Loading the dataset

In [473]:

```
import pandas as pd
import os
import boto3
```

In [474]:

```
s3 = boto3.resource('s3')
s3.Bucket('aws-logs-862438390833-us-east-1').download_file('elasticmapreduce/tp2
-dataset.zip','tp2-dataset.zip')
```

In [475]:

```
import zipfile
with zipfile.ZipFile("tp2-dataset.zip","r") as zip_ref:
    zip_ref.extractall('tp2-dataset')
```

In [476]:

```
files = os.listdir("tp2-dataset")
```

In [477]:

```
data = []
for file in files:
    data.append(pd.read_csv("tp2-dataset/" + file))
```

In [478]:

```
data[5].head()
```

Out[478]:

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	O3	TEMP	PRES	DEWP
0	1	2013	3	1	0	5.0	14.0	4.0	12.0	200.0	85.0	-0.5	1024.5	-21.4
1	2	2013	3	1	1	8.0	12.0	6.0	14.0	200.0	84.0	-0.7	1025.1	-22.1
2	3	2013	3	1	2	3.0	6.0	5.0	14.0	200.0	83.0	-1.2	1025.3	-24.6
3	4	2013	3	1	3	5.0	5.0	5.0	14.0	200.0	84.0	-1.4	1026.2	-25.5
4	5	2013	3	1	4	5.0	5.0	6.0	21.0	200.0	77.0	-1.9	1027.1	-24.5

In [479]:

```
data[4].shape
```

Out[479]:

```
(35064, 18)
```

In [480]:

```
df = pd.concat(data)
```

In [481]:

```
df.shape
```

Out[481]:

```
(385704, 18)
```

In [482]:

```
df_pipeline = df
```

In [483]:

```
df_test = df_pipeline
```

In [484]:

```
df_test.head()
```

Out[484]:

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	O3	TEMP	PRES	DEWP
0	1	2013	3	1	0	6.0	18.0	5.0	NaN	800.0	88.0	0.1	1021.1	-18.6
1	2	2013	3	1	1	6.0	15.0	5.0	NaN	800.0	88.0	-0.3	1021.5	-19.0
2	3	2013	3	1	2	5.0	18.0	NaN	NaN	700.0	52.0	-0.7	1021.5	-19.8
3	4	2013	3	1	3	6.0	20.0	6.0	NaN	NaN	NaN	-1.0	1022.7	-21.2
4	5	2013	3	1	4	5.0	17.0	5.0	NaN	600.0	73.0	-1.3	1023.0	-21.4

Data Preprocessing

Dropping the rows where the temperature (label) is null

In [485]:

```
df = df[df['TEMP'].notna()]
df_pipeline = df_pipeline[df_pipeline['TEMP'].notna()]
```

In [486]:

```
df.shape
```

Out[486]:

```
(385325, 18)
```

In [487]:

```
df.describe()
```

Out[487]:

	No	year	month	day	hour	PI
count	385325.000000	385325.000000	385325.000000	385325.000000	385325.000000	377282.00
mean	17524.858732	2014.661431	6.526161	15.726086	11.499804	79.31
std	10120.701659	1.176838	3.447381	8.800716	6.922627	80.33
min	1.000000	2013.000000	1.000000	1.000000	0.000000	2.00
25%	8759.000000	2014.000000	4.000000	8.000000	5.000000	20.00
50%	17531.000000	2015.000000	7.000000	16.000000	11.000000	55.00
75%	26289.000000	2016.000000	10.000000	23.000000	18.000000	110.00
max	35064.000000	2017.000000	12.000000	31.000000	23.000000	957.00

Counting the null values for each explanatory variable

In [488]:

```
print(df['PM2.5'].isna().sum())
```

```
8043
```

In [489]:

```
print(df['PM10'].isna().sum())
```

```
5963
```

In [490]:

```
df['SO2'].isna().sum()
```

Out[490]:

```
8352
```

In [491]:

```
df['NO2'].isna().sum()
```

Out[491]:

```
11361
```

In [492]:

```
df['CO'].isna().sum()
```

Out[492]:

```
19403
```

In [493]:

```
df['O3'].isna().sum()
```

Out[493]:

```
12192
```

In [494]:

```
df['PRES'].isna().sum()
```

Out[494]:

```
2
```

In [495]:

```
df['DEWP'].isna().sum()
```

Out[495]:

```
5
```

In [496]:

```
df['RAIN'].isna().sum()
```

Out[496]:

```
6
```

In [497]:

```
df['WSPM'].isna().sum()
```

Out[497]:

```
4
```

In [498]:

```
df['wd'].isna().sum()
```

Out[498]:

```
1439
```

In [499]:

```
df.iloc[0].values
```

Out[499]:

```
array([1, 2013, 3, 1, 0, 6.0, 18.0, 5.0, nan, 800.0, 88.0, 0.1, 1021
.1,
       -18.6, 0.0, 'NW', 4.4, 'Gucheng'], dtype=object)
```

In [500]:

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer
import numpy as np

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
df[['PM2.5','PM10', 'SO2', 'NO2', 'CO', 'O3', 'PRES', 'DEWP', 'RAIN', 'WSPM']] =
imputer.fit_transform(
    df[['PM2.5','PM10', 'SO2', 'NO2', 'CO', 'O3', 'PRES', 'DEWP', 'RAIN', 'WSPM'
]].values)


############# Pipeline #################
numbers_features = ['PRES', 'DEWP', 'RAIN', 'WSPM']
transfo_numbers = Pipeline(steps=[
    ('number', SimpleImputer(missing_values=np.nan, strategy='mean'))])

string_features = ['wd']
transfo_string = Pipeline(steps=[
    ('string', SimpleImputer(strategy="most_frequent"))])

imputer_tranformation = ColumnTransformer(
    transformers=[
        ('numbers', transfo_numbers, numbers_features),
    ('strings', transfo_string, string_features)])
```

```
/Users/charles-olivierfavreau/anaconda3/lib/python3.6/site-packages/
ipykernel_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
  if __name__ == '__main__':
/Users/charles-olivierfavreau/anaconda3/lib/python3.6/site-packages/
pandas/core/indexing.py:543: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
```

In [501]:

```
imputerString = SimpleImputer(strategy="most_frequent")
df['wd'] = imputerString.fit_transform(df['wd'].values.reshape(-1, 1))
```

```
/Users/charles-olivierfavreau/anaconda3/lib/python3.6/site-packages/
ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
```

Convert temperature into categories

In [502]:

```
def convertToCustomCategories(x):
    if x < 0:
        return 0
    if x < 10 and x >= 0:
        return 1
    if x < 20 and x >= 10:
        return 2
    if x < 30 and x >= 20:
        return 3
    if x >= 30:
        return 4
    else:
        print('Value not in range')
        print(x)
        return 2 #returning the halfway value (this gives less weight to an inco
rrect value)
```

In [503]:

```
df.TEMP = df.TEMP.apply(lambda x : convertToCustomCategories(x))
```

```
/Users/charles-olivierfavreau/anaconda3/lib/python3.6/site-packages/
pandas/core/generic.py:4401: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
  self[name] = value
```

In [504]:

```
df.head()
```

Out[504]:

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	O3
0	1	2013	3	1	0	6.0	18.0	5.000000	50.200872	800.000000	88.000000
1	2	2013	3	1	1	6.0	15.0	5.000000	50.200872	800.000000	88.000000
2	3	2013	3	1	2	5.0	18.0	15.711225	50.200872	700.000000	52.000000
3	4	2013	3	1	3	6.0	20.0	6.000000	50.200872	1217.891556	57.480654
4	5	2013	3	1	4	5.0	17.0	5.000000	50.200872	600.000000	73.000000

Convert Wind into One hot encoded variable

In [505]:

```
from sklearn.preprocessing import OneHotEncoder

wdOneHotEncoder = OneHotEncoder(handle_unknown='ignore')

df_wind = pd.get_dummies(df['wd'])
```

In [506]:

```
from sklearn.preprocessing import FunctionTransformer


transfo_WD = Pipeline(steps=[
    ('oneHotWD', OneHotEncoder(handle_unknown='ignore'))])

transfo_Station = Pipeline(steps=[
    ('oneHotStation', OneHotEncoder(handle_unknown='ignore'))])


oneHot_features = ['wd']
oneHot_station = ['station']

def removeWD(X):
    return X.drop(columns=['wd'])

def removeStation(X):
    return X.drop(columns=['station'])


oneHot_tranformation = ColumnTransformer(
    transformers=[
        ('numbers', transfo_numbers, numbers_features),
        ('strings', transfo_string, string_features),
        ('oneHot', transfo_WD, ['wd']),
        ('oneHotstation', transfo_Station, ['station'])
    ])
```

In [507]:

```
df_wind.head()
```

Out[507]:

	E	ENE	ESE	N	NE	NNE	NNW	NW	S	SE	SSE	SSW	SW	W	WNW	WSW
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

In [508]:

```
df = pd.concat([df,df_wind], axis=1)
```

In [509]:

```python
df.head()
```

Out[509]:

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	...	NNW
0	1	2013	3	1	0	6.0	18.0	5.000000	50.200872	800.000000	...	0
1	2	2013	3	1	1	6.0	15.0	5.000000	50.200872	800.000000	...	0
2	3	2013	3	1	2	5.0	18.0	15.711225	50.200872	700.000000	...	0
3	4	2013	3	1	3	6.0	20.0	6.000000	50.200872	1217.891556	...	0
4	5	2013	3	1	4	5.0	17.0	5.000000	50.200872	600.000000	...	0

5 rows × 34 columns

In [510]:

```python
df.shape
```

Out[510]:

```
(385325, 34)
```

Convert station into One hot encoded variable

In [511]:

```python
df = pd.concat([df,pd.get_dummies(df['station'])], axis=1)
```

In [512]:

```python
df.shape
```

Out[512]:

```
(385325, 45)
```

In [513]:

```
df.head()
```

Out[513]:

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	...	Changp
0	1	2013	3	1	0	6.0	18.0	5.000000	50.200872	800.000000	...	
1	2	2013	3	1	1	6.0	15.0	5.000000	50.200872	800.000000	...	
2	3	2013	3	1	2	5.0	18.0	15.711225	50.200872	700.000000	...	
3	4	2013	3	1	3	6.0	20.0	6.000000	50.200872	1217.891556	...	
4	5	2013	3	1	4	5.0	17.0	5.000000	50.200872	600.000000	...	

5 rows × 45 columns

Separating dataset into train and test

In [514]:

```
print(df.columns.values)
```

```
['No' 'year' 'month' 'day' 'hour' 'PM2.5' 'PM10' 'SO2' 'NO2' 'CO' 'O
3'
 'TEMP' 'PRES' 'DEWP' 'RAIN' 'wd' 'WSPM' 'station' 'E' 'ENE' 'ESE' '
N'
 'NE' 'NNE' 'NNW' 'NW' 'S' 'SE' 'SSE' 'SSW' 'SW' 'W' 'WNW' 'WSW'
 'Aotizhongxin' 'Changping' 'Dingling' 'Dongsi' 'Guanyuan' 'Gucheng'
 'Huairou' 'Nongzhanguan' 'Shunyi' 'Tiantan' 'Wanliu']
```

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import FunctionTransformer

X = df[[ 'DEWP',  'No', 'PRES', 'RAIN','WSPM',
         'hour', 'month', 'year',
      'E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE', 'SSE', 'SSW',
'SW', 'W',
 'WNW', 'WSW', 'E', 'ENE', 'ESE', 'N', 'NE','NNE', 'NNW', 'NW', 'S', 'SE', 'SSE'
,
 'SSW', 'SW', 'W', 'WNW', 'WSW', 'Aotizhongxin', 'Changping', 'Dingling', 'Dongs
i',
 'Guanyuan', 'Gucheng', 'Huairou', 'Nongzhanguan', 'Shunyi', 'Tiantan','Wanliu']
].values
```

```
X_pipeline = X


normalizer = Normalizer()

X = normalizer.fit_transform(X)

PCA = PCA(n_components=10)

SVD = TruncatedSVD(n_components=30)

#X = PCA.fit_transform(X)

Y = df['TEMP'].values

def toDense(X):
    return X.todense()

transformer = FunctionTransformer(toDense)

pipeline = Pipeline(steps=[
                                ('imputer',SimpleImputer(strategy="most_frequent
")),
                                ('oneHotStation', OneHotEncoder(handle_unknown='
ignore')),
                                ('normalizer', normalizer),
    ('toDense', transformer),
                                ('dimensionReduction', PCA)

                               ])

X_pipeline = df_pipeline[[ 'DEWP', 'PRES', 'RAIN','WSPM',
         'hour', 'month', 'year', 'station', 'wd']]

X_preprocessed_pipeline = pipeline.fit_transform(X_pipeline.values)

X_train, X_test, Y_train, Y_test = train_test_split(X_preprocessed_pipeline, Y,
test_size=0.15)
```

In [ ]:

```
X_preprocessed_pipeline.shape
```

Training

In [ ]:

```
import pickle
from joblibspark import register_spark


from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.utils import parallel_backend
from sklearn.model_selection import cross_val_score
from spark_sklearn.util import createLocalSparkSession
from sklearn.model_selection import GridSearchCV


os.environ['PYSPARK_PYTHON'] = '/usr/bin/python3'

register_spark()



SVM_Model = LinearSVC()

Random_Forest_Model = RandomForestClassifier()

Gradient_Boosting_Model = GradientBoostingClassifier()

print("Starting parallel tasks :")


with parallel_backend('spark', n_jobs=5):
    SVM_Model.fit(X_train, Y_train)
    Random_Forest_Model.fit(X_train, Y_train)
    Gradient_Boosting_Model.fit(X_train, Y_train)

print("Models training done")

with parallel_backend('spark', n_jobs=5):
    scoresSVM = cross_val_score(SVM_Model, X_train, Y_train, cv=2)


print("SVM_Model cross-validation done")

with parallel_backend('spark', n_jobs=5):
    scoresRandomForest = cross_val_score(Random_Forest_Model, X_train, Y_train,
cv=2)


print("Random_Forest_Model cross-validation  done")

with parallel_backend('spark', n_jobs=5):
    scoresGradientBoosting = cross_val_score(Gradient_Boosting_Model, X_train, Y
_train, cv=2)

print("Gradient_Boosting_Model cross-validation done")

print("Random_Forest score : ")
print(scoresRandomForest)
print( "SVM score : ")
print(scoresSVM)
print("Gradient_Boosting score : ")

print(scoresGradientBoosting)

fileName = "Pipeline"
f = open(fileName + '.pckl', 'wb')
pickle.dump(pipeline , f)
f.close()

fileName = "SVM"
f = open(fileName + '.pckl', 'wb')
pickle.dump(SVM_Model , f)
f.close()

fileName = "Random_Forest"
f = open(fileName + '.pckl', 'wb')
pickle.dump(Random_Forest_Model , f)
f.close()

fileName = "Gradient_Boosting"
f = open(fileName + '.pckl', 'wb')
pickle.dump(Gradient_Boosting_Model , f)
f.close()
```

In [ ]:

```
from sklearn.metrics import classification_report

Y_test_Predicted = Random_Forest_Model.predict(X_test)

print(classification_report(Y_test_Predicted, Y_test))
```

In [ ]:

```
Y_test_Predicted
```

In [ ]: