S&DS 355/555: Assignment 1

NetID: sh2432

Due: Sep 17, 2019 11:59pm

In [25]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Problem 1: Simple Linear Regression (25 points)

Problem 1.a:

In class we considered linear regression with the model

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

where $\epsilon \sim N(0, \sigma^2)$ for $i = 1, 2, \dots, n$. Suppose that we believe that the true value of β_0 is zero. In this case we now consider the simpler model $Y_i = \beta_1 X_i + \epsilon_i$. Find an expression for β_1, the estimate of β_1 that minimizes the sum of squared residuals for this simpler model

Since there are no assumptions about ϵ here. We would like to find the β that can minimize the Euclidean distance between Y and $X\beta$, i.e., the ℓ_2 norm of $Y - X\beta$. Therefore:

$$\hat{\beta} = argmin_{\beta \in R^p} ||Y - X\beta||^2.$$

To get the $\hat{\beta}$ which minimize the Euclidean distance, the derivation should calculate the gradient of the objective function $f(\beta) = ||Y - X\beta||^2$, and find the β which makes the gradient 0. We can express the solution as a function of the matrix X and the vector Y:

$$\frac{\partial ||Y - X\beta||^2}{\partial \beta} = -2X^T(Y - X\beta)$$

$$X^T(Y - X\hat{\beta}) = 0$$

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Problem 1.b:

Download the `fatherson.csv` file on Canvas with the Jupyter notebook for this homework assignment. This dataset, collected by Galton, contains the height of sons and the height of their father. To read it in, use the function below:

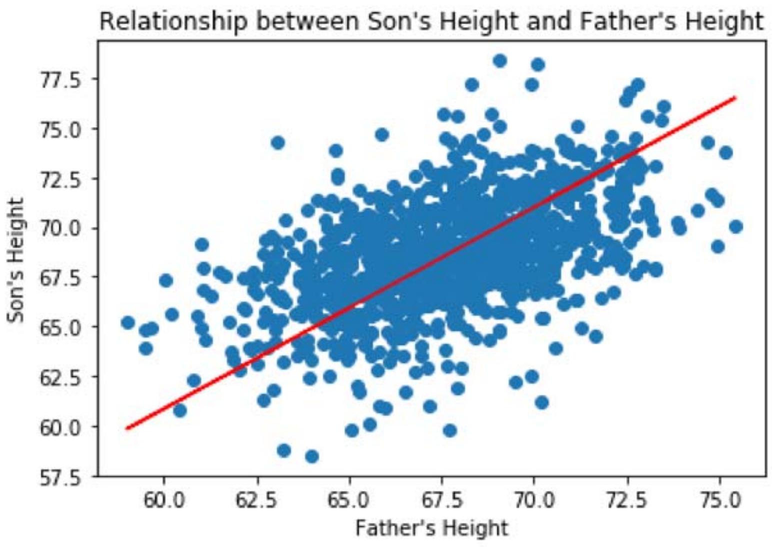
```
x = pd.read_csv("fatherson.csv")
```

After reading in this dataset, create a scatterplot of the sons' heights (on the Y-axis) versus the fathers' heights. Use your answer from (a) to calculate the slope of the least-squares line under the model with no intercept:

$$\text{Son}_i = \beta_1 \text{Father}_i + \epsilon_i$$

Add the fitted line to the scatterplot.

In [26]:
```
# load the data
x = pd.read_csv("fatherson.csv")
# scatter plot father's height vs son's height
plt.scatter(x["fheight"], x["sheight"])
plt.title("Relationship between Son's Height and Father's Height")
plt.xlabel("Father's Height")
plt.ylabel("Son's Height")
# calculate the scope
son = np.transpose(np.asmatrix(x["sheight"]))
father = np.transpose(np.asmatrix(x["fheight"]))
beta = np.linalg.inv(np.transpose(father) * father) * np.transpose(father) * s
on #calculate the estimated beta
beta
# fit the line
slope = np.asscalar(np.array(beta))
y=slope*father
plt.plot(father, y, 'r');
```



Problem 1.c:

Interpret the meaning of the coefficient β_1 in the context of Galton’s father-son dataset.

The meaning of β_1 : On average, one unit increase of father's height will introduce 1.01 unit increase of son's height.

Problem 1.d:

Use the equations provided in class (for the least-squares coefficients of the linear regression model that includes an intercept) to calculate the least-squares estimates of the coefficients for the linear model that includes a slope and an intercept:

$$\text{Son}_i = \beta_0 + \beta_1 \text{Father}_i + \epsilon_i$$

In [27]:
```
x_bar = np.mean(x["fheight"])
y_bar = np.mean(x["sheight"])
a= x["fheight"]-x_bar
b= x["sheight"]-y_bar
beta1=sum(a*b)/sum(a**2)
beta0=y_bar-beta1*x_bar
beta1
beta0;
```

The estimated β_1 is 0.514, while the estimated β_0 is 33.887. Therefore:

$$\hat{S}on_i = 33.887 + 0.514 * Father_i$$

Problem 2: Linear regression and classification (30 points)

Citi Bike is a public bicycle sharing system in New York City. There are hundreds of bike stations scattered throughout the city. Customers can check out a bike at any station and return it at any other station. Citi Bike caters to both commuters and tourists. Details on this program can be found at https://www.citibikenyc.com/ (https://www.citibikenyc.com/)

For this problem, you will build models to predict Citi Bike usage, in number of trips per day. The dataset consists of Citi Bike usage information and weather data recorded from Central Park.

In the `citibike_*.csv` files, we see:

1. date
2. trips: the total number of Citi Bike trips. This is the outcome variable.
3. n_stations: the total number of Citi Bike stations in service
4. holiday: whether or not the day is a work holiday
5. month: taken from the date variable
6. dayofweek: taken from the date variable

In the `weather.csv` file, we have:

1. date
2. PRCP: amount precipitation (i.e. rainfall amount) in inches
3. SNWD: snow depth in inches
4. SNOW: snowfall in inches
5. TMAX: maximum temperature for the day, in degrees F
6. TMIN: minimum temperature for the day, in degrees F
7. AWND: average windspeed

You are provided a training set consisting of data from 7/1/2013 to 3/31/2016, and a test set consisting of data after 4/1/2016. The weather file contains weather data for the entire year.

Problem 2.a: Read in and merge the data.

To read in the data, you can run, for example:

```
train = pd.read_csv("citibike_train.csv")
test = pd.read_csv("citibike_test.csv")
```

Merge the training and test data with the weather data, by date. Once you have successfully merged the data, you may drop the "date" variable; we will not need it for the rest of this assignment.

In [91]:
```
train = pd.read_csv("citibike_train.csv")
test = pd.read_csv("citibike_test.csv")
weather = pd.read_csv("weather.csv")
test1= pd.merge(test,weather, on="date")
train1=pd.merge(train,weather, on="date")
del test1["date"]
del train1["date"]
test1.head(n=10)
train1.head(n=10)
```

Out[91]:

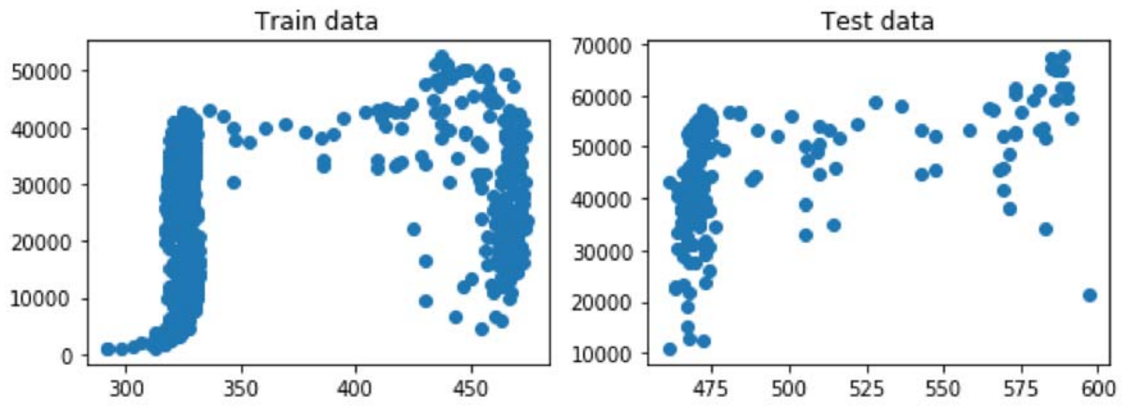
	trips	n_stations	holiday	month	dayofweek	PRCP	SNWD	SNOW	TMAX	TMIN	AWND
0	16650	326	False	Jul	Mon	0.84	0.0	0.0	77	72	3.1
1	22745	327	False	Jul	Tues	0.08	0.0	0.0	82	72	2.7
2	21864	326	False	Jul	Wed	0.53	0.0	0.0	83	73	4.3
3	22326	326	True	Jul	Thurs	0.00	0.0	0.0	87	75	4.3
4	21842	325	False	Jul	Fri	0.00	0.0	0.0	90	76	4.9
5	20467	324	False	Jul	Sat	0.00	0.0	0.0	92	78	4.5
6	20477	326	False	Jul	Sun	0.00	0.0	0.0	92	78	4.3
7	21615	326	False	Jul	Mon	0.22	0.0	0.0	89	73	4.7
8	26641	326	False	Jul	Tues	0.23	0.0	0.0	88	74	3.1
9	25732	326	False	Jul	Wed	0.00	0.0	0.0	85	75	4.3

For the rest of this problem, you will train your models on the training data and evaluate them on the test data.

As always, before you start any modeling, you should look at the data. Make scatterplots of some of the numeric variables. Look for outliers and strange values. Comment on any steps you take to remove entries or otherwise process the data. Also comment on whether any predictors are strongly correlated with each other.

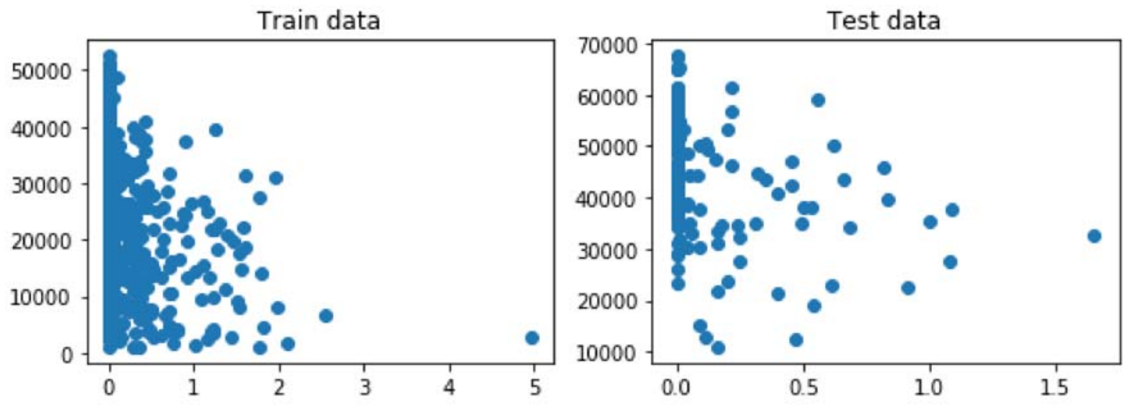
In [72]:
```
print("n_stations vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["n_stations"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["n_stations"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
n_stations vs trips
```



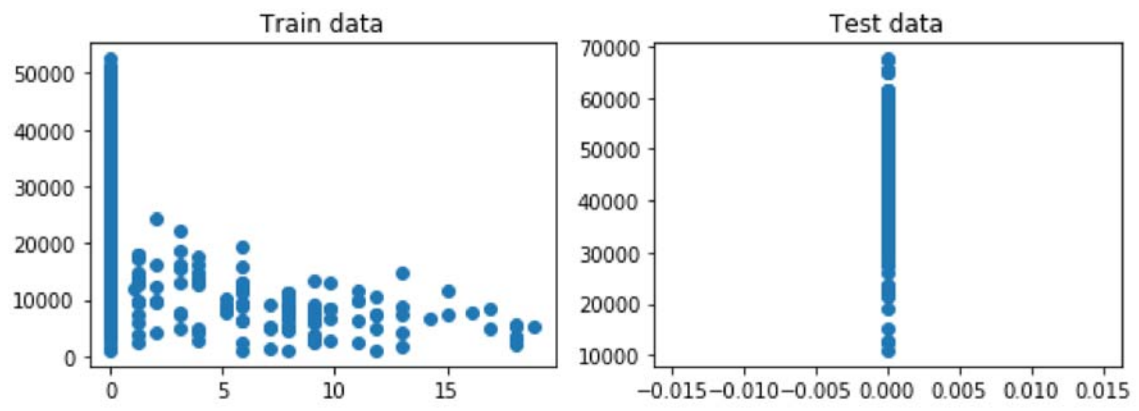
In [73]:
```
print("PRCP vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["PRCP"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["PRCP"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
PRCP vs trips
```



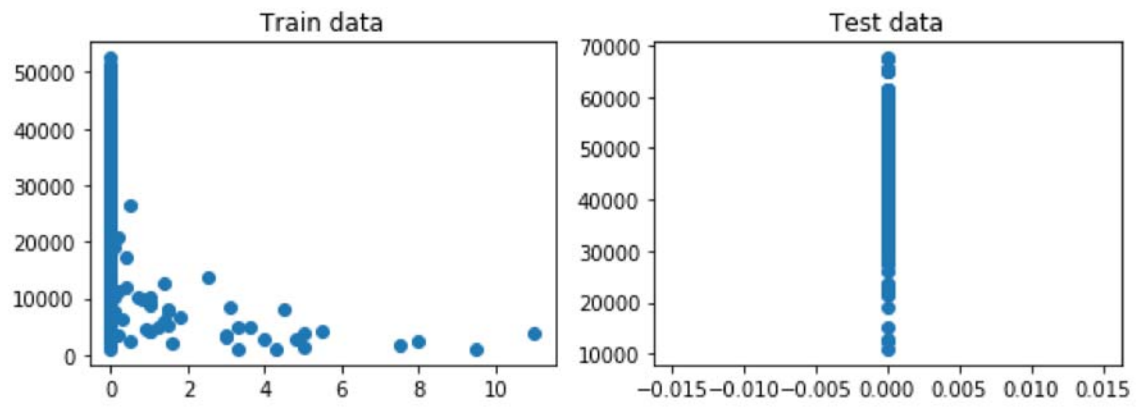
In [74]:
```
print("SNWD vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["SNWD"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["SNWD"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
SNWD vs trips
```



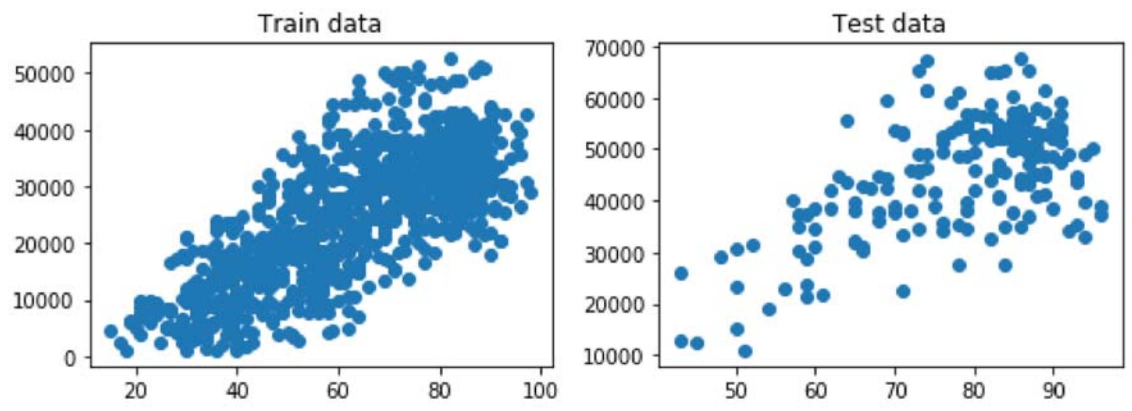
In [75]:
```
print("SNOW vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["SNOW"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["SNOW"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
SNOW vs trips
```



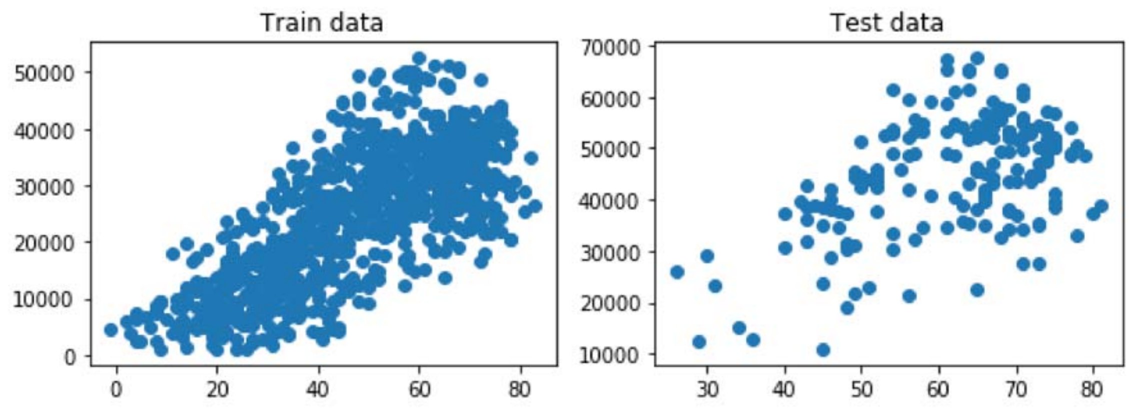
In [76]:
```python
print("TMAX vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["TMAX"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["TMAX"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
TMAX vs trips
```



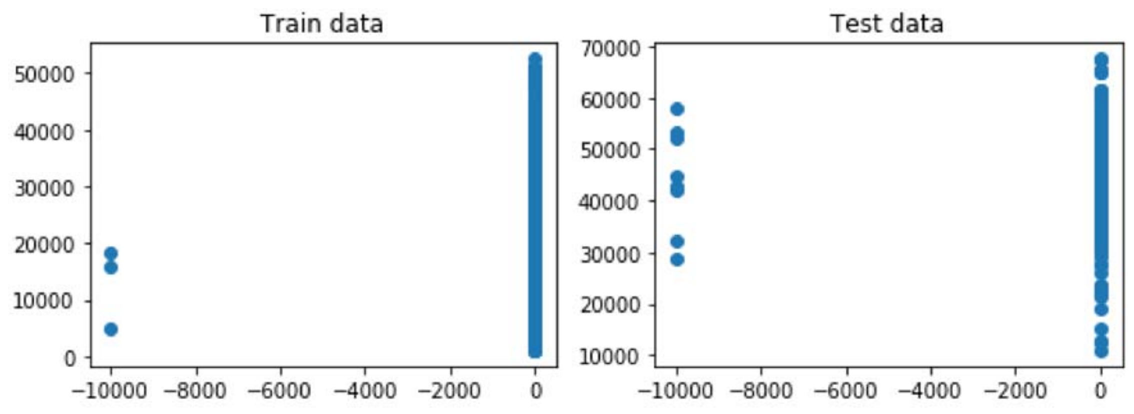
In [77]:
```python
print("TMIN vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["TMIN"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["TMIN"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
TMIN vs trips
```



In [78]:
```
print("AWND vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["AWND"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["AWND"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

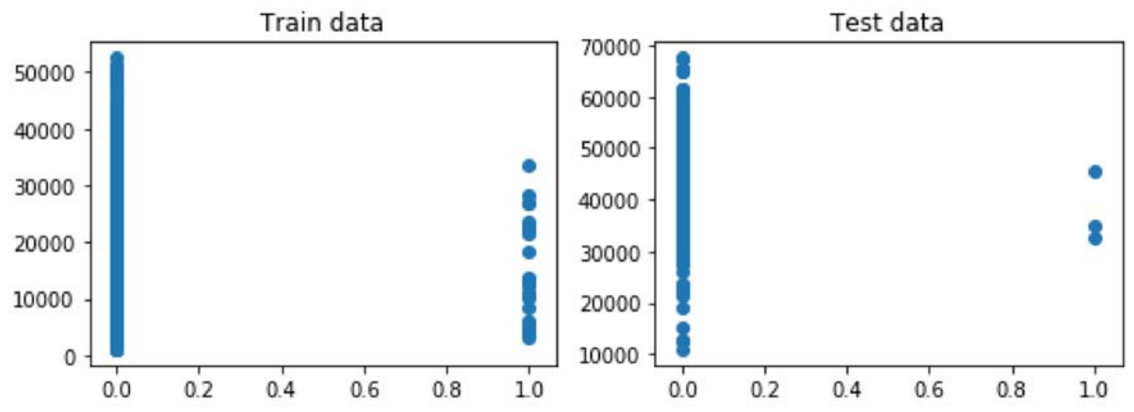
```
AWND vs trips
```



In [79]:
```
#replace -9999 to NA
test1["AWND"]=test1["AWND"].replace(-9999, np.NaN)
train1["AWND"]=train1["AWND"].replace(-9999, np.NaN)
#plt.scatter(test1["AWND"], test1["trips"])
#plt.scatter(train1["AWND"], train1["trips"]);
```

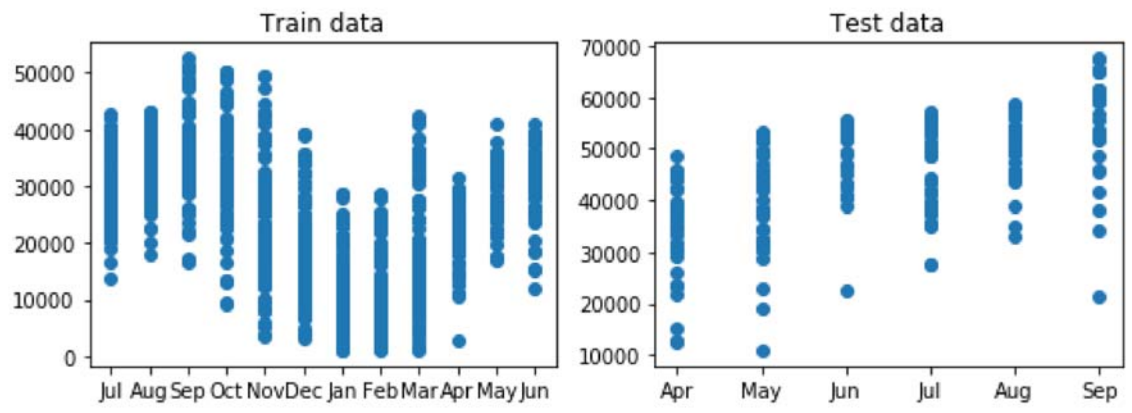
In [80]:
```
print("holiday vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["holiday"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["holiday"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
holiday vs trips
```



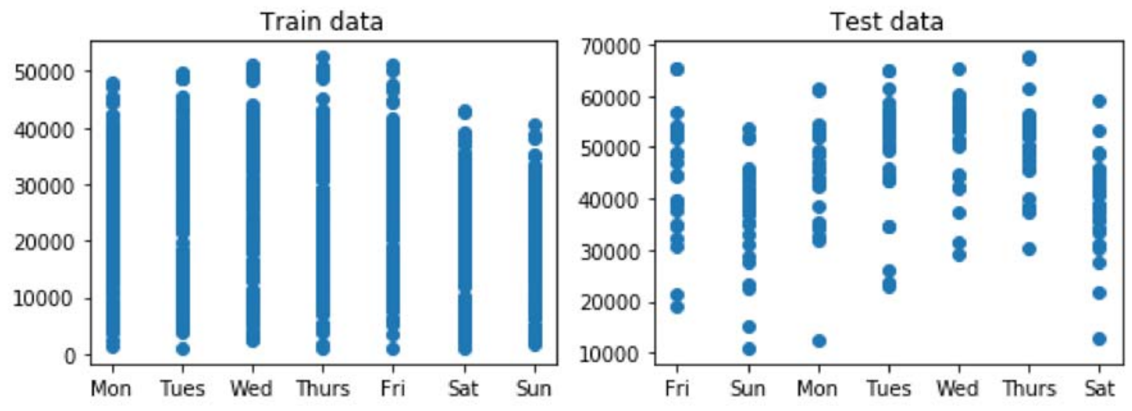
In [81]:
```
print("month vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["month"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["month"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
month vs trips
```



In [82]:
```
print("dayofweek vs trips")
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))
axes[0].scatter(train1["dayofweek"], train1["trips"])
axes[0].set_title("Train data");
axes[1].scatter(test1["dayofweek"], test1["trips"])
axes[1].set_title("Test data")
fig.tight_layout()
```

```
dayofweek vs trips
```



AWND is the average windspeed whcih should be positive. However, There are several negative values (-9999) existing, and I set those values to NA in both the test and training datasets. SNOW and SNWD are both listed as 0 in the test dataset. In both the test and training set, TMAX and TMIN are strongly correlated with trips.

Problem 2.b: Linear regression

Fit a linear regression model to predict the number of trips. Include all the covariates in the data. You may import the `statsmodels.api` module to get a R-like statistical output. You may write code as:

```
import statsmodels.api as sm

X = sm.add_constant(X) # to get the intercept term

model = sm.OLS(y,X).fit()

model.summary()
```

Next, find the "best" linear model that uses only p variables, for each $p = 1, 2, 3, 4, 5$. It is up to you to choose how to select the "best" subset of variables. (A categorical variable or factor such as "month" corresponds to a single variable.) Describe how you selected each model. Give the R^2 and the mean squared error (MSE) on the training and test set for each of the models. Which model gives the best fit to the data? Comment on your findings.

In [83]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
y=train1["trips"]
#full model
r1 = smf.ols('y ~ n_stations+C(holiday)+C(month)+C(dayofweek)+PRCP+SNWD+SNOW+T
MAX+TMIN+AWND', data=train1).fit()
r1.summary()
r1.mse_model
#MSE=4635540122.581449, R^2=0.877

#get rid of SNOW
r2 = smf.ols('y ~ n_stations+C(holiday)+C(month)+C(dayofweek)+PRCP+SNWD+TMAX+T
MIN+AWND', data=train1).fit()
r2.summary()
r2.mse_model
#MSE=4828570297.333403, R^2=0.877

#get rid of dayofweek
r3 = smf.ols('y ~ n_stations+C(holiday)+C(month)+PRCP+SNWD+TMAX+TMIN+AWND', da
ta=train1).fit()
r3.summary()
r3.mse_model
#MSE=6050178035.11984, R^2=0.824

#get rid of TWIN
r4 = smf.ols('y ~ n_stations+C(holiday)+C(month)+PRCP+SNWD+TMAX+AWND', data=tr
ain1).fit()
r4.summary()
r4.mse_model
#MSE=6402134321.374876, R^2=0.824

#get rid of month
r5 = smf.ols('y ~ n_stations+C(holiday)+PRCP+SNWD+TMAX+AWND', data=train1).fit
()
r5.summary()
r5.mse_model
#MSE=16962547235.228205, R^2=0.770

#get rid of holiday
r6 = smf.ols('y ~ n_stations+PRCP+SNWD+TMAX+AWND', data=train1).fit()
r6.summary()
r6.mse_model
#MSE=19828820706.66823, R^2=0.750
```

Out[83]:
```
19828820706.66823
```

In [84]:
```
model=np.array([1,2,3,4,5,6])
mse_train=np.array([4635540122.581449, 4828570297.333403, 6050178035.11984, 64
02134321.374876, 16962547235.228205,19828820706.66823])
r2_train=np.array([0.877, 0.877, 0.824, 0.824, 0.770, 0.750])
```

In [85]:
```
y1=test1["trips"]
#full model
rt1 = smf.ols('y1 ~ n_stations+C(holiday)+C(month)+C(dayofweek)+PRCP+SNWD+SNOW
+TMAX+TMIN+AWND', data=test1).fit()
rt1.summary()
rt1.mse_model
#MSE=1093859397.9515193, R^2= 0.813

#get rid of SNOW
rt2 = smf.ols('y1 ~ n_stations+C(holiday)+C(month)+C(dayofweek)+PRCP+SNWD+TMAX
+TMIN+AWND', data=test1).fit()
rt2.summary()
rt2.mse_model
#MSE=1093859397.9515193, R^2= 0.813

#get rid of dayofweek
rt3 = smf.ols('y1 ~ n_stations+C(holiday)+C(month)+PRCP+SNWD+TMAX+TMIN+AWND',
data=test1).fit()
rt3.summary()
rt3.mse_model
#MSE=1331767382.130918, R^2= 0.640

#get rid of TWIN
rt4 = smf.ols('y1 ~ n_stations+C(holiday)+C(month)+PRCP+SNWD+TMAX+AWND', data=
test1).fit()
rt4.summary()
rt4.mse_model
#MSE=1438711875.4733171, R^2= 0.629

#get rid of month
rt5 = smf.ols('y1 ~ n_stations+C(holiday)+PRCP+SNWD+TMAX+AWND', data=test1).fi
t()
rt5.summary()
rt5.mse_model
#MSE=2711088756.0772038, R^2= 0.593

#get rid of holiday
rt6 = smf.ols('y1 ~ n_stations+PRCP+SNWD+TMAX+AWND', data=test1).fit()
rt6.summary()
rt6.mse_model
#MSE=3387215758.3988085, R^2=0.592
```

Out[85]:
```
3387215758.3988085
```

In [86]:
```
mse_test=np.array([1093859397.9515193, 1093859397.9515193, 1331767382.130918,
1438711875.4733171, 2711088756.0772038, 3387215758.3988085])
r2_test=np.array([0.813, 0.813, 0.640, 0.629, 0.593, 0.592])
```

In [87]:
```
report= pd.DataFrame(columns=[model, mse_train, r2_train, mse_test, r2_test])
report=report.transpose()
print("train & test models MSE & r^2")
report
```

```
train & test models MSE & r^2
```

Out[87]:

1	**4.635540e+09**	**0.877**	**1.093859e+09**	**0.813**
2	**4.828570e+09**	**0.877**	**1.093859e+09**	**0.813**
3	**6.050178e+09**	**0.824**	**1.331767e+09**	**0.640**
4	**6.402134e+09**	**0.824**	**1.438712e+09**	**0.629**
5	**1.696255e+10**	**0.770**	**2.711089e+09**	**0.593**
6	**1.982882e+10**	**0.750**	**3.387216e+09**	**0.592**

I first consider all variables into the model and genearte a full modle, then based on the p-values of each coefficient, I took one variable out each time until I have the best 5 predictors left in the model. The final linear model with n_stations, PRCP, SNWD, TMAX, and AWND as independent variables should give the best fit to the data. For both the training and test set, this model has the highest r-squared value and lowest MSE.

Problem 2.c: KNN Classification

Now we will transform the outcome variable to allow us to do classification. Create a new vector Y with entries:

$$Y[i] = \mathbf{1}\{trips[i] > median(trips)\}$$

Use the median of the variable from the full data (training and test combined). After computing the binary outcome variable Y, you should drop the original trips variable from the data.

In [92]:
```
#append test and train datasets
append = pd.concat([test1, train1], axis=0)
append["median"]=np.median(append["trips"])
m=np.median(append["trips"])
append["n"]=np.where(append.trips>=m, 1, 0)
del append["trips"]
```

Recall that in k-nearest neighbors classification, the predicted value $\hat{Y}$ of X is the majority vote of the labels for the k nearest neighbors X_i to X. We will use the Euclidean distance as our measure of distance between points. Note that the Euclidean distance doesn't make much sense for factor variables, so just drop the predictors that are categorical for this problem. Standardize the numeric predictors so that they have mean zero and constant standard deviation.

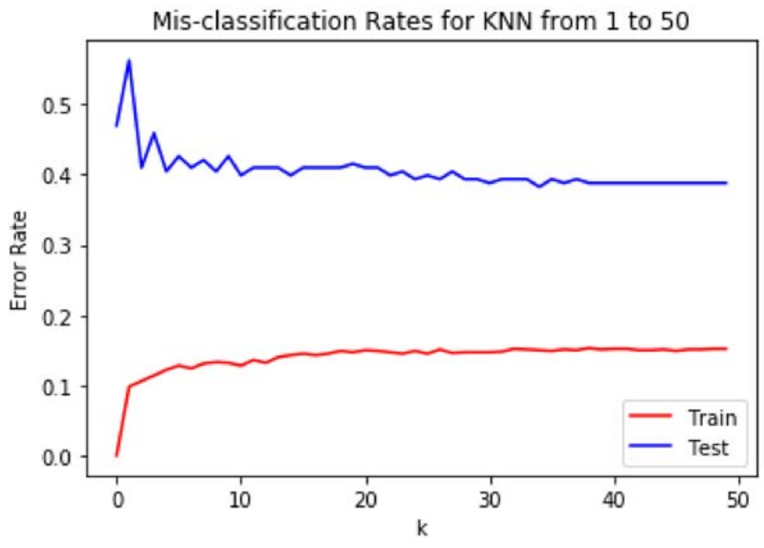
In [93]:
```
test2=test1.drop(columns=['holiday', 'month','dayofweek'])
test2.head(n=10)
train2=train1.drop(columns=['holiday', 'month','dayofweek'])
train2.head(n=10)

from sklearn import preprocessing as pr
n1 = train2.columns
n2 = test2.columns
scaler = pr.StandardScaler()
#standardized numeric predictors
##train data
train_s = scaler.fit_transform(train2)
train_s = pd.DataFrame(train_s, columns=n1)
train_s =train_s.drop(columns=['trips'])
train_s.head(n=10)
##test data
test_s = scaler.fit_transform(test2)
test_s = pd.DataFrame(test_s, columns=n2)
test_s =test_s.drop(columns=['trips'])
test_s.head(n=10)
```

Out[93]:

	n_stations	PRCP	SNWD	SNOW	TMAX	TMIN	AWND
0	-0.456371	-0.356864	0.0	0.0	0.112166	-0.070534	0.214871
1	-0.690572	-0.438091	0.0	0.0	-2.327444	-2.704759	0.213991
2	-0.526631	-0.397478	0.0	0.0	-1.065577	-1.651069	0.214969
3	-0.526631	0.374175	0.0	0.0	-1.570324	-1.475454	0.214431
4	-0.550051	-0.438091	0.0	0.0	-1.654448	-1.914492	0.213796
5	-0.526631	-0.438091	0.0	0.0	-1.317950	-1.651069	0.214774
6	-0.573471	-0.438091	0.0	0.0	-1.065577	-1.738877	0.213454
7	-0.620312	-0.438091	0.0	0.0	-0.813203	-1.651069	0.213991
8	-0.573471	-0.438091	0.0	0.0	-0.224332	-1.563262	0.213014
9	-0.526631	-0.438091	0.0	0.0	0.364539	-0.948609	0.213454

You may use the `KNeighborsClassifier` function from the `sklearn.neighbors` module to perform k-nearest neighbor classification, using as the neighbors the labeled points in the training set. Fit a classifier for $k = 1 : 50$, and find the mis-classification rate on both the training and test sets for each k. On a single plot, show the training set error and the test set error as a function of k. How would you choose the optimal k? Comment on your findings, and in particular on the possibility of overfitting.

In [94]:
```
#conert dataframe to array
y_test=np.where(test1.trips>=m, 1, 0)
y_train=np.where(train1.trips>=m, 1, 0)
x_test=test_s.to_numpy()
x_train=train_s.to_numpy()

from sklearn.neighbors import KNeighborsClassifier as kn
from sklearn import metrics as me

#KNN loop from 1 to 50
train_err=[i for i in range(1,51)]
test_err=[i for i in range(1,51)]
for k in range(1,51) :
    knn = kn(n_neighbors = k)
    knn.fit(x_train, y_train)
    train_err[k-1]=1-knn.score(x_train,y_train)
    test_err[k-1]=1-knn.score(x_test,y_test)

#import matplotlib.pyplot as plt
plt.plot(train_err, 'r', label='Train')
plt.title("Mis-classification Rates for KNN from 1 to 50")
plt.plot(test_err, 'b', label='Test')
plt.legend(loc='lower right')
plt.xlabel("k")
plt.ylabel("Error Rate");
```



I will choose the optimal k which makes the error rates in both training and test datasets relatively small. If k only minimizes the error rate in the training set, there might be overfitting issue and the predicting power for the test dataset will be limited. Based on the graph, a k around 10 is optional.

Problem 3: Classification for a Gaussian Mixture (25 points)

A Gaussian mixture model is a random combination of multiple Gaussians. Specifically, we can generate n data points from such a distribution in the following way. First generate labels $Y_1, \cdots, Y_n$ according to

$$Y_i = \begin{cases} 0 & \text{with probability } 1/2 \\ 1 & \text{with probability } 1/2. \end{cases}$$

Then, generate the data $X_1, \cdots, X_n$ according to

$$X_i \sim \begin{cases} N(\mu_0, \sigma_0^2) & \text{if } Y_i = 0 \\ N(\mu_1, \sigma_1^2) & \text{if } Y_i = 1. \end{cases}$$

Given such data $\{X_i\}$, we may wish to recover the true labels Y_i, which is a classification task.

Problem 3.a.

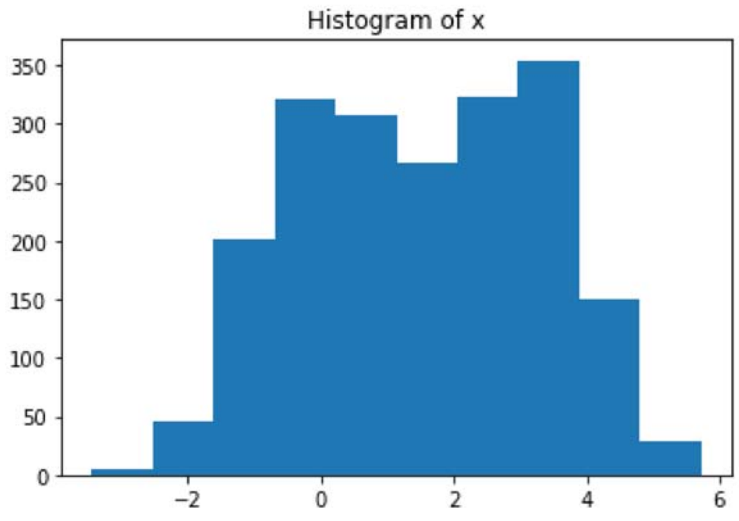
Suppose the parameters of the above model are: $\mu_0 = 0, \mu_1 = 3, \sigma_0^2 = \sigma_1^2 = 1$. Then the Bayes classifier is given by

$$f(X) = I\{X > 1.5\},$$

where I is the indicator function (take note of the 1.5, and it's relation with the means of the two Normal distributions).

Now generate $n = 2000$ data points from this dataset. Plot a histogram of the X's. This histogram is meant to be a sanity check for you; it should help you verify that you've generated the data properly.

In [106]:
```
y=np.random.binomial(1, 0.5, size=2000)
x=[i for i in range(1,2001)]
for k in range(1,2001) :
    if y[k-1] == 0:
        x[k-1] = np.asscalar(np.random.normal(0,1,1))
    else :
        x[k-1] = np.asscalar(np.random.normal(3,1,1))

plt.hist(x)
plt.title("Histogram of x");
```



Set aside a randomly-selected test set of $n/5$ points. We will refer to the rest of the data as the training data. Use the labels of the training data to calculate the group means. That is, calculate the mean value of all the X_i's in the training data with label $Y_i = 0$. Call this sample mean $\hat{\mu}_0$. Do the same thing to find $\hat{\mu}_1$. To be explicit, let $C_j = \{i : Y_i = j\}$, and define

$$\hat{\mu}_j = \frac{1}{|C_j|} \sum_{i \in C_j} X_i$$

Now classify the data in your test set. To do this, recall that your rule in Part a. depended on the true data means $\mu_0 = 0$ and $\mu_1 = 3$. Plug in the sample means $\hat{\mu}_j$ instead. Evaluate the estimator's performance using the loss:

$$\frac{1}{n} \sum_{i=1}^{n} 1\{\hat{Y}_i \neq Y_i\}$$

In [107]:
```
from random import sample
select=[i for i in range(1,2001)]
```

In [108]:
```
#randomly sample 400 values
l=sample(select, 400)
x=pd.DataFrame(x)
y=pd.DataFrame(y)
x_test=x.iloc[l,:]
y_test=y.iloc[l,:]
test= pd.concat([x_test, y_test], axis=1)
test.columns=(['x','y'])
x_train=x[~x.index.isin(l)]
y_train=y[~y.index.isin(l)]
train=pd.concat([x_train, y_train], axis=1)
train.columns=(['x','y'])
u0=np.mean(train['x'] [train['y']==0])
u1=np.mean(train['x'] [train['y']==1])
print(u0, u1)
```

```
-0.018745070893756783 2.9989780394541006
```

In [109]:
```
#calculate the boundary
boundary = (u0+u1)/2
boundary
```

Out[109]:
```
1.490116484280172
```

In [110]:
```
test.loc[test['x'] >boundary, 'pre_y'] =1
test.loc[test['x'] <=boundary, 'pre_y'] =0
test.loc[test['y'] !=test['pre_y'], "n"] =0
test.loc[test['y'] ==test['pre_y'], "n"] =1
rate=(400-sum(test["n"]))/400
rate
```

Out[110]:
```
0.06
```

The estimator's performance is pretty good given that the loss rate is very low (around 0.06).

Problem 3.b.

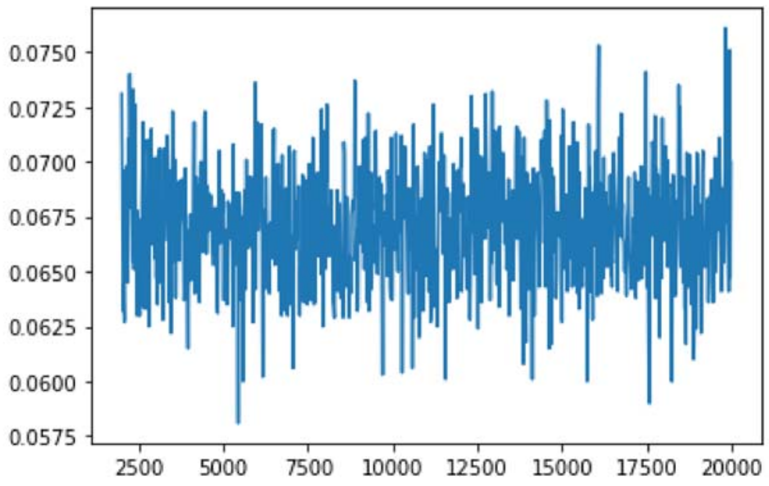
Now you train and evaluate classifiers for training sets of increasing size n, as specified below. For each n, you should

1. Generate a training set of size n from the above model (with the same parameters).
2. Generate a test set of size 10,000. Note that the test set itself will change on each round, but the size will always be the same: 10,000.
3. Compute the sample means on the training data.
4. Classify the test data as described in Part c.
5. Compute the error rate.

Plot the error rate as a function of n. Comment on your findings. What is happening to the error rate as n grows?

In [127]:
```
seq_n = np.arange(start = 2000, stop = 20000, step = 20)

error = [i for i in range(900)]
for i in range(1,901):
    n=seq_n[i-1]
    Y = list(np.random.binomial(1, 0.5, size=n))
    X = [i for i in range(n+1)]

    Ytest = list(np.random.binomial(1, 0.5, size=10000))
    Xtest = [i for i in range(10001)]

    for j in range(1,n+1) :
        if Y[j-1] == 0:
            X[j-1] = np.asscalar(np.random.normal(0,1,1))
        else:
            X[j-1] = np.asscalar(np.random.normal(3,1,1))
    Ytest = list(np.random.binomial(1, 0.5, size=10000))

    for k in range(1,10001) :
        if Ytest[k-1] == 0:
            Xtest[k-1] = np.asscalar(np.random.normal(0,1,1))
        else:
            Xtest[k-1] = np.asscalar(np.random.normal(3,1,1))

    x=pd.DataFrame(X)
    y=pd.DataFrame(Y)
    x_test=pd.DataFrame(Xtest)
    y_test=pd.DataFrame(Ytest)
    x_test=Xtest
    y_test=Ytest
    x_train = x
    y_train = y
    train = pd.concat([x_train,y_train],axis=1)
    train.columns=(['x','y'])
    u0=np.mean(train['x'] [train['y']==0])
    u1=np.mean(train['x'] [train['y']==1])
    #calculate the boundary
    boundary = (u0+u1)/2
    test.loc[test['x'] >boundary, 'pre_y'] =1
    test.loc[test['x'] <=boundary, 'pre_y'] =0
    test.loc[test['y'] !=test['pre_y'], "n"] =0
    test.loc[test['y'] ==test['pre_y'], "n"] =1
    error[i-1]=(400-sum(test["n"]))/400
```

In [130]:
```
plt.plot(seq_n, error);
```



From the plot we can see that, as n grows, error rates fluctuate around a value (near 0.0675).