Case 3 Report

Web Application for Machine Learning on Blog Feedback Data Set

INFO 7390

Advanced Data Science & Architecture

Professor:

Srikanth Krishnamurthy

Students:

Team 2

Chenlian Xu

Qianli Ma

Date:

Apr 13th , 2018

TABLE OF CONTENTS

PART 1: MODEL DESIGN AND BUILDING

1.1 EXPLORATORY DATA ANALYSIS

1.1.1 VARIABLE DESCRIPTION

Variables Description	Number of features
Average, standard deviation, min, max and median of the Attributes 51...60 for the source of the current blog post	1...50
Total number of comments before basetime	51
Number of comments in the last 24 hours before the basetime	52
Let T1 denote the datetime 48 hours before basetime, Let T2 denote the datetime 24 hours before basetime. This attribute is the number of comments in the time period between T1 and T2	53
Number of comments in the first 24 hours after the publication of the blog post, but before basetime	54
The difference of Attribute 52 and Attribute 53	55
The same features as the attributes 51...55, but features 56...60 refer to the number of links (trackbacks), while features 51...55 refer to the number of comments	56...60
The length of time between the publication of the blog post and basetime	61
The length of the blog post	62
The 200 bag of words features for 200 frequent words of the text of the blog post	63...262
binary indicator features (0 or 1) for the weekday (Monday...Sunday) of the basetime	263...269
binary indicator features (0 or 1) for the weekday (Monday...Sunday) of the date of publication of the blog post	270...276
Number of parent pages: we consider a blog post P as a parent of blog post B, if B is a reply (trackback) to blog post P	277
Minimum, maximum, average number of comments that the parents received	278...280
The target: the number of comments in the next 24 hours (relative to basetime)	281

1.1.2 BASIC INFORMATION

```
df_train.head()
```

	0	1	2	3	4	5	6	7	8	9	...	271	272	273	274	275	276	277	278	279	280
0	40.30467	53.845657	0.0	401.0	15.0	15.52416	32.44188	0.0	377.0	3.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	40.30467	53.845657	0.0	401.0	15.0	15.52416	32.44188	0.0	377.0	3.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	40.30467	53.845657	0.0	401.0	15.0	15.52416	32.44188	0.0	377.0	3.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	40.30467	53.845657	0.0	401.0	15.0	15.52416	32.44188	0.0	377.0	3.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	40.30467	53.845657	0.0	401.0	15.0	15.52416	32.44188	0.0	377.0	3.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	27.0

5 rows × 281 columns

```
describe_table = df_train.describe().T
describe_table
```

	count	mean	std	min	25%	50%	75%	max
1	52397.0	39.444167	79.121821	0.000000	2.285714	10.630660	40.304670	1122.666600
2	52397.0	46.806717	62.359996	0.000000	5.214318	19.353120	77.442830	559.432600
3	52397.0	0.358914	6.840717	0.000000	0.000000	0.000000	0.000000	726.000000
4	52397.0	339.853102	441.430109	0.000000	29.000000	162.000000	478.000000	2044.000000
5	52397.0	24.681661	69.598976	0.000000	0.000000	4.000000	15.000000	1314.000000
6	52397.0	15.214611	32.251189	0.000000	0.891566	4.150685	15.998589	442.666660
7	52397.0	27.959159	38.584013	0.000000	3.075076	11.051215	45.701206	359.530060
8	52397.0	0.002748	0.131903	0.000000	0.000000	0.000000	0.000000	14.000000
9	52397.0	258.666030	321.348052	0.000000	22.000000	121.000000	387.000000	1424.000000
10	52397.0	5.829151	23.768317	0.000000	0.000000	1.000000	2.000000	588.000000
11	52397.0	14.053114	28.664559	0.000000	0.775000	3.817239	14.640625	438.000000

1.1.3 DIVIDE DATASET

For feature analysis, we divided the data set into four parts:

1.basic features: number of links and feedbacks in the previous 24 hours relative to baseTime; number of links and feedbacks in the time interval from 48 hours prior to baseTime to 24 hours prior to baseTime; how the number of links and feedbacks increased/decreased in the past (the past is seen relative to baseTime); number of links and feedbacks in the last 24 hours after the publication of the document, but before baseTime; aggregation of the above features by source.

2.textual features: the most discriminative bag of words features.

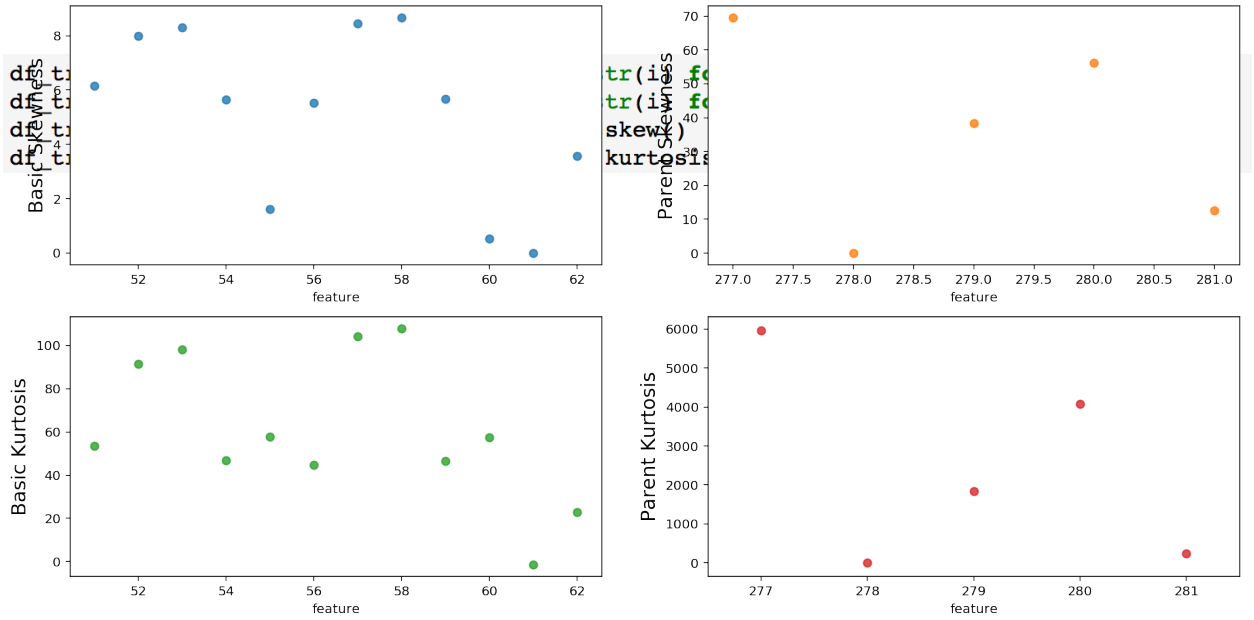
3.binary indicator features that describe on which day of the week the main text of the document was published and for which day of the week the prediction has to be calculated.

4.we consider a document dP as a patent of document d, if d is a reply to dP, i.e., there is a trackback link on dP that points to d; parent features are the number of parents, minimum, maximum and average number of feedbacks that the parents received.

1.1.4 SKEWED AND OUTLIERS

(1) SKEWNESS & KURTOSIS ANALYSIS

The skewness and kurtosis of basic and parent data except the aggregated data are analyzed in this part.



The picture shows that there are two features has high skewness and Kurtosis, which are Number of parent pages and the target.

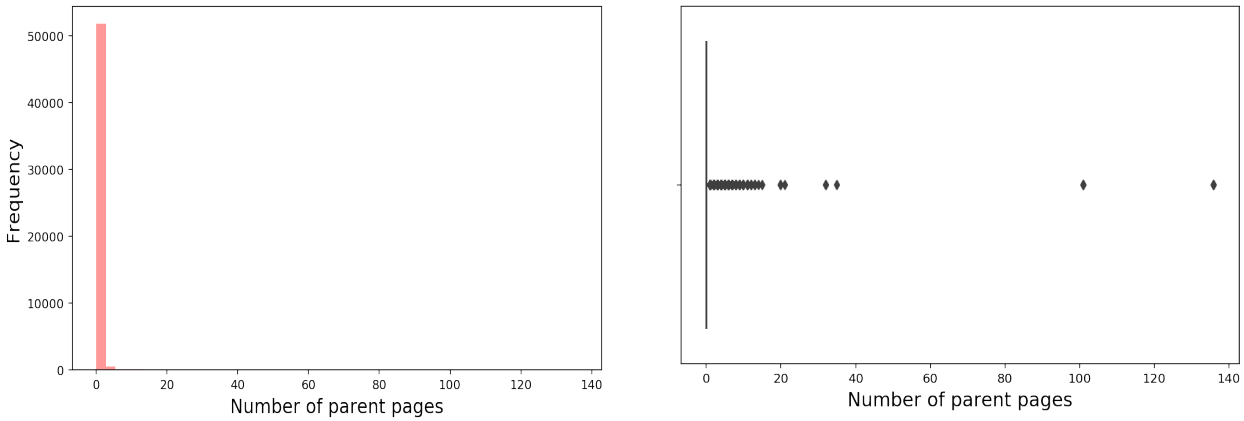
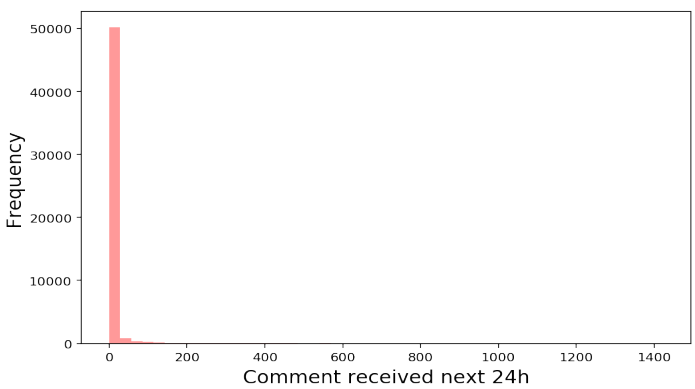
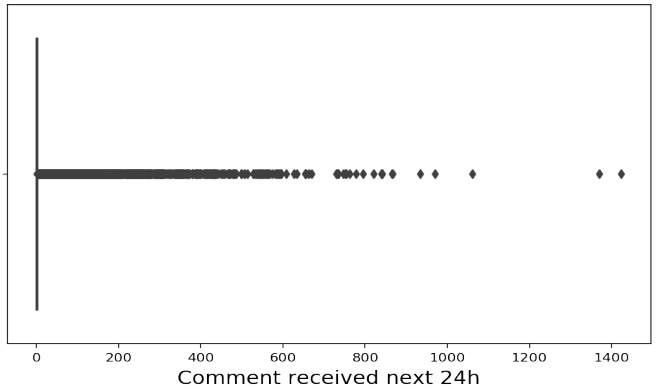


Figure are histogram and boxplot of Number of parent pages' distribution. The histogram shows the frequency of parent's page number in a certain interval. As we can see most of the data are 0. The long tail of the histogram means there are some large values with very low frequencies. The boxplot reveal that most data is around 0. Both of the graphs show that

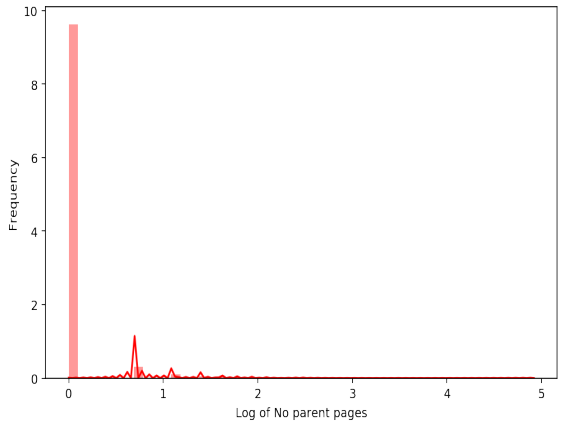


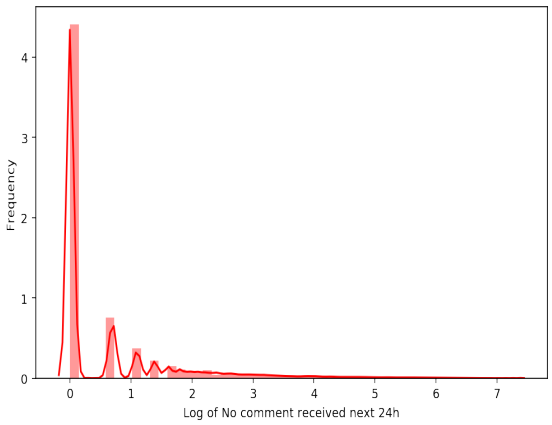


the data above the median value is more dispersed. There are many outliers appears in the boxplot which are the black rhombus over the upper whisker.

The above figure are histogram and boxplot of comment received next 24h distribution. The histogram shows the frequency of comment number in a certain interval. As we can see most of the data located between 0 and 200. The boxplot reveal that most data is around 0. Both of the graphs show that the data above the median value is more dispersed. There are many outliers appears in the boxplot which are the black rhombus over the upper whisker.

(2) LOG TRANSFORM





We apply log transform to these two features: number of parent pages, number of comment received next 24h. the picture X illustrates that log transform will make a variable become more normal.

(3) DETECT OUTLIERS

The definition of outlier is following:

Major Outlier = Upper Quartile + Interquartile Range

Minor Outlier = Lower Quartlie - Interquartile Range

Interquartile Range = 1.5 X (Upper Quartile - Lower Quartile)

There are four methods dealing with the outliers:

1.If it is obvious that the outlier is due to incorrectly entered or measured data, you should drop the outlier;

2.If the outlier does not change the results but does affect assumptions, you may drop the outlier. But note that in a footnote of your paper;

3.More commonly, the outlier affects both results and assumptions. In this situation, it is not legitimate to simply drop the outlier. You may run the analysis both with and without it, but you should state in at least a footnote the dropping of any such data points and how the results changed;

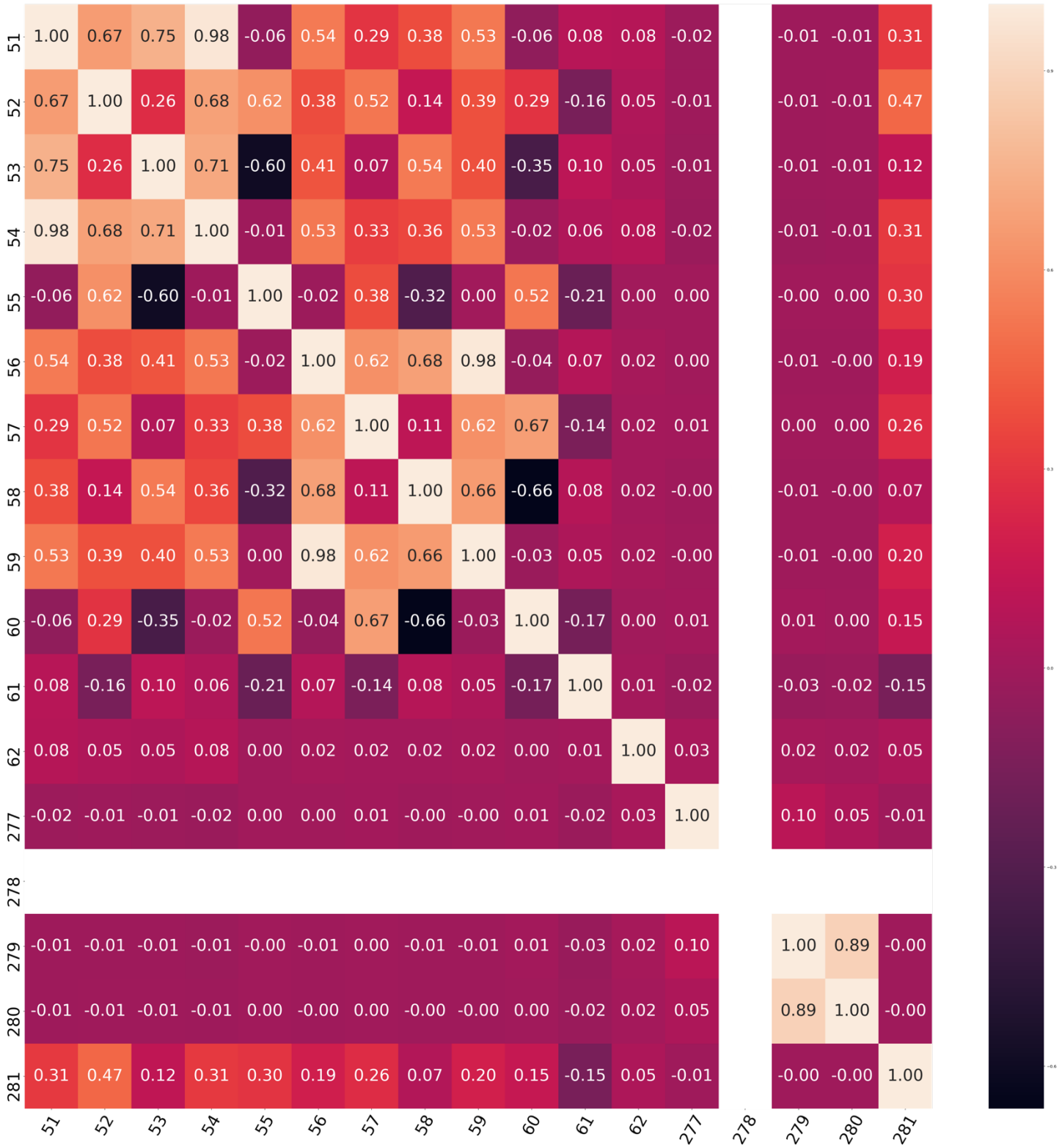
4.If the outlier creates a significant association, you should drop the outlier and should not report any significance from your analysis.

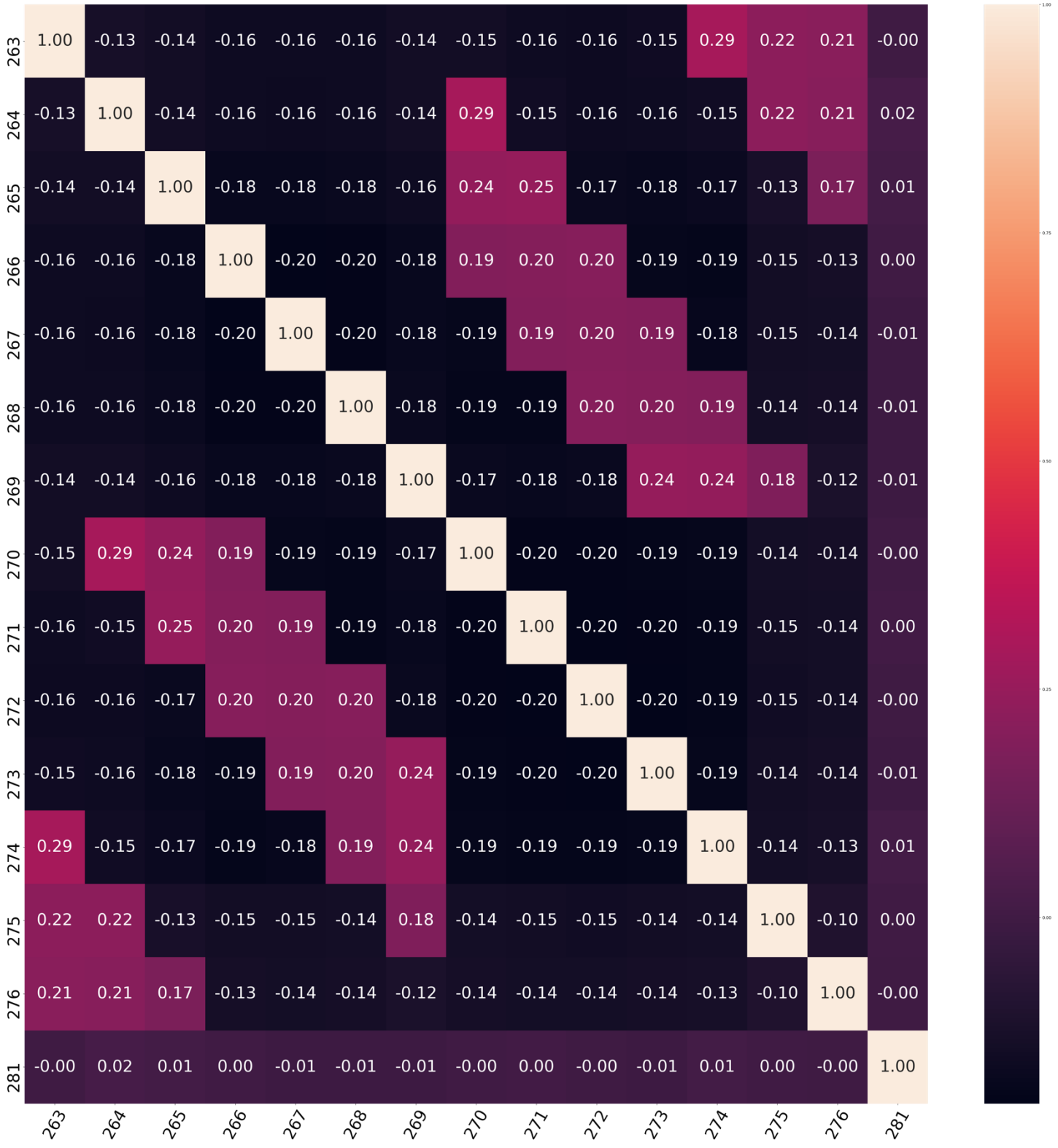
The basic data's outliers are detected in this part. The picture shows that there are many outliers in these columns.

```
for i in [str(i) for i in range(51,63)]:
    print(i,outlier_count(df_train, i))
```

```
51 (0, 8162)
52 (0, 8971)
53 (0, 9274)
54 (0, 8531)
55 (8789, 8262)
56 (0, 9736)
57 (0, 4729)
58 (0, 4203)
59 (0, 9122)
60 (3647, 3869)
61 (0, 0)
62 (0, 2295)
```

1.1.5 PAIR PLOT

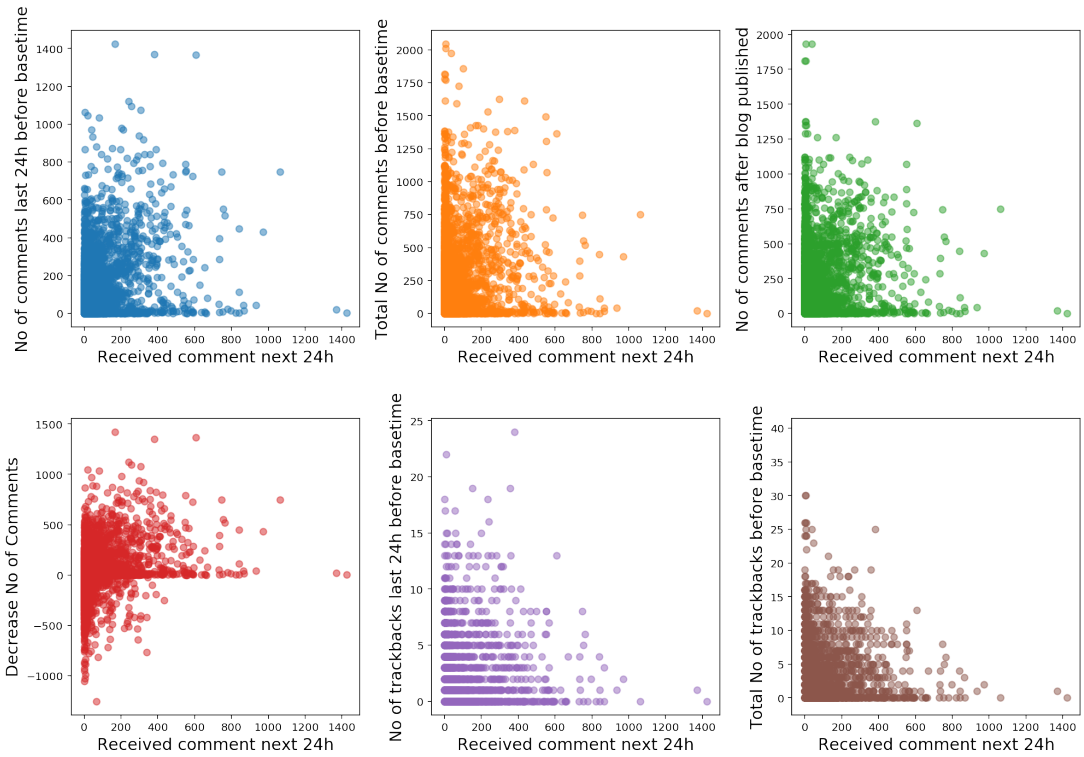



263 1.00 -0.13 -0.14 -0.16 -0.16 -0.16 -0.14 -0.15 -0.16 -0.16 -0.15 0.29 0.22 0.21 -0.00
264 -0.13 1.00 -0.14 -0.16 -0.16 -0.16 -0.14 0.29 -0.15 -0.16 -0.16 -0.15 0.22 0.21 0.02
265 -0.14 -0.14 1.00 -0.18 -0.18 -0.18 -0.16 0.24 0.25 -0.17 -0.18 -0.17 -0.13 0.17 0.01
266 -0.16 -0.16 -0.18 1.00 -0.20 -0.20 -0.18 0.19 0.20 0.20 -0.19 -0.19 -0.15 -0.13 0.00
267 -0.16 -0.16 -0.18 -0.20 1.00 -0.20 -0.18 -0.19 0.19 0.20 0.19 -0.18 -0.15 -0.14 -0.01
268 -0.16 -0.16 -0.18 -0.20 -0.20 1.00 -0.18 -0.19 -0.19 0.20 0.20 0.19 -0.14 -0.14 -0.01
269 -0.14 -0.14 -0.16 -0.18 -0.18 -0.18 1.00 -0.17 -0.18 -0.18 0.24 0.24 0.18 -0.12 -0.01
270 -0.15 0.29 0.24 0.19 -0.19 -0.19 -0.17 1.00 -0.20 -0.20 -0.19 -0.19 -0.14 -0.14 -0.00
271 -0.16 -0.15 0.25 0.20 0.19 -0.19 -0.18 -0.20 1.00 -0.20 -0.20 -0.19 -0.15 -0.14 0.00
272 -0.16 -0.16 -0.17 0.20 0.20 0.20 -0.18 -0.20 -0.20 1.00 -0.20 -0.19 -0.15 -0.14 -0.00
273 -0.15 -0.16 -0.18 -0.19 0.19 0.20 0.24 -0.19 -0.20 -0.20 1.00 -0.19 -0.14 -0.14 -0.01
274 0.29 -0.15 -0.17 -0.19 -0.18 0.19 0.24 -0.19 -0.19 -0.19 -0.19 1.00 -0.14 -0.13 0.01
275 0.22 0.22 -0.13 -0.15 -0.15 -0.14 0.18 -0.14 -0.15 -0.15 -0.14 -0.14 1.00 -0.10 0.00
276 0.21 0.21 0.17 -0.13 -0.14 -0.14 -0.12 -0.14 -0.14 -0.14 -0.14 -0.13 -0.10 1.00 -0.00
281 -0.00 0.02 0.01 0.00 -0.01 -0.01 -0.01 -0.00 0.00 -0.00 -0.01 0.01 0.00 -0.00 1.00
263 264 265 266 267 268 269 270 271 272 273 274 275 276 281
1.00
0.75
0.50
0.25
0.00

The figures are pair plots which drawn by seaborn. These plots could provide a better insight of the relationships between the variables with the target column. These Figures show the Pearson correlation which indicates that the basic features has higher correlation values than the parent features.

Number of features	Variable Description	corelation
52	Number of comments in the last 24 hours before the basetime	0.47
51	Total Number of comments before basetime	0.31
54	Number of comments after the blog published	0.31
55	Different between number of comments received in different time interval	0.30
57	Number of trackbacks in the last 24 hours before the basetime	0.26
56	Total Number of trackbacks before basetime	0.19
60	Different between number of trackbacks received in different time interval	0.15

The table shows the rank of Pearson Correlation between the target column and some variables the pair plots. Number of comments in last 24h before baseline has the highest correlation with the target column, and then total number of comments before baseline.



1.2 FEATURE ENGINEERING

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is an informal topic, but it is considered essential in applied machine learning.

In this part, feature engineering is applied to the dataset we described in the last part. Since the dataset has already been cleaned (no missing value) and there are no text features, no image features in the dataset.

1.2.1 DETECT AND DROP THE LOW VARIANCE FEATURES

In the EDA part, many columns' variance verified to be zero, they are dropped in the feature engineering.

```
selector = VarianceThreshold()
selector.fit_transform(df_train)
df_train.columns[~selector.get_support()]
```

```
Index(['13', '33', '38', '278'], dtype='object')
```

The features found with zero variance are following:

> Feature 13: min value of the comments received in mid 24h time interval of the source blog

> Feature 33: min value of the trackbacks received in last 24h before the basetime of the source blog

> Feature 38: min value of the trackbacks received in mid 24h time interval of the source blog

> Feature 278: min value of the comments parent page received

1.2.2 DROP COLUMNS THAT ARE HIGHLY CORRELATED

Blog feedback data sets has hundreds of features, but many of them are high correlated. In this part, columns that are highly correlated is dropped.

1.3 FEATURE SELECTION & MODEL VALIDATION

1.3.1 CREATE ERROR METRICS CLASS

Error Metrics class is created in advance to compute the different error metrics of each model. RMS, MAPE, R^2 and MAE are included in the function to measure the prediction performance of each model.

Linear regression, random forest and neural network are used here to fit the data set. For random forest, number of estimators are randomly picked as 100.

```
class ErrorMetrics:

    def __init__(self, X_train, y_train, X_test, y_test):
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.error_metric = pd.DataFrame({'rmse_train': [],
                                          'rmse_test': [],
                                          'mae_train': [],
                                          'mae_test': [],
                                          'mape_train': [],
                                          'mape_test': [],
                                          'r_train': [],
```

```
def cal_metric(self, modelname, model):
    y_train_pre = model.predict(self.X_train)
    y_test_pre = model.predict(self.X_test)

    rmse_train = math.sqrt(mean_squared_error(self.y_train, y_train_pre))
    rmse_test = math.sqrt(mean_squared_error(self.y_test, y_test_pre))

    mae_train = mean_absolute_error(self.y_train, y_train_pre)
    mae_test = mean_absolute_error(self.y_test, y_test_pre)

    mape_train = np.mean(np.abs((self.y_train - y_train_pre) / self.y_train)) * 100
    mape_test = np.mean(np.abs((self.y_test - y_test_pre) / self.y_test)) * 100

    r_train = r2_score(self.y_train, y_train_pre)
    r_test = r2_score(self.y_test, y_test_pre)

    error_metric_local = pd.DataFrame({'Model': [modelname],
                                       'rmse_train': [rmse_train],
                                       'rmse_test': [rmse_test],
                                       'mae_train': [mae_train],
                                       'mae_test': [mae_test],
                                       'mape_train': [mape_train],
                                       'mape_test': [mape_test],
                                       'r_train': [r_train],
                                       'r_test': [r_test]})

    if self.error_metric.columns[0] == 'Model' and not self.error_metric.loc[self.error_metric['Model'] == modelname].empty:
        self.error_metric = self.error_metric[self.error_metric.Model != modelname]
    else:
        pass

    self.error_metric = pd.concat([self.error_metric, error_metric_local])
```

1.3.2 RANDOM FOREST MODEL FEATURE SELECTION

(1) BENCHMARK RANDOM FOREST

First train the benchmark random forest model

```
rf_b = RandomForestRegressor()
with parallel_backend('dask.distributed'):
    rf_b.fit(X_train, y_train)
```

(2) BORUTA FEATURE SELECTION

Packages used: Boruta

Use Boruta to select features, Boruta keeps 6 features. Train a new random forest model with these 6 features.

```
X_boruta = df_train.loc[:, df_train.columns != '281'].values
y_boruta = df_train['281'].values
rf_regressor = RandomForestRegressor(n_estimators=100, n_jobs=2)
feat_selector = BorutaPy(rf_regressor, n_estimators='auto', verbose=2)
feat_selector.fit(X_boruta,y_boruta)
```

```
X_train_rf = df_train.loc[:,feat_selector.support_]
X_test_rf = df_test.loc[:,feat_selector.support_]
y_train = df_train['281']
y_test = df_test['281']
```

```
rf = RandomForestRegressor()
rf.fit(X_train_rf, y_train)
```

(3) TREE IMPORTANCE FEATURE

Packages used: sklenarn.ensemble, RandomForestReressor

Random forest regression is a method which use the ensembles of decision tree, which can calculate the relative importance of the features in dataset, thus we use this property to evaluate the importance in this part. The outcome shows that there are many features' importance is zero. Building a model with the features that has a feature importance greater than 0.

```
ft_tree = pd.DataFrame(feature_importances)
ft_tree.columns = ['feature','importance']
feat_list = ft_tree[ft_tree.importance > 0]['feature'].values
X_train_tree = df_train.loc[:, feat_list]
X_test_tree = df_test.loc[:,feat_list]
```

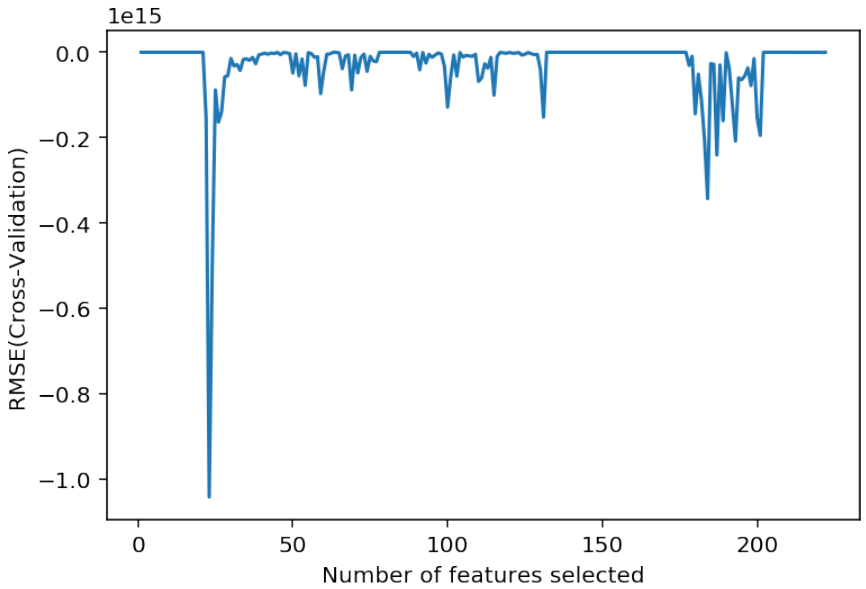
```
pd.concat([error_metrics_rf_b.error_metric, error_metrics_tree.error_metric,error_metrics_rf.error_metric], axis=0)
```

	Model	mae_test	mae_train	mape_test	mape_train	r_test	r_train	rmse_test	rmse_train
0	rf_benchmark	5.395912	2.124367	inf	inf	0.365769	0.902819	24.289692	11.754497
0	tree_rf	5.389772	2.088427	inf	inf	0.350078	0.910594	24.588321	11.274456
0	boruta_rf	5.247830	2.190253	inf	inf	0.435653	0.902168	22.912446	11.793793

The error metrics shows that using features selected by the Boruta can get lowest RMSE.

1.3.3 LINEAR MODEL FEATURE SELECTION

REF Feature Selection



The picture shows that RMSE value is a straight line parallel to the X axis except some destabilization.

1.3.4 NEURAL NETWORK FEATURE SELECTION

(1) BENCHMARK FEATURE SELECTION

:
```
def report(grid_scores, n_top=3):
    top_scores = sorted(grid_scores, key=itemgetter(1), reverse=True)[:n_top]
    for i, score in enumerate(top_scores):
        print("Model with rank: {0}".format(i + 1))
        print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
              score.mean_validation_score,
              np.std(score.cv_validation_scores)))
        print("Parameters: {0}".format(score.parameters))
        print("")

mlp = MLPRegressor(hidden_layer_sizes=(50,), max_iter=10000,batch_size=600,
                   solver='sgd',  random_state=1,activation="logistic",learning_rate="constant")

mlp.fit(X_train, y_train)


print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

(2) GRID SEARCH FEATURE SELECTION

```
param_grid = { "alpha":[.1,.01,.01],
               "momentum":[.01,.05,0.1],
               "learning_rate_init":[.1,.01,.01]
}
mlp_grid= MLPRegressor(hidden_layer_sizes=(50,), max_iter=10000,batch_size=600,
                   solver='sgd',  random_state=1,activation="logistic",learning_rate="constant")

random_search = RandomizedSearchCV(mlp_grid, param_distributions= param_grid,n_iter = 8, cv=5)
random_search.fit(X_train.values, y_train.values)
```

```
RandomizedSearchCV(cv=5, error_score='raise',
          estimator=MLPRegressor(activation='logistic', alpha=0.0001, batch_size=600, beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(50,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=10000, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='sgd', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False),
          fit_params=None, iid=True, n_iter=8, n_jobs=1,
          param_distributions={'alpha': [0.1, 0.01, 0.01], 'momentum': [0.01, 0.05, 0.1], 'learning_rate_init': [0.1,
0.01, 0.01]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

```
pd.concat([error_metrics_nn_opt.error_metric,error_metrics_nn.error_metric])
```

	Model	mae_test	mae_train	mape_test	mape_train	r_test	r_train	rmse_test	rmse_train
0	optimal neural network	9.154931	10.899370	inf	inf	0.005573	0.002941	30.414787	37.650720
0	neural network	10.046736	11.646153	inf	inf	0.002695	0.002315	30.458771	37.662533

1.4 FINAL PIPELINE

Airflow is a platform to programmatically author, schedule and monitor workflows.

Use airflow to author workflows as directed acyclic graphs (DAGs) of tasks. The airflow scheduler executes your tasks on an array of workers while following the specified dependencies. Rich command line utilities make performing complex surgeries on DAGs a snap. The rich user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed.

When workflows are defined as code, they become more maintainable, versionable, testable, and collaborative.

1.4.1 CLEAN DIRECTORY

Clean the directory, every file in this project will located in this directory.

```
def clean_dir(download_dir,**kwargs):
    if not os.path.exists(download_dir):
        os.makedirs(download_dir, mode=0o777)
    else:
        shutil.rmtree(os.path.join(os.path.dirname(__file__), download_dir), ig
        os.makedirs(download_dir, mode=0o777)
```

5.1 Download zip file from s3 and unzip file

```
def download_zip(url, zip_dir, unzipped_dir, **kwargs):
    resp = requests.get(url)
    open(zip_dir, 'wb').write(resp.content)
    zip_ref = zipfile.ZipFile(zip_dir, 'r')
    for file in zip_ref.namelist():
        zip_ref.extract(file, unzipped_dir)
    zip_ref.close()
```

1.4.2 READ DATAFRAME

Read csv files into pandas dataframe.

```
def read_df(unzipped_dir, **kwargs):
    filename_test = glob.glob(unzipped_dir + '/*test*.csv')
    list_ = []
    for file in filename_test:
        df = pd.read_csv(file, header=None)
        list_.append(df)
    df_test = pd.concat(list_)
    df_train = pd.read_csv(unzipped_dir + '/blogData_train.csv',header=None)
    df_train.columns = [str(i) for i in range(1,282)]
    df_test.columns = [str(i) for i in range(1,282)]
    return df_train, df_test
```

1.4.3FUTURE ENGINEER

After reading data, drop variable with low variances and drop variable that are highly correlated.

```
def feture_engineer(**kwargs):

    #drop variable with low variance
    df_train, df_test = kwargs['ti'].xcom_pull(task_ids='read')
    selector = VarianceThreshold()
    selector.fit_transform(df_train)
    #create mask for var
    mask_var = selector.get_support()

    #drop variable that are highly correlated
    corr_matrix = df_train.corr().abs()
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astyp
    to_drop = [column for column in upper.columns if any(upper[column] > 0.9
    #create mask for cor
    mask_corr = ~np.in1d(df_train.columns, to_drop)
    not_drop = df_train.columns[np.logical_and(mask_corr,mask_var)]
    drop = df_train.columns[~np.logical_and(mask_corr,mask_var)]

    df_train = df_train.drop(drop, axis=1)
    df_test = df_test.drop(drop, axis=1)
    with open(blog_dir + "/not_drop.pkl", "wb") as fp:
        pickle.dump(not_drop, fp, protocol = 2)
    return df_train, df_test
```

1.4.4 TRAIN MODEL

Train tree model: 1. Linear model, 2. Random forest model, 3. Neural Network. After training, pickle all model and save at directory specified.

```
def train_model(**kwargs):
    #train linear model and pickle
    df_train, df_test = kwargs['ti'].xcom_pull(task_ids='feature_engineer')
    X_train = df_train.loc[:, df_train.columns != '281']
    y_train = df_train['281']
    X_test = df_test.loc[:,df_test.columns != '281']
    y_test = df_test['281']


    #train linear model and pickle
    lm_index = X_train.columns.tolist()

    lm = LinearRegression()
    lm.fit(X_train, y_train)
    cal_metric('linear', lm ,X_train, y_train, X_test, y_test)
    filename_l = blog_dir + '/finalized_linear_model.pkl'
    pickle.dump(lm,open(filename_l,'wb'), protocol = 2)

    lm_index_file = blog_dir + '/lm_index.pkl'
    with open(lm_index_file, "wb") as fp:
        pickle.dump(lm_index, fp, protocol = 2)
```

```
#train random forest and pickle
tree_index = ['1','2','52','55','61','62']
rf = RandomForestRegressor()
X_train_rf = X_train[tree_index]
X_test_rf = X_test[tree_index]
rf.fit(X_train_rf, y_train)
cal_metric('random forest', rf ,X_train_rf, y_train, X_test_rf, y_test)
filename_rf = blog_dir + '/finalized_random_forest_model.pkl'
with open(filename_rf, 'wb') as fp:
    pickle.dump(rf,fp, protocol = 2)

tree_index_file = blog_dir + '/tree_index.pkl'
with open(tree_index_file, "wb") as fp:
    pickle.dump(tree_index, fp, protocol = 2)


#train nn
nn_index = X_train.columns.tolist()
nn = MLPRegressor(activation='logistic', alpha=0.1, batch_size=600, beta_1=
   beta_2=0.999, early_stopping=False, epsilon=1e-08,
   hidden_layer_sizes=(50,), learning_rate='constant',
   learning_rate_init=0.1, max_iter=10000, momentum=0.01,
   nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
   solver='sgd', tol=0.0001, validation_fraction=0.1, verbose=False,
   warm_start=False)

nn.fit(X_train, y_train)
cal_metric('network', nn ,X_train, y_train, X_test, y_test)
filename_nn = blog_dir + '/finalized_neural_network_model.pkl'
pickle.dump(nn,open(filename_nn,'wb'), protocol = 2)

nn_index_file = blog_dir + '/nn_index.pkl'
with open(nn_index_file, "wb") as fp:
    pickle.dump(nn_index, fp, protocol = 2)


##save error_metrics

error_metric.to_csv(blog_dir + '/error_metrics.csv', index = False)
```

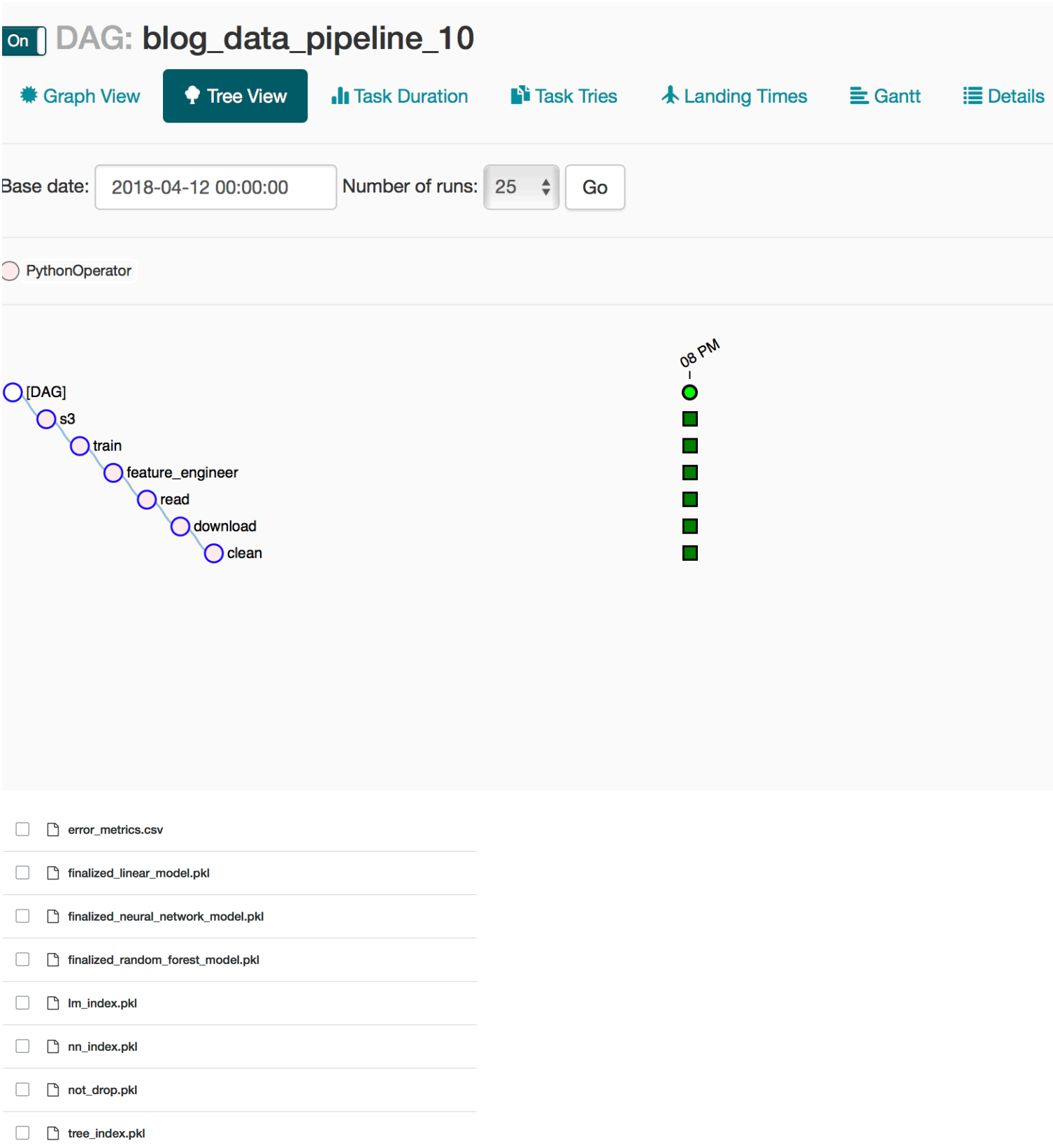
1.4.5 UPLOAD MODEL AND ERROR METRICS TO S3

```
def connect():
    accessKey = 'your key'
    secretAccessKey='your secret key'
    AWS_ACCESS_KEY_ID = accessKey
    AWS_SECRET_ACCESS_KEY = secretAccessKey

    try:
        error_file = blog_dir + '/error_metrics.csv'
        linear_file = blog_dir + '/finalized_linear_model.pkl'
        random_file = blog_dir + '/finalized_random_forest_model.pkl'
        nn_file = blog_dir + '/finalized_neural_network_model.pkl'
        not_drop_file = blog_dir + '/not_drop.pkl'
        tree_index_file = blog_dir + '/tree_index.pkl'
        lm_index_file = blog_dir + '/lm_index.pkl'
        nn_index_file = blog_dir + '/nn_index.pkl'



        bucket_name ='pipelinefiletest7'+ AWS_ACCESS_KEY_ID.lower() +  time.str
        conn = boto.connect_s3(AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY)
        bucket = conn.create_bucket(bucket_name, location=Location.DEFAULT)
        print ('bucket created')

        k1 = Key(bucket)
        k1.key = 'finalized_linear_model.pkl'
        k1.set_contents_from_filename(linear_file)

        k2 = Key(bucket)
        k2.key = 'finalized_random_forest_model.pkl'
        k2.set_contents_from_filename(random_file)


        k3 = Key(bucket)
        k3.key = 'finalized_neural_network_model.pkl'
        k3.set_contents_from_filename(nn_file)


        k4 = Key(bucket)
        k4.key = 'index.pkl'
        k4.set_contents_from_filename(not_drop_file)


        k5 = Key(bucket)
```

1.4.6 CREATE DAG

Create dag use PythonOperator, and define the relationship between the tasks.

```
args = {
    'owner': 'airflow',
    'start_date': airflow.utils.dates.days_ago(2)
}


dag = DAG('blog_data_pipeline_10', default_args=args, schedule_interval= '@once'

t0 = PythonOperator(
    task_id='clean',
    python_callable=clean_dir,
    provide_context = True,
    op_kwargs = {'download_dir':blog_dir},
    dag=dag)

t1 = PythonOperator(
    task_id='download',
    python_callable=download_zip,
    provide_context = True,
    op_kwargs = {'url':url_, 'zip_dir':zip_dir_, 'unzipped_dir':unzipped_dir_},
    dag=dag)

t2 = PythonOperator(
    task_id='read',
    python_callable=read_df,
    provide_context = True,
    op_kwargs = {'unzipped_dir':unzipped_dir_},
    dag=dag)

t3 = PythonOperator(
    task_id='feature_engineer',
    python_callable=feture_engineer,
    provide_context = True,
    dag=dag)
```



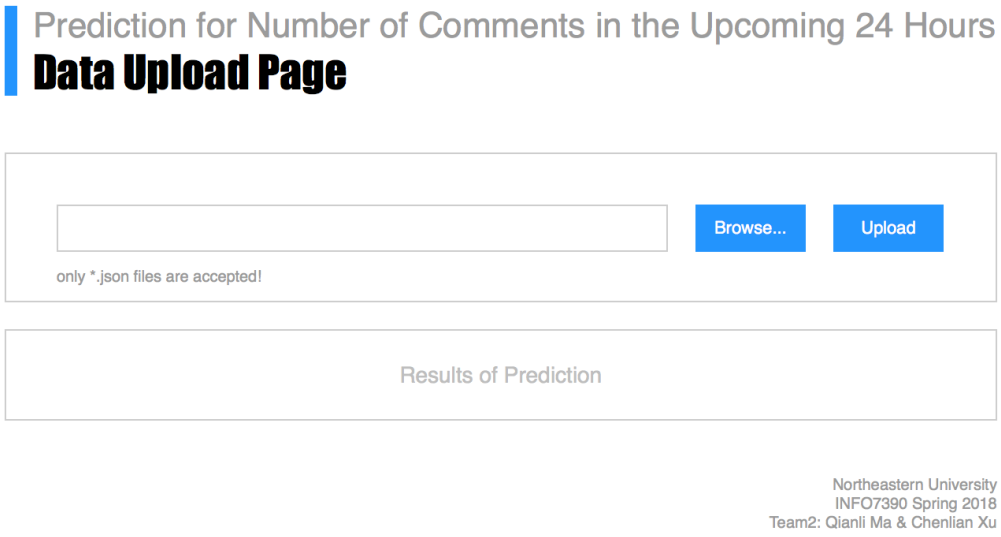
The picture shows that the model was successfully finished and our amazon s3 bucket has all the files needed.

PART 2: MODEL DEPLOYMENT

2.1 LOCAL FLASK APPLICATION DESIGN

Firstly, it requires us to create a web application using Flask. Our application basically built upon three functions:

```
@app.route('/', methods=['GET'])
def upload_file():
    return render_template('upload.html')
```

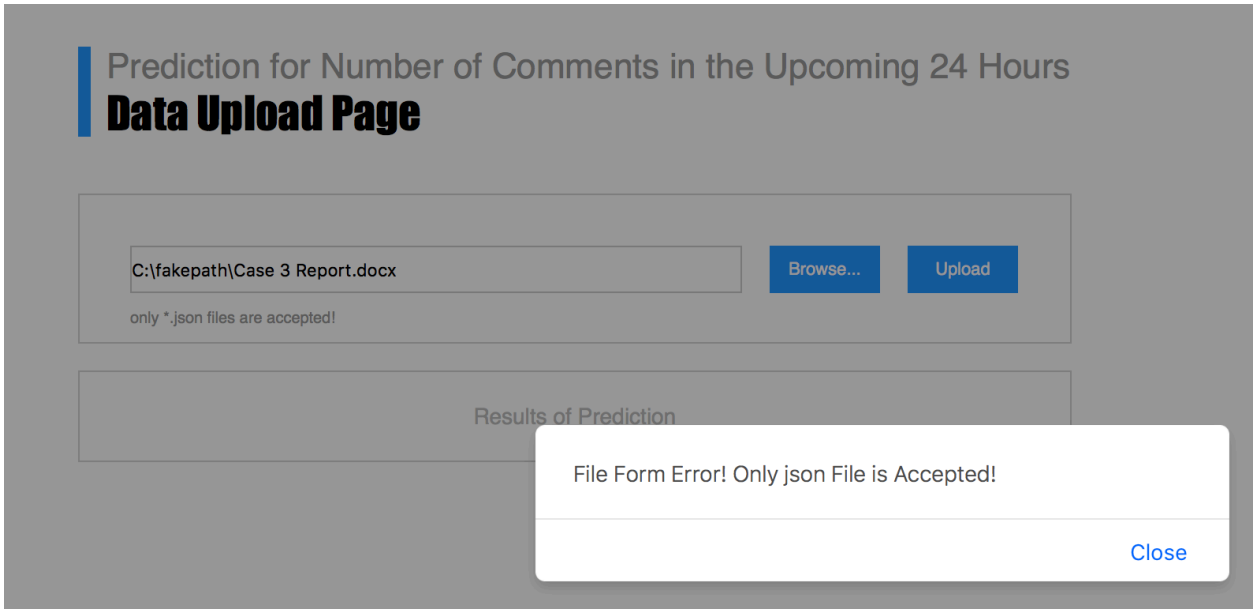


We return the data upload page to user at the very beginning when user send “GET” request to the server. In this stage, the application want to receive the “.json” file from user for prediction.

```
@app.route('/', methods=['POST'])
def main():
    try:
        file_uploaded = request.files['upload_file']
        preview_parameter = data_prediction.data_processing(file_uploaded, UPLOAD_FOLDER)
        output_path = data_prediction.form_download_file(OUTPUT_FOLDER, preview_parameter[0], preview_parameter[1])
        return render_template('download.html', output_column=preview_parameter[0], output_row=preview_parameter[1],
                               total_rows=preview_parameter[2], output_path=output_path)
    except BaseError as e:
        print e.message
        setLogger().exception(e.message)
        raise BaseError(code=e.code, message=e.message)
```

After user uploading a file, the application will return the download page to user. In this page, user could preview the prediction results, download the ".csv" file of results and upload another file for prediction as well.

```
@app.errorhandler(500)
def internal_server_error(e):
    return render_template("error.html", message=e.message)
```



The last function is for handling the error in the application like no file uploaded, uploaded file is not ".json" or cannot be predicted by the pickled model and so on. After alarming the error, the application will return a page just like the first one for file uploading.

```
class BaseError(Exception):
    def __init__(self, code, message):
        self.code = code
        self.message = message
```

A class named “BaseError” is created for alarming all kinds of errors which could happen when user upload the data. The application will return specific pages with alarm according to the message that “BaseError” says. This part is actually the top level code which is comparatively abstract and will not show the detail of how the application itself do the prediction, form the download file and capture the exceptions.

2.2 WEB APPLICATION DEPLOYMENT

To deploy the application online, a server is registered on DigitalOcean. This is pretty much like rebuild the whole coding environment on another system. Depended packages need to be installed and Amazon AWS configuration need to be done as well, just like what we do in the virtual environment on our laptop. Here simply using “pip install awscli” to call the “aws configure” command. Then type in the access key, secret key in the command line. The url for the application is http://159.65.231.188:5000.

```
[root@neu-info7390-2018spring-team2 ~]# aws configure
```

```
AWS Access Key ID [None]: AKI
AWS Secret Access Key [None]:
Default region name [None]: us
```

2.3 DATA INPUT/OUTPUT

The application allows user to upload single data for prediction as well as a batch of data.

```
file_uploaded_name = secure_filename(input_file.filename)
suffix = file_uploaded_name.rsplit('.', 1)[1]
new_filename = str(local_time) + '.' + suffix
data = pd.read_json(input_file, typ='frame', numpy=True, orient='records')
total_rows = data.shape[0]
```

```
for i in range(0, total_rows):
    output_row.append([targets[0][i], targets[1][i], targets[2][i]])
```

The code is actually general for these two situation since single data is just a special case of a batch of data (“total_rows” equals to 1). After reading the json file user uploaded, the application also stores the file in a fold. From here we actually reach to the bottom level of the code which shows evert detail of the logics and implementations.

Recall the second function in the top level code, after data processing, a function named “form_download_file” is called to prepare the csv file for user.

```
preview_parameter = data_prediction.data_processing(file_uploaded, UPLOAD_FOLDER)
output_path = data_prediction.form_download_file(OUTPUT_FOLDER, preview_parameter[0], preview_parameter[1])
```

```
def form_download_file(output_folder, output_column, output_row):
    try:
        output_filename = str(local_time) + '_result.' + 'csv'
        output_path = os.path.join(output_folder, output_filename)
        download_file = pd.DataFrame(columns=output_column, data=output_row)
        download_file.to_csv(output_path)
        return output_path
    except:
        raise BaseError(code=500, message="Fail to Form Download File!")
```

Also, a table is formed in the web page for previewing the prediction results. The front-end code looks like below: it receives output columns and rows from the back-end and simply uses a loop to print them in one table.

Finally, all the uploaded files and download files are stored in the project repository: in the “upload” fold and “./static/output” fold.

```
(venv_a3) [root@neu-info7390-2018spring-team2 a3_web_application]# ls
common           data_prediction.py   __init__.py  prediction_application.py       __pycache__       static     upload_files
dask-worker-space  data_prediction.pyc  log.txt      prediction_application.py.save  requirements.txt  templates
```

```
<div style="...">
    <table border="1" width="588" class="preview_table">
        <tr>
            <th colspan="3"> <div align="center" style="...">Prediction Results</div></th>
        </tr>
        <tr>
            <th><div align="center" style="...">{{output_column[0]}}</div></th>
            <th><div align="center" style="...">{{output_column[1]}}</div></th>
            <th><div align="center" style="...">{{output_column[2]}}</div></th>
        </tr>
        {% for i in range(0, total_rows) %}
        <tr>
            <td><div align="center" style="...">{{output_row[i][0]}}</div></td>
            <td><div align="center" style="...">{{output_row[i][1]}}</div></td>
            <td><div align="center" style="...">{{output_row[i][2]}}</div></td>
        </tr>
        {% endfor %}
    </table>
</div>
```

2.4 UNPICKLE MODELS AND PARAMETERS FROM S3

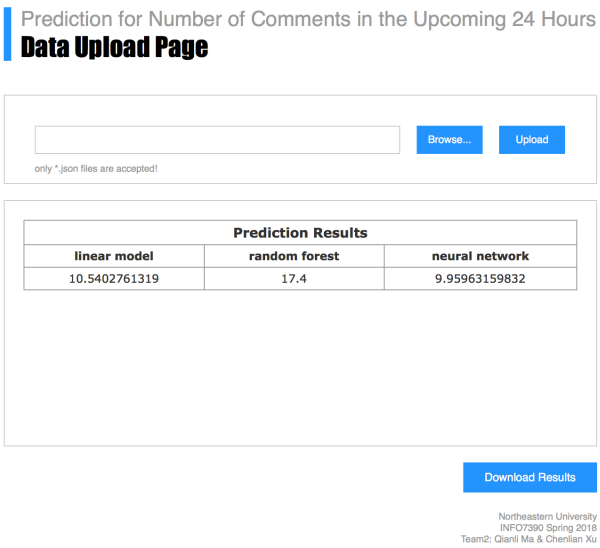
```
try:
    S3 = boto3.client('s3', region_name='us-east-1')
except:
    raise BaseError(code=500, message="Fail to connect to S3!")
```

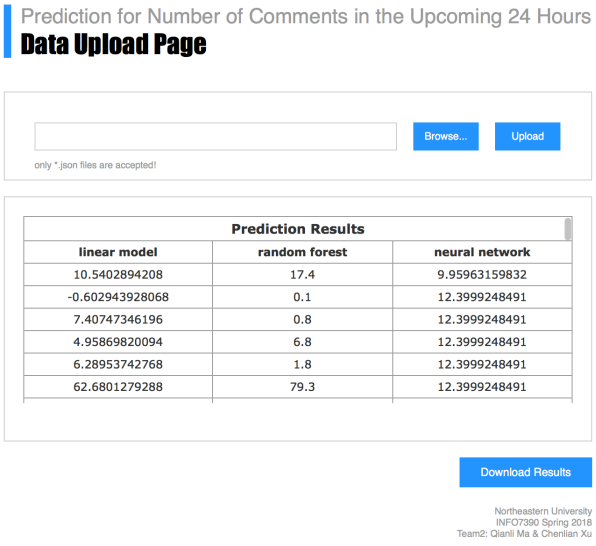
```python
def load_model(key):
    try:
        # Load model from S3 bucket
        response = S3.get_object(Bucket=BUCKET_NAME, Key=key)
        # Load pickle model
        model_str = response['Body'].read()
        model = pickle.loads(model_str)
        return model
    except:
        raise BaseError(code=500, message="Fail to Load Model!")
```

2.5 USE CASE

As the picture shows, the application is successfully predict for the uploaded data both on single data and batch data.

These two test data sets all also store in the "./static" fold for reusing.





2.6 CREATE JUPYTER NOTEBOOK USING REPO2DOCKER:

```
    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://0.0.0.0:59396/?token=332cad1ff9d8c703d5c2eec47f19b94b8446964ee1cb
783c
```

2.7 DISTRIBUTED COMPUTATION

If user uploads a file which contains more than 10 data rows, the application will create a “Dask” cluster and schedule the computation work (predicting the data) to the workers.

```
if total_rows > 10:
    client = Client(processes=False)
    prediction = client.submit(model.predict, data_).result().tolist()
else:
    prediction = model.predict(data_).tolist()
```