Mercedez Benz Greener Manufacturing

The 49th Solution

Submission and Description	Private Score	Public Score	Use for Final Score
submission_xgb_rf_stack_ridge_ohe.csv 2 minutes ago by Shibam Banerjee Stack with xgb ohe corr	0.55176	0.55587	☐
submission_xgb_rf_stack_ridge_label.csv 15 minutes ago by Shibam Banerjee Stack with xgb label corr	0.55316	0.55803	☐

47	▲1685	jonnyx	0.55318	23	3y
48	▲111	Data000	0.55317	69	3y
49	▲515	Valhak	0.55315	138	3y
50	▲641	Andreisun	0.55313	158	3y

0.55316 lies between the 48th and 49th position :)

1.Business/Real-world Problem

1.1. About Mercedez

Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include, for example, the passenger safety cell with crumple zone, the airbag and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium car makers. Daimler's Mercedes-Benz cars are leaders in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

1.2. Problem Statement

In this competition, Daimler is challenging Kagglers to tackle the curse of dimensionality and reduce the time that cars spend on the test bench. Competitors will work with a dataset representing different permutations of Mercedes-Benz car features to predict the time it takes to pass testing. Winning algorithms will contribute to speedier testing, resulting in lower carbon dioxide emissions without reducing Daimler's standards.

1.3 Source

To ensure the safety and reliability of each and every unique car configuration before they hit the road, Daimler's engineers have developed a robust testing system. But, optimizing the speed of their testing system for so many possible feature combinations is complex and time-consuming without a powerful algorithmic approach. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Daimler's production lines.

https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/overview

1.4. Real-world/Business objectives and constraints.

1. Reduce time taken by a particular model on test bench
2. Should predict the test time in few seconds or minutes but, not hours

2. Machine Learning Problem

2.1. Data

1. The Data has been provided by Daimler(Mercedez)
2. There are two data files provided. One for Train and one for Test.
3. Each files contains 4209 Data Points and 377 features.
4. There are 8 categorical features and the rest are numerical features.

2.2. Type of Machine Learning Problem

It is a Regression Problem. We have to predict the time taken by a vehicle on the test bench, which can be any real value.

2.3. Performance Metric

The Performance metric to be used is R2

Sum Squared Regression Error

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}}$$

Sum Squared Total Error

3. Exploratory Data Analysis

In [6]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
```

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from scipy import stats
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import Normalizer
import string
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.cm as cm
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from prettytable import PrettyTable
import pickle
from sklearn.model_selection import RepeatedKFold,KFold
from sklearn.metrics import r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.feature_extraction import DictVectorizer
from xgboost import plot_importance
from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDRegressor
from scipy import stats
import random
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.svm import SVR
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.model_selection import cross_validate
```

In [2]:

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

print("Number of data points:",train_df.shape[0])
```

```
Number of data points: 4209
```

In [3]:

```
train_df.columns
```

Out[3]:

```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=378)
```

In [4]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

In [5]:

```
#Checking whether there are any rows with null values
nan_rows = train_df[train_df.isnull().any(1)]
print (nan_rows)
```

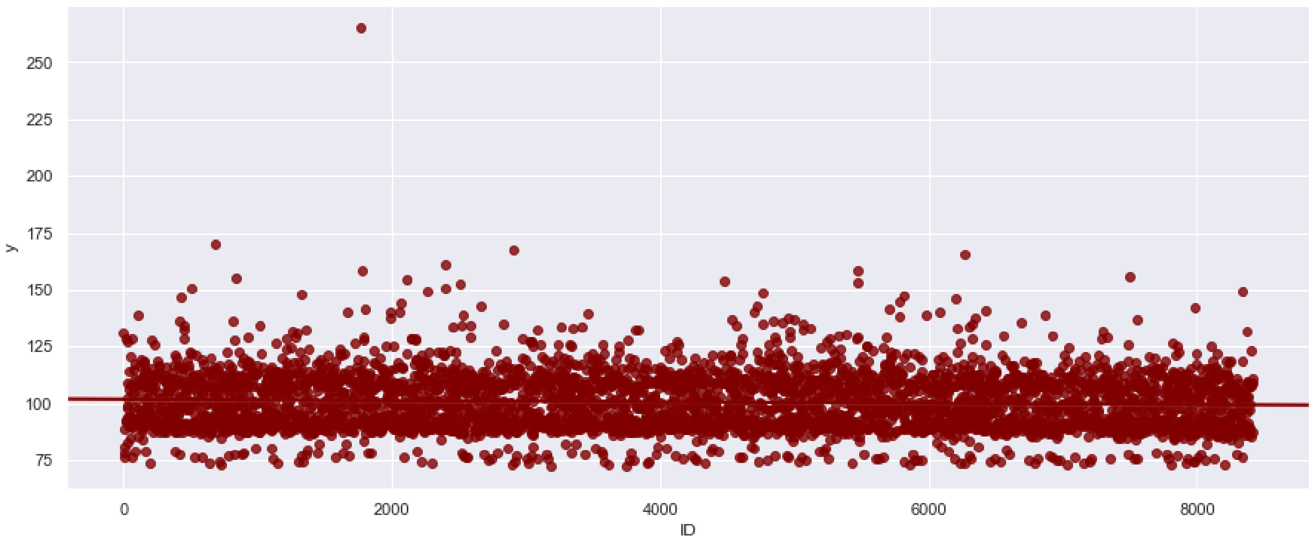
```
Empty DataFrame
Columns: [ID, y, X0, X1, X2, X3, X4, X5, X6, X8, X10, X11, X12, X13, X14, X15, X16, X17, X18, X19,
X20, X21, X22, X23, X24, X26, X27, X28, X29, X30, X31, X32, X33, X34, X35, X36, X37, X38, X39, X40
, X41, X42, X43, X44, X45, X46, X47, X48, X49, X50, X51, X52, X53, X54, X55, X56, X57, X58, X59, X
60, X61, X62, X63, X64, X65, X66, X67, X68, X69, X70, X71, X73, X74, X75, X76, X77, X78, X79, X80,
X81, X82, X83, X84, X85, X86, X87, X88, X89, X90, X91, X92, X93, X94, X95, X96, X97, X98, X99, X10
0, X101, ...]
Index: []

[0 rows x 378 columns]
```

In [13]:

```
sns.set(rc={'figure.figsize':(15,6)})
sns.regplot(x='ID', y='y', data=train_df,color='maroon')
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x245c3678708>
```



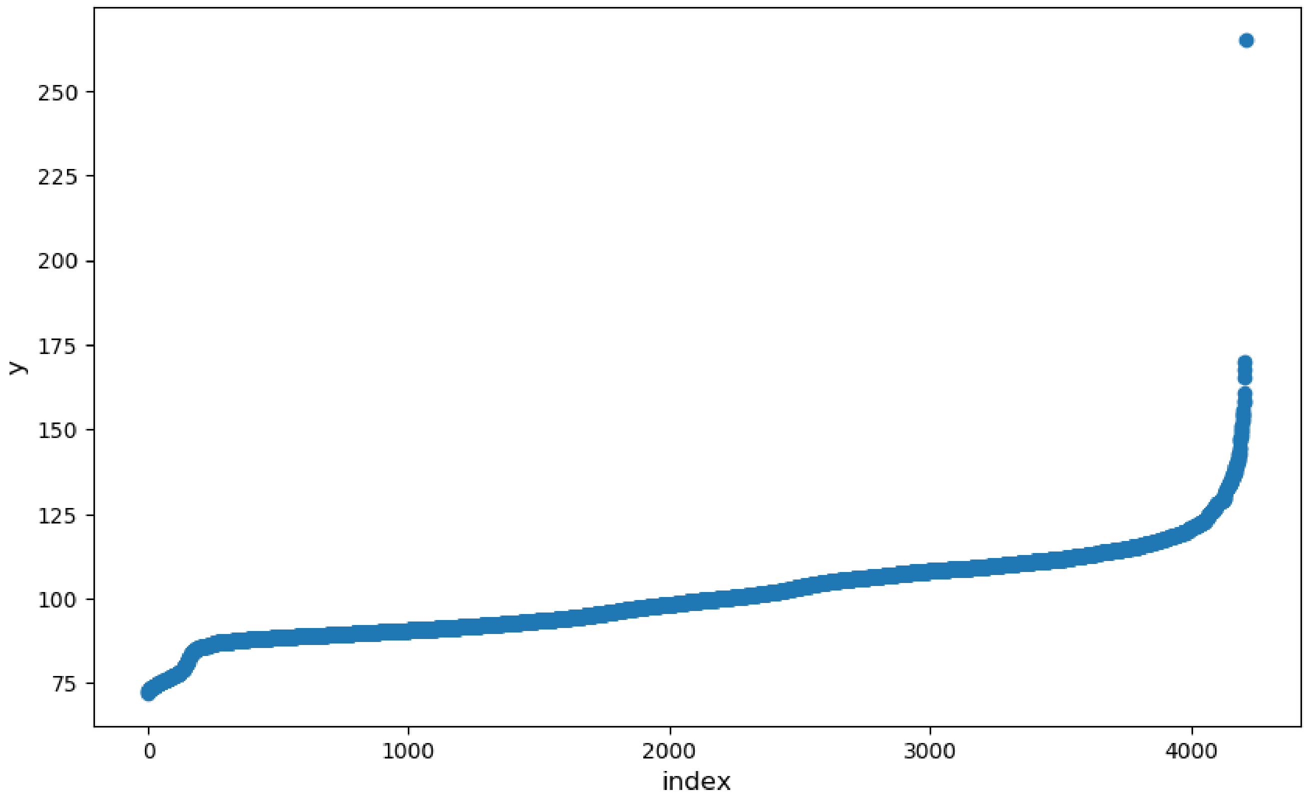
This shows a very slight decreasing trend of y with respect to the ID , maybe cars later in the series took less time in test bench. This gives ID an importance while estimating y hence will use it as a feature.

In [16]:

```
plt.style.use('default')
plt.figure(figsize=(10,6))
plt.scatter(range(train_df.shape[0]), np.sort(train_df.y.values))
plt.xlabel('index',fontsize=12)
plt.ylabel('y', fontsize=12)
```

Out[16]:

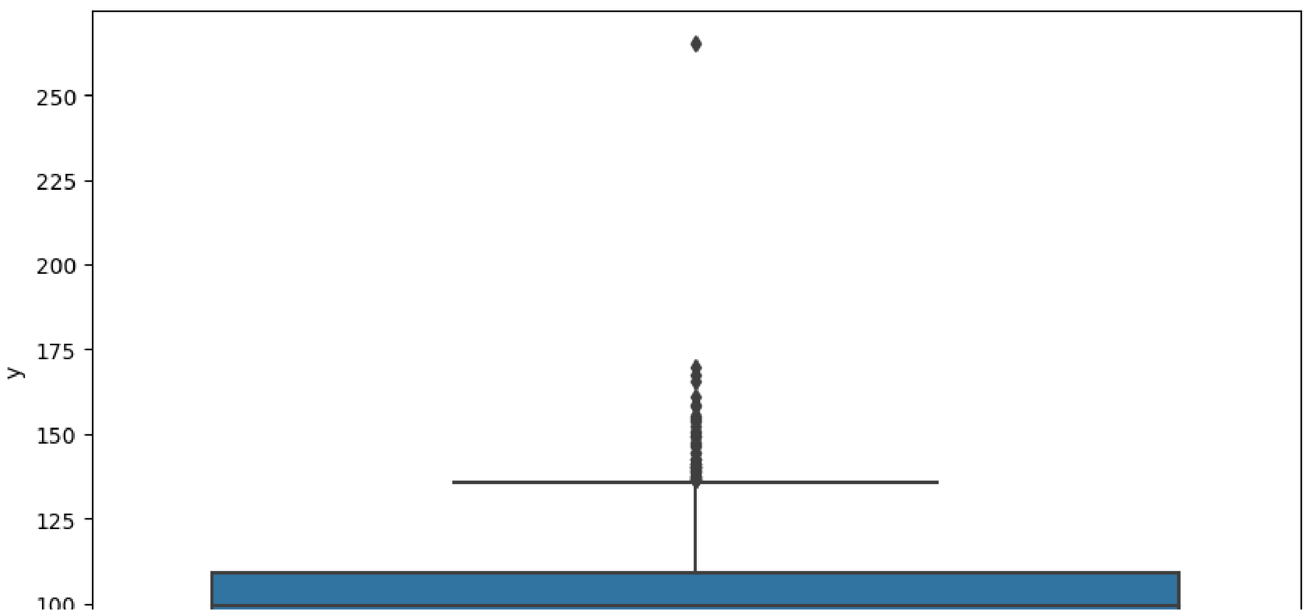
```
Text(0, 0.5, 'y')
```

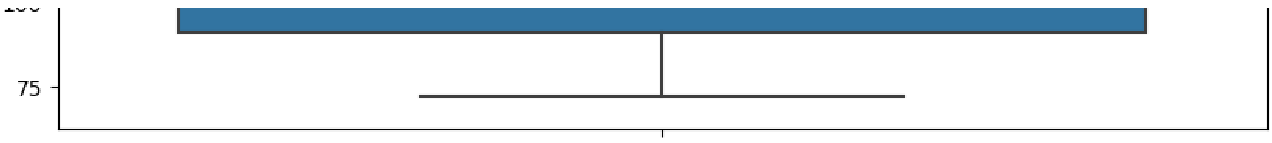


From the above plot it is clear that there are some outliers. Let's look into it further.

In [18]:

```
# the skewed box plot shows us the presence of outliers
plt.figure(figsize=(10,6))
sns.boxplot(y=train_df['y'], data =train_df)
plt.show()
```





From the above plot we can say most points above 140 might be outliers. So, we can set 150 as a threshold value. But, before doing that lets look furthur.

In [10]:

```
#calculating 0-100th percentile to find a the correct percentile value for removal of outliers
for i in range(0,100,10):
    var =train_df['y'].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
0 percentile value is 72.11
10 percentile value is 88.07
20 percentile value is 89.96
30 percentile value is 91.91
40 percentile value is 94.84
50 percentile value is 99.15
60 percentile value is 103.77
70 percentile value is 107.77
80 percentile value is 110.6
90 percentile value is 115.25
100 percentile value is  265.32
```

In [11]:

```
#looking further from the 90th percecntile
for i in range(90,100):
    var =train_df['y'].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
90 percentile value is 115.25
91 percentile value is 116.07
92 percentile value is 116.93
93 percentile value is 118.06
94 percentile value is 119.08
95 percentile value is 120.81
96 percentile value is 122.4
97 percentile value is 125.91
98 percentile value is 129.32
99 percentile value is 137.44
100 percentile value is  265.32
```

We can clearly see that the 99th percentile value is 137.44 and 100th percentile value is 265.32, which is an outlier point. Let's look furthur.

In [204]:

```
from franges import frange
#looking further from the 99th percecntile
for i in frange(99, 100, 0.1):
    var =train_df['y'].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 137.44
99.1 percentile value is 139.16
99.2 percentile value is 140.25
99.3 percentile value is 141.09
99.4 percentile value is 142.71
99.5 percentile value is 146.3
```

```
99.6 percentile value is 149.52
99.7 percentile value is 152.32
99.8 percentile value is 154.87
99.9 percentile value is 160.87
100 percentile value is  265.32
```

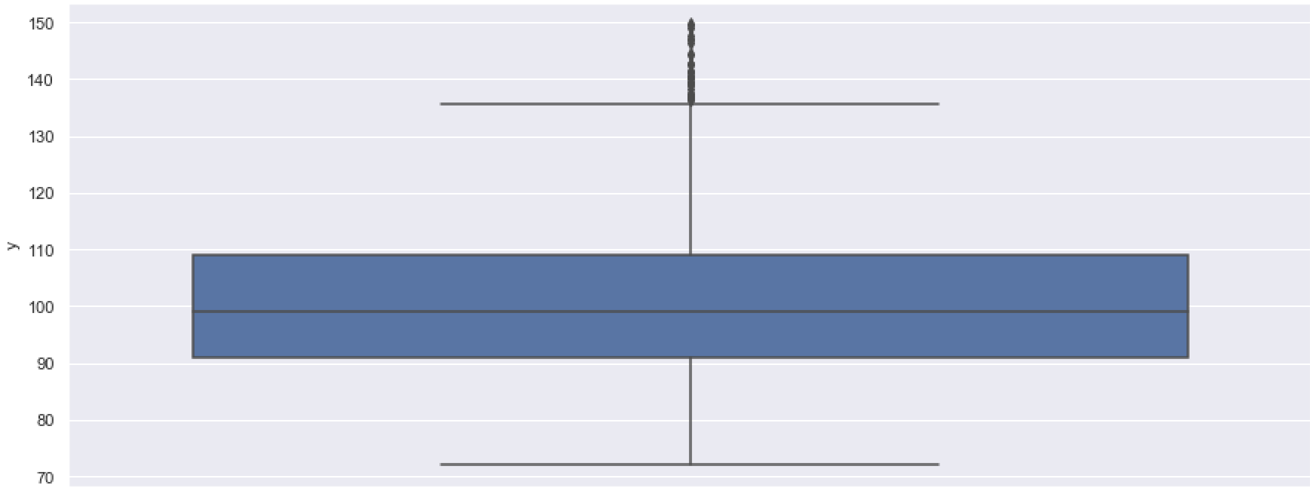
Taking 150 as threshold, will remove everything that is above 150 as outliers.

In [7]:

```
#removing data based on our analysis
train_df_modified=train_df[train_df.y<150]
```

In [207]:

```
#box-plot after removal of outliers
sns.boxplot(y=train_df_modified['y'], data =train_df_modified)
plt.show()
```



From the above plot we can see that some outliers have been removed.

Now lets check for duplicate features

In [8]:

```
rem_cols=[]
#removing duplicate columns and leaving the original behind.
dups=list(train_df_modified.T.index[train_df_modified.T.duplicated(keep= 'first')].values)
print(dups)
rem_cols.extend(dups)
```

```
['X35', 'X37', 'X39', 'X76', 'X84', 'X93', 'X94', 'X102', 'X107', 'X113', 'X119', 'X122', 'X134',
'X146', 'X147', 'X172', 'X199', 'X213', 'X214', 'X216', 'X222', 'X226', 'X227', 'X232', 'X233',
'X235', 'X239', 'X242', 'X243', 'X244', 'X245', 'X247', 'X248', 'X253', 'X254', 'X262', 'X266',
'X268', 'X279', 'X289', 'X290', 'X293', 'X296', 'X297', 'X299', 'X302', 'X320', 'X324', 'X326',
'X330', 'X339', 'X347', 'X360', 'X364', 'X365', 'X382', 'X385']
```

In [9]:

```
#X4 Found to have really low variance so will remove it
train_df_modified.X4.value_counts()
```

Out[9]:

```
d    4190
a       2
c       1
b       1
Name: X4, dtype: int64
```

In [10]:

```
df_num = train_df_modified.loc[:,train_df_modified.dtypes==np.int64]
```

In [11]:

```
#Removing features with 0 variance
temp = []
for i in df_num.columns:
    if train_df_modified[i].var()==0:
        temp.append(i)
print(len(temp))
print(temp)
```

```
13
['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297', 'X330', 'X339',
'X347']
```

In [12]:

```
rem_cols.extend(temp)
rem_cols= list(set(rem_cols))
rem_cols.append('X4')#Dropping X4 as it has very low variance
train_df_modified= train_df_modified.drop(rem_cols,axis=1)
train_df_modified.shape
```

Out[12]:

```
(4194, 319)
```

In [13]:

```
print("Number of data points:",train_df_modified.shape[0])
```

```
Number of data points: 4194
```

In [14]:

```
print("Number of data points removed:",train_df.shape[0]-train_df_modified.shape[0])
```

```
Number of data points removed: 15
```

In [15]:

```
print("Number of features removed:",train_df.shape[1]-train_df_modified.shape[1])
```

```
Number of features removed: 59
```

In [16]:

```
Y_train = train_df_modified['y']
```

In [17]:

```
train_df_modified.drop(['y'],axis=1,inplace=True)
X_train = train_df_modified
```

In [18]:

```
train_df_modified.shape
```

Out[18]:

```
(4194, 318)
```

In [19]:

```
X_train_cat = X_train.loc[:,X_train.dtypes==np.object]
```

In [20]:

```
X_train_cat.shape
```

Out[20]:

```
(4194, 7)
```

In [21]:

```
X_train_cat.head(2)
```

Out[21]:

	X0	X1	X2	X3	X5	X6	X8
0	k	v	at	a	u	j	o
1	k	t	av	e	y	l	o

In [22]:

```
X_train_num = X_train.loc[:,X_train.dtypes==np.int64]
```

In [23]:

```
X_train_num.shape
```

Out[23]:

```
(4194, 311)
```

In [24]:

```
X_train_num.head(2)
```

Out[24]:

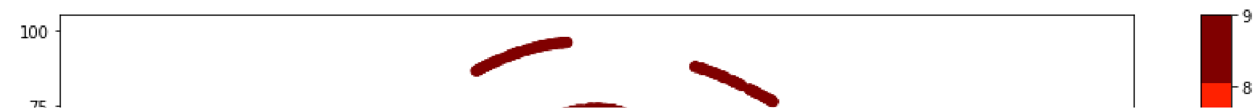
	ID	X10	X12	X13	X14	X15	X16	X17	X18	X19	...	X373	X374	X375	X376	X377	X378	X379	X380	X383	X384
0	0	0	0	1	0	0	0	0	1	0	...	0	0	0	0	1	0	0	0	0	0
1	6	0	0	0	0	0	0	0	1	0	...	0	0	1	0	0	0	0	0	0	0

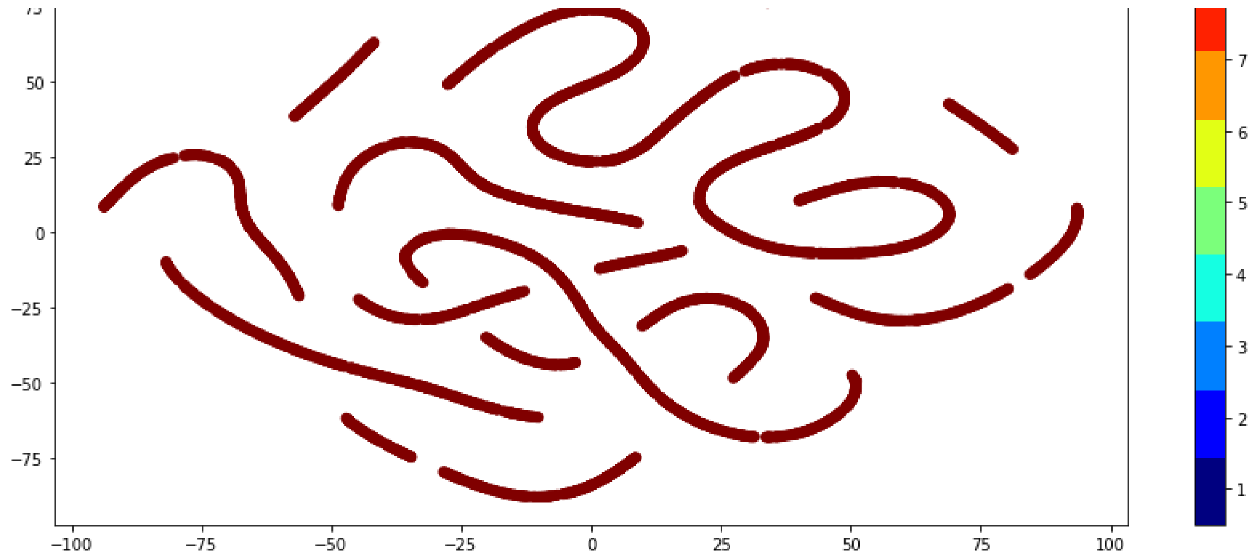
2 rows × 311 columns

Lets try plotting TSNE on numerical features:

In [36]:

```
xtsne=TSNE(perplexity=20)
results=xtsne.fit_transform(X_train_num)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=Y_train, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```





The above plot seems to tell that some points can be easily separated. Lets try PCA

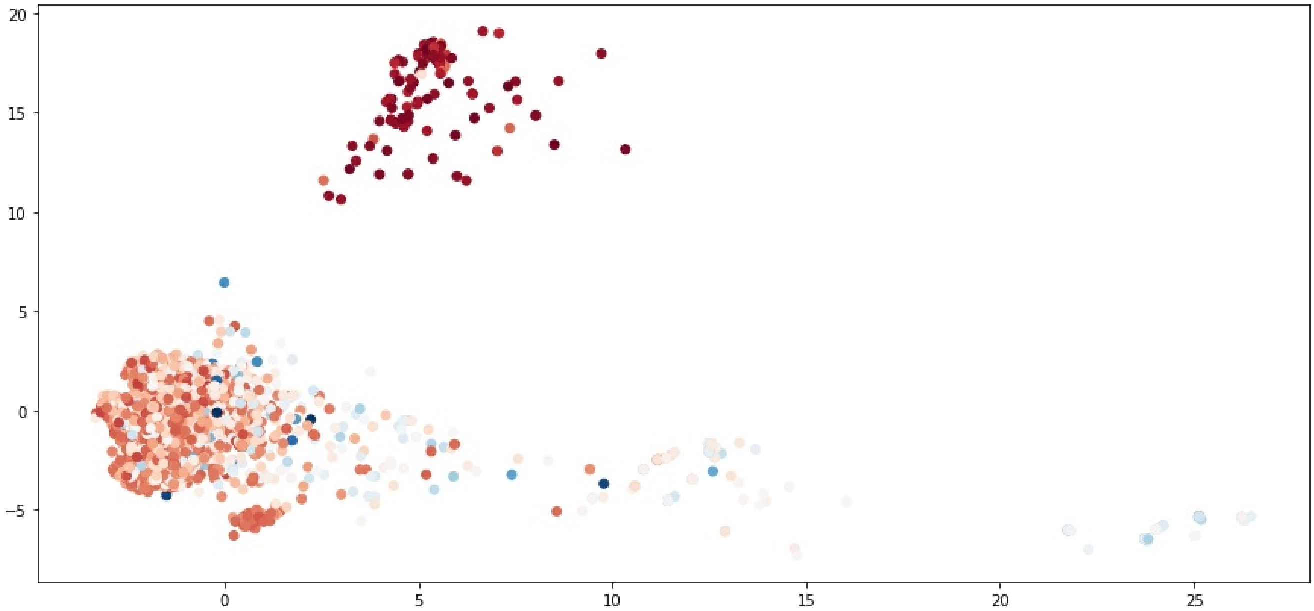
In [37]:

```
standardized_data = StandardScaler().fit_transform(X_train_num)
print(standardized_data.shape)
pca = PCA()
pca.n_components = 2
pca_data = pca.fit_transform(standardized_data)
print("shape of PCA reduced shape = ", pca_data.shape)
pca_data = np.vstack((pca_data.T, Y_train)).T
```

```
(4194, 311)
shape of PCA reduced shape =  (4194, 2)
```

In [38]:

```
pca_df = pd.DataFrame(data=pca_data, columns=("1st", "2nd", "label"))
plt.scatter(pca_df['1st'],pca_df['2nd'], c=pca_df['label'], cmap="RdBu")
plt.show()
```



From the above plot it is clear that reducing the dimention to 2, one cluster is separable from others. It might be a useful feature. Lets experiment with it.

In [25]:

```
#taking corr 0.25 as threshold on experimental grounds
dic={}
for i in X_train_num.columns:
    if i!='y':
        if train_df_modified[i].corr(Y_train)>0.25 or train_df_modified[i].corr(Y_train)<-0.25:
            dic[i]=train_df_modified[i].corr(Y_train)
print("Important Features with there respective correlations are ",'\n','--------------------------
-------------------------------',' \n',dic)
```

```
Important Features with there respective correlations are
 -----------------------------------------------------------
 {'X28': -0.2615483878531125, 'X29': -0.39798467184249353, 'X54': -0.39362263688451005, 'X80': -0.
2566304628986175, 'X118': 0.29113400781216325, 'X127': -0.5359508861669309, 'X136':
0.39362263688451005, 'X162': -0.380960152680421, 'X166': -0.3469061103890673, 'X178': -
0.31054903426087876, 'X185': -0.25654857309239787, 'X234': -0.2753088641090846, 'X250': -
0.3231881489692971, 'X261': 0.6184684577479753, 'X263': 0.39798467184249364, 'X272': -
0.3677994456153429, 'X275': 0.2929709300575139, 'X276': -0.37663134331800774, 'X313': -
0.34537856983725806, 'X314': 0.6371978536813555, 'X316': -0.2747484119054768, 'X328': -
0.3839243197734772, 'X348': -0.2575483559803369, 'X378': -0.27115936517391365}
```

There are some positive correlations, which seems interesting. Combining such features might give better results.

Now, preparing the Test Data:

In [26]:

```
test_df = pd.read_csv('test.csv')


print("Number of data points:",test_df.shape[0])
```

```
Number of data points: 4209
```

In [27]:

```
test_df.head(2)
```

Out[27]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0	0	0	0

2 rows × 377 columns

In [28]:

```
ID = test_df['ID']
```

In [29]:

```
#test_df.drop(['ID'],axis=1,inplace=True)
X_test = test_df
```

In [30]:

```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 377 entries, ID to X385
dtypes: int64(369), object(8)
memory usage: 12.1+ MB
```

In [31]:

```
test_df_modified= test_df.drop(rem_cols,axis=1)
test_df_modified.shape
```

Out[31]:

```
(4209, 318)
```

In [32]:

```
X_test_cat = test_df_modified.loc[:,test_df.dtypes==np.object]
```

In [33]:

```
X_test_cat.shape
```

Out[33]:

```
(4209, 7)
```

In [34]:

```
X_test_num = test_df_modified.loc[:,test_df.dtypes==np.int64]
```

In [35]:

```
X_test_num.shape
```

Out[35]:

```
(4209, 311)
```

4. Pre-processing of features

One Hot Encoding:

In [36]:

```
vectorizer=LabelBinarizer(sparse_output=True)
#Using One Hot Encoding
X_train_x0_ohe=vectorizer.fit_transform(X_train['X0'].values)
X_test_x0_ohe=vectorizer.transform(X_test['X0'].values)

print("After vectorizations : X0")
print(X_train_x0_ohe.shape)
print(X_test_x0_ohe.shape)
```

```
After vectorizations : X0
(4194, 47)
(4209, 47)
```

In [37]:

```
vectorizer=LabelBinarizer(sparse_output=True)
#Using One Hot Encoding
X_train_x1_ohe=vectorizer.fit_transform(X_train['X1'].values)
X_test_x1_ohe=vectorizer.transform(X_test['X1'].values)

print("After vectorizations : X1")
print(X_train_x1_ohe.shape)
print(X_test_x1_ohe.shape)
```

```
After vectorizations : X1
(4194, 27)
(4209, 27)
```

In [38]:

```
vectorizer=LabelBinarizer(sparse_output=True)
#Using One Hot Encoding
X_train_x2_ohe=vectorizer.fit_transform(X_train['X2'].values)
X_test_x2_ohe=vectorizer.transform(X_test['X2'].values)

print("After vectorizations : X2")
print(X_train_x2_ohe.shape)
print(X_test_x2_ohe.shape)
```

```
After vectorizations : X2
(4194, 44)
(4209, 44)
```

In [39]:

```
vectorizer=LabelBinarizer(sparse_output=True)
#Using One Hot Encoding
X_train_x3_ohe=vectorizer.fit_transform(X_train['X3'].values)
X_test_x3_ohe=vectorizer.transform(X_test['X3'].values)

print("After vectorizations : X3")
print(X_train_x3_ohe.shape)
print(X_test_x3_ohe.shape)
```

```
After vectorizations : X3
(4194, 7)
(4209, 7)
```

In [40]:

```
vectorizer=LabelBinarizer(sparse_output=True)
#Using One Hot Encoding
X_train_x5_ohe=vectorizer.fit_transform(X_train['X5'].values)
X_test_x5_ohe=vectorizer.transform(X_test['X5'].values)

print("After vectorizations : X5")
print(X_train_x5_ohe.shape)
print(X_test_x5_ohe.shape)
```

```
After vectorizations : X5
(4194, 29)
(4209, 29)
```

In [41]:

```
vectorizer=LabelBinarizer(sparse_output=True)
#Using One Hot Encoding
X_train_x6_ohe=vectorizer.fit_transform(X_train['X6'].values)
X_test_x6_ohe=vectorizer.transform(X_test['X6'].values)

print("After vectorizations : X6")
print(X_train_x6_ohe.shape)
print(X_test_x6_ohe.shape)
```

```
After vectorizations : X6
(4194, 12)
(4209, 12)
```

In [42]:

```
vectorizer=LabelBinarizer(sparse_output=True)
#Using One Hot Encoding
X_train_x8_ohe=vectorizer.fit_transform(X_train['X8'].values)
X_test_x8_ohe=vectorizer.transform(X_test['X8'].values)

print("After vectorizations : X8")
print(X_train_x8_ohe.shape)
```

```
print(X_test_x8_ohe.shape)
```

```
After vectorizations : X8
(4194, 25)
(4209, 25)
```

Final Data:

In [43]:

```
from scipy.sparse import hstack
print('Final Matrix:')
X_train_ohe = hstack((X_train_x0_ohe,X_train_x1_ohe,X_train_x2_ohe,X_train_x3_ohe,X_train_x5_ohe,X_
train_x6_ohe,X_train_x8_ohe,X_train_num)).tocsr()
print(X_train_ohe.shape)
X_test_ohe =
hstack((X_test_x0_ohe,X_test_x1_ohe,X_test_x2_ohe,X_test_x3_ohe,X_test_x5_ohe,X_test_x6_ohe,X_test_
x8_ohe,X_test_num)).tocsr()
print(X_test_ohe.shape)
```

```
Final Matrix:
(4194, 502)
(4209, 502)
```

In [49]:

```
X_train_ohe_full = X_train_ohe
X_test_ohe_full = X_test_ohe
```

Label Encoding:

In [44]:

```
vocab = []
vocab.extend(X_train_cat.values.ravel())
vocab.extend(X_test_cat.values.ravel())
```

In [45]:

```
vectorizer=LabelEncoder()
vectorizer.fit(vocab)
#Using Label Encoding
X_train_x0_le=vectorizer.transform(X_train['X0'].values).reshape(len(X_train),1)
X_test_x0_le=vectorizer.transform(X_test['X0'].values).reshape(len(X_test),1)

X_train_x1_le=vectorizer.transform(X_train['X1'].values).reshape(len(X_train),1)
X_test_x1_le=vectorizer.transform(X_test['X1'].values).reshape(len(X_test),1)

X_train_x2_le=vectorizer.transform(X_train['X2'].values).reshape(len(X_train),1)
X_test_x2_le=vectorizer.transform(X_test['X2'].values).reshape(len(X_test),1)

X_train_x3_le=vectorizer.transform(X_train['X3'].values).reshape(len(X_train),1)
X_test_x3_le=vectorizer.transform(X_test['X3'].values).reshape(len(X_test),1)

X_train_x5_le=vectorizer.transform(X_train['X5'].values).reshape(len(X_train),1)
X_test_x5_le=vectorizer.transform(X_test['X5'].values).reshape(len(X_test),1)

X_train_x6_le=vectorizer.transform(X_train['X6'].values).reshape(len(X_train),1)
X_test_x6_le=vectorizer.transform(X_test['X6'].values).reshape(len(X_test),1)

X_train_x8_le=vectorizer.transform(X_train['X8'].values).reshape(len(X_train),1)
X_test_x8_le=vectorizer.transform(X_test['X8'].values).reshape(len(X_test),1)
```

In [46]:

```
from scipy.sparse import hstack
print('Categorical Label Encoded Matrix:')
X_train_cat_le =
np.concatenate((X_train_x0_le,X_train_x1_le,X_train_x2_le,X_train_x3_le,X_train_x5_le,X_train_x6_le
```

```
,X_train_x8_le),axis=1)
print(X_train_cat_le.shape)
X_test_cat_le = np.concatenate((X_test_x0_le,X_test_x1_le,X_test_x2_le,X_test_x3_le,X_test_x5_le,X_test_x6_le,X_test_x8_le),axis=1)
print(X_test_cat_le.shape)
```

```
Categorical Label Encoded Matrix:
(4194, 7)
(4209, 7)
```

In [47]:

```
normalizer=Normalizer()
#Initializing the normalizer to normalize the train and test data
X_train_cat_le=normalizer.fit_transform(X_train_cat_le)
X_test_cat_le=normalizer.transform(X_test_cat_le)
```

In [48]:

```
from scipy.sparse import hstack
print('Final Matrix:')
X_train_le = hstack((X_train_cat_le,X_train_num)).tocsr()
print(X_train_le.shape)
X_test_le = hstack((X_test_cat_le,X_test_num)).tocsr()
print(X_test_le.shape)
```

```
Final Matrix:
(4194, 318)
(4209, 318)
```

As we can see that the dimentions of the One Hot Encoded features are much more than that of thr Label Encoded. Lets see if reducing it and matching it to the Label Encoded makes any difference. The dataset tends to overfit and reducing the dimention will remove some unnecessary features and also might increase the performance.

Reducing the dimentions of One Hot Encoded Features to match the Label Encoded Features:

In [50]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(with_mean=False)
X_train_ohe = scaler.fit_transform(X_train_ohe)
X_test_ohe = scaler.transform(X_test_ohe)
```

In [51]:

```
from sklearn.feature_selection import SelectKBest
selector = SelectKBest(k=X_train_le.shape[1])
selector.fit(X_train_ohe, Y_train)
X_train_ohe = selector.transform(X_train_ohe)
X_test_ohe = selector.transform(X_test_ohe)
print(X_train_ohe.shape)
print(X_test_ohe.shape)
```

```
(4194, 318)
(4209, 318)
```

Adding PCA Transformed as Features

In [52]:

```
#https://blog.goodaudience.com/stacking-ml-algorithm-for-mercedes-benz-greener-manufacturing-competition-5600762186ae
standardized_data_tr = StandardScaler().fit_transform(X_train_num)
standardized_data_te = StandardScaler().fit_transform(X_test_num)
print(standardized_data_tr.shape)
print(standardized_data_te.shape)
pca = PCA()
```

```
pca.n_components = 6 #Chosen on experimental grounds
pca_data_tr = pca.fit_transform(standardized_data_tr)
pca_data_te = pca.transform(standardized_data_te)
print('After Transformation:')
print(pca_data_tr.shape)
print(pca_data_te.shape)
```

```
(4194, 311)
(4209, 311)
After Transformation:
(4194, 6)
(4209, 6)
```

In [53]:

```
train_df_modified_pca = train_df_modified.copy()
train_df_modified_pca["PCA_1"] = pca_data_tr[:,0]
train_df_modified_pca["PCA_2"] = pca_data_tr[:,1]
train_df_modified_pca["PCA_3"] = pca_data_tr[:,2]
train_df_modified_pca["PCA_4"] = pca_data_tr[:,3]
train_df_modified_pca["PCA_5"] = pca_data_tr[:,4]
train_df_modified_pca["PCA_6"] = pca_data_tr[:,5]
test_df_modified_pca = test_df_modified.copy()
test_df_modified_pca["PCA_1"] = pca_data_te[:,0]
test_df_modified_pca["PCA_2"] = pca_data_te[:,1]
test_df_modified_pca["PCA_3"] = pca_data_te[:,2]
test_df_modified_pca["PCA_4"] = pca_data_te[:,3]
test_df_modified_pca["PCA_5"] = pca_data_te[:,4]
test_df_modified_pca["PCA_6"] = pca_data_te[:,5]
```

One Hot Encoding(K Best):

In [54]:

```
from scipy.sparse import hstack
print('Final PCA Matrix:')
X_train_ohe_PCA = hstack((X_train_ohe,pca_data_tr)).tocsr()
print(X_train_ohe_PCA.shape)
X_test_ohe_PCA = hstack((X_test_ohe,pca_data_te)).tocsr()
print(X_test_ohe_PCA.shape)
```

```
Final PCA Matrix:
(4194, 324)
(4209, 324)
```

One Hot Encoding with all features:

In [56]:

```
from scipy.sparse import hstack
print('Final PCA Matrix:')
X_train_ohe_full_PCA = hstack((X_train_ohe_full,pca_data_tr)).tocsr()
print(X_train_ohe_PCA.shape)
X_test_ohe_full_PCA = hstack((X_test_ohe_full,pca_data_te)).tocsr()
print(X_test_ohe_PCA.shape)
```

```
Final PCA Matrix:
(4194, 324)
(4209, 324)
```

Label Encoding:

In [55]:

```
from scipy.sparse import hstack
print('Final PCA Matrix:')
X_train_le_PCA = hstack((X_train_le,pca_data_tr)).tocsr()
print(X_train_le_PCA.shape)
X_test_le_PCA = hstack((X_test_le,pca_data_te)).tocsr()
```

```
print(X_test_le_PCA.shape)
```

```
Final PCA Matrix:
(4194, 324)
(4209, 324)
```

5. Modelling

Linear Regression with One Hot Encoding (K Best): Baseline

In [57]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
```

In [134]:

```
#Using RandomizedSearchCV with 10_fold
neigh=LinearRegression(n_jobs=-1)
parameters = {'fit_intercept':[True,False]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe,Y_train)
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.0s
[Parallel(n_jobs=-1)]: Done  10 out of  20 | elapsed:    2.5s remaining:    2.5s
[Parallel(n_jobs=-1)]: Done  15 out of  20 | elapsed:    2.9s remaining:    0.9s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    3.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    3.0s finished
```

Out[134]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                              n_jobs=-1, normalize=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'fit_intercept': [True, False]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [151]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
print('Best train score {} and cv score {}'.format(train_r2[0], cv_r2[0]))
```

```
Best train score 0.660205936447419 and cv score 0.6056558008234205
```

In [135]:

```
clf.best_estimator_
```

Out[135]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
```

In [57]:

```
model_lr = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
model_lr.fit(X_train_ohe,Y_train)
```

Out[57]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
```

In [58]:

```
pred_test_lr=model_lr.predict(X_test_ohe)
```

In [60]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_lr]}
data = pd.DataFrame(data)
data.to_csv("submission_lr.csv", index=False)
```

Linear Regression with One Hot Encoding (All Features): Baseline

In [58]:

```
#Using RandomizedSearchCV with 10_fold
neigh=LinearRegression(n_jobs=-1)
parameters = {'fit_intercept':[True,False]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_full,Y_train)
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.3s
[Parallel(n_jobs=-1)]: Done  10 out of  20 | elapsed:    3.5s remaining:    3.5s
[Parallel(n_jobs=-1)]: Done  15 out of  20 | elapsed:    3.7s remaining:    1.2s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    4.1s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    4.1s finished
```

Out[58]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                              n_jobs=-1, normalize=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'fit_intercept': [True, False]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [59]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
print('Best train score {} and cv score {}'.format(train_r2[0], cv_r2[0]))
```

```
Best train score 0.6741094203792876 and cv score 0.538044995004267
```

In [60]:

```
clf.best_estimator_
```

Out[60]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
```

In [61]:

```
model_lr = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
model_lr.fit(X_train_ohe_full,Y_train)
```

Out[61]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
```

In [62]:

```
pred_test_lr=model_lr.predict(X_test_ohe_full)
```

In [63]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_lr]}
data = pd.DataFrame(data)
data.to_csv("submission_lr_full.csv", index=False)
```

From the above 2 models, it is clear that the One Hot Encoded Features is causing overfitting and the K-Best One Hot Encoded Features are performing much more better. Therefore, dropped all the One-Hot Encoded Features and used only the K-Best features.

Linear Regression with Label Encoding: Baseline

In [82]:

```
#Using RandomizedSearchCV with 10_fold
neigh=LinearRegression(n_jobs=-1)
parameters = {'fit_intercept':[True,False]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le,Y_train)
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.4s
[Parallel(n_jobs=-1)]: Done  10 out of  20 | elapsed:    3.5s remaining:    3.5s
[Parallel(n_jobs=-1)]: Done  15 out of  20 | elapsed:    3.8s remaining:    1.2s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    4.1s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    4.1s finished
```

Out[82]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                              n_jobs=-1, normalize=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'fit_intercept': [True, False]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [83]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
print('Best train score {} and cv score {}'.format(train_r2[0], cv_r2[0]))
```

```
Best train score 0.6472747876597993 and cv score 0.5869231412489889
```

In [184]:

```
model_lr_le = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
model_lr_le.fit(X_train_le,Y_train)
```

Out[184]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
```

In [185]:

```
pred_test_lr_label=model_lr_le.predict(X_test_le)
```

In [186]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_lr_label]}
data = pd.DataFrame(data)
data.to_csv("submission_lr_label.csv", index=False)
```

From the above tests we can conclude that One Hot Encoding with the reduced dimention performed better than the Label Encoded features in cross-validation.

0.60565 is taken as the Baseline Score

ElasticNet with One Hot Encoding (K-Best):

In [154]:

```
#Using RandomizedSearchCV with 5_fold
neigh=ElasticNet()
parameters = {'alpha':[0.001,0.01,0.1,1]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe,Y_train)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.8s
[Parallel(n_jobs=-1)]: Done  34 out of  40 | elapsed:    7.6s remaining:    1.3s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:    9.0s finished
```

Out[154]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=1000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'alpha': [0.001, 0.01, 0.1, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [155]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

In [156]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6570656548088905 CV:0.6137555782792445
Train:0.644281801689052 CV:0.6214897234479223
Train:0.6008887444847001 CV:0.5923676689101759
Train:0.38884414072726764 CV:0.3840739255485175
```

In [157]:

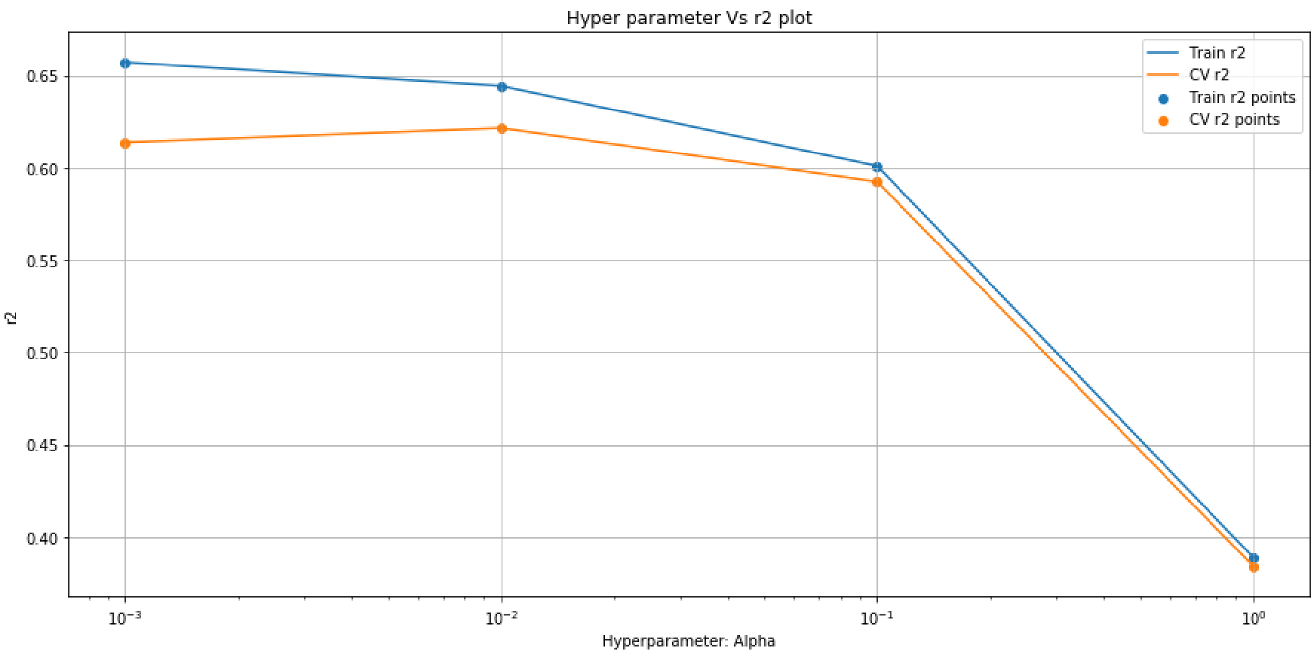
```
clf.best_estimator_
```

Out[157]:

```
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

In [158]:

```
Alpha = [0.001,0.01,0.1,1]
plt.figure(figsize=(15,7))
plt.plot(Alpha,train_r2,label='Train r2')
#https://stackoverflow.com/a/48803361/4084039
plt.plot(Alpha,cv_r2,label='CV r2')
#https://stackoverflow.com/a/48803361/4084039
plt.scatter(Alpha,train_r2,label='Train r2 points')
plt.scatter(Alpha,cv_r2,label='CV r2 points')
plt.legend()
plt.xscale('log')
plt.xlabel("Hyperparameter: Alpha")
plt.ylabel("r2")
plt.title("Hyper parameter Vs r2 plot")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6214897234479223
```

From the above plot we can see that as the alpha value decreses, the tendency to overfit increases. However with 0.01 Alpha value, there seem to be no overfitting and the CV score is maximum.

In [61]:

```
model_el = ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
model_el.fit(X_train_ohe,Y_train)
```

Out[61]:

```
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

In [62]:

```
pred_test_el=model_el.predict(X_test_ohe)
```

In [63]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_el]}
data = pd.DataFrame(data)
data.to_csv("submission_el.csv", index=False)
```

ElasticNet with One Hot Encoding(K-Best) + PCA components:

In [64]:

```
#Using RandomizedSearchCV with 5_fold
neigh=ElasticNet()
parameters = {'alpha':[0.001,0.01,0.1,1]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_PCA,Y_train)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    6.7s
[Parallel(n_jobs=-1)]: Done  34 out of  40 | elapsed:   14.6s remaining:    2.5s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   16.1s finished
```

Out[64]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=1000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'alpha': [0.001, 0.01, 0.1, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [61]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

In [62]:

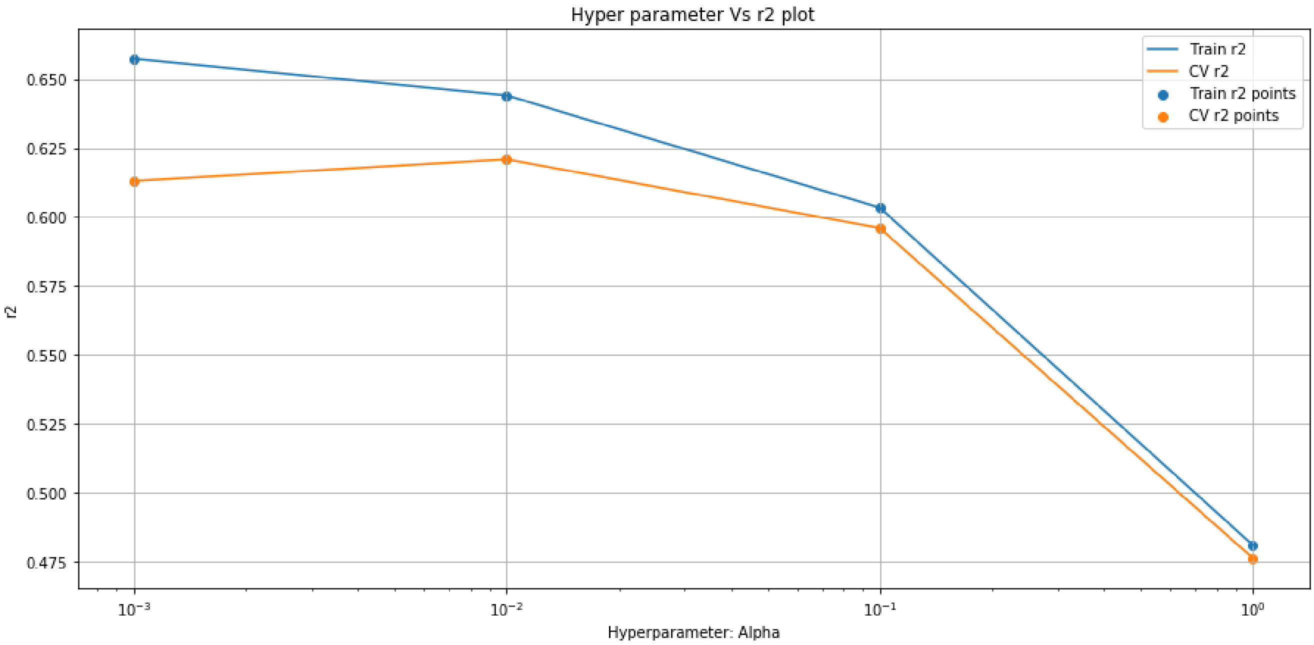
```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6574032393839928 CV:0.6130490497854113
Train:0.6441149023704883 CV:0.6209219850984878
Train:0.6033504200879262 CV:0.5959548434483816
Train:0.4808994719719625 CV:0.4762108227197895
```

In [63]:

```
Alpha = [0.001,0.01,0.1,1]
plt.figure(figsize=(15,7))
plt.plot(Alpha,train_r2,label='Train r2')
#https://stackoverflow.com/a/48803361/4084039
plt.plot(Alpha,cv_r2,label='CV r2')
#https://stackoverflow.com/a/48803361/4084039
plt.scatter(Alpha,train_r2,label='Train r2 points')
plt.scatter(Alpha,cv_r2,label='CV r2 points')
plt.legend()
```

```
plt.xscale('log')
plt.xlabel("Hyperparameter: Alpha")
plt.ylabel("r2")
plt.title("Hyper parameter Vs r2 plot")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6209219850984878
```

We can see that adding the PCA components in One Hot Encoded Features didnot improve the performance.

In [65]:

```
clf.best_estimator_
```

Out[65]:

```
ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

In [68]:

```
model_el_PCA = ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
model_el_PCA.fit(X_train_ohe_PCA,Y_train)
```

Out[68]:

```
ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

In [69]:

```
pred_test_el_PCA=model_el_PCA.predict(X_test_ohe_PCA)
```

In [71]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_el_PCA]}
data = pd.DataFrame(data)
data.to_csv("submission_el_pca.csv", index=False)
```

ElasticNet with Label Encoding:

In [98]:

```
#Using RandomizedSearchCV with 5_fold
neigh=ElasticNet()
parameters = {'alpha':[0.001,0.01,0.1,1]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le,Y_train)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    5.6s
[Parallel(n_jobs=-1)]: Done  34 out of  40 | elapsed:    8.1s remaining:    1.4s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   10.3s finished
```

Out[98]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=1000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'alpha': [0.001, 0.01, 0.1, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [99]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

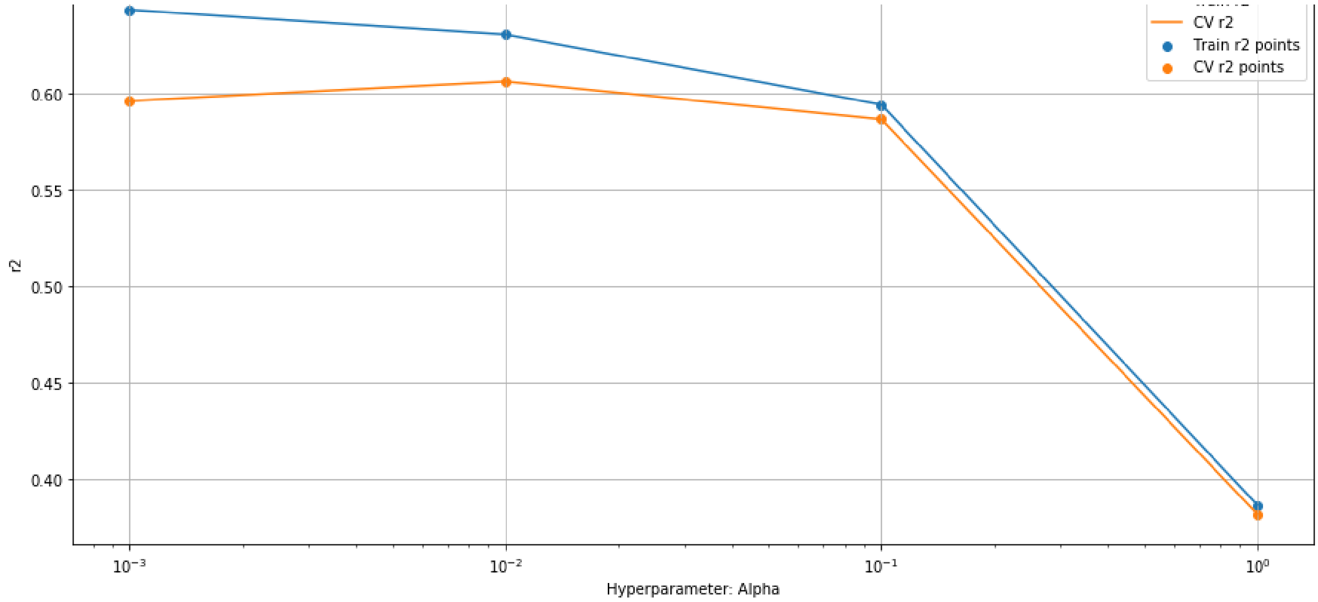
In [100]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6433845603206471 CV:0.5961162258800853
Train:0.6305972277767411 CV:0.6062484900646521
Train:0.5942080802897161 CV:0.586587430533187
Train:0.3863534725351519 CV:0.3815940361622423
```

In [101]:

```
Alpha = [0.001,0.01,0.1,1]
plt.plot(Alpha,train_r2,label='Train r2')
#https://stackoverflow.com/a/48803361/4084039
plt.plot(Alpha,cv_r2,label='CV r2')
#https://stackoverflow.com/a/48803361/4084039
plt.scatter(Alpha,train_r2,label='Train r2 points')
plt.scatter(Alpha,cv_r2,label='CV r2 points')
plt.legend()
plt.xscale('log')
plt.xlabel("Hyperparameter: Alpha")
plt.ylabel("r2")
plt.title("Hyper parameter Vs r2 plot")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6062484900646521
```

From the above plot we can see that as the alpha value decreses, the tendency to overfit increases. However with 0.01 Alpha value, there seem to be no overfitting and the CV score is maximum.

In [187]:

```
model_el_label = ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
model_el_label.fit(X_train_le,Y_train)
```

Out[187]:

```
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

In [188]:

```
pred_test_el_le=model_el_label.predict(X_test_le)
```

In [189]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_el_le]}
data = pd.DataFrame(data)
data.to_csv("submission_el_label.csv", index=False)
```

Since Elastic Net Reduces Overfitting, the model performed well with One Hot Encoded Features. Let's try Random Forest with it and see how both performs.

ElasticNet with Label Encoding + PCA Components:

In [72]:

```
#Using RandomizedSearchCV with 5_fold
neigh=ElasticNet()
parameters = {'alpha':[0.001,0.01,0.1,1]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le_PCA,Y_train)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    7.8s
[Parallel(n_jobs=-1)]: Done  34 out of  40 | elapsed:   12.2s remaining:    2.1s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   14.1s finished
```

Out[72]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=1000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'alpha': [0.001, 0.01, 0.1, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [63]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

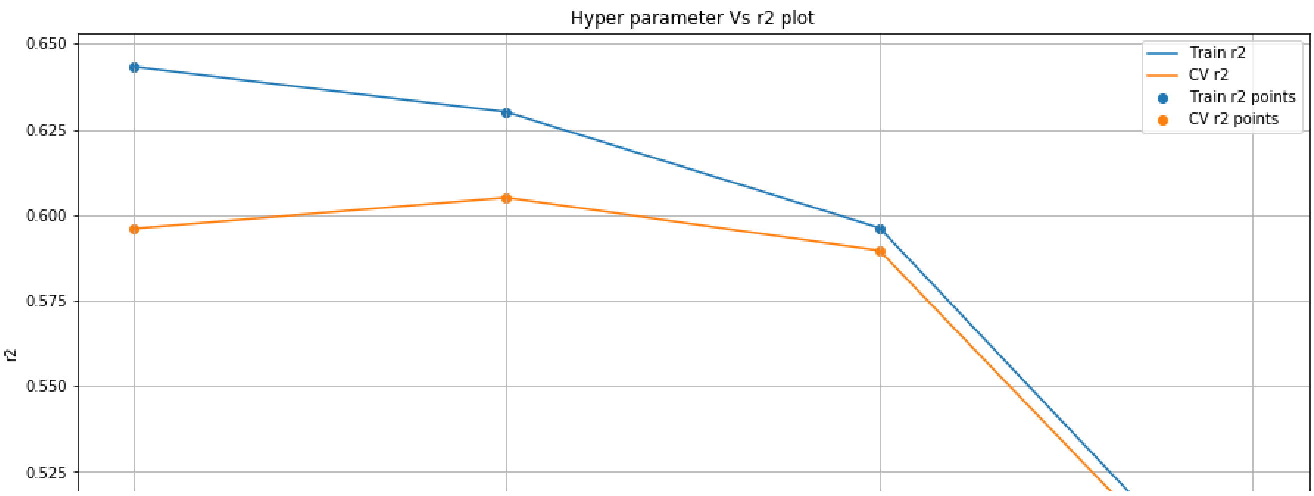
In [64]:

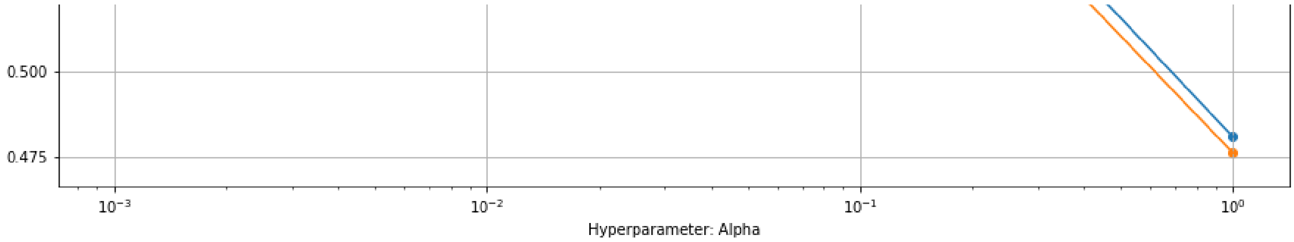
```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6433546238442829 CV:0.5960524621732116
Train:0.6300854979976347 CV:0.6051291260398493
Train:0.596269891446502 CV:0.5896501848759883
Train:0.4808212719059772 CV:0.4761545344608481
```

In [66]:

```
Alpha = [0.001,0.01,0.1,1]
plt.figure(figsize=(15,8))
plt.plot(Alpha,train_r2,label='Train r2')
#https://stackoverflow.com/a/48803361/4084039
plt.plot(Alpha,cv_r2,label='CV r2')
#https://stackoverflow.com/a/48803361/4084039
plt.scatter(Alpha,train_r2,label='Train r2 points')
plt.scatter(Alpha,cv_r2,label='CV r2 points')
plt.legend()
plt.xscale('log')
plt.xlabel("Hyperparameter: Alpha")
plt.ylabel("r2")
plt.title("Hyper parameter Vs r2 plot")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```






```
The Best Score 0.6051291260398493
```

Even with Label Encoding, the PCA components didnot improve the performance. Lets try with Ensembles.

In [73]:

```
clf.best_estimator_
```

Out[73]:

```
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

In [74]:

```
model_el_label_pca = ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
model_el_label_pca.fit(X_train_le_PCA,Y_train)
```

Out[74]:

```
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

In [75]:

```
pred_test_el_le_pca=model_el_label_pca.predict(X_test_le_PCA)
```

In [76]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_el_le_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_el_label_pca.csv", index=False)
```

Random Forest with One Hot Encoding(K-Best):

In [64]:

```
from sklearn.ensemble import RandomForestRegressor

neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
              'max_depth':[1,2,3,5,7,10],
              'min_samples_split':[2,3,4,5,6,7,8,9,10],
              'max_features': [0.95],
              'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
              'random_state':[30,42]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
```

```
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   29.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.5min finished
```

Out[64]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [1, 2, 3, 5, 7, 10],
                                        'max_features': [0.95],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [100, 150, 200, 300,
                                                         350, 500],
                                        'random_state': [30, 42]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [65]:

```python
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

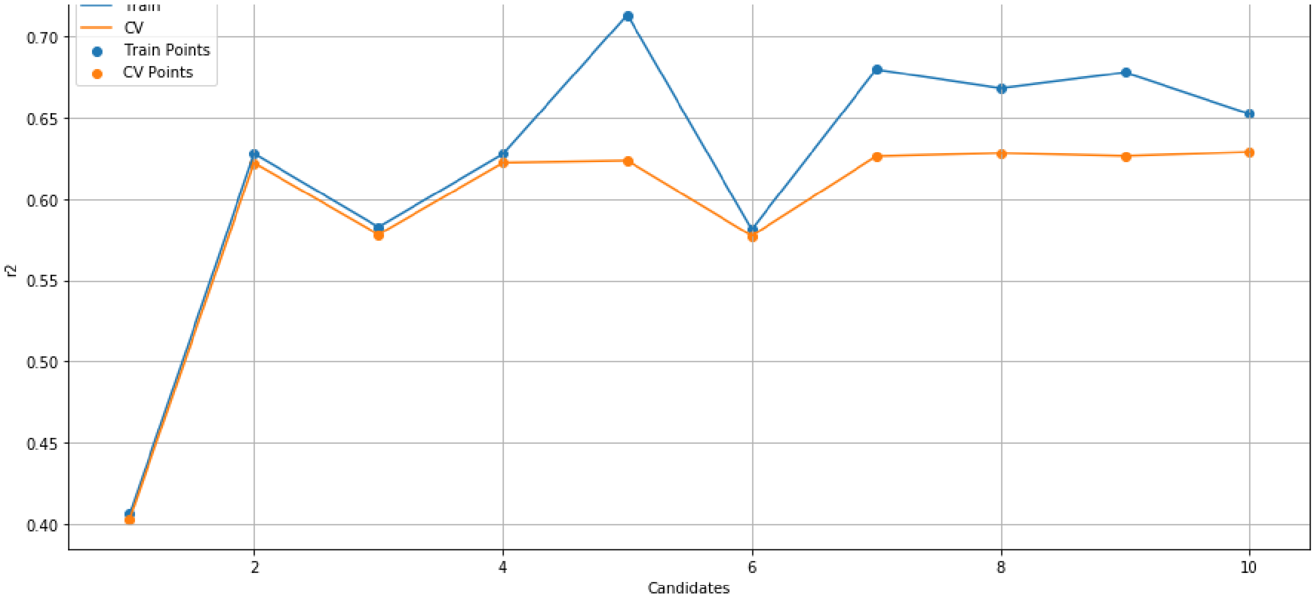
In [66]:

```python
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.40586013647808583 CV:0.4028998194049397
Train:0.6278981197441376 CV:0.6221540191827072
Train:0.5827475161120216 CV:0.5781088784929068
Train:0.627698399165009 CV:0.6222939261585287
Train:0.7135156336765881 CV:0.6235965038430735
Train:0.5812606532980403 CV:0.5773401664966583
Train:0.6797806976139157 CV:0.6262808430407196
Train:0.66822756060244 CV:0.6283353313523661
Train:0.6780152251260286 CV:0.6264481484983765
Train:0.6524105546171005 CV:0.6289963854000105
```

In [67]:

```python
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6289963854000105
```

Candidate 10 seems to have the best cv score and there seems no overfitting

In [68]:

```
clf.best_estimator_
```

Out[68]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=3,
                      min_samples_split=4, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=30, verbose=0, warm_start=False)
```

In [77]:

```
model_rf = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=3,
                      min_samples_split=4, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=30, verbose=0, warm_start=False)


model_rf.fit(X_train_ohe,Y_train)
```

Out[77]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=3,
                      min_samples_split=4, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=30, verbose=0, warm_start=False)
```

In [78]:

```
pred_test_rf = model_rf.predict(X_test_ohe)
```

In [79]:

```
data={'ID':[i for i in ID],
```

```
    'y':[j for j in pred_test_rf]}
```

In [80]:

```
data = pd.DataFrame(data)
data.to_csv("submission_rf.csv", index=False)
```

Random Forest with One Hot Encoding(K-Best) + PCA Components:

In [81]:

```
from sklearn.ensemble import RandomForestRegressor

neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
             'max_depth':[1,2,3,5,7,10],
             'min_samples_split':[2,3,4,5,6,7,8,9,10],
             'max_features': [0.95],
             'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
             'random_state':[30,42]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_PCA,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    5.0s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   30.7s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  2.4min finished
```

Out[81]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [1, 2, 3, 5, 7, 10],
                                        'max_features': [0.95],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [100, 150, 200, 300,
                                                         350, 500],
                                        'random_state': [30, 42]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [82]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

In [83]:

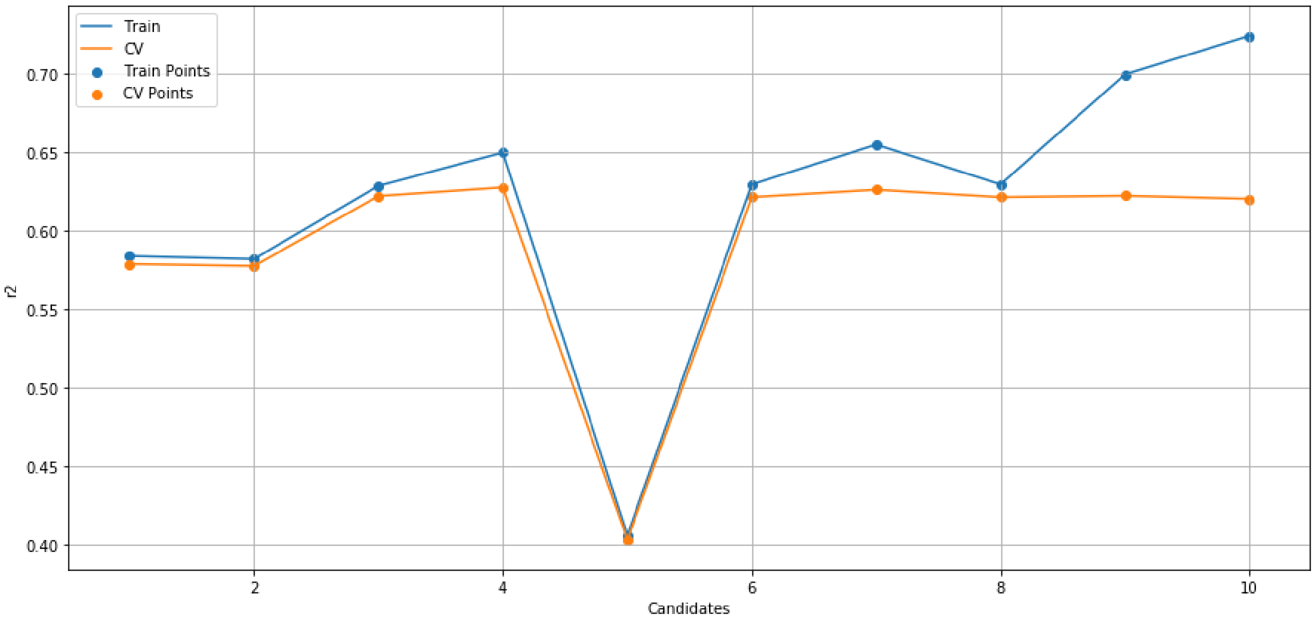
```
for i,j in zip(train_r2,cv_r2):
```

```
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.5835775115859713 CV:0.5784739463698005
Train:0.5816936136201437 CV:0.5772348705913866
Train:0.6282595666400754 CV:0.6217674044720016
Train:0.6494770760200155 CV:0.6272912779771278
Train:0.40579100868743423 CV:0.40303143489059456
Train:0.6291855436116494 CV:0.6210652246272389
Train:0.6545085578649356 CV:0.6258026544813079
Train:0.6292186195443621 CV:0.6210384289518897
Train:0.6992329958824666 CV:0.6219649882207907
Train:0.7238323772400144 CV:0.6199772532231972
```

In [84]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6272912779771278
```

Even here we see no improvement with PCA components. Let's try with Label Encoding and see how it perfroms.

In [85]:

```
clf.best_estimator_
```

Out[85]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=9,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=-1, oob_score=False,
                      random_state=30, verbose=0, warm_start=False)
```

In [86]:

```
model_rf_pca = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=9,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=-1, oob_score=False,
                      random_state=30, verbose=0, warm_start=False)


model_rf_pca.fit(X_train_ohe_PCA,Y_train)
```

Out[86]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=9,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=-1, oob_score=False,
                      random_state=30, verbose=0, warm_start=False)
```

In [87]:

```
pred_test_rf_pca = model_rf_pca.predict(X_test_ohe_PCA)
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_rf_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_rf_pca.csv", index=False)
```

Random Forest with Label Encoding:

In [106]:

```
from sklearn.ensemble import RandomForestRegressor

neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
             'max_depth':[2,3,5,7,10],
             'min_samples_split':[2,3,4,5,6,7,8,9,10],
             'max_features': [.95],
             'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   30.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  2.1min finished
```

Out[106]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   param_distributions={'max_depth': [2, 3, 5, 7, 10],
```

```
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1,
                                                                  10],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [100, 150, 200, 300,
                                                         350, 500]},
                    pre_dispatch='2*n_jobs', random_state=None, refit=True,
                    return_train_score=True, scoring='r2', verbose=5)
```

In [107]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

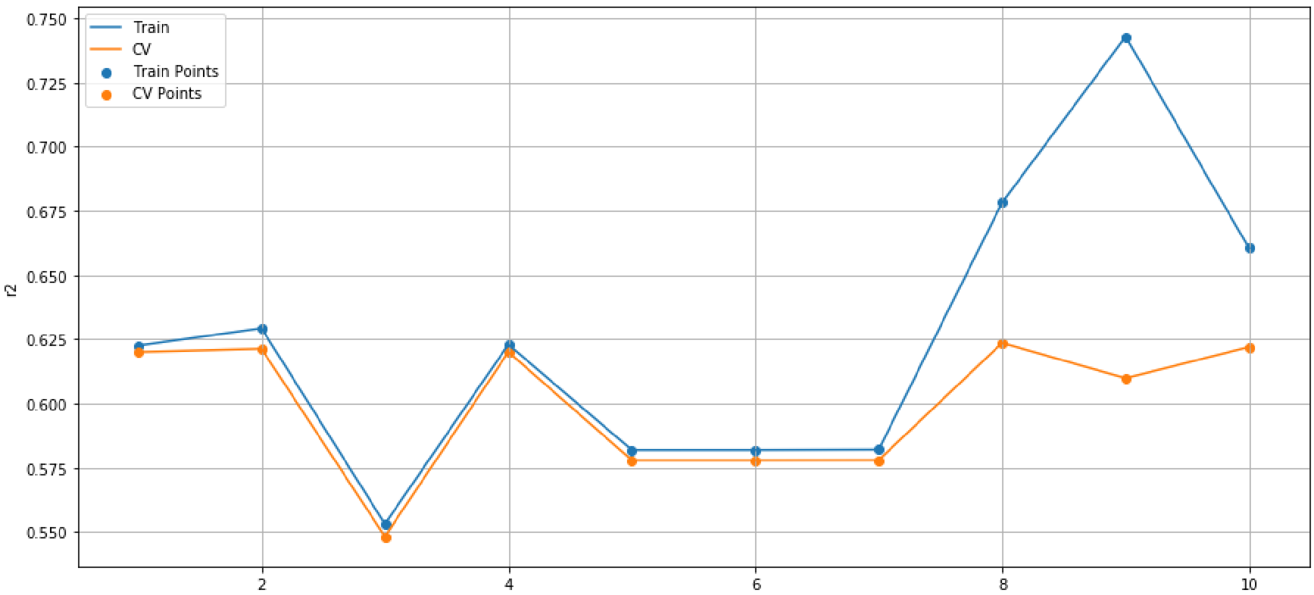
In [108]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6224652029383098 CV:0.6199696501453823
Train:0.6292038537845753 CV:0.6212464927001642
Train:0.5530626754610907 CV:0.5481569684056152
Train:0.6228867556358585 CV:0.6199799012542913
Train:0.581834559042354 CV:0.5778599056274163
Train:0.5818471604343075 CV:0.5778616966661542
Train:0.5819825433600808 CV:0.5779069936452852
Train:0.6781288819886699 CV:0.6235074198005414
Train:0.742795939197664 CV:0.609786497420799
Train:0.6604376588536514 CV:0.6219339764888597
```

In [109]:

```
candidates = list(range(1,11))
plt.figure(figsize=(10,6))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```



```
The Best Score 0.6235074198005414
```

Candidate 8 seems to have the best cv score and also there is not much difference between the train and cv score.

In [110]:

```
clf.best_estimator_
```

Out[110]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=7, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.001,
                      min_impurity_split=None, min_samples_leaf=3,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=300, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [111]:

```
model_rf_le = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=7, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.001,
                      min_impurity_split=None, min_samples_leaf=3,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=300, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
model_rf_le.fit(X_train_le,Y_train)
```

Out[111]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=7, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.001,
                      min_impurity_split=None, min_samples_leaf=3,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=300, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [112]:

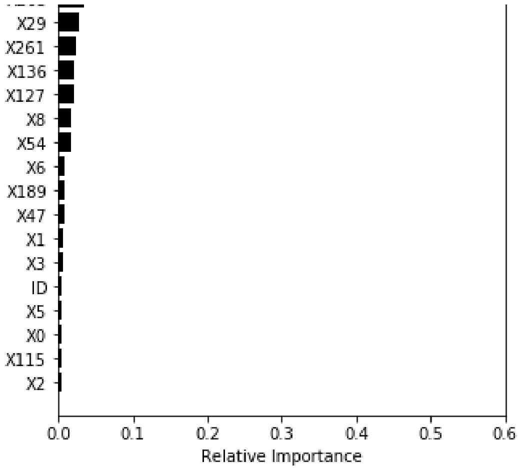
```
pred_test_rf_le = model_rf_le.predict(X_test_le)
```

In [287]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_rf_le]}
data = pd.DataFrame(data)
data.to_csv("submission_rf_label.csv", index=False)
```

In [113]:

```
features = train_df_modified.columns
importances = model_rf_le.feature_importances_
indices = (np.argsort(importances))[-20:]
plt.figure(figsize=(5,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='k', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Looking at the feature importances, it is clear that X314 is the most important feature. We can try combining some features to see how they perform as we had seen some positive correlations.

Random Forest with Label Encoding + PCA components:

In [88]:

```
from sklearn.ensemble import RandomForestRegressor

neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
             'max_depth':[2,3,5,7,10],
             'min_samples_split':[2,3,4,5,6,7,8,9,10],
             'max_features': [.95],
             'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le_PCA,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.8s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  3.4min finished
```

Out[88]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   param_distributions={'max_depth': [2, 3, 5, 7, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1,
                                                                  10],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
```

```
                                'n_estimators': [100, 150, 200, 300,
                                                 350, 500]},
                 pre_dispatch='2*n_jobs', random_state=None, refit=True,
                 return_train_score=True, scoring='r2', verbose=5)
```

In [89]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

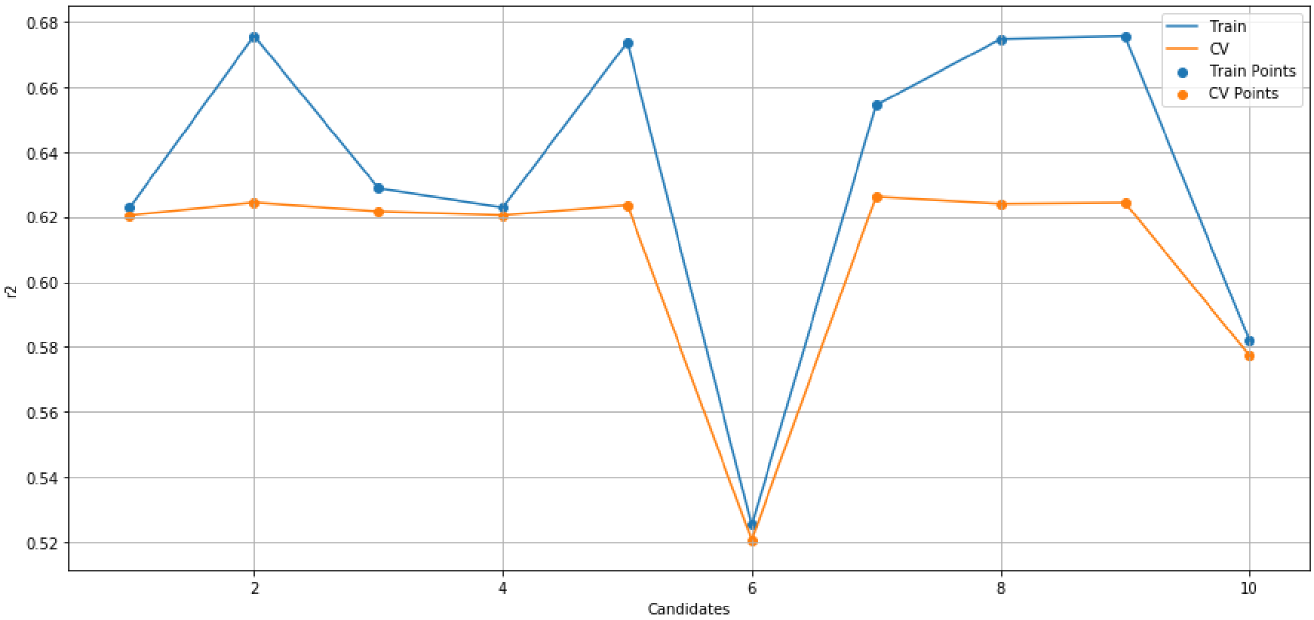
In [90]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6228228020245483 CV:0.6204295713041138
Train:0.6755581939964526 CV:0.6244366930568159
Train:0.6287555984748742 CV:0.621656133986573
Train:0.6228561814869941 CV:0.6204684943510421
Train:0.6735896621828007 CV:0.6235987438073941
Train:0.52537163054095 CV:0.5207839889814412
Train:0.6543008721757129 CV:0.6261952425694972
Train:0.6745665409708252 CV:0.6240041236653123
Train:0.6755177220028836 CV:0.6243512748548665
Train:0.5820620959507181 CV:0.5774058132101934
```

In [91]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6261952425694972
```

Here we can see some minimal improvement by adding the PCA components. Lets get the Feature importance.

In [92]:

```
clf.best_estimator_
```

Out[92]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0,
                      min_impurity_split=None, min_samples_leaf=5,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [93]:

```
model_rf_le_pca = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0,
                      min_impurity_split=None, min_samples_leaf=5,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
model_rf_le_pca.fit(X_train_le_PCA,Y_train)
```

Out[93]:

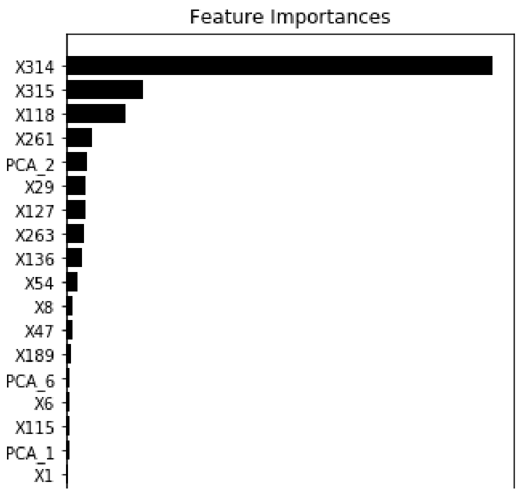
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0,
                      min_impurity_split=None, min_samples_leaf=5,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

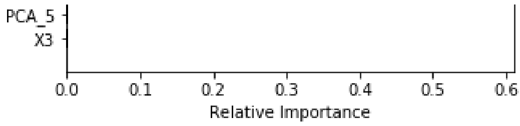
In [95]:

```
pred_test_rf_le_pca = model_rf_le_pca.predict(X_test_le_PCA)
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_rf_le_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_rf_label_pca.csv", index=False)
```

In [94]:

```
features = train_df_modified_pca.columns
importances = model_rf_le_pca.feature_importances_
indices = (np.argsort(importances))[-20:]
plt.figure(figsize=(5,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='k', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```





From the above plot we can see that the PCA component 2 has some importance but the rest have very small to no importance. X314 still stands at the top.

From the above feature importances, lets try some feature engineering:

Taking X314 and X315

In [96]:

```
#https://www.cs.waikato.ac.nz/~mhall/thesis.pdf
#https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/discussion/37700
train_df_modified['X314_plus_X315'] = train_df_modified.apply(lambda row: row.X314 + row.X315, axis=1)
test_df_modified['X314_plus_X315'] = test_df_modified.apply(lambda row: row.X314 + row.X315, axis=1)
train_df_modified_pca['X314_plus_X315'] = train_df_modified.apply(lambda row: row.X314 + row.X315, axis=1)
test_df_modified_pca['X314_plus_X315'] = test_df_modified.apply(lambda row: row.X314 + row.X315, axis=1)
```

In [97]:

```
print("Correalation between X314_plus_X315 and y is : ",Y_train.corr(train_df_modified['X314_plus_X315']))
```

```
Correalation between X314_plus_X315 and y is :   0.6990819224017307
```

The correlation between X314 and X315 is high which is excellent.

Taking X118 , X314 and X315:

In [98]:

```
train_df_modified['X118_plus_X314_plus_X315'] = train_df_modified.apply(lambda row: row.X118 + row.X314 + row.X315, axis=1)
test_df_modified['X118_plus_X314_plus_X315'] = test_df_modified.apply(lambda row: row.X118 + row.X314 + row.X315, axis=1)
train_df_modified_pca['X118_plus_X314_plus_X315'] = train_df_modified.apply(lambda row: row.X118 + row.X314 + row.X315, axis=1)
test_df_modified_pca['X118_plus_X314_plus_X315'] = test_df_modified.apply(lambda row: row.X118 + row.X314 + row.X315, axis=1)
```

In [77]:

```
print("Correalation between X118_plus_X314_plus_X315 and y is :  ",Y_train.corr(train_df_modified['X118_plus_X314_plus_X315']))
```

```
Correalation between X118_plus_X314_plus_X315 and y is :   0.6837266223799761
```

The correlation between X118, X314 and X315 is high which is excellent.

Taking X118 and X263

In [99]:

```
train_df_modified['X118_plus_X263'] = train_df_modified.apply(lambda row: row.X118 + row.X263, axis=1)
test_df_modified['X118_plus_X263'] = test_df_modified.apply(lambda row: row.X118 + row.X263, axis=1)
```

```
train_df_modified_pca['X118_plus_X263'] = train_df_modified.apply(lambda row: row.X118 + row.X263, axis=1)
test_df_modified_pca['X118_plus_X263'] = test_df_modified.apply(lambda row: row.X118 + row.X263, axis=1)
```

In [79]:

```
print("Correalation between X118_plus_X263 and y is : ",Y_train.corr(train_df_modified['X118_plus_X263']))
```

```
Correalation between X118_plus_X263 and y is :   0.3864652751823678
```

Taking X29, X118 and X263

In [100]:

```
train_df_modified['X29_plus_X118_plus_X263'] = train_df_modified.apply(lambda row: row.X29 + row.X118 + row.X263, axis=1)
test_df_modified['X29_plus_X118_plus_X263'] = test_df_modified.apply(lambda row: row.X29 + row.X118 + row.X263, axis=1)
train_df_modified_pca['X29_plus_X118_plus_X263'] = train_df_modified.apply(lambda row: row.X29 + row.X118 + row.X263, axis=1)
test_df_modified_pca['X29_plus_X118_plus_X263'] = test_df_modified.apply(lambda row: row.X29 + row.X118 + row.X263, axis=1)
```

In [83]:

```
print("Correalation between X29_plus_X118_plus_X263 and y is :  ",Y_train.corr(train_df_modified['X29_plus_X118_plus_X263']))
```

```
Correalation between X29_plus_X118_plus_X263 and y is :   0.2911340078121632
```

Now Adding these features to the label encoded features:

In [101]:

```
from scipy.sparse import hstack
print('Final Matrix:')
X_train_le_corr = hstack((X_train_le,train_df_modified['X314_plus_X315'].values.reshape(-1,1),train_df_modified['X118_plus_X314_plus_X315'].values.reshape(-1,1),train_df_modified['X118_plus_X263'].values.reshape(-1,1),train_df_modified['X29_plus_X118_plus_X263'].values.reshape(-1,1))).tocsr()
print(X_train_le_corr.shape)
X_test_le_corr = hstack((X_test_le,test_df_modified['X314_plus_X315'].values.reshape(-1,1),test_df_modified['X118_plus_X314_plus_X315'].values.reshape(-1,1),test_df_modified['X118_plus_X263'].values.reshape(-1,1),test_df_modified['X29_plus_X118_plus_X263'].values.reshape(-1,1))).tocsr()
print(X_test_le_corr.shape)
```

```
Final Matrix:
(4194, 322)
(4209, 322)
```

Lets also add them to the One Hot Encoded Features:

In [102]:

```
from scipy.sparse import hstack
print('Final Matrix:')
X_train_ohe_corr = hstack((X_train_ohe,train_df_modified['X314_plus_X315'].values.reshape(-1,1),train_df_modified['X118_plus_X314_plus_X315'].values.reshape(-1,1),train_df_modified['X118_plus_X263'].values.reshape(-1,1),train_df_modified['X29_plus_X118_plus_X263'].values.reshape(-1,1))).tocsr()
print(X_train_ohe_corr.shape)
X_test_ohe_corr = hstack((X_test_ohe,test_df_modified['X314_plus_X315'].values.reshape(-1,1),test_df_modified['X118_plus_X314_plus_X315'].values.reshape(-1,1),test_df_modified['X118_plus_X263'].values.reshape(-1,1),test_df_modified['X29_plus_X118_plus_X263'].values.reshape(-1,1))).tocsr()
print(X_test_ohe_corr.shape)
```

```
Final Matrix:
(4194, 322)
```

(4194, 322)
(4209, 322)

Now Adding these features to the label encoded + PCA features:

In [103]:

```
from scipy.sparse import hstack
print('Final PCA Matrix:')
X_train_le_PCA_corr = hstack((X_train_le_PCA,train_df_modified['X314_plus_X315'].values.reshape(-1,
1),train_df_modified['X118_plus_X314_plus_X315'].values.reshape(-1,1),train_df_modified['X118_plus_
X263'].values.reshape(-1,1),train_df_modified['X29_plus_X118_plus_X263'].values.reshape(-1,1))).toc
sr()
print(X_train_le_PCA_corr.shape)
X_test_le_PCA_corr = hstack((X_test_le_PCA,test_df_modified['X314_plus_X315'].values.reshape(-1,1),
test_df_modified['X118_plus_X314_plus_X315'].values.reshape(-1,1),test_df_modified['X118_plus_X263'
].values.reshape(-1,1),test_df_modified['X29_plus_X118_plus_X263'].values.reshape(-1,1))).tocsr()
print(X_test_le_PCA_corr.shape)
```

```
Final PCA Matrix:
(4194, 328)
(4209, 328)
```

Lets also add them to the One Hot Encoded Features + PCA features:

In [104]:

```
from scipy.sparse import hstack
print('Final PCA Matrix:')
X_train_ohe_PCA_corr = hstack((X_train_ohe_PCA,train_df_modified['X314_plus_X315'].values.reshape(
-1,1),train_df_modified['X118_plus_X314_plus_X315'].values.reshape(-1,1),train_df_modified['X118_pl
us_X263'].values.reshape(-1,1),train_df_modified['X29_plus_X118_plus_X263'].values.reshape(-1,1))).
tocsr()
print(X_train_ohe_PCA_corr.shape)
X_test_ohe_PCA_corr = hstack((X_test_ohe_PCA,test_df_modified['X314_plus_X315'].values.reshape(-1,1
),test_df_modified['X118_plus_X314_plus_X315'].values.reshape(-1,1),test_df_modified['X118_plus_X26
3'].values.reshape(-1,1),test_df_modified['X29_plus_X118_plus_X263'].values.reshape(-1,1))).tocsr()
print(X_test_ohe_PCA_corr.shape)
```

```
Final PCA Matrix:
(4194, 328)
(4209, 328)
```

Now lets check again with Random Forest:

Random Forest with One Hot(K-Best) + Correlation Features:

In [88]:

```
neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
              'max_depth':[2,3,5,7,10],
              'min_samples_split':[2,3,4,5,6,7,8,9,10],
              'max_features': [.95],
              'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
              'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.4s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   33.3s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   53.8s finished
```

Out[88]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   param_distributions={'max_depth': [2, 3, 5, 7, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1,
                                                                  10],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [100, 150, 200, 300,
                                                         350, 500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [90]:

```python
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

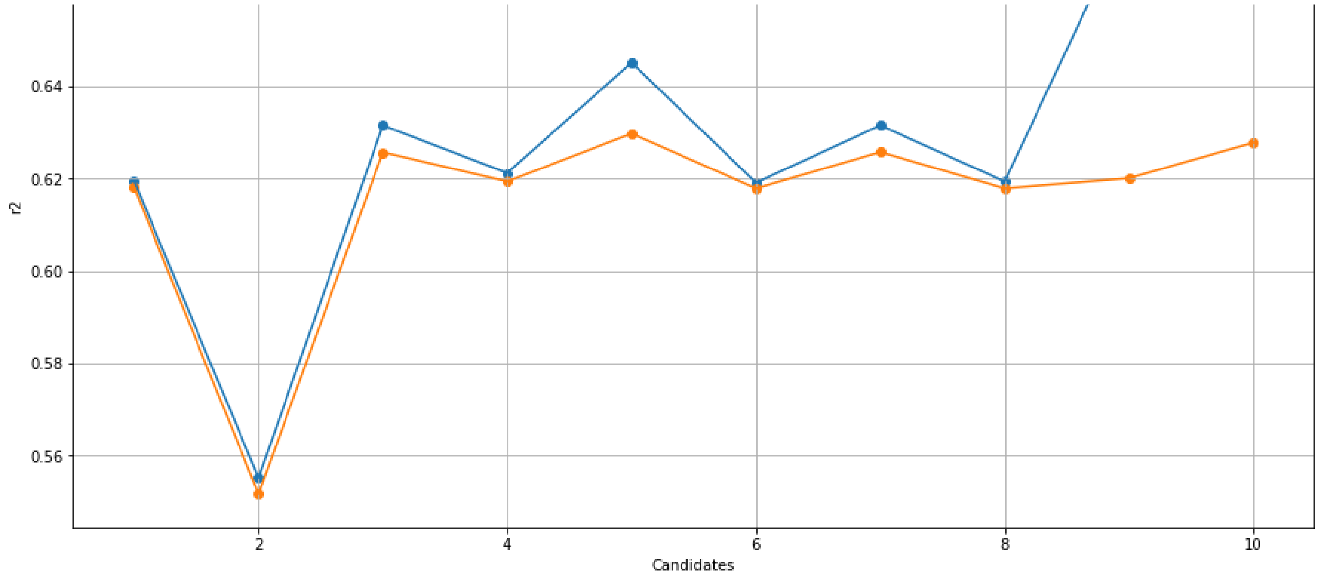
In [91]:

```python
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6194719551628827 CV:0.6180951754403438
Train:0.5552613366344892 CV:0.5517620249977933
Train:0.6315654520780074 CV:0.6257536761130631
Train:0.6212581091173065 CV:0.6194222619458919
Train:0.6451791109864617 CV:0.6298087269540957
Train:0.6191560979843267 CV:0.6179190395793939
Train:0.6315703495570235 CV:0.6257892054899995
Train:0.6194188774951195 CV:0.6179242814346162
Train:0.6767475349889928 CV:0.620137989993172
Train:0.6727266748429491 CV:0.6278397294140874
```

In [92]:

```python
candidates = list(range(1,11))
plt.figure(figsize=(15,8))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6298087269540957
```

From the above plot we see some improvements by the addition of the new features.

In [93]:

```
clf.best_estimator_
```

Out[93]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=6, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [105]:

```
model_rf_ohe = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=6, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)

model_rf_ohe.fit(X_train_ohe_corr,Y_train)
```

Out[105]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=6, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [106]:

```
pred_test_rf_ohe_corr = model_rf_ohe.predict(X_test_ohe_corr)
```

In [107]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_rf_ohe_corr]}
data = pd.DataFrame(data)
```

```
data.to_csv("submission_rf_ohe_corr.csv", index=False)
```

Random Forest with One Hot(K-Best) + PCA Components + Correlation Features:

In [96]:

```
neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
             'max_depth':[2,3,5,7,10],
             'min_samples_split':[2,3,4,5,6,7,8,9,10],
             'max_features': [.95],
             'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_PCA_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.1s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.7min finished
```

Out[96]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   param_distributions={'max_depth': [2, 3, 5, 7, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1,
                                                                  10],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [100, 150, 200, 300,
                                                         350, 500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [97]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

In [98]:

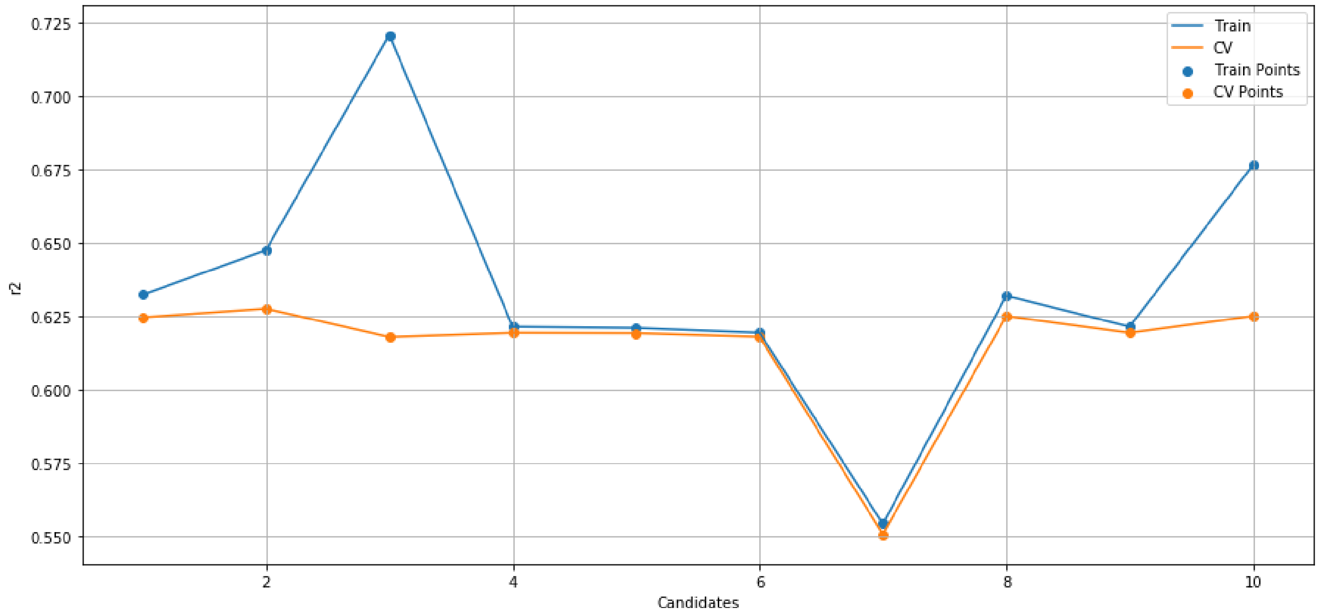
```
for i,j in zip(train_r2,cv_r2):
    print("Train:{}  CV:{}".format(i,j))
```

```
Train:0.6324175732802452  CV:0.6246866437334818
Train:0.6475198571364849  CV:0.6276369842690707
Train:0.7208066408182654  CV:0.6180810308762139
```

```
Train:0.6215397294497171  CV:0.6194857987092968
Train:0.6211255143407077  CV:0.6193803470381302
Train:0.6195136024501534  CV:0.6181386636036571
Train:0.5544886437337351  CV:0.5509540999699605
Train:0.6320409198492671  CV:0.6251237720078686
Train:0.6215396905304476  CV:0.6194857987092971
Train:0.6765466024726843  CV:0.6251418089427515
```

In [99]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6276369842690707
```

Again we see decrese in performance with PCA features.

In [100]:

```
clf.best_estimator_
```

Out[100]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [108]:

```
model_rf_ohe_pca = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
```

```
                        n_estimators=150, n_jobs=-1, oob_score=False,
                        random_state=42, verbose=0, warm_start=False)

model_rf_ohe_pca.fit(X_train_ohe_PCA_corr,Y_train)
```

Out[108]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [109]:

```
pred_test_rf_ohe_pca = model_rf_ohe_pca.predict(X_test_ohe_PCA_corr)
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_rf_ohe_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_rf_pca_ohe_corr.csv", index=False)
```

Random Forest with Label + Correlation Features:

In [123]:

```
neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
             'max_depth':[2,3,5,7,10],
             'min_samples_split':[2,3,4,5,6,7,8,9,10],
             'max_features': [.95],
             'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    5.8s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  2.1min finished
```

Out[123]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   param_distributions={'max_depth': [2, 3, 5, 7, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1,
                                                                  10],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
```

```
                                          'n_estimators': [100, 150, 200, 300,
                                                           350, 500]},
                       pre_dispatch='2*n_jobs', random_state=None, refit=True,
                       return_train_score=True, scoring='r2', verbose=5)
```

In [124]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

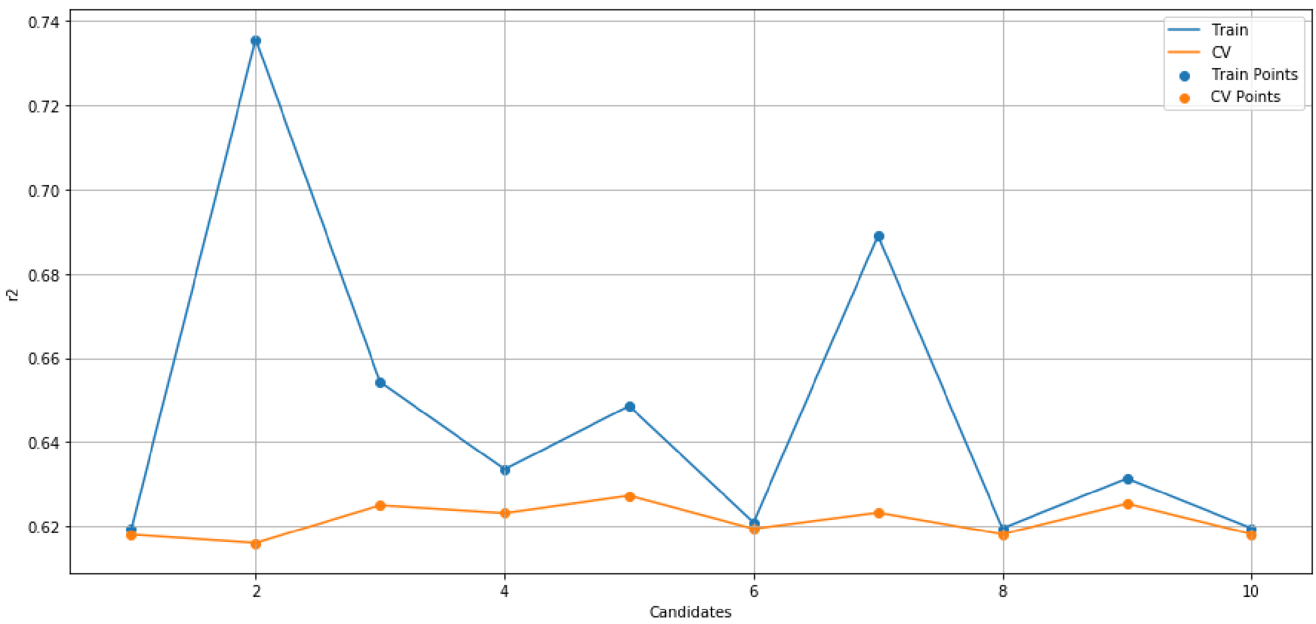
In [125]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6192018443474979 CV:0.617968092687764
Train:0.7356937522369491 CV:0.6159777162083706
Train:0.6542065066548359 CV:0.6249025528182883
Train:0.6334699868635237 CV:0.6230345194700629
Train:0.6485234717276304 CV:0.6272154483295912
Train:0.6208335239992933 CV:0.6193363796880387
Train:0.6889716243270283 CV:0.6231483280335468
Train:0.6194409569869214 CV:0.6180697234468345
Train:0.6313081277548646 CV:0.6252982674256713
Train:0.6194631252938168 CV:0.618097787104906
```

In [126]:

```
candidates = list(range(1,11))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6272154483295912
```

From the above plot we can see that some combinations of hyper-parameters tend to overfit.

In [127]:

```
clf.best_estimator_
```

Out[127]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=7, min_weight_fraction_leaf=0.0,
                      n_estimators=300, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [143]:

```
model_rf_le = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=7, min_weight_fraction_leaf=0.0,
                      n_estimators=300, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)

model_rf_le.fit(X_train_le_corr,Y_train)
```

Out[143]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=5, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.1,
                      min_impurity_split=None, min_samples_leaf=8,
                      min_samples_split=7, min_weight_fraction_leaf=0.0,
                      n_estimators=300, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [129]:

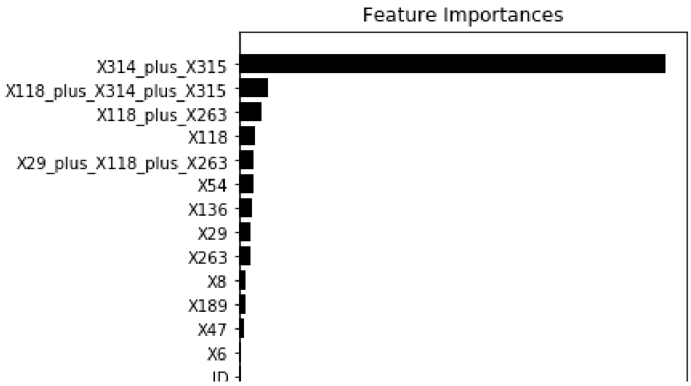
```
pred_test_rf_le_corr = model_rf_le.predict(X_test_le_corr)
```

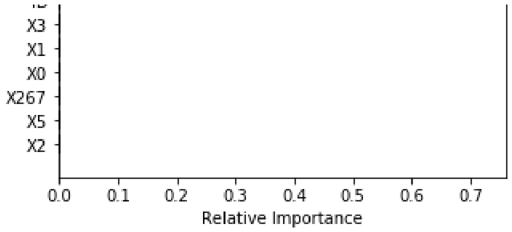
In [190]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_rf_le_corr]}
data = pd.DataFrame(data)
data.to_csv("submission_rf_le_corr.csv", index=False)
```

In [130]:

```
features = train_df_modified.columns
importances = model_rf_le.feature_importances_
indices = (np.argsort(importances))[-20:]
plt.figure(figsize=(5,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='k', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```





The newly added features have good contribution to the models. Now lets move forward with it.

Random Forest with Label + PCA components + Correlation Features:

In [111]:

```
neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
              'max_depth':[2,3,5,7,10],
              'min_samples_split':[2,3,4,5,6,7,8,9,10],
              'max_features': [.95],
              'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
              'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le_PCA_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   25.5s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  3.3min finished
```

Out[111]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   param_distributions={'max_depth': [2, 3, 5, 7, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1,
                                                                  10],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [100, 150, 200, 300,
                                                         350, 500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [112]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

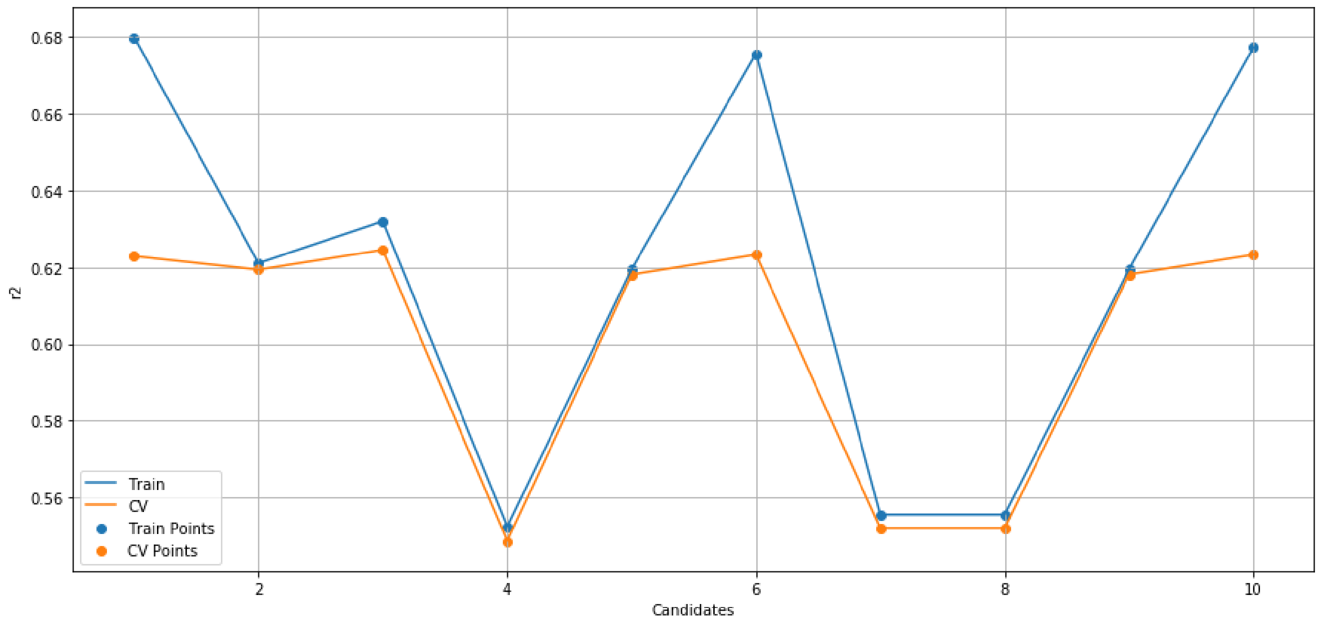
In [113]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6799751563421534 CV:0.6229164403858526
Train:0.6210805169019136 CV:0.619390940522801
Train:0.6319815729760734 CV:0.6245333381882067
Train:0.5523113465360013 CV:0.5486258146846824
Train:0.6195131031120934 CV:0.6181113989894185
Train:0.6757257655575323 CV:0.6233022267506353
Train:0.5554926127725347 CV:0.5519653067861569
Train:0.5554926127725346 CV:0.5519653067861569
Train:0.6194591890351718 CV:0.618139602057531
Train:0.6773783313318548 CV:0.6232879324037267
```

In [114]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6245333381882067
```

Again, there is no improvement with the PCA features.

In [115]:

```
clf.best_estimator_
```

Out[115]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=3, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.01,
                      min_impurity_split=None, min_samples_leaf=6,
                      min_samples_split=5, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=-1, oob_score=False,
```

```
                      random_state=42, verbose=0, warm_start=False)
```

In [116]:

```
model_rf_le_pca_corr = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=3, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.01,
                      min_impurity_split=None, min_samples_leaf=6,
                      min_samples_split=5, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)

model_rf_le_pca_corr.fit(X_train_le_PCA_corr,Y_train)
```

Out[116]:

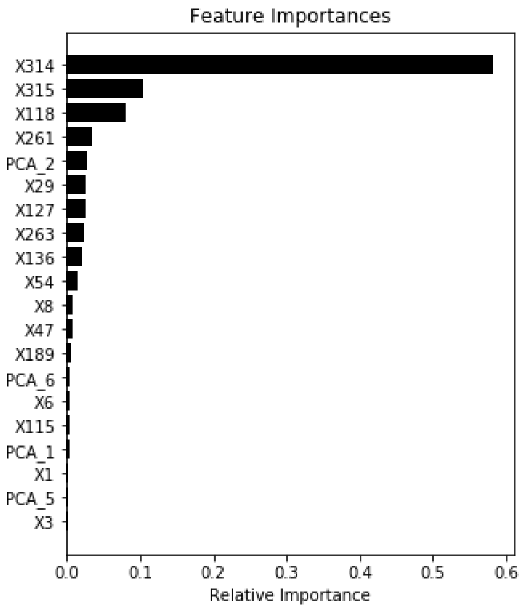
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=3, max_features=0.95, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.01,
                      min_impurity_split=None, min_samples_leaf=6,
                      min_samples_split=5, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [117]:

```
pred_test_rf_label_pca = model_rf_le_pca_corr.predict(X_test_le_PCA_corr)
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_rf_label_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_rf_pca_label_corr.csv", index=False)
```

In [118]:

```
features = train_df_modified_pca.columns
importances = model_rf_le_pca.feature_importances_
indices = (np.argsort(importances))[-20:]
plt.figure(figsize=(5,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='k', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Here we see that only the 2nd component from PCA has some importance, the rest have very less to no importance. Let's try with other models.

XG Boost with One Hot(K-Best) + Correlation Features:

In [121]:

```
from xgboost import XGBRegressor
```

In [103]:

```
neigh=XGBRegressor(random_state=42,n_jobs=-1)
parameters = {'learning_rate':[0.001,0.01,0.05,0.1,1],
              'n_estimators':[100,150,200,500],
              'max_depth':[2,3,5,10],
              'colsample_bytree':[0.1,0.5,0.7,1],
              'subsample':[0.2,0.3,0.5,1],
              'gamma':[1e-2,1e-3,0,0.1,0.01,0.5,1],
              'reg_alpha':[1e-5,1e-3,1e-1,1,1e1]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   13.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   58.7s finished
```

```
[21:57:45] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[103]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:linear',
                                          random_state=42, reg_alp...
                   param_distributions={'colsample_bytree': [0.1, 0.5, 0.7, 1],
                                        'gamma': [0.01, 0.001, 0, 0.1, 0.01,
                                                  0.5, 1],
                                        'learning_rate': [0.001, 0.01, 0.05,
                                                          0.1, 1],
                                        'max_depth': [2, 3, 5, 10],
                                        'n_estimators': [100, 150, 200, 500],
                                        'reg_alpha': [1e-05, 0.001, 0.1, 1,
                                                      10.0],
                                        'subsample': [0.2, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [104]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

In [105]:

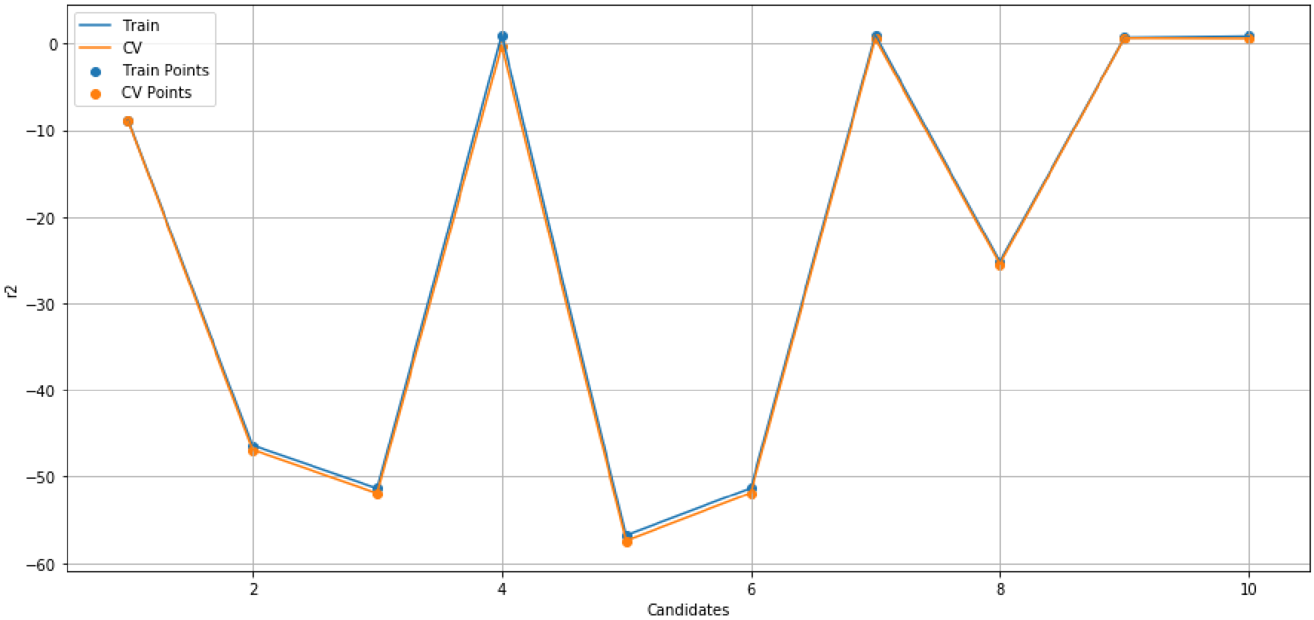
```
for i,j in zip(train_r2,cv_r2):
    print("Train:{}  CV:{}".format(i,j))
```

```
Train:-8.817719594116848  CV:-8.90009310479437
Train:-46.41941994356934  CV:-46.92474987553644
```

```
Train:-51.370167472810394  CV:-51.93260364998207
Train:0.9988780707147674  CV:-0.29439914299757214
Train:-56.782332031811926  CV:-57.40663383965817
Train:-51.34588888176463  CV:-51.90691819552019
Train:0.9590294154905082  CV:0.5670218974810713
Train:-25.201203971505688  CV:-25.46249746290781
Train:0.6718489243607368  CV:0.6240673141289591
Train:0.8327582311699638  CV:0.5843458048540501
```

In [106]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6240673141289591
```

The new features seem to work well with xgboost as we see very less tendency to overfit.

In [107]:

```
clf.best_estimator_
```

Out[107]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.1,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=5, min_child_weight=1, missing=None, n_estimators=500,
             n_jobs=-1, nthread=None, objective='reg:linear', random_state=42,
             reg_alpha=1, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.2, verbosity=1)
```

In [122]:

```
model_xgb_ohe = XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.1,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=5, min_child_weight=1, missing=None, n_estimators=500,
```

```
            n_jobs=-1, nthread=None, objective='reg:linear', random_state=42,
            reg_alpha=1, reg_lambda=1, scale_pos_weight=1, seed=None,
            silent=None, subsample=0.2, verbosity=1)

model_xgb_ohe.fit(X_train_ohe_corr,Y_train)
```

```
[16:13:35] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[122]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.1,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=5, min_child_weight=1, missing=None, n_estimators=500,
             n_jobs=-1, nthread=None, objective='reg:linear', random_state=42,
             reg_alpha=1, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.2, verbosity=1)
```

In [153]:

```
pred_test_xgb_ohe = model_xgb_ohe.predict(X_test_ohe_corr)
```

In [154]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_xgb_ohe]}
data = pd.DataFrame(data)
data.to_csv("submission_xgb_ohe_corr.csv", index=False)
```

XG Boost with One Hot(K-Best) + Correlation Features + PCA:

In [128]:

```
neigh=XGBRegressor(random_state=42,n_jobs=-1)
parameters = {'learning_rate':[0.001,0.01,0.05,0.1,1],
             'n_estimators':[100,150,200,500],
             'max_depth':[2,3,5,10],
             'colsample_bytree':[0.1,0.5,0.7,1],
             'subsample':[0.2,0.3,0.5,1],
             'gamma':[1e-2,1e-3,0,0.1,0.01,0.5,1],
             'reg_alpha':[1e-5,1e-3,1e-1,1,1e1]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_PCA_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    1.5s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   24.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   38.4s finished
```

```
[16:19:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[128]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:linear',
```

```
                 random_state=42, reg_alp...
       param_distributions={'colsample_bytree': [0.1, 0.5, 0.7, 1],
                            'gamma': [0.01, 0.001, 0, 0.1, 0.01,
                                      0.5, 1],
                            'learning_rate': [0.001, 0.01, 0.05,
                                              0.1, 1],
                            'max_depth': [2, 3, 5, 10],
                            'n_estimators': [100, 150, 200, 500],
                            'reg_alpha': [1e-05, 0.001, 0.1, 1,
                                          10.0],
                            'subsample': [0.2, 0.3, 0.5, 1]},
       pre_dispatch='2*n_jobs', random_state=None, refit=True,
       return_train_score=True, scoring='r2', verbose=5)
```

In [129]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

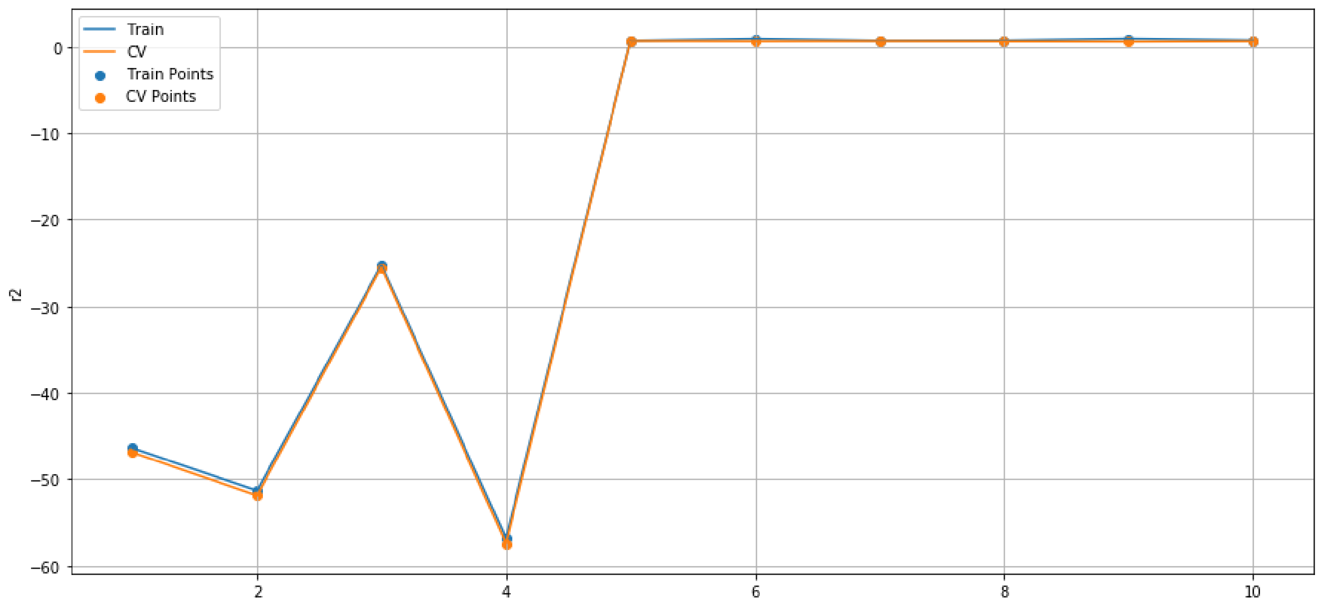
In [130]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{}  CV:{}".format(i,j))
```

```
Train:-46.402052410687666  CV:-46.90612044916012
Train:-51.31000312257861  CV:-51.87048074468812
Train:-25.26569481585763  CV:-25.528536506327328
Train:-56.76340022617859  CV:-57.38734198612519
Train:0.6498168246884497  CV:0.6249606924536788
Train:0.8597056558314582  CV:0.5939104176761425
Train:0.6561807884379848  CV:0.61272121646475
Train:0.6735289997329108  CV:0.5808003901174412
Train:0.8725801484773126  CV:0.552735617001503
Train:0.6962066642787013  CV:0.5956462917122277
```

In [131]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```



```
The Best Score 0.6249606924536788
```

Here we see very small improvement with PCA features

In [132]:

```
clf.best_estimator_
```

Out[132]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.5, gamma=0.5,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=500,
             n_jobs=-1, nthread=None, objective='reg:linear', random_state=42,
             reg_alpha=0.1, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.3, verbosity=1)
```

In [133]:

```
model_xgb_ohe_pca = XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.5, gamma=0.5,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=500,
             n_jobs=-1, nthread=None, objective='reg:linear', random_state=42,
             reg_alpha=0.1, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.3, verbosity=1)

model_xgb_ohe_pca.fit(X_train_ohe_PCA_corr,Y_train)
```

```
[16:21:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[133]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.5, gamma=0.5,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=500,
             n_jobs=-1, nthread=None, objective='reg:linear', random_state=42,
             reg_alpha=0.1, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.3, verbosity=1)
```

In [135]:

```
pred_test_xgb_ohe_pca = model_xgb_ohe_pca.predict(X_test_ohe_PCA_corr)
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_xgb_ohe_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_xgb_ohe_corr_pca.csv", index=False)
```

XG Boost with Label + Correlation Features:

In [155]:

```
neigh=XGBRegressor(random_state=42, n_jobs=-1)
parameters = {'learning_rate':[0.001,0.01,0.05,0.1,1],
              'n_estimators':[100,150,200,350,400,500],
              'max_depth':[2,3,5,7,8,10],
              'colsample_bytree':[0.1,0.2,0.3,0.4,0.5,0.7,0.8,1],
              'max_features': [.95],
              'subsample':[0.2,0.3,0.5,0.6,0.7,0.8,1],
              'gamma':[1e-2,1e-3,0,0.1,0.01,0.5,1],
              'reg_alpha':[1e-5,1e-3,1e-1,1,1e1]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    5.6s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   39.3s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.1min finished
```

```
[21:55:15] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[155]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:linear',
                                          random_state=42, reg_alp...
                                                              0.5, 0.7, 0.8, 1],
                                        'gamma': [0.01, 0.001, 0, 0.1, 0.01,
                                                  0.5, 1],
                                        'learning_rate': [0.001, 0.01, 0.05,
                                                          0.1, 1],
                                        'max_depth': [2, 3, 5, 7, 8, 10],
                                        'max_features': [0.95],
                                        'n_estimators': [100, 150, 200, 350,
                                                         400, 500],
                                        'reg_alpha': [1e-05, 0.001, 0.1, 1,
                                                      10.0],
                                        'subsample': [0.2, 0.3, 0.5, 0.6, 0.7,
                                                      0.8, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [156]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

In [157]:

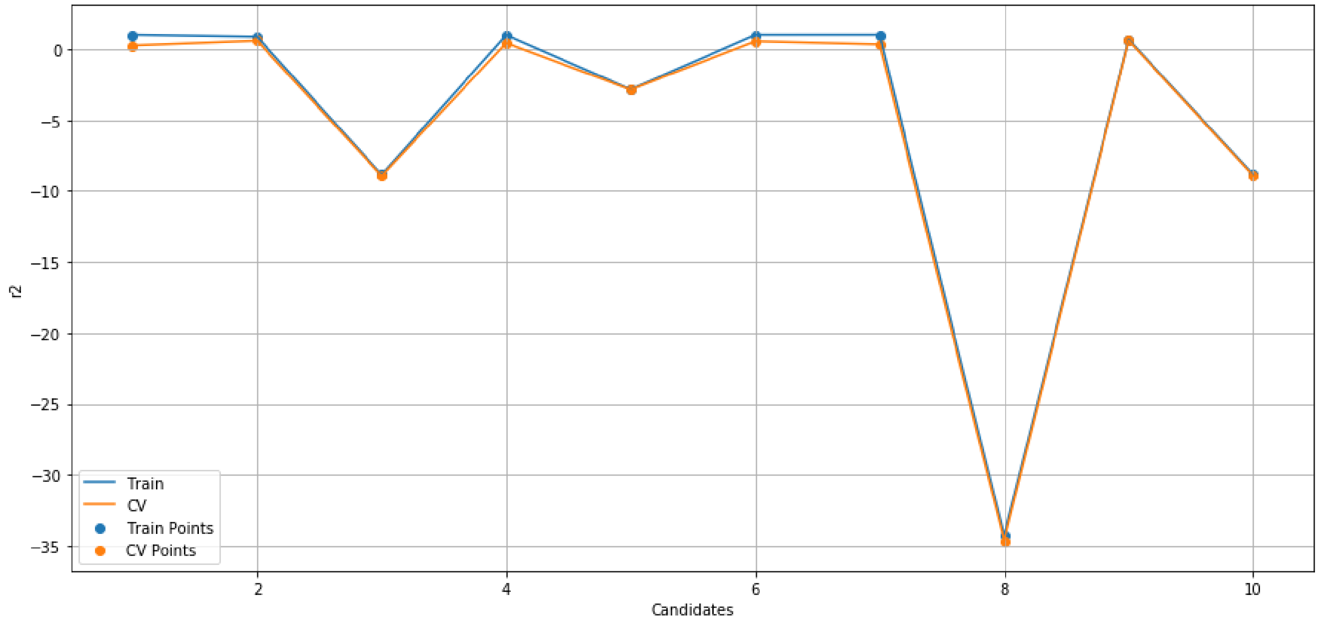
```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.9999480375511396 CV:0.2597710021595677
Train:0.8658404584134534 CV:0.5877504330189331
Train:-8.847812999716734 CV:-8.930636304196359
Train:0.990111547733156 CV:0.43299599538816125
Train:-2.8357825636036416 CV:-2.850611840912435
Train:0.9988664722028933 CV:0.5502340497520202
Train:0.9999825923459469 CV:0.3356770730090916
Train:-34.28237686496566 CV:-34.64705175732184
Train:0.6782960529883043 CV:0.6061904122236089
Train:-8.849545787115975 CV:-8.931780705525552
```

In [158]:

```
candidates = list(range(1,11))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
```

```
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6061904122236089
```

The new features seem to work well with xgboost as we see very less tendency to overfit.

In [159]:

```
clf.best_estimator_
```

Out[159]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.1,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=2, max_features=0.95, min_child_weight=1, missing=None,
             n_estimators=350, n_jobs=-1, nthread=None, objective='reg:linear',
             random_state=42, reg_alpha=1, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=0.7, verbosity=1)
```

In [160]:

```
model_xgb_label = XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.1,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=2, max_features=0.95, min_child_weight=1, missing=None,
             n_estimators=350, n_jobs=-1, nthread=None, objective='reg:linear',
             random_state=42, reg_alpha=1, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=0.7, verbosity=1)

model_xgb_label.fit(X_train_le_corr,Y_train)
```

```
[21:56:49] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[160]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.1,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=2, max_features=0.95, min_child_weight=1, missing=None,
             n_estimators=350, n_jobs=-1, nthread=None, objective='reg:linear',
             random_state=42, reg_alpha=1, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=0.7, verbosity=1)
```

In [161]:

```
pred_test_xgb_label = model_xgb_label.predict(X_test_le_corr)
```

In [162]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_xgb_label]}
data = pd.DataFrame(data)
data.to_csv("submission_xgb_label.csv", index=False)
```

XG Boost with Label + Correlation Features + PCA:

In [150]:

```
neigh=XGBRegressor(random_state=42, n_jobs=-1)
parameters = {'learning_rate':[0.001,0.01,0.05,0.1,1],
              'n_estimators':[100,150,200,350,400,500],
              'max_depth':[2,3,5,7,8,10],
              'colsample_bytree':[0.1,0.2,0.3,0.4,0.5,0.7,0.8,1],
              'max_features': [.95],
              'subsample':[0.2,0.3,0.5,0.6,0.7,0.8,1],
              'gamma':[1e-2,1e-3,0,0.1,0.01,0.5,1],
              'reg_alpha':[1e-5,1e-3,1e-1,1,1e1]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le_PCA_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   26.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   44.3s finished
```

```
[16:46:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[150]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:linear',
                                          random_state=42, reg_alp...
                                                              0.5, 0.7, 0.8, 1],
                                        'gamma': [0.01, 0.001, 0, 0.1, 0.01,
                                                  0.5, 1],
                                        'learning_rate': [0.001, 0.01, 0.05,
                                                          0.1, 1],
                                        'max_depth': [2, 3, 5, 7, 8, 10],
                                        'max_features': [0.95],
                                        'n_estimators': [100, 150, 200, 350,
                                                         400, 500],
                                        'reg_alpha': [1e-05, 0.001, 0.1, 1,
                                                      10.0],
                                        'subsample': [0.2, 0.3, 0.5, 0.6, 0.7,
                                                      0.8, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [151]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

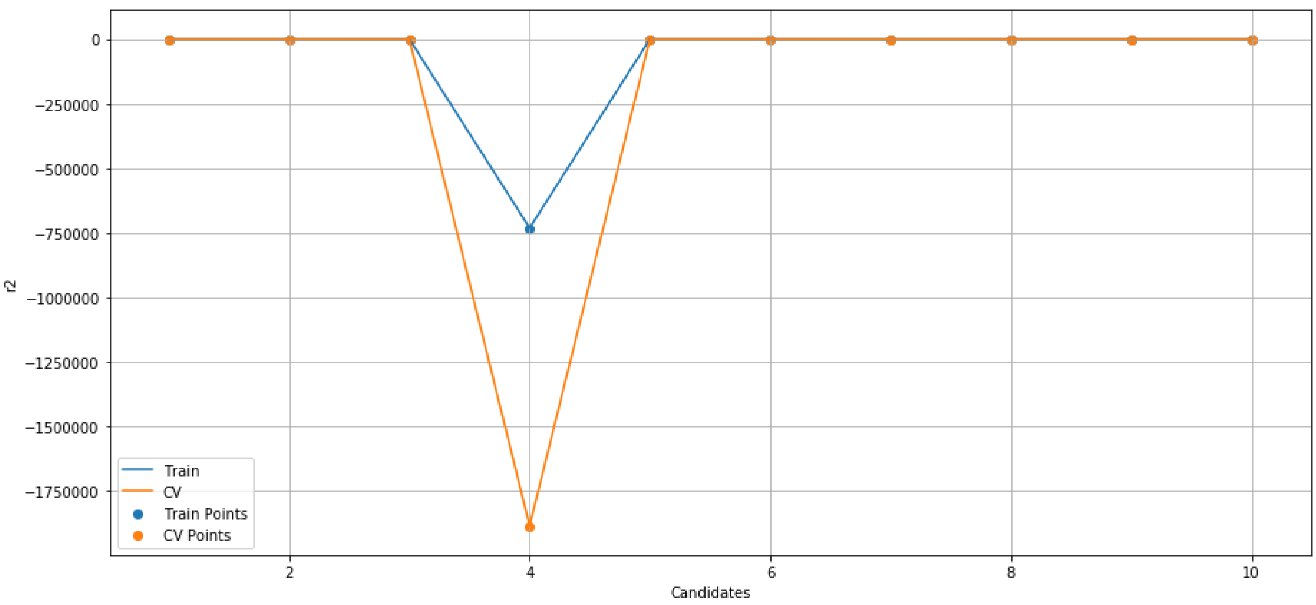
In [152]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6883387036948128 CV:0.5900404044953341
Train:-34.2765281321071 CV:-34.64086151616521
Train:-8.795273224676796 CV:-8.87729079555423
Train:-731453.420549727 CV:-1884084.319833497
Train:-0.6336737967436796 CV:-0.6634062164882415
Train:0.7663291368395259 CV:0.2509120688554729
Train:-56.802373166318475 CV:-57.4264508372318
Train:0.9548680715751479 CV:0.3630279743690944
Train:-2.835766301284559 CV:-2.855254285711628
Train:-34.192407248693726 CV:-34.5556598711098
```

In [153]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.5900404044953341
```

There is no improvement with PCA features.

In [154]:

```
clf.best_estimator_
```

Out[154]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
```

```
             colsample_bynode=1, colsample_bytree=1, gamma=0.01,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=2, max_features=0.95, min_child_weight=1, missing=None,
             n_estimators=500, n_jobs=-1, nthread=None, objective='reg:linear',
             random_state=42, reg_alpha=1, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=0.2, verbosity=1)
```

In [157]:

```
model_xgb_label_pca = XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.01,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=2, max_features=0.95, min_child_weight=1, missing=None,
             n_estimators=500, n_jobs=-1, nthread=None, objective='reg:linear',
             random_state=42, reg_alpha=1, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=0.2, verbosity=1)

model_xgb_label_pca.fit(X_train_le_PCA_corr,Y_train)
```

```
[16:55:03] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[157]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.01,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=2, max_features=0.95, min_child_weight=1, missing=None,
             n_estimators=500, n_jobs=-1, nthread=None, objective='reg:linear',
             random_state=42, reg_alpha=1, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=0.2, verbosity=1)
```

In [ ]:

```
pred_test_xgb_label_pca = model_xgb_label_pca.predict(X_test_le_PCA_corr)
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_xgb_label_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_xgb_label_pca.csv", index=False)
```

Extra Trees Regressor with Label Encoding + Correlation Features:

In [163]:

```
neigh=ExtraTreesRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[150,200,300,350,400,500],
             'max_depth':[2,3,4,5,7,8,10],
             'min_samples_split':[2,3,4,5,6,7,8,10],
             'max_features': [.95],
             'min_samples_leaf': [3,4,5,6,7,8,10],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    5.1s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  2.4min finished
```

Out[163]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0,
                                                 criterion='mse',
                                                 max_depth=None,
                                                 max_features='auto',
                                                 max_leaf_nodes=None,
```

```
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=-1,
                                              oob_score=False,...
                   param_distributions={'max_depth': [2, 3, 4, 5, 7, 8, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1, 10,
                                                                  100],
                                        'min_samples_leaf': [3, 4, 5, 6, 7, 8,
                                                             10],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 10],
                                        'n_estimators': [150, 200, 300, 350,
                                                         400, 500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [164]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

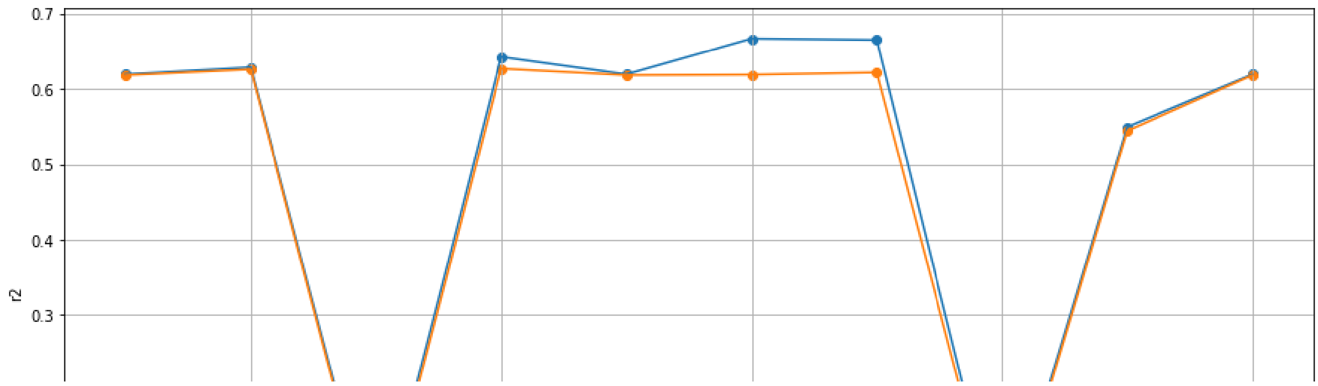
In [165]:

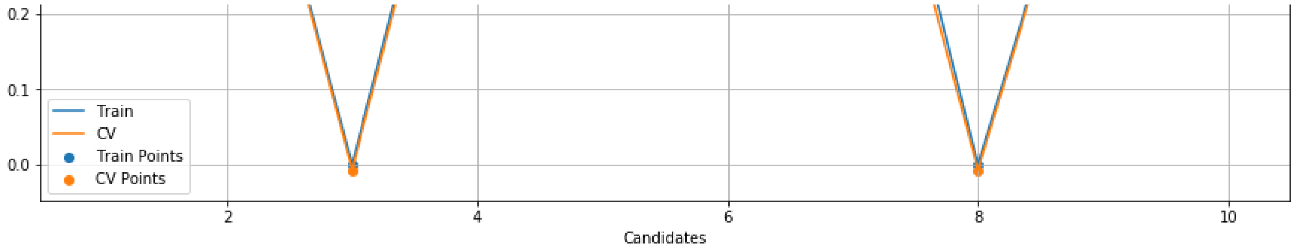
```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6193737474398027 CV:0.6182674853356076
Train:0.6286375626050076 CV:0.6259796730431303
Train:0.0 CV:-0.007406229142669729
Train:0.6424923030961838 CV:0.6269027068738552
Train:0.619398801089396 CV:0.6183506695103598
Train:0.6667121366685109 CV:0.6189514172674329
Train:0.6649969571029601 CV:0.621807099438097
Train:2.2204460492503132e-17 CV:-0.0074062291426711 5
Train:0.549643723053264 CV:0.5444181227924083
Train:0.6193810097345069 CV:0.6182992803803881
```

In [166]:

```
candidates = list(range(1,11))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```






```
The Best Score 0.6269027068738552
```

We see very less tendency to overfit even with extra-tress with most combinations. The train and cv scores are very close. The new features are doing well.

In [167]:

```
clf.best_estimator_
```

Out[167]:

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=8, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.1,
                    min_impurity_split=None, min_samples_leaf=6,
                    min_samples_split=8, min_weight_fraction_leaf=0.0,
                    n_estimators=200, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)
```

In [169]:

```
model_xt_le = ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=8, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.1,
                    min_impurity_split=None, min_samples_leaf=6,
                    min_samples_split=8, min_weight_fraction_leaf=0.0,
                    n_estimators=200, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)
model_xt_le.fit(X_train_le_corr,Y_train)
```

Out[169]:

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=8, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.1,
                    min_impurity_split=None, min_samples_leaf=6,
                    min_samples_split=8, min_weight_fraction_leaf=0.0,
                    n_estimators=200, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)
```

In [170]:

```
pred_test_xt_le = model_xt_le.predict(X_test_le_corr)
```

In [171]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_xt_le]}
data = pd.DataFrame(data)
data.to_csv("submission_xt_label_corr.csv", index=False)
```

Extra Trees Regressor with Label Encoding + Correlation Features + PCA:

In [159]:

```
neigh=ExtraTreesRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[150,200,300,350,400,500],
              'max_depth':[2,3,4,5,7,8,10],
```

```
             'min_samples_split':[2,3,4,5,6,7,8,10],
             'max_features': [.95],
             'min_samples_leaf': [3,4,5,6,7,8,10],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_le_PCA_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    5.4s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   50.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.4min finished
```

Out[159]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0,
                                                 criterion='mse',
                                                 max_depth=None,
                                                 max_features='auto',
                                                 max_leaf_nodes=None,
                                                 max_samples=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=100, n_jobs=-1,
                                                 oob_score=False,...
                   param_distributions={'max_depth': [2, 3, 4, 5, 7, 8, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1, 10,
                                                                  100],
                                        'min_samples_leaf': [3, 4, 5, 6, 7, 8,
                                                             10],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 10],
                                        'n_estimators': [150, 200, 300, 350,
                                                         400, 500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [160]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

In [161]:

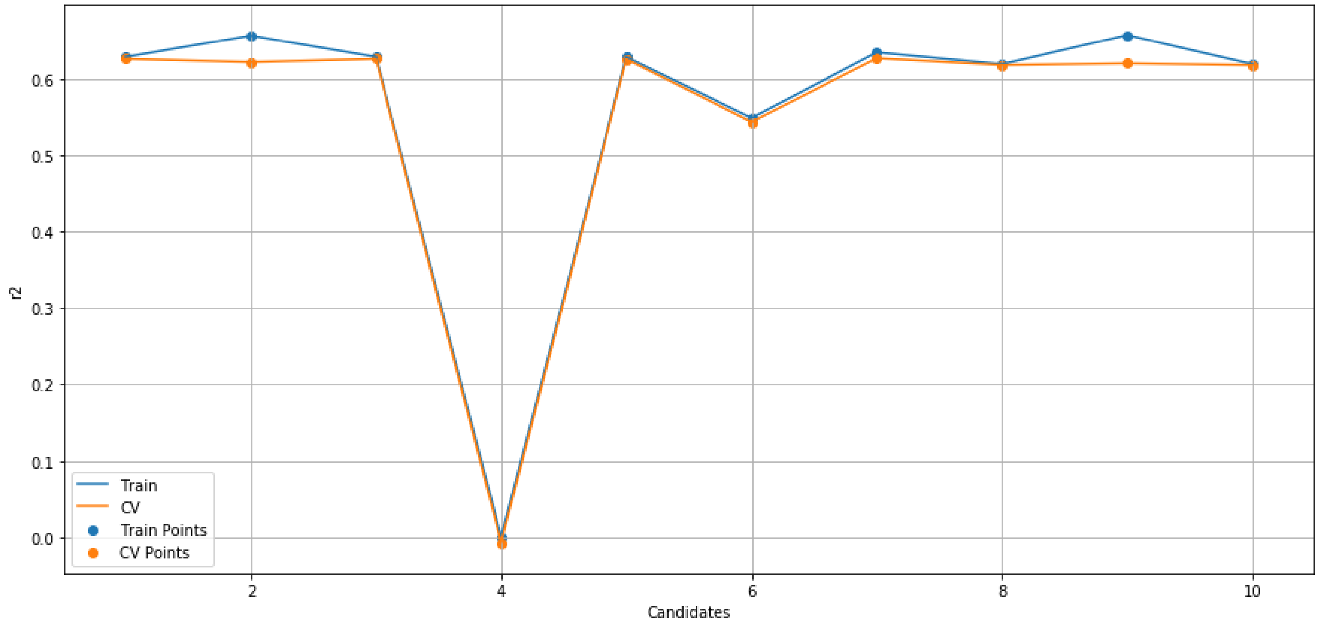
```
for i,j in zip(train_r2,cv_r2):
    print("Train:{} CV:{}".format(i,j))
```

```
Train:0.6288579790587152 CV:0.626248000833604
Train:0.6566965725073606 CV:0.622096567717919
Train:0.6288496968699937 CV:0.6262139379868087
Train:2.2204460492503132e-17 CV:-0.00740622914267175
Train:0.6287107048711112 CV:0.6258923034304187
Train:0.5490853777784092 CV:0.543887969853859
Train:0.6347448881533764 CV:0.6270604403297837
Train:0.6193578745295168 CV:0.6182380134597121
Train:0.6574246864570465 CV:0.6203680149962333
Train:0.6193534068448435 CV:0.6181628164804917
```

In [162]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
```

```
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6270604403297837
```

Here we see a very small improvement with the PCA features along with the Label Encoded Features.

In [163]:

```
clf.best_estimator_
```

Out[163]:

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0,
                    min_impurity_split=None, min_samples_leaf=7,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=150, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)
```

In [164]:

```
model_xt_le_pca = ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0,
                    min_impurity_split=None, min_samples_leaf=7,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=150, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)
model_xt_le_pca.fit(X_train_le_PCA_corr,Y_train)
```

Out[164]:

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0,
                    min_impurity_split=None, min_samples_leaf=7,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=150, n_jobs=-1, oob_score=False,
```

```
                random_state=42, verbose=0, warm_start=False)
```

In [166]:

```
pred_test_xt_le_pca = model_xt_le_pca.predict(X_test_le_PCA_corr)
```

In [167]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_xt_le_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_xt_label_corr_pca.csv", index=False)
```

Extra Trees Regressor with One Hot Encoding(K-Best) + Correlation Features:

In [109]:

```
neigh=ExtraTreesRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[150,200,300,500],
             'max_depth':[2,3,4,5,10],
             'min_samples_split':[2, 5, 10],
             'max_features': [.95],
             'min_samples_leaf': [3,4,5,6,7,10],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   46.6s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.8min finished
```

Out[109]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0,
                                                 criterion='mse',
                                                 max_depth=None,
                                                 max_features='auto',
                                                 max_leaf_nodes=None,
                                                 max_samples=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=100, n_jobs=-1,
                                                 oob_score=False,...
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [2, 3, 4, 5, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1, 10,
                                                                  100],
                                        'min_samples_leaf': [3, 4, 5, 6, 7, 10],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [150, 200, 300, 500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [110]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

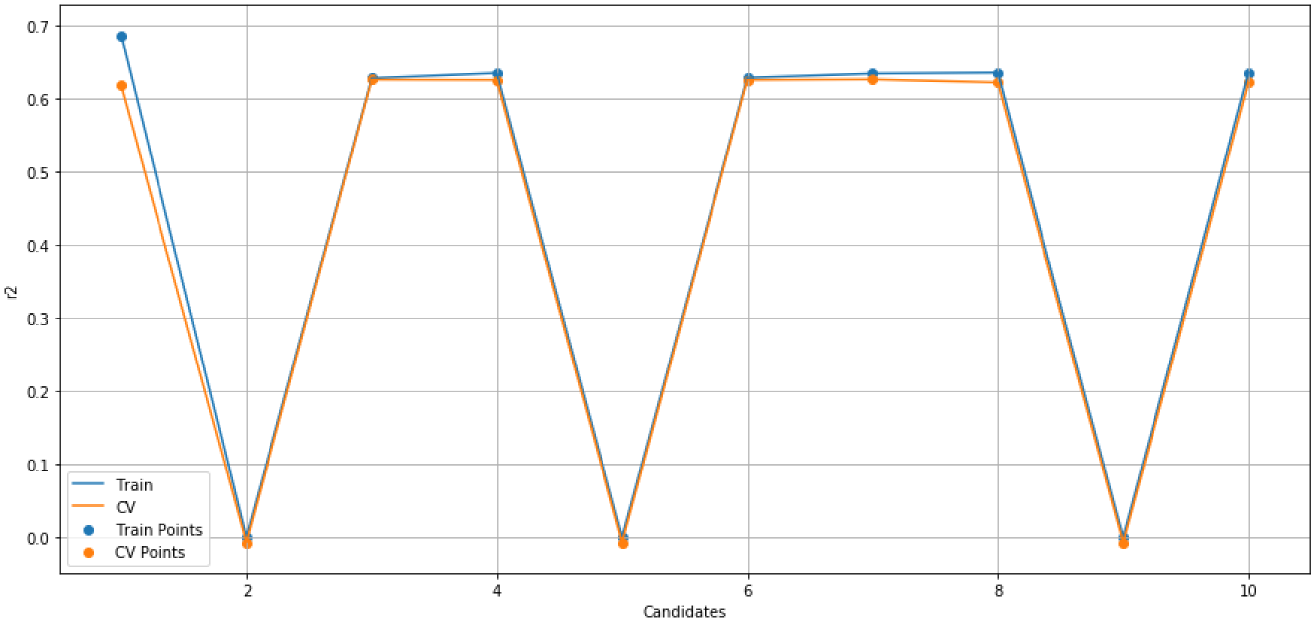
In [111]:

```
for i,j in zip(train_r2,cv_r2):
    print("Train:{}   CV:{}".format(i,j))
```

```
Train:0.6875287329805064  CV:0.6184192214134601
Train:4.440892098500626e-17  CV:-0.007406229142671838
Train:0.6283671536862963  CV:0.6264089374276749
Train:0.6354740267697148  CV:0.6260196312798841
Train:0.0  CV:-0.007406229142669729
Train:0.6289823097705043  CV:0.6260344223050112
Train:0.6347351267576264  CV:0.6267503723353872
Train:0.6359513186844128  CV:0.6224618678601749
Train:4.440892098500626e-17  CV:-0.007406229142668442
Train:0.6359977073244075  CV:0.6224413468639516
```

In [113]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```




```
The Best Score 0.6267503723353872
```

Reducing the One-Hot Encoded Features tend to remove any tendency to overfit.

In [114]:

```
clf.best_estimator_
```

Out[114]:

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.01,
                    min_impurity_split=None, min_samples_leaf=5,
                    min_samples_split=10, min_weight_fraction_leaf=0.0,
                    n_estimators=500, n_jobs=-1, oob_score=False,
```

```
                    random_state=42, verbose=0, warm_start=False)
```

In [123]:

```
model_xt_ohe = ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.01,
                    min_impurity_split=None, min_samples_leaf=5,
                    min_samples_split=10, min_weight_fraction_leaf=0.0,
                    n_estimators=500, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)


model_xt_ohe.fit(X_train_ohe_corr,Y_train)
```

Out[123]:

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.01,
                    min_impurity_split=None, min_samples_leaf=5,
                    min_samples_split=10, min_weight_fraction_leaf=0.0,
                    n_estimators=500, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)
```

In [178]:

```
pred_test_xt_ohe = model_xt_ohe.predict(X_test_ohe_corr)
```

In [179]:

```
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_xt_ohe]}
data = pd.DataFrame(data)
data.to_csv("submission_xt_ohe_corr.csv", index=False)
```

Extra Trees Regressor with One Hot Encoding(K-Best) + Correlation Features + PCA:

In [137]:

```
neigh=ExtraTreesRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[150,200,300,500],
             'max_depth':[2,3,4,5,10],
             'min_samples_split':[2, 5, 10],
             'max_features': [.95],
             'min_samples_leaf': [3,4,5,6,7,10],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]}
clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs=-1,verbos
e=5)#Uisng k-fold cross validation with k=5
clf.fit(X_train_ohe_PCA_corr,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.3s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   28.5s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   44.8s finished
```

Out[137]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0,
                                                 criterion='mse',
                                                 max_depth=None,
                                                 max_features='auto',
                                                 max_leaf_nodes=None,
                                                 max_samples=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
```

```
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=-1,
                                              oob_score=False,...
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [2, 3, 4, 5, 10],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1, 10,
                                                                  100],
                                        'min_samples_leaf': [3, 4, 5, 6, 7, 10],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [150, 200, 300, 500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

In [138]:

```
results=pd.DataFrame.from_dict(clf.cv_results_)
train_r2=results['mean_train_score']
cv_r2=results['mean_test_score']
```

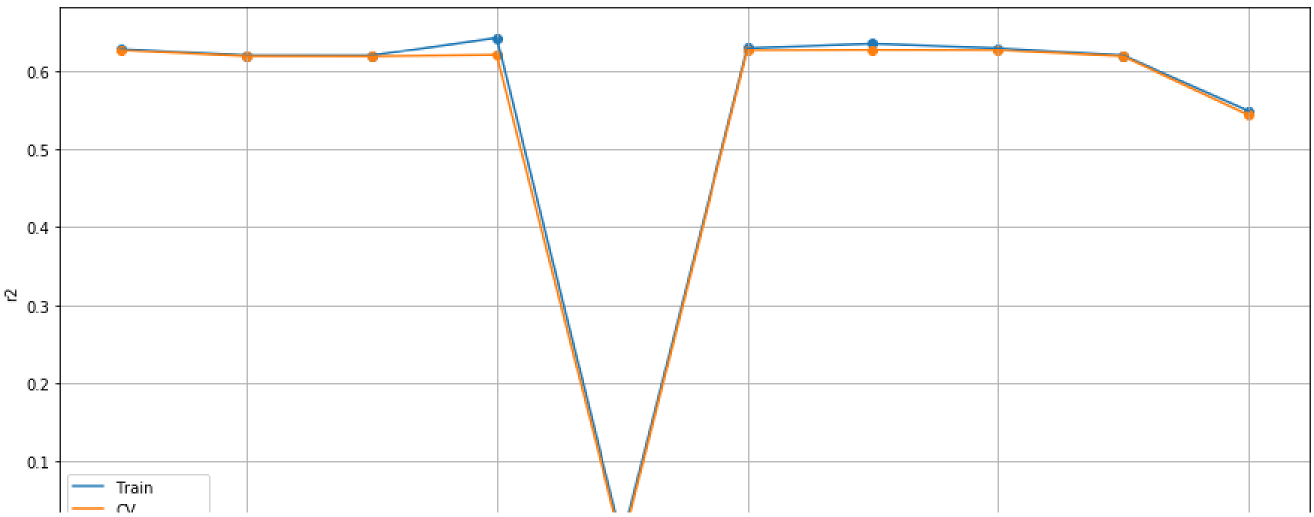
In [139]:

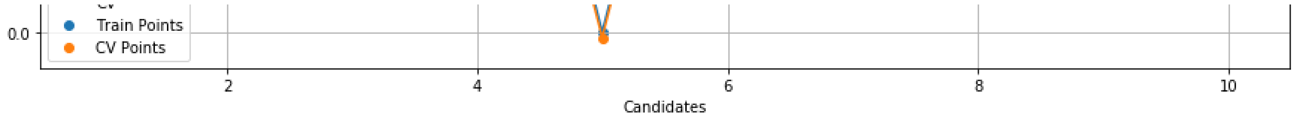
```
for i,j in zip(train_r2,cv_r2):
    print("Train:{}  CV:{}".format(i,j))
```

```
Train:0.6277672255036292  CV:0.6260612657844848
Train:0.6194032884171115  CV:0.6183592713991782
Train:0.6193526908476743  CV:0.6181455897796904
Train:0.642327785807353  CV:0.6199687010772075
Train:4.4408920985006264e-17  CV:-0.007406229142671838
Train:0.6290031585356612  CV:0.6260276290880571
Train:0.6348635009307351  CV:0.626708224229555
Train:0.6289964107094271  CV:0.6263292825199442
Train:0.6193498968495594  CV:0.6182069636151921
Train:0.5485523670330967  CV:0.5433137528756034
```

In [140]:

```
candidates = list(range(1,11))
plt.figure(figsize=(15,7))
plt.plot(candidates,train_r2,label='Train')
plt.plot(candidates,cv_r2,label='CV')
plt.scatter(candidates,train_r2,label='Train Points')
plt.scatter(candidates,cv_r2,label='CV Points')
plt.legend()
plt.xlabel("Candidates")
plt.ylabel("r2")
plt.grid()
plt.show()
print("The Best Score",clf.best_score_)
```






```
The Best Score 0.626708224229555
```

The score dropped by a small margin with the usage of PCA components

In [142]:

```
clf.best_estimator_
```

Out[142]:

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.001,
                    min_impurity_split=None, min_samples_leaf=5,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=150, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)
```

In [143]:

```
model_xt_ohe_pca = ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.001,
                    min_impurity_split=None, min_samples_leaf=5,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=150, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)


model_xt_ohe_pca.fit(X_train_ohe_PCA_corr,Y_train)
```

Out[143]:

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=4, max_features=0.95, max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.001,
                    min_impurity_split=None, min_samples_leaf=5,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=150, n_jobs=-1, oob_score=False,
                    random_state=42, verbose=0, warm_start=False)
```

In [144]:

```
pred_test_xt_ohe_pca = model_xt_ohe_pca.predict(X_test_ohe_PCA_corr)
data={'ID':[i for i in ID],
      'y':[j for j in pred_test_xt_ohe_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_xt_ohe_corr_pca.csv", index=False)
```

Stacking Classifier with Label Encoding + Correlation Features:

Here I have stacked 3 models. The pretuned Random Forest, XgBoost and Extra Trees on both Label Encoded and One-Hot Encoded + Correlation Features. For the final meta regressor I have used Ridge Regressor with regularization set to 0 so that it doesn't impact on the stacked models.

In [180]:

```
ridge= Ridge(random_state=42,fit_intercept= False,alpha=0)
stack = StackingCVRegressor(regressors=(model_rf_le, model_xgb_label, model_xt_le),
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=5)

cv_score=cross_val_score(stack,X_train_le_corr,Y_train,scoring='r2',cv= 5,verbose=5,n_jobs=-1)
```

```
print('Mean Score:',cv_score.mean())
print('Standard Deviation:',cv_score.std())
stack.fit(X_train_le_corr,Y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  1.2min remaining:  1.8min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  1.2min finished
```

```
Mean Score: 0.6022386878976362
Standard Deviation: 0.02323366721254506
[22:09:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:09:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:09:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:09:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:09:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:09:50] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[180]:

```
StackingCVRegressor(cv=5,
                    meta_regressor=Ridge(alpha=0, copy_X=True,
                                         fit_intercept=False, max_iter=None,
                                         normalize=False, random_state=42,
                                         solver='auto', tol=0.001),
                    n_jobs=None, pre_dispatch='2*n_jobs', random_state=None,
                    refit=True,
                    regressors=(RandomForestRegressor(bootstrap=True,
                                                      ccp_alpha=0.0,
                                                      criterion='mse',
                                                      max_depth=5,
                                                      max_features=0.95,
                                                      max_leaf_nodes=None...
                                                    max_depth=8,
                                                    max_features=0.95,
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.1,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=6,
                                                    min_samples_split=8,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=200, n_jobs=-1,
                                                    oob_score=False,
                                                    random_state=42, verbose=0,
                                                    warm_start=False)),
                    shuffle=True, store_train_meta_features=False,
                    use_features_in_secondary=False, verbose=0)
```

In [181]:

```
y_pred_stack_label = stack.predict(X_test_le_corr)
data={'ID':[i for i in ID],
      'y':[j for j in y_pred_stack_label]}
data = pd.DataFrame(data)
data.to_csv("submission_xgb_rf_stack_ridge_label.csv", index=False)
```

Stacking Classifier with Label Encoding + Correlation Features + PCA:

In [168]:

```
ridge= Ridge(random_state=42,fit_intercept= False,alpha=0)
stack = StackingCVRegressor(regressors=(model_rf_le_pca_corr, model_xgb_label_pca, model_xt_le_pca)
,
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=5)

cv_score=cross_val_score(stack,X_train_le_PCA_corr,Y_train,scoring='r2',cv= 5,verbose=5,n_jobs=-1)
```

```
cv_score=cross_val_score(stack,X_train_le_PCA_corr,Y_train,scoring='r2',cv= 5,verbose=5,n_jobs=-1)
print('Mean Score:',cv_score.mean())
print('Standard Deviation:',cv_score.std())
stack.fit(X_train_le_PCA_corr,Y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  1.0min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  1.1min finished
```

```
Mean Score: 0.6144316224118507
Standard Deviation: 0.022523490703981894
[17:19:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:19:59] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:19:59] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:20:00] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:20:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:20:08] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[168]:

```
StackingCVRegressor(cv=5,
                    meta_regressor=Ridge(alpha=0, copy_X=True,
                                         fit_intercept=False, max_iter=None,
                                         normalize=False, random_state=42,
                                         solver='auto', tol=0.001),
                    n_jobs=None, pre_dispatch='2*n_jobs', random_state=None,
                    refit=True,
                    regressors=(RandomForestRegressor(bootstrap=True,
                                                      ccp_alpha=0.0,
                                                      criterion='mse',
                                                      max_depth=3,
                                                      max_features=0.95,
                                                      max_leaf_nodes=None...
                                                    max_depth=4,
                                                    max_features=0.95,
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=7,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=150, n_jobs=-1,
                                                    oob_score=False,
                                                    random_state=42, verbose=0,
                                                    warm_start=False)),
                    shuffle=True, store_train_meta_features=False,
                    use_features_in_secondary=False, verbose=0)
```

In [169]:

```
y_pred_stack_label_pca = stack.predict(X_test_le_PCA_corr)
data={'ID':[i for i in ID],
      'y':[j for j in y_pred_stack_label_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_xgb_rf_stack_ridge_label_pca.csv", index=False)
```

Stacking Classifier with One Hot Encoding(K-Best) + Correlation Features:

In [124]:

```
ridge= Ridge(random_state=42,fit_intercept= False,alpha=0)
stack = StackingCVRegressor(regressors=(model_rf_ohe, model_xgb_ohe, model_xt_ohe),
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=5)

cv_score=cross_val_score(stack,X_train_ohe_corr,Y_train,scoring='r2',cv= 5,verbose=5,n_jobs=-1)
```

```
cv_score=cross_val_score(stack,X_train_ohe_corr,Y_train,scoring='r2',cv= 5,verbose=5,n_jobs=-1)
print('Mean Score:',cv_score.mean())
print('Standard Deviation:',cv_score.std())
stack.fit(X_train_ohe_corr,Y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  1.1min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  1.1min finished
```

```
Mean Score: 0.6162820932360659
Standard Deviation: 0.02429116741974889
[16:15:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:15:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:15:29] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:15:30] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:15:31] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:15:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[124]:

```
StackingCVRegressor(cv=5,
                    meta_regressor=Ridge(alpha=0, copy_X=True,
                                         fit_intercept=False, max_iter=None,
                                         normalize=False, random_state=42,
                                         solver='auto', tol=0.001),
                    n_jobs=None, pre_dispatch='2*n_jobs', random_state=None,
                    refit=True,
                    regressors=(RandomForestRegressor(bootstrap=True,
                                                      ccp_alpha=0.0,
                                                      criterion='mse',
                                                      max_depth=5,
                                                      max_features=0.95,
                                                      max_leaf_nodes=None...
                                                    max_features=0.95,
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.01,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=5,
                                                    min_samples_split=10,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=500, n_jobs=-1,
                                                    oob_score=False,
                                                    random_state=42, verbose=0,
                                                    warm_start=False)),
                    shuffle=True, store_train_meta_features=False,
                    use_features_in_secondary=False, verbose=0)
```

In [125]:

```
y_pred_stack_ohe = stack.predict(X_test_ohe_corr)
data={'ID':[i for i in ID],
      'y':[j for j in y_pred_stack_ohe]}
data = pd.DataFrame(data)
data.to_csv("submission_xgb_rf_stack_ridge_ohe.csv", index=False)
```

Stacking Classifier with One Hot Encoding(K-Best) + Correlation Features + PCA:

In [147]:

```
ridge= Ridge(random_state=42,fit_intercept= False,alpha=0)
stack = StackingCVRegressor(regressors=(model_rf_ohe_pca, model_xgb_ohe_pca, model_xt_ohe_pca),
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=5)

cv_score=cross_val_score(stack,X_train_ohe_PCA_corr,Y_train,scoring='r2',cv= 5,verbose=5,n_jobs=-1)
print('Mean Score:',cv_score.mean())
```

```
print('Mean Score:',cv_score.mean())
print('Standard Deviation:',cv_score.std())
stack.fit(X_train_ohe_PCA_corr,Y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:   38.8s remaining:   58.3s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   39.0s finished
```

```
Mean Score: 0.6148074134948918
Standard Deviation: 0.026054456819448856
[16:39:09] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:39:10] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:39:11] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:39:11] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:39:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[16:39:18] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[147]:

```
StackingCVRegressor(cv=5,
                    meta_regressor=Ridge(alpha=0, copy_X=True,
                                         fit_intercept=False, max_iter=None,
                                         normalize=False, random_state=42,
                                         solver='auto', tol=0.001),
                    n_jobs=None, pre_dispatch='2*n_jobs', random_state=None,
                    refit=True,
                    regressors=(RandomForestRegressor(bootstrap=True,
                                                      ccp_alpha=0.0,
                                                      criterion='mse',
                                                      max_depth=5,
                                                      max_features=0.95,
                                                      max_leaf_nodes=None...
                                                    max_features=0.95,
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.001,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=5,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=150, n_jobs=-1,
                                                    oob_score=False,
                                                    random_state=42, verbose=0,
                                                    warm_start=False)),
                    shuffle=True, store_train_meta_features=False,
                    use_features_in_secondary=False, verbose=0)
```

In [148]:

```
y_pred_stack_ohe_pca = stack.predict(X_test_ohe_PCA_corr)
data={'ID':[i for i in ID],
      'y':[j for j in y_pred_stack_ohe_pca]}
data = pd.DataFrame(data)
data.to_csv("submission_xgb_rf_stack_ridge_ohe_pca.csv", index=False)
```

6. Conclusion

In [73]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Categorical_encoding",'CV', "Private Score","Public Score"]
x.add_row(["Linear Regression",'One Hot(K-Best)',0.60565, 0.52534,0.53055])
x.add_row(["Linear Regression",'One Hot(Full)',0.53804, 0.51365,0.52612])
x.add_row(["Linear Regression",'Label',0.5869, 0.50863,0.51959])
x.add_row(["Elastic Net Regressor",'One Hot(K-Best)',0.62148, 0.53420,0.53287])
```

```
x.add_row(["Elastic Net Regressor",'Label',0.60624, 0.53322,0.53762])
x.add_row(["Elastic Net Regressor + PCA components",'One Hot(K-Best)',0.62092, 0.53971,0.54083])
x.add_row(["Elastic Net Regressor + PCA components",'Label',0.60514, 0.53216,0.53670])
x.add_row(["Random Forest",'One Hot(K-Best)',0.62899, 0.55083,0.55887])
x.add_row(["Random Forest + PCA components",'One Hot(K-Best)',0.62729, 0.54981,0.55986])
x.add_row(["Random Forest",'Label',0.62350, 0.54222,0.55200])
x.add_row(["Random Forest + PCA components",'Label',0.62619, 0.55078,0.55793])
x.add_row(["Random Forest + corr features",'One Hot(K-Best)',0.62981, 0.55188,0.55898])
x.add_row(["Random Forest + corr features + PCA components",'One Hot(K-Best)',0.62763, 0.55183,0.5
6012])
x.add_row(["Random Forest + corr features",'Label',0.62721, 0.55197,0.55788])
x.add_row(["Random Forest + corr features + PCA components",'Label', 0.62453, 0.54863,0.55551])
x.add_row(["XgBoost + corr features",'One Hot(K-Best)',0.62406, 0.53783,0.54428])
x.add_row(["XgBoost + corr features + PCA",'One Hot(K-Best)',0.62496, 0.54287,0.55029])
x.add_row(["XgBoost + corr features",'Label',0.606190, 0.54426,0.54681])
x.add_row(["XgBoost + corr features + PCA",'Label',0.59004, 0.53922,0.54287])
x.add_row(["Extra Trees + corr features",'Label',0.626902, .55058,.55296])
x.add_row(["Extra Trees + corr features + PCA",'Label',0.626902, 0.54816,0.55065])
x.add_row(['Extra Trees + corr features','One Hot(K-Best)',0.62675,0.55005,0.55270])
x.add_row(['Extra Trees + corr features + PCA','One Hot(K-Best)',0.62670,0.54768,0.55030])
x.add_row(['Stack + corr features','Label',0.602238,0.55316, 0.55803])
x.add_row(['Stack + corr features + PCA','Label',0.61443,0.54963, 0.55437])
x.add_row(['Stack + corr features','One Hot(K-Best)',0.616546,0.55112,0.55682])
x.add_row(['Stack + corr features + PCA','One Hot(K-Best)',0.61480,0.54935,0.55341])
print(x)
```

```
+-------------------------------------------------+---------------------+----------+---------------
------------+
|                      Model                      | Categorical_encoding |    CV    | Private Score
Public Score |
+-------------------------------------------------+---------------------+----------+---------------
------------+
|                Linear Regression                |   One Hot(K-Best)   | 0.60565  |    0.52534
0.53055    |
|                Linear Regression                |    One Hot(Full)    | 0.53804  |    0.51365
0.52612    |
|                Linear Regression                |        Label        |  0.5869  |    0.50863
0.51959    |
|              Elastic Net Regressor              |   One Hot(K-Best)   | 0.62148  |     0.5342
0.53287    |
|              Elastic Net Regressor              |        Label        | 0.60624  |    0.53322
0.53762    |
|      Elastic Net Regressor + PCA components     |   One Hot(K-Best)   | 0.62092  |    0.53971
|   0.54083    |
|      Elastic Net Regressor + PCA components     |        Label        | 0.60514  |    0.53216
0.5367     |
|                  Random Forest                  |   One Hot(K-Best)   | 0.62899  |    0.55083
0.55887    |
|          Random Forest + PCA components         |   One Hot(K-Best)   | 0.62729  |    0.54981
|   0.55986    |
|                  Random Forest                  |        Label        |  0.6235  |    0.54222
0.552      |
|          Random Forest + PCA components         |        Label        | 0.62619  |    0.55078
0.55793    |
|          Random Forest + corr features          |   One Hot(K-Best)   | 0.62981  |    0.55188
|   0.55898    |
| Random Forest + corr features + PCA components |   One Hot(K-Best)   | 0.62763  |    0.55183
|   0.56012    |
|          Random Forest + corr features          |        Label        | 0.62721  |    0.55197
0.55788    |
| Random Forest + corr features + PCA components |        Label        | 0.62453  |    0.54863
|   0.55551    |
|             XgBoost + corr features             |   One Hot(K-Best)   | 0.62406  |    0.53783
0.54428    |
|          XgBoost + corr features + PCA          |   One Hot(K-Best)   | 0.62496  |    0.54287
|   0.55029    |
|             XgBoost + corr features             |        Label        | 0.60619  |    0.54426
0.54681    |
|          XgBoost + corr features + PCA          |        Label        | 0.59004  |    0.53922
0.54287    |
|           Extra Trees + corr features           |        Label        | 0.626902 |    0.55058
0.55296    |
|        Extra Trees + corr features + PCA        |        Label        | 0.626902 |    0.54816
0.55065    |
|           Extra Trees + corr features           |   One Hot(K-Best)   | 0.62675  |    0.55005
0.5527     |
|        Extra Trees + corr features + PCA        |   One Hot(K-Best)   |  0.6267  |    0.54768
```

Extra Trees + corr features + PCA	One Hot(K-Best)	0.6267	0.54768	0.5503
Stack + corr features	Label	0.602238	0.55316	0.55803
Stack + corr features + PCA	Label	0.61443	0.54963	0.55437
Stack + corr features	One Hot(K-Best)	0.616546	0.55112	0.55682
Stack + corr features + PCA	One Hot(K-Best)	0.6148	0.54935	0.55341

1. The Best Performance was provided by Random Forest with One Hot Encoded Features and correlation features with a CV score of 0.62981.
2. But according to the Kaggle private leaderboard the stacked model with label encoded features and correlation features gave the best score which is 0.55316.
3. There might be some slight overfitting which can be overcomed by much more Hyper-parameter Tuning.
4. The Features created based on the correlation proved to have increased the performance of the models.
5. For some reason the label encoded feature in the Kaggle Leaderboard perfromed better than the One Hot encoded feature with the stacking model. As, in some other cases the One Hot performed better in CV score. This may be due to the fact that the correlation features had more impact on the models than the label encoded or one-hot encoded features.
6. The PCA components performed a bit better in the final model with both Label and One Hot Encoded features based on the CV scores. However, the Kaggle score was less than the model without PCA components.
7. Reducing the dimentions for One-Hot Encoded features using selectkbest seems to improve the model performance and reduce overfitting which is clear from the Linear Regression Baseline Model.