Assignment 5 - Kaggle Competition and Unsupervised Learning

Alisa Tian

Netid: WT83

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.

In [ ]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_curve, auc, precision_recall_curve,average_precision_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import StackingClassifier
import seaborn as sns
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,StackingClassifier
from sklearn.pipeline import Pipeline
from pandas import DataFrame
from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
from sklearn.cluster import SpectralClustering
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import time
import warnings
warnings.filterwarnings('ignore')
```

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( "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

a. Explore the data

In [ ]:
```
# Looking at the shape of data
print(X_train_ohe.shape)
print(X_test_ohe.shape)
```

```
(95512, 940)
(23878, 940)
```

In [ ]:
```
X_train_ohe.describe()
```

Out[ ]:

	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nig
count	95512.000000	95512.000000	95512.000000	95512.000000	95512.000000	95512.000
mean	103.849768	2016.157205	27.152902	15.823038	0.928491	2.503
std	106.722804	0.707470	13.611204	8.786777	0.999940	1.918
min	0.000000	2015.000000	1.000000	1.000000	0.000000	0.000
25%	18.000000	2016.000000	16.000000	8.000000	0.000000	1.000
50%	69.000000	2016.000000	27.000000	16.000000	1.000000	2.000
75%	160.000000	2017.000000	38.000000	23.000000	2.000000	3.000
max	709.000000	2017.000000	53.000000	31.000000	19.000000	50.000

8 rows × 940 columns

In [ ]:
```
X_test_ohe.describe()
```

Out[ ]:

	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nig
count	23878.000000	23878.000000	23878.000000	23878.000000	23878.000000	23878.000
mean	104.658012	2016.153949	27.214256	15.699054	0.924030	2.488
std	107.422248	0.707509	13.581023	8.756480	0.993302	1.868
min	0.000000	2015.000000	1.000000	1.000000	0.000000	0.000
25%	18.000000	2016.000000	16.000000	8.000000	0.000000	1.000
50%	70.000000	2016.000000	28.000000	16.000000	1.000000	2.000
75%	161.000000	2017.000000	38.000000	23.000000	2.000000	3.000
max	737.000000	2017.000000	53.000000	31.000000	18.000000	42.000

8 rows × 940 columns

In [ ]:
```
merge= pd.merge(X_train_original, y_train, left_index=True, right_index=True)
```

Babies

In [ ]:
```
merge.groupby('babies')['is_canceled'].mean().reset_index()
```

Out[ ]:

	babies	is_canceled
0	0	0.371905
1	1	0.188652
2	2	0.153846
3	9	0.000000

It seems that the number of babies is associated with the cancel rate.

children

In [ ]:
```
merge.groupby('children')['is_canceled'].mean().reset_index()
```

Out[ ]:

	children	is_canceled
0	0.0	0.370968
1	1.0	0.327028
2	2.0	0.417008
3	3.0	0.222222

distribution_channel

In [ ]:
```
merge.groupby('distribution_channel')['is_canceled'].mean().reset_index()
```

Out[ ]:

	distribution_channel	is_canceled
0	Corporate	0.218585
1	Direct	0.173026
2	GDS	0.180124
3	TA/TO	0.410754
4	Undefined	0.666667

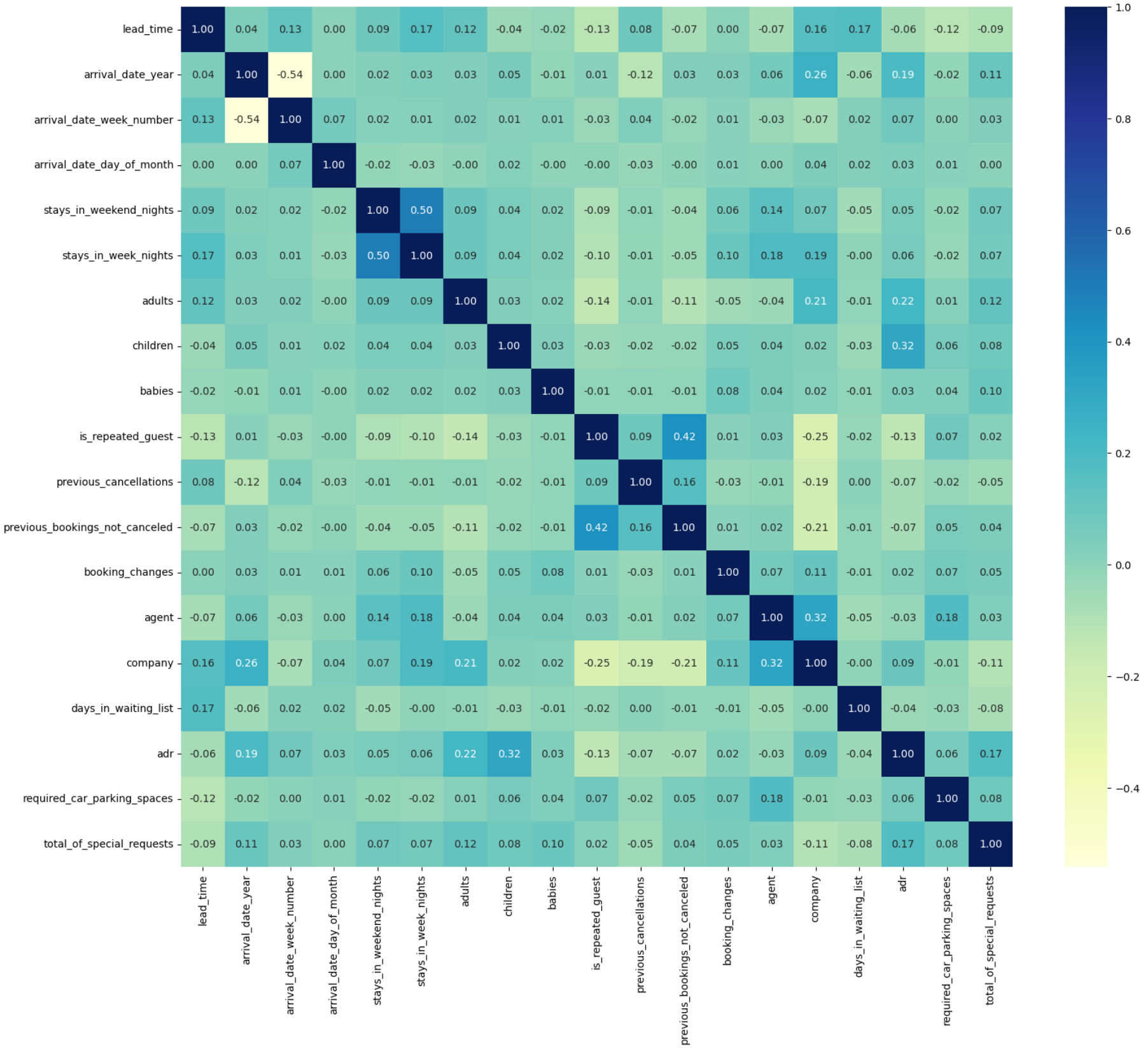
Adults

In [ ]:
```
merge.groupby('adults')['is_canceled'].mean().reset_index()
```

Out[ ]:

	adults	is_canceled
0	0	0.275000
1	1	0.287065
2	2	0.393972
3	3	0.348082
4	4	0.265306
5	5	1.000000
6	6	1.000000
7	10	1.000000
8	20	1.000000
9	26	1.000000
10	27	1.000000
11	40	1.000000
12	50	1.000000
13	55	1.000000

In [ ]:
```
plt.subplots(figsize=(18,15))
sns.heatmap(X_train_original.corr(), annot=True, fmt=".2f",cmap="YlGnBu")
plt.show()
```



I want to explore the correlation between different variables and created this heatmap. We can see that there seems to be no strong correlation between different variables in the dataset.

b. Preprocess the data

Deal with the missing values

In [ ]:
```
X_train_original.isnull().sum()
```

Out[ ]:
```
hotel                             0
lead_time                         0
arrival_date_year                 0
arrival_date_month                0
arrival_date_week_number          0
arrival_date_day_of_month         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          2
babies                            0
meal                              0
country                         395
market_segment                    0
distribution_channel              0
is_repeated_guest                 0
previous_cancellations            0
previous_bookings_not_canceled    0
reserved_room_type                0
assigned_room_type                0
booking_changes                   0
deposit_type                      0
agent                         13081
company                       90059
days_in_waiting_list              0
customer_type                     0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
dtype: int64
```

I am interested in the number of missing values here, and find that sume variables have many missing values. When dealing with the missing value for children, I decided to fill it with the mean value.

In [ ]:
```
X_test_ohe["children"].fillna(round(X_test_ohe["children"].mean()), inplace=True)
X_train_ohe["children"].fillna(round(X_train_ohe["children"].mean()), inplace=True)
```

Normalize the data

In [ ]:
```
X_train_ohe = StandardScaler().fit_transform(X_train_ohe)
X_test_ohe = StandardScaler().transform(X_test_ohe)
```

The data have been normalized here.

Split the data into 2 sets

Then, I split the data into training and validation set.

In [ ]:
```
X_train_new, X_val_new, y_train_new, y_val_new = train_test_split(
    X_train_ohe, y_train, test_size = 0.2, random_state=42
)
```

c. Train

Linear Regression

I started with the simplest one - linear regression, and wanted to see the performance. It seems that the performance of the first trail is not that bad.

In [ ]:
```
pipe = Pipeline([
    ('classifier', LogisticRegression()),
])

parameters = {
    'classifier__solver': ['liblinear', 'saga'],
}

grid = GridSearchCV(pipe, parameters, cv=2)
grid.fit(X_train_new, y_train_new)

train_score = grid.score(X_train_new, y_train_new)
val_score = grid.score(X_val_new, y_val_new)
print('Logistic Regression Model (Grid Search) Training set score:', train_score)
print('Logistic Regression Model (Grid Search) Validation set score:', val_score)

best_parameter = grid.best_params_
print('Best parameters:', best_parameter)
```

```
best_lr = grid.best_estimator_
print('Optimum Logistic Regression Model:', best_lr)
```

```
Logistic Regression Model (Grid Search) Training set score: 0.8226910442487142
Logistic Regression Model (Grid Search) Validation set score: 0.8257341778778202
Best parameters: {'classifier__solver': 'liblinear'}
Optimum Logistic Regression Model: Pipeline(steps=[('classifier', LogisticRegression(solver='liblinear'))])
```

In [ ]:
```
start_time_lr_train = time.time()
best_lr.fit(X_train_new, y_train_new)
end_time_lr_train = time.time()
lr_train= end_time_lr_train - start_time_lr_train
print('Time taken to fit the best logist regression:', lr_train)
```

```
Time taken to fit the best logist regression: 1.052922010421753
```

In [ ]:
```
start_time_lr_vali = time.time()
best_lr.predict(X_val_new)
end_time_lr_vali = time.time()
lr_vali=end_time_lr_vali - start_time_lr_vali
print('Time taken to validate the best logist regression:', lr_vali)
```

```
Time taken to validate the best logist regression: 0.035804033279418945
```

In [ ]:
```
fpr_lr, tpr_lr, threshold_lr = metrics.roc_curve(y_val_new, best_lr.predict_proba(X_val_new)[:,1])
```

In [ ]:
```
auc_lr=metrics.auc(fpr_lr, tpr_lr)
```

In [ ]:
```
precision_lr, recall_lr, thresholds_lr = precision_recall_curve(y_val_new, best_lr.predict_proba(X_val_new)[:,1])
ap_lr = average_precision_score(y_val_new, best_lr.predict_proba(X_val_new)[:,1])
```

Fit on the whole traning set and create submission

In [ ]:
```
submit_lr = best_lr.fit(X_train_ohe, y_train)
create_submission(submit_lr.predict_proba(X_test_ohe)[:, 1], "Logistic.csv")
```

Best KNN

I then chose the KNN method and tuned the classifier__n_neighbors and classifier__weights.

In [ ]:
```
pipe = Pipeline([
    ('classifier', KNeighborsClassifier()),
])

parameters = {
    'classifier__n_neighbors': [3, 5],
    'classifier__weights': ['uniform', 'distance'],
}

grid = GridSearchCV(pipe, parameters, cv=2)
grid.fit(X_train_new, y_train_new)

train_score = grid.score(X_train_new, y_train_new)
val_score = grid.score(X_val_new, y_val_new)
print('KNN Model (Grid Search) Training set score:', train_score)
print('KNN Model (Grid Search) Validation set score:', val_score)

best_parameter = grid.best_params_
print('Best parameters:', best_parameter)

best_knn = grid.best_estimator_
print('Optimum KNN Model:', best_knn)
```

```
KNN Model (Grid Search) Training set score: 0.9962308105066157
KNN Model (Grid Search) Validation set score: 0.7916557608752552
Best parameters: {'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}
Optimum KNN Model: Pipeline(steps=[('classifier', KNeighborsClassifier(weights='distance'))])
```

In [ ]:
```
start_time_knn_train = time.time()
best_knn.fit(X_train_new, y_train_new)
end_time_knn_train = time.time()
knn_train=end_time_knn_train-start_time_knn_train
print('Time taken to fit the best KNN:', knn_train)

start_time_knn_vali = time.time()
best_knn.predict(X_val_new)
end_time_knn_vali = time.time()
knn_vali=end_time_knn_vali-start_time_knn_vali
```

```
Time taken to fit the best KNN: 0.22998404502868652
Time taken to validate the best KNN: 8.643601894378662
```

In [ ]:
```
print('Time taken to validate the best KNN:', knn_vali)
```

```
Time taken to validate the best KNN: 8.643601894378662
```

In [ ]:
```
fpr_knn, tpr_knn, threshold_knn = metrics.roc_curve(y_val_new, best_knn.predict_proba(X_val_new)[:,1])

auc_knn = metrics.auc(fpr_knn, tpr_knn)
```

In [ ]:
```
precision_knn, recall_knn, thresholds_knn = precision_recall_curve(
    y_val_new, best_knn.predict_proba(X_val_new)[:, 1]
)
ap_knn = average_precision_score(y_val_new, best_knn.predict_proba(X_val_new)[:, 1])
```

Fit on the whole traning set and create submission

In [ ]:
```
submit_knn = best_knn.fit(X_train_ohe, y_train)
create_submission(submit_knn.predict_proba(X_test_ohe)[:, 1], "knn.csv")
```

Best RF

Before tuning the random forest, I also tuned the neural network. I tuned too many hyperparameters and it takes several hours. So, I decide to tune fewer parameters here. I start by using different estimators for the random forest.

In [ ]:
```
pipe = Pipeline([
    ('classifier', RandomForestClassifier()),
])
pipe.fit(X_train_new, y_train_new)
train_score = pipe.score(X_train_new, y_train_new)
val_score = pipe.score(X_val_new, y_val_new)
print('Random Forest Model (Pipeline) Training set score:', train_score)
print('Random Forest Model (Pipeline) Validation set score:', val_score)

parameters = {
    'classifier__n_estimators': [50, 100, 150, 200, 500, 1000],
}

grid = GridSearchCV(pipe, parameters, cv=2)
grid.fit(X_train_new, y_train_new)
train_score = grid.score(X_train_new, y_train_new)
val_score = grid.score(X_val_new, y_val_new)
print('Random Forest Model (Grid Search) Training set score:', train_score)
print('Random Forest Model (Grid Search) Validation set score:', val_score)


best_parameter = grid.best_params_
print('Best parameters:', best_parameter)

best_rf0 = grid.best_estimator_
print('Optimum Random Forest Model:', best_rf0)
```

```
Random Forest Model (Pipeline) Training set score: 0.9964271224593961
Random Forest Model (Pipeline) Validation set score: 0.8895984923833953
Random Forest Model (Grid Search) Training set score: 0.9964271224593961
Random Forest Model (Grid Search) Validation set score: 0.8903313615662461
Best parameters: {'classifier__n_estimators': 500}
Optimum Random Forest Model: Pipeline(steps=[('classifier', RandomForestClassifier(n_estimators=500))])
Result Dataframe columns: Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_classifier__n_estimators', 'params', 'split0_test_score',
       'split1_test_score', 'mean_test_score', 'std_test_score',
       'rank_test_score'],
      dtype='object')
```

In [ ]:
```
start_time_rf_train = time.time()
best_rf0.fit(X_train_new, y_train_new)
end_time_rf_train = time.time()
rf_train=end_time_rf_train-start_time_rf_train
print('Time taken to fit the best random forest:',rf_train)
```

```
Time taken to fit the best random forest: 92.781320810318
```

In [ ]:
```
start_time_rf_vali = time.time()
best_rf0.predict(X_val_new)
end_time_rf_vali = time.time()
rf_vali=end_time_rf_vali-start_time_rf_vali
print('Time taken to validate the best random forest:', rf_vali)
```

```
Time taken to validate the best random forest: 2.0317602157592773
```

In [ ]:
```
fpr_rf, tpr_rf, threshold_rf = metrics.roc_curve(y_val_new, best_rf0.predict_proba(X_val_new)[:,1])
```

In [ ]:
```
auc_rf=metrics.auc(fpr_rf, tpr_rf)
```

In [ ]:
```
precision_rf, recall_rf, thresholds_rf = precision_recall_curve(y_val_new, best_rf0.predict_proba(X_val_new)[:,1])
ap_rf = average_precision_score(y_val_new, best_rf0.predict_proba(X_val_new)[:,1])
```

Fit on the whole traning set and create submission

In [ ]:
```
submit_rf = best_rf0.fit(X_train_ohe, y_train)
create_submission(submit_rf.predict_proba(X_test_ohe)[:, 1], "randomforest.csv")
```

Best RF tune tunes the min_samples_split

The performs of my previous random forest model is pretty good. So, I decided to continue using random forest and tune the minimum sample split numbers.

In [ ]:
```
pipe = Pipeline([
    ('classifier', RandomForestClassifier()),
])
pipe.fit(X_train_new, y_train_new)
train_score = pipe.score(X_train_new, y_train_new)
val_score = pipe.score(X_val_new, y_val_new)
print('Random Forest Model (Pipeline) Training set score:', train_score)
print('Random Forest Model (Pipeline) Validation set score:', val_score)

parameters = {
    'classifier__min_samples_split': [2, 5, 10],
}

grid = GridSearchCV(pipe, parameters, cv=2)
grid.fit(X_train_new, y_train_new)
train_score = grid.score(X_train_new, y_train_new)
val_score = grid.score(X_val_new, y_val_new)
print('Random Forest Model (Grid Search) Training set score:', train_score)
print('Random Forest Model (Grid Search) Validation set score:', val_score)


best_parameter = grid.best_params_
print('Best parameters:', best_parameter)

bestrf2 = grid.best_estimator_
```

```
Random Forest Model (Pipeline) Training set score: 0.9964271224593961
Random Forest Model (Pipeline) Validation set score: 0.8904360571637963
Random Forest Model (Grid Search) Training set score: 0.9964271224593961
Random Forest Model (Grid Search) Validation set score: 0.8887609276029943
Best parameters: {'classifier__min_samples_split': 2}
```

In [ ]:
```
start_time_bestrf2_train = time.time()
bestrf2.fit(X_train_new, y_train_new)
end_time_bestrf2_train= time.time()
```

In [ ]:
```
rf2_train= end_time_bestrf2_train - start_time_bestrf2_train
```

In [ ]:
```
print('Time taken to fit the best random forest :',rf2_train)
```

```
Time taken to fit the best random forest : 19.280537128448486
```

In [ ]:
```
start_time_bestrf2_vali = time.time()
bestrf2.predict(X_val_new)
end_time_bestrf2_vali = time.time()
rf2_vali=end_time_bestrf2_vali - start_time_bestrf2_vali
print('Time taken to validate the best random forest:', rf2_vali)
```

```
Time taken to validate the best random forest: 0.445544958114624
```

In [ ]:
```
fpr_rf2, tpr_rf2, threshold_rf2 = metrics.roc_curve(y_val_new, bestrf2.predict_proba(X_val_new)[:,1])
```

In [ ]:
```
auc_rf2=metrics.auc(fpr_rf2, tpr_rf2)
```

In [ ]:
```
precision_rf2, recall_rf2, thresholds_rf2 = precision_recall_curve(y_val_new, bestrf2.predict_proba(X_val_new)[:,1]
ap_rf2 = average_precision_score(y_val_new, bestrf2.predict_proba(X_val_new)[:,1])
```

Fit on the whole traning set and create submission

In [ ]:
```
submit_rf2 = bestrf2.fit(X_train_ohe, y_train)
create_submission(submit_rf2.predict_proba(X_test_ohe)[:, 1], "randomforest2.csv")
```

Best RF+ LR: stacking

I am very interested in the performance of the ensemble method, and tried the stacking method. Here, I stacked my best logistic regression and random forest method.

In [ ]:
```
rf = RandomForestClassifier(n_estimators=500)
nn = LogisticRegression(solver='liblinear')

pipe = [
```

```
    ('rf', rf),
    ('nn', nn),
]

stack = StackingClassifier(
    estimators=pipe,
    final_estimator=GradientBoostingClassifier()
)
```

In [ ]:
```
start_time_stack_train = time.time()
stack.fit(X_train_new,y_train_new)
end_time_stack_train  = time.time()
```

In [ ]:
```
stack_train=end_time_stack_train - start_time_stack_train
```

In [ ]:
```
print('Time taken to fit the best RF and LR model (stack):', stack_train)
```

```
Time taken to fit the best RF and LR model (stack): 531.6984670162201
```

In [ ]:
```
start_time_stack_val = time.time()
stack.predict(X_val_new)
end_time_stack_val  = time.time()
stack_vali=end_time_stack_val - start_time_stack_val
```

In [ ]:
```
print('Time taken to validate the best RF and LR model (stack):',stack_vali)
```

```
Time taken to validate the best RF and LR model (stack): 2.2214701175689697
```

In [ ]:
```
precision_stack, recall_stack, thresholds_stack = precision_recall_curve(
    y_val_new, stack.predict_proba(X_val_new)[:, 1]
)
ap_stack = average_precision_score(y_val_new, stack.predict_proba(X_val_new)[:, 1])
```

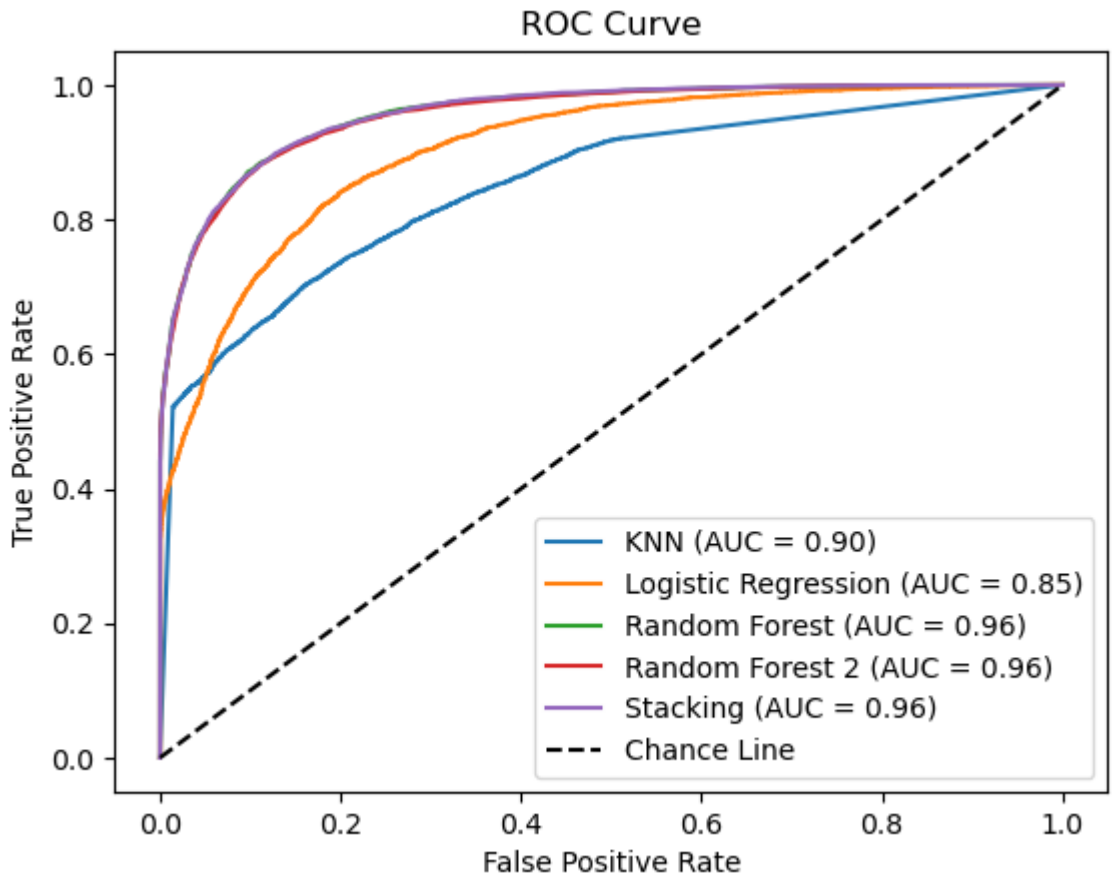
Fit on the whole traning set and create submission

In [ ]:
```
submit_stack = stack.fit(X_train_ohe, y_train)
create_submission(submit_stack.predict_proba(X_test_ohe)[:, 1], "stacking.csv")
```

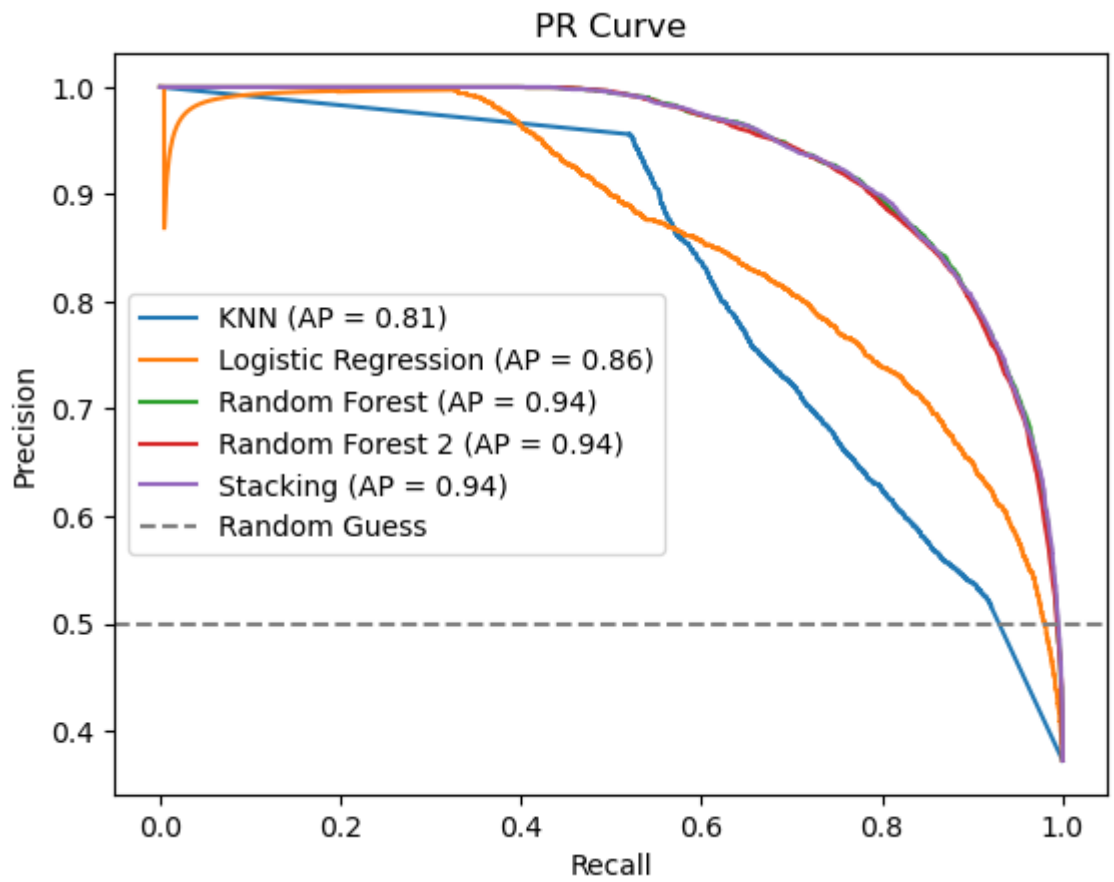
I also tuned the different hyper parameter for the neural network. However, it might be because I tuned too many parameters, it takes more than 3 hours and I did not include it here.

My best performance is from the Random Forest I tuned the number of estimators. It has a score of 0.95934. Surprisingly, the score of my stacking method is very low. This method also takes the most time to fit.

In [ ]:
```
plt.plot(fpr_knn, tpr_knn, label='KNN (AUC = %0.2f)' % auc_lr)
plt.plot(fpr_lr, tpr_lr, label='Logistic Regression (AUC = %0.2f)' % auc_knn)
plt.plot(fpr_rf, tpr_rf, label='Random Forest (AUC = %0.2f)' % auc_rf)
plt.plot(fpr_rf2, tpr_rf2, label='Random Forest 2 (AUC = %0.2f)' % auc_rf2)
plt.plot(fpr_stack, tpr_stack, label='Stacking (AUC = %0.2f)' % auc_stack)

# random guess
plt.plot([0, 1], [0, 1], 'k--',label='Chance Line')


plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')


plt.legend()
plt.show()
```



In [ ]:
```
plt.plot(recall_knn, precision_knn, label='KNN (AP = %0.2f)' % ap_knn)
plt.plot(recall_lr, precision_lr, label='Logistic Regression (AP = %0.2f)' % ap_lr)
plt.plot(recall_rf, precision_rf, label='Random Forest (AP = %0.2f)' % ap_rf)
plt.plot(recall_rf2, precision_rf2, label='Random Forest 2 (AP = %0.2f)' % ap_rf2)
plt.plot(recall_stack, precision_stack, label='Stacking (AP = %0.2f)' % ap_stack)
plt.axhline(y=0.49850, color='gray', linestyle='--', label='Random Guess')


# Set the axis labels and title
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PR Curve')

# Add the legend and show the plot
plt.legend()
plt.show()
```

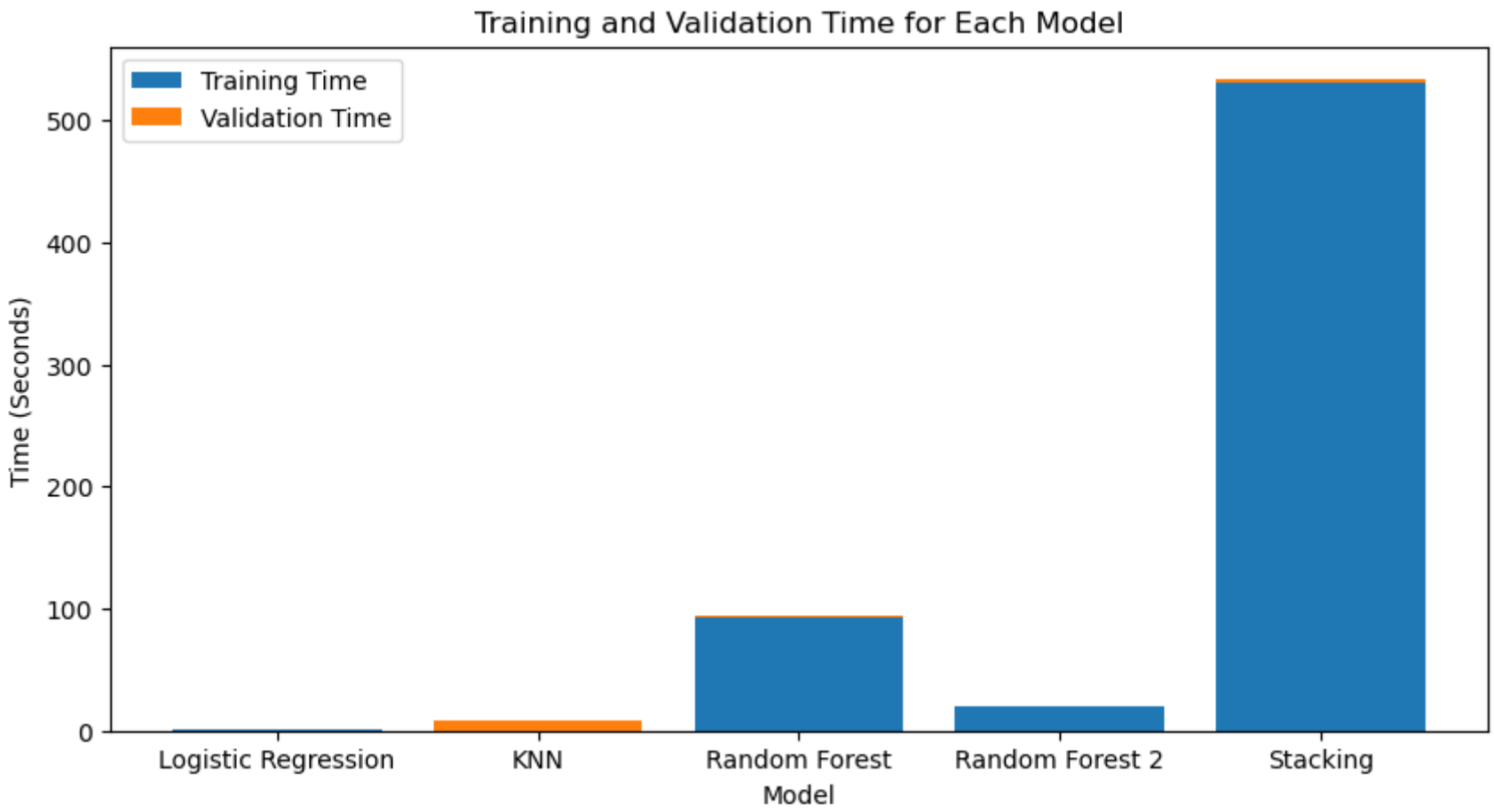


Time

The graph below show the training and validation time for different methods. The Ensemble model(stacking) takes a lot of time.

In [ ]:
```
training_times = [lr_train, knn_train, rf_train, rf2_train, stack_train]
validation_times = [lr_vali, knn_vali, rf_vali, rf2_vali, stack_vali]


model_labels = ['Logistic Regression', 'KNN', 'Random Forest', 'Random Forest 2', 'Stacking']


fig, ax = plt.subplots(figsize=(10, 5))
ax.bar(model_labels, training_times, label='Training Time')
ax.bar(model_labels, validation_times, label='Validation Time', bottom=training_times)
```

```
ax.set_xlabel('Model')
ax.set_ylabel('Time (Seconds)')
ax.set_title('Training and Validation Time for Each Model')
ax.legend()
plt.show()
```



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
```

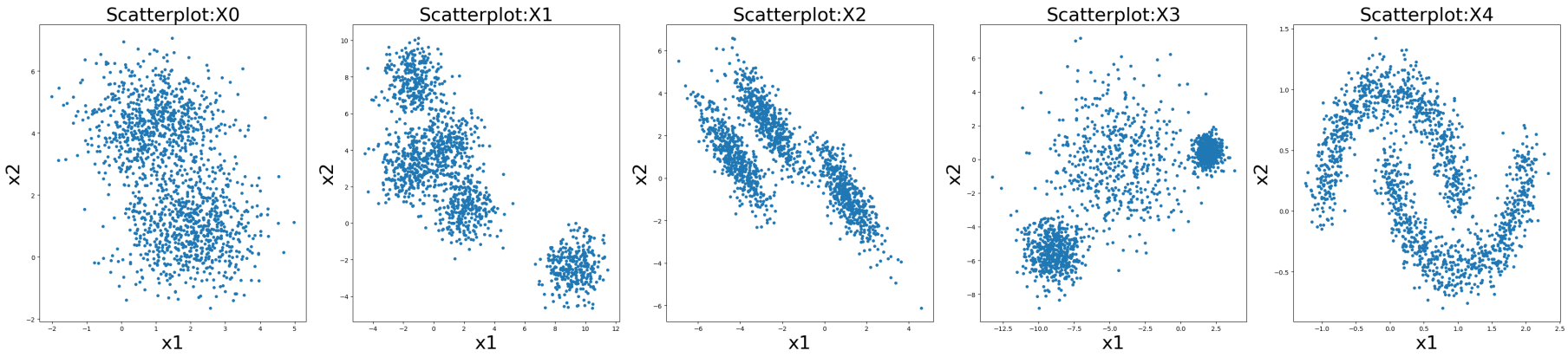
In [ ]:
```
################################
# Code to plot clusters
################################
def plot_cluster(ax, data, cluster_assignments):
    '''Plot two-dimensional data clusters

    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

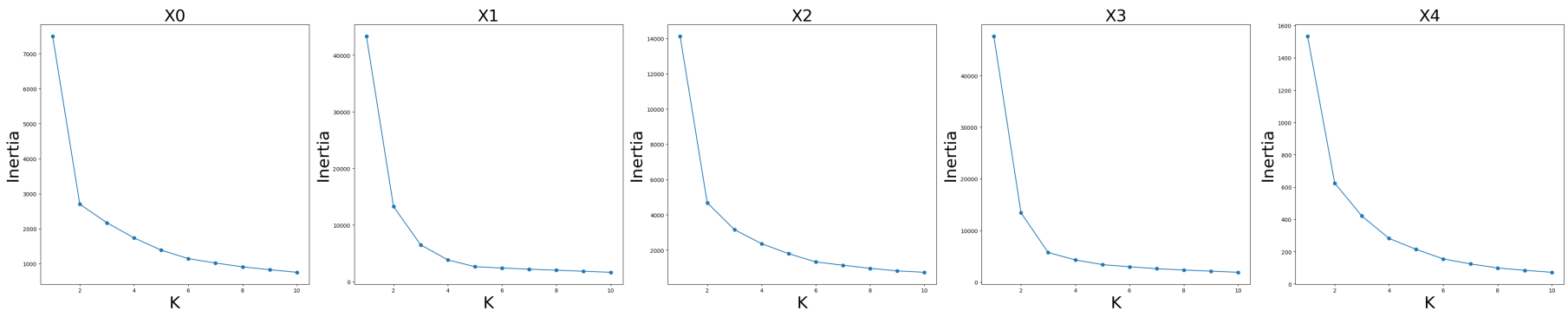
plot

In [ ]:
```
labels = ["Scatterplot:X0","Scatterplot:X1","Scatterplot:X2","Scatterplot:X3","Scatterplot:X4"  ]


count = 1
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(35, 8))

for i, lab, ax in zip(X, labels, axs.flatten()):
    ax.scatter(i[:, 0], i[:, 1], s=16)
    sns.color_palette("pastel")
    ax.set_title(lab,fontsize=30)
    ax.set_xlabel("x1",fontsize=30)
    ax.set_ylabel("x2",fontsize=30)
    if count == 6:
        ax.set_axis_off()
    count += 1

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



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

In [ ]:
```
labels = ["X0", "X1", "X2", "X3", "X4"]


fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(40, 8))


for i, lab, ax in zip(X, labels, axs.flatten()):
    inte = []
    for k in range(1, 11):
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(i)
        inte.append(kmeans.inertia_)
    ax.plot(range(1, 11), inte, marker='o')
    ax.set_title(lab,fontsize=30)
    ax.set_xlabel('K',fontsize=30)
    ax.set_ylabel('Inertia',fontsize=30)


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

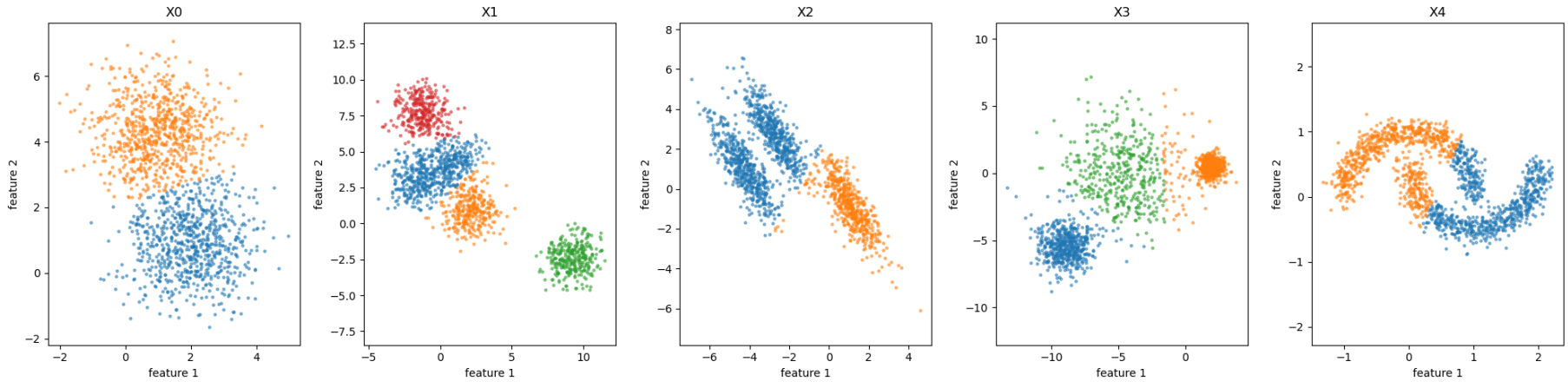


For each datasets, where is the elbow in the curve of within-cluster sum-of-squares and why?

The elbow is not always visible, and I think it is betwewen 2 and 3. The elbow occurs when the inertia's decreasing rate gradually slow down. The elbow is not always visible, and I think it is betwewen 2 and 3. The elbow occurs when the inertia's decreasing rate gradually slow down. So for X0 and X4, the elbow is around 2. For X1 and X3, the elbow is around 3. For X2, the elbow is around 4.

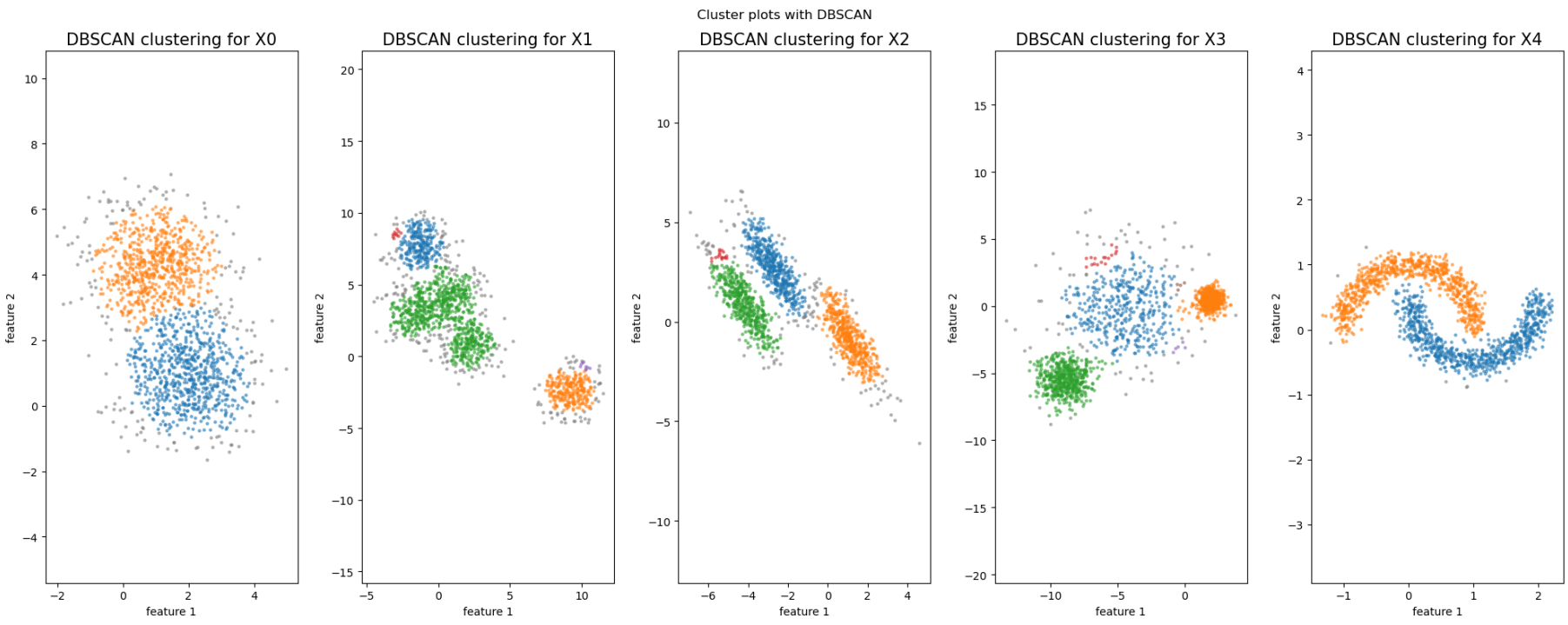
Plot your clustered data

In [ ]:
```
k=[2,4,2,3,2]


fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(20, 5))

for i, num in enumerate(axs):
    mean= KMeans(n_clusters=k[i]).fit(X[i]).labels_
    plot_cluster(num, X[i], mean)
    num.set_title(labels[i])
plt.tight_layout()
plt.show()
```



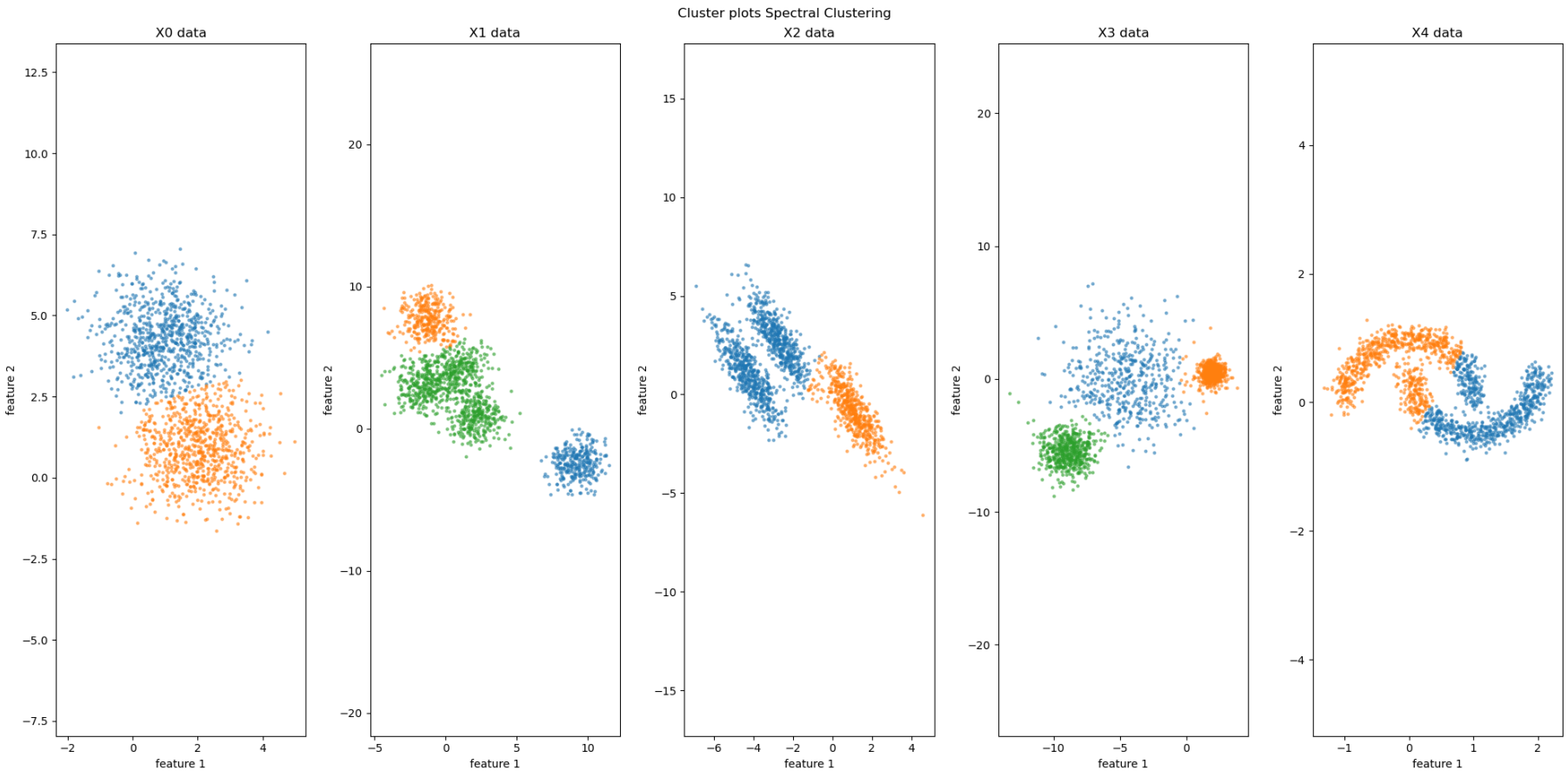
b. Apply DBSCAN

In [ ]:
```
fig, axs = plt.subplots(1, 5, figsize=(20, 8))


# set hyperparameters for as in kmeans and experiment with different parameter values
eps_values = [0.8, 0.4, 0.4, 0.6, 0.2]
min_samples_values = [120, 10, 15, 5, 30]

for i, ax in enumerate(axs):
    dbscan = DBSCAN(eps=eps_values[i], min_samples=min_samples_values[i]).fit(X[i])
    plot_cluster(ax, X[i], dbscan.labels_)
    ax.set_title(f'DBSCAN clustering for X{i}', fontsize=15)

plt.suptitle('Cluster plots with DBSCAN')
plt.tight_layout()
plt.show()
```



c. Apply Spectral Clustering

In [ ]:
```
fig, axs = plt.subplots(1, 5, figsize=(20, 10))

clusters = [2, 3, 2, 3, 2]

for i, ax in enumerate(axs):
    cluster = SpectralClustering(n_clusters=clusters[i], assign_labels="discretize", random_state=2018).fit(X[i]).l
    plot_cluster(ax, X[i], cluster)
    ax.set_title("X{} data".format(i))

plt.suptitle("Cluster plots Spectral Clustering")
plt.tight_layout()
plt.show()
```



c. Comment

When I used the values chosen from elbow pllots, I get the worst performance in Kmeans plots (the third and last plot). Generally, my DBSCAN has the best performance. I experimented with different parameter to get as close as I can to the same number of clusters of K-means. The third and last plots of spectral clustering do not perform well.

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.

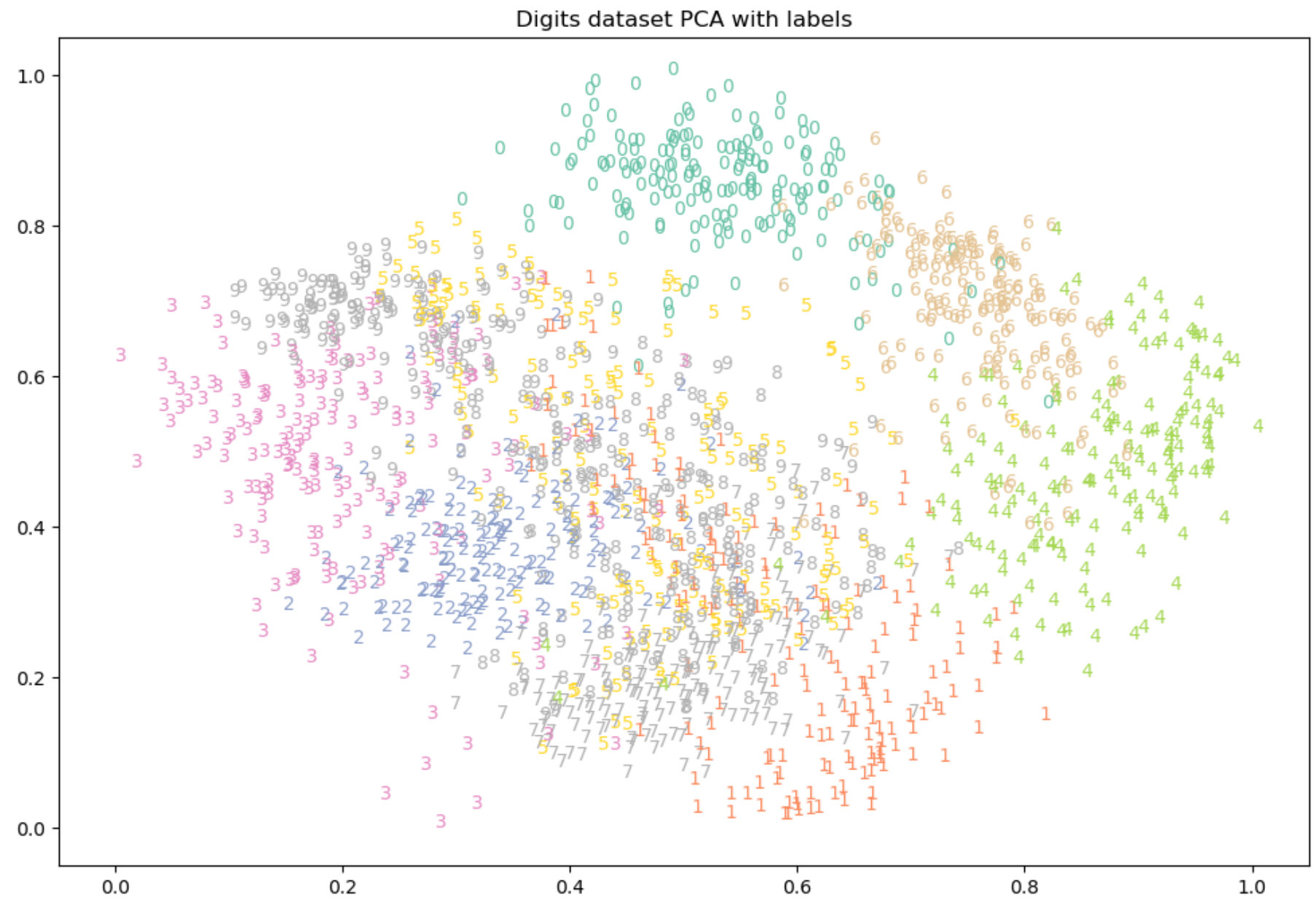
In [ ]:
```
################################
# Load the data
################################
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# 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

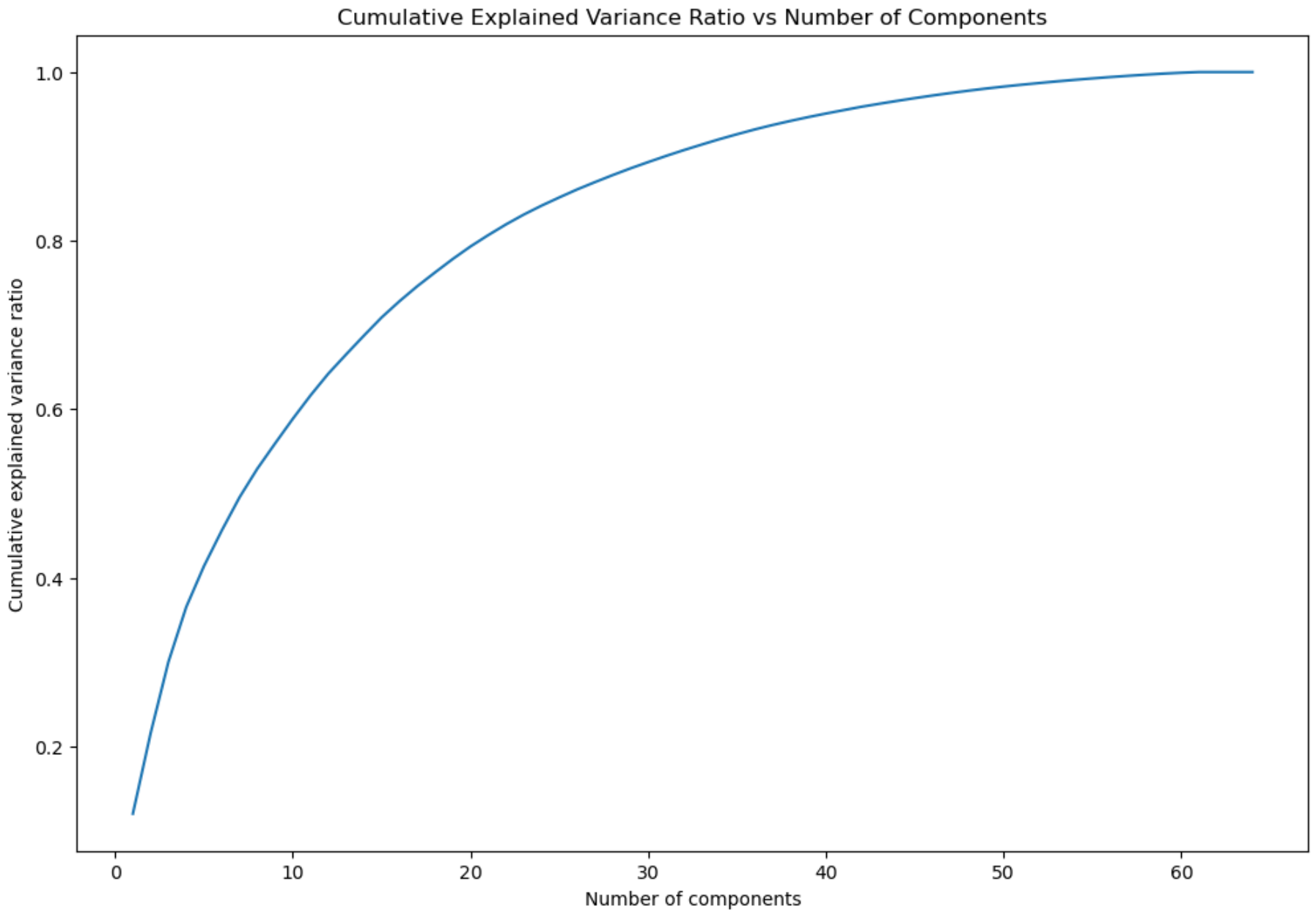
In [ ]:
```
# 2 dimensional PCA
pca = PCA(n_components=2)
digits_x = pca.fit_transform(X_digits)

plt.figure(figsize=(12, 8))

x_min, x_max = np.min(digits_x, 0), np.max(digits_x, 0)
X_norm = (digits_x - x_min) / (x_max - x_min)

for i in range(X_norm.shape[0]):
    plt.text(X_norm[i, 0], X_norm[i, 1], str(y_digits[i]),
             color=plt.cm.Set2(y_digits[i]), fontdict={'size': 10})

# set plot limits
plt.ylim([-0.05, 1.05])
plt.xlim(-0.05, 1.05)
```

```
plt.title('Digits dataset PCA with labels')
plt.show()
```



b create a plot

In [ ]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_digits)
pca = PCA(n_components=X_digits.shape[1])
pca.fit(X_scaled)
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

fig, axs = plt.subplots(1, 1, figsize = (12, 8))
axs.plot(range(1, X_digits.shape[1] + 1), cumulative_variance_ratio)
axs.set_xlabel('Number of components')
axs.set_ylabel('Cumulative explained variance ratio')
axs.set_title('Cumulative Explained Variance Ratio vs Number of Components')
plt.show()
```



In [ ]:
```
print(f"The first component explains {cumulative_variance_ratio[0] * 100:.2f}% of the variance in the data.")
print(f"The second component explain {cumulative_variance_ratio[1] * 100:.2f}% of the variance in the data.")
```

```
The first component explains 12.03% of the variance in the data.
The second component explain 21.59% of the variance in the data.
```

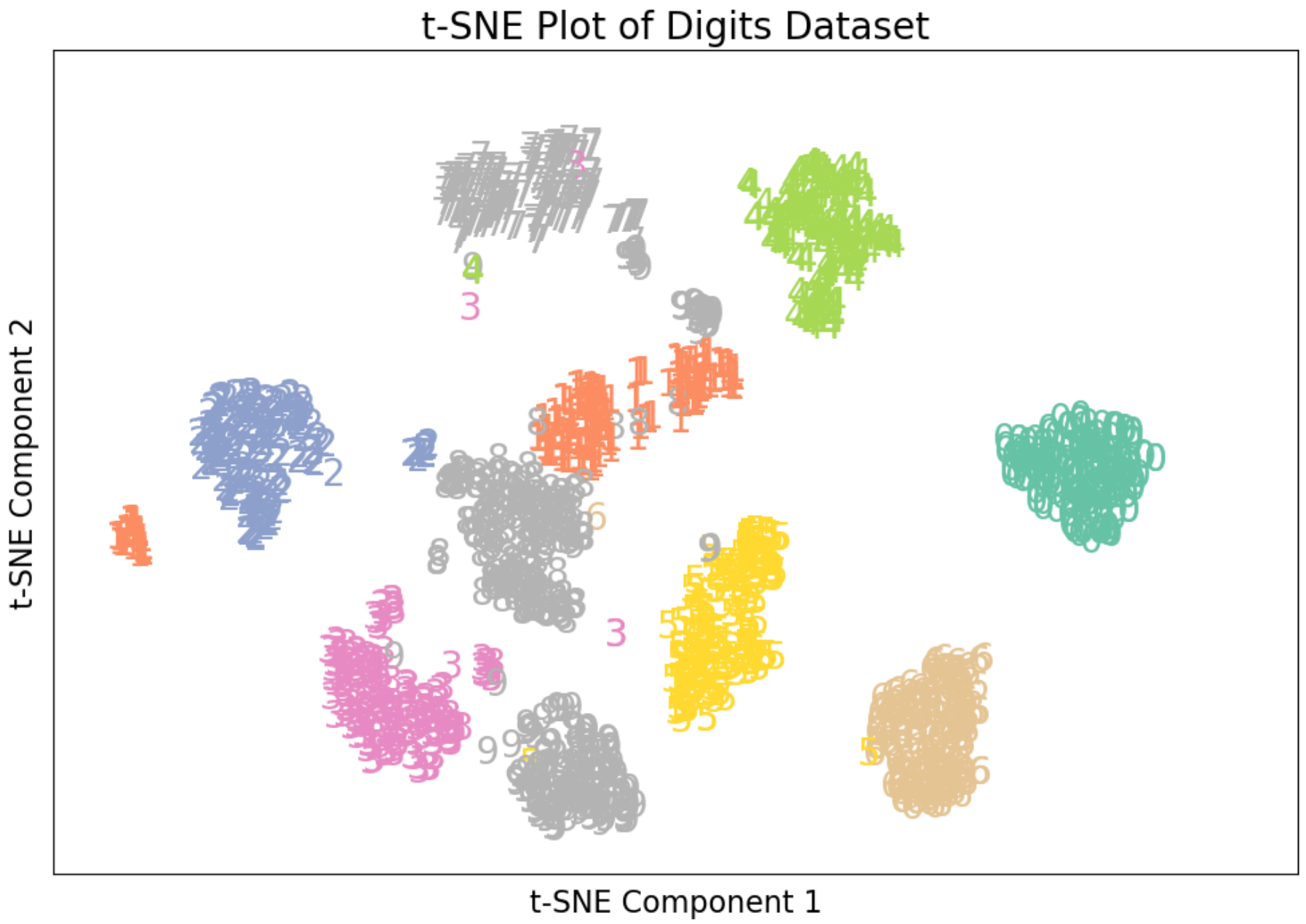
In [ ]:
```
print(f"The first component leaves {100 - cumulative_variance_ratio[0] * 100:.2f}% of the variance unexplained in t
print(f"The second component leaves {100 - cumulative_variance_ratio[1] * 100:.2f}% of the variance unexplained in
```

```
The first component leaves 87.97% of the variance unexplained in the data.
The second component leaves 78.41% of the variance unexplained in the data.
```

The difference of unexplained variance be caused because the principal components cluster the data well gradually.

C

In [ ]:
```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# perform t-SNE on the input data
X_tsne = TSNE(n_components=2, learning_rate="auto", init="random").fit_transform(X_digits)

# plot the t-SNE components
x_min, x_max = np.min(X_tsne, 0), np.max(X_tsne, 0)
X_norm = (X_tsne - x_min) / (x_max - x_min)

plt.figure(figsize=(12, 8))
for i in range(X_norm.shape[0]):
    plt.text(
        X_norm[i, 0], X_norm[i, 1], str(y_digits[i]), color=plt.cm.Set2(y_digits[i]), fontdict={"size": 20}
    )

plt.xticks([]),
plt.yticks([]),
plt.ylim([-0.05, 1.15])
plt.xlim(-0.05, 1.15)
plt.title('t-SNE Plot of Digits Dataset', fontsize=20)
plt.xlabel('t-SNE Component 1', fontsize=16)
plt.ylabel('t-SNE Component 2', fontsize=16)
plt.show()
```



d. Briefy compare the performance of these two techniques.

Based on the two plots I created, the TSNE outperforms than the PCA method. The clusters are more separate than PCA's plot. Different clusters are further. In addition, fewer data seems to have the wrong classification according to the plots.

TSNE does not have `fit` method because it is an unsupervised nonlinear dimensionality reduction method. So we cannot fit on the training set and validate on the validation set. It creates a probability distribution for high-dimensional objects and assigns high probabilities for similar ones. Then, it tries to find lower-dimensional representations that similar probabilities TSNE are mostly used for visualization rather than model predicting. It is also sensitive to different choices of hyperparameters.