# Assignment 2 - Supervised Machine Learning Fundamentals

## *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
import pandas as pd
from matplotlib import pyplot as plt
import math
from scipy.stats import mode
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_moons
import random
import time
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
```

## Learning Objectives:

By successfully completing this assignment you will be able to...

- Explain the bias-variance tradeoff of supervised machine learning and the impact of model flexibility on algorithm performance
- Perform supervised machine learning training and performance evaluation
- Implement a k-nearest neighbors machine learning algorithm from scratch in a style similar to that of popular machine learning tools like `scikit-learn`
- Describe how KNN classification works, the method's reliance on distance measurements, and the impact of higher dimensionality on computational speed
- Apply regression (linear regression) and classification (KNN) supervised learning techniques to data and evaluate the performance of those methods
- Construct simple feature transformations for improving model fit in linear models

- Fit a `scikit-learn` supervised learning technique to training data and make predictions using it

In [ ]:
```
# MAC USERS TAKE NOTE:
# For clearer plots in Jupyter notebooks on macs, run the following line of
%config InlineBackend.figure_format = 'retina'
```

---

# Conceptual Questions on Supervised Learning

## 1

**[4 points]** For each part below, indicate whether we would generally expect the performance of a flexible statistical learning method to be *better* or *worse* than an inflexible method. Justify your answer.

1. The sample size $n$ is extremely large, and the number of predictors $p$ is small.
2. The number of predictors $p$ is extremely large, and the number of observations $n$ is small.
3. The relationship between the predictors and response is highly non-linear.
4. The variance of the error terms, i.e. $\sigma^2 = Var(\epsilon)$, is extremely high

**ANSWER**

1. Better. When the sample size is large and the number of p is small, the flexible model can better capture the complicated relationship given enough information. Since the number of predictors is also small, the probability of overfitting is lower.

2. Worse. When there are many predictors while the sample size is small, it might lead to the issue of over-fitting. The model would rather fit the noise, which is problematic and impact the model's performance.

3. Better. The flexible model can better capture complicated relationships such as the non-linear relationship. While the inflexible model like the linear model fails to capture this kind of relationship well.

4. Worse. The variance can be very high when there is noise in the data. The model might overfit the noise. The performance of the flexible model is poor when different datasets are given.

---

## 2

**[6 points]** For each of the following, (i) explain if each scenario is a classification or regression problem AND why, (ii) indicate whether we are most interested in inference or prediction for that problem AND why, and (iii) provide the sample size $n$ and number of predictors $p$ indicated for each scenario.

**(a)** We collect a set of data on the top 500 firms in the US. For each firm we record profit, number of employees, industry and the CEO salary. We are interested in understanding which factors affect CEO salary.

**(b)** We are considering launching a new product and wish to know whether it will be a success or a failure. We collect data on 20 similar products that were previously launched. For each product we have recorded whether it was a success or failure, price charged for the product, marketing budget, competition price, and ten other variables.

**(c)** We are interesting in predicting the % change in the US dollar in relation to the weekly changes in the world stock markets. Hence we collect weekly data for all of 2012. For each week we record the % change in the dollar, the % change in the US market, the % change in the British market, and the % change in the German market.

**ANSWER**

### (a).

#### (i)

Regression. We want to explore the association between different factors and the CEO salary, which is continuous.

#### (ii)

Inference. Rather than predicting something, the goal is to understand which factors affect the CEO's salary.

#### (iii)

n=500. p=3.

### (b).

#### (i)

Classification. The outcome variable: success or failure is binary.

#### (ii)

Prediction. In this example, we want to predict whether the new product will be a success or a failure, rather than explore what factors will affect the success or failure result.

#### (iii)

n=20. p=13.

### (c).

#### (i)

Regression. Our dependent variable is the percentage change in $, and our independent variables are the % change in the US market, the % change in the British market, and the % change in the German market, which are all continuous.

#### (ii)

Prediction. The reason is that we are trying to predict the percentage change in $ in relation to the weekly changes in different stock markets.

#### (iii)

n= 52 (total number of weeks in 2012: 365/7=52). p=3.

---

# Practical Questions

## 3

**[6 points] Classification using KNN**. The table below provides a training dataset containing six observations (a.k.a. samples) ($n = 6$) each with three predictors (a.k.a. features) ($p = 3$), and one qualitative response variable (a.k.a. target).

*Table 1. Training dataset with $n = 6$ observations in $p = 3$ dimensions with a categorical response, $y$*

| **Obs.** | $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|---|
| **1** | 0 | 3 | 0 | Red |
| **2** | 2 | 0 | 0 | Red |
| **3** | 0 | 1 | 3 | Red |
| **4** | 0 | 1 | 2 | Blue |
| **5** | -1 | 0 | 1 | Blue |
| **6** | 1 | 1 | 1 | Red |

We want to use the above training dataset to make a prediction, $\hat{y}$, for an unlabeled test data observation where $x_1 = x_2 = x_3 = 0$ using $K$-nearest neighbors. You are given some code below to get you started. *Note: coding is only required for part (a), for (b)-(d) please provide your reasoning based on your answer to part (a).*

**(a)** Compute the Euclidean distance between each observation and the test point, $x_1 = x_2 = x_3 = 0$. Present your answer in a table similar in style to Table 1 with observations 1-6 as the row headers.

**(b)** What is our prediction, $\hat{y}$, when $K = 1$ for the test point? Why?

**(c)** What is our prediction, $\hat{y}$, when $K = 3$ for the test point? Why?

**(d)** If the Bayes decision boundary (the optimal decision boundary) in this problem is highly nonlinear, then would we expect the *best* value of $K$ to be large or small? Why?

```
In [ ]: import numpy as np

        X = np.array([[ 0, 3, 0],
                      [ 2, 0, 0],
                      [ 0, 1, 3],
                      [ 0, 1, 2],
                      [-1, 0, 1],
                      [ 1, 1, 1]])
        y = np.array(['r','r','r','b','b','r'])
```

**ANSWER**:

## a. Compute the distance

```
In [ ]: T=[0,0,0]
        D1=math.dist(X[0],T)
        print("The distance between Obs.1 and the test point is {a:.2f} .".format(a=

        D2=math.dist(X[1],T)
        print("The distance between Obs.2 and the test point is {D2:.2f} .".format(D

        D3=math.dist(X[2],T)
        print("The distance between Obs.3 and the test point is {D3:.2f} .".format(D

        D4=math.dist(X[3],T)
        print("The distance between Obs.4 and the test point is {D4:.2f} .".format(D

        D5=math.dist(X[4],T)
        print("The distance between Obs.5 and the test point is {D5:.2f} .".format(D

        D6=math.dist(X[5],T)
        print("The distance between Obs.6 and the test point is {D6:.2f} .".format(D
        The distance between Obs.1 and the test point is 3.00 .
        The distance between Obs.2 and the test point is 2.00 .
        The distance between Obs.3 and the test point is 3.16 .
        The distance between Obs.4 and the test point is 2.24 .
        The distance between Obs.5 and the test point is 1.41 .
        The distance between Obs.6 and the test point is 1.73 .
```

| **Obs.** | $Distance\ from\ test\ point$ | **x1** | **x2** | **x3** | **y** |
|---|---|---|---|---|---|
| **1** | 3.00 | 0 | 3.00 | 0 | R |
| **2** | 2.00 | 2.00 | 0 | 0 | R |
| **3** | 3.16 | 0 | 1.00 | 3.00 | R |
| **4** | 2.24 | 0 | 1.00 | 2.00 | B |
| **5** | 1.41 | -1.00 | 0 | 1.00 | B |
| **6** | 1.73 | 1.00 | 1.00 | 1.00 | R |

### b. Prediction

The prediction is blue.

For K=1, we are looking at the closest observation from test point. Based on the calculation from question (a), the prediction is blue.

### c. Prediction

The prediction is red.

For K=3, we are looking at three closest observations from test point, which are red, blue and red respectively (Observations 2,5 and 6). There are 2 closest red points and one closest blue point. So, the prediction is red.

### d.

The best value of K would be expected to be small. When k is smaller, the model is more flexible with low bias and high variance, it can predict more complex non-linear relationships.

---

## 4

**[18 points] Build your own classification algorithm**.

**(a)** Build a working version of a binary KNN classifier using the skeleton code below. We'll use the `sklearn` convention that a supervised learning algorithm has the methods `fit` which trains your algorithm (for KNN that means storing the data) and `predict` which identifies the K nearest neighbors and determines the most common class among those K neighbors. *Note: Most classification algorithms typically also have a method `predict_proba` which outputs the confidence score of each prediction, but we will explore that in a later assignment.*

**(b)** Load the datasets to be evaluated here. Each includes training features ($\mathbf{X}$), and test features ($\mathbf{y}$) for both a low dimensional dataset ($p = 2$ features/predictors) and a higher dimensional dataset ($p = 100$ features/predictors). For each of these datasets there are $n = 1000$ observations of each. They can be found in the `data` subfolder in the `assignments` folder on github. Each file is labeled similar to `A2_X_train_low.csv`, which lets you know whether the dataset is of features, $X$, targets, $y$; training or testing; and low or high dimensions.

**(c)** Train your classifier on first the low dimensional dataset and then the high dimensional dataset with $k = 5$. Evaluate the classification performance on the corresponding test data for each of those trained models. Calculate the time it takes each model to make the predictions and the overall accuracy of those predictions for each corresponding set of test data - state each.

**(d)** Compare your implementation's accuracy and computation time to the scikit learn KNeighborsClassifier class. How do the results and speed compare to your implementation?

**(e)** Some supervised learning algorithms are more computationally intensive during training than testing. What are the drawbacks of the prediction process being slow? In what cases in practice might slow testing (inference) be more problematic than slow training?

**ANSWER**:

## a. Build my KNN classifier

In [ ]:
```
class Knn:
# k-Nearest Neighbor class object for classification training and testing
    def __init__(self):
        self.x_train=[]
        self.y_train=[]
    def fit(self, x, y):
        self.x_train=x
        self.y_train=y
    def predict(self, x, k):
        y_hat = []
        for test_i in x:
            estimate_distance=[]
            for train_i in self.x_train:
                estimate_distance.append(math.dist(test_i,train_i))
            sort=np.argpartition(estimate_distance,k)[:k]
            num=mode(self.y_train[sort])[0]
            y_hat.append(num)
        return np.concatenate(y_hat)

def accuracy(y,y_hat):
    nvalues = len(y)
    accuracy = sum(y == y_hat) / nvalues
    return accuracy
```

## b. Load the datasets

In [ ]:
```
x_train_low=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids7
y_train_low=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids7
x_test_low=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids70
y_test_low=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids70

x_train_high=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids
y_train_high=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids
x_test_high=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids7
y_test_high=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids7
```

## c. Train the classifier

In [ ]:
```
knn_low = Knn()

start_time_1 = time.time()
knn_low.fit(x_train_low, y_train_low)
y_hat_low = knn_low.predict(x_test_low, 5)
end_time_1 = time.time() - start_time_1

accuracy_1=accuracy(y_test_low, y_hat_low)[0]*100

print(f"The time for my KNN classifier to make predictions for low dimensior
```

```
The time for my KNN classifier to make predictions for low dimensional data
set is 0.79 seconds, with the accuracy of 92.50%.
```

In [ ]:
```
knn_high = Knn()

start_time_2=time.time()
knn_high.fit(x_train_high, y_train_high)
y_hat_high = knn_high.predict(x_test_high, 5)
end_time_2 = time.time() - start_time_2

accuracy_2=accuracy(y_test_high, y_hat_high)[0]*100
print(f"The time for my KNN classifier to make predictions for high dimensic
```

```
The time for my KNN classifier to make predictions for high dimensional dat
aset is 5.96 seconds, with the accuracy of 99.30%.
```

## d. Compare the results

In [ ]:
```
# knn
s_knn=KNeighborsClassifier(n_neighbors=5)

start_time_3=time.time()
s_knn.fit(x_train_low,y_train_low)
s_y_pred_low=s_knn.predict(x_test_low)
end_time_3=time.time() - start_time_3


accuracy_3=s_knn.score(x_test_low,y_test_low)*100

print(f"The time for scikit-learn KNN classifier to make predictions for low
```

```
The time for scikit-learn KNN classifier to make predictions for low dimens
ional dataset is 0.01 second, with the accuracy of 92.50%.
```

In [ ]:
```
s_knn=KNeighborsClassifier(n_neighbors=5)

start_time_4=time.time()
s_knn.fit(x_train_high,y_train_high)
s_y_pred_high=s_knn.predict(x_test_high)
end_time_4=time.time() - start_time_4


accuracy_4=s_knn.score(x_test_high,y_test_high)*100
print(f"The time for scikit-learn KNN classifier to make predictions for hig
```

```
The time for scikit-learn KNN classifier to make predictions for high dimen
sional dataset is 0.01 second, with the accuracy of 99.30%.
```

The overall accuracy scores for my KNN and scikit-learn KNN classifier for both low and high dimensional dataset are the same. However, the speed of my implementation is slower than the scikit-learn's KNN classifier.

## e. Analysis of the drawbacks

When the prediction process is slow, there are drawbacks like latency in prediction, and increasing the cost for prediction. When the computing resources are constrained, the long testing process can be problematic that impact the performance significantly. The cost for a long prediction time can be a lot in real-world applications. Moreover, in some cases when the real-time application is preferred, the slow testing can lead to latency in the decision-making process. For example, when detecting landmines during a war (in a very limited time). The latency of real-time translation can also significantly impact the users' experience.

---

# 5

**[20 points] Bias-variance tradeoff: exploring the tradeoff with a KNN classifier**. This exercise will illustrate the impact of the bias-variance tradeoff on classifier performance by investigating how model flexibility impacts classifier decision boundaries. For this problem, please us Scikit-learn's KNN implementation rather than your own implementation, as you did at the end of the last question.

**(a)** Create a synthetic dataset (with both features and targets). Use the `make_moons` module with the parameter `noise=0.35` to generate 1000 random samples.

**(b)** Visualize your data: scatterplot your random samples with each class in a different color.

**(c)** Create 3 different data subsets by selecting 100 of the 1000 data points at random three times (with replacement). For each of these 100-sample datasets, fit three separate k-Nearest Neighbor classifiers with: $k = \{1, 25, 50\}$. This will result in 9 combinations (3 datasets, each with 3 trained classifiers).

**(d)** For each combination of dataset and trained classifier plot the decision boundary (similar in style to Figure 2.15 from *Introduction to Statistical Learning*). This should form a 3-by-3 grid. Each column should represent a different value of $k$ and each row should represent a different dataset.

**(e)** What do you notice about the difference between the rows and the columns. Which decision boundaries appear to best separate the two classes of data? Which decision boundaries vary the most as the data change?

**(f)** Explain the bias-variance tradeoff using the example of the plots you made in this exercise and its implications for training supervised machine learning algorithms.

Notes and tips for plotting decision boundaries (as in part d):

- *Resource for plotting decision boundaries with meshgrid and contour: https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html*
- If you would like to change the colors of the background, and do not like any of the existing cmap available in matplotlib, you can make your own cmap using the 2 sets of rgb values. Sample code (replace r, g, b with respective rgb values):

**ANSWER**

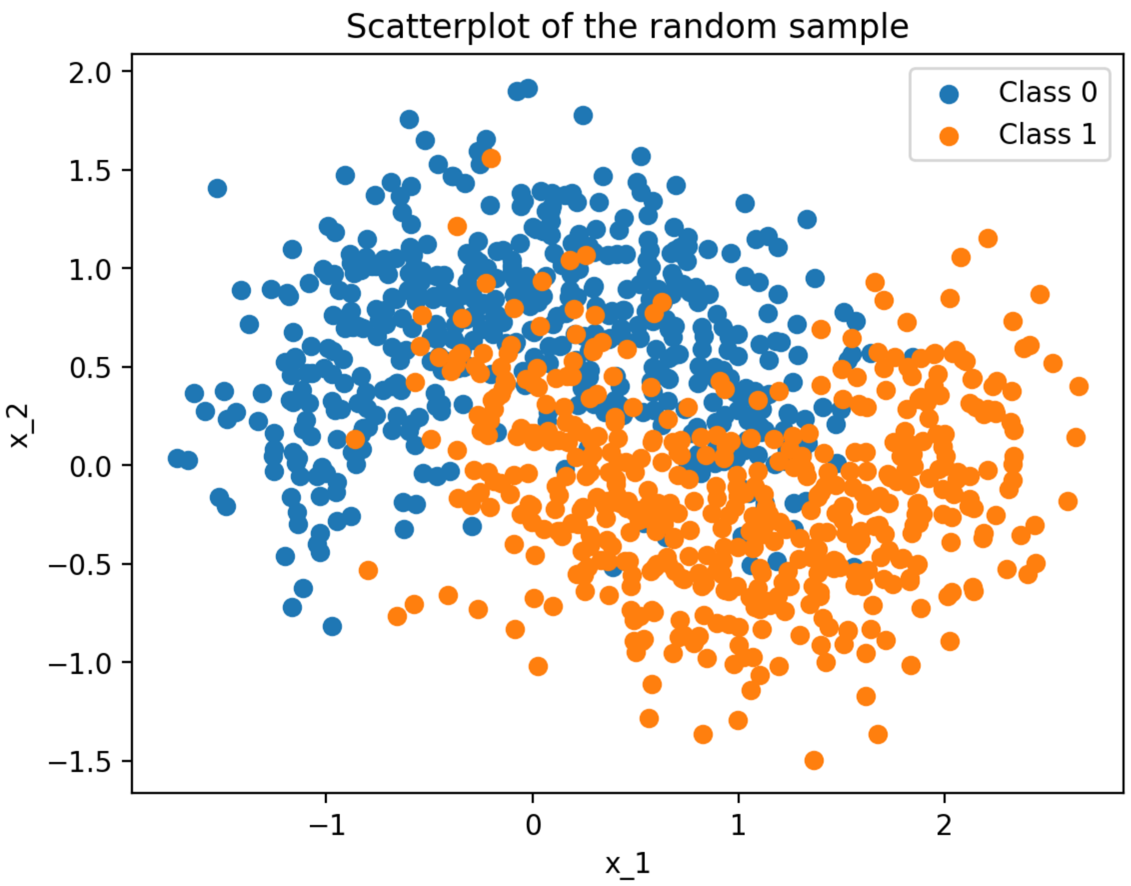
## a. Create synthetic dataset

```
In [ ]: x_train,y_train=make_moons(n_samples=1000,noise=0.35)
```

## b. Visualize the data

In [ ]:
```
plt.scatter(x_train[y_train==0, 0], x_train[y_train==0, 1],  label='Class 0'
plt.scatter(x_train[y_train==1, 0], x_train[y_train==1, 1],  label='Class 1'
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.legend()
plt.title("Scatterplot of the random sample")
```

Out[ ]:
```
Text(0.5, 1.0, 'Scatterplot of the random sample')
```

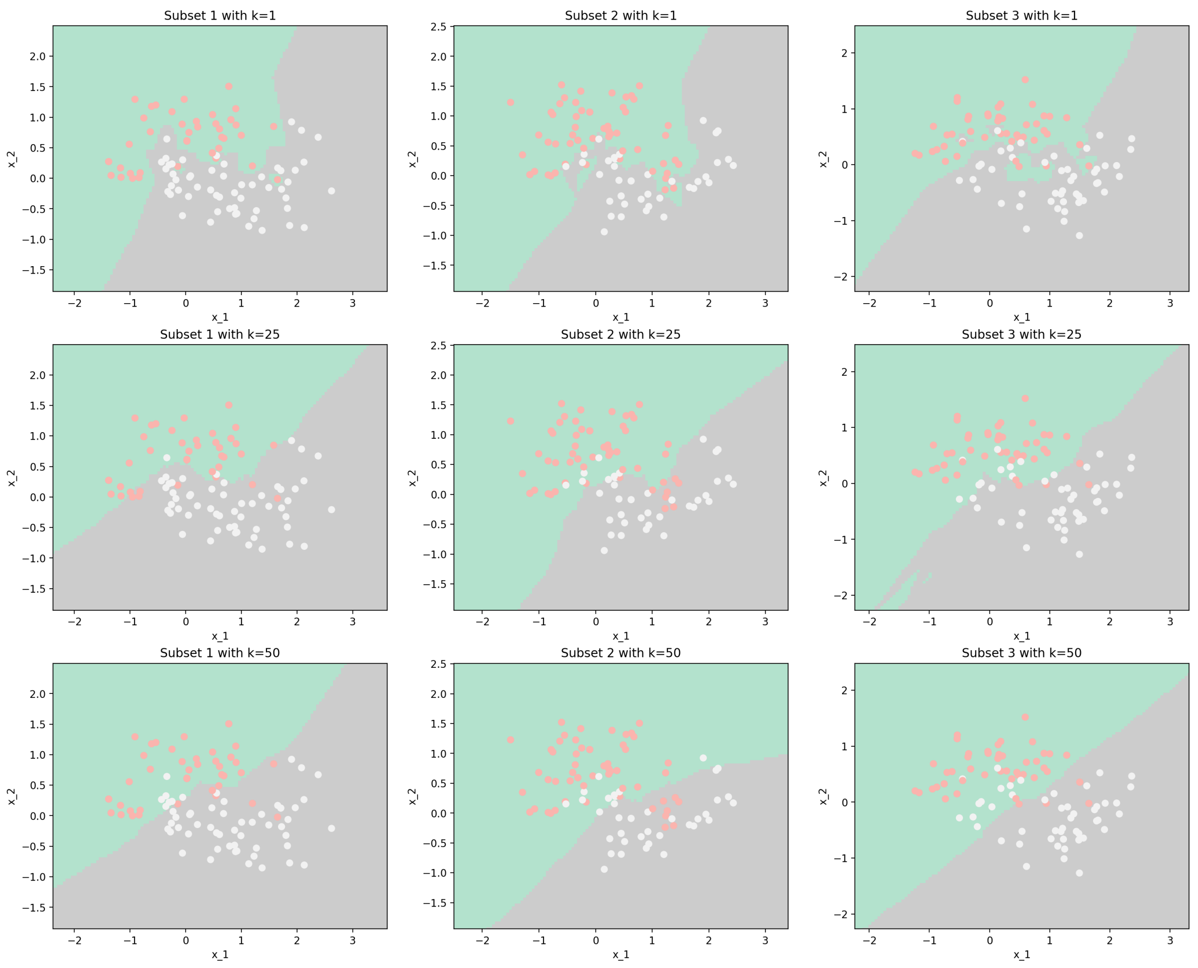


## c & d. Create data subset, fit KNN and plot the decision boundary

In [ ]:
```
df5=pd.DataFrame(
    {'x1':x_train[:, 0],
     'x2':x_train[:,1],
     'y':y_train}
)
```

In [ ]:
```
np.random.seed(42)
d_1=np.array(random.choices(df5.values,k=100))

d_2=np.array(random.choices(df5.values,k=100))

d_3=np.array(random.choices(df5.values,k=100))
```

In [ ]:
```
k_range=[1,25,50]
figure=plt.figure(figsize=(20, 16))
# iterate for k in k_range, and for each k, iterate for each df in (d_1,d_2,

for n1, k in enumerate(k_range):
    knn_5=KNeighborsClassifier(n_neighbors=k)
    for n2, df in enumerate([d_1,d_2,d_3]):
        knn_5.fit(df[:,0:2].tolist(),df[:,2].tolist())
        x_min,x_max = df[:,0].min() -1, df[:,0].max()+1
        y_min,y_max = df[:,1].min() -1, df[:,1].max()+1
        # set the step size to be 0.05 in meshgrid
        xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.05), np.arange(y_min,
        Z = knn_5.predict(np.c_[xx.ravel(), yy.ravel()])
        # create 3*3 subplots each time in the loop
        plt.subplot(3, 3, (1+3 * n1 + n2))
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z)
        plt.set_cmap(plt.cm.Pastel2)
        plt.pcolormesh(xx, yy, Z)######
        plt.scatter(df[:,0].tolist(), df[:,1].tolist(),c=df[:,2].tolist(),cm
        plt.xlabel('x_1')
        plt.ylabel('x_2')
        plt.title(f"Subset {n2+1} with k={k}")
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        pass

plt.show()
```



## e

From the plots in the previous question, in terms of generalization performance, when k=25, the decision boundaries seem the best that classify the moon-shape data (observed from the scatterplot). When k =1, the model best separates the data for each single case. However, it can lead to the issue of overfitting and high variance (when a different dataset is given).In the case when k=1, the decision boundaries vary most as the data change. The more flexible the model is likely to lead to the issue of overfitting and variation among different samples.

## f. Bias-variance tradeoff

According to previous plots, as k continues to increase (from 1 to 50), the model tends to be less flexible - there is little variation when different samples are provided, which also means that there is little variance. In this case, however, the bias tends to be larger to model with a smaller k. More data points have been tagged with the incorrect class (according to plots in the previous question). In comparison, when given a smaller k (k=1 in this example), there is low bias - most data are predicted correctly. However, the variance increases - the decision boundary vary a lot for different dataset. Thus, for training algorithms for machine learning, we want to find a balance where both bias and variance are relatively small. For this example, k=25 would be better since it fits the moon-shape data well, with lower variance and bias.

---

## 6

**[18 points] Bias-variance trade-off II: Quantifying the tradeoff**. This exercise explores the impact of the bias-variance tradeoff on classifier performance by looking at the performance on both training and test data.

Here, the value of $k$ determines how flexible our model is.

**(a)** Using the function created earlier to generate random samples (using the `make_moons` function setting the `noise` parameter to 0.35), create a new set of 1000 random samples, and call this dataset your test set and the previously created dataset your training set.

**(b)** Train a kNN classifier on your training set for $k = 1, 2, \ldots 500$. Apply each of these trained classifiers to both your training dataset and your test dataset and plot the classification error (fraction of incorrect predictions).

**(c)** What trend do you see in the results?

**(d)** What values of $k$ represent high bias and which represent high variance?

**(e)** What is the optimal value of $k$ and why?

**(f)** In KNN classifiers, the value of k controls the flexibility of the model - what controls the flexibility of other models?

**ANSWER**

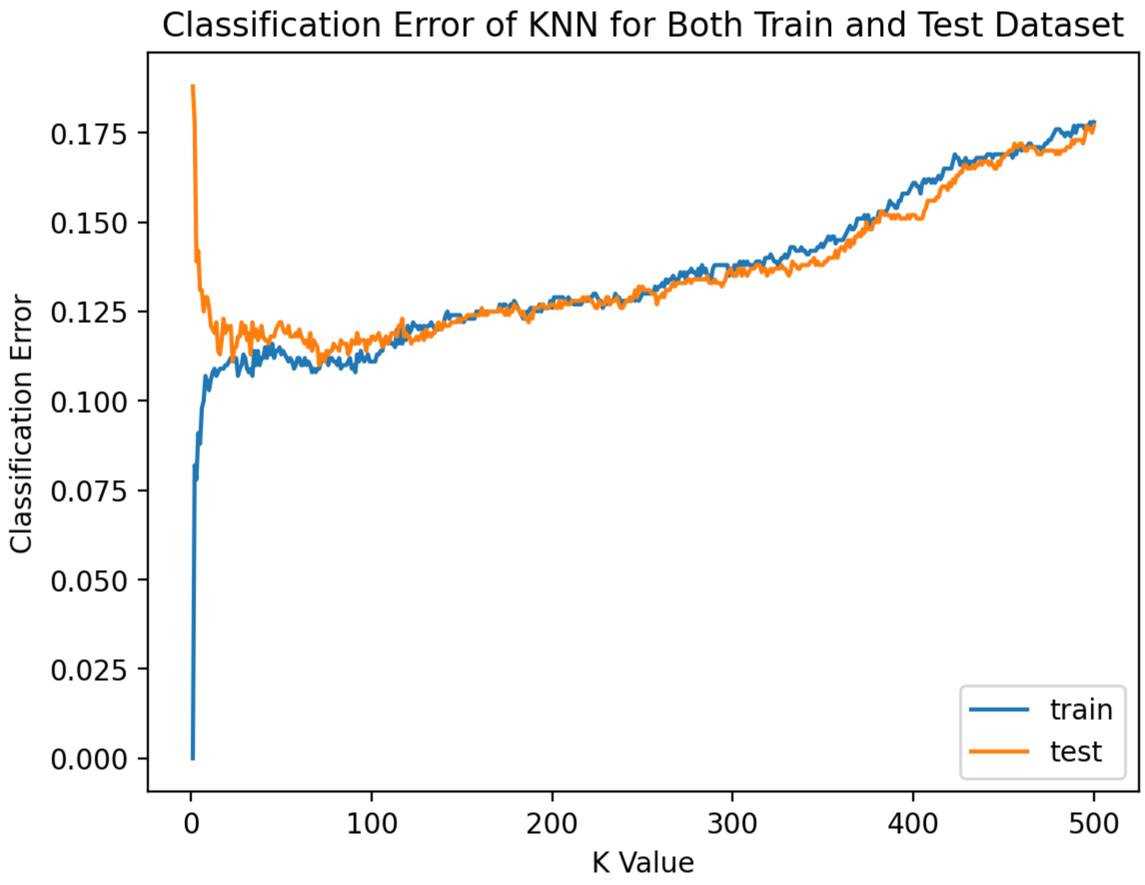
### a. Generate random samples

In [ ]:
```
x_test_6,y_test_6=make_moons(n_samples=1000,noise=0.35)
```

### b. Visualization of the classification error

In [ ]:
```
train=[]
test=[]
k_6=list(range(1,501))


for k in k_6:
    knn_6=KNeighborsClassifier(n_neighbors=k)
    knn_6.fit(x_train,y_train)
    train.append(1-knn_6.score(x_train,y_train))
    test.append(1-knn_6.score(x_test_6,y_test_6))
```

In [ ]:
```
plt.plot(k_6,train,label='train')
plt.plot(k_6,test,label='test')
plt.legend(loc='lower right')
plt.xlabel("K Value")
plt.ylabel("Classification Error")
plt.title("Classification Error of KNN for Both Train and Test Dataset")
plt.show()
```



## c. Trend

From the result, I find that when k is below approximately 10, the overall trend of the classification error for the train and test dataset is different. As k increases the error of test data decreases but that of training data increases. This change is significant. In this case, the misclassification error for test dataset is very large but that for the training data is small. It is due to the issue of overfitting. However, when k is greater than approximately 10, the error rate for both the training and test dataset continues to increase as k increases. Both of them fluctuate. At first, the error rate for test data is slightly greater than that of train data. When k is greater than approximately 270, the error of training data is greater than that of test data.

## d

The smaller k represents a high variance. In this case, the model is flexible but can overfit our data. When the test error is extremely high given the low train error, it represents the variance among different datasets. The larger k represents high bias, it is the case when the classification error of train and test data are relatively high.

## e. Optimal value of k

```
In [ ]: test.index(min(test))
```

```
Out[ ]: 20
```

From the graph, The optimal value of k is approximately in the interval [20,50]. Within this interval, the overall trend of the error is decreasing. Although my calculation above suggests that the lowest classification error for test data appears when k=20, this single value can be the "pseudo fitting" of the test data. For optimal value of k, I suggest that it is within the interval of [20,50].

## f

The model performance is influenced by hyperparameters and the number of features. Other examples are the learning rate for gradient descent as well as the coefficient of linear regression. Moreover, adding more features also increases flexibility. For example, adding quadratic term and cubic term in regression increase flexibility.

---

## 7

**[18 points] Linear regression and nonlinear transformations**. Linear regression can be used to model nonlinear relationships when feature variables are properly transformed to represent the nonlinearities in the data. In this exercise, you're given training and test data contained in files "A2_Q7_train.csv" and "A2_Q7_test.csv" in the "data" folder for this assignment. Your goal is to develop a regression algorithm from the training data that performs well on the test data.

*Hint: Use the scikit learn LinearRegression module.*

**(a)** Create a scatter plot of your training data.

**(b)** Estimate a linear regression model ($y = a_0 + a_1 x$) for the training data and calculate both the $R^2$ value and mean square error for the fit of that model for the training data. Also provide the equation representing the estimated model (e.g. $y = a_0 + a_1 x$, but with the estimated coefficients inserted. Consider this your baseline model against which you will compare other model options. *Evaluating performance on the training data is not a measure of how well this model would generalize to unseen data. We will evaluate performance on the test data once we see our models fit the training data decently well.*

**(c)** If features can be nonlinearly transformed, a linear model may incorporate those non-linear feature transformation relationships in the training process. From looking at the scatter plot of the training data, choose a transformation of the predictor variable, $x$ that may make sense for these data. This will be a multiple regression model of the form $y = a_0 + a_1 z_1 + a_2 z_2 + \ldots + a_n z_n$. Here $z_i$ could be any transformations of x - perhaps it's $\frac{1}{x}$, $log(x)$, $sin(x)$, $x^k$ (where $k$ is any power of your choosing). Provide the estimated equation for this multiple regression model (e.g. if you chose your predictors to be $z_1 = x$ and $z_2 = log(x)$, your model would be of the form $y = a_0 + a_1 x + a_2 log(x)$. Also provide the $R^2$ and mean square error of the fit for the training data.

**(d)** Visualize the model fit to the training data. Using both of the models you created in parts (b) and (c), plot the original data (as a scatter plot) AND the curves representing your models (each as a separate curve) from (b) and (c).

**(e)** Now its time to compare your models and evaluate the generalization performance on held out test data. Using the models above from (b) an (c), apply them to the test data and estimate the $R^2$ and mean square error of the test dataset.

**(f)** Which models perform better on the training data, and which on the test data? Why?

**(g)** Imagine that the test data were significantly different from the training dataset. How might this affect the predictive capability of your model? How would the accuracy of generalization performance be impacted? Why?

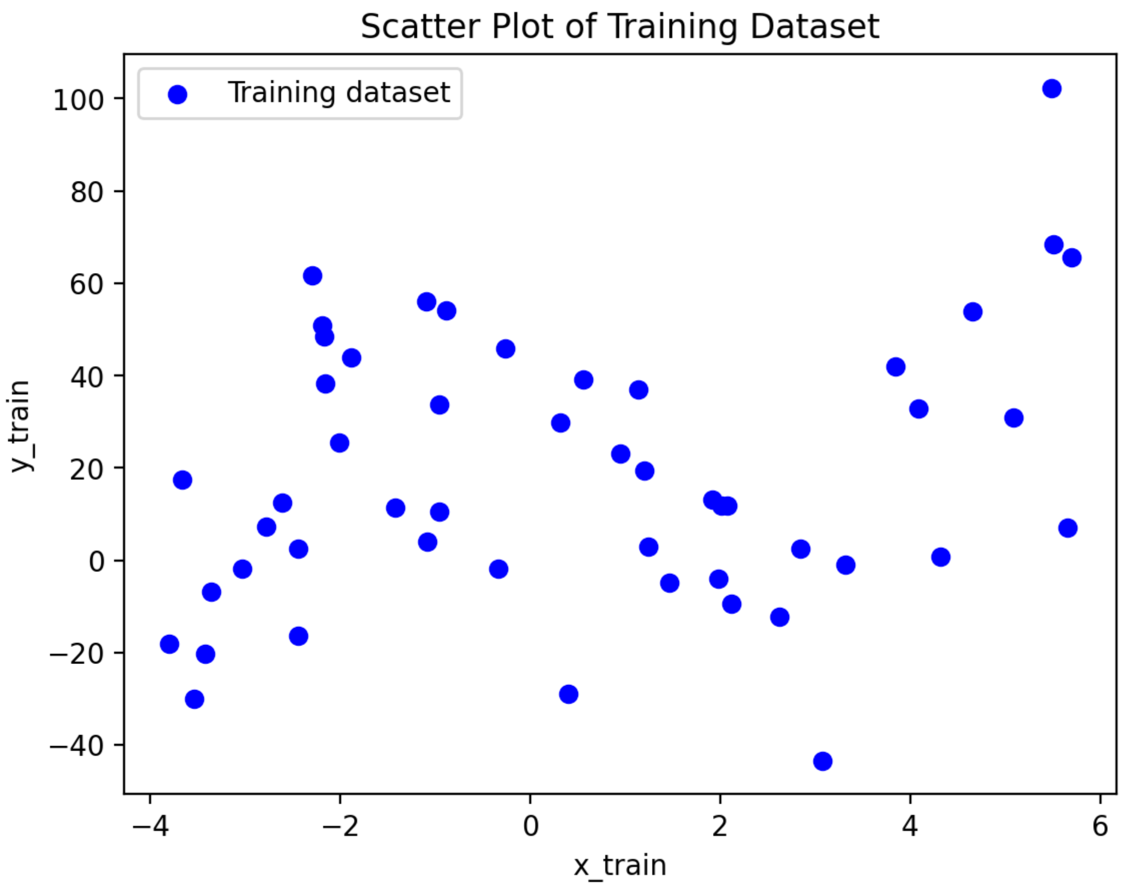
*To help get you started - here's some code to help you load in the data for this exercise (you'll just need to update the path)*:

**ANSWER**

## a. Create scatterplot of training data

```
In [ ]: train=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids705/mai
        test=pd.read_csv("https://raw.githubusercontent.com/kylebradbury/ids705/mair

        x_train_7 = train.x.values
        y_train_7 = train.y.values

        x_test_7 = test.x.values
        y_test_7 = test.y.values
```

In [ ]:
```
plt.scatter(x_train_7, y_train_7, c='b', label='Training dataset')
plt.title("Scatter Plot of Training Dataset")
plt.legend()
plt.xlabel("x_train")
plt.ylabel("y_train")
plt.show()
```



## b. Fit the linear model

In [ ]:
```
reg=LinearRegression().fit(x_train_7.reshape(-1,1),y_train_7)
predict1=reg.predict(x_train_7.reshape(-1,1))
r_score_1=metrics.r2_score(y_train_7,predict1)*100
MSE_1=metrics.mean_squared_error(y_train_7,predict1)

print(reg.coef_)
print(reg.intercept_)
```

```
[2.59072826]
17.204928179405222
```

In [ ]:
```
print(
    f"The R-squared value for this linear model is {r_score_1:.2f}%. The mea
)
```

```
The R-squared value for this linear model is 6.49%. The mean squared error
is 791.42.
```

This equation represents the linear regression: $\hat{y} = 17.2 + 2.59\hat{x}$

## c. Nonlinear transformation

In [ ]:
```
transformed_x_train_72=x_train_7**2
transformed_x_train_73=x_train_7**3

# combine x_train_7 with transformed_x_train_72 and transformed_x_train_73
x_train_matrix =np.column_stack((x_train_7,transformed_x_train_72,transforme

reg2=LinearRegression().fit(x_train_matrix,y_train_7)
predict2=reg2.predict(x_train_matrix)

r_score_2=metrics.r2_score(y_train_7,predict2)*100
MSE_2=metrics.mean_squared_error(y_train_7,predict2)


print(reg2.coef_)
print(reg2.intercept_)
```

```
[-9.25191526 -2.12568583  0.89700921]
24.155434157795092
```

In [ ]:
```
print(
    f"The R-squared value for the transformed model is {r_score_2:.2f}%. The
)
```

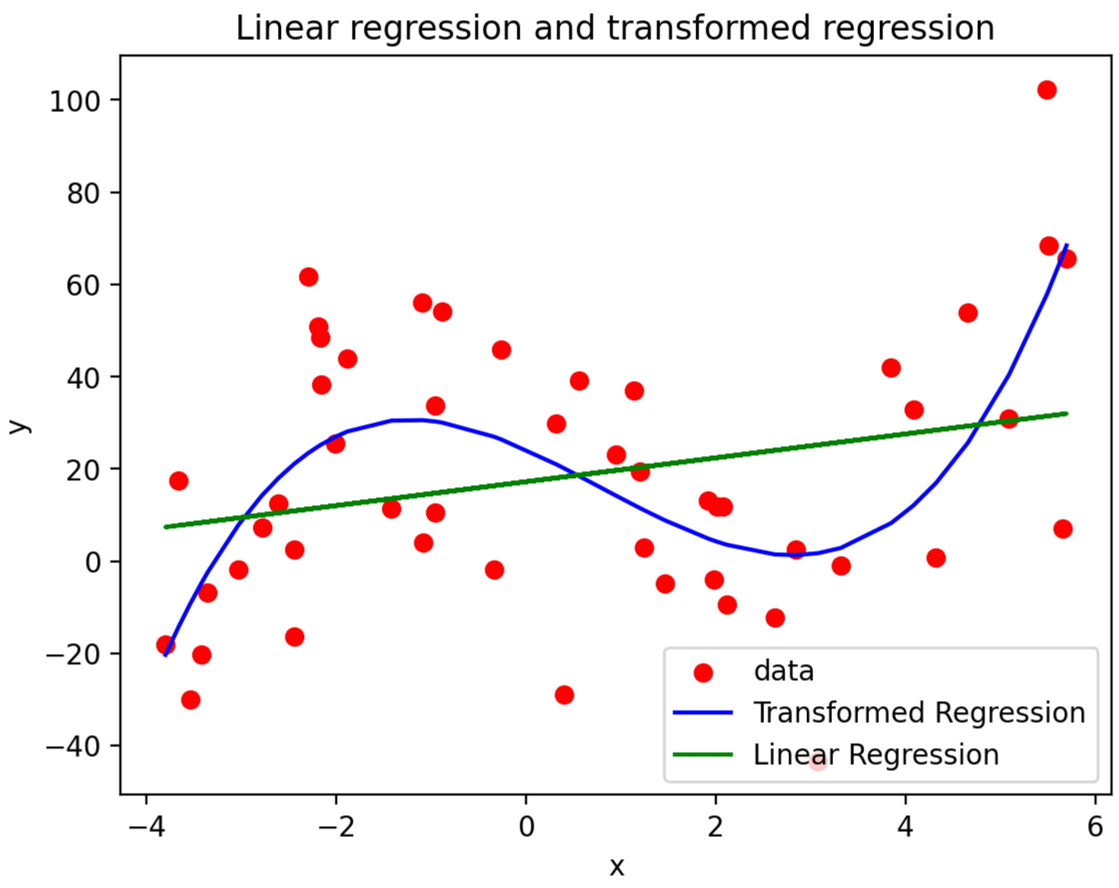
```
The R-squared value for the transformed model is 39.63%. The mean squared e
rror is 510.88.
```

This equation represents the transformed regression:

$$\hat{y} = -9.25\hat{x} - 2.13\hat{x^2} + 0.90\hat{x^3} + 24.16$$

## d. Visualize the data

In [ ]:
```
x_train_sorted = sorted(x_train_7)
ix = np.argsort(x_train_7.flatten())
y_t= predict2[ix]

plt.scatter(x_train_7, y_train_7, c='r', label='data')
plt.plot(x_train_sorted, y_t, c='b', label='Transformed Regression')
plt.plot(x_train_7, predict1, c='g', label='Linear Regression')
plt.legend(loc='lower right')
plt.title("Linear regression and transformed regression")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



## e. Compare the models and evaluate the generalization performance

In [ ]:
```
e_pred1=reg.predict(x_test_7.reshape(-1,1))
transformed_x_test_2=x_test_7**2
transformed_x_test_3=x_test_7**3
x_test_matrix =np.column_stack((x_test_7,transformed_x_test_2,transformed_x_

e_pred2=reg2.predict(x_test_matrix)
```

In [ ]:
```
mean_squared_error_e1=metrics.mean_squared_error(y_test_7, e_pred1)
mean_squared_error_e2=metrics.mean_squared_error(y_test_7, e_pred2)

r2_score_e1=metrics.r2_score(y_test_7, e_pred1)*100
r2_score_e2=metrics.r2_score(y_test_7, e_pred2)*100

print(f"The R-squared value of test data for linear regression is {r2_score_
print(f"The R-squared value of test data for transformed regression is {r2_s
```

```
The R-squared value of test data for linear regression is -13.29%. The mean
squared error for this is 1116.66.
The R-squared value of test data for transformed regression is 22.95%. The
mean squared error for this is 759.50.
```

## f. Comparison

The model after transformation performs better for both training and test data. For the transformed model, the mean squared error is lower and the R2 score is higher. For training data, the R-squared value for is 39.63%, and the mean squared error is 510.88. The R-squared value of test data for transformed regression is 22.95%, and the mean squared error is 759.50. All of them are better than the linear model.

The scatterplot of the data indicates that the relationship is likely to be nonlinear. So the regression after transformation can better capture this kind of nonlinear relationship.

## g. Different test data

Both the predictive capability and generalization performance will be impacted negatively if the test data is significantly different from the training data. It can cause problems of bias and overfitting. When the prediction of test data is biased, the generalization performance can be very poor. On the other hand, if the model fits too well for the training data, the performance can also be poor because it is overfitting. For this question, both the linear regression and transformed regression capture the trend of increasing y value. When the data of the opposite trend is given, both models will performance poorly. In order to avoid overfitting and achieve a better generalization performance, it is important that the test data are representative.