به نام خدا

گزارش فوق براساس مشاهدات و نظرات شخصی اینجانب حسین اصحابی به شماره دانشجویی ۹۴۴۴۰۱۰۴۳۱ بوده، در طی روند بدست آوردن مدلی برای پیدا کردن مقدار رتبه یک شخص در Twitter. امید است که بتوانم گوشه‌ایی از تلاش‌های استاد خوبم، آقای دکتر صیدی را به کار ببندم.

گام نخست

در ابتدا نیاز به وارد کردن داده‌ها در برنامه را داریم. این روند با الهام گرفتن از ویدیویی معرفی شده در کانال انجام شده است. بر طبق این ویدیو ابتدا مقدار key که دیتا فریم در آن قرار دارد را بدست آورده و توسط تابع get آن را بصورت دیتا‌فریم ذخیره می‌کنیم.

همچنین دیکشنری برای ورود درست داده بصورت ذیل داریم:

dtype\_dict = {'Tweet Id': int, 'User Name': str, 'Favs': int, 'RTs': int, 'Followers': int, 'Listed': int, 'likes': int, 'tweets': int, 'reply': int,  
}'URLs': str, 'Tweet content': str,

مطابقا هر کاری که روی دیتا فریم مورد نظر از نظر تغییر داده انجام می‌دهم روی داده‌های StudentTest هم صورت خواهم داد. برای شروع نگاهی به دیتافریم کنیم. من از دستور isnull().sum()برای این منظور بهره میگیرم.

ویژگی‌های Favs و RTs مقدار زیادی nans دارند. مقدار قابل ملاحضه‌ایی از داده‌های رتبه rank هم که برابر با ۱۲۵ است دارای مقدار nan است.

isnull().sum()

Tweet Id 0

User Name 0

Favs 3194

RTs 2642

Followers 26

Following 69

Listed 458

likes 1

tweets 1

reply 1

rank 125

URLs 0

Tweet content 1

dtype: int64

در واقع بی ‌ارزش ترین ردیف ها rows همین ردیف هایی هستند که مقدار رتبه rank آنها nans است.  
تمام داده‌های دیتا فریم به قصد پیدا کردن این مقدار آموزش میبینند، پس وجود آنها برای آموزش مدل ما مضر می‌باشد.

در اینجا قصد داریم پیش از اینکه داده ها nan را همانطور که صورت مسئله خواسته است با مقدار صفر پر کنیم. آنها را از دیتا‌فریم حذف نمایم. پس از این کار تنها یک داده وجود دارد که برای ویژگی‌های tweets وreply و likes مقدار nan دارد. این ویژگی را هم حذف می‌کنم. این داده در ردیف ۲۵۲ دیتا‌فریم قرار دارد.

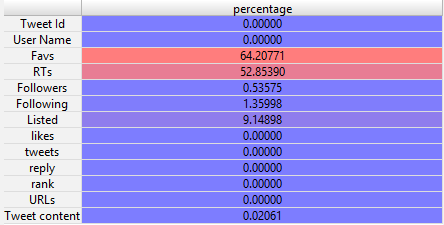
برای شناسایی این داده از دستور فوق استفاده کردم:

df[df.reply.isnull()]

Tweet Id ... Tweet content

252 8.787658e+17 ... RT @OfficeofEdTech: Want to learn more about O...

ویژگی‌های Favs و RTs هم هر کدام با ۶۴٪ و ۵۲٪ درصد بیشترین missing value ها را به خود اختصاص داده اند. این حجم از دیتا را نمیتوان نادیده گرفت و همه را با مقدار ۰ پر کرد. به همین خاطر آنها را بصورت ستونی حذف می‌کنم.



حال که داده های بی‌ استفاده را حذف نموده ایم وقت آن رسید است که داده‌های nans را با مقادیر صفر پر کنیم. البته پیش از اینکار به دلیل ماهیت ویژگی Listed میخواهم بجای پر کردن این ويژگی، آن را با مقدار most frequent occur پر کنم. برای دیگر ویژگی ها که nans دارند همانطور که صورت مسئله خواسته است عمل می‌کنیم.

در ادامه خواسته شده است که یک ورودی جدید به دیتا فریم اضافه کنیم. قرار بر این است که این ورودی تعداد کلمات در ویژگی Tweet content را بشمارد و آن را با نام word count ذخیره کنیم.

برای این منظور هم از تابع split از کتابخانه pandas بهره می‌گیریم.

همچنین ویژگی های بی مورد همچون Tweet Id که مطمئنا تاثیری در بهبود مدل ندارد (اگر مدل را خراب نکند) حذف می‌کنم تا دیتا فریم تمیز تر و بهتری از نظر بهبود دید بصری خودم داشته باشم.

دیتا فریم ما تا این مرحله بدین شکل و ترتیب است:

Followers Following Listed ... reply rank word count

0 3630 3549 345 ... 322 1282619 28

1 4311 106 140 ... 1450 2 18

2 47 174 5 ... 541 194367 13

… … … … … …

4976 85808 2227 192 ... 1154 70 21

4977 31 72 1 ... 840 508 1

4978 1130 981 45 ... 329 9042459 38

[4853 rows x 8 columns]

در این بین چند ویژگی نیز خود افزودیم:

df[**'Followers\_sub\_Following'**] = df[**'Followers'**] - df[**'Following'**]  
df[**'tweets\_reply'**] = df[**'tweets'**] \* df[**'reply'**]  
df[**'log\_likes'**] = df[**'likes'**].apply(**lambda** x: np.log(x))

مطمنا رابطه ایی بین فالور ها و فالوینگ ها وجود دارد و کسی که رتبه خوبی دارد باید مقدار مثبتی از این ویژگی را داشته باشد.

در قسمت دوم هم هر دو ویژگی مورد نظر reply و tweets از آنجا که رابطه مستقیمی دارند، وقتی که هر دو مقدار بزرگ باشند، مقدار بزرگی را خواهیم داشت. این را می‌گویند interaction.

و در قسمت سوم هم سعی شده که لایک های کم از میان کسانی یا پست هایی که لایک های فراوانی دارد جدا شود.

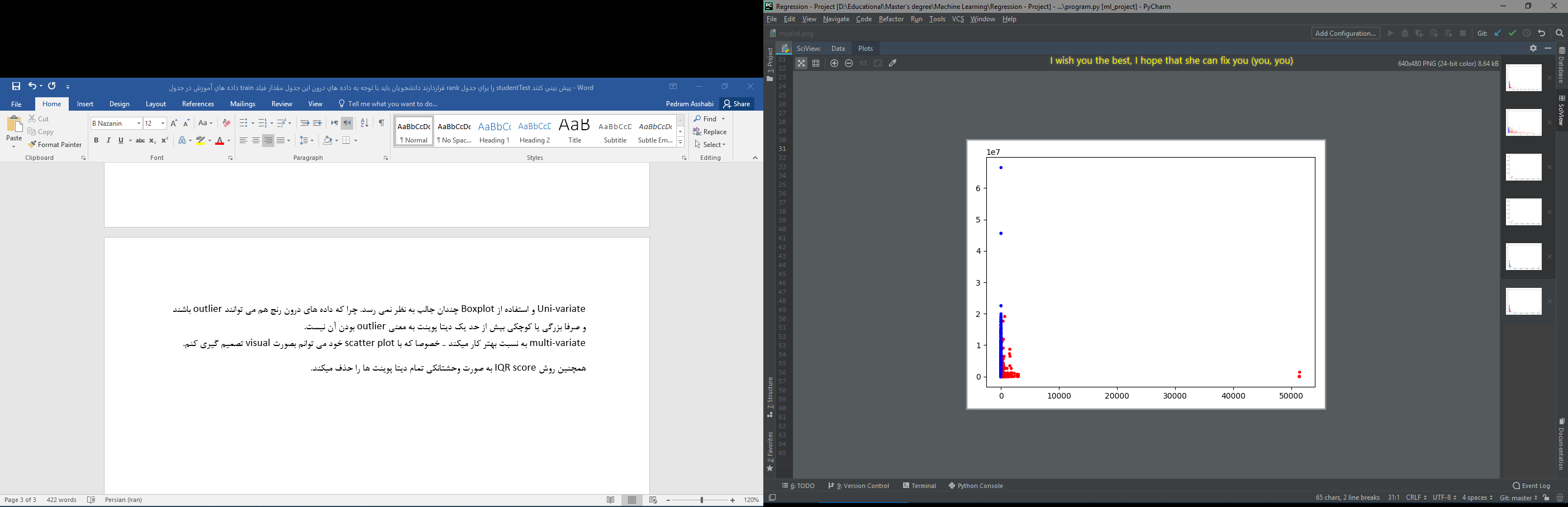
حال نوبت به حذف داده های outlier رسیده. سوال اینجاست چه روشی را برای حذف داده های outlier انتخاب کنیم. روشهای ساده تا پیچیده ایی وجود دارند .

* <https://www.kdnuggets.com/2017/01/3-methods-deal-outliers.html>
* <https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba>

**Outliers**

Uni-variate و استفاده از Boxplot چندان جالب به نظر نمی رسد. چرا که داده های درون رنج هم می توانند outlier باشند و صرفا بزرگی یا کوچکی بیش از حد یک دیتا پوینت به معنی outlier بودن آن نیست.  
در قیاس با روش دیگر که multi-variate نام دارد، به نسبت بهتر کار میکند.

همچنین روش دیگری با نام IQR score را امتحان کردم که به صورت وحشتانکی تمام دیتا پوینت ها را حذف می‌کرد. برای نمونه در زیر ورودی RTs را داریم که با فرمول IQR دیتا پوینت های به رنگ قرمز در آمده را حذف کرده است.



در میان این روش ها، استفاده از مدل خطی رگرسیون و پیدا کردن دیتاپوینت هایی که، سبب بیشترین error در یک مدل شده اند و همچنین استفاده از انحراف معیار و z\_score را بهتر می بینم. ترکیب این دو روش در کنار نمودار ها به باور من می تواند نتیجه بهتری را ارايه کند.

شبه کد کلاس OutlierDetectHelper که برای کمک به تشخیص outlier ها نوشته شده است. این کلاس یک مدل خطی ساده را از داده ها بدست می آورد و error را بصورت یک بردار نزولی اندیس گذاری شده برمی گرداند. انتخاب k تا از این بردار، نقاط outlier را می تواند بطور ملموسی برگرداند.

class OutlierDetectHelper:

def outlier\_selection\_by\_linear\_model(self, data\_frame, x, y):

learn a model

compute error

*find the top most data point with highest error:  
 get diff of real output and predicted output*

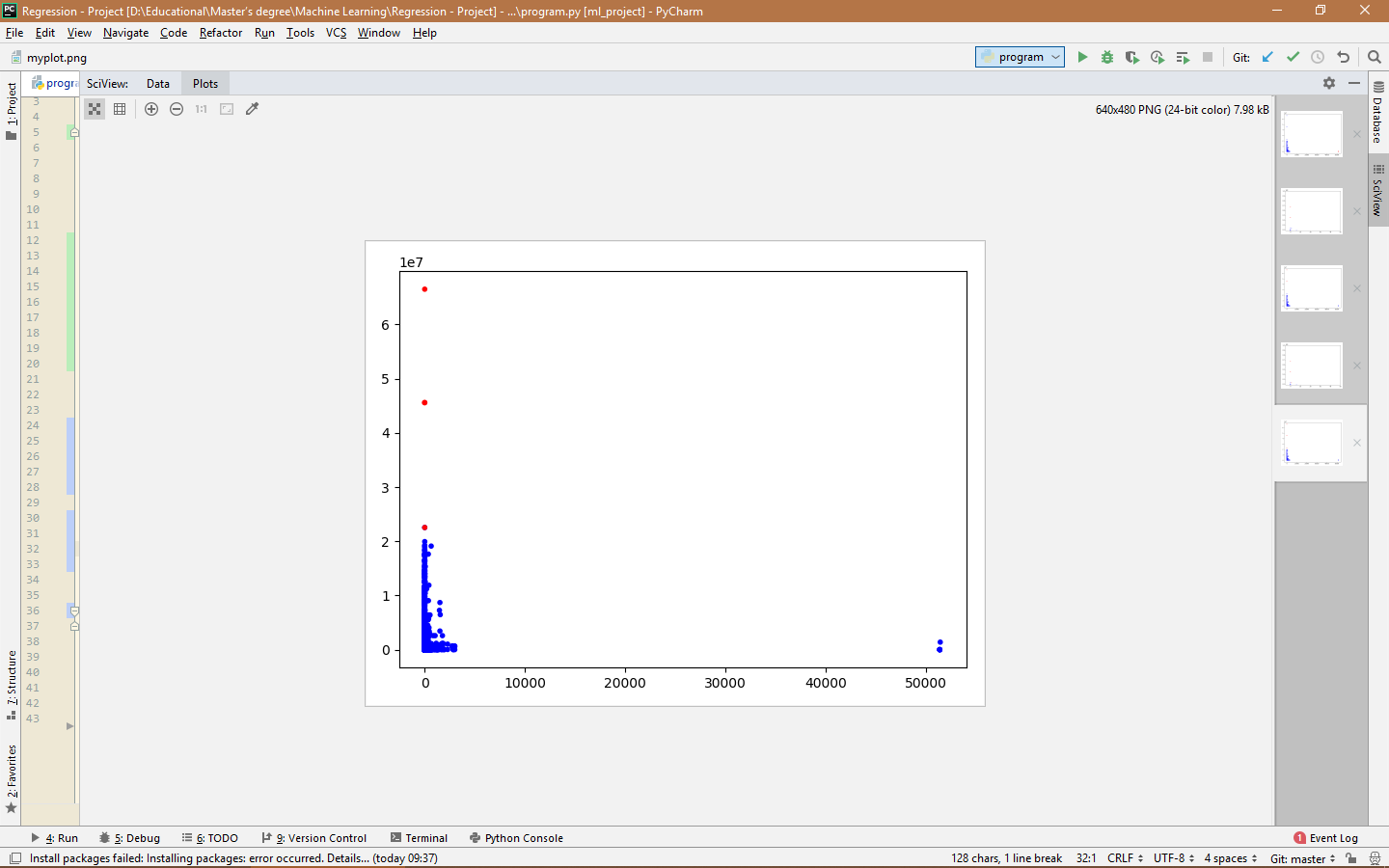
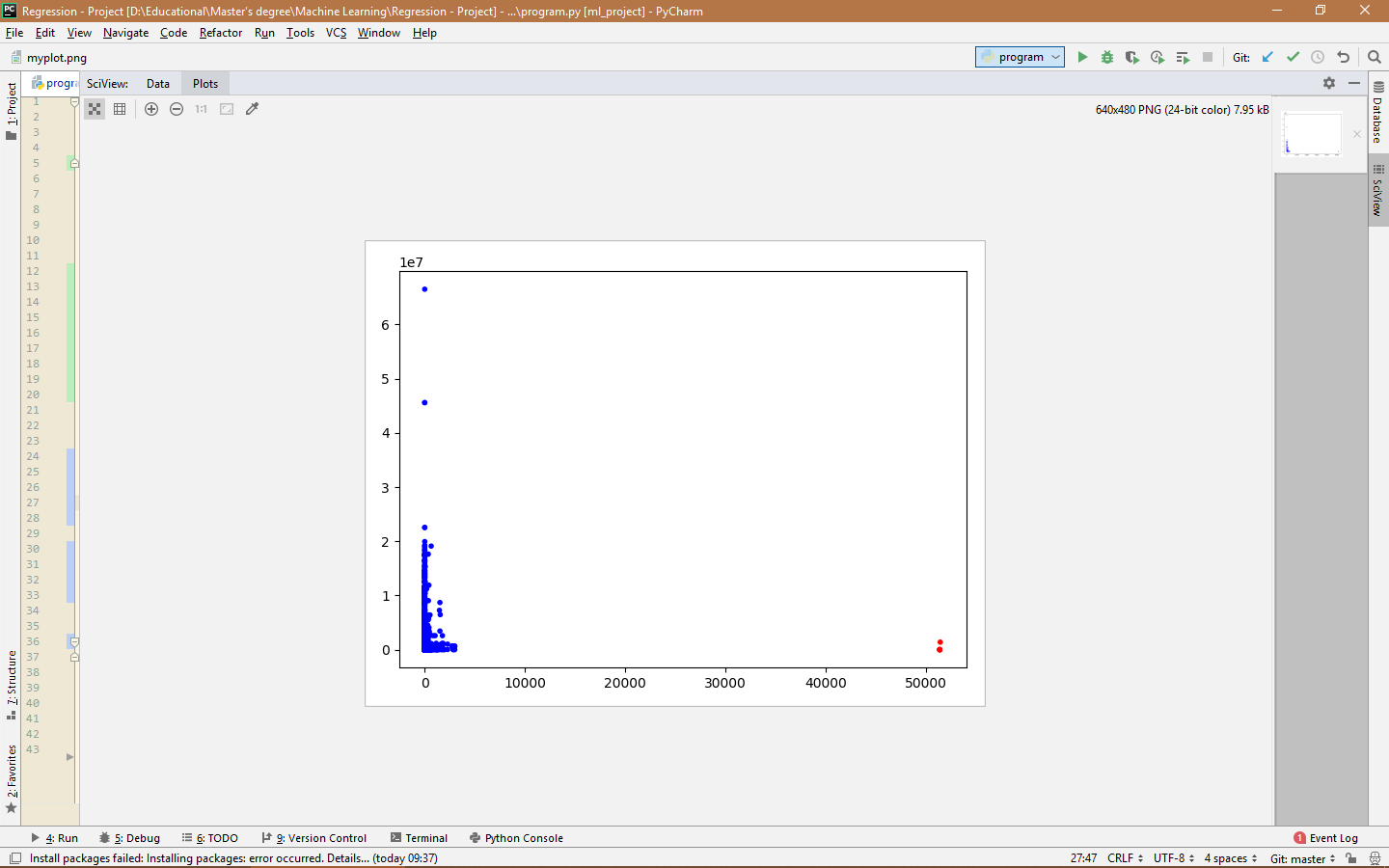
*sort the indices in descending order:* return

def plot\_excluded\_outliers(self, xlim=None, ylim=None, k: int = 10):

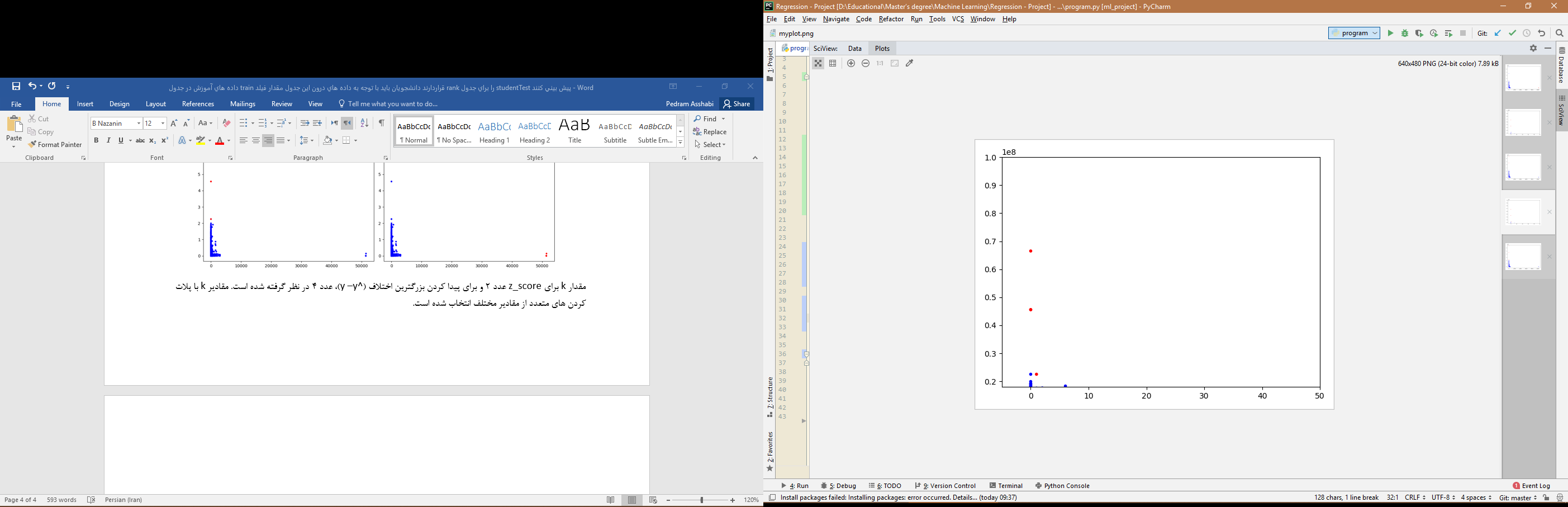
k\_indices = *# select k data point  
  
 # make a vector of str colors with 'r' and 'b' color, blue for included data point  
 # and red for excluded data points:*  *# visualization the data points:* plt.scatter(self.\_\_df[self.\_\_x], self.\_\_df[self.\_\_y], marker=**'.'**, c=col)  
 plt.show()  
 return  
  
def get\_k\_outlier(self, k):  
 return desc\_sorted\_indices[0:k] *# select k data point*

برای مشاهده کامل کد [اینجا](https://github.com/passhabi/regression_project/blob/master/learnt/regression/OutlierDetectHelper.py) کلیک کنید.

در زیر سمت چپ برای ورودی RTs، استفاده از مدل ساده-خطی رگریسیون و پیدا کردن k تا از بزرگترین دیتاپوینت هایی که باعث تحمیل بیشترین فشار (High leverage points) به مدل شده اند را مشاهده گر هستیم؛ و همینطور در سمت راست فاصله دیتاپوینت از مقدار mean، واختصاص یک Thershold برای پیدا کردن < z\_score k را مشاهده گر هستیم:



مقدار k برای z\_score عدد ۲ و برای پیدا کردن بزرگترین اختلاف (y –y^)، عدد ۴ در نظر گرفته شده است. مقادیر k با پلات کردن های متعدد از مقادیر مختلف انتخاب شده است.



در ابتدای کار هم کاملا عیان بود که داده های فوق outlier هستند؛ گرچه اکنون اعتماد خاطر بیشتری نسبت به این داده ها وجود دارد اما ظاهرا همان ویژوال کردن Scatterبرای تشخیص این داده ها کافی است.

تصویر سمت چپ از نمای نزدیکتر

تابع محاسبه z\_score:

**def** show\_delete\_outlier(df: pd.DataFrame, threshold: int, x: str, y: str = **'rank'**, delete: bool = **False**):  
 z\_scores = np.abs(stats.zscore(df[x]))  
  
 *# visualisation with threshold:* colors = np.where(z\_scores > threshold, **'r'**, **'b'**)  
 plt.scatter(df[x], df[y], c=colors, marker=**'.'**)  
 plt.show()  
  
 **if** delete **is True**:  
 *# deleting outliers form data frame:* indices = np.where(z\_scores > threshold)[0] *# get the outlier corresponding indices* df = df.drop(index=indices) *# drop the outliers* **return** df

گرچه انحراف معیار نتیجه خوبی را ارائه می دهد اما با وجود انتخاب Threshold مناسب برای هر ورودی، برتری نسبت به روش ویژوال کردن ( PlotBox یا بهتر Scatter plot ) ندارد. شاید بتوان با داده های validation نیز به مقدار درست در این روش و رگریسون دست یافت.

در نتیجه همه این صحبت ها، می توان با پلات کردن همان نتیجه مناسب را برای هر ورودی یافت. راهی ساده و کم دردسر. ورودی هر ستون را یکی یکی پلات کرده، داده های outlier را با زوم و مقایس مناسب از روی plot دیده و شناسایی کرده و با یک Thershold مناسب که خود از این طریق آن را در نظر میگیرم، outlier ها را حذف میکنم. نکته قابل توجه در پلات کردن این است که گرچه از scatter plot استفاده میکنم. اما در راستای یک بعد outlier ها را باید در نظر گرفت.

برای تمام ورودی ها مراحل فوق را انجام داده و داده های outlier را شناسایی می کنم.

نمونه کد حذف outlier ها در ورودی RTs در زیر آورده شده است:

plt.scatter(df[**'RTs'**], df[**'rank'**], marker=**'.'**)  
plt.show()  
df = df[df[**'RTs'**] < 20000]  
print(len(df[df[**'RTs'**] > 20000]), **'data points deleted from data frame due to RTs outlier removing.'**)

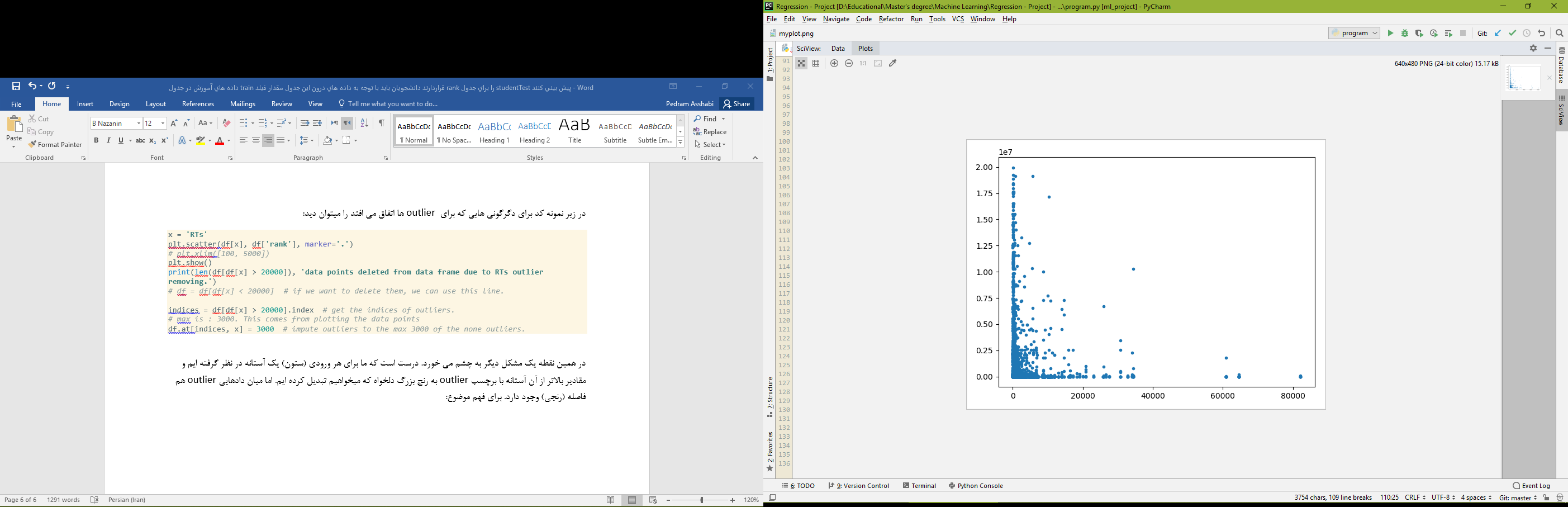
به جای حذف ردیف هایی که مقادیر outlier دارند، روش های دیگری نیز وجود دارد. در واقع با حذف دیتاپوینت ها می توان اطلاعات ارزشمندی را از دست داد. به همین خاطر به روشی دیگر سعی میکنم که این داده ها را حذف کنم. به این صورت که این داده را با مقدار max هر ستون ورودی پر میکنم. با این کار گرچه خطایی را به مدل وارد میکنم اما اطلاعاتی را از دست نداده و مدلی که بدست خواهد آمد از دیدگاه اینجانب به مراتب بهتر خواهد بود تا مدلی که outlier های آن بطور کلی حذف شده باشد.

در زیر نمونه کد برای تغییر داده‌های outlier را میتوان دید:

x = **'RTs'**plt.scatter(df[x], df[**'rank'**], marker=**'.'**)  
*# plt.xlim([100, 5000])*plt.show()  
print(len(df[df[x] > 20000]), **'data points deleted from data frame due to RTs outlier removing.'**)  
*# df = df[df[x] < 20000] # if we want to delete them, we can use this line.*indices = df[df[x] > 20000].index *# get the indices of outliers.*  
*# max is : 3000. This comes from plotting the data points*df.at[indices, x] = 3000 *# impute outliers to the max 3000 of the none outliers.*

در همین نقطه یک مشکل دیگر خودنمایی می‌کند. درست است که ما برای هر ورودی (ستون) یک آستانه در نظر گرفته ایم و مقادیر بالاتر از آن آستانه با برچسب outlier به رنج بزرگ دلخواه که میخواهیم تبدیل کرده ایم. اما میان دادهایی outlier هم فاصله (رنجی) وجود دارد. برای فهم موضوع:

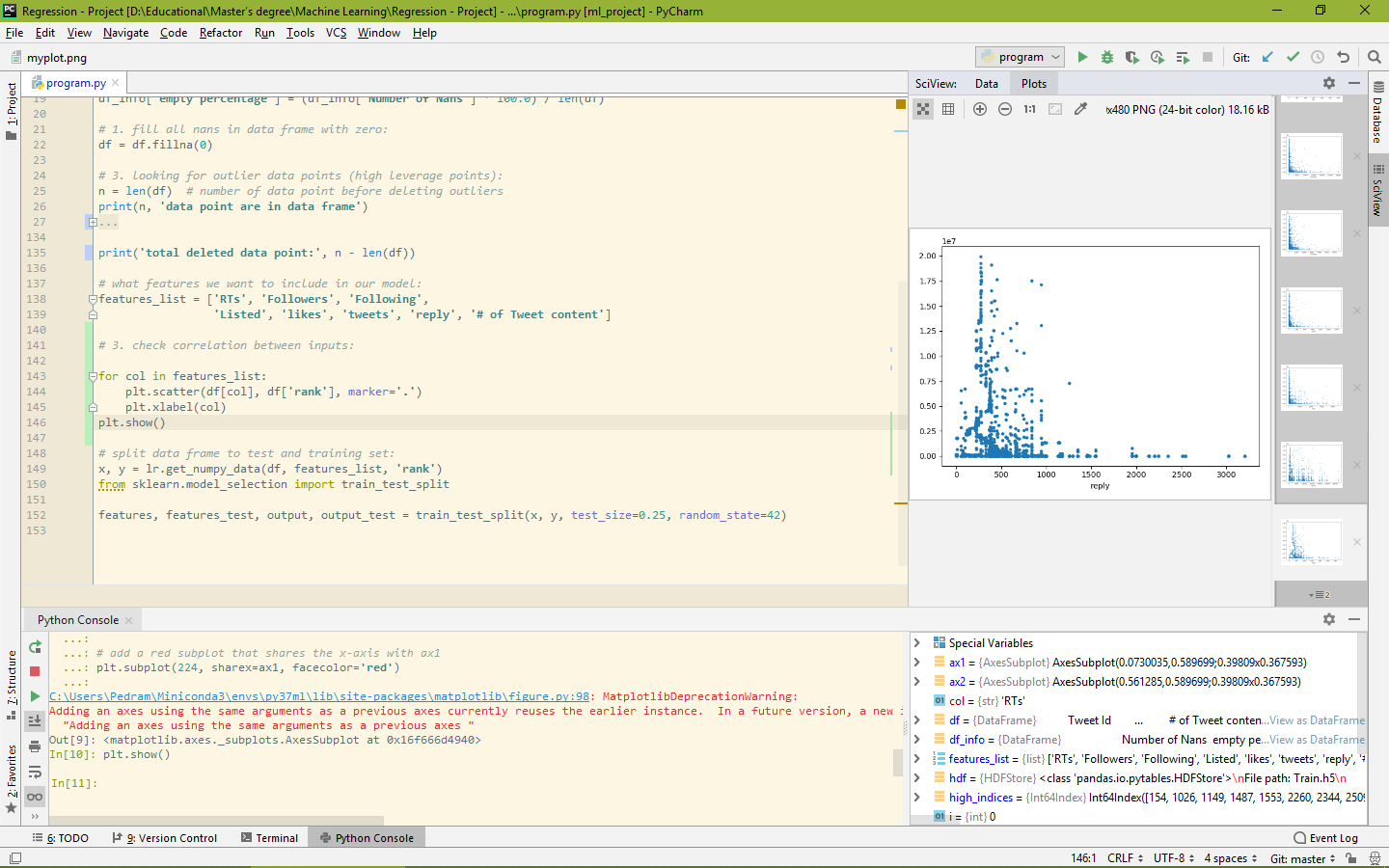
مقدار استانه



در این صورت رنجی که میان داده های outlier داریم را از بین میبرم و همه مساوی با ۴۰۰۰۰ قرار خواهند گرفت. در صورتی که میان این داده ها فرق وجود دارد. هر چند وقتی در یک بُعد این موضوع بررسی می‌شود از اهمیت آن کاسته میشود اما به مراتب باید روش های بهتری را به کار برد تا این اختلافات حفظ شود. در اینجا بصورت موقت (کد زنی) سعی میکنم که این اختلاف را تا حدودی حفظ کنم و موضوع بحث داده های outlier را خلاصه کنم.

1. **ﺍﺭﺗﺒﺎﻁ correlation ﻫﻤﻪ ﻓﻴﭽﺮﻫﺎ ﺑﺎ ﻳﮑﺪﻳﮕﺮ ﻭ ﺑﺎ ﺧﺮﻭﺟﻲ ﺭﺍ ﺑﺮﺭﺳﻲ ﮐﻨﻴﺪ.**

شاید بهتر بود پیش از حذف داده های outlier به ارتباط آنها فکر میکردم. رابطه rank معکوس انچه انتظار داریم است. رنک پایینتر = رنک بهتر است. در نتیجه آنچه پیش تر فکر میکردم، داده ها اکنون بیشتر مورد انتظار انچه که فکر می‌کنم هستند. هر چه ورودی ها مقدار کمتری دارند رنک پایینتر است و با افزایش مقدار در بیشتر ورودی ها (ستون ها) رنک کاهش میابد.  
  
انطور که از توزیع دیتا ها برداشت می شود کم اهمیت ترین فیچر میتواند تعداد کلمات در تویت باشد و پر اهمیت تر فیچر باید تعداد توویت های یک فرد و یا تعداد فالور های او باشد. از انجا که پیش از این پلات های زیادی را در متن این گزارش درج کرده ام... از اضافه کردن پلات هایی دیگر برای این مضمون خودداری میکنم.



برای نمونه بیاید ورودی reply را مد نظر قرار دهیم:

قابل مشاهده است، کسانی که پیج (صفحه توویت) با رنک خوبی (رنک خوب = رنک پایینتر) دارند و بی قولی مشهور هستند. خیلی پاسخ دیگران را نمی دهند.

**5- ﺭﻓﺘﺎﺭ ﻓﻴﭽﺮﻫﺎ ﻧﺴﺒﺖ ﺑﻪ ﺧﺮﻭﺟﻲ ﺭﺍ ﺑﺎ plot ﮐﺮﺩﻥ ﺁﻧﻬﺎ ﻧﺴﺒﺖ ﺑﻪ ﻳﮑﺪﻳﮕﺮ ﺑﺮﺭﺳﻲ ﮐﻨﻴﺪ ﺧﻄﻲ ﺍﺳﺖ ﺩﺭﺟﻪ ﺩﻭ ﻳﺎ ...**

در بیشتر ورودی ها می توان رابطه درجه دوم، سوم و شاید هم چهارم را نسبت به خروجی در نظر گرفت. ولی در کل یک منحنی شکل به صورت زیر برای اکثر ورودی ها معقول به نظر می رسد.

rank

# words in a tweet

به جز ورودی تعداد کلمات در یک توییت

rank

که خود اضافه کردیم:

other inputs

**ب) simple regression**

**1- ﺑﺎ ﮐﺪﺍﻡ ﺗﮏ ﻓﻴﭽﺮ ﻣﺪﻝ ﺑﻬﺘﺮﻱ ﺑﺎ simple regression ﺑﺪﺳﺖ ﻣﻲ ﺁﻭﺭﻳﺪ**

بهترین مدل توسط تک فیچر Favs ایجاد شد، علیرغم انتظارم که این فیچر مقادیر بالایی از NaN را داشت؛ اما چناچه که پیداست رابطه خوبی نسبت به rankدر آن وجود دارد.

Best simple model made by [ Favs ] feature

its rss is: 1.7229040907157622e+20

**ج) multiple regression**

1. **مرﺍﻗﺐ ﻓﻴﭽﺮﻫﺎﻱ high correlated ﺑﺎﺷﻴﺪ!**

<https://chrisalbon.com/machine_learning/feature_selection/drop_highly_correlated_features/>

بر این اساس فیچری پیدا نشد. کد ها در برنامه درج شده اند.

# Create correlation matrix  
corr\_matrix = df.corr().abs()  
  
# Select upper triangle of correlation matrix  
upper = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(np.bool))  
  
# Find index of feature columns with correlation greater than 0.95  
to\_drop = [column for column in upper.columns if any(upper[column] > 0.95)]  
  
df = df.drop(df.columns[to\_drop], axis=1)

1. **ﺍﮔﺮ ﻓﻴﭽﺮ categorical ﺩﺍﺭﻳﺪ ﺁﻧﻬﺎ ﺭﺍ ﻋﺪﺩﻱ ﮐﻨﻴﺪ!**

خدا رو شکر فیچر categorical هم نداریم.

1. ﻣﺮﺍﻗﺐ scale ﻓﻴﭽﺮﻫﺎ ﺑﺎﺷﻴﺪ، ﺑﻬﺘﺮ ﺍﺳﺖ ﺁﻧﻬﺎ ﺭﺍ normal ﮐﻨﻴﺪ ﻳﺎ standardization ﺍﻧﺠﺎﻡ ﺩﻫﻴﺪ ﻳﺎ ﻫﺮ ﺭﻭﺵ ﺩﻳﮕﺮ ﺑﺮﺍﻱ ﺍﻳﻦ ﻣﻨﻈﻮﺭ، ﻫﺮ ﺭﻭﺷﻲ ﺍﺳﺘﻔﺎﺩﻩ ﻣﻲ ﮐﻨﻴﺪ ﻳﺎ ﻧﻤﻲ ﮐﻨﻴﺪ، ﺩﻟﻴﻞ ﺁﻥ ﻭﺍﻟﮕﻮﺭﻳﺘﻢ ﺁﻥ ﻭ ﻣﺰﺍﻳﺎ ﻭ ﻣﻌﺎﻳﺐ ﺁﻥ ﺭﺍ ﺷﺮﺡ ﺩﻫﻴﺪ.

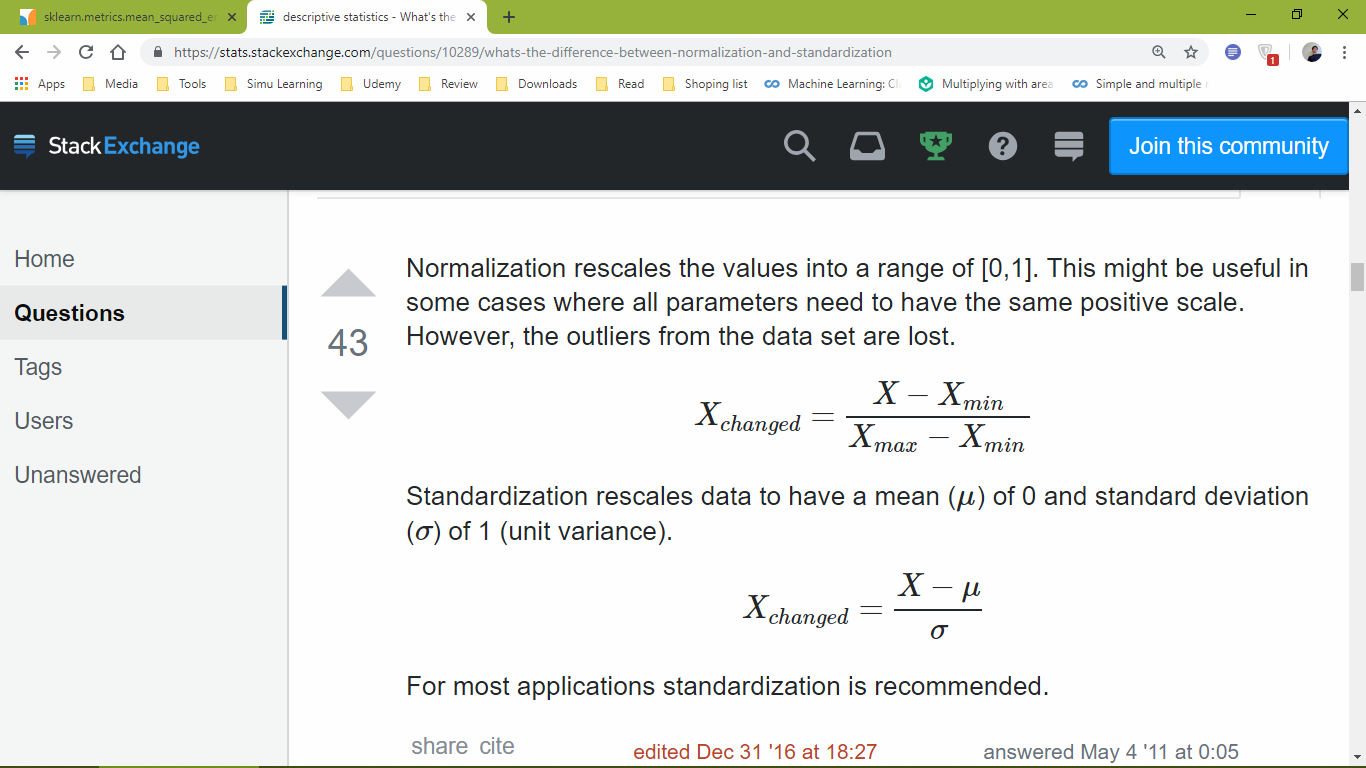
<https://stats.stackexchange.com/questions/10289/whats-the-difference-between-normalization-and-standardization>

<https://www.quora.com/When-should-you-perform-feature-scaling-and-mean-normalization-on-the-given-data-What-are-the-advantages-of-these-techniques>

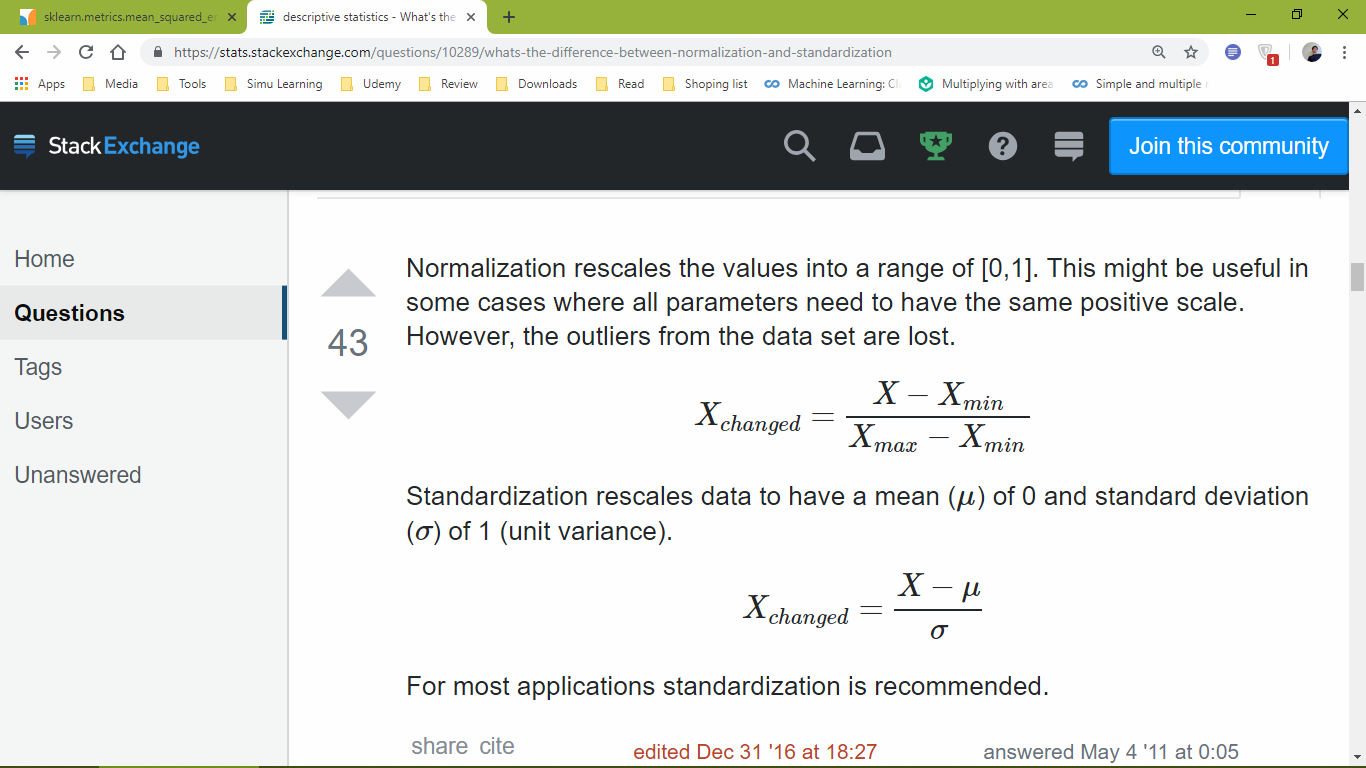
<https://www.codecademy.com/articles/normalization>

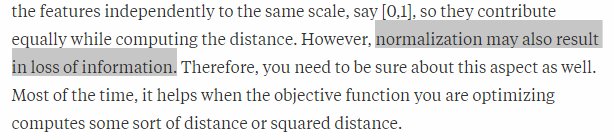
نرمال کردن داده ها باعث هم رنج کردن آنها بین بازه ۰ و ۱ میشود. در مواردی که نیاز به پارامتر های مثبت باشد، نرمال کردن مفید است. در استاندار کردن که یک روش از scale کردن فیچر ها است داده ها دارای میانگین ۰ می شوند.

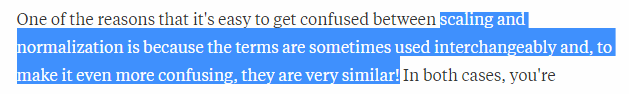
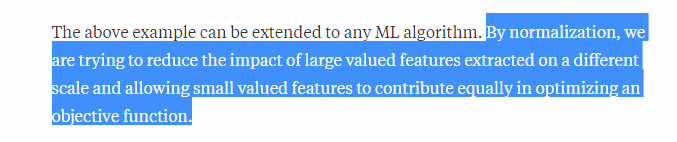
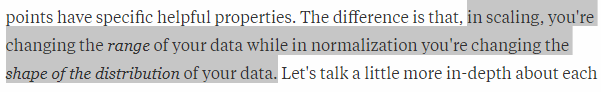
نرمال کردن:

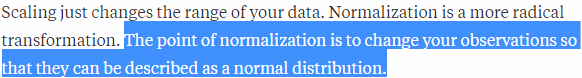


استاندار کردن:









Tip: Which Method To Use

It is hard to know whether rescaling your data will improve the performance of your algorithms before you apply them. If often can, but not always.

A good tip is to create rescaled copies of your dataset and race them against each other using your test harness and a handful of algorithms you want to spot check. This can quickly highlight the benefits (or lack there of) of rescaling your data with given models, and which rescaling method may be worthy of further investigation.

با احتساب موارد فوق گرچه استاندار کردن بهتر در این مسئله به نظر می رسید اما از آنجا که با نرمال کردن داده ها آشنایی بیشتری دارم، از آن استفاده میکنم.

def normalize\_features(feature\_matrix):  
 *"""Normalizes columns of a given feature matrix.  
  
 The function returns a pair ‘(normalized\_features, norms)’,  
 where the second item contains the norms of original* ***:param*** *feature\_matrix:  
 an ndarray matrix or a data frame of features* ***:return****:  
 return normalized features as matrix and also the norms of feature matrix  
 """* # normalization formula: hj(x) / √ ∑ hj(x)²  
 # simply feature dived by norms  
 # the following code using numpy satisfy the norm behavior for each columns:  
 norms = np.linalg.norm(feature\_matrix, axis=0)  
 normalized\_features = feature\_matrix / norms  
 return normalized\_features, norms

با استفاده از همه فیچر ها، multiple regression خطای 8166200974195 را داریم. همچنین روی داده تست خطای 7305295832687 را داریم. در این روش مشکلاتی از جمله انتخاب step\_size مناسب و tolerance را داریم. انتخاب آنها باید هوشمندانه باشه و انتخاب ناصحیح این موارد میتواند به همگرا نشدن تابع بیانجامد.

در اینجا مقادیر را با مقادیر مشخص شده در تمارین هفته دوم که موضوع بحث بوده، مقدار دهی کرده ام.

step\_size = 4e-12

tolerance = 1e9

مقدار تالرنس عدد بالایی در نظر گرفته شده است. درست یا نادرست بودن آن چندان مد نظر نیست. اما با کمی بازی کردن با این مقادیر الگوریتم برای ساعتها در حال اجرا شدن بود و به راحتی همگرا نمی شد.

ضرایب هر ویژگی (فیچر) هم در زیر قابل مشاهده است:

* intercept: -4.85855781e-04
* Favs: -1.82636882e-04
* RTs: -7.58690219e-05
* Followers: -8.59971939e-05
* Following: -1.86208824e-04
* Listed : -1.05506793e-04
* Likes: -7.63038863e-05
* Tweets: -1.53395344e-0
* Reply: -2.50771746e-0
* # of Tweet content: -4.59890248e-04

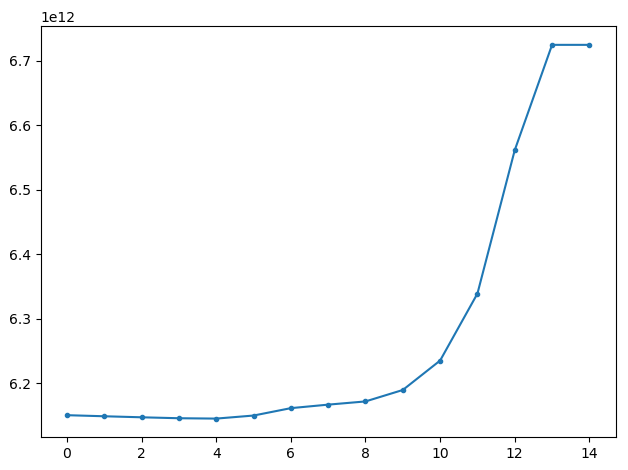
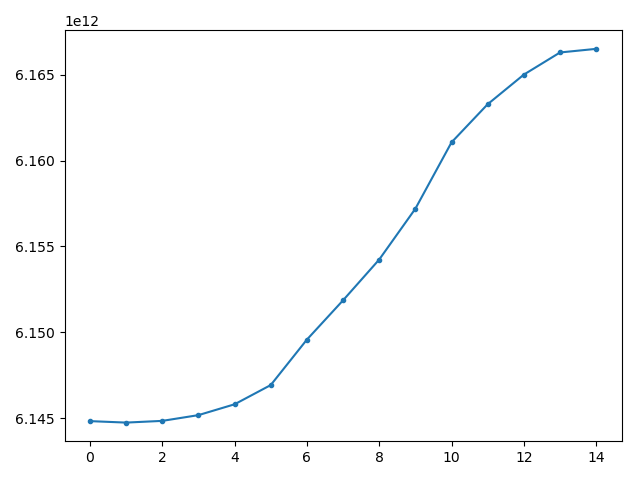
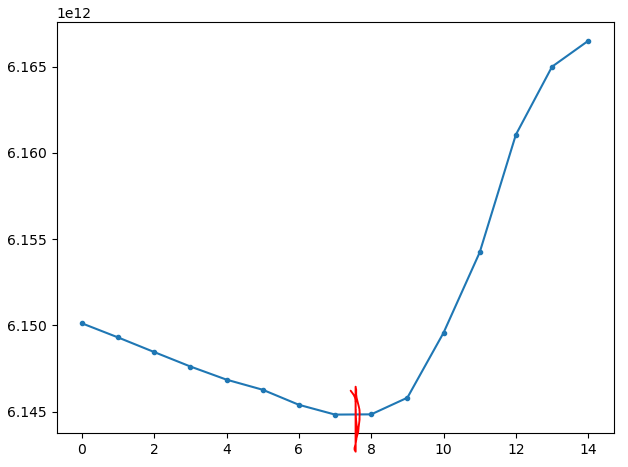
از این منظر که فیچر ها نرمال گشته اند میتوان مهمترین فیچر را در اینجا شناسایی کرد. بیشتر مقدار قدر مطلق که به هر یک از ضرایب فیچر اختصاص داده شده است، از اهمیت آن فیچر می گوید.

در ابتدا بخش ه را انجام داده و سپس بخش د.

**ه) مدل Lasso Regression**

* **از تحلیل های بخش ج استفاده کنید.**
* **با lasso فیچر های خوب را شناسایی کرده و سپس ridge بزنید.**

با استفاده از یک حلقه و یک رنج بزرگ از np.logspace در جستوجوی مقدار مناسب لاندا برای lasso هستیم. در این قسمت ابتدا تمام فیچر ها را در مدل وارد کرده ایم. همینطور در یک ارایه مقادیر تابع هزینه را نیز ذخیره می کنیم و در هر بار اجرا از این روند مقدار لگاریتمی (به صورت صعودی) لاندا را پلات میگیریم. در ابتدا مقادیر ۰ تا ۳۰ برای تابع np.logspace در نظر گرفته شده و باکمی تغییر و نزدیک شدن به مقدار لاندای مناسب به رنج ۶ تا ۷ میرسیم. در زیر چند نمونه از پلاتهایی که برای پیدا کردن مقدار مناسب لاندا صورت گرفته شده بود، آورده شده است. نا گفته نماند که این مقادیر روی داده های validation تست شده اند تا به مقدار درست لاندا برسیم.

مقدار مناسب لاندا 3562248 در نظر گرفته شده است. در زیر میتوانیم فیچر های حذف شده از مدل را ببینیم:

* intercept : 126586958.38269988
* Favs : 971505.9748369572
* RTs : -1790262.3139011832
* Followers : -2391814.3542934186
* Following : 0.0
* Listed : -929442.96893861
* likes : 7607912.799146898
* tweets : -79899285.0134189
* reply : -5125422.879342301
* # of Tweet content : 0.0

فیچر، تعداد کلمات که خود اضافه کردیم و فیچر فالورس ها از مدل حذف شده اند. همانطور که انتظار می رفت و از روی پلات داده حدس زده بودیم با مقدار 1e7 برای لاندا تنها فیچر باقی مانده tweets ها و عرض از مبدا است. این فیچر از مهمترین ورودی های مدل می باشد.

تابع هزینه، (جذر میانگین مربع یا mse) هم برای داده های train و test حساب شده اند. جای خوشحالیست که خطای داده های تست از داده های آموزش کمتر است.

* mse error on train data 7216307958763
* mse error on test data 6279048461167

**د) مدل Ridge regression**

* **ﺍﺯ ﺗﺤﻠﻴﻞ ﻫﺎﻱ ﺑﺨﺶ ﺝ ﺍﺳﺘﻔﺎﺩﻩ ﮐﻨﻴﺪ.**

در این قسمت سعی دارم که پلات ها و روند کاری را با جزییات بیشتری درج کنم. حال که می دانیم که چه فیچرهایی مناسب مدل هستند و چه فیچرهایی اضافی، می توانیم به دنبال پیدا کردن مدل مناسب برویم.

ابتدا featuer\_list را تنها با فیچر هایی که میخواهم در مدل وجود داشته باشند، بازنویسی می‌کنم.

features\_list = ['Favs', 'RTs', 'Followers', 'Listed', 'likes', 'tweets', 'reply']

همانطور که قابل مشاهده است. فیچرهای که lasso مقدار صفر را برای ضرایب آنها در نظر گرفته است را از مدل اصلی حذف کرده ایم.

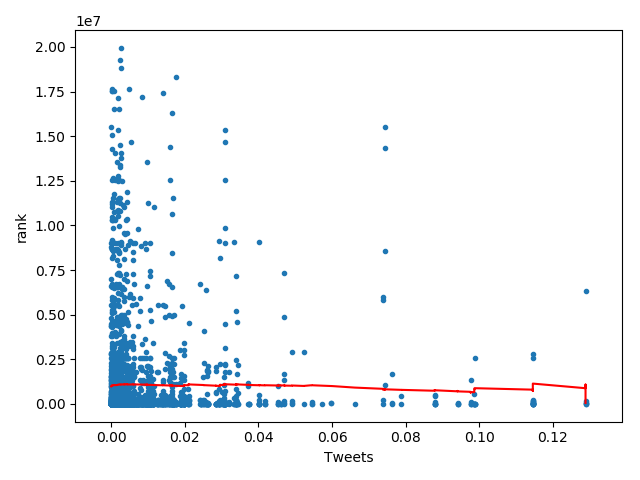
کدهای lasso را کامنت گذاری می‌کنم، چرا که دیگر نیازی به آنها نیست. اگر مسئله نخواسته بود روش K-Fold را به جای تقسیم داده ها به سه مجموعه، انجام می دادم.

متاسفانه کد نوشته شده برای ridge regression خطای سریز زیر را می‌دهد. کد نوشته شده در طول ترم و طی تمرین ها بارها امتحان شده و حتی اکنون هم برای داده های تمرین (kc\_house) به خوبی کار می‌کند. ولی الان..

* C:\Users\Pedram\Miniconda3\envs\py37ml\lib\site-packages\sklearn\metrics\regression.py:241: RuntimeWarning: overflow encountered in square
* output\_errors = np.average((y\_true - y\_pred) \*\* 2, axis=0,
* D:\Educational\Master’s degree\Machine Learning\Regression - Project\learnt\regression\ridge.py:34: RuntimeWarning: overflow encountered in double\_scalars
* derivative = derivative + 2 \* (l2\_penalty \* weight)
* Traceback (most recent call last):
* File "C:\Users\Pedram\Miniconda3\envs\py37ml\lib\site-packages\IPython\core\interactiveshell.py", line 3267, in run\_code
* exec(code\_obj, self.user\_global\_ns, self.user\_ns)
* File "<ipython-input-15-384906fa556e>", line 6, in <module>
* cost\_dict[l2\_penalty] = mse\_error(output\_valid, pred)
* File "C:\Users\Pedram\Miniconda3\envs\py37ml\lib\site-packages\sklearn\metrics\regression.py", line 239, in mean\_squared\_error
* y\_true, y\_pred, multioutput)
* File "C:\Users\Pedram\Miniconda3\envs\py37ml\lib\site-packages\sklearn\metrics\regression.py", line 77, in \_check\_reg\_targets
* y\_pred = check\_array(y\_pred, ensure\_2d=False)
* File "C:\Users\Pedram\Miniconda3\envs\py37ml\lib\site-packages\sklearn\utils\validation.py", line 573, in check\_array
* allow\_nan=force\_all\_finite == 'allow-nan')
* File "C:\Users\Pedram\Miniconda3\envs\py37ml\lib\site-packages\sklearn\utils\validation.py", line 56, in \_assert\_all\_finite
* raise ValueError(msg\_err.format(type\_err, X.dtype))
* ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

چند بار اول در حلقه به خوبی کار می‌کند ولی بعد از چند تکرار ابتدا خطای سریز داده و سپس مقادیری برای weight باز نمی‌گرداند. دلیل درج این موضوع در متن گزارش این است که به احتمال بالا، اگر درصد رفع خطای فوق باشم. زمان محدود خود را برای ارايه گزارش از دست می‌دهم. پس بطور موقت از توابع پیش نوشته شده sklearn استفاده می‌کنم و اگر زمان اضافه آمد به این موضوع باز می‌گردم. همچنین کد نوشته شده برای ridge regression را علاوه بر فایل های ضمینه پروژه می توانید در [اینجا](https://github.com/passhabi/ml-regression/blob/W4-2/learnt/regression/ridge.py) نگاه کنید.

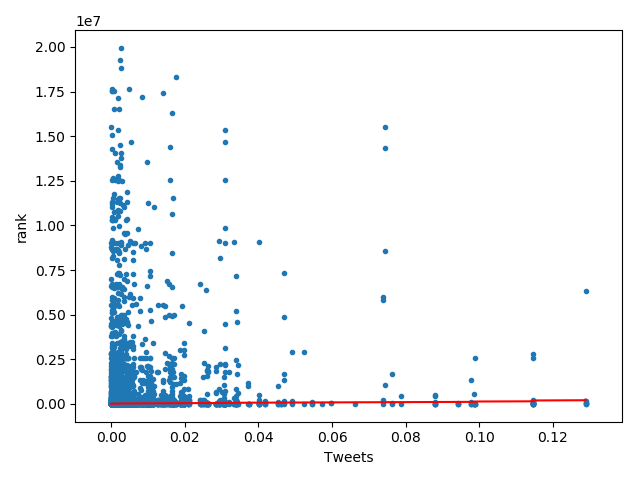
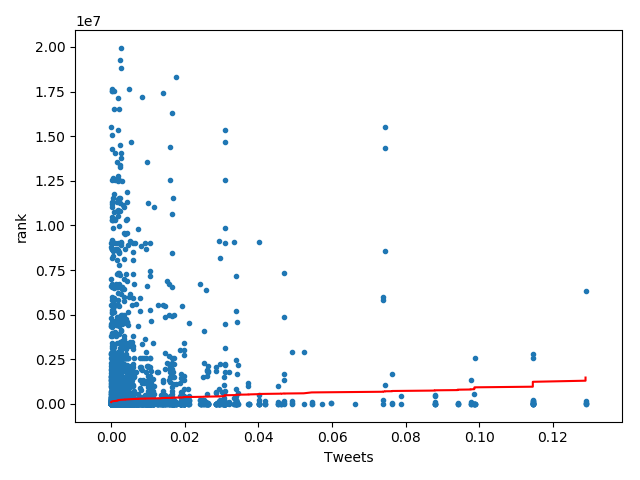
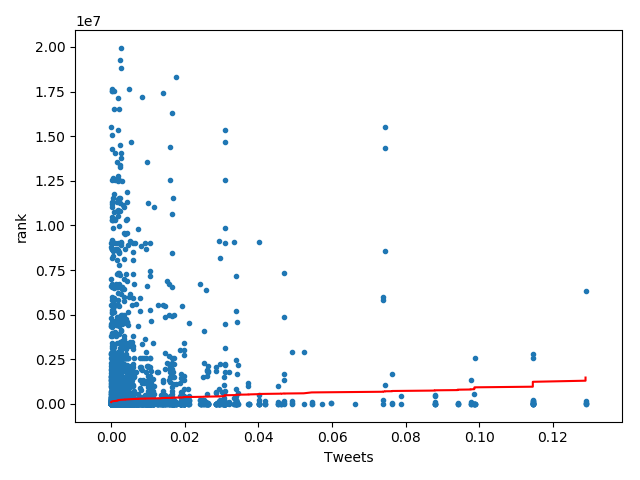
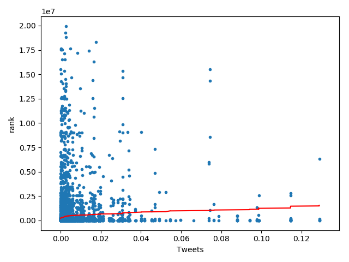
ابتدا مقدار صفر را برای ridge در نظر گرفته ام یعنی **W**LS ، شکل مدل در زیر قابل مشاهده است، گرچه مدل فوق روی روی فیچر های انتخابی عمل کرده و پلات ما فقط فیچر Tweets را نشان می‌دهد؛ اما مدل فیت شده تقریبا یک خط است و خطی بودن مدل اصلا منطقی نیست و انتظار منحنی را از مدل داشتم.



پیش‌تر دریافتیم که مهمترین فیچر ما Tweets است. شاید بد نبود قبل از lasso، یک چندجمله ایی درجه nام از این فیچر در مدل اضافه می کردیم.

در هر صورت از مدل مقابل انتظاری نمی رود که پیش بینی درستی داشته باشد. با این حال مقدار مناسب لاندا را باید پیدا کرد.

در ادامه روند اضافه شدن مقدار لاندا به ترتیب از چپ به راست نشان داده شده است.



از چنین مدلی که خیلی به خط شبیه است. انتظار نمی‌رود که مقادیر بالاتر لاندا، کمک شایانی به مدل کند. در ادامه بهترین لاندا را انتخاب می کنیم. با یک حلقه از رنج های لاگاریتمی .. مقدار لاندا 1.0 در نظر گرفته شده است. لازم به تاکید دوباره است این مقدار از روی داده های validation انتخاب شده است.

در ادامه خطاها را روی داده های آموزش و تست می توان دید:

* mse error on train data 6993063131468.775
* mse error on test data 6957695033217.962

خطای مورد انتظار و خوب روی تست داریم اما مدل جالبی نداریم. این خطا بسیار زیاد است.

برای نمونه دیتا پوینت شماره یک در داده های تست مقدار rank 2899683.0 را دارد و مقدار پیش بینی شده ما 779405.1 است. خطای فوق بسیار بالاست. در عمل این مدل بی‌استفاده است.

برای داده دوم هم مقدار پیش بینی شده 1024088.5 است و اختلاف فراوانی با مقدار اصلی که 671481.0 است دارد.

**مقادیر پیشینی شده را برای داده های Test اصلی ذخیره می‌کنیم:**

برای این امر دادهstudent\_test.xlsx را باز می کنیم. با مدلی که از ridge بدست آوردیم برای این داده ها پیشبنی را انجام داده و مقادیر را بعه عنوان یک دیتا فریم با ستون rank ذخیره می‌کنیم. در انتها توسط دستور زیر دو دیتا فریم را به هم متصل کرده و خروجی را ذخیره میکنیم.

* student\_test\_df = student\_test\_df.join(rank\_df)
* student\_test\_df.to\_excel(r'\ridge model.xlsx', index = None, header=True)

**ﺯ) ﺑﺎ ﺭﻭﺷﻬﺎﻱ forward stepwise ﻭ backward stepwise ﻓﻴﭽﺮﻫﺎ ﺭﺍ ﺑﻪ ﺗﺮﺗﻴﺐ ﺩﺍﺭﺍﻱ ﺍﻫﻤﻴﺖ ﺑﻮﺩﻥ**

**ﻣﺸﺨﺺ ﮐﻨﻴﺪ**.

با روش forward stepwise داریم:

['tweets', 'likes', 'reply', 'Followers', 'RTs', 'Favs', 'Listed']

همچنین با روش backward stepwise داریم:

['Listed', 'Favs', 'RTs', 'Followers', 'reply', 'likes']

**ﺡ) ﺍﻳﻦ ﻣﺴﺎﻟﻪ ﺭﺍ ﺑﺎ ﺭﻭﺷﻬﺎﻱ ﻏﻴﺮ ﭘﺎﺭﺍﻣﺘﺮﻳﮏ ﮐﻪ ﺩﺭ week 6 ﺭﮔﺮﺳﻴﻮﻥ ﻓﺮﺍ ﮔﺮﻓﺘﻪ ﺍﻳﺪ ﺣﻞ ﮐﻨﻴﺪ**

**1- ﺭﮔﺮﺳﻴﻮﻥ ﻧﺰﺩﻳﮑﺘﺮﻳﻦ ﻫﻤﺴﺎﻳﻪ**

بین روش های غیر پارمتریک که در بالا بیان شده صرفا مثالی میزنم و انتخاب اصلی مدل و K مناسب را در ادامه انجام خواهیم. داد. برای مثال در ابتدا رنک اولین فردی که در مجموعه تست وجود دارد را با روش 1NN حل میکنیم:

oneNN = lr.knn.k\_nearest\_neighbors(1, features, features\_test[0])  
print('Prediction by 1NN for the 1st person in the data set is:', output[oneNN])

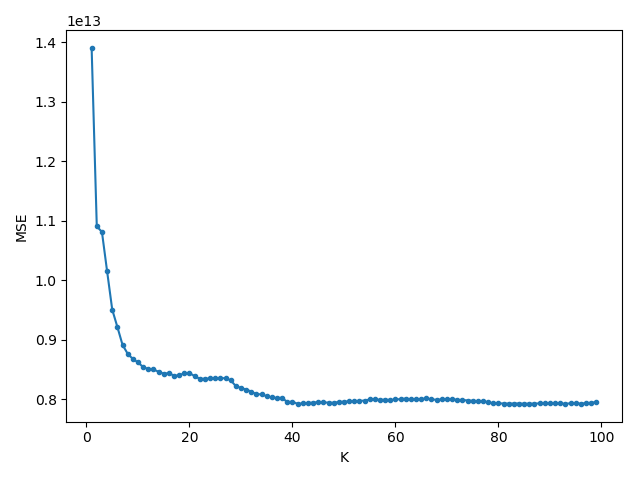
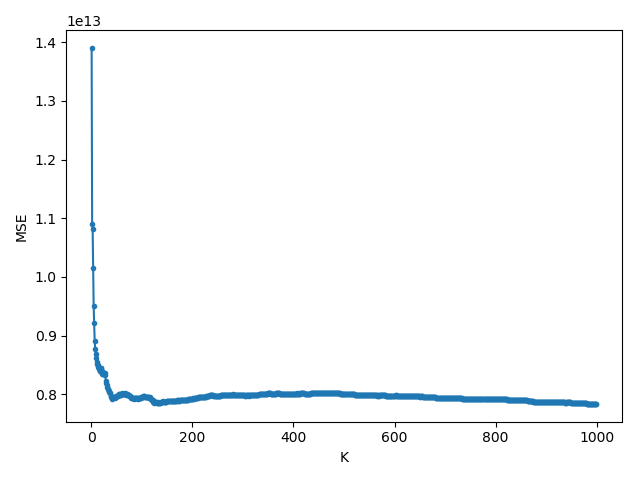
<<< Prediction by 1NN: 2046728.0

مقدار واقعی رنک: 2899683.0 .. فوق العاده ساده ... و کارآمد. در واقع مدل های غیرپارمتریک نزدیکترین همسایه به روش واقعی (کاری که انسان ها انجام می‌دهند) دارد عمل می‌کند، پس جای تعجب ندارد که حتی با 1NN هم پاسخ های به مراتب خیلی بهتری، نسبت به مدل قبلی میگیرم.

**2- ﺭﮔﺮﺳﻴﻮﻥ k ﻧﺰﺩﻳﮏ ﺗﺮﻳﻦ ﻫﻤﺴﺎﻳﻪ**

پس بریم به دنبال K مناسب، حلقه و داده‌ validation:

با در نظر گرفتن رنجی بین 1 تا 1000 برای مقدار k روی داده های validation داریم:



همانطور که از روی پلات پیداست، مقدار مناسب k عدد ۴۰ می‌باشد.

گرچه مقدار knn با 998 کمترین خطا را داده است، اما این مقدار از نظر محاسباتی بسیار بالاست. عدد 40 با اختلاف کمی در خطا، مناسب تر است.

mse error on test data 40NN 6515040035594.248

جالب است که در قیاس با مدل ریج خطای بیشتری را داریم. البته باز در هم ریخته اند و شاید دلیل این امر باشد.  
نکته ایی که مهم است این است که بعد از اینکه مقدار مناسب k را بدست آوردم از تمام داده های train و validation و حتی test در knn برای محاسبه مقادیر جدید که نیاز به پیشبینی دارند، استفاده می‌کنم. از نظر اینجانب استفاده از دادهای تست بعد از اینکه مدل انتخاب شده است. می تواند به بهبود پیشبینی مدل در آینده با داده هایی که تاکنون مدل ندیده است، کمک شایانی کند.

**kernel regression 3- ﺭﻭﺵ**

از انجا که کدی پیش از این برای kernel regression و یا غیر ننوشته ایم. از توابع از پیش نوشته شده sklearn.kernel\_ridge. KernelRidge استفاده می‌کنیم. هرچند تغییر کد knn هم زیاد سخت نیست اما خیلی هم ساده و بدون خطا نیست.

برای کرنل رگرسیون هم خطای 6918473152095 را داریم.

در آخر دو پیشبنی روی داده test\_student انجام شده است. یکی با نام KNN model و یکی با نام Ridge model. اگر معیار قضاوت برای هر دانشجو یک مدل قرار است باشد؛ مدل KNN را در نظر بگیرید. مدل کرنل رگرسیون به مراتب بهتر است و پاسخ بهتری می دهد، با این حال نتیجه آن را در فایل اکسل نیاوردم.

**نقد پروژه:**

در ابتدا که وقتی که صورت های سوال را خواندم به ظاهر خیلی ساده آمدند و سعی داشتم پروژه را از یک دید تجربی نگاه کنم که به آموخته های من می‌افزاید. اما با پیش رفتن در روند پروژه، فرآیند عذاب آور و طولانی شد. صورت های سوال که هر کدام بخشی را طلب می‌کردند ما را مجبور میساخت که بجای پیدا کردن یک مدل خوب به دنبال سر هم کردن پاسخی برای سوالات باشیم.

بخش پیش پردازش را دوست داشتم چرا که بسیار مفید بود اما زمان زیادی را در جستوجوی سوالات خود گذران کردم، شاید اگر دیگر منابع لازم مانند وارد کردن داده hdf که گذاشته شده بود، قرار داده میشد خیلی خیلی کمک کننده تر می‌بود. چرا که خیلی وقت ها پاسخ سوالات جلوی ما هستن اما به دلیل عدم آشنایی آنها را نمی‌بینیم.

همچنین بخشهایی از سوالات، قسمتهایی را درخواست می‌کرد که به هیچ عنوان در تمارین خواسته نشده بود و کدی برای آنها از قبل ننوشته بودیم.

حرف آخر اینکه، تاکید و اجرای تمارین در طول ترم بزرگترین و مهمترین قسمت یادگیری است، واقعا با وجود تمارین لازم به انجام یک پروژه سنگین نیست! از آنجا که سایه پروژه روی هفته هایی که باید خوانده شوند و تمرین هایی که باید کد زده شوند می افتد، عملا همه از فرآیند درسی جدا شده و به فکر نمره پروژه می‌افتند.

قدران زحمات بسیار شما

حسین اصحابی و تعدادی از دیگر دانشجویان