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train_80, y_train_80)

y_prob_ne = neigh.predict_proba(X_train_20)[:, 1]
y_pred_ne = neigh.predict(X_train_20)
fpr_ne, tpr_ne, th_ne = roc_curve(y_train_20, y_prob_ne)

auc_score_ne = roc_auc_score(y_train_20, y_prob_ne)
```

In [ ]:

```
print("Run time  train: 14m 30s")
print("Run time validation: 3s ")

run_time_train["KNN"] = 870
run_time_validation["KNN"] = 3
```

```
Run time  train: 14m 30s
Run time validation: 3s
```

In [ ]:

```
#### Neural networks
from sklearn.neural_network import MLPClassifier
from scipy.stats import loguniform
import warnings

warnings.simplefilter("ignore")

# bz_list = [20, 50, 100, 250, 500]
lr_log_list = loguniform(1e-5, 1e0).rvs(20)
# rp_log_list = loguniform(1e-8, 1e2).rvs(20)


param_dict = {
    "learning_rate_init": lr_log_list,
}

mcv2 = MLPClassifier()

random_search = RandomizedSearchCV(
    estimator=mcv2,
    param_distributions=param_dict,
    n_iter=10,
    scoring="roc_auc",
    cv=3,
    random_state=1,
    n_jobs=-1,
)

random_search.fit(X_train_80, y_train_80)

print("Best hyperparameters:", random_search.best_params_)
```

```
/Users/danielajimenez/miniconda3/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
91: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  warnings.warn(
/Users/danielajimenez/miniconda3/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
91: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  warnings.warn(
/Users/danielajimenez/miniconda3/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
91: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  warnings.warn(
/Users/danielajimenez/miniconda3/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
91: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  warnings.warn(
/Users/danielajimenez/miniconda3/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
91: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  warnings.warn(
/Users/danielajimenez/miniconda3/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
91: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  warnings.warn(
Best hyperparameters: {'learning_rate_init': 0.008368229549763117}
```

```
In [ ]: mlp = MLPClassifier(random_state=1, learning_rate_init=0.008368229549763117).fit(
            X_train_80, y_train_80
        )
```

```
In [ ]: y_prob_mlp = mlp.predict_proba(X_train_20)[:, 1]
        y_pred_mlp = mlp.predict(X_train_20)
        fpr_mlp, tpr_mlp, th_mlp = roc_curve(y_train_20, y_prob_mlp)

        auc_score_mlp = roc_auc_score(y_train_20, y_prob_mlp)

        auc_score_mlp
```

```
Out[ ]: 0.9254578239087662
```

Gradient Boosting

```
In [ ]: from sklearn.ensemble import GradientBoostingClassifier
        import warnings

        warnings.simplefilter("ignore")

        param_dist = {
            "loss": ["log_loss", "exponential"],
            "n_estimators": (100, 150, 300, 400, 500),
            "learning_rate": np.logspace(-5, 0, 20),
        }
        gb_rc = GradientBoostingClassifier()

        random_search = RandomizedSearchCV(
            estimator=gb_rc, param_distributions=param_dist, n_iter=5, cv=3
        )

        random_search.fit(X_train_80, y_train_80)

        print("Best hyperparameters:", random_search.best_params_)
```

```
Best hyperparameters: {'n_estimators': 500, 'loss': 'log_loss', 'learning_rate': 0.026366508987303583}
```

```
In [ ]: gb = GradientBoostingClassifier(
            n_estimators=500, learning_rate=0.0263, loss="log_loss", random_state=0
        ).fit(X_train_80, y_train_80)
```

```
In [ ]: y_prob_gb = gb.predict_proba(X_train_20)[:, 1]

        y_pred_gb = gb.predict(X_train_20)
```

```
fpr_gb, tpr_gb, th_gb = roc_curve(y_train_20, y_prob_gb)

auc_score_gb = roc_auc_score(y_train_20, y_prob_gb)
print("AUC_score validation GB with hyperparameter tuning", auc_score_gb)
```

```
AUC_score validation GB with hyperparameter tuning 0.9310892852053994
```

In [ ]:

```
print("Run time on train: 3m 22s")
print("Run time validation: 04s")

run_time_train["GB"] = 112.9
run_time_validation["GB"] = 0.4
```

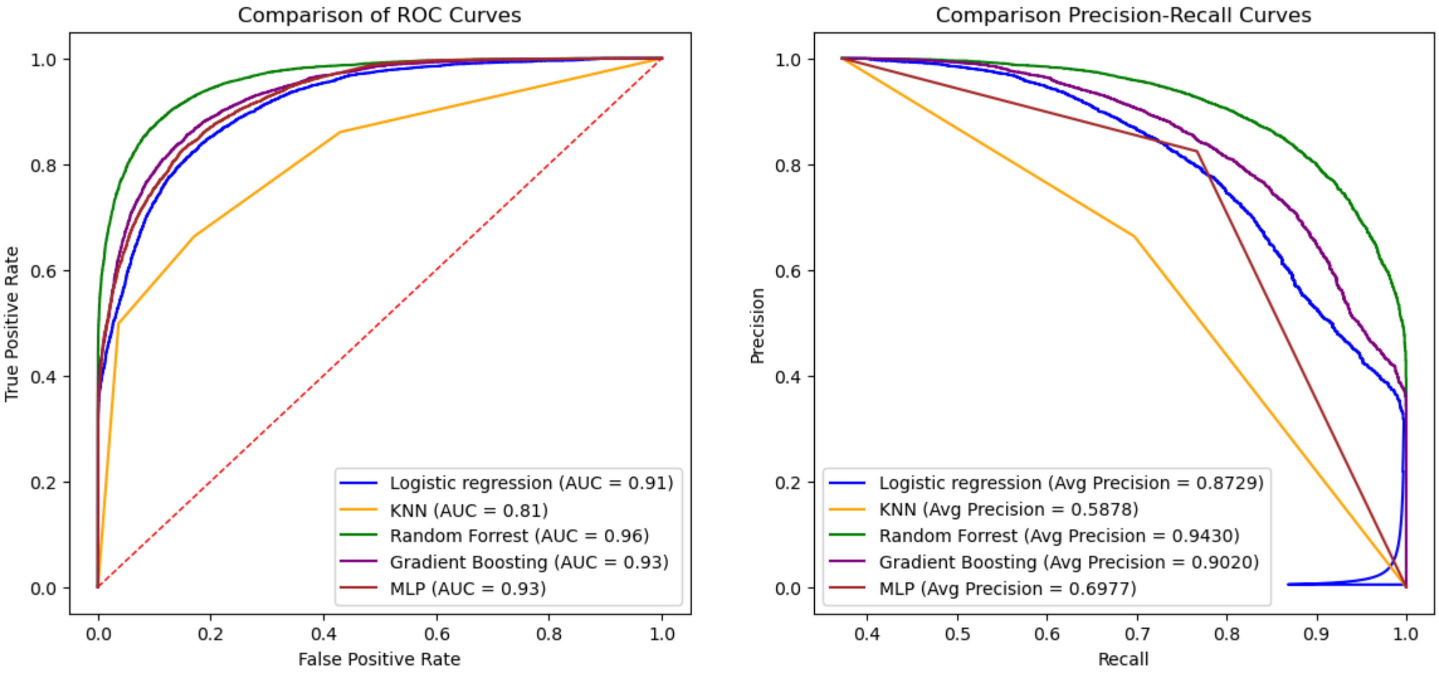
```
Run time on train: 3m 22s
Run time validation: 04s
```

In [ ]:

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score

y_hat_list = [y_prob_lgr, y_pred_ne, y_prob_rf_fs, y_prob_gb, y_pred_mlp]
model_list = [
    "Logistic regression",
    "KNN",
    "Random Forrest",
    "Gradient Boosting",
    "MLP",
]
colors_list = ["blue", "orange", "green", "purple", "brown"]
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# ROC plot
ax1.plot(
    fpr_lgr,
    tpr_lgr,
    color="blue",
    label="Logistic regression (AUC = %0.2f)" % auc_score_lgr,
)
ax1.plot(fpr_ne, tpr_ne, color="orange", label="KNN (AUC = %0.2f)" % auc_score_ne)
ax1.plot(
    fpr_rf_fs,
    tpr_rf_fs,
    color="green",
    label="Random Forrest (AUC = %0.2f)" % auc_score_rf_fs,
)
ax1.plot(
    fpr_gb,
    tpr_gb,
    color="purple",
    label="Gradient Boosting (AUC = %0.2f)" % auc_score_gb,
)
ax1.plot(fpr_mlp, tpr_mlp, color="brown", label="MLP (AUC = %0.2f)" % auc_score_mlp)
ax1.plot([0, 1], [0, 1], color="red", lw=1, linestyle="--")
ax1.set_xlabel("False Positive Rate")
ax1.set_ylabel("True Positive Rate")
ax1.set_title("Comparison of ROC Curves")
ax1.legend(loc="lower right")

# Precision plot

for i, y_hat in enumerate(y_hat_list):
    precision_lgr, recall_lgr, _ = precision_recall_curve(y_train_20, y_hat)
    avg_precision = average_precision_score(y_train_20, y_hat)
    ax2.plot(
        precision_lgr,
        recall_lgr,
        label=f"{model_list[i]} (Avg Precision = {avg_precision:.4f})",
        color=colors_list[i],
    )

ax2.set_xlabel("Recall")
ax2.set_ylabel("Precision")
```

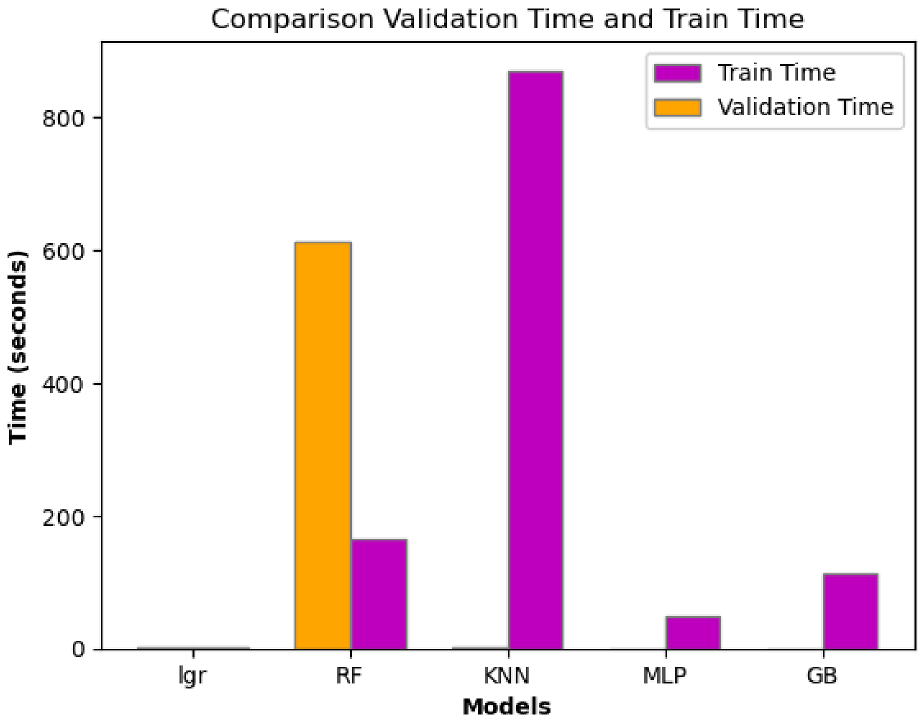
```
ax2.set_title("Comparison Precision-Recall Curves")
ax2.legend(loc="lower left")
```

Out[ ]: `<matplotlib.legend.Legend at 0x1778c4690>`



In [ ]:

```
import matplotlib.pyplot as plt

models = list(run_time_validation.keys())
validation_times = [run_time_validation[model] for model in models]
train_times = [run_time_train[model] for model in models]
bar_width = 0.35

r1 = range(len(models))
r2 = [x + bar_width for x in r1]

plt.bar(
    r2, train_times, color="m", width=bar_width, edgecolor="grey", label="Train Time"
)
plt.bar(
    r1,
    validation_times,
    color="orange",
    width=bar_width,
    edgecolor="grey",
    label="Validation Time",
)

plt.xlabel("Models", fontweight="bold")
plt.ylabel("Time (seconds)", fontweight="bold")
plt.xticks([r + bar_width / 2 for r in range(len(models))], models)

plt.legend()
plt.title("Comparison Validation Time and Train Time")
plt.show()
```



> Out of all the models chosen, the model that performed best was the Random Forrest based on both the AUC score (.96) and the Average Precision (.94), however Random Forrest was computationally intensive compared to the other models (except KNN).

d)

In [ ]:

```
X_train_ohe = X_train_ohe.copy()
y_train = y_train.copy()
X_test_ohe = X_test_ohe.copy()
# Logistic Regression

lgr2 = LogisticRegression(
    random_state=0,
    solver="liblinear",
    penalty="l1",
    C=78.475,
    max_iter=2500,
).fit(X_train_ohe, y_train)

y_prob_rfsf_final = lgr2.predict_proba(X_test_ohe)[:, 1]


# Random Forest

rfo = RandomForestClassifier(n_estimators=2000)

sfm2 = SelectFromModel(rfo, threshold=0.000100)
sfm2.fit(X_train_ohe, y_train)

X_train_fs_f = sfm2.transform(X_train_ohe)

rf3 = RandomForestClassifier(n_estimators=2000)
rf3.fit(X_train_fs_f, y_train)
X_test_fs = sfm2.transform(X_test_ohe)
y_prob_rfsf_final = rf3.predict_proba(X_test_fs)[:, 1]
create_submission(y_prob_rfsf_final, "data_out/rf_0530.csv")


# KNN

neigh2 = KNeighborsClassifier(n_neighbors=3)
neigh2.fit(X_train_ohe, y_train)
```

```
y_prob_ne_final = neigh2.predict_proba(X_test_ohe)[:, 1]

# GBoost
gb2 = GradientBoostingClassifier(
    n_estimators=500, learning_rate=0.0263, loss="log_loss", random_state=0
).fit(X_train_ohe, y_train)

y_prob_gb_final = gb2.predict_proba(X_test_ohe)[:, 1]

# MLP


mlp2 = MLPClassifier(
    random_state=1,
    learning_rate_init=1.7380420498195203e-05,
    batch_size=250,
    alpha=0.091853,
    solver="sgd",
    tol=1e-5,
    early_stopping=False,
    activation="relu",
    n_iter_no_change=1000,
    hidden_layer_sizes=(3, 3),
    max_iter=500,
).fit(X_train_ohe, y_train)

y_mlp_final = mlp2.predict_proba(X_test_ohe)[:, 1]
```

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=0.12)

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

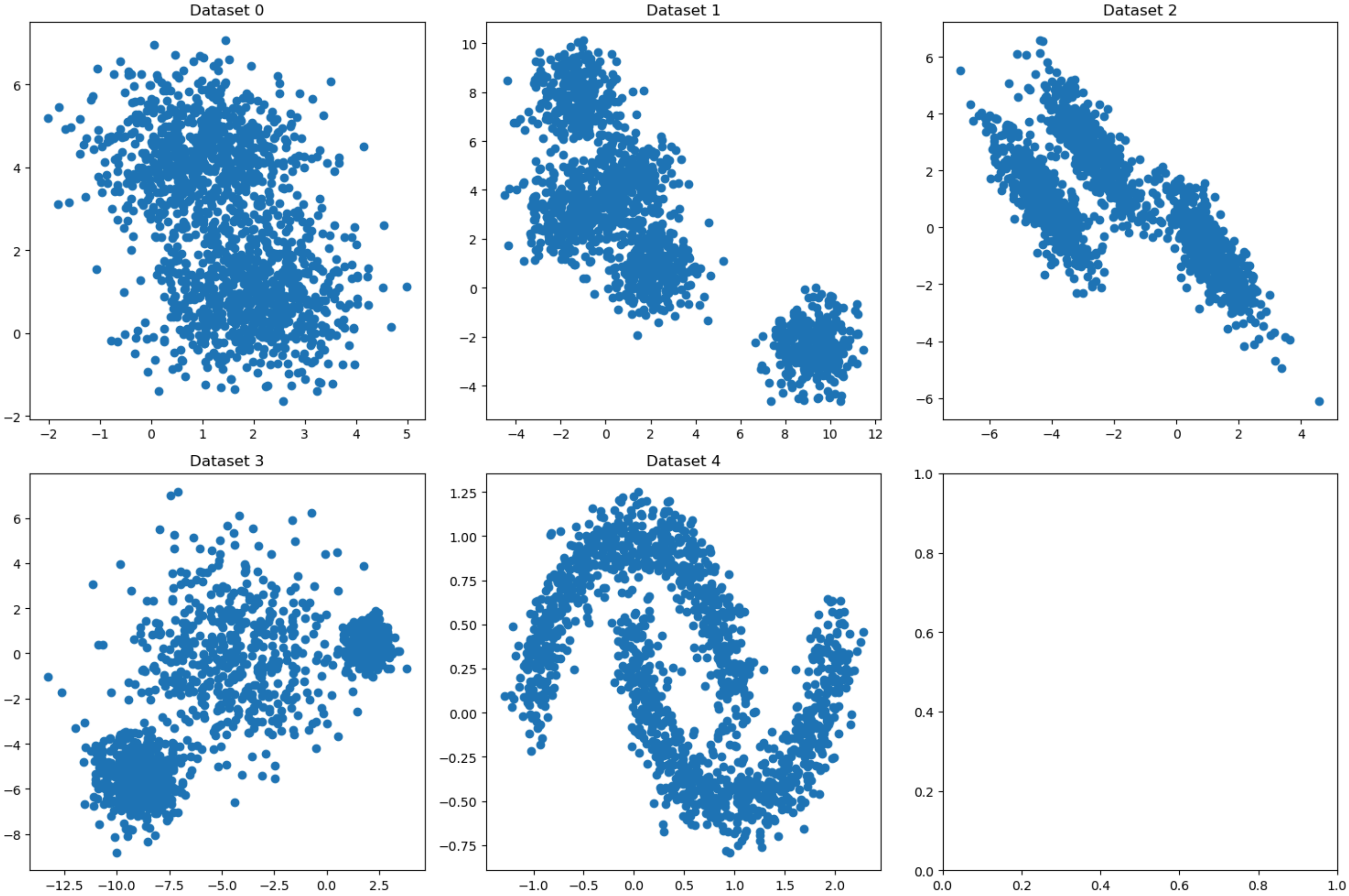
In [ ]:

```
################################
# 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
    clusster_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

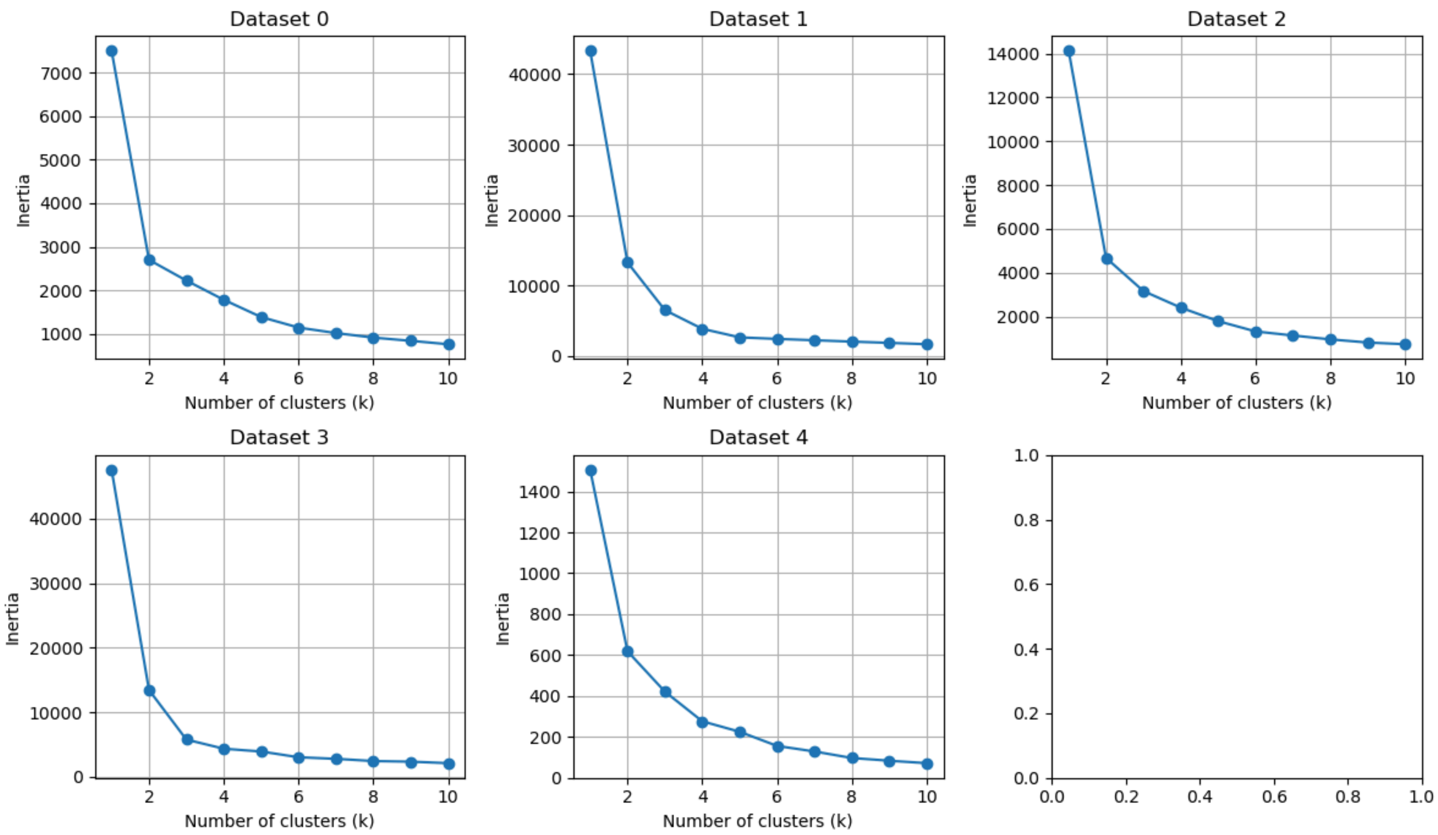
a) K- means

In [ ]:
```
fig, axs = plt.subplots(2, 3, figsize=(15, 10))
for i, ax in enumerate(axs.flat):
    if i < len(X):
        x = [p[0] for p in X[i]]
        y = [p[1] for p in X[i]]
        ax.scatter(x, y)
        ax.set_title(f"Dataset {i}")
plt.tight_layout()
plt.show()
```



In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans


fig, axs = plt.subplots(2, 3, figsize=(12, 7))

for i, ax in enumerate(axs.flat):
    if i < len(X):
        inertias = []
        for k in range(1, 11):
            kmeans = KMeans(n_clusters=k, random_state=0)
            kmeans.fit(X[i])
            inertias.append(kmeans.inertia_)
        # Plot the elbow curve
        ax.plot(range(1, 11), inertias, marker="o")
        ax.set_title(f"Dataset {i}")
        ax.set_xlabel("Number of clusters (k)")
```

```python
        ax.set_ylabel("Inertia")
        ax.grid(True)
plt.tight_layout()
plt.show()
```

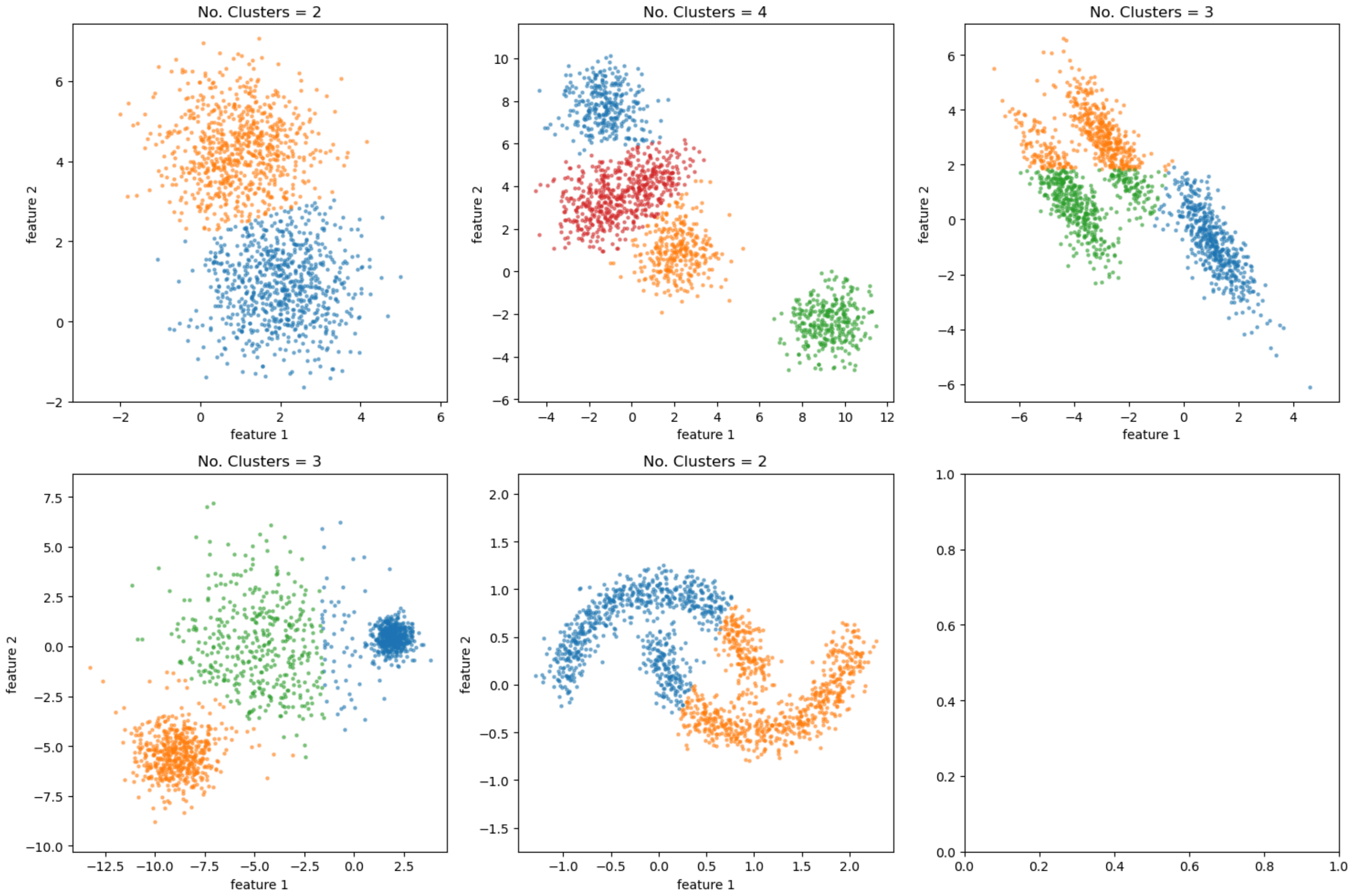


Where is the elbow within the sum-of-squeres:

| Dataset |Elbow Point| Why? | -------- | --------- | --- | | X0 | 2 | At this point the curve gradually starts to flatten towards the x axis| | X1 | 4 | At this point the curve starts to flatten towards the x axis| | X2 | 3 | At this point the curve gradually starts to flatten towards the x axis | | X3 | 3 | At this point the curve starts to flatten towards the x axis | | X4 | 2 | At this the curve gradually starts to flatten towards the x axis |

In [ ]:

```python
list_clusters = [2, 4, 3, 3, 2]

fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i, ax in enumerate(axs.flat):
    if i < len(X):
        kmeans = KMeans(n_clusters=list_clusters[i], random_state=0)
        kmeans.fit(X[i])
        cluster_assignments = kmeans.fit_predict(X[i])
        plot_cluster(ax, X[i], cluster_assignments)

plt.tight_layout()
plt.show()
```



b) DBSCAN

In [ ]:
```python
from sklearn.cluster import DBSCAN
import numpy as np

list_clusters = [2, 4, 3, 3, 2]
eps_min_list = []
eps = np.arange(0.1, 1.1, 0.1)
min_s = np.arange(2, 3001, 1)
for e in eps:
    for m in min_s:
        eps_min_list.append((e, m))
```

In [ ]:
```python
# function to find the combination of samples and eps for the number of clusters


def optimal_combination(db, combinations, k_cluster_n):
    new_tups_list = []
    for t in combinations:
        dbsl = DBSCAN(eps=t[0], min_samples=t[1]).fit(db)
        cluster_temp = dbsl.labels_
        n_clusters = len(np.unique(cluster_temp)) - 1
        num_no_label = np.sum(cluster_temp == -1)
        if n_clusters == k_cluster_n:
            new_tups = (t[0], t[1], num_no_label)
            new_tups_list.append(new_tups)
    return new_tups_list
```

In [ ]:
```python
test = optimal_combination(X[4], eps_min_list, list_clusters[4])
```

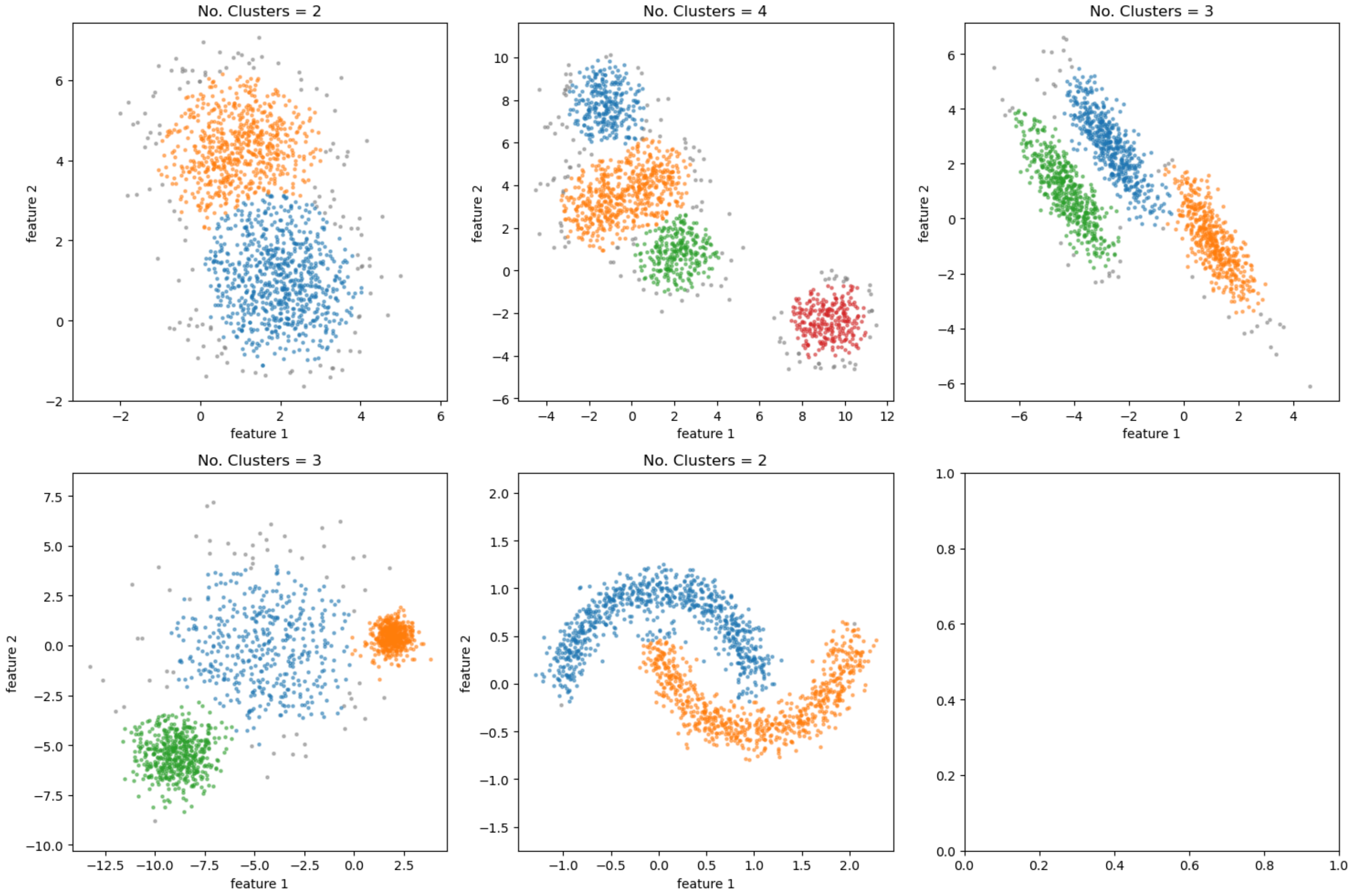
In [ ]:
```python
sorted(test, key=lambda x: x[2])[:5]
```

Out[ ]:

```
[(0.2, 33, 2),
 (0.30000000000000004, 93, 2),
 (0.2, 34, 3),
 (0.2, 35, 3),
 (0.2, 36, 3)]
```

In [ ]:

```
list_clusters = [2, 4, 3, 3, 2]
e_m_optimal = [(0.9, 146), (1, 81), (0.5, 16), (0.9, 10), (0.3, 93)]

fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i, ax in enumerate(axs.flat):
    if i < len(X):
        e_m = e_m_optimal[i]
        dbs = DBSCAN(eps=e_m[0], min_samples=e_m[1]).fit(X[i])
        cluster_assignments = dbs.fit_predict(X[i])
        plot_cluster(ax, X[i], cluster_assignments)

plt.tight_layout()
plt.show()
```

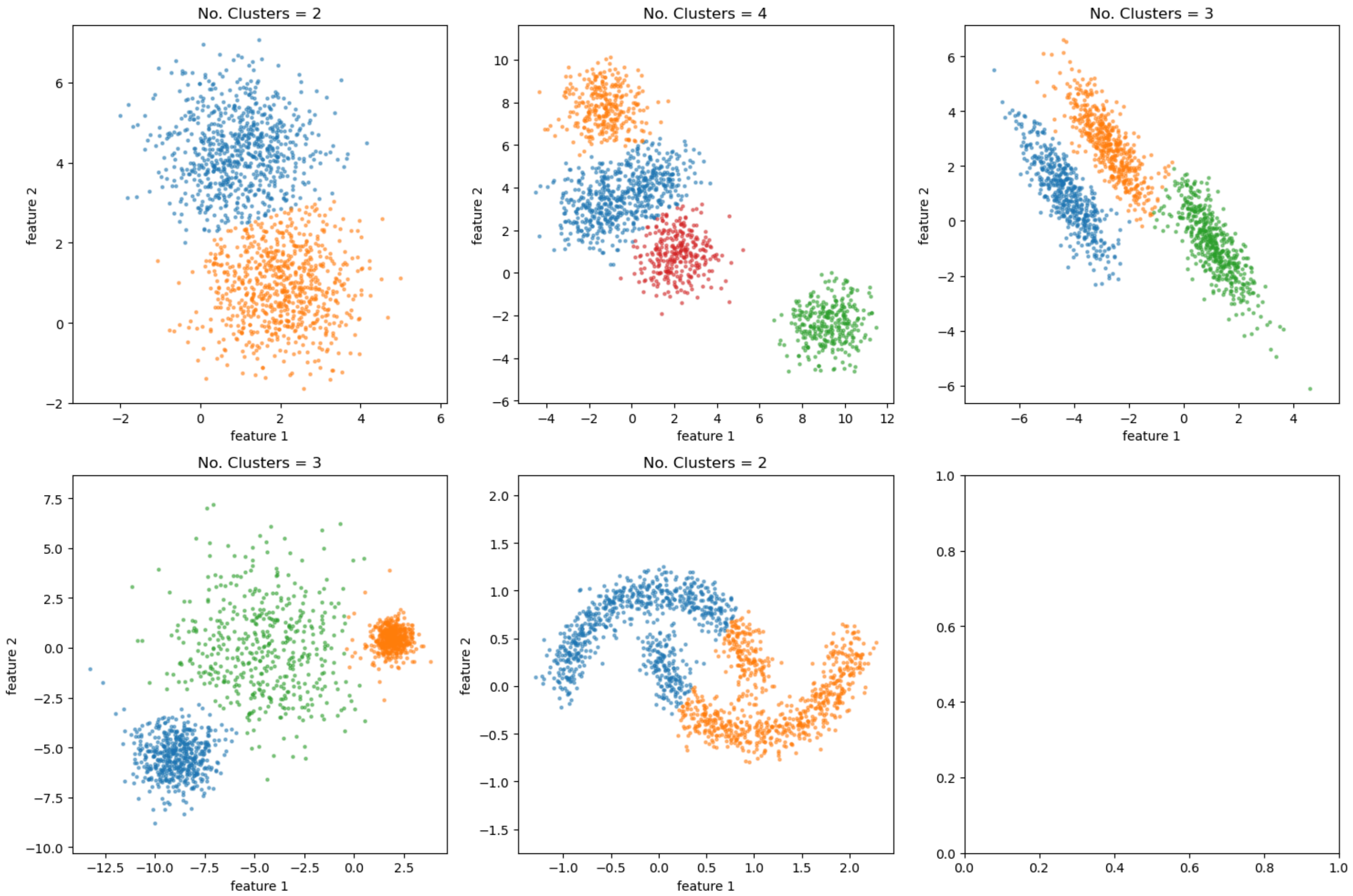


(c) Spectral Clustring

In [ ]:

```
from sklearn.cluster import SpectralClustering

list_clusters = [2, 4, 3, 3, 2]

fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i, ax in enumerate(axs.flat):
    if i < len(X):
        spc = SpectralClustering(
            n_clusters=list_clusters[i], assign_labels="discretize", random_state=0
        ).fit(X[i])
        cluster_assignments = spc.fit_predict(X[i])
        plot_cluster(ax, X[i], cluster_assignments)
```

```
plt.tight_layout()
plt.show()
```



(d) Comment on the strengths and weaknesses of each approach

> DBSCAN worked best for data sets X2 anad X4 which had arbitrary shaped clusters, though it had some amount of non asigned points for the regular shaped datasets. K means did not perform well on the arbitary shaped data and Spectral clustering performed better in this type of data than K-means but not as well as DBSCAN.

> DBSCAN requierd more effort to determine the right epsilon and minimun number of points for the clusters that were needed. Also, K-means required a method for determining the adecuate number of clusters, but it way less computationaly intensive.

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). Consider whether any preprocessing may need to be applied (do the data need to be normalized?). 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)

The data needs to be normalized as the ranges vary throughout the dataset:

In [ ]:

```
import pandas as pd

df = pd.DataFrame(X_digits)
df.describe()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	
count	1797.0	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.
mean	0.0	0.303840	5.204786	11.835838	11.848080	5.781859	1.362270	0.129661	0.005565	1.
std	0.0	0.907192	4.754826	4.248842	4.287388	5.666418	3.325775	1.037383	0.094222	3
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.0	0.000000	1.000000	10.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.
50%	0.0	0.000000	4.000000	13.000000	13.000000	4.000000	0.000000	0.000000	0.000000	0.
75%	0.0	0.000000	9.000000	15.000000	15.000000	11.000000	0.000000	0.000000	0.000000	3.
max	0.0	8.000000	16.000000	16.000000	16.000000	16.000000	16.000000	15.000000	2.000000	16.

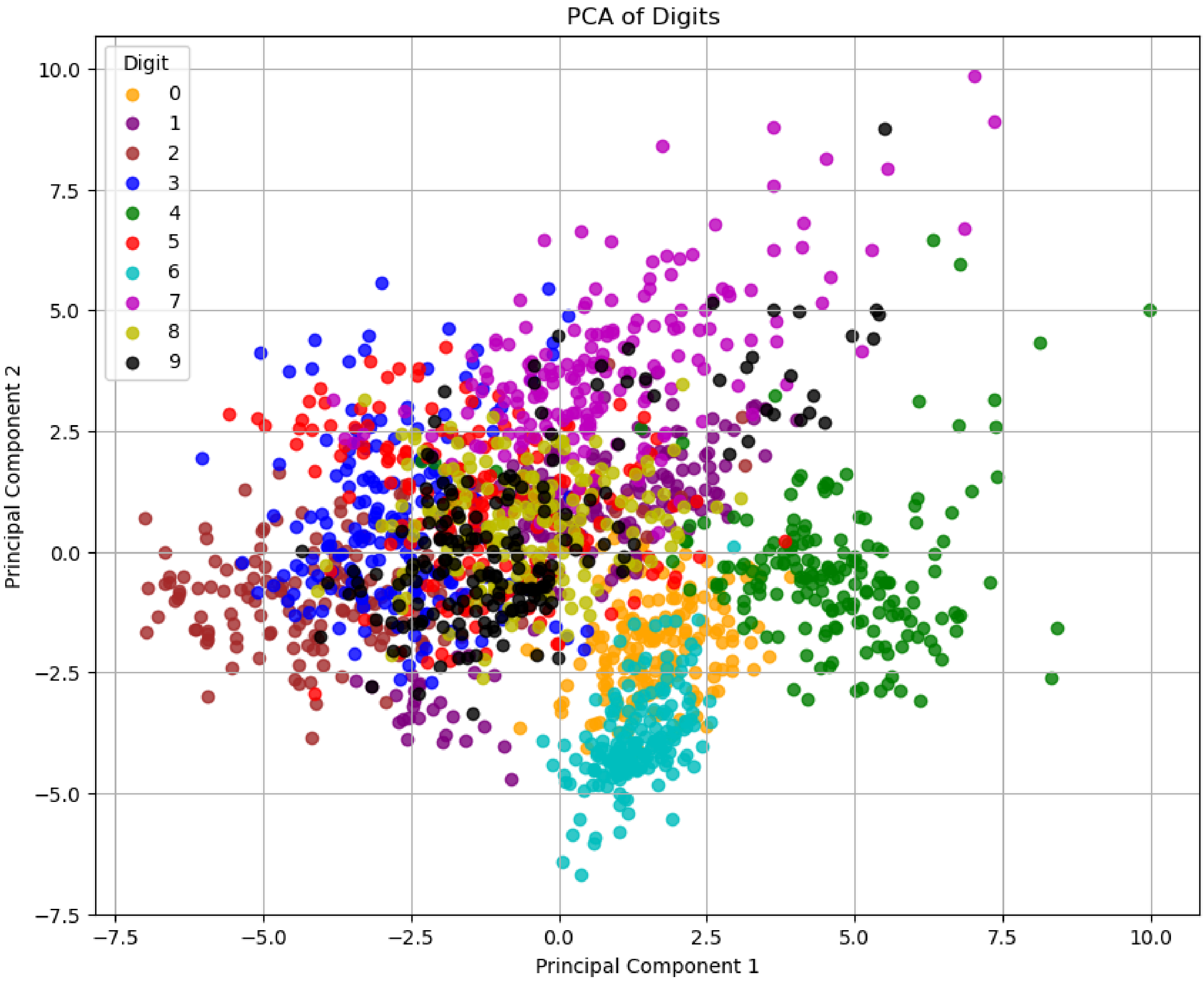
8 rows × 64 columns

In [ ]:

```
from sklearn.decomposition import PCA
```

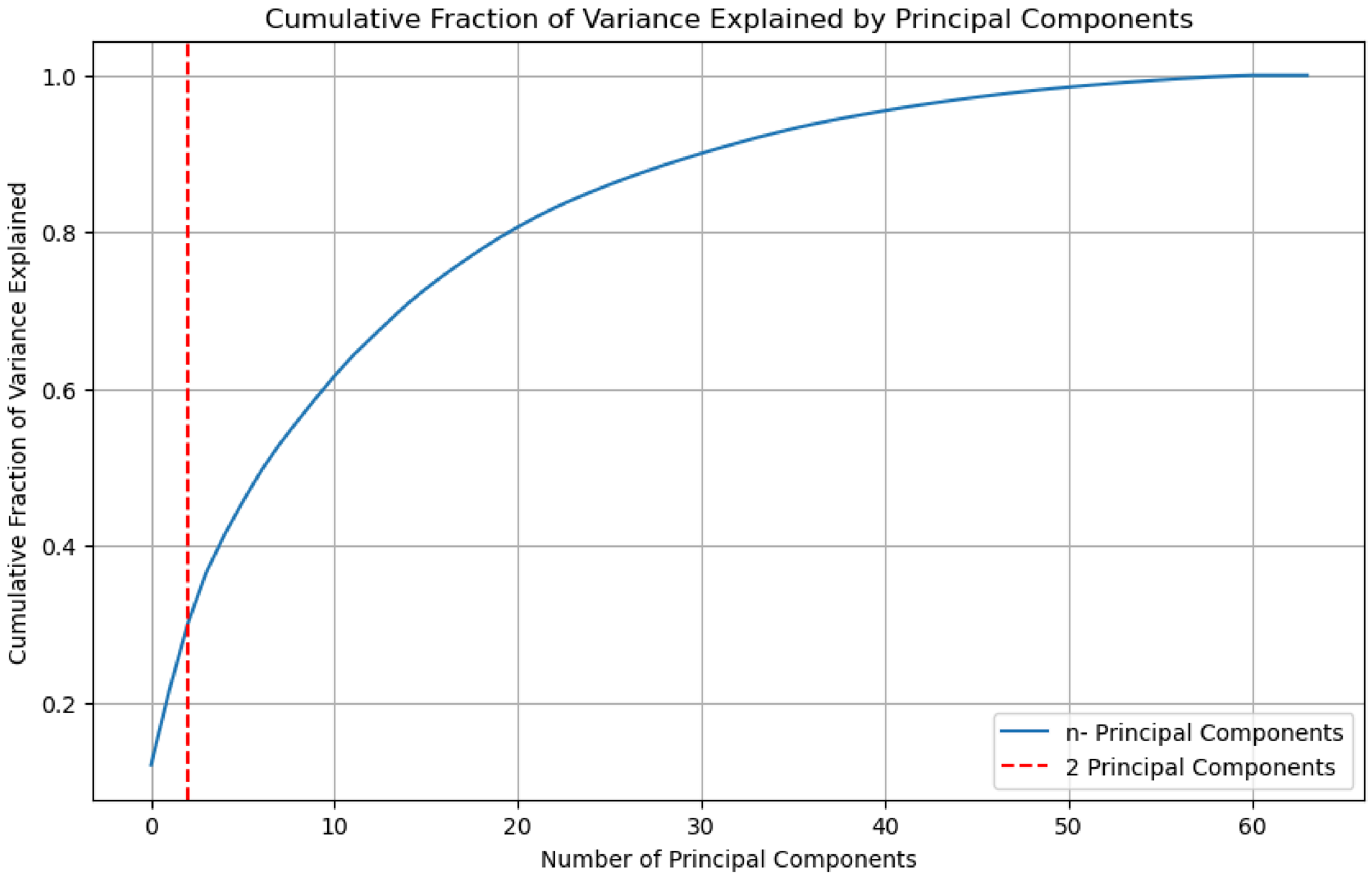
In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# normalize
scaler = StandardScaler()
X_digits_n = scaler.fit_transform(X_digits)

# PCA
pca = PCA(n_components=2, random_state=42).fit(X_digits_n)

X_pca = pca.transform(X_digits_n)


plt.figure(figsize=(10, 8))
colors = [
    "orange",
    "purple",
    "brown",
    "b",
    "g",
    "r",
    "c",
    "m",
    "y",
    "k",
]
for i in range(10):
    plt.scatter(
        X_pca[y_digits == i, 0],
        X_pca[y_digits == i, 1],
        label=str(i),
        color=colors[i],
        alpha=0.8,
    )
plt.title("PCA of Digits ")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Digit")
plt.grid(True)
plt.show()
```



In [ ]:

```
pca_c = PCA()
pca_c.fit(X_digits_n)

c_v_ratio = np.cumsum(pca_c.explained_variance_ratio_)


plt.figure(figsize=(10, 6))
plt.plot(c_v_ratio, label="n- Principal Components")
plt.axvline(x=2, color="r", linestyle="--", label="2 Principal Components")
plt.title("Cumulative Fraction of Variance Explained by Principal Components")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Fraction of Variance Explained")
plt.grid(True)
plt.legend()
plt.show()

n_features = X_digits.shape[1]
unexplained_variance = 1 - np.sum(pca.explained_variance_ratio_[:2])
print(
    "Fraction of variance unexplained by the first two principal components:",
    unexplained_variance,
)
```



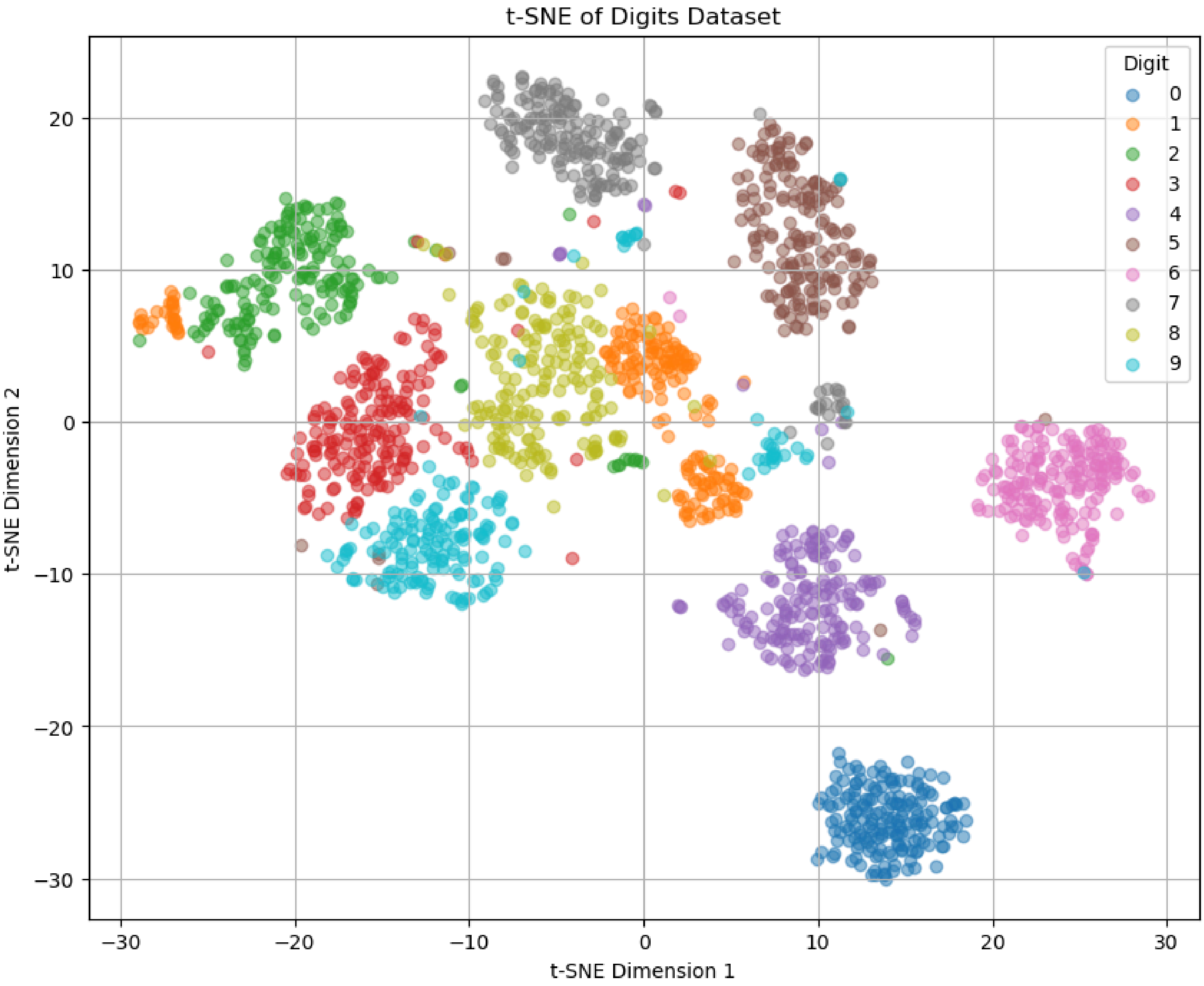
```
Fraction of variance unexplained by the first two principal components: 0.7840502950637295
```

The fraction of variance unexplained by the first two principal components is of 78.4%, remaining high and as we can see from the overlapping clusters in the PCA plot. This will imact negatively in the clustering of the data.

(c)

In [ ]:

```
tsne = TSNE(
    n_components=2, perplexity=100, random_state=42, n_iter_without_progress=1000
)
X_tsne = tsne.fit_transform(X_digits_n)

# Plot the data points with labels in 2D space
plt.figure(figsize=(10, 8))
for i in range(10):
    plt.scatter(
        X_tsne[y_digits == i, 0], X_tsne[y_digits == i, 1], label=str(i), alpha=0.5
    )

plt.title("t-SNE of Digits Dataset")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.legend(title="Digit")
plt.grid(True)
plt.show()
```



(d)

- Tsne seems to cluster the data better as it has less overlapping clusters than the PSA. TSNE is better suited for high dimensional data, as is digits dataset, than PCA.
- TSNE's optimization process is integrated with the embedding process. In the optimization, TSNE aims to minimize the mismatch between local similarities in the low dimensional space from the high dimensional one and that process yields the transformed embeddings. This means that the model cannot transform new data based on the model and a new model has to be trained each time for new data.