Contents

0.1 S&DS 355/555: Assignment 1

0.1.1 NetID: sa857

Due: Sep 17, 2019 11:59pm

0.1.2 Imports and settings

In [317]:
```
%reset -f
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [318]:
```
# settings
pd.set_option('display.max_columns', 999)
```

1 Problem 1: Simple Linear Regression (25 points)

1.1 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 $i = 1, 2, \ldots, n$. 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

1.1.1 Answer

Usually, $RSS = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \ldots (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2$

Now, $RSS = (y_1 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_1 x_2)^2 + \ldots (y_n - \hat{\beta}_1 x_n)^2$

$RSS = \sum_{i=1}^n (y_i - \hat{\beta}_1 x_i)^2$

$\frac{\partial RSS}{\partial \beta_1} = \sum_{i=1}^n 2(y_i - \hat{\beta}_1 x_i)(-x_i) = 0$

$\sum_{i=1}^n (y_i - \hat{\beta}_1 x_i)(x_i) = 0$

$\sum_{i=1}^n x_i y_i = \sum_{i=1}^n \hat{\beta}_1 (x_i)^2$

$\frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n (x_i)^2} = \hat{\beta}_1$

1.2 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.

1.2.1 Answer

Contents

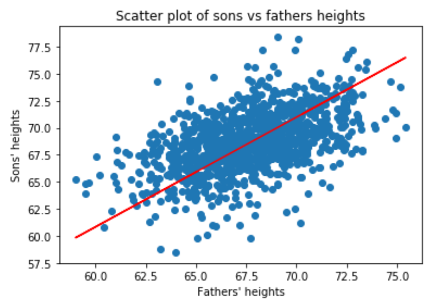
In [319]:
```
# read in data
fsData = pd.read_csv("fatherson.csv")

# scatter plot
plt.scatter(fsData.fheight, fsData.sheight)
plt.title("Scatter plot of sons vs fathers heights")
plt.xlabel("Fathers' heights")
plt.ylabel("Sons' heights")

# calculate beta
beta_1 = sum(fsData.fheight.values * fsData.sheight.values) / sum((fsData.fheight.values)**2)
print(f"Beta 1: {beta_1}")

# plot line
plt.plot(fsData.fheight, beta_1 * fsData.fheight, "r")
plt.show()
```

```
Beta 1: 1.0139079627134635
```



1.3 Problem 1.c:

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

1.3.1 Answer

Every unit increase in the height of a father leads to a 1.014 increase in the height of his son. In other words, the son's height is a constant multiple of the father's height.

1.4 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:

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

1.4.1 Answer

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$$

In [320]:
```
# calculate estimates
yBar = np.mean(fsData.sheight)
xBar = np.mean(fsData.fheight)
xSubXBar = fsData.fheight.values - xBar
ySubYBar = fsData.sheight.values - yBar
beta1 = sum(xSubXBar * ySubYBar) / sum(xSubXBar ** 2)
beta0 = yBar - (beta1 * xBar)
print(f"B1: {beta1}, B0: {beta0}")
```

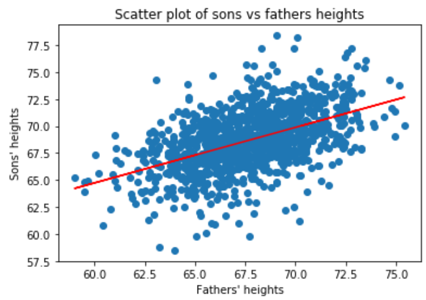
```
B1: 0.5140930386233066, B0: 33.886604354077996
```

The estimates are: B1: 0.5140930386233066, B0: 33.886604354077996

Contents

In [321]:
```
# plot to test
plt.scatter(fsData.fheight, fsData.sheight)
plt.title("Scatter plot of sons vs fathers heights")
plt.xlabel("Fathers' heights")
plt.ylabel("Sons' heights")
plt.plot(fsData.fheight, (beta1*fsData.fheight) + beta0, "r")
```

Out[321]: [<matplotlib.lines.Line2D at 0x12ee7c510>]



2 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.

2.1 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.

2.1.1 Import data

In [322]:
```
train = pd.read_csv("citibike_train.csv")
test = pd.read_csv("citibike_test.csv")
weather = pd.read_csv("weather.csv")
```

2.1.2 Merge data

In [323]:
```
# merge the weather by "date", and then drop "date" column
train = train.merge(weather, left_on="date", right_on="date").drop(columns=["date"])
test = test.merge(weather, left_on="date", right_on="date").drop(columns=["date"])
```

Contents

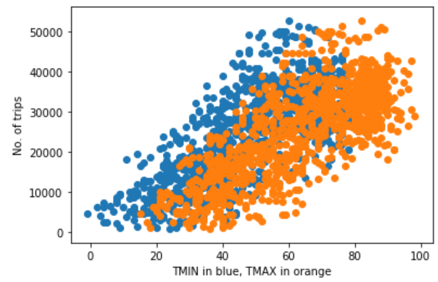
For the rest of this problem, you will train your models on the training data and evaluate them on the test data.

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.

2.1.3 Explore data

2.1.3.1 Scatterplot of TMIN vs TMAX

In [324]:
```
# make scatterplots
plt.scatter(train.TMIN, train.trips)
plt.scatter(train.TMAX, train.trips)
plt.xlabel("TMIN in blue, TMAX in orange")
plt.ylabel("No. of trips")
plt.show()
```



TMAX and TMIN are strongly correlated with each other. As temperatures rise, the number of train trips increase. TMAX and TMIN are just cutoffs for the same temperature rise phenomenon.

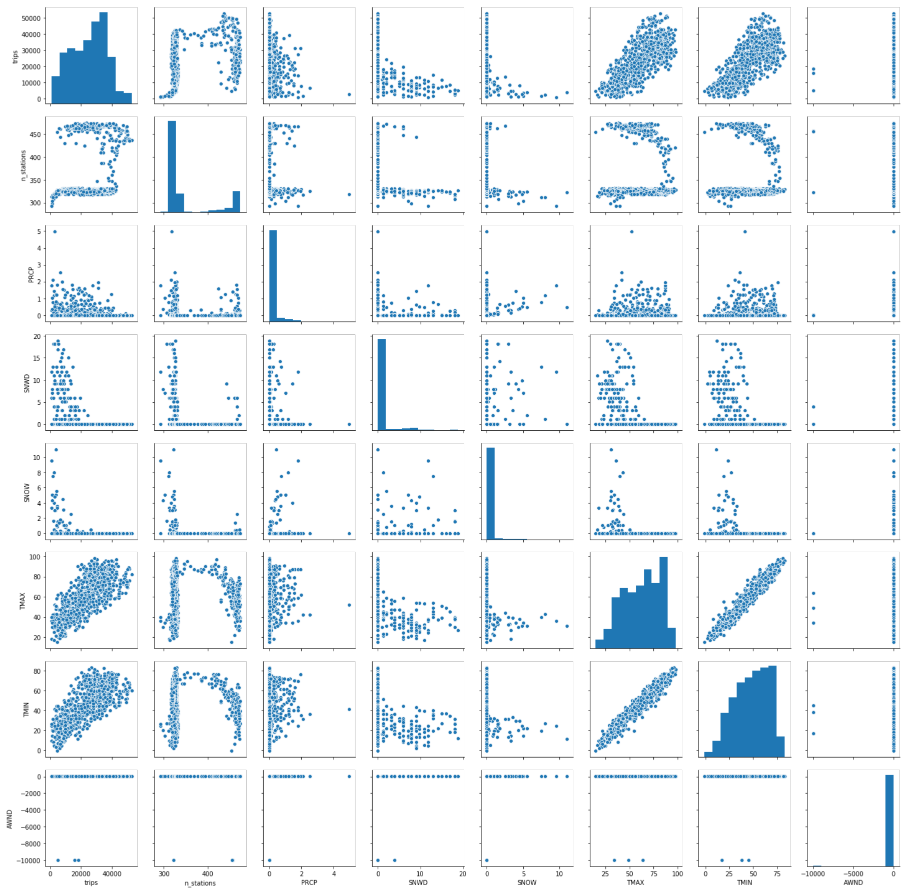
2.1.3.2 Pair plot of all numerical variables

The pair plot below shows strong correlations between "TMAX" and "TMIN", and between "trips" and each of "TMAX" and "TMIN". There is perhaps weak negative correlation between "trips" and "PRCP"

Contents

In [325]:
```
pairPlotDf = train.drop(columns=["holiday", "month", "dayofweek"])
sns.pairplot(pairPlotDf)
```

Out[325]: `<seaborn.axisgrid.PairGrid at 0x12b5d0a10>`



2.2 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.

2.2.1 Preprocessing

2.2.1.1 Categorical variables

Contents

In [326]:
```
# PROCESSING

# if statement to run the conversion once only
if 'holiday' in train.columns:
    train['holiday'] = train['holiday'].astype('category')
    test['holiday'] = test['holiday'].astype('category')

# put dummies
train = pd.get_dummies(train)
test = pd.get_dummies(test)

# see if test is missing columns
one = set(test.columns.values)
two = set(train.columns.values)
missing_columns = list(two.difference(one))

# fill in columns of zeros to get train and test to be same shape
for col in missing_columns:
    test[col] = 0
```

2.2.1.2 Reordering of columns

In [327]:
```
# sort to be the same column order
test = test.reindex(sorted(test.columns), axis=1)
train = train.reindex(sorted(train.columns), axis=1)
```

2.2.2 Model all covariates

Contents

In [328]:

```python
# fit a linear regression model using all covariates
import statsmodels.api as sm
covariates = train.drop(columns="trips")
X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
allCovariatesResult = model.fit()

print(allCovariatesResult.summary())
print(f"\n\nMSE: {allCovariatesResult.mse_resid}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.873
Model:                            OLS   Adj. R-squared:                  0.870
Method:                 Least Squares   F-statistic:                     268.5
Date:                Tue, 17 Sep 2019   Prob (F-statistic):               0.00
Time:                        22:08:45   Log-Likelihood:                -9747.3
No. Observations:                1001   AIC:                         1.955e+04
Df Residuals:                     975   BIC:                         1.967e+04
Df Model:                          25
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const           -1.255e+04    754.560    -16.636      0.000    -1.4e+04   -1.11e+04
AWND                0.7647      0.244      3.138      0.002       0.287       1.243
PRCP            -8214.3391    396.200    -20.733      0.000   -8991.842   -7436.836
SNOW                2.3998    188.650      0.013      0.990    -367.808     372.607
SNWD             -215.1752     62.855     -3.423      0.001    -338.521     -91.829
TMAX              352.8827     29.709     11.878      0.000     294.581     411.184
TMIN              -73.3230     32.807     -2.235      0.026    -137.704      -8.942
dayofweek_Fri    -309.4888    337.238     -0.918      0.359    -971.285     352.308
dayofweek_Mon   -1039.1399    341.749     -3.041      0.002   -1709.788    -368.492
dayofweek_Sat   -5414.7253    341.635    -15.849      0.000   -6085.150   -4744.301
dayofweek_Sun   -6381.7640    344.752    -18.511      0.000   -7058.306   -5705.222
dayofweek_Thurs   387.7770    336.822      1.151      0.250    -273.202    1048.757
dayofweek_Tues   -401.5068    343.769     -1.168      0.243   -1076.118     273.105
dayofweek_Wed     606.1591    341.981      1.772      0.077     -64.944    1277.263
holiday_False    -985.5413    468.510     -2.104      0.036   -1904.946     -66.137
holiday_True    -1.157e+04    624.110    -18.534      0.000   -1.28e+04   -1.03e+04
month_Apr       -2258.5999    524.291     -4.308      0.000   -3287.468   -1229.732
month_Aug        2209.4244    579.423      3.813      0.000    1072.365    3346.484
month_Dec       -5023.3082    482.377    -10.414      0.000   -5969.925   -4076.691
month_Feb       -7612.2728    699.954    -10.875      0.000   -8985.863   -6238.682
month_Jan       -7134.2280    606.453    -11.764      0.000   -8324.331   -5944.125
month_Jul         461.3105    617.169      0.747      0.455    -749.821    1672.442
month_Jun        2268.7701    612.624      3.703      0.000    1066.557    3470.983
month_Mar       -5783.7715    508.398    -11.376      0.000   -6781.451   -4786.091
month_May        1488.4128    565.880      2.630      0.009     377.930    2598.895
month_Nov        -188.1675    454.469     -0.414      0.679   -1080.017     703.682
month_Oct        4373.2223    434.020     10.076      0.000    3521.501    5224.944
month_Sep        4646.5193    512.235      9.071      0.000    3641.310    5651.728
n_stations         68.9045      2.888     23.855      0.000      63.236      74.573
==============================================================================
Omnibus:                       56.833   Durbin-Watson:                   1.150
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              123.489
Skew:                          -0.345   Prob(JB):                     1.53e-27
Kurtosis:                       4.577   Cond. No.                     1.06e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.68e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.


MSE: 17254735.842562452
```

2.2.3 Backward selection

2.2.3.1 Useful functions

In [329]:

```python
# useful functions
def whichColumnShouldIDrop(modelResult):
    pSeries = modelResult.pvalues
    print(f"There were {len(pSeries) - 1} variables in the model.")
    highestPvalue = max(pSeries)
    varName = pSeries[pSeries == highestPvalue].keys()[0]
    print(f"The one with the highest pvalue of '{highestPvalue}' was '{varName}'")
    print(f"The column to drop should therefore be {varName}")
```

In [330]:

```python
def allColumnsWithXInName(x):
    allCols = list(train.columns)
    return list(filter(lambda col: x in col, allCols))
```

In [331]:

```python
def calculateTestMSE(modelResult, y_test, X_test):
    y_pred = modelResult.predict(X_test)
    subs = [(y_test.trips[i] - y_pred[i]) ** 2 for i in range(len(y_test))]
    p = len(modelResult.pvalues) - 1
    return sum(subs) / (len(y_test) - p)
```

Contents

2.2.3.2 Best model p == 5

Five rounds of selection

In [332]:
```
whichColumnShouldIDrop(allCovariatesResult)
```

```
There were 28 variables in the model.
The one with the highest pvalue of '0.989853175886976' was 'SNOW'
The column to drop should therefore be SNOW
```

In [333]:
```
covariates = train.drop(columns=["trips", "SNOW"])
X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
firstSelectionResult = model.fit()
print(firstSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.873
Model:                            OLS   Adj. R-squared:                  0.870
Method:                 Least Squares   F-statistic:                     280.0
Date:                Tue, 17 Sep 2019   Prob (F-statistic):               0.00
Time:                        22:08:46   Log-Likelihood:                -9747.3
No. Observations:                1001   AIC:                         1.954e+04
Df Residuals:                     976   BIC:                         1.967e+04
Df Model:                          24
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const             -1.255e+04    750.890    -16.716      0.000    -1.4e+04   -1.11e+04
AWND                  0.7647      0.244      3.140      0.002       0.287       1.243
PRCP              -8213.1439    384.700    -21.349      0.000   -8968.079   -7458.209
SNWD               -215.0749     62.326     -3.451      0.001    -337.383     -92.767
TMAX                352.8805     29.693     11.884      0.000     294.610     411.151
TMIN                -73.3368     32.773     -2.238      0.025    -137.650      -9.024
dayofweek_Fri      -309.3256    336.822     -0.918      0.359    -970.304     351.653
dayofweek_Mon     -1038.9093    341.093     -3.046      0.002   -1708.269    -369.549
dayofweek_Sat     -5414.6308    341.379    -15.861      0.000   -6084.553   -4744.709
dayofweek_Sun     -6381.7238    344.561    -18.521      0.000   -7057.890   -5705.558
dayofweek_Thurs     387.9528    336.366      1.153      0.249    -272.131    1048.037
dayofweek_Tues     -401.2177    342.841     -1.170      0.242   -1074.008     271.573
dayofweek_Wed       606.0606    341.718      1.774      0.076     -64.526    1276.647
holiday_False      -984.9759    466.158     -2.113      0.035   -1899.764     -70.188
holiday_True      -1.157e+04    623.253    -18.559      0.000   -1.28e+04   -1.03e+04
month_Apr         -2258.8742    523.579     -4.314      0.000   -3286.343   -1231.405
month_Aug          2209.6592    578.832      3.817      0.000    1073.761    3345.558
month_Dec         -5023.3601    482.113    -10.419      0.000   -5969.457   -4077.263
month_Feb         -7612.0582    699.392    -10.884      0.000   -8984.544   -6239.572
month_Jan         -7133.8050    605.230    -11.787      0.000   -8321.507   -5946.103
month_Jul           461.5060    616.661      0.748      0.454    -748.628    1671.640
month_Jun          2268.8147    612.300      3.705      0.000    1067.238    3470.391
month_Mar         -5783.7481    508.134    -11.382      0.000   -6780.909   -4786.587
month_May          1488.4185    565.590      2.632      0.009     378.507    2598.330
month_Nov          -188.3026    454.112     -0.415      0.678   -1079.450     702.845
month_Oct          4373.2353    433.797     10.081      0.000    3521.954    5224.517
month_Sep          4646.7208    511.727      9.080      0.000    3642.508    5650.933
n_stations           68.9022      2.881     23.914      0.000      63.248      74.556
==============================================================================
Omnibus:                       56.836   Durbin-Watson:                   1.150
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              123.434
Skew:                          -0.345   Prob(JB):                     1.57e-27
Kurtosis:                       4.576   Cond. No.                     1.06e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.68e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [334]:
```
whichColumnShouldIDrop(firstSelectionResult)
```

```
There were 27 variables in the model.
The one with the highest pvalue of '0.6784808069754316' was 'month_Nov'
The column to drop should therefore be month_Nov
```

Contents

In [335]:
```python
# remove all the month variables
covariates = train.drop(columns=["trips", "SNOW"] + allColumnsWithXInName("month"))
X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
secondSelectionResult = model.fit()
print(secondSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.820
Model:                            OLS   Adj. R-squared:                  0.817
Method:                 Least Squares   F-statistic:                     345.3
Date:                Tue, 17 Sep 2019   Prob (F-statistic):               0.00
Time:                        22:08:46   Log-Likelihood:                -9923.2
No. Observations:                1001   AIC:                         1.987e+04
Df Residuals:                     987   BIC:                         1.994e+04
Df Model:                          13
Covariance Type:            nonrobust
=====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const            -1.674e+04    846.118    -19.788      0.000   -1.84e+04   -1.51e+04
AWND                 0.6034      0.286      2.106      0.035       0.041       1.166
PRCP             -8684.1951    453.859    -19.134      0.000   -9574.835   -7793.555
SNWD              -411.2585     61.553     -6.681      0.000    -532.048    -290.469
TMAX               408.0427     32.228     12.661      0.000     344.799     471.286
TMIN                37.2733     34.333      1.086      0.278     -30.100     104.646
dayofweek_Fri     -770.4382    397.535     -1.938      0.053   -1550.548       9.672
dayofweek_Mon    -1664.1143    402.489     -4.135      0.000   -2453.948    -874.281
dayofweek_Sat    -5911.2112    403.187    -14.661      0.000   -6702.413   -5120.010
dayofweek_Sun    -7139.6523    404.772    -17.639      0.000   -7933.965   -6345.340
dayofweek_Thurs   -164.0064    396.880     -0.413      0.680    -942.832     614.819
dayofweek_Tues   -1060.1334    403.953     -2.624      0.009   -1852.839    -267.428
dayofweek_Wed      -33.6629    402.266     -0.084      0.933    -823.057     755.732
holiday_False    -2812.0451    537.708     -5.230      0.000   -3867.227   -1756.863
holiday_True     -1.393e+04    717.202    -19.424      0.000   -1.53e+04   -1.25e+04
n_stations          60.2811      2.916     20.672      0.000      54.559      66.003
==============================================================================
Omnibus:                       23.647   Durbin-Watson:                   0.878
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               44.622
Skew:                          -0.120   Prob(JB):                     2.04e-10
Kurtosis:                       4.006   Cond. No.                     6.76e+18
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.57e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [336]:
```python
whichColumnShouldIDrop(secondSelectionResult)
```

```
There were 15 variables in the model.
The one with the highest pvalue of '0.9333252469205151' was 'dayofweek_Wed'
The column to drop should therefore be dayofweek_Wed
```

Contents

In [337]:

```
# remove all the day variables
covariates = train.drop(
    columns=["trips", "SNOW"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek"))

X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
thirdSelectionResult = model.fit()
print(thirdSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.767
Model:                            OLS   Adj. R-squared:                  0.765
Method:                 Least Squares   F-statistic:                     466.3
Date:                Tue, 17 Sep 2019   Prob (F-statistic):          1.21e-308
Time:                        22:08:46   Log-Likelihood:                -10052.
No. Observations:                1001   AIC:                         2.012e+04
Df Residuals:                     993   BIC:                         2.016e+04
Df Model:                           7
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const         -1.808e+04   1049.460    -17.229      0.000   -2.01e+04    -1.6e+04
AWND              1.0357      0.323      3.202      0.001       0.401       1.671
PRCP          -8328.7629    512.770    -16.243      0.000   -9335.001   -7322.525
SNWD           -406.4006     69.802     -5.822      0.000    -543.376    -269.425
TMAX            395.5854     36.289     10.901      0.000     324.373     466.797
TMIN             50.9168     38.676      1.316      0.188     -24.980     126.813
holiday_False -4239.3928    631.843     -6.710      0.000   -5479.294   -2999.491
holiday_True   -1.384e+04   842.140    -16.436      0.000   -1.55e+04   -1.22e+04
n_stations       61.4942      3.306     18.599      0.000      55.006      67.982
==============================================================================
Omnibus:                        4.582   Durbin-Watson:                   0.921
Prob(Omnibus):                  0.101   Jarque-Bera (JB):                5.352
Skew:                          -0.048   Prob(JB):                       0.0689
Kurtosis:                       3.345   Cond. No.                     1.10e+18
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.48e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [338]:

```
whichColumnShouldIDrop(thirdSelectionResult)
```

```
There were 8 variables in the model.
The one with the highest pvalue of '0.18831446195624318' was 'TMIN'
The column to drop should therefore be TMIN
```

Contents

In [339]:
```
covariates = train.drop(
    columns=["trips", "SNOW", "TMIN"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek"))

X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
fourthSelectionResult = model.fit()
print(fourthSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.766
Model:                            OLS   Adj. R-squared:                  0.765
Method:                 Least Squares   F-statistic:                     543.3
Date:                Tue, 17 Sep 2019   Prob (F-statistic):         1.18e-309
Time:                        22:08:46   Log-Likelihood:                -10053.
No. Observations:                1001   AIC:                         2.012e+04
Df Residuals:                     994   BIC:                         2.015e+04
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const         -1.83e+04   1035.981    -17.669      0.000   -2.03e+04   -1.63e+04
AWND             1.0467      0.324      3.235      0.001       0.412       1.682
PRCP         -8280.9945    511.674    -16.184      0.000   -9285.079   -7276.910
SNWD          -417.5572     69.311     -6.024      0.000    -553.569    -281.545
TMAX           441.3231     10.486     42.088      0.000     420.747     461.900
holiday_False -4374.5435   623.678     -7.014      0.000   -5598.419   -3150.668
holiday_True  -1.393e+04   839.753    -16.589      0.000   -1.56e+04   -1.23e+04
n_stations      61.3480      3.306     18.559      0.000      54.861      67.835
==============================================================================
Omnibus:                        3.885   Durbin-Watson:                   0.925
Prob(Omnibus):                  0.143   Jarque-Bera (JB):                4.422
Skew:                          -0.037   Prob(JB):                        0.110
Kurtosis:                       3.317   Cond. No.                     7.95e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.76e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [340]:
```
whichColumnShouldIDrop(fourthSelectionResult)
```

```
There were 7 variables in the model.
The one with the highest pvalue of '0.0012545608107361673' was 'AWND'
The column to drop should therefore be AWND
```

In [341]:
```
covariates = train.drop(
    columns=["trips", "SNOW", "TMIN", "AWND"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek"))

X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
fifthSelectionResult = model.fit()
print(fifthSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.764
Model:                            OLS   Adj. R-squared:                  0.763
Method:                 Least Squares   F-statistic:                     643.7
Date:                Tue, 17 Sep 2019   Prob (F-statistic):         8.01e-309
Time:                        22:08:46   Log-Likelihood:                -10058.
No. Observations:                1001   AIC:                         2.013e+04
Df Residuals:                     995   BIC:                         2.016e+04
Df Model:                           5
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const        -1.822e+04   1040.561    -17.509      0.000   -2.03e+04   -1.62e+04
PRCP         -8252.3813    514.026    -16.054      0.000   -9261.080   -7243.683
SNWD          -417.8816     69.640     -6.001      0.000    -554.539    -281.224
TMAX           442.2684     10.531     41.995      0.000     421.602     462.935
holiday_False -4354.2871   626.606     -6.949      0.000   -5583.909   -3124.666
holiday_True  -1.387e+04   843.497    -16.438      0.000   -1.55e+04   -1.22e+04
n_stations      60.7974      3.317     18.329      0.000      54.288      67.306
==============================================================================
Omnibus:                        4.606   Durbin-Watson:                   0.924
Prob(Omnibus):                  0.100   Jarque-Bera (JB):                5.349
Skew:                          -0.053   Prob(JB):                       0.0690
Kurtosis:                       3.342   Cond. No.                     7.58e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.31e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

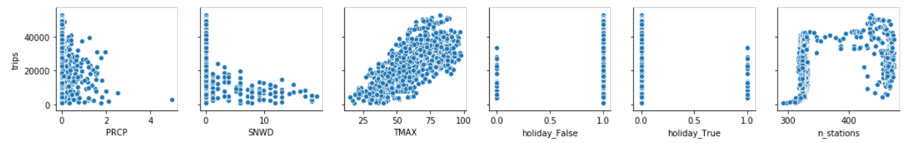
Contents

At the end of five rounds of selection, the model with the following variables was chosen: `[PRCP, SNWD, TMAX, holiday, n_stations]`

Pair plot

In [342]:
```
sns.pairplot(train.drop(
    columns=["SNOW", "TMIN", "AWND"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek")),
    y_vars=['trips'],
    x_vars=["PRCP", "SNWD", "TMAX", "holiday_False", "holiday_True", "n_stations"])
```

Out[342]: `<seaborn.axisgrid.PairGrid at 0x12c072150>`



Predictions

R^2 ***and MSE***

In [343]:
```
fifthSelectionResult.mse_resid
```

Out[343]: `31482485.09965653`

In [344]:
```
X_test = train.drop(
    columns=["SNOW", "TMIN", "AWND"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek"))

y_test = train[["trips"]]

calculateTestMSE(fifthSelectionResult, y_test, X_test)
```

Out[344]: `12795186806733.623`

Training data

R^2 according to the model summary was: 0.764

MSE was: 31482485.09965653

Test data

R^2: According to the rubric and Parker's answer on Piazza, this is a meaningless statistic so I have not computed it.

MSE was: 12795186806733.623

2.2.3.3 Best model p == 4

Another round of selection

In [345]:
```
whichColumnShouldIDrop(fifthSelectionResult)
```

```
There were 6 variables in the model.
The one with the highest pvalue of '2.749583091977293e-09' was 'SNWD'
The column to drop should therefore be SNWD
```

Contents

In [346]:
```
covariates = train.drop(
    columns=["trips", "SNOW", "TMIN", "AWND", "SNWD"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek"))

X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
sixthSelectionResult = model.fit()
print(sixthSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.755
Model:                            OLS   Adj. R-squared:                  0.754
Method:                 Least Squares   F-statistic:                     768.6
Date:                Tue, 17 Sep 2019   Prob (F-statistic):          1.28e-302
Time:                        22:08:47   Log-Likelihood:                -10076.
No. Observations:                1001   AIC:                         2.016e+04
Df Residuals:                     996   BIC:                         2.019e+04
Df Model:                           4
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const         -2.065e+04    975.130    -21.178      0.000   -2.26e+04   -1.87e+04
PRCP          -8236.0150    522.974    -15.748      0.000   -9262.272   -7209.758
TMAX            471.0558      9.539     49.385      0.000     452.338     489.774
holiday_False -5666.5170    597.439     -9.485      0.000   -6838.900   -4494.134
holiday_True  -1.498e+04    836.951    -17.904      0.000   -1.66e+04   -1.33e+04
n_stations       65.2022      3.291     19.812      0.000      58.744      71.660
==============================================================================
Omnibus:                        3.845   Durbin-Watson:                   0.916
Prob(Omnibus):                  0.146   Jarque-Bera (JB):                4.236
Skew:                          -0.058   Prob(JB):                        0.120
Kurtosis:                       3.297   Cond. No.                     8.09e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.02e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

R^2 and MSE

In [347]:
```
sixthSelectionResult.mse_resid
```

Out[347]: `32589036.105076745`

In [348]:
```
X_test = train.drop(
    columns=["SNOW", "TMIN", "AWND", "SNWD"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek"))

y_test = train[["trips"]]

calculateTestMSE(sixthSelectionResult, y_test, X_test)
```

Out[348]: `18657357119633.902`

Training data

R^2 according to the model summary was: 0.755

MSE was: 32589036.105076745

Test data

MSE was: 18657357119633.902

2.2.3.4 Best model p == 3

Another selection

In [349]:
```
whichColumnShouldIDrop(sixthSelectionResult)
```

```
There were 5 variables in the model.
The one with the highest pvalue of '1.7255263964426543e-20' was 'holiday_False'
The column to drop should therefore be holiday_False
```

Contents

In [350]:

```
covariates = train.drop(
    columns=["trips", "SNOW", "TMIN", "AWND", "SNWD"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek") +
    allColumnsWithXInName("holiday"))

X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
seventhSelectionResult = model.fit()
print(seventhSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.737
Model:                            OLS   Adj. R-squared:                  0.736
Method:                 Least Squares   F-statistic:                     931.2
Date:                Tue, 17 Sep 2019   Prob (F-statistic):          1.57e-288
Time:                        22:08:47   Log-Likelihood:                -10112.
No. Observations:                1001   AIC:                         2.023e+04
Df Residuals:                     997   BIC:                         2.025e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2.685e+04   1432.620    -18.743      0.000   -2.97e+04    -2.4e+04
PRCP       -8085.1685    541.629    -14.927      0.000   -9148.033   -7022.304
TMAX         476.5096      9.863     48.315      0.000     457.156     495.863
n_stations    64.9314      3.410     19.040      0.000      58.239      71.623
==============================================================================
Omnibus:                       10.038   Durbin-Watson:                   0.832
Prob(Omnibus):                  0.007   Jarque-Bera (JB):               11.724
Skew:                          -0.157   Prob(JB):                      0.00285
Kurtosis:                       3.427   Cond. No.                     2.79e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.79e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

R^2 and MSE

In [351]:

```
seventhSelectionResult.mse_resid
```

Out[351]: 34994570.75404534

In [352]:

```
X_test = train.drop(
    columns=["SNOW", "TMIN", "AWND", "SNWD"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek") +
    allColumnsWithXInName("holiday"))

y_test = train[["trips"]]

calculateTestMSE(seventhSelectionResult, y_test, X_test)
```

Out[352]: 1998366500725.9263

Training data

R^2 according to the model summary was: 0.737

MSE was: 34994570.75404534

Test data

MSE was: 1998366500725.9263

2.2.3.5 Best model p == 2

Another selection

In [353]:

```
whichColumnShouldIDrop(seventhSelectionResult)
```

```
There were 3 variables in the model.
The one with the highest pvalue of '1.2561082989927261e-45' was 'PRCP'
The column to drop should therefore be PRCP
```

Contents

In [354]:
```
covariates = train.drop(
    columns=["trips", "SNOW", "TMIN", "AWND", "SNWD", "PRCP"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek") +
    allColumnsWithXInName("holiday"))

X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
eighthSelectionResult = model.fit()
print(eighthSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.678
Model:                            OLS   Adj. R-squared:                  0.678
Method:                 Least Squares   F-statistic:                     1052.
Date:                Tue, 17 Sep 2019   Prob (F-statistic):          1.93e-246
Time:                        22:08:47   Log-Likelihood:                -10213.
No. Observations:                1001   AIC:                         2.043e+04
Df Residuals:                     998   BIC:                         2.045e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2.874e+04   1577.678    -18.215      0.000   -3.18e+04   -2.56e+04
TMAX         479.3314     10.902     43.968      0.000     457.938     500.724
n_stations    67.0363      3.767     17.796      0.000      59.644      74.428
==============================================================================
Omnibus:                       18.280   Durbin-Watson:                   0.958
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               18.879
Skew:                          -0.335   Prob(JB):                     7.95e-05
Kurtosis:                       3.054   Cond. No.                     2.78e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.78e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

R^2 and MSE

In [355]:
```
eighthSelectionResult.mse_resid
```

Out[355]: 42772978.27762087

In [356]:
```
X_test = train.drop(
    columns=["SNOW", "TMIN", "AWND", "SNWD", "PRCP"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek") +
    allColumnsWithXInName("holiday"))

y_test = train[["trips"]]

calculateTestMSE(eighthSelectionResult, y_test, X_test)
```

Out[356]: 260141784482.93845

Training data

R^2 according to the model summary was: 0.678

MSE was: 42772978.27762087

Test data

MSE was: 260141784482.93845

2.2.3.6 Best model p == 1

Another selection

In [357]:
```
whichColumnShouldIDrop(eighthSelectionResult)
```

```
There were 2 variables in the model.
The one with the highest pvalue of '9.615354198374857e-62' was 'n_stations'
The column to drop should therefore be n_stations
```

Contents

In [358]:

```
covariates = train.drop(
    columns=["trips", "SNOW", "TMIN", "AWND", "SNWD", "PRCP", "n_stations"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek") +
    allColumnsWithXInName("holiday"))

X = sm.add_constant(covariates)
y = train[["trips"]]
model = sm.OLS(y, X)
ninthSelectionResult = model.fit()
print(ninthSelectionResult.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  trips   R-squared:                       0.576
Model:                            OLS   Adj. R-squared:                  0.576
Method:                 Least Squares   F-statistic:                     1358.
Date:                Tue, 17 Sep 2019   Prob (F-statistic):          2.22e-188
Time:                        22:08:48   Log-Likelihood:                -10351.
No. Observations:                1001   AIC:                         2.071e+04
Df Residuals:                     999   BIC:                         2.072e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -3659.5444    813.732     -4.497      0.000   -5256.364   -2062.725
TMAX         458.0092     12.430     36.846      0.000     433.616     482.402
==============================================================================
Omnibus:                       12.667   Durbin-Watson:                   0.735
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               12.782
Skew:                           0.263   Prob(JB):                      0.00168
Kurtosis:                       3.170   Cond. No.                         225.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

R^2 and MSE

In [359]:

```
ninthSelectionResult.mse_resid
```

Out[359]: 56289579.39584944

In [360]:

```
X_test = train.drop(
    columns=["SNOW", "TMIN", "AWND", "SNWD", "PRCP", "n_stations"] +
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek") +
    allColumnsWithXInName("holiday"))

y_test = train[["trips"]]

calculateTestMSE(ninthSelectionResult, y_test, X_test)
```

Out[360]: 152850992425903.4

Training data

R^2 according to the model summary was: 0.576

MSE was: 56289579.39584944

Test data

MSE was: 152850992425903.4

2.3 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.

2.3.1 Compute binary Y, find median, drop "trips"

Contents

In [361]:
```
median = np.median(np.concatenate((test.trips, train.trips)))

knnYTrain = [1 if trips > median else 0 for trips in train.trips]
knnXTrain = train.drop(columns=["trips"])

knnYTest = [1 if trips > median else 0 for trips in test.trips]
knnXTest = test.drop(columns=["trips"])
```

2.3.2 Drop categorical variables

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 [362]:
```
knnXTrain = knnXTrain.drop(columns=(
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek") +
    allColumnsWithXInName("holiday")))

knnXTest = knnXTest.drop(columns=(
    allColumnsWithXInName("month") +
    allColumnsWithXInName("dayofweek") +
    allColumnsWithXInName("holiday")))
```

2.3.3 Feature scaling: Standardization

In [363]:
```
from sklearn.preprocessing import StandardScaler
scX = StandardScaler()
```

$X = \frac{Xtest - \mu train}{\sigma train}$

The reason we just `transform` test data instead of `fit_transform` is so that we can have the same train parameters apply to the test data (the `scX` maintains some internal state).

In [364]:
```
# why do fit_transform on train and just transform on test?
knnXTrain = scX.fit_transform(knnXTrain)
knnXTest = scX.transform(knnXTest)
```

2.3.4 Run the KNN

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.

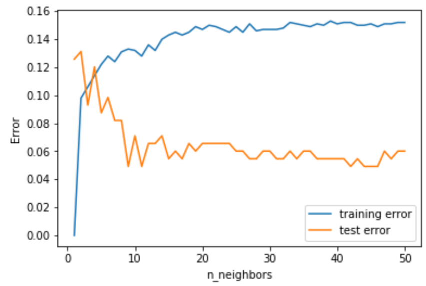
In [365]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
```

In [366]:
```
neighbors_settings = range(1, 51)
training_accuracy = []
test_accuracy = []
for n_neighbors in neighbors_settings:
    # instantiate a classifier
    classifier = KNeighborsClassifier(n_neighbors=n_neighbors)
    # fit the classifier to the training data
    classifier.fit(knnXTrain, knnYTrain)
    # find the training set accuracy to the actual labels (did it learn the model well?)
    # and find the test set accuracy to the actual labels (did it generalize well?)
    training_accuracy.append(1 - classifier.score(knnXTrain, knnYTrain))
    test_accuracy.append(1 - classifier.score(knnXTest, knnYTest))
```

Contents

In [367]:
```
plt.plot(neighbors_settings, training_accuracy, label="training error")
plt.plot(neighbors_settings, test_accuracy, label="test error")
plt.ylabel("Error")
plt.xlabel("n_neighbors")
plt.legend()
```

Out[367]: <matplotlib.legend.Legend at 0x11bf26f50>



2.3.5 Best model

The best model is the one that minimizes the test error/misclassification, around k = 40.

We might get over-fitting if we just try to minimize training misclassification, because that corresponds to k = 1.

3 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.

3.1 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.

3.1.1 Useful functions and imports

In [368]:
```
import random
from sklearn.model_selection import train_test_split
```

3.1.2 Generate Ys

In [369]:
```
gaussian_y = [random.choice([0, 1]) for x in range(0, 2000)]
```

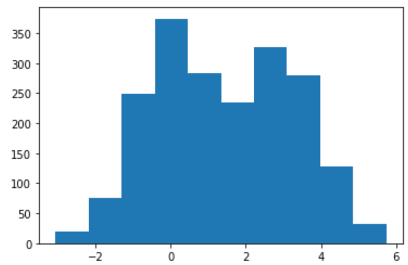
3.1.3 Generate Xs

In [370]:
```
gaussian_x = [np.random.normal(loc=0, scale=1)
              if gaussian_y[x] == 0 else
              np.random.normal(loc=3, scale=1)
              for x in range(0, 2000)]
```

Contents

In [371]:
```
# sanity check
plt.hist(gaussian_x)
```

Out[371]:
```
(array([ 20.,  75., 248., 374., 283., 234., 327., 280., 127.,  32.]),
 array([-3.0577953 , -2.17814123, -1.29848716, -0.4188331 ,  0.46082097,
         1.34047504,  2.2201291 ,  3.09978317,  3.97943724,  4.8590913 ,
         5.73874537]),
 <a list of 10 Patch objects>)
```



3.1.4 Train-test split

Set aside a randomly-selected test set of $n/5$ points.

In [372]:
```
X_train, X_test, y_train, y_test = train_test_split(gaussian_x, gaussian_y, test_size=0.2)
```

3.1.5 Calculate group means

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

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

In [373]:
```
def calculateMeanOfAllXWithLabel(label, X_train, y_train):
    X_train_with_label = []
    for index, y in enumerate(y_train):
        if y == label:
            X_train_with_label.append(X_train[index])
    return np.mean(X_train_with_label)
```

In [374]:
```
# calculate u0 and u1
u_0_hat = calculateMeanOfAllXWithLabel(0, X_train, y_train)
u_1_hat = calculateMeanOfAllXWithLabel(1, X_train, y_train)
```

3.1.6 Classify using indicator function

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 [375]:
```
def indicatorFunction(x, mean1, mean2):
    if x > ((mean1 + mean2) / 2):
        return 1
    return 0
```

In [376]:
```
y_pred = [indicatorFunction(x, u_0_hat, u_1_hat) for x in X_test]
```

3.1.7 Compute error rate

In [377]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [378]:
```
misclassified_count = cm[1][0] + cm[0][1]
error_rate = misclassified_count / len(y_pred)
print(f"Error rate is: {error_rate}")
```

```
Error rate is: 0.0875
```

Contents

3.2 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?

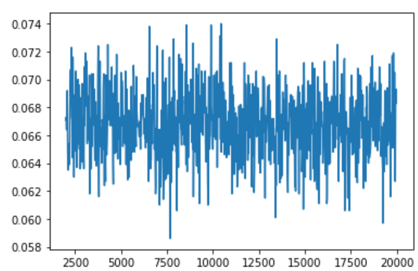
In [379]:
```
## -- please write your answer here. -- ##
seq_n = np.arange(start = 2000, stop = 20000, step = 20)
error_rates = []
```

In [380]:
```
for n_count in seq_n:
    # create labels according to n_count + 10000 == total count
    gaussian_y = [random.choice([0, 1]) for x in range(0, n_count + 10000)]
    # create feature data
    gaussian_x = [np.random.normal(loc=0, scale=1)
                  if gaussian_y[x] == 0 else
                  np.random.normal(loc=3, scale=1)
                  for x in range(0, n_count + 10000)]
    # train test split
    X_train, X_test, y_train, y_test = train_test_split(gaussian_x, gaussian_y, test_size=10000)
    # calculate means
    u_0_hat = calculateMeanOfAllXWithLabel(0, X_train, y_train)
    u_1_hat = calculateMeanOfAllXWithLabel(1, X_train, y_train)
    # make the predictions
    y_pred = [indicatorFunction(x, u_0_hat, u_1_hat) for x in X_test]
    # compute error rate
    cm = confusion_matrix(y_test, y_pred)
    misclassified_count = cm[1][0] + cm[0][1]
    error_rate = misclassified_count / len(y_pred)
    error_rates.append(error_rate)
```

3.2.1 Plot error rates

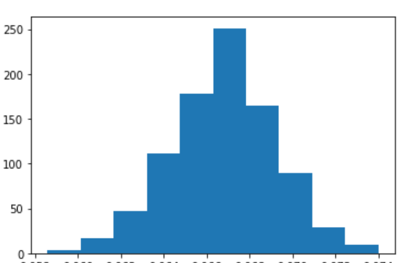
In [381]:
```
plt.plot(seq_n, error_rates)
```

Out[381]:
```
[<matplotlib.lines.Line2D at 0x12ff76d50>]
```



In [382]:
```
plt.hist(error_rates)
```

Out[382]:
```
(array([  3.,  17.,  47., 111., 178., 251., 164.,  90.,  29.,  10.]),
 array([0.0586 , 0.06014, 0.06168, 0.06322, 0.06476, 0.0663 , 0.06784,
        0.06938, 0.07092, 0.07246, 0.074  ]),
 <a list of 10 Patch objects>)
```



3.2.2 Comments

Error rate is fairly constant, and normally distributed around 0.067.