Lab 2 - ML Programming

November 19, 2021

1 EXERCISE 1

1.1 Part A: Interesting Stats

[5]:
```
## First we want to import the necessary packages as well as the data we will␣
↪be working on
## We can look at the different variables present in the csv files

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
## the following lets me see the output of all my code, not just the last result
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

train = pd.read_csv('train.csv')
store = pd.read_csv('store.csv')
```

```
C:\Users\Nikita\anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3165: DtypeWarning: Columns (7) have
mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

[6]:
```
## Find the store that has the maximum sale recorded. Print the store id, date␣
↪and the sales on that day

## First I obtained the row index of the maximum sale recorded
maxsaleindex = np.flatnonzero(train.Sales == np.max(train.Sales))

## Then I used the row index from above in the loc function to return the info␣
↪we were looking for
maxsaleinfo = train.loc[maxsaleindex, ['Store','Date','Sales']]
maxsaleinfo
```

[6]:

	Store	Date	Sales
44393	909	2015-06-22	41551

[7]:
```python
## Find the store(s) that has/ve the least possible and maximum possible␣
↪competition distance(s).

## We can implement the same method we used above for max sales
maxdistindex = np.flatnonzero(store.CompetitionDistance == np.max(store.
↪CompetitionDistance))
maxdistindex = maxdistindex.tolist()
maxdistinfo = store.loc[maxdistindex, ['Store','CompetitionDistance']]
print("Stores with max. dist.:")
maxdistinfo

mindistindex = np.flatnonzero(store.CompetitionDistance == np.min(store.
↪CompetitionDistance))
mindistindex = mindistindex.tolist()
mindistinfo = store.loc[mindistindex, ['Store','CompetitionDistance']]
print("Stores with min. dist.:")
mindistinfo
```

Stores with max. dist.:

[7]:

	Store	CompetitionDistance
452	453	75860.0

Stores with min. dist.:

[7]:

	Store	CompetitionDistance
515	516	20.0

[109]:
```python
## What is the maximum timeline a store has ran a "Promo" for?
## Which store was that, and what dates did the promotion cover?

uptrain = train.copy()
uptrain[["Day", "Month", "Year"]] = uptrain["Date"].str.split("-", expand =␣
↪True)
uptrain
```

[109]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	\
0	1	5	2015-07-31	5263	555	1	1	
1	2	5	2015-07-31	6064	625	1	1	
2	3	5	2015-07-31	8314	821	1	1	
3	4	5	2015-07-31	13995	1498	1	1	
4	5	5	2015-07-31	4822	559	1	1	
…	…	…	…	…	…	…	…	
1017204	1111	2	2013-01-01	0	0	0	0	
1017205	1112	2	2013-01-01	0	0	0	0	
1017206	1113	2	2013-01-01	0	0	0	0	

```
1017207    1114           2  2013-01-01      0          0     0      0
1017208    1115           2  2013-01-01      0          0     0      0

        StateHoliday  SchoolHoliday   Day Month  Year
0                  0              1  2015    07    31
1                  0              1  2015    07    31
2                  0              1  2015    07    31
3                  0              1  2015    07    31
4                  0              1  2015    07    31
…              …            …   …   …  …
1017204            a              1  2013    01    01
1017205            a              1  2013    01    01
1017206            a              1  2013    01    01
1017207            a              1  2013    01    01
1017208            a              1  2013    01    01

[1017209 rows x 12 columns]
```

[9]:

```
## What is the difference in the mean of sales (across all stores) when␣
↪offering a Promo and not?

## To calculate the mean with and without promos I take the mean conditional on␣
↪a variable
## The variable indicates whether or not the store had a promo on that day
nopromo = train.Promo == 0
mean_for_sales_nopromo = train.loc[nopromo, 'Sales'].mean()
mean_for_sales_nopromo = round(mean_for_sales_nopromo, 2)
print(f"The mean in sales when there is no promo is ${mean_for_sales_nopromo}.")

yespromo = train.Promo == 1
mean_for_sales_yespromo = train.loc[yespromo, 'Sales'].mean()
mean_for_sales_yespromo = round(mean_for_sales_yespromo, 2)
print(f"The mean in sales when there is a promo is ${mean_for_sales_yespromo}.")

print(f"In other words, the average is about ${round(mean_for_sales_yespromo -␣
↪mean_for_sales_nopromo)} higher when there is a promo.")
```

```
The mean in sales when there is no promo is $4406.05.
The mean in sales when there is a promo is $7991.15.
In other words, the average is about $3585 higher when there is a promo.
```

[10]:

```
# Are there any anomalies in the data where the store was "Open" but had no␣
↪sales recorded? or vice versa?

## To check if there is an anomaly where the store was open but had no sales I␣
↪look for the minimum sale for stores listed as open
```

```
## There is a minimum of 0 which means that such an anomaly does exist
store_open = train.Open == 1
min_for_store_open = train.loc[store_open, 'Sales'].min()
min_for_store_open
## To find where and when this anomaly occurred we can simply find the row␣
↪index and return the store ID and the date
zerosale_anomaly = np.flatnonzero(np.logical_and(train.Sales == 0, train.Open␣
↪== 1))
zerosale_anomaly_stores = train.loc[zerosale_anomaly, ['Store','Date']]
zerosale_anomaly_stores

## To see if there is an anomaly where a store recorded sales when it was␣
↪closed, I look for the maximum sale for stores listed as closed
## The maximum is 0 so there is no anomaly in this case
store_closed = train.Open == 0
max_for_store_closed = train.loc[store_closed, 'Sales'].max()
max_for_store_closed
```

[10]: 0

[10]:

	Store	Date
86825	971	2015-05-15
142278	674	2015-03-26
196938	699	2015-02-05
322053	708	2014-10-01
330176	357	2014-09-22
340348	227	2014-09-11
340860	835	2014-09-11
341795	835	2014-09-10
346232	548	2014-09-05
346734	28	2014-09-04
347669	28	2014-09-03
348604	28	2014-09-02
386065	102	2014-07-24
386173	238	2014-07-24
386227	303	2014-07-24
386304	387	2014-07-24
387652	882	2014-07-23
387656	887	2014-07-23
397285	102	2014-07-12
406384	925	2014-07-03
407532	57	2014-07-01
437311	1017	2014-06-05
438426	1017	2014-06-04
477534	1100	2014-04-30
478649	1100	2014-04-29
506085	661	2014-04-04

```
512964     850  2014-03-29
525365     986  2014-03-18
531396     327  2014-03-12
561199      25  2014-02-13
562314      25  2014-02-12
582982     623  2014-01-25
584097     623  2014-01-24
591147     983  2014-01-18
592262     983  2014-01-17
744697     663  2013-09-02
750000     391  2013-08-28
772836     927  2013-08-08
805283    1039  2013-07-10
806398    1039  2013-07-09
817174     665  2013-06-29
818289     665  2013-06-28
843969     700  2013-06-05
872940     681  2013-05-10
874853     364  2013-05-08
875968     364  2013-05-07
885113     589  2013-04-29
889932     948  2013-04-25
933937     353  2013-03-16
975098     259  2013-02-07
982983     339  2013-01-31
984098     339  2013-01-30
990681     232  2013-01-24
999016     762  2013-01-17
```

[10]: 0

[11]:
```
## Which store type ('a','b' etc.) has had the most sales?

## First we have to merge the 2 dataframes based on the store ID
merged_df = pd.merge(train,store, on='Store')

## Then we can create a dataframe that groups the data by the store type
storetype_df = merged_df.groupby('StoreType')

## Using the store type dataframe, we can create a dataframe of the means of␣
↪sales for each store type
## This shows us that store type A has the most sales
sumsales_df = storetype_df['Sales'].sum()
sumsales_df = sumsales_df.reset_index()
sumsales_df = sumsales_df.sort_values('Sales', ascending=False)
sumsales_df
```

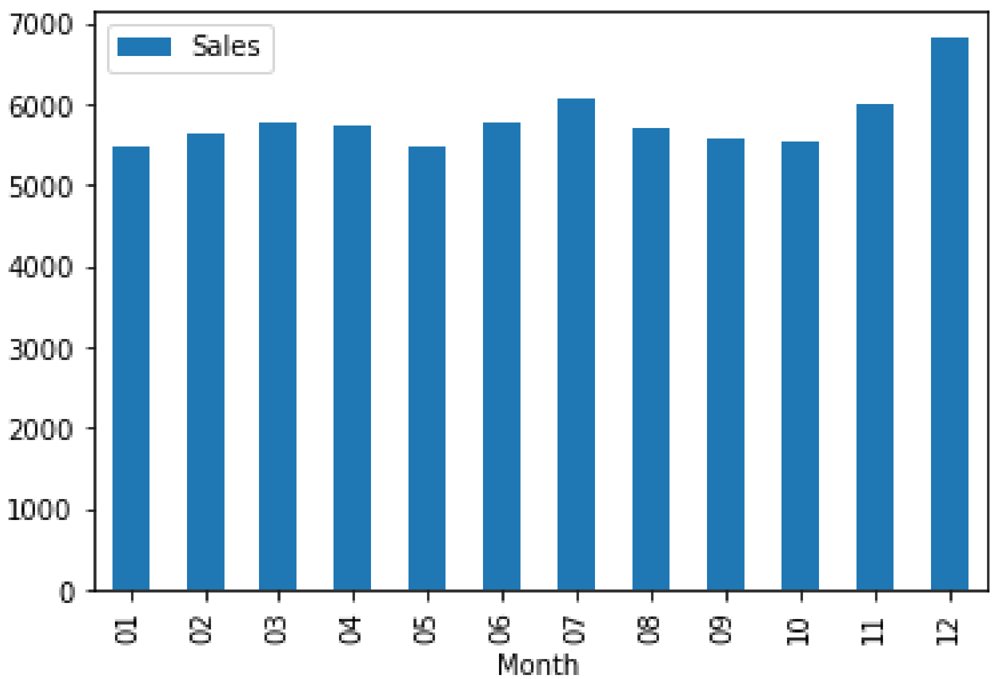
[11]:
```
  StoreType       Sales
0         a  3165334859
3         d  1765392943
2         c   783221426
1         b   159231395
```

2 EXERCISE 1

2.1 Part B: Plotting

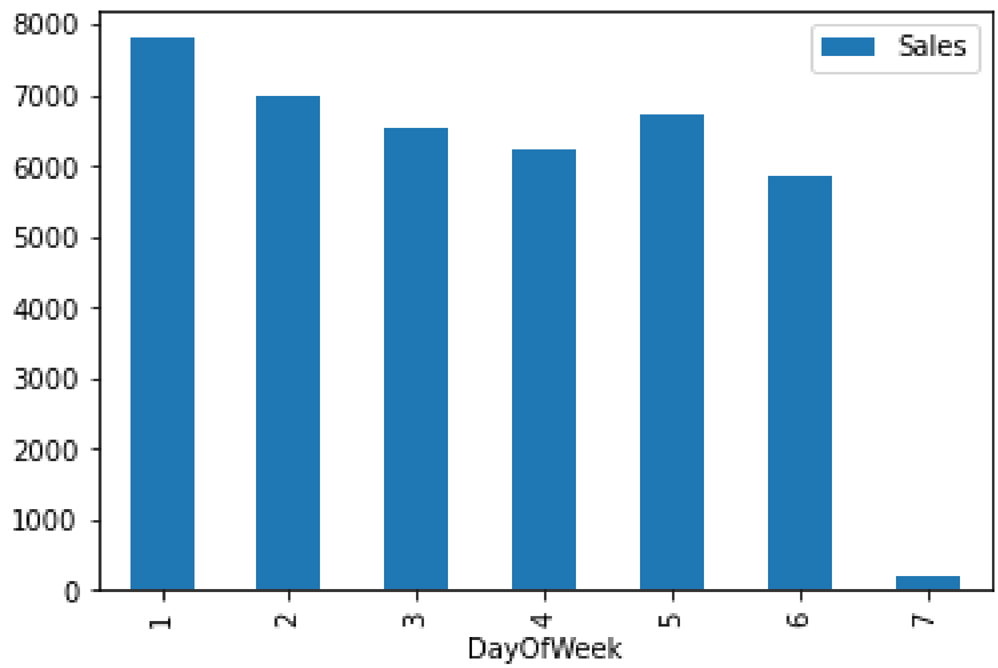
[12]:
```python
## On a monthly basis how do the mean of sales vary (across all stores)? Plot␣
↪these.
## The bar plot shows us that the mean of sales is pretty steady throughout the␣
↪year with a large spike in December

train[["Day", "Month", "Year"]] = train["Date"].str.split("-", expand = True)
month_df = train.groupby('Month')
monthsales_df = month_df['Sales'].mean()
monthsales_df = monthsales_df.reset_index()

monthsales_df.plot(x ='Month', y='Sales', kind='bar')
```

[12]: <AxesSubplot:xlabel='Month'>



[13]:
```
## On a daily basis how do the mean of sales vary (across all stores)? Plot␣
↪these.
## Monday has the highest mean and Sunday is very very low because most Rossman␣
↪stores are closed on this day

daily_df = train.groupby("DayOfWeek")
dailysales_df = daily_df['Sales'].mean()
dailysales_df = dailysales_df.reset_index()

dailysales_df.plot(x='DayOfWeek', y='Sales', kind='bar')
```

[13]: <AxesSubplot:xlabel='DayOfWeek'>



[14]:
```
## For the first store id, plot its cumulative sales for the first year

store1index = np.flatnonzero(train.Store == 1)
store1index = store1index.tolist()
store1info = train.loc[store1index, ['Sales']]
store1info = store1info[0:366]
store1info.insert(loc= 1, column='DayOfYear',value=list(range(0,366)))
store1info['CumSales'] = store1info['Sales'].cumsum()

store1info.plot(x='DayOfYear', y='CumSales',kind='line')
plt.title('Store 1 Cumulative Sales')
plt.xlabel('Day # in First Year Opened')
```

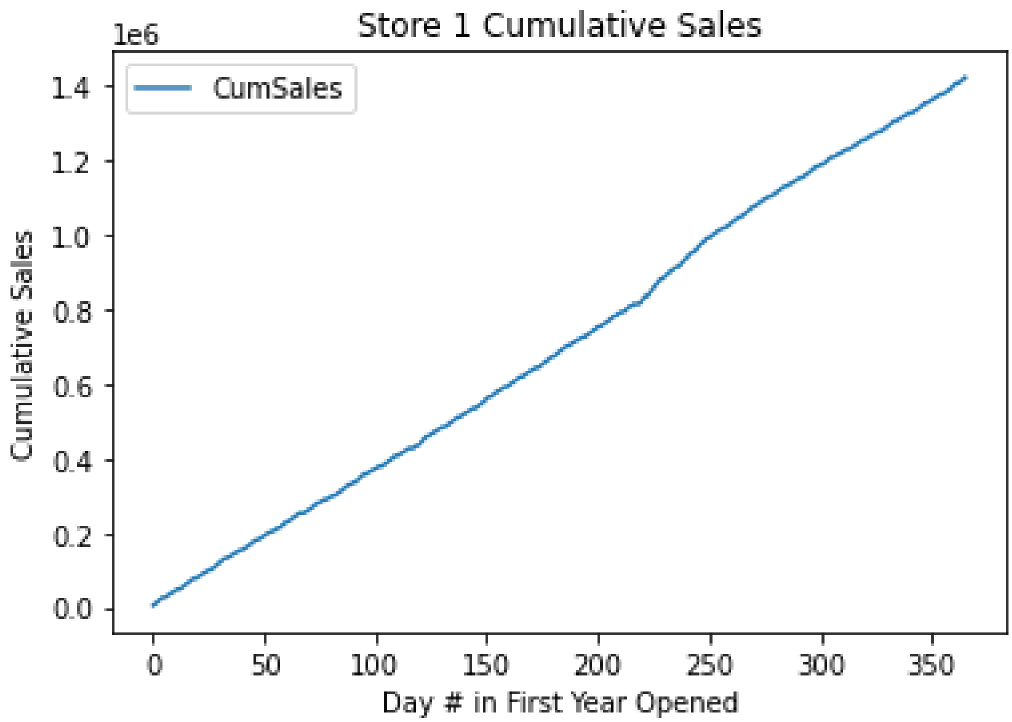
```
plt.ylabel('Cumulative Sales')
```

[14]: <AxesSubplot:xlabel='DayOfYear'>

[14]: Text(0.5, 1.0, 'Store 1 Cumulative Sales')

[14]: Text(0.5, 0, 'Day # in First Year Opened')

[14]: Text(0, 0.5, 'Cumulative Sales')



[15]:
```
## Plot and comment on the following relationships: customers(x-axis) vs.
↪sales(y-axis)
## 2) competitiondistance(x-axis) vs. sales(y-axis)

Customers = train['Customers']
Sales = train['Sales']

plt.scatter(Customers, Sales, c='Crimson')
m, b = np.polyfit(Customers, Sales, 1)
plt.plot(Customers, m*Customers + b)
plt.xlabel('# of Customers')
plt.ylabel('Sales')
## this scatter plot shows a clear positive relationship between # of customers
↪and sales
```

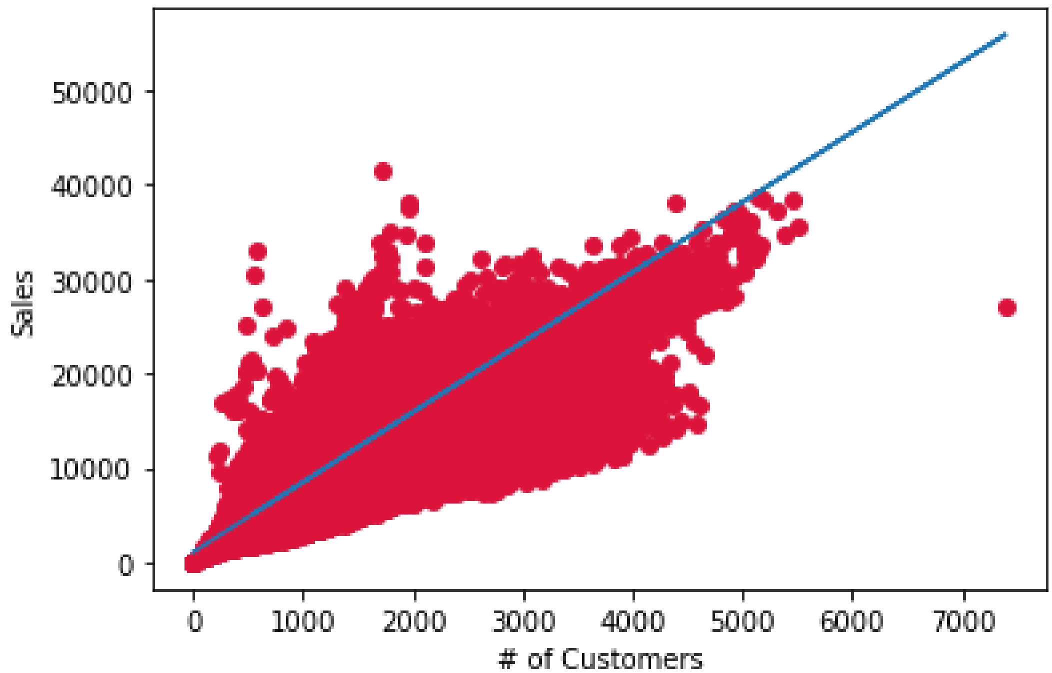
```
## the more customers the higher the sales tend to be
```

[15]: <matplotlib.collections.PathCollection at 0x1731abbecd0>

[15]: [<matplotlib.lines.Line2D at 0x1731a19d9d0>]

[15]: Text(0.5, 0, '# of Customers')

[15]: Text(0, 0.5, 'Sales')



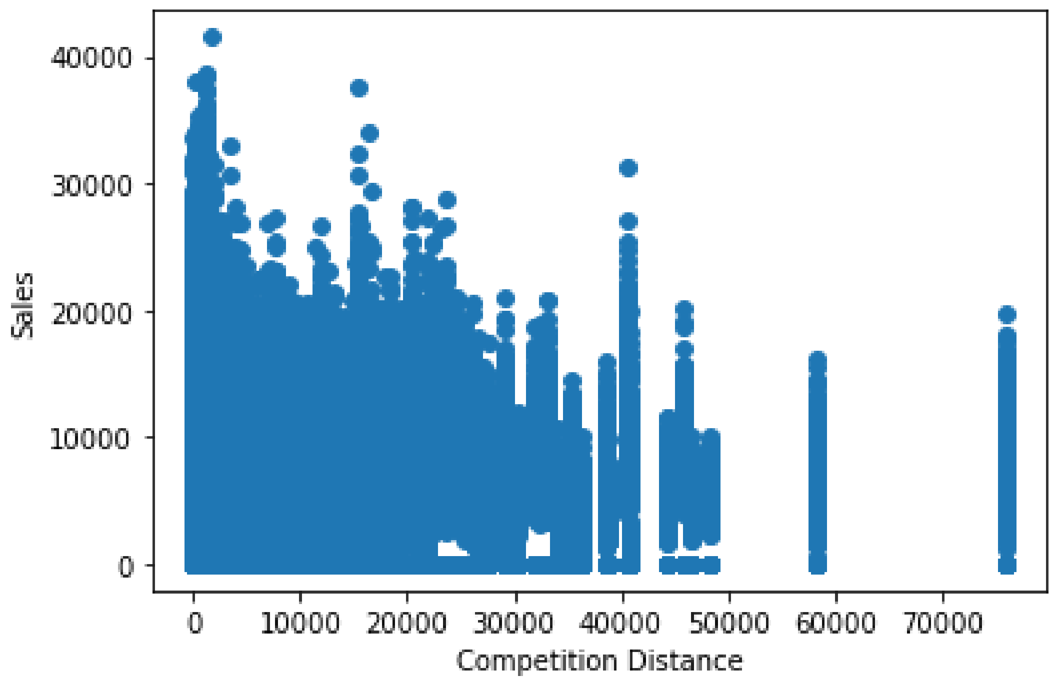
[16]:
```
## Plot and comment on the following relationships: competitiondistance(x-axis)
↪vs. sales(y-axis)

CompDist = merged_df['CompetitionDistance']
mSales = merged_df['Sales']

plt.scatter(CompDist, mSales)
plt.xlabel('Competition Distance')
plt.ylabel('Sales')
## the relationship in this graph is less clear but this coincides with
↪concepts in economics
## conventional distance decay curves vary by region and population
## therefore stores that are close but are in an unpopulated area have low
↪sales
```

```
## while stores that are close in a populated area have high sales and␣
↪increased competition
```

[16]: <matplotlib.collections.PathCollection at 0x17319f82280>

[16]: Text(0.5, 0, 'Competition Distance')

[16]: Text(0, 0.5, 'Sales')



[17]:
```
## Plot an array of Pearson correlations between all features.
## Remember to do the merge operation between the dataframes store and train.

merged_df.corr().style.background_gradient(cmap="Blues")
```

[17]: <pandas.io.formats.style.Styler at 0x1731a0a40a0>

[18]:
```
## For the first 10 stores (id'ed) draw boxplots of their sales
## Which store has the highest median sales?

for i in range(1,11):
    storeindex = np.flatnonzero(train.Store == i)
    storeindex = storeindex.tolist()
    storeinfo = train.loc[storeindex, ['Sales']]
    plt.boxplot(storeinfo, positions=[i])
## based on the boxplots, store 4 has the highest median sales
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1731869ac40>,
  <matplotlib.lines.Line2D at 0x1731869afa0>],
 'caps': [<matplotlib.lines.Line2D at 0x17318725340>,
  <matplotlib.lines.Line2D at 0x173187256a0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1731869a8b0>],
 'medians': [<matplotlib.lines.Line2D at 0x17318725a00>],
 'fliers': [<matplotlib.lines.Line2D at 0x17318725d60>],
 'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x17318b12550>,
  <matplotlib.lines.Line2D at 0x17318b128b0>],
 'caps': [<matplotlib.lines.Line2D at 0x17318b12c10>,
  <matplotlib.lines.Line2D at 0x17318b12f70>],
 'boxes': [<matplotlib.lines.Line2D at 0x17318b121f0>],
 'medians': [<matplotlib.lines.Line2D at 0x17318a20310>],
 'fliers': [<matplotlib.lines.Line2D at 0x17318a20670>],
 'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x17318a20d60>,
  <matplotlib.lines.Line2D at 0x173189f8100>],
 'caps': [<matplotlib.lines.Line2D at 0x173189f8460>,
  <matplotlib.lines.Line2D at 0x173189f87c0>],
 'boxes': [<matplotlib.lines.Line2D at 0x17318a20a90>],
 'medians': [<matplotlib.lines.Line2D at 0x173189f8b20>],
 'fliers': [<matplotlib.lines.Line2D at 0x173189f8e80>],
 'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x173189d7640>,
  <matplotlib.lines.Line2D at 0x173189d7970>],
 'caps': [<matplotlib.lines.Line2D at 0x173189d7d00>,
  <matplotlib.lines.Line2D at 0x17318970040>],
 'boxes': [<matplotlib.lines.Line2D at 0x173189d7310>],
 'medians': [<matplotlib.lines.Line2D at 0x173189703a0>],
 'fliers': [<matplotlib.lines.Line2D at 0x17318970700>],
 'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x17318970df0>,
  <matplotlib.lines.Line2D at 0x17318909190>],
 'caps': [<matplotlib.lines.Line2D at 0x17318909520>,
  <matplotlib.lines.Line2D at 0x17318909880>],
 'boxes': [<matplotlib.lines.Line2D at 0x17318970b20>],
 'medians': [<matplotlib.lines.Line2D at 0x17318909be0>],
 'fliers': [<matplotlib.lines.Line2D at 0x17318909f40>],
 'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x173188ec610>,
  <matplotlib.lines.Line2D at 0x173188ec970>],
 'caps': [<matplotlib.lines.Line2D at 0x173188eccd0>,
```

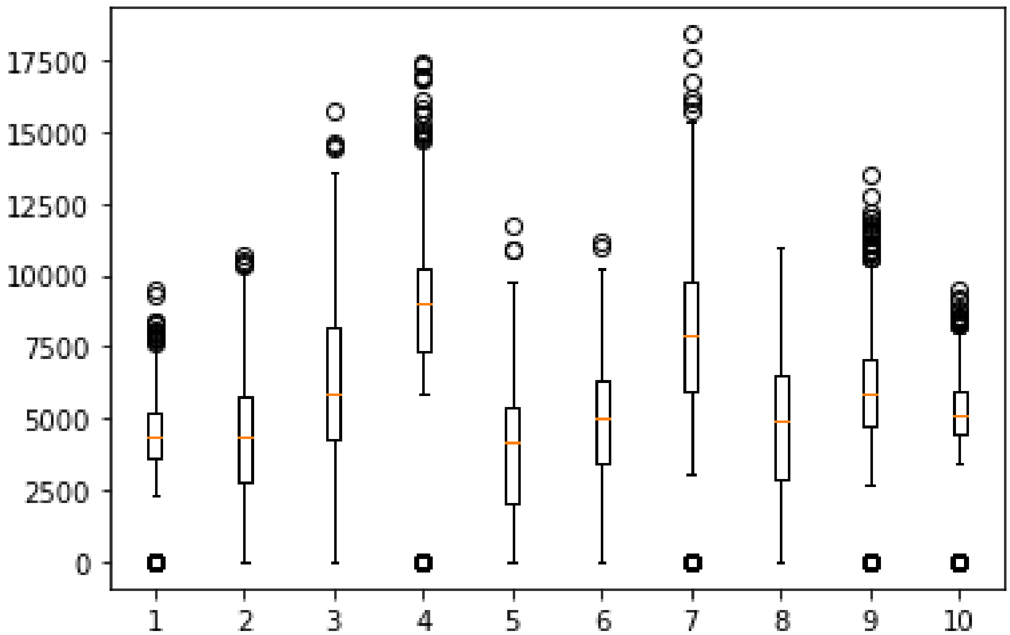
```
 <matplotlib.lines.Line2D at 0x17318885070>],
'boxes': [<matplotlib.lines.Line2D at 0x173188ec340>],
'medians': [<matplotlib.lines.Line2D at 0x173188853d0>],
'fliers': [<matplotlib.lines.Line2D at 0x17318885730>],
'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x17318885e20>,
  <matplotlib.lines.Line2D at 0x173188681c0>],
 'caps': [<matplotlib.lines.Line2D at 0x17318868520>,
  <matplotlib.lines.Line2D at 0x17318868880>],
 'boxes': [<matplotlib.lines.Line2D at 0x17318885b50>],
 'medians': [<matplotlib.lines.Line2D at 0x17318868c40>],
 'fliers': [<matplotlib.lines.Line2D at 0x17318868f70>],
 'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x173188056a0>,
  <matplotlib.lines.Line2D at 0x17318805a00>],
 'caps': [<matplotlib.lines.Line2D at 0x17318805d90>,
  <matplotlib.lines.Line2D at 0x173187e00d0>],
 'boxes': [<matplotlib.lines.Line2D at 0x173188053d0>],
 'medians': [<matplotlib.lines.Line2D at 0x173187e0430>],
 'fliers': [<matplotlib.lines.Line2D at 0x173187e0790>],
 'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x173187e0e80>,
  <matplotlib.lines.Line2D at 0x1731877d220>],
 'caps': [<matplotlib.lines.Line2D at 0x1731877d580>,
  <matplotlib.lines.Line2D at 0x1731877d8e0>],
 'boxes': [<matplotlib.lines.Line2D at 0x173187e0bb0>],
 'medians': [<matplotlib.lines.Line2D at 0x1731877dc40>],
 'fliers': [<matplotlib.lines.Line2D at 0x1731877dfa0>],
 'means': []}
```

[18]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x173187556d0>,
  <matplotlib.lines.Line2D at 0x17318755a30>],
 'caps': [<matplotlib.lines.Line2D at 0x17318755d90>,
  <matplotlib.lines.Line2D at 0x1731863b130>],
 'boxes': [<matplotlib.lines.Line2D at 0x17318755400>],
 'medians': [<matplotlib.lines.Line2D at 0x1731863b490>],
 'fliers': [<matplotlib.lines.Line2D at 0x1731863b7f0>],
 'means': []}
```



3 EXERCISE 2

3.1 Part A: Implementing Gaussian Elimination

[73]:
```
## Generate a matrix X with dimensions 100×10. Initialize it with normal distribution μ = 2 and  = 0.01
## Since we are denoting  0 as the parameter for the bias/intecept, make the first column of X all ones

X = np.random.normal(2,0.01,(100,10))
ones = np.ones((100,1))
X = np.append(ones, X, axis = 1)

## Generate a matrix Y with dimensions 100 × 1. Initialize it with random uniform distribution.
Y = np.random.uniform(0,1,(100,1))
```

[74]:
```
## Implement linear regression algorithm and train it using matrix X to learn values of  0:10.
## Implement the algorithm given in Fig.1 to solve system of linear equations

def gauss(A, b, n):
    ## GAUSSIAN ELIMINATION
    p = [0 for x in range(n)]
    s = [0 for x in range(n)]
```

```
    for i in range(n):
        p[i] = i
        smax = 0
        for j in range(n):
            if abs(A[i][j]) > smax:
                smax = abs(A[i][j])
        s[i] = smax

    for k in range(n - 1):
        rmax = 0
        for i in range(k, n):
            r = abs(A[p[i]][k]) / s[p[i]]
            if r > rmax:
                rmax = r
                rindex = i
        temp = p[k]
        p[k] = p[rindex]
        p[rindex] = temp
        for i in range(k + 1, n):
            factor = A[p[i]][k] / A[p[k]][k]
            for j in range(k, n):
                A[p[i]][j] = A[p[i]][j] - factor * A[p[k]][j]
    ## FORWARD ELIMINATION
            b[p[i]] = b[p[i]] - factor * b[p[k]]

    global beta
    beta = np.zeros((n,1))
    ## BACKWARD SOLVE
    for i in reversed(range(n)):
        Sum = b[p[i]]
        for j in range(i + 1, n):
            Sum = Sum - A[p[i]][j] * beta[j]
        beta[i] = Sum / A[p[i]][i]

    for i in range(0,n):
        print(f"Beta[{i}] = {beta[i]}")

gauss(X,Y,11)
```

```
Beta[0] = [152.81340525]
Beta[1] = [-13.37703378]
Beta[2] = [-50.08371643]
Beta[3] = [-21.44454319]
Beta[4] = [-48.40849103]
Beta[5] = [-17.90357071]
Beta[6] = [-1.9272884]
Beta[7] = [36.58801019]
Beta[8] = [-16.78459348]
```

```
Beta[9] = [35.14493092]
Beta[10] = [22.12326335]
```

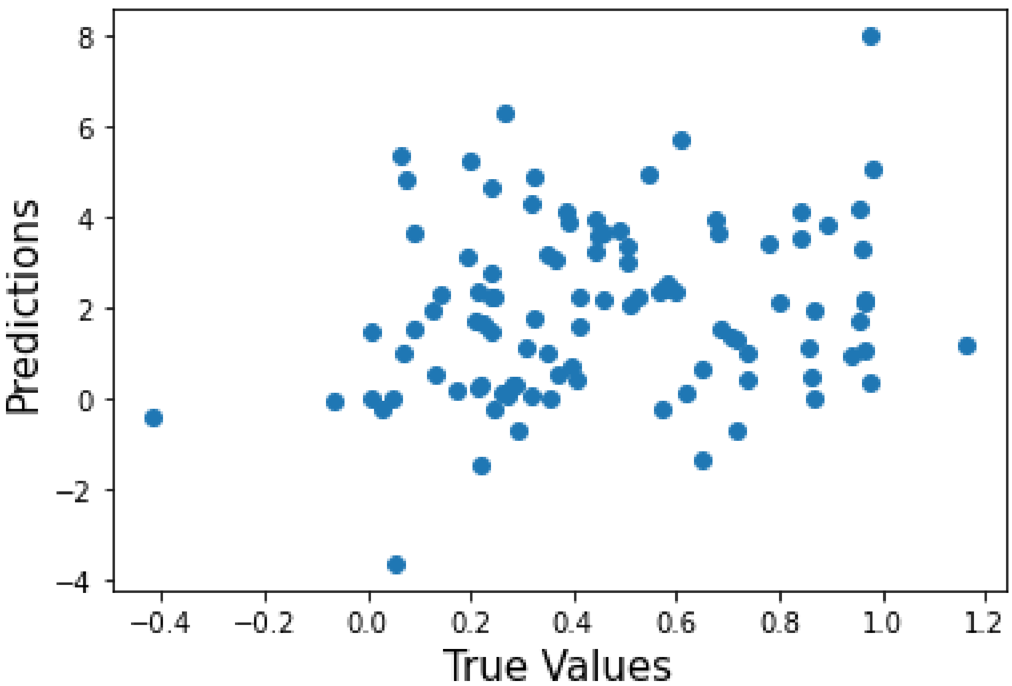
[75]:
```
## Implement the corresponding pred. algorithm and calc. the points for each␣
↪training example in matrix X

Y_predicted = np.matmul(X,beta)
```

[61]:
```
## Plot the training points from matrix Y and predicted values in the form of␣
↪scatter graph

plt.scatter(Y, Y_predicted)
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
```

[61]:
```
<matplotlib.collections.PathCollection at 0x1730bc558e0>
```

[61]:
```
Text(0.5, 0, 'True Values')
```

[61]:
```
Text(0, 0.5, 'Predictions')
```



[76]:
```
## In the end use numpy.linalg.lstsq to learn 0:10 and plot the predictions␣
↪from these parameters

newbetas = np.linalg.lstsq(X,Y, rcond = None)[0]
```

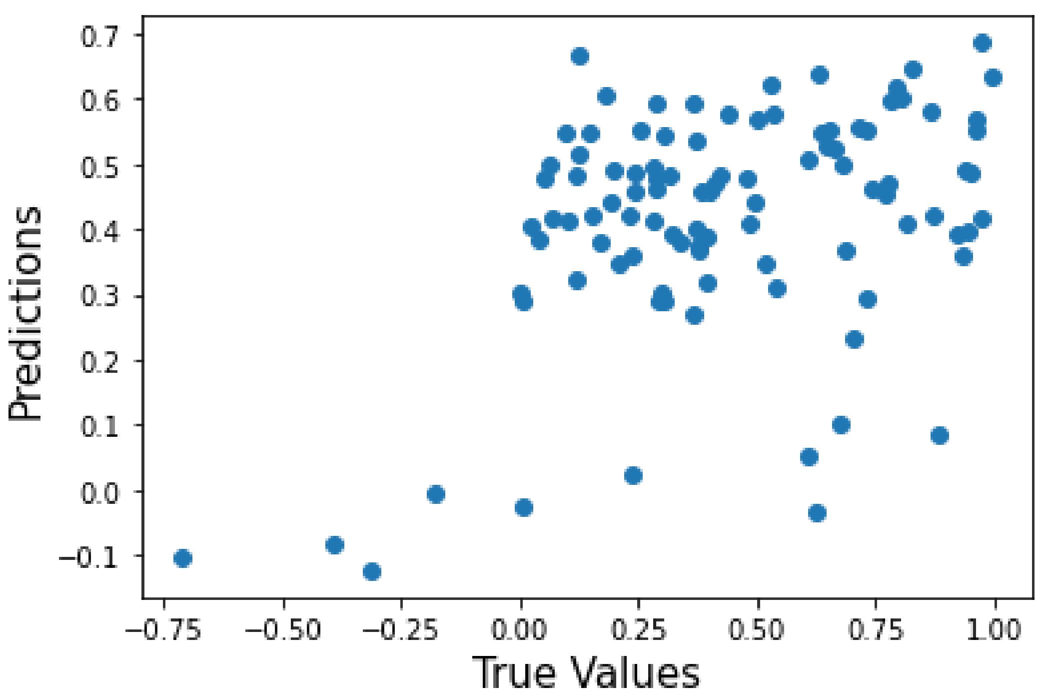
```
newbetas

newY_predicted = np.matmul(X,newbetas)

plt.scatter(Y, newY_predicted)
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
```

[76]:
```
array([[-13.8591854 ],
       [ -3.12570677],
       [  4.94614296],
       [  2.8711654 ],
       [ -2.62648504],
       [ -2.01119914],
       [ -2.22094696],
       [  5.37899586],
       [ -1.46899049],
       [  3.19850595],
       [  2.2159244 ]])
```

[76]:
```
<matplotlib.collections.PathCollection at 0x17317b12a90>
```

[76]:
```
Text(0.5, 0, 'True Values')
```

[76]:
```
Text(0, 0.5, 'Predictions')
```



4 EXERCISE 2

4.1 Part B: Multiple Linear (Auto) Regression

[29]:
```
## Find all the stores that have sales recorded for 942 days
## Create a data matrix of the shape (#_of_stores, 942) for the daily sales record of these stores

## We can sort the data according to store ID so that we can pivot the dataframe based on this value
matrix942 = train.sort_values(by=['Store'])
matrix942 = matrix942.pivot(index='Store',columns='Date',values='Sales')
## Getting rid of any rows with missing values ensures that only stores with 942 days of data remain
matrix942 = matrix942.dropna()
matrix942
```

[29]:
```
Date   2013-01-01  2013-01-02  2013-01-03  2013-01-04  2013-01-05  2013-01-06  \
Store
1             0.0      5530.0      4327.0      4486.0      4997.0         0.0
2             0.0      4422.0      4159.0      4484.0      2342.0         0.0
3             0.0      6823.0      5902.0      6069.0      4523.0         0.0
4             0.0      9941.0      8247.0      8290.0     10338.0         0.0
5             0.0      4253.0      3465.0      4456.0      1590.0         0.0
…               …           …           …           …           …           …
1111          0.0      5097.0      4579.0      4640.0      3325.0         0.0
1112          0.0     10797.0      8716.0      9788.0      9513.0         0.0
1113          0.0      6218.0      5563.0      5524.0      5194.0         0.0
1114          0.0     20642.0     18463.0     18371.0     18856.0         0.0
1115          0.0      3697.0      4297.0      4540.0      4771.0         0.0

Date   2013-01-07  2013-01-08  2013-01-09  2013-01-10  …  2015-07-22  \
Store                                                   …
1          7176.0      5580.0      5471.0      4892.0  …      3464.0
2          6775.0      6318.0      6763.0      5618.0  …      5093.0
3         12247.0      9800.0      8001.0      7772.0  …      5414.0
4         12112.0     10031.0      8857.0      9472.0  …      8503.0
5          6978.0      5718.0      5974.0      4999.0  …      3595.0
…               …           …           …           …  …  …        …
1111       9444.0      6472.0      5307.0      5887.0  …      4021.0
1112      25165.0     17058.0     14724.0     14366.0  …      6029.0
1113       8984.0      6866.0      6115.0      7508.0  …      4565.0
1114      21237.0     18816.0     17073.0     18075.0  …     20424.0
1115       6905.0      5243.0      4649.0      5007.0  …      5342.0

Date   2015-07-23  2015-07-24  2015-07-25  2015-07-26  2015-07-27  2015-07-28  \
Store
```

```
1          3769.0     3706.0     4364.0        0.0     6102.0     5011.0
2          4108.0     3854.0     2512.0        0.0     6627.0     5671.0
3          5702.0     5080.0     3878.0        0.0     8107.0     8864.0
4          7286.0     8322.0     9322.0        0.0    11812.0    10275.0
5          3713.0     3815.0     2030.0        0.0     7059.0     6083.0
…             …          …          …          …          …          …
1111       3587.0     3918.0     2177.0        0.0     7742.0     6793.0
1112       6730.0     6220.0     6216.0        0.0    14383.0     9583.0
1113       6410.0     6399.0     4784.0        0.0     7582.0     6468.0
1114      20564.0    19627.0    21312.0        0.0    26720.0    25518.0
1115       6150.0     5816.0     6897.0        0.0    10712.0     8093.0

Date   2015-07-29  2015-07-30  2015-07-31
Store
1          4782.0      5020.0      5263.0
2          6402.0      5567.0      6064.0
3          7610.0      8977.0      8314.0
4         10514.0     10387.0     13995.0
5          5899.0      4943.0      4822.0
…             …           …           …
1111       4907.0      5263.0      5723.0
1112       9179.0      9652.0      9626.0
1113       6640.0      7491.0      7289.0
1114      25840.0     24395.0     27508.0
1115       7661.0      8405.0      8680.0

[934 rows x 942 columns]
```

[107]:

```
## Use the first 800 stores in this data matrix for training and the rest for␣
↪testing
## Split the sales data into 2 parts:
## the 1st part contains the information about the first 900 days of sales
## and the 2nd contains the information about the last 42 days of sales

Xtrain = matrix942.iloc[:800, :900]
Xtest = matrix942.iloc[800:, :900]
Ytrain = matrix942.iloc[:800, 900:]
Ytest = matrix942.iloc[800:, 900:]
```

[103]:

```
## Iteratively build multiple linear regression models for column vectors of

for col in Ytrain.columns:
    I = Ytrain[col]
    beta = np.linalg.inv(Xtrain.transpose().dot(Xtrain)).dot(Xtrain.
↪transpose()).dot(I)
    print(len(beta))
```

```
900
```

```
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
900
```

[104]:

```
## Use the  0:900 for each of the 42 models to make predictions for each day ahead
## In total 42 days

Ytrain_predicted = np.matmul(Xtrain,beta)
```

[106]:
```
## Calculate and print the daily RMSE and MAE for all 42 sales values using␣
↪test split (    as input)
## Also calculate and print overall average RMSE and MAE. (i.e. just the mean␣
↪RMSE of all 42 models)

## Use numpy to calculate the MSE and just raise it to the 0.5th power to get␣
↪RMSE
MSE = np.square(np.subtract(Xtest,Ytrain_predicted)).mean()
RMSE = (MSE)**0.5

MAE = np.abs(np.subtract(Xtest,Ytrain_predicted)).mean()
```

[ ]:
```
## Use the following approaches and report the overall average RMSE and MAE for␣
↪them:
    ## i. Repeating last sale value per store
    ## ii. Repeating mean value per store
    ## iii. Repeating mean value per store per weekday
```

[ ]:
```
## Reason why or why not Linear Regression is a good choice for this task

'''
Because there are so many different beta values being used in the model, linear␣
↪regression is not optimal.
With this many beta values, the regression model will overfit itself to the␣
↪data.
This results in poor prediction capabilities.
'''
```