ABHINAV SRIVASTAVA Data Analytics Scientist (GE Power Digital)

Aconex Assignment

Code in Python 3:

Since project duration is of prime importance we need a measure which can be taken as a proxy for project duration. We will leverage the mail **sentDate** information present in each project. The first and last mail sent on a project can be taken as a measure of the project duration. In each sorted (by date) project dataframe, the difference of first and last entry will give information of project duration **[calculated in days]** for that specific project.

As the variations seen on projects cost money in general we need to extract information about these variations as well. The potential variations are:

- The number of Organizations taking part in project. Large number of participating organizations comes with inherent variations and communication delays. We can extract the number of participating organizations **fromOrganizationId**.
- The number of people involved in a project. This information is available in form of **fromUserID.**
- Kind of communication happening on mails. This information can be extracted from **correspondencetypeId**.

We can repeat this process for each project dataframe, and create a blank dataframe and keep adding these in each row. We will also keep track of the number of mails sent on a project.

The head of dataframe created is sown below : `[3723 rows x 6 columns]`

Out[3]:

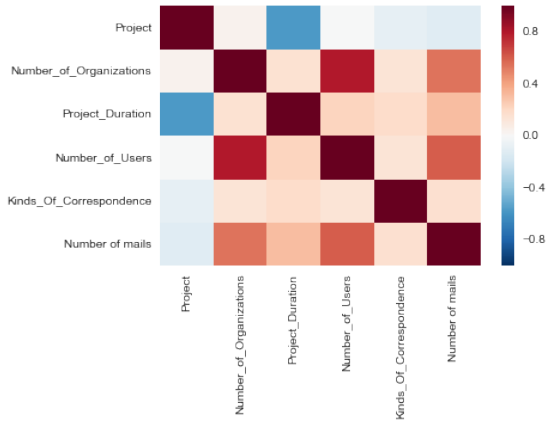
	Project	Number_of_Organizations	Project_Duration	Number_of_Users	Kinds_Of_Correspondence	Number of mails
0	10417	28	1505	67	35	2008
1	10664	23	1805	62	38	11631
2	10769	679	5761	1923	235	20089
3	134217739	39	1189	78	17	3208
4	134217742	76	1748	199	32	9143

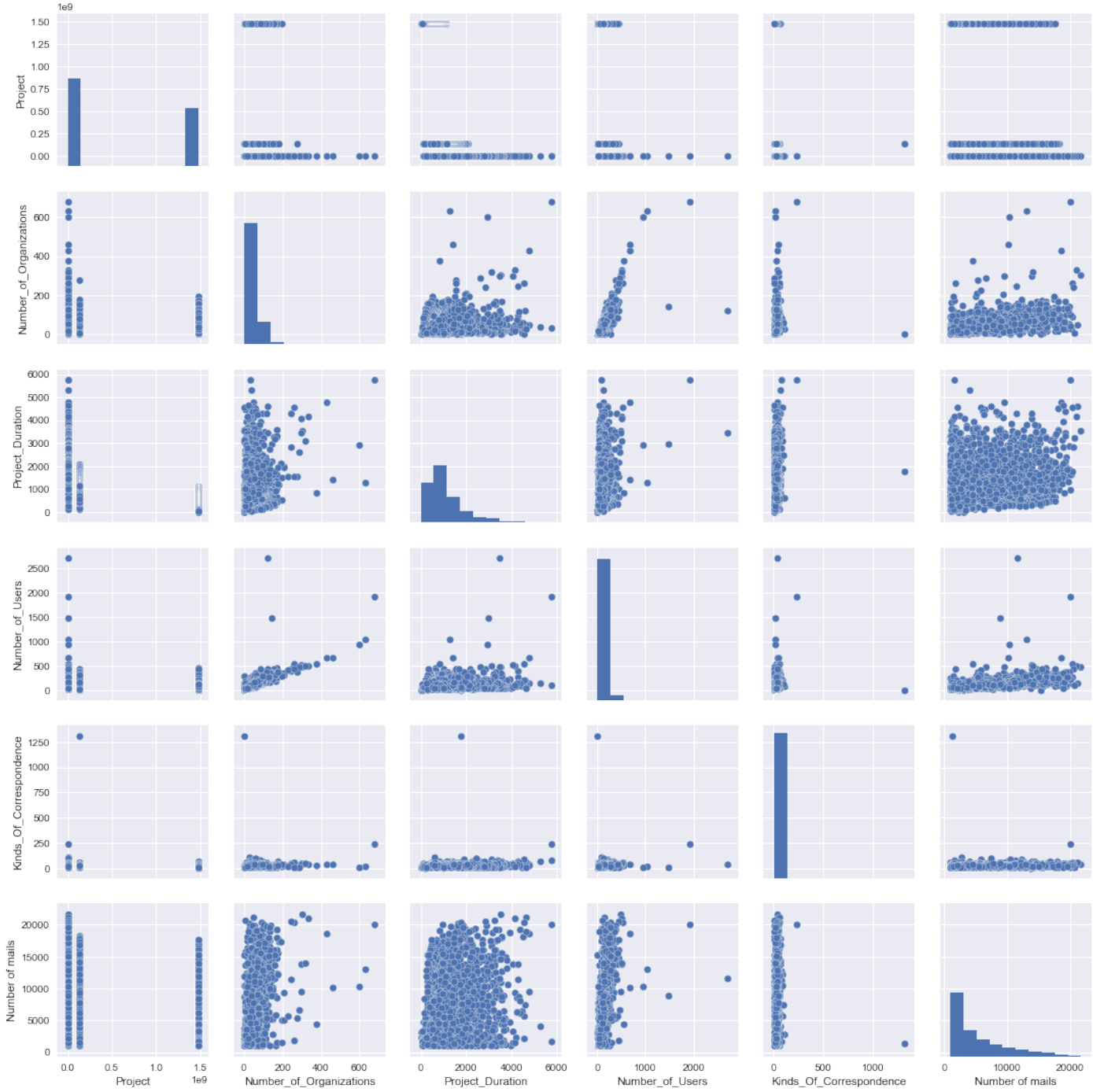
Code:

```
def create_Summary(project,row):

    project['sentDate'] = pd.to_datetime(project['sentDate'],format='%Y-%m-%d %H:%M:%S.%f')

    project = project[project['sentDate'].notnull()]
    project = project.sort_values('sentDate')

    df = pd.DataFrame()
    df.loc[row,'Project'] = project['projectId'][project.index[0]]

    project.groupby('fromOrganizationId')
    df.loc[row,'Number_of_Organizations'] = len(project.groupby('fromOrganizationId').count())

    df.loc[row,'Project_Duration'] = int((project['sentDate'][project.index[-1]] - project['sentDate'][project.index[0]

    project.groupby('fromUserId')
    df.loc[row,'Number_of_Users'] = len(project.groupby('fromUserId').count())

    project.groupby('correspondenceTypeId')
    df.loc[row,'Kinds_Of_Correspondence'] = len(project.groupby('correspondenceTypeId').count())

    df.loc[row,'Number of mails'] = len(project)

    return df
```

Let us examine the correlation and heatmap for the dataset:

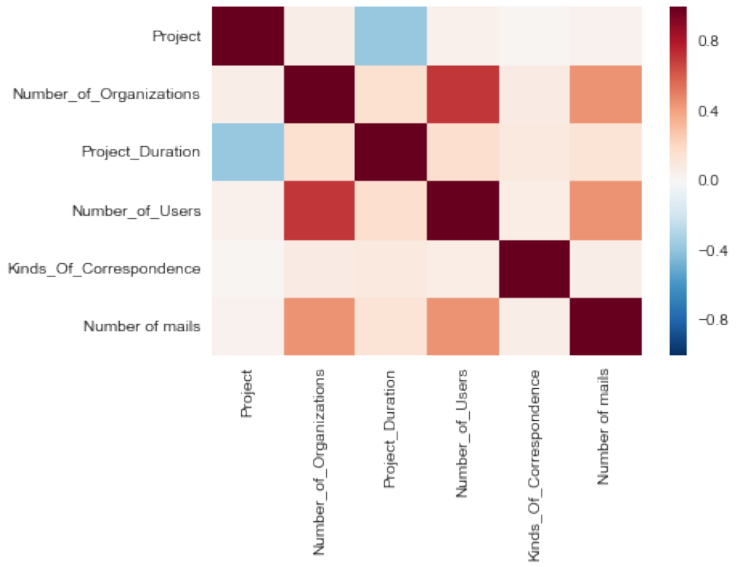
Clearly, the number of participating organizations and number of users are strongly correlated, while a number of weaker correlations are observed amongst other features.



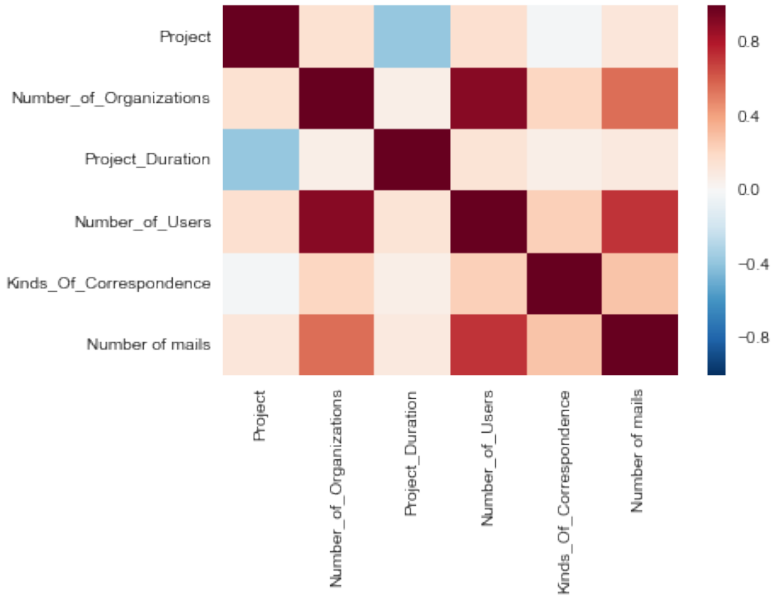


Let’s examine the trends in different segment of the project durations.

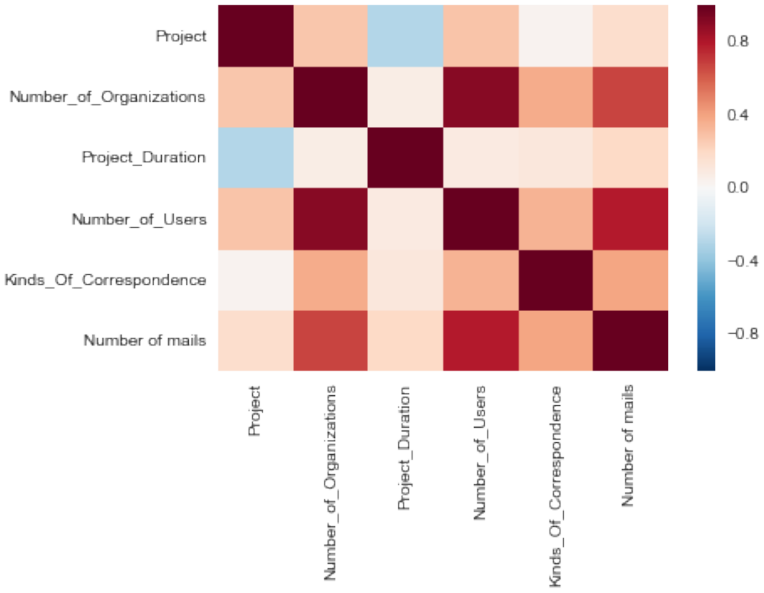
- Top 33% in project duration:



- Middle 33% in project duration:



- Bottom 33% in project duration:



Though the linear relationship is stronger amongst features for projects which were finished relatively quickly than those which went for longer time.
In summary, we are seeing non-linear relationships amongst the entire feature space.

Let us model, the duration of duration of project based on feature vectors.

We will split the dataset into training and testing portions. Training will be cone on 70% data while testing on remaining 30%.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(frame_model_X, frame_model_y, test_size = 0.3, random_state = 100)
```

Support vector Regressor :

```
from sklearn.svm import SVR
from sklearn import metrics

regressor = SVR(kernel = 'rbf')
regressor.fit(X_train,y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
# Training Error
predictions = regressor.predict(X_train)
print('MAE:', metrics.mean_absolute_error(y_train, predictions))
print('MSE:', metrics.mean_squared_error(y_train, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, predictions)))
```

```
MAE: 549.982769549
MSE: 668721.417857
RMSE: 817.753885871
```

```
# Prediction Error
predictions = regressor.predict(X_test)
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 547.013159304
MSE: 636368.040543
RMSE: 797.726795678
```

Decision Tree Regressor :

```
from sklearn.tree import DecisionTreeRegressor

dtree = DecisionTreeRegressor()
dtree.fit(X_train,y_train)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_split=1e-07,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, presort=False, random_state=None,
           splitter='best')
```

```
# Training Error
predictions = dtree.predict(X_train)
print('MAE:', metrics.mean_absolute_error(y_train, predictions))
print('MSE:', metrics.mean_squared_error(y_train, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, predictions)))
```

```
MAE: 0.0
MSE: 0.0
RMSE: 0.0
```

```
# Prediction Error
predictions = dtree.predict(X_test)
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 693.298119964
MSE: 955980.77171
RMSE: 977.742691975
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(n_estimators=1000)
forest.fit(X_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=1000, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

```
# Training Error
predictions = forest.predict(X_train)
print('MAE:', metrics.mean_absolute_error(y_train, predictions))
print('MSE:', metrics.mean_squared_error(y_train, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, predictions)))
```

```
MAE: 193.715068688
MSE: 70781.4534326
RMSE: 266.047840496
```

```
# Prediction Error
predictions = forest.predict(X_test)
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 514.289182632
MSE: 487002.194184
RMSE: 697.855424987
```

Predictive Model Assessment:

As the dataset comes with inherent non-linearity, the support vector regressor isn't doing well in training and has high training error. And the same is being reflected in prediction stage as well

The decision tree is getting over-trained over the data. Here we expect more variability in prediction based on moderate changes in the training data. To cater to this problem, we should opt for the ensemble as an appropriate model.

The Random forest ensemble model does best on the training phase and also in the prediction phase as the corresponding errors are smaller.

The ensemble model should be chosen to predict the time taken on a project.

In general, we are seeing higher errors in model predictions. These are because of the presence of significant outliers in feature sets. Also, the prediction models can be improved by tuning the hyper-parameters.