Project 1 & 2 - Regression

Machine Learning Spring 2021

UCI Bike Accidents Dataset

Data set Overview:

This UCI dataset describes how various factors like season, weather, weekday or holiday, temperature, climate, wind speed, humidity, etc have affected the total number of accidents in the United States. Here Target variable is the total number of accidents. The dataset has 17 variables and 17379 registered records.

Dataset Description and Link

Variables

1. **instant** : Unique id
2. **dteday** : day
3. **season** : Winter, Summer, Spring, Fall
4. **yr** : 2011 or 2012
5. **mnth** : 1-12
6. **hr** : 0-23
7. **holiday** : 1, if holiday, otherwise 0.
8. **weekday** : 0-6
9. **workingday** : 1 if neither holiday nor weekend, else 0
10. **weathersit** : Rain, Storm, Sunny, Cloudy, Snow, Thunderstorm, Fog, Mist
11. **temp** : Degree Celsius
12. **atemp** : Feels like Temperature in Degree Celsius
13. **hum** : humidity index
14. **windspeed** : wind speed
15. **casual** : no. of casual riders
16. **registered** : no. of registered riders
17. **cnt** : registered + casual riders

UCI DataSet: https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset (https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset)

Step 1 : Data Initializations

In [12]:

```
#python packages
import pandas as pds
import numpy as npy
import seaborn as sb
```

In [13]:

```
#import python packages for visualization/Graph
import matplotlib.pyplot as plot
%matplotlib inline
```

In [14]:

```
##import excel file
dataframe_bike = pds.read_csv("hour_kaggle.csv")
```

In [15]:

```
##labelizing with right term
dataframe_bike = dataframe_bike.rename(columns = {'instant':'id','yr': 'year','weathers
dataframe_bike['season'] = dataframe_bike['season'].map({1:'Fall',2:'Summer',3:'Spring'
dataframe_bike['year'] = dataframe_bike['year'].map({1:'2012',0:'2011'})
dataframe_bike['holiday'] = dataframe_bike['holiday'].map({1:'Yes', 0:'No'})
dataframe_bike['day_in_week'] = dataframe_bike['day_in_week'].map({0:'Monday', 1:'Tuesd
dataframe_bike['is_working_day'] = dataframe_bike['is_working_day'].map({1:'Yes',0:'No'
dataframe_bike['weather_type'] = dataframe_bike['weather_type'].map({1:'Storm', 2:'Rair
```

In [16]:

```
#dataframe_bike.shape
dataframe_bike.head()
```

Out[16]:

	id	dteday	season	year	month	hr	holiday	day_in_week	is_working_day	weather_type
0	1	01-01-2011	Fall	2011	1	0	No	Sunday	No	Storm
1	2	01-01-2011	Fall	2011	1	1	No	Sunday	No	Storm
2	3	01-01-2011	Fall	2011	1	2	No	Sunday	No	Storm
3	4	01-01-2011	Fall	2011	1	3	No	Sunday	No	Storm
4	5	01-01-2011	Fall	2011	1	4	No	Sunday	No	Storm

Step 2 : Adding Missing Values

In [17]:

```
##add 7% missing values of total
import random
miss_number_of_accidents = round(len(dataframe_bike['id']) * 0.07)
#temperature_celcius
miss_temperature_index = pds.Series(random.sample(range(1,len(dataframe_bike['id'])),mi
for value in miss_temperature_index:
    dataframe_bike.at[value,'temperature_celcius'] = npy.nan
#humidity_normalized
miss_humidty_index = pds.Series(random.sample(range(1,len(dataframe_bike['id'])),miss_r
for value in miss_humidty_index:
    dataframe_bike.at[value,'humidity_normalized'] = npy.nan
#windspeed
miss_windspeed_index = pds.Series(random.sample(range(1,len(dataframe_bike['id'])),miss
for value in miss_windspeed_index:
    dataframe_bike.at[value,'windspeed'] = npy.nan
```

In [18]:

```
print('missing values : ' + str(miss_number_of_accidents), dataframe_bike.isna().sum()
```

```
missing values : 1217

id                        0
dteday                    0
season                    0
year                      0
month                     0
hr                        0
holiday                   0
day_in_week               0
is_working_day            0
weather_type              0
temperature_celcius    1217
atemp                     0
humidity_normalized    1217
windspeed              1217
casual                    0
registered                0
number_of_accidents       0
dtype: int64
```

Step 3: Data Preprocessing

In [19]:

```
# select random smaller sample
sample_for_visualization = dataframe_bike.sample(frac=.1, random_state=5)
sample_for_visualization = sample_for_visualization.drop(columns=['id','dteday'])
sample_for_visualization = sample_for_visualization.drop(columns=['year','hr','month']
sample_for_visualization = sample_for_visualization.drop(columns=['holiday','day_in_wee
sample_for_visualization = sample_for_visualization.drop(columns=['atemp'])
sample_for_visualization = sample_for_visualization.drop(columns=['casual','registered'
```

In [20]:

```
sample_for_visualization.head()                  #column head
#sample_for_visualization.info()                 # column details
```

Out[20]:

	season	is_working_day	weather_type	temperature_celcius	humidity_normalized	winds
4707	Spring	Yes	Storm	0.70	0.84	0
13755	Spring	Yes	Rain	0.66	0.78	0
10794	Summer	Yes	Storm	0.38	0.66	0
7665	Winter	Yes	Rain	0.46	0.94	0
9751	Fall	Yes	Rain	0.30	0.70	0

In [21]:

```
sample_for_visualization.describe()          # statistics
```

Out[21]:

	temperature_celcius	humidity_normalized	windspeed	number_of_accidents
count	1634.000000	1619.000000	1636.000000	1738.000000
mean	0.498556	0.628011	0.188698	189.784810
std	0.188697	0.191657	0.121486	185.894058
min	0.020000	0.000000	0.000000	1.000000
25%	0.340000	0.480000	0.104500	37.250000
50%	0.520000	0.630000	0.194000	139.000000
75%	0.660000	0.790000	0.253700	279.000000
max	1.000000	1.000000	0.686600	905.000000

Step 4: Replacing Null Values

In [22]:

```
transformer = lambda x: x.fillna(x.mean())
sample_for_visualization['temperature_celcius'] = sample_for_visualization.groupby('sea
sample_for_visualization['humidity_normalized'] = sample_for_visualization.groupby('sea
sample_for_visualization['windspeed'] = sample_for_visualization.groupby('season')['wi
```

In [23]:

```
sample_for_visualization.head()                  # data peak
#sample_for_visualization.describe()              # statistics
#sample_for_visualization.shape                   # column row number_of_accidents
#sample_for_visualization.info()                  # column details
```

Out[23]:

	season	is_working_day	weather_type	temperature_celcius	humidity_normalized	winds
4707	Spring	Yes	Storm	0.70	0.84	0
13755	Spring	Yes	Rain	0.66	0.78	0
10794	Summer	Yes	Storm	0.38	0.66	0
7665	Winter	Yes	Rain	0.46	0.94	0
9751	Fall	Yes	Rain	0.30	0.70	0

Step 5: Visualization

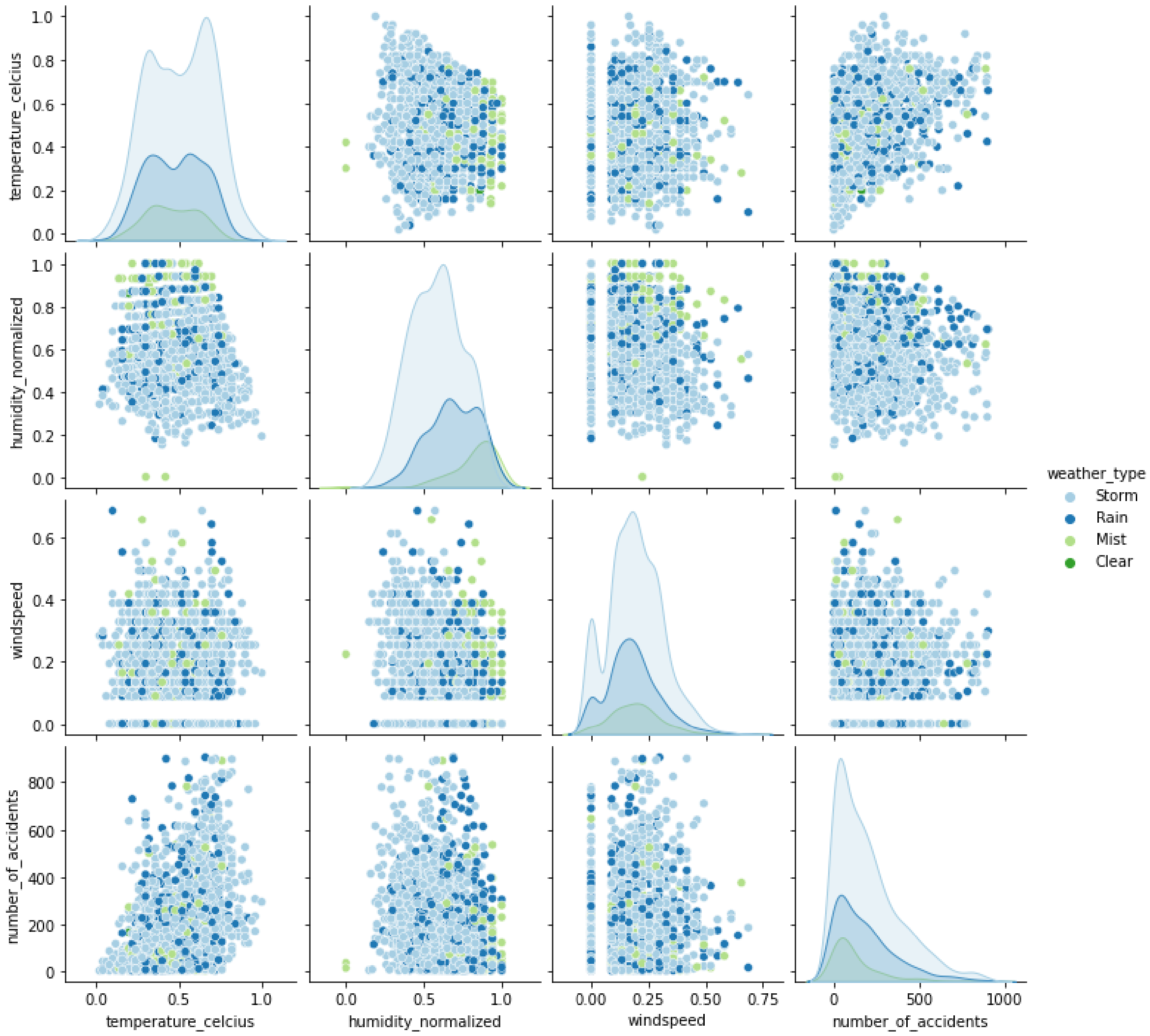
In [24]:

```
# Analyzing correlation between number_of_accidents and season
sb.pairplot(sample_for_visualization,hue='weather_type',palette='Paired')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:305: Use
rWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:305: Use
rWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:305: Use
rWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:305: Use
rWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
```

Out[24]:

```
<seaborn.axisgrid.PairGrid at 0xa0cf25e130>
```

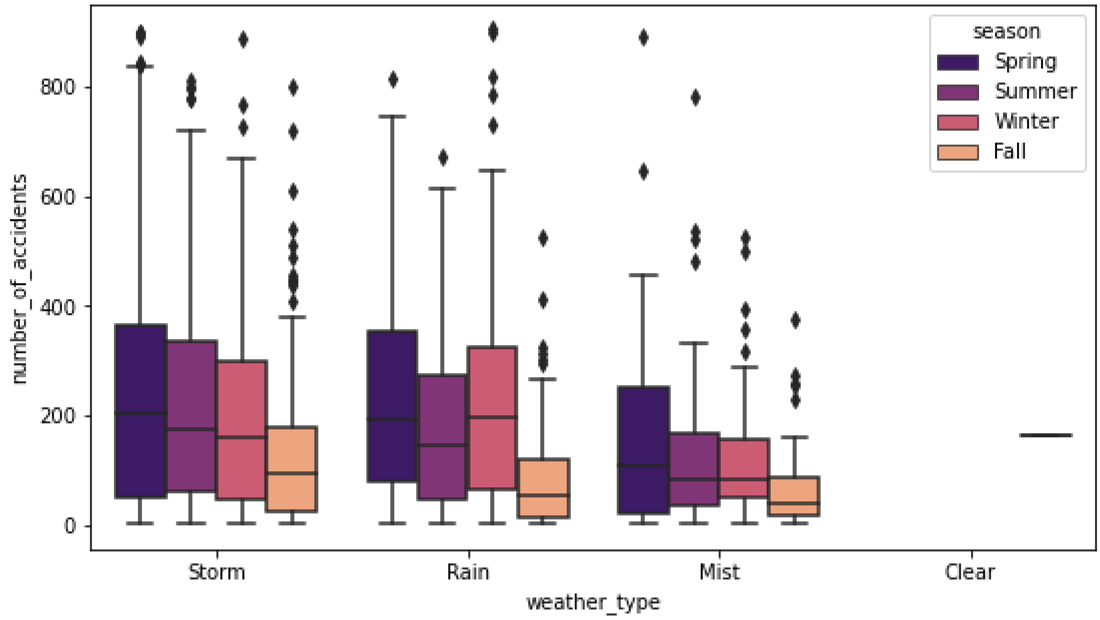


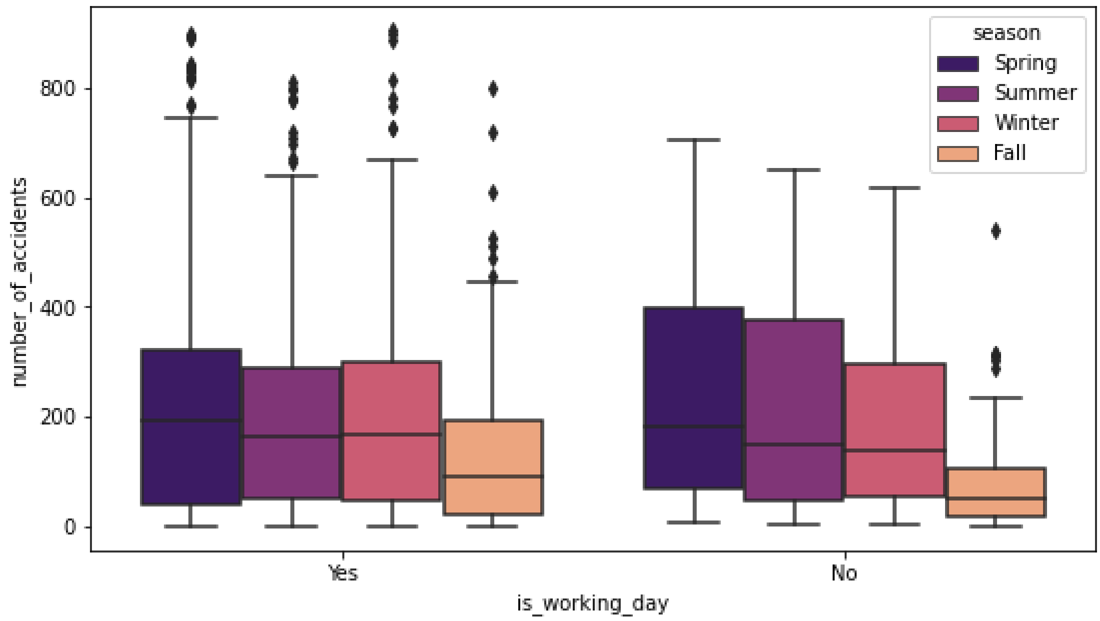
In [25]:

```
# Categorical data distribution for seasons and working day as per weather_type conditi
_, axis = plot.subplots(2,1,figsize=(9,11))
sb.boxplot(x='weather_type', y='number_of_accidents', hue='season',data=sample_for_visu
sb.boxplot(x='is_working_day', y='number_of_accidents', hue='season',data=sample_for_vi
```

Out[25]:

```
<AxesSubplot:xlabel='is_working_day', ylabel='number_of_accidents'>
```





Step 6: Handeling Ordinal Data

In [26]:

```
# convert season to dummy variables
sample_for_visualization = pds.get_dummies(sample_for_visualization, columns=['season']
# convert working day yes, no to 1, 0 respectively
sample_for_visualization['is_working_day'] = sample_for_visualization['is_working_day']
# convert weather_type to orderinal numbers as per severity
sample_for_visualization['weather_type'] = sample_for_visualization['weather_type'].map
# push number_of_accidents to end
sample_for_visualization = sample_for_visualization[[column for column in sample_for_vi
```

In [27]:

```
#sample_for_visualization.shape                # column row number_of_accidents
#sample_for_visualization.describe()           # statistic details
# sample_for_visualization.info()               # column list
sample_for_visualization.head()
```

Out[27]:

	is_working_day	weather_type	temperature_celcius	humidity_normalized	windspeed	sea
4707	1	4	0.70	0.84	0.0000	
13755	1	3	0.66	0.78	0.1940	
10794	1	4	0.38	0.66	0.1343	
7665	1	3	0.46	0.94	0.0000	
9751	1	3	0.30	0.70	0.0896	

Step 7: Exporting Sample Dataset

In [28]:

```
sample_for_visualization.to_csv('input_sample_for_visualization.csv')
```

In [29]:

```
# import sample set
sample_for_visualization = pds.read_csv("input_sample_for_visualization.csv")
sample_for_visualization = sample_for_visualization.drop(sample_for_visualization.colur
```

Regression Task

In [30]:

```
# import regression packages

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression

from sklearn.neighbors import KNeighborsRegressor
from math import sqrt

from sklearn.svm import LinearSVR
from sklearn.svm import SVR

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_squared_error, r2_score

```

In [31]:

```
# spliting training and testing data sets
trainset , testset = train_test_split(sample_for_visualization, test_size = 0.299)

x_trainset = trainset.drop('number_of_accidents', axis=1)
y_trainset = trainset['number_of_accidents']

x_testset = testset.drop('number_of_accidents', axis=1)
y_testset = testset['number_of_accidents']


X_sample = sample_for_visualization.drop('number_of_accidents', axis=1)
Y_sample = sample_for_visualization['number_of_accidents']
```

In [32]:

```
# scaling features
minMaxScalar = MinMaxScaler(feature_range=(0, 1))
# trainset and testset scaling
x_trainset_scaled = minMaxScalar.fit_transform(x_trainset)
x_trainset = pds.DataFrame(x_trainset_scaled)
x_testset_scaled = minMaxScalar.transform(x_testset)
x_testset = pds.DataFrame(x_testset_scaled)
```

In [33]:

```
# creating arrays to store results
model_results = {'regression_model':[],'regression_cvs':[]}
```

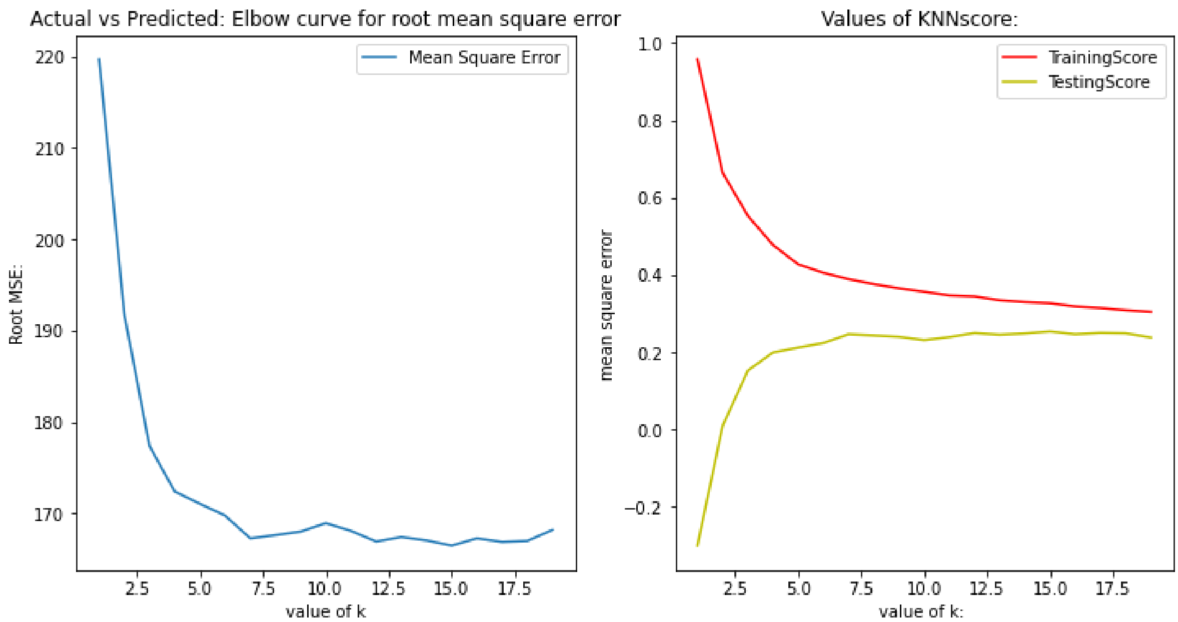
Regression Model 1: KNN Regressor

In [34]:

```
error_value = []
trainset_score_array = []
testset_score_array = []
for j in range(1,20):
    knn_regressor = KNeighborsRegressor(j)
    knn_regressor.fit(x_trainset, y_trainset)

    prediction=knn_regressor.predict(x_testset)
    error = sqrt(mean_squared_error(y_testset,prediction))
    error_value.append(error)

    trainset_score_array.append(knn_regressor.score(x_trainset, y_trainset))
    testset_score_array.append(knn_regressor.score(x_testset, y_testset))
```

In [35]:

```
# Visualization for KNN
_, _ = plot.subplots(1,2,figsize=(12,6))
x_axis = range(1,20)
for plot_position in range(1, 3):
    plot.subplot(1, 2, plot_position)
    if plot_position == 1:
        plot.plot(x_axis,error_value, label='Mean Square Error')
        plot.legend()
        plot.xlabel('value of k')
        plot.ylabel('Root MSE:')
        plot.title('Actual vs Predicted: Elbow curve for root mean square error')
    if plot_position == 2:
        plot.plot(x_axis, trainset_score_array, c = 'r', label = 'TrainingScore')
        plot.plot(x_axis, testset_score_array, c = 'y', label = 'TestingScore')
        plot.legend()
        plot.xlabel('value of k: ')
        plot.ylabel('mean square error')
        plot.title('Values of KNNscore: ')
```



1.1: GridSearchCV for KNN Regressor

In [36]:

```
regressor_parameters = {'n_neighbors':range(2,11)}
knn_regressor = KNeighborsRegressor()

best_knn_regressor = GridSearchCV(knn_regressor, regressor_parameters, cv=5)
best_knn_regressor.fit(x_trainset,y_trainset)
best_k = best_knn_regressor.best_params_['n_neighbors']

model_results['regression_model'].append('KNNRegression')
model_results['regression_cvs'].append(best_knn_regressor.best_score_)
```

In [37]:

```
print('Regression 1: KNN')
print('---> Best K: %d' % best_k)
print('---> TrainSet Score: %.3f' % best_knn_regressor.score(x_trainset, y_trainset))
print('---> TestSet Score: %.3f' % best_knn_regressor.score(x_testset, y_testset))
print('---> CVS: %.3f' % best_knn_regressor.best_score_)
```

```
Regression 1: KNN
---> Best K: 10
---> TrainSet Score: 0.356
---> TestSet Score: 0.231
---> CVS: 0.194
```

1.2 : Cross Validation for KNN Regressor

In [38]:

```
errors = cross_val_score(knn_regressor, X_sample, Y_sample, cv=5)
mean_errors = npy.mean(errors)
print('Mean CVS: %.3f' % mean_errors)
```

```
Mean CVS: 0.193
```

Regression Model 2: Linear Regression

In [39]:

```
# linear regression on all attributes
linear_regressor = LinearRegression()
linear_regressor.fit(x_trainset, y_trainset)
print('TrainSet Score: %.4f' % linear_regressor.score(x_trainset, y_trainset))
print('TestSet Score: %.3f' %linear_regressor.score(x_testset, y_testset))
print('Equation:')
print('number_of_accidents')
print('= %.3f' % linear_regressor.intercept_)
for i in range(len(linear_regressor.coef_)):
    print(' + %.3f * ' % linear_regressor.coef_[i] + sample_for_visualization.columns[i
```

```
TrainSet Score: 0.2354
TestSet Score: 0.229
Equation:
number_of_accidents
= -19242724239350164.000
 + 9.833 * is_working_day
 + -11.493 * weather_type
 + 466.805 * temperature_celcius
 + -255.539 * humidity_normalized
 + 26.799 * windspeed
 + 19242724239350292.000 * season_Fall
 + 19242724239350252.000 * season_Spring
 + 19242724239350248.000 * season_Summer
 + 19242724239350328.000 * season_Winter
```

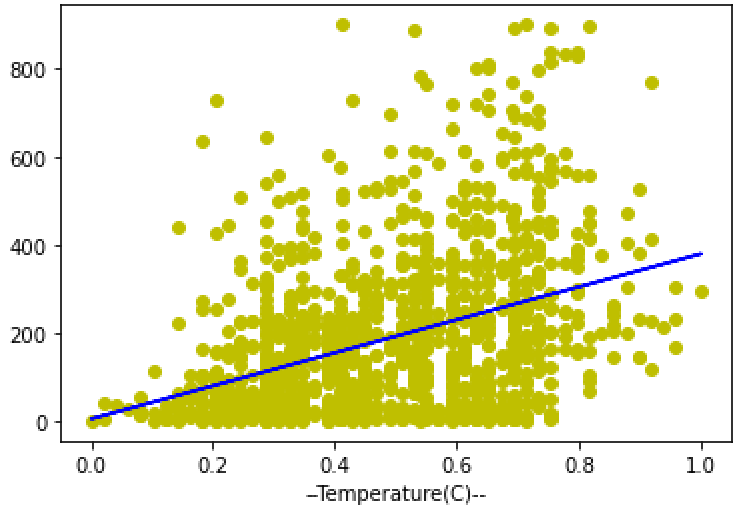
In [40]:

```
# linear model for temperature_celcius vs number_of_accidents
x_trainset_temp = x_trainset[[2]]
linear_regressor_temp = LinearRegression()
linear_regressor_temp.fit(x_trainset_temp, y_trainset)
Y_predicted = linear_regressor_temp.predict(x_trainset_temp)
# equation
print('number_of_accidents vs Temperature')
print('number_of_accidents = %.3f + %.3f * temperature_celcius' % (linear_regressor_ter
# linear plot
plot.plot(x_trainset_temp, Y_predicted, c = 'b')
plot.scatter(x_trainset_temp,y_trainset, c='y')
plot.xlabel('--Temperature(C)--')
```

```
number_of_accidents vs Temperature
number_of_accidents = 5.559 + 375.204 * temperature_celcius
```

Out[40]:

```
Text(0.5, 0, '--Temperature(C)--')
```



2.1: Cross Validation & Grid Search for Linear Regressor

In [41]:

```
linear_regressor = LinearRegression()
params = {'normalize':[False,True]}
best_linear_regressor = GridSearchCV(linear_regressor,params, cv=5, return_train_score=
best_linear_regressor.fit(x_trainset, y_trainset)
print("The finest Params: {}".format(best_linear_regressor.best_params_))
print("The top CVSscore: {:.3f}".format(best_linear_regressor.best_score_))
model_results['regression_model'].append('LinearRegression')
model_results['regression_cvs'].append(best_linear_regressor.best_score_)
mean_square_errors = cross_val_score(linear_regressor, X_sample, Y_sample, cv=5)
average_MSE = npy.mean(mean_square_errors)
print('Average CVS: %.3f' % average_MSE)
```

```
The finest Params: {'normalize': True}
The top CVSscore: 0.231
Average CVS: 0.230
```

Regression Model 3: Ridge Regressor

In [42]:

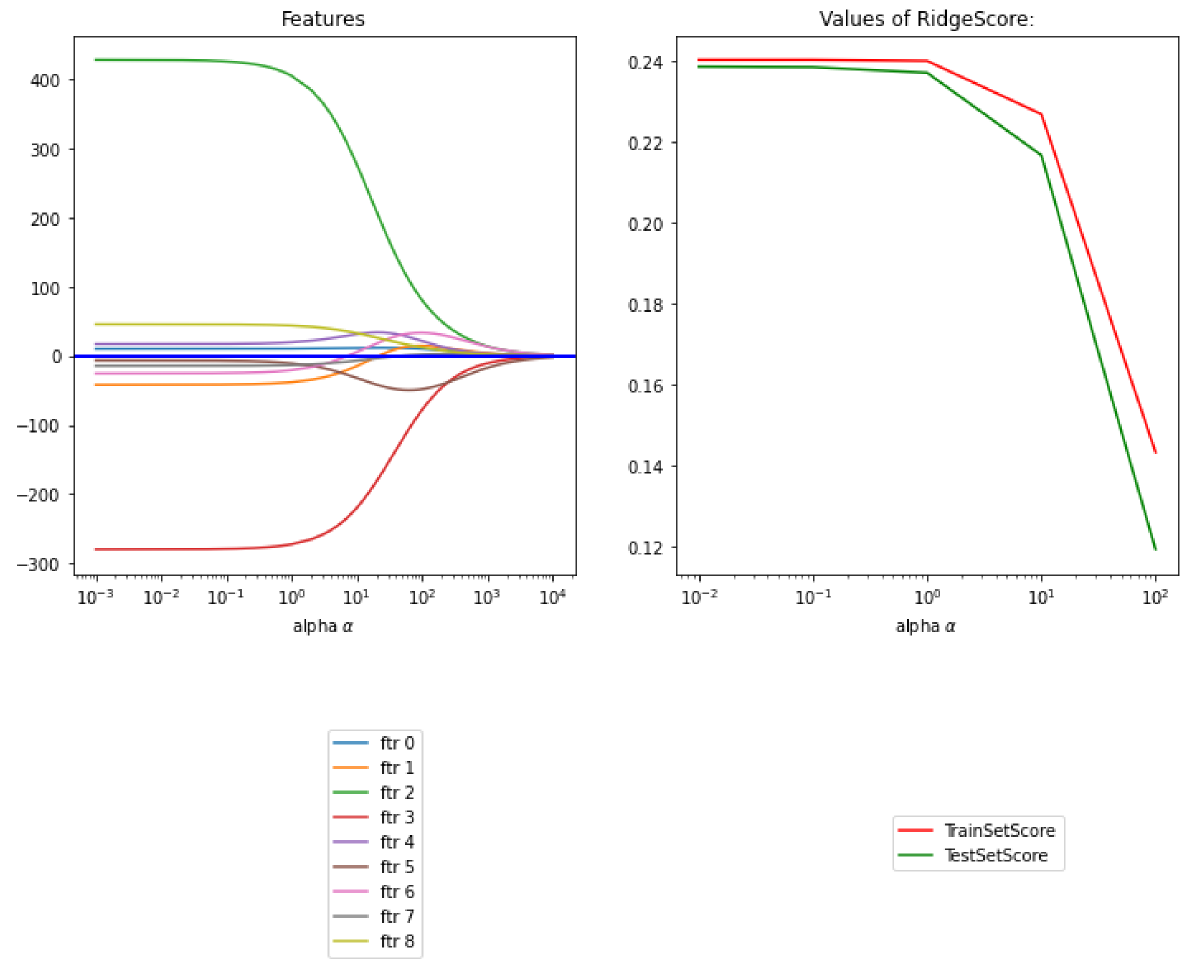
```
x_range_exp = [pow(10,i) for i in range(-2,3)]
trainset_score_array = []
testset_score_array = []

# Select best value of alpha to get high ridge score
for a in x_range_exp:
    ridge_regressor = Ridge(a)
    ridge_regressor.fit(x_trainset,y_trainset)
    trainset_score_array.append(ridge_regressor.score(x_trainset,y_trainset))
    testset_score_array.append(ridge_regressor.score(x_testset, y_testset))
```

In [43]:

```
x_range_exp1 = npy.linspace(0.001, 1, 100).reshape(-1,1)
x_range_exp2 = npy.linspace(1, 10000, 10000).reshape(-1,1)
x_range = npy.append(x_range_exp1, x_range_exp2)
coefficients = []
for a in x_range:
    ridge_regressor = Ridge(a)
    ridge_regressor.fit(x_trainset,y_trainset)
    coefficients.append(ridge_regressor.coef_ )

coefficients = npy.array(coefficients)
```

In [44]:

```
# Visualization of Feature response & Ridge scores

_, _ = plot.subplots(1,2,figsize=(12,6))
x_axis = range(1,20)
for plot_position in range(1, 3):
    plot.subplot(1, 2, plot_position)
    if plot_position == 1:
        for i in range(0,9):
            plot.plot(x_range, coefficients[:,i], label = 'ftr {:d}'.format(i))
        plot.axhline(y=0, xmin=0.0001, xmax=10000, linewidth=2, c ='b')
        plot.xlabel(r'alpha $\alpha$')
        plot.xscale('log')
        plot.legend(loc='center', bbox_to_anchor=(0.6, -0.5))
        plot.title('Features')
    if plot_position == 2:
        plot.plot(x_range_exp, trainset_score_array, c = 'r', label = 'TrainSetScore')
        plot.plot(x_range_exp, testset_score_array, c = 'g', label = 'TestSetScore')
        plot.xscale('log')
        plot.legend(loc='center', bbox_to_anchor=(0.6, -0.5))
        plot.xlabel(r'alpha $\alpha$')
        plot.title('Values of RidgeScore:')
```



3.1: Cross Validation & Grid Search for Ridge Regressor

In [45]:

```
ridge_regressor = Ridge()
parameters = {'alpha':[pow(10,i) for i in range(-3,5)]}
best_ridge_regression = GridSearchCV(ridge_regressor, parameters, cv=5)
best_ridge_regression.fit(X_sample, Y_sample)
print('The best Alpha Value: %.3f' % best_ridge_regression.best_params_['alpha'])
print('The best Score Value: %.3f' % best_ridge_regression.best_score_)
model_results['regression_model'].append('RidgeRegression')
model_results['regression_cvs'].append(best_ridge_regression.best_score_)
```

```
The best Alpha Value: 0.100
The best Score Value: 0.232
```

Regression Model 4: Lasso

In [46]:

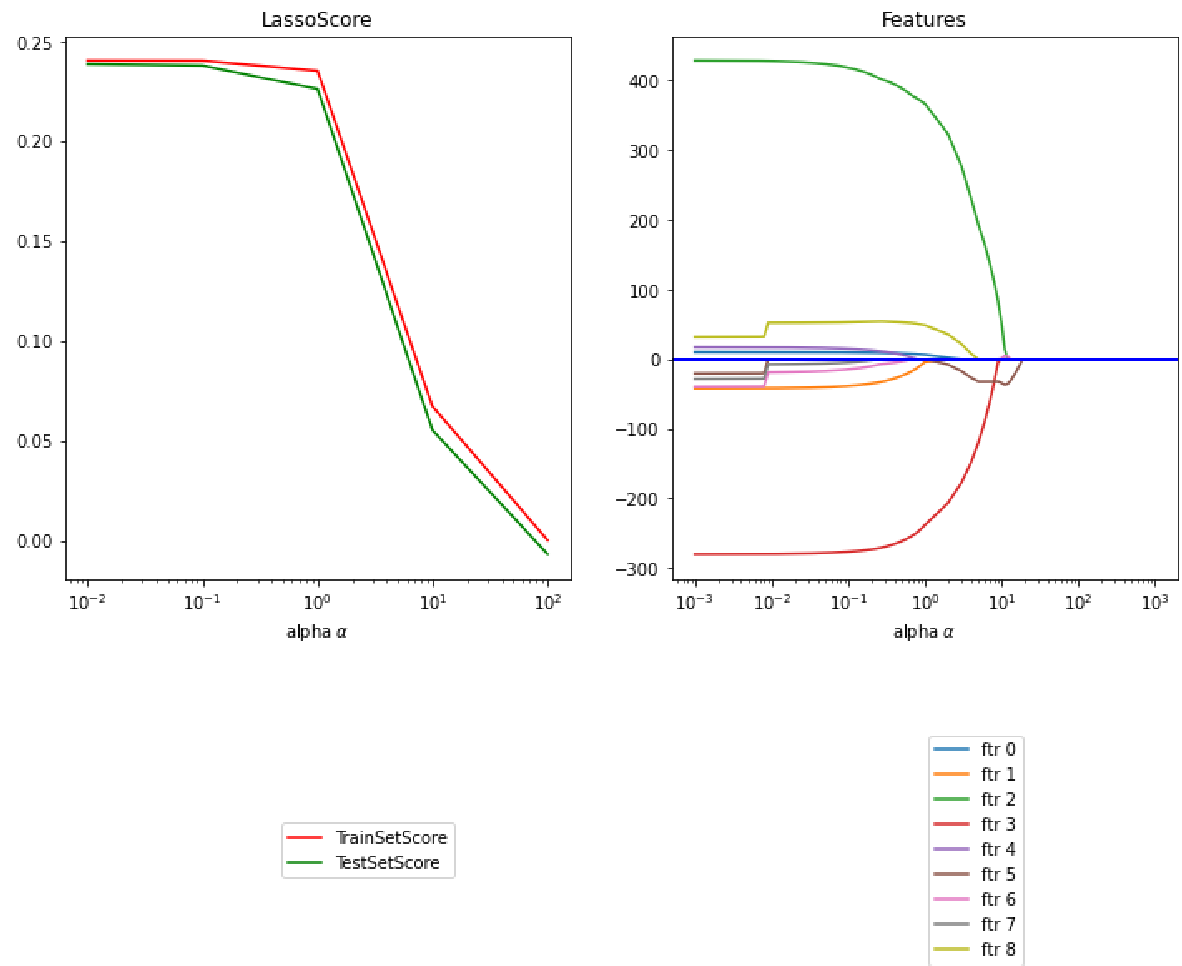
```
x_range_exp = [pow(10,i) for i in range(-2,3)]
trainset_score_array = []
testset_score_array = []
for a in x_range_exp:
    lasso_regressor = Lasso(a)
    lasso_regressor.fit(x_trainset,y_trainset)
    trainset_score_array.append(lasso_regressor.score(x_trainset,y_trainset))
    testset_score_array.append(lasso_regressor.score(x_testset, y_testset))
```

In [47]:

```
x_range_exp1 = npy.linspace(0.001, 1, 1000).reshape(-1,1)
x_range_exp2 = npy.linspace(1, 1000, 1000).reshape(-1,1)
x_range = npy.append(x_range_exp1, x_range_exp2)
coefficients = []
for a in x_range:
    lasso_regressor = Lasso(a)
    lasso_regressor.fit(x_trainset,y_trainset)
    coefficients.append(lasso_regressor.coef_ )

coefficients = npy.array(coefficients)
```

In [48]:

```python
# Visualization of Feature responses and LassoScore

_, _ = plot.subplots(1,2,figsize=(12,6))
x_axis = range(1,20)
for plot_position in range(1, 3):
    plot.subplot(1, 2, plot_position)
    if plot_position == 1:
        plot.plot(x_range_exp, trainset_score_array, c = 'r', label = 'TrainSetScore')
        plot.plot(x_range_exp, testset_score_array, c = 'g', label = 'TestSetScore')
        plot.xscale('log')
        plot.legend(loc='center', bbox_to_anchor=(0.6, -0.5))
        plot.xlabel(r'alpha $\alpha$')
        plot.title('LassoScore')
    if plot_position == 2:
        for i in range(0,9):
            plot.plot(x_range, coefficients[:,i], label = 'ftr {:d}'.format(i))
        plot.axhline(y=0, xmin=0.0001, xmax=10000, linewidth=2, c ='b')
        plot.xlabel(r'alpha $\alpha$')
        plot.xscale('log')
        plot.legend(loc='center', bbox_to_anchor=(0.6, -0.5))
        plot.title('Features')
```



4.1 Cross Validation & Grid Search for Lasso Regressor

In [49]:

```
lasso_regressor = Lasso()
parameters = {'alpha':[pow(10,i) for i in range(-3,5)]}
best_lasso_regressor = GridSearchCV(lasso_regressor, parameters, cv=5)
best_lasso_regressor.fit(X_sample, Y_sample)
print('The best AlphaValue: %.3f' % best_lasso_regressor.best_params_['alpha'])
print('The best ScoreValue: %.3f' % best_lasso_regressor.best_score_)
model_results['regression_model'].append('LassoRegression')
model_results['regression_cvs'].append(best_lasso_regressor.best_score_)
```

```
The best AlphaValue: 0.100
The best ScoreValue: 0.232
```

Regression Model 5: Polynomial Regression

In [50]:

```
polynomial_pipeline=Pipeline([
    ('features_polynomial', PolynomialFeatures()),
    ('min_max_scaler',MinMaxScaler()),
    ('linear_regression', LinearRegression())
])

# choosing grid search upto 5 degree
parameters_polynomial = {'features_polynomial__degree':range(1,5)}
best_polynomial_regressor = GridSearchCV(polynomial_pipeline, parameters_polynomial,cv=
best_polynomial_regressor.fit(x_trainset, y_trainset)
model_results['regression_model'].append('PolynomialRegression')
model_results['regression_cvs'].append(best_polynomial_regressor.best_score_)

# regressor predictions
y_trainset_pred = best_polynomial_regressor.predict(x_trainset)
y_testset_pred = best_polynomial_regressor.predict(x_testset)
```

5.1: Grid Search & Cross Validation for Polynomial Regressor

In [51]:

```
# Polynomial Regressor performance:
print('TrainSet:')
print('The Mean Square Error: {}'.format(mean_squared_error(y_trainset, y_trainset_pred
print('Root Mean Square Error: {}'.format(sqrt(mean_squared_error(y_trainset, y_trainse
print('The R2 Error: {}'.format(r2_score(y_trainset, y_trainset_pred)))
print('TestSet')
print('The Mean Square Error: {}'.format(mean_squared_error(y_testset, y_testset_pred))
print('Root Mean Square Error: {}'.format(sqrt(mean_squared_error(y_testset, y_testset_
print('The R2 Error: {}'.format(r2_score(y_testset, y_testset_pred)))
# top parameters
print('The Best params: ')
print(best_polynomial_regressor.best_params_)
# Calculate Score
print("CVSscores - Train ", best_polynomial_regressor.cv_results_['mean_train_score'])
print("CVSscores - Test ", best_polynomial_regressor.cv_results_['mean_test_score'])
```

```
TrainSet:
The Mean Square Error: 25343.216543513958
Root Mean Square Error: 159.19552928243291
The R2 Error: 0.24037165270120642
TestSet
The Mean Square Error: 28246.639423076922
Root Mean Square Error: 168.06736572897464
The R2 Error: 0.23873842165458914
The Best params:
{'features_polynomial__degree': 1}
CVSscores - Train  [0.24040483 0.26111871 0.27763293 0.42019237]
CVSscores - Test  [ 2.33791303e-01  2.09225490e-01 -2.96903744e+20 -4.963828
63e+22]
```

Regression Model 6: SimpleSVM

In [52]:

```
parmeters = {'C': [pow(10,i) for i in range(-2,3)], 'epsilon' : [pow(10,i) for i in rar

# Searching Best Params
support_vector_regressor = LinearSVR()
best_support_vector_regressor = GridSearchCV(estimator = support_vector_regressor, para
best_support_vector_regressor.fit(x_trainset, y_trainset)
support_vector_result = pds.DataFrame(best_support_vector_regressor.cv_results_)
model_results['regression_model'].append('Simple SVRRegression')
model_results['regression_cvs'].append(best_support_vector_regressor.best_score_)

# Create best SVM
support_vector_regressor = LinearSVR(C = best_support_vector_regressor.best_params_['C
support_vector_regressor.fit(x_trainset, y_trainset)

# Calculate Score
kfoldsplit10 = KFold(n_splits=10)
score_result = cross_val_score(support_vector_regressor, x_trainset, y_trainset, cv=kfc
```

6.1 Cross Validation & Grid Search for SimpleSVM

In [53]:

```
print('Top Model:')
print('The bestParams: {}'.format(best_support_vector_regressor.best_params_))
print('CVS: {:.5f}'.format(best_support_vector_regressor.best_score_))
print('TrainSetScore: %.3f' % support_vector_regressor.score(x_trainset, y_trainset))
print('TestSetScore: %.3f' % support_vector_regressor.score(x_testset, y_testset))
print('Mean CVS: %.3f' % npy.mean(score_result))
```

```
Top Model:
The bestParams: {'C': 100, 'epsilon': 100}
CVS: 0.21003
TrainSetScore: 0.215
TestSetScore: 0.201
Mean CVS: 0.210
```

In [54]:

```
# Visualization
plot.plot(range(support_vector_result.shape[0]), support_vector_result['mean_train_scor
plot.plot(range(support_vector_result.shape[0]), support_vector_result['mean_test_score
plot.xticks(range(support_vector_result.shape[0]), support_vector_result['param_C'], ro
plot.plot([best_support_vector_regressor.best_index_], support_vector_result['mean_tra
plot.plot([best_support_vector_regressor.best_index_], support_vector_result['mean_test
plot.grid()
plot.legend()
plot.xlabel(r'alpha $\alpha$')
```

Out[54]:

```
Text(0.5, 0, 'alpha $\\alpha$')
```

Model 7: LinearSVM

In [55]:

```
parameters = {'C': [pow(10,i) for i in range(-2,3)]}

# Searching Best Params
support_vector_linear_regressor = SVR(kernel='linear')
best_sv_linear_regressor = GridSearchCV(estimator = support_vector_linear_regressor, pa
best_sv_linear_regressor.fit(x_trainset,y_trainset)
support_vector_result = pds.DataFrame(best_sv_linear_regressor.cv_results_)
model_results['regression_model'].append('SVR LinearRegression')
model_results['regression_cvs'].append(best_sv_linear_regressor.best_score_)

# Create best SVM
support_vector_linear_regressor = SVR(kernel = 'linear',C = best_sv_linear_regressor.be
support_vector_linear_regressor.fit(x_trainset, y_trainset)

# Calculate Score
kfoldsplit6 = KFold(n_splits = 6)
score_result = cross_val_score(support_vector_linear_regressor, x_trainset, y_trainset,
```

7.1: Cross Validation & Grid Search for LinearSVM

In [56]:

```
print('Top Model:')
print('Params: {}'.format(best_sv_linear_regressor.best_params_))
print('CVS: {:.5f}'.format(best_sv_linear_regressor.best_score_))
print('TrainSetScore: %.3f' % support_vector_linear_regressor.score(x_trainset, y_trair
print('TestSetScore: %.3f' % support_vector_linear_regressor.score(x_testset, y_testset
print('Average CVS: %.3f' % npy.mean(score_result))
```

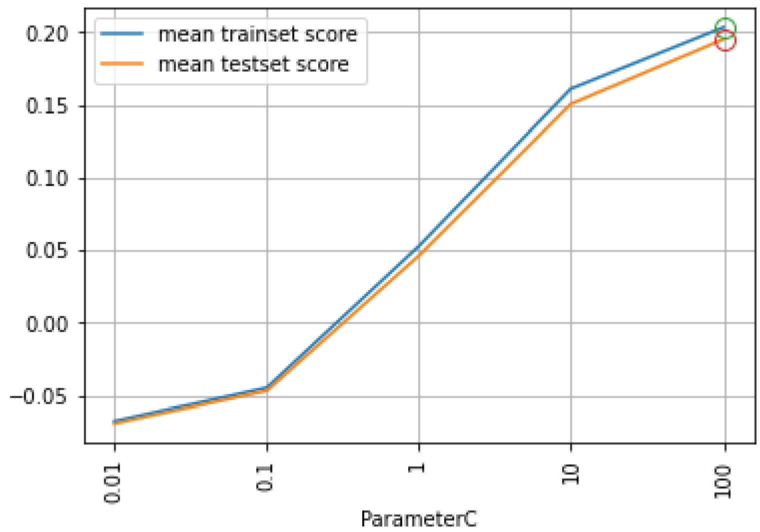
```
Top Model:
Params: {'C': 100}
CVS: 0.19524
TrainSetScore: 0.204
TestSetScore: 0.199
Average CVS: 0.195
```

In [57]:

```
# Visualization
plot.plot(range(support_vector_result.shape[0]), support_vector_result['mean_train_scor
plot.plot(range(support_vector_result.shape[0]), support_vector_result['mean_test_score
plot.xticks(range(support_vector_result.shape[0]), support_vector_result['param_C'], ro
plot.plot([best_sv_linear_regressor.best_index_], support_vector_result['mean_train_sco
plot.plot([best_sv_linear_regressor.best_index_], support_vector_result['mean_test_scor
plot.grid()
plot.legend()
plot.xlabel('ParameterC')
```

Out[57]:

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



Regression Model 8: SVM with RBF kernel

In [58]:

```
parameters = {'C': [pow(10,i) for i in range(-1,3)],'gamma':[pow(10,i) for i in range(-

# Searching Best Params
support_vector_radius_regressor = SVR(kernel='rbf')
best_support_vector_regressor = GridSearchCV(estimator = support_vector_radius_regresso
best_support_vector_regressor.fit(x_trainset,y_trainset)
support_vector_result = pds.DataFrame(best_support_vector_regressor.cv_results_)
model_results['regression_model'].append('SVR RBFRegression')
model_results['regression_cvs'].append(best_support_vector_regressor.best_score_)

# Create best SVM
support_vector_radius_regressor = SVR(kernel = 'rbf',C = best_support_vector_regressor.
support_vector_radius_regressor.fit(x_trainset, y_trainset)

# Calculate Score
kfpldsplit6 = KFold(n_splits = 6)
score_result = cross_val_score(support_vector_radius_regressor, x_trainset, y_trainset,
```

8.1 Cross Validation & Grid Search for SVM with RBF kernel

In [59]:

```
print('Top Model:')
print('Parmas: {}'.format(best_support_vector_regressor.best_params_))
print('CVS: {:.5f}'.format(best_support_vector_regressor.best_score_))
print('TrainSetScore: %.3f' % support_vector_radius_regressor.score(x_trainset, y_trair
print('TestSetScore: %.3f' % support_vector_radius_regressor.score(x_testset, y_testset
print('Average CVS: %.3f' % npy.mean(score_result))
```

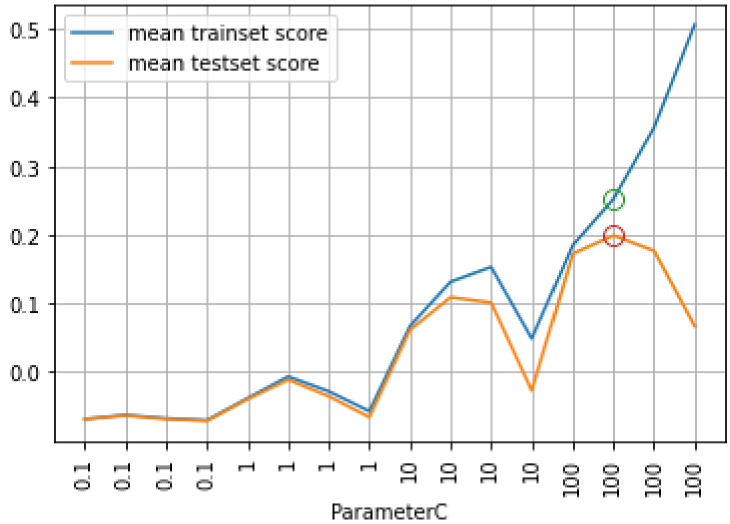
```
Top Model:
Parmas: {'C': 100, 'gamma': 1}
CVS: 0.19913
TrainSetScore: 0.251
TestSetScore: 0.195
Average CVS: 0.208
```

In [60]:

```
# Visualization
plot.plot(range(support_vector_result.shape[0]), support_vector_result['mean_train_scor
plot.plot(range(support_vector_result.shape[0]), support_vector_result['mean_test_score
plot.xticks(range(support_vector_result.shape[0]), support_vector_result['param_C'], ro
plot.plot([best_support_vector_regressor.best_index_], support_vector_result['mean_tra:
plot.plot([best_support_vector_regressor.best_index_], support_vector_result['mean_test
plot.grid()
plot.legend()
plot.xlabel('ParameterC')
```

Out[60]:

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



Regression Model 9: PolySVM

In [61]:

```
parameters = {'C': [pow(10,i) for i in range(0,5)],'degree':[1,3]}

# Searching Best Params
support_vector_polynomial_regressor = SVR(kernel='poly')
best_sv_polynomial_regressor = GridSearchCV(estimator = support_vector_polynomial_regre
best_sv_polynomial_regressor.fit(x_trainset,y_trainset)
support_vector_result = pds.DataFrame(best_sv_polynomial_regressor.cv_results_)
model_results['regression_model'].append('SVR PolynomialRegression')
model_results['regression_cvs'].append(best_sv_polynomial_regressor.best_score_)

# Create best SVM
support_vector_polynomial_regressor = SVR(kernel = 'linear',C = best_sv_polynomial_regr
support_vector_polynomial_regressor.fit(x_trainset, y_trainset)

# Calculate Score
kfoldsplit6 = KFold(n_splits = 6)
score_result = cross_val_score(support_vector_polynomial_regressor, x_trainset, y_trair
```

9.1 Cross Validation & Grid Search for PolySVM

In [62]:

```
print('Top Model:')
print('Params: {}'.format(best_sv_polynomial_regressor.best_params_))
print('CVS: {:.4f}'.format(best_sv_polynomial_regressor.best_score_))
print('TrainSetScore: %.3f' % support_vector_polynomial_regressor.score(x_trainset, y_t
print('TestSetScore: %.3f' % support_vector_polynomial_regressor.score(x_testset, y_tes
print('Average CVS: %.3f' % npy.mean(score_result))
```

```
Top Model:
Params: {'C': 1000, 'degree': 3}
CVS: 0.2139
TrainSetScore: 0.207
TestSetScore: 0.202
Average CVS: 0.198
```

In [63]:

```
# Visualization
plot.plot(range(support_vector_result.shape[0]), support_vector_result['mean_train_scor
plot.plot(range(support_vector_result.shape[0]), support_vector_result['mean_test_score
plot.xticks(range(support_vector_result.shape[0]), support_vector_result['param_C'], ro
plot.plot([best_sv_polynomial_regressor.best_index_], support_vector_result['mean_trair
plot.plot([best_sv_polynomial_regressor.best_index_], support_vector_result['mean_test_
plot.grid()
plot.legend()
plot.xlabel('ParameterC')
```

Out[63]:

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

Regression Model 10: DecisionTree

In [64]:

```
#import DecisionTree package
from sklearn.tree import DecisionTreeRegressor
```

In [65]:

```
#creating datasets for DecisionTree Reressor
X_selected = x_trainset.to_numpy()[:50,3].reshape(-1,1)
Y_selected = y_trainset[:50]

tree_regressor = DecisionTreeRegressor()
tree_regressor.fit(X_selected, Y_selected)

X_selected_new=npy.linspace(X_selected.min(), X_selected.max(), 50).reshape(50, 1)
Y_predicted = tree_regressor.predict(X_selected_new)
```

In [66]:

```
parmas = {'min_samples_leaf':range(4,31,4),
          'min_samples_split':range(4,31,4)}
```

In [67]:

```
grid_tree_regressor = GridSearchCV(tree_regressor, parmas, cv=5, return_train_score=Tru
grid_tree_regressor.fit(x_trainset, y_trainset)
```

Out[67]:

```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid={'min_samples_leaf': range(4, 31, 4),
                         'min_samples_split': range(4, 31, 4)},
             return_train_score=True)
```

In [68]:

```
df_tree_regressor = pds.DataFrame(grid_tree_regressor.cv_results_)
df_tree_regressor.loc[:,['params','mean_train_score','mean_test_score','rank_test_score
```

Out[68]:

	params	mean_train_score	mean_test_score	rank_test_score
0	{'min_samples_leaf': 4, 'min_samples_split': 4}	0.591580	-0.124168	49
1	{'min_samples_leaf': 4, 'min_samples_split': 8}	0.591580	-0.122743	48
2	{'min_samples_leaf': 4, 'min_samples_split': 12}	0.543380	-0.056009	47
3	{'min_samples_leaf': 4, 'min_samples_split': 16}	0.499259	-0.006406	46
4	{'min_samples_leaf': 4, 'min_samples_split': 20}	0.461998	0.044053	45
5	{'min_samples_leaf': 4, 'min_samples_split': 24}	0.442953	0.058457	44
6	{'min_samples_leaf': 4, 'min_samples_split': 28}	0.425702	0.067245	39
7	{'min_samples_leaf': 8, 'min_samples_split': 4}	0.454935	0.065287	40
8	{'min_samples_leaf': 8, 'min_samples_split': 8}	0.454935	0.065287	40
9	{'min_samples_leaf': 8, 'min_samples_split': 12}	0.454935	0.065287	40
10	{'min_samples_leaf': 8, 'min_samples_split': 16}	0.454935	0.065074	43
11	{'min_samples_leaf': 8, 'min_samples_split': 20}	0.437323	0.084246	38
12	{'min_samples_leaf': 8, 'min_samples_split': 24}	0.426735	0.102123	37
13	{'min_samples_leaf': 8, 'min_samples_split': 28}	0.410600	0.108167	36
14	{'min_samples_leaf': 12, 'min_samples_split': 4}	0.400734	0.129939	29
15	{'min_samples_leaf': 12, 'min_samples_split': 8}	0.400734	0.129579	35
16	{'min_samples_leaf': 12, 'min_samples_split': 12}	0.400734	0.129939	29
17	{'min_samples_leaf': 12, 'min_samples_split': 16}	0.400734	0.129939	29
18	{'min_samples_leaf': 12, 'min_samples_split': 20}	0.400734	0.129939	29
19	{'min_samples_leaf': 12, 'min_samples_split': 24}	0.400734	0.129939	29
20	{'min_samples_leaf': 12, 'min_samples_split': 28}	0.391953	0.129876	34
21	{'min_samples_leaf': 16, 'min_samples_split': 4}	0.366616	0.166023	24

	params	mean_train_score	mean_test_score	rank_test_score
22	{'min_samples_leaf': 16, 'min_samples_split': 8}	0.366616	0.166382	22
23	{'min_samples_leaf': 16, 'min_samples_split': 12}	0.366616	0.166023	24
24	{'min_samples_leaf': 16, 'min_samples_split': 16}	0.366616	0.166023	24
25	{'min_samples_leaf': 16, 'min_samples_split': 20}	0.366616	0.166382	22
26	{'min_samples_leaf': 16, 'min_samples_split': 24}	0.366616	0.166023	24
27	{'min_samples_leaf': 16, 'min_samples_split': 28}	0.366616	0.166023	24
28	{'min_samples_leaf': 20, 'min_samples_split': 4}	0.346302	0.181525	17
29	{'min_samples_leaf': 20, 'min_samples_split': 8}	0.346302	0.181525	17
30	{'min_samples_leaf': 20, 'min_samples_split': 12}	0.346302	0.181525	17
31	{'min_samples_leaf': 20, 'min_samples_split': 16}	0.346302	0.181832	15
32	{'min_samples_leaf': 20, 'min_samples_split': 20}	0.346302	0.181832	15
33	{'min_samples_leaf': 20, 'min_samples_split': 24}	0.346302	0.181525	17
34	{'min_samples_leaf': 20, 'min_samples_split': 28}	0.346302	0.181525	17
35	{'min_samples_leaf': 24, 'min_samples_split': 4}	0.331966	0.203453	10
36	{'min_samples_leaf': 24, 'min_samples_split': 8}	0.331966	0.203694	3
37	{'min_samples_leaf': 24, 'min_samples_split': 12}	0.331966	0.203453	10
38	{'min_samples_leaf': 24, 'min_samples_split': 16}	0.331966	0.203453	10
39	{'min_samples_leaf': 24, 'min_samples_split': 20}	0.331966	0.203453	10
40	{'min_samples_leaf': 24, 'min_samples_split': 24}	0.331966	0.203694	3
41	{'min_samples_leaf': 24, 'min_samples_split': 28}	0.331966	0.203453	10
42	{'min_samples_leaf': 28, 'min_samples_split': 4}	0.324127	0.203748	1
43	{'min_samples_leaf': 28, 'min_samples_split': 8}	0.324127	0.203546	5
44	{'min_samples_leaf': 28, 'min_samples_split': 12}	0.324127	0.203748	1
45	{'min_samples_leaf': 28, 'min_samples_split': 16}	0.324127	0.203546	5
46	{'min_samples_leaf': 28, 'min_samples_split': 20}	0.324127	0.203546	5

	params	mean_train_score	mean_test_score	rank_test_score
47	{'min_samples_leaf': 28, 'min_samples_split': 24}	0.324127	0.203546	5
48	{'min_samples_leaf': 28, 'min_samples_split': 28}	0.324127	0.203546	5

In [69]:

```
print("Best CV accuracy: {:.5f}".format(grid_tree_regressor.best_score_))
print("The BestParams: {}".format(grid_tree_regressor.best_params_))
print("TestSetScore: {:.2f}".format(grid_tree_regressor.score(x_testset, y_testset)))
print("TrainSetScore: {:.2f}".format(grid_tree_regressor.score(x_trainset, y_trainset))
```

```
Best CV accuracy: 0.20375
The BestParams: {'min_samples_leaf': 28, 'min_samples_split': 4}
TestSetScore: 0.21
TrainSetScore: 0.32
```

In [70]:

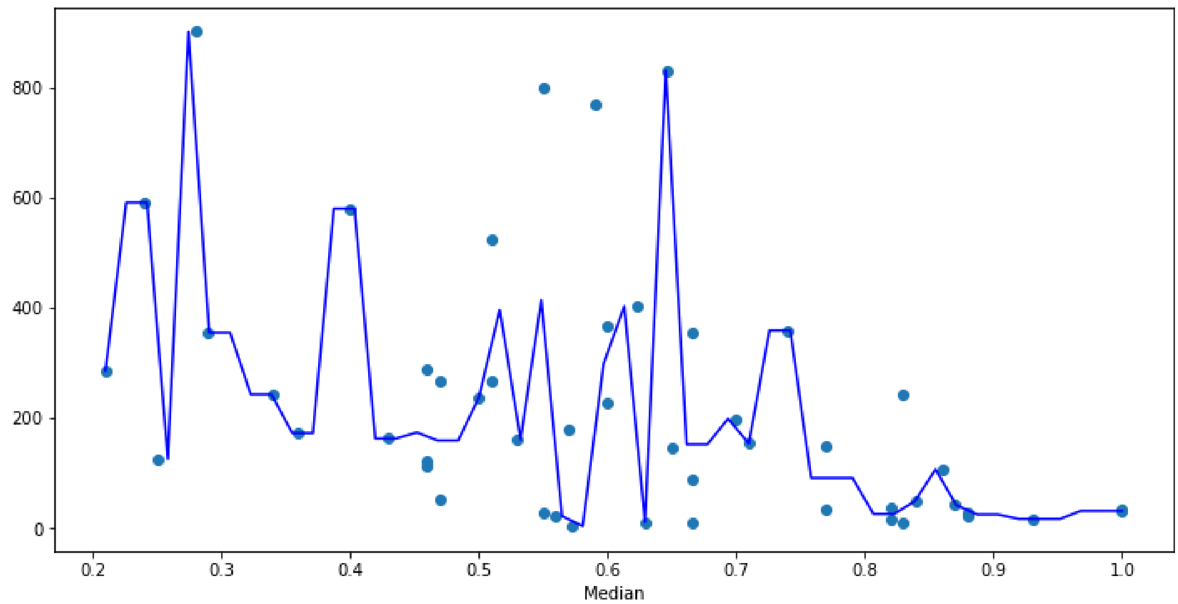
```
model_results['regression_model'].append('Decision Tree Regression')
model_results['regression_cvs'].append(grid_tree_regressor.best_score_)
```

In [71]:

```
#visualization
plot.subplots(figsize = (12,6))
plot.plot(X_selected_new, Y_predicted, c = 'b')
plot.xlabel('Median')
plot.scatter(X_selected, Y_selected)

```

Out[71]:

```
<matplotlib.collections.PathCollection at 0xa0d29638e0>
```

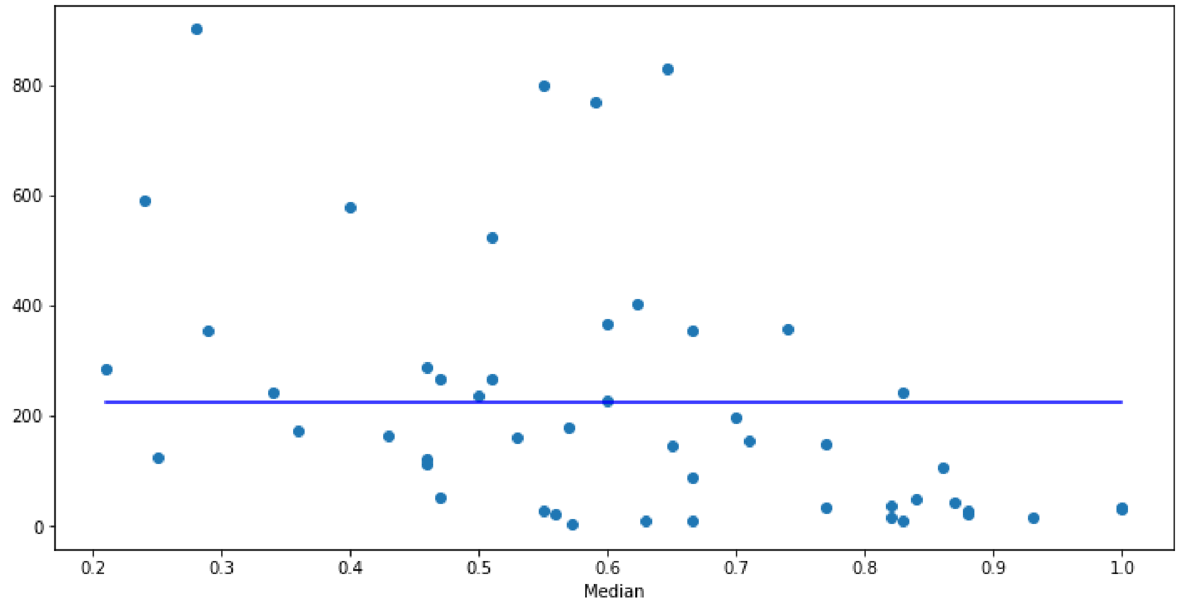


In [72]:

```
tree_regressor = DecisionTreeRegressor(min_samples_split=5,min_samples_leaf=25)
tree_regressor.fit(X_selected, Y_selected)

X_selected_new=npy.linspace(X_selected.min(), X_selected.max(), 50).reshape(50, 1)
Y_predicted = tree_regressor.predict(X_selected_new)

plot.subplots(figsize = (12,6))
plot.plot(X_selected_new, Y_predicted, c = 'b')
plot.xlabel('Median')
plot.scatter(X_selected, Y_selected)

```

Out[72]:

```
<matplotlib.collections.PathCollection at 0xa0d29d6250>
```



Cross Validation Score for each Regressors:

In [73]:

```
model_results
```

Out[73]:

```
{'regression_model': ['KNNRegression',
  'LinearRegression',
  'RidgeRegression',
  'LassoRegression',
  'PolynomialRegression',
  'Simple SVRRegression',
  'SVR LinearRegression',
  'SVR RBFRegression',
  'SVR PolynomialRegression',
  'Decision Tree Regression'],
 'regression_cvs': [0.19382051280140317,
  0.23111685352016992,
  0.23177909863391913,
  0.23177203525750628,
  0.23379130337674514,
  0.2100338286597027,
  0.19524455156203965,
  0.1991313920206221,
  0.2139199935228424,
  0.20374793014620046]}
```

SUMMARY: Best Regressor Model

In [74]:

```
model_dataframe = pds.DataFrame(data = model_results)
model_dataframe
```

Out[74]:

	regression_model	regression_cvs
0	KNNRegression	0.193821
1	LinearRegression	0.231117
2	RidgeRegression	0.231779
3	LassoRegression	0.231772
4	PolynomialRegression	0.233791
5	Simple SVRRegression	0.210034
6	SVR LinearRegression	0.195245
7	SVR RBFRegression	0.199131
8	SVR PolynomialRegression	0.213920
9	Decision Tree Regression	0.203748

In [75]:

```
# selecting best regressor with maximum cvScore
top_model = model_dataframe.loc[model_dataframe['regression_cvs'].idxmax()]
top_model
```

Out[75]:

```
regression_model    PolynomialRegression
regression_cvs                  0.233791
Name: 4, dtype: object
```

In [76]:

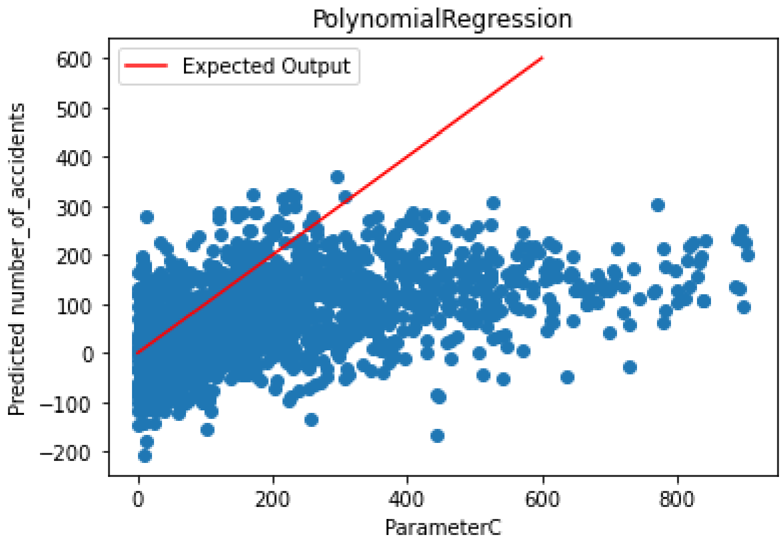
```
top_model_name = top_model['regression_model']

# selecting best regressor for all predictions
if top_model_name == 'KNNRegression':
    y_predicted = best_knn_regressor.predict(X_sample)
if top_model_name == 'LinearRegression':
    y_predicted = best_linear_regressor.predict(X_sample)
if top_model_name == 'RidgeRegression':
    y_predicted = best_ridge_regression.predict(X_sample)
if top_model_name == 'LassoRegression':
    y_predicted = best_lasso_regressor.predict(X_sample)
if top_model_name == 'PolynomialRegression':
    y_predicted = best_polynomial_regressor.predict(X_sample)
if top_model_name == 'Simple SVRRegression':
    y_predicted = best_support_vector_regressor.predict(X_sample)
if top_model_name == 'SVR LinearRegression':
    y_predicted = best_sv_linear_regressor.predict(X_sample)
if top_model_name == 'SVR RBFRegression':
    y_predicted = best_support_vector_regressor.predict(X_sample)
if top_model_name == 'SVR PolynomialRegression':
    y_predicted = best_sv_polynomial_regressor.predict(X_sample)
```

In [77]:

```
## Visualization: Actual number_of_accident vs Predicted number_of_accidents
plot.scatter(Y_sample, y_predicted)
plot.title(top_model['regression_model'])
plot.xlabel('Actual number_of_accidents')
plot.ylabel('Predicted number_of_accidents')
plot.plot([0, 600], [0, 600], 'red', label = 'Expected Output')
plot.xlabel('ParameterC')
plot.legend()

```

Out[77]:

```
<matplotlib.legend.Legend at 0xa0d44ec070>
```



In [78]:

```
model_results
```

Out[78]:

```
{'regression_model': ['KNNRegression',
   'LinearRegression',
   'RidgeRegression',
   'LassoRegression',
   'PolynomialRegression',
   'Simple SVRRegression',
   'SVR LinearRegression',
   'SVR RBFRegression',
   'SVR PolynomialRegression',
   'Decision Tree Regression'],
  'regression_cvs': [0.19382051280140317,
  0.23111685352016992,
  0.23177909863391913,
  0.23177203525750628,
  0.23379130337674514,
  0.2100338286597027,
  0.19524455156203965,
  0.1991313920206221,
  0.2139199935228424,
  0.20374793014620046]}
```

By observing the CV Scores of all 10 Regressors above, below stated Regression model has the

highest CV score.

Best Regression model for this DataSet:

In [79]:

```
# Best Model among 10
top_model_name
```

Out[79]:

```
'PolynomialRegression'
```

Hence, above stated regressor is the best model for this dataset so far.

End of Project 1 : Regression

Initials:

-rp

Project Part 2: Regression

Step 1: Initializations

In [80]:

```
# Import Relevant libraries
from sklearn.metrics import accuracy_score
from sklearn.ensemble import VotingRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.decomposition import PCA
from keras.models import Sequential
from keras.layers import Dense
```

Recommended changes in Dataframes

In [96]:

```
# model dataframe
model_dataframe = pds.DataFrame(data = model_results)
model_dataframe.columns = ['Model','Grid Search']
model_dataframe['PCA'] = npy.nan
model_dataframe
```

Out[96]:

	Model	Grid Search	PCA
0	KNNRegression	0.193821	NaN
1	LinearRegression	0.231117	NaN
2	RidgeRegression	0.231779	NaN
3	LassoRegression	0.231772	NaN
4	PolynomialRegression	0.233791	NaN
5	Simple SVRRegression	0.210034	NaN
6	SVR LinearRegression	0.195245	NaN
7	SVR RBFRegression	0.199131	NaN
8	SVR PolynomialRegression	0.213920	NaN
9	Decision Tree Regression	0.203748	NaN

In [77]:

```
# Creating funtion for displaying model statistics:
def printModSpecs(mod):
    print(f'The Best Mean CVSscore: {mod.best_score_}')
    print(f'The Best params: {mod.best_params_}')
    print(f'Train datset score: {mod.score(x_trainset,y_trainset)}')
    print(f'Test dataset score: {mod.score(x_testset,y_testset)}')
    print('r2Score: ', r2_score(y_testset,y_predicted))
```

Task 1: Bagging 1: SimpleSVR

In [78]:

```
# Simple SVR
bag_svrR = BaggingRegressor(base_estimator=LinearSVR(), bootstrap=True, random_state=0,
bag_svrR_param = {'base_estimator__C': [pow(10,i) for i in range(-2,3)],'base_estimator
                  'n_estimators': [10,25]}
best_bag_svrR = GridSearchCV(bag_svrR, bag_svrR_param, cv=6, return_train_score=True, )
best_bag_svrR.fit(x_trainset,y_trainset)
y_predicted = best_bag_svrR.predict(x_testset)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:976: Converg
enceWarning: Liblinear failed to converge, increase the number of iteration
s.
  warnings.warn("Liblinear failed to converge, increase "
```

In [79]:

```
# Statistics
model_dataframe.at[5,'Bagging'] = best_bag_svrR.best_score_
printModSpecs(best_bag_svrR)
```

```
Best Mean Cross Validation Score: 0.18243483483299197
Best Parameters: {'base_estimator__C': 100, 'base_estimator__epsilon': 100,
'n_estimators': 25}
Train score: 0.19407469108335318
Test score: 0.23410301171300174
r2_score:  0.23410301171300174
```

Task 1: Bagging 2: LinearSVR

In [80]:

```
# Linear SVR
bag_svrLR = BaggingRegressor(base_estimator=SVR(kernel='linear'), bootstrap=True, rando
bag_svrLR_param = {'base_estimator__C': [pow(10,i) for i in range(-2,3)],'n_estimators'
best_bag_svrLR = GridSearchCV(bag_svrLR, bag_svrLR_param, cv=6, return_train_score=True
best_bag_svrLR.fit(x_trainset,y_trainset)
y_predicted = best_bag_svrLR.predict(x_testset)
```

In [81]:

```
# Statistics
model_dataframe.at[6,'Bagging'] = best_bag_svrLR.best_score_
printModSpecs(best_bag_svrLR)
```

```
Best Mean Cross Validation Score: 0.16103014538953012
Best Parameters: {'base_estimator__C': 100, 'n_estimators': 25}
Train score: 0.16982267242507365
Test score: 0.2136540614797091
r2_score:  0.2136540614797091
```

Task 2: Pasting 1: SimpleSVR

In [106]:

```
# Simple SVR
pas_svrR = BaggingRegressor(base_estimator=SVR(), bootstrap=False, random_state=0, oob_
pas_svrR_param = {'base_estimator__C': [pow(10,i) for i in range(-2,3)],'base_estimator
                  'n_estimators': [10,25]}
best_pas_svrR = GridSearchCV(pas_svrR, pas_svrR_param, cv=6, return_train_score=True, )
best_pas_svrR.fit(x_trainset,y_trainset)
y_predicted = best_pas_svrR.predict(x_testset)
```

In [107]:

```
# Statistics
model_dataframe.at[5,'Pasting'] = best_pas_svrR.best_score_
printModSpecs(best_pas_svrR)
```

```
Best Mean Cross Validation Score: 0.19029663767959648
Best Parameters: {'base_estimator__C': 100, 'base_estimator__epsilon': 100,
'n_estimators': 10}
Train score: 0.2217538056205457
Test score: 0.2378640819642739
r2_score:  0.2378640819642739
```

Task 2: Pasting 2: LinearSVR

In [305]:

```
# Linear SVR
pas_svrLR = BaggingRegressor(base_estimator=SVR(kernel='linear'), bootstrap=False, ranc
pas_svrLR_param = {'base_estimator__C': [pow(10,i) for i in range(-2,3)],'base_estimatc
                  'n_estimators': [10,25]}
best_pas_svrLR = GridSearchCV(pas_svrLR, pas_svrLR_param, cv=6, return_train_score=True
best_pas_svrLR.fit(x_trainset,y_trainset)
y_predicted = best_pas_svrLR.predict(x_testset)
```

In [372]:

```
# Statistics
model_dataframe.at[6,'Pasting'] = best_pas_svrLR.best_score_
printModSpecs(best_pas_svrLR)
```

```
Best Mean Cross Validation Score: 0.2067496703829704
Best Parameters: {'base_estimator__C': 100, 'base_estimator__epsilon': 100,
'n_estimators': 25}
Train score: 0.22021452143271414
Test score: 0.2804394915883337
r2_score:  0.3163589705657385
```

Task 3: Adaboosting 1: SimpleSVR

In [307]:

```
# Simple SVR
adr_svrR =AdaBoostRegressor(base_estimator=SVR(),random_state=42)
adr_svrR_param = {'base_estimator__C': [pow(10,i) for i in range(-2,3)],'base_estimator
                  'n_estimators' : [100,150],'learning_rate' : [0.5,1.0,2],}
best_adr_svrR = GridSearchCV(adr_svrR, adr_svrR_param,cv=5, return_train_score=True, )
best_adr_svrR.fit(x_trainset,y_trainset)
y_predicted = best_adr_svrR.predict(x_testset)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: Future
Warning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly
to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: Future
Warning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly
to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: Future
Warning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly
to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: Future
Warning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly
to 'auto' or 'scale' to avoid this warning.
```

In [373]:

```
# Statistics
model_dataframe.at[5,'Adaboosting'] = best_adr_svrR.best_score_
printModSpecs(best_adr_svrR)
```

```
Best Mean Cross Validation Score: 0.21680841081746885
Best Parameters: {'base_estimator__C': 100, 'base_estimator__epsilon': 1, 'l
earning_rate': 0.5, 'n_estimators': 100}
Train score: 0.24134983186629663
Test score: 0.29764276063912254
r2_score:  0.3163589705657385
```

Task 3: Adaboosting 2: LinearSVR

In [309]:

```
# Linear SVR
adr_svrLR =AdaBoostRegressor(base_estimator=SVR(kernel='linear'),random_state=42)
adr_svrLR_param = {'base_estimator__C': [pow(10,i) for i in range(-2,3)],'base_estimat
                   'n_estimators' : [100,150],'learning_rate' : [0.5,1.0,2],}
best_adr_svrLR = GridSearchCV(adr_svrLR, adr_svrLR_param,cv=5, return_train_score=True,
best_adr_svrLR.fit(x_trainset,y_trainset)
y_predicted = best_adr_svrLR.predict(x_testset)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:814: DeprecationWarning: The default of the `iid` parameter will change fr
om True to False in version 0.22 and will be removed in 0.24. This will chan
ge numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

In [374]:

```
# Statistics
model_dataframe.at[6,'Adaboosting'] = best_adr_svrLR.best_score_
printModSpecs(best_adr_svrLR)
```

```
Best Mean Cross Validation Score: 0.21415342407485893
Best Parameters: {'base_estimator__C': 100, 'base_estimator__epsilon': 1, 'l
earning_rate': 0.5, 'n_estimators': 100}
Train score: 0.22458185052979318
Test score: 0.2882730165399616
r2_score:  0.3163589705657385
```

Task 4: Gradient Boosting 1

In [311]:

```
# Gradient boosting Regression

gbr= GradientBoostingRegressor(random_state=42)
gbr_param = {
               'max_depth' : [2,3,4],
               'n_estimators' : [25,100],
               'learning_rate' : [0.5,1.0,2],
              }
best_gbr = GridSearchCV(gbr, gbr_param,cv=5, return_train_score=True, )
best_gbr.fit(x_trainset,y_trainset)
y_predicted = best_gbr.predict(x_testset)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:814: DeprecationWarning: The default of the `iid` parameter will change fr
om True to False in version 0.22 and will be removed in 0.24. This will chan
ge numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

In [312]:

```
# Statistics
printModSpecs(best_gbr)
```

```
Best Mean Cross Validation Score: 0.21323662990755018
Best Parameters: {'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 25}
Train score: 0.33723565346890694
Test score: 0.3163589705657385
r2_score:  0.3163589705657385
```

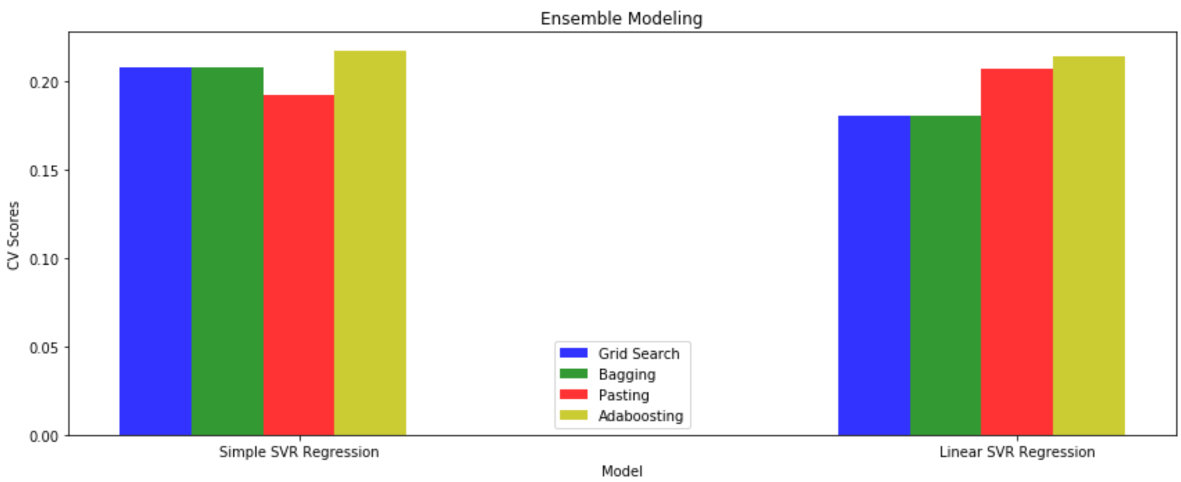
Task 5: Ensemble Comparison

In [83]:

```
# Model Comparison
check_name = model_dataframe['Model'] == 'Simple SVR Regression'
ensemble_df = model_dataframe[check_name]
ensemble_df = ensemble_df.append(model_dataframe[model_dataframe['Model'] == 'Linear SV
```

In [376]:

```
# Visualization
fig, ax = plot.subplots(figsize=(12,5))
index = npy.arange(2)
bar_width = 0.1
opacity = 0.8

rects1 = plot.bar(index, ensemble_df['Grid Search'], bar_width, alpha=opacity, color='b
rects2 = plot.bar(index + bar_width, ensemble_df['Bagging'], bar_width, alpha=opacity,
rects3 = plot.bar(index + 2 * bar_width, ensemble_df['Pasting'], bar_width, alpha=opaci
rects4 = plot.bar(index + 3 * bar_width, ensemble_df['Adaboosting'], bar_width, alpha=c

plot.xlabel('Model')
plot.ylabel('CV Scores')
plot.title('Ensemble Modeling')
plot.xticks(index + 2 * bar_width, ensemble_df['Model'])
plot.legend()

plot.tight_layout()
plot.show()
```



Task 6: PCA Model: Data Initialization

In [82]:

```
#Creating PCA Model
pca = PCA(n_components = 0.95, random_state = 3)
pca.fit(x_trainset)

# creating x_trainset and x_testset
x_train_pca = pca.transform(x_trainset)
x_test_pca = pca.transform(x_testset)
```

Task 7: PCA Models for comparison

Regression Model 1: KNN

In [83]:

```
#KNN Regression

params = {'n_neighbors':[2,3,4,5,6,7,8,9,10]}
knn_R = KNeighborsRegressor()

best_KnnR_pca = GridSearchCV(knn_R, params, cv=5)
best_KnnR_pca.fit(x_train_pca,y_trainset)
k = best_KnnR_pca.best_params_['n_neighbors']

model_dataframe.at[0,'PCA'] = best_KnnR_pca.best_score_
print('KNN Regression')
print('The Best params: {}'.format(best_KnnR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_KnnR_pca.best_score_))
```

```
KNN Regression
The Best params: {'n_neighbors': 10}
The Best CVSscore: 0.1450
```

Regression Model 2: Linear

In [84]:

```
#Linear Regression

lin_R = LinearRegression()
parameters = {'normalize':[True,False]}

best_linR_pca = GridSearchCV(lin_R,parameters, cv=6, return_train_score=True)
best_linR_pca.fit(x_train_pca, y_trainset)

model_dataframe.at[1,'PCA'] = best_linR_pca.best_score_
print('Linear Regression')
print('The Best params: {}'.format(best_linR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_linR_pca.best_score_))
```

```
Linear Regression
The Best params: {'normalize': False}
The Best CVSscore: 0.1446
```

Regression Model 3: Ridge

In [85]:

```
# Ridge Regression

rid_R = Ridge()
params = {'alpha':[0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}

best_ridR_pca = GridSearchCV(rid_R, params, cv=5)
best_ridR_pca.fit(x_train_pca, y_trainset)

model_dataframe.at[2,'PCA'] = best_ridR_pca.best_score_
print('Ridge Regression')
print('The Best params: {}'.format(best_ridR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_ridR_pca.best_score_))
```

```
Ridge Regression
The Best params: {'alpha': 0.1}
The Best CVSscore: 0.1482
```

Regression Model 4: Lasso

In [86]:

```
# Lasso Regression

las_R = Lasso()
params = {'alpha':[0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}
best_lasR_pca = GridSearchCV(las_R, params, cv=5)
best_lasR_pca.fit(x_train_pca, y_trainset)

model_dataframe.at[3,'PCA'] = best_lasR_pca.best_score_
print('Lasso Regression')
print('The Best params: {}'.format(best_lasR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_lasR_pca.best_score_))
```

```
Lasso Regression
The Best params: {'alpha': 0.001}
The Best CVSscore: 0.1482
```

Regression Model 5: Polynomial

In [87]:

```
# Polynomial Regression

pipe_poly=Pipeline([
    ('polynomialfeatures', PolynomialFeatures()),
    ('scaler',MinMaxScaler()),
    ('norm_reg', LinearRegression())
])
param_poly = {'polynomialfeatures__degree':range(1,5)}

best_polR_pca = GridSearchCV(pipe_poly, param_poly,cv=5, n_jobs=-1, return_train_score
best_polR_pca.fit(x_train_pca, y_trainset)

model_dataframe.at[4,'PCA'] = best_polR_pca.best_score_
print('Polynomial Regression')
print('The Best params: {}'.format(best_polR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_polR_pca.best_score_))
```

```
Polynomial Regression
The Best params: {'polynomialfeatures__degree': 3}
The Best CVSscore: 0.2192
```

Regression Model 6: SimpleSVM

In [88]:

```
# Simple SVM Regression
parms_svr = {'C': [pow(10,i) for i in range(-2,3)], 'epsilon' : [pow(10,i) for i in rar
svr_R = LinearSVR()

best_svrR_pca = GridSearchCV(estimator = svr_R, param_grid = parms_svr, return_train_sc
best_svrR_pca.fit(x_train_pca, y_trainset)

model_dataframe.at[5,'PCA'] = best_svrR_pca.best_score_
print('Simple SVM Regression')
print('The Best params: {}'.format(best_svrR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_svrR_pca.best_score_))
```

```
Simple SVM Regression
The Best params: {'C': 100, 'epsilon': 100}
The Best CVSscore: 0.1222
```

Regression Model 7: LinearSVM

In [89]:

```
# Linear SVM Regression
parms_svr = {'C': [0.01,0.1, 1, 10, 100]}
svrL_R = SVR(kernel='linear')

best_svrLR_pca = GridSearchCV(estimator = svrL_R, param_grid = parms_svr, return_train_
best_svrLR_pca.fit(x_train_pca,y_trainset)

model_dataframe.at[6,'PCA'] = best_svrLR_pca.best_score_
print('Linear SVM Regression')
print('The Best params: {}'.format(best_svrLR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_svrLR_pca.best_score_))
```

```
Linear SVM Regression
The Best params: {'C': 100}
The Best CVSscore: 0.1029
```

Regression Model 8: RBF SVM

In [90]:

```
# RBF SVM Regression
parms_svr = {'C': [0.1, 1, 10, 100],'gamma':[0.1, 1, 10, 100]}
svrR_R = SVR(kernel='rbf')

best_svrRR_pca = GridSearchCV(estimator = svrR_R, param_grid = parms_svr, return_train_
best_svrRR_pca.fit(x_train_pca,y_trainset)

model_dataframe.at[7,'PCA'] = best_svrRR_pca.best_score_
print('RBF SVM Regression')
print('The Best params: {}'.format(best_svrRR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_svrRR_pca.best_score_))
```

```
RBF SVM Regression
The Best params: {'C': 100, 'gamma': 1}
The Best CVSscore: 0.1310
```

Regression Model 9: Poly SVM Regression

In [91]:

```
# Poly SVM Regression
parms_svr = {'C': [1, 10, 100,1000,10000],'degree':[1,3]}
svrP_R = SVR(kernel='poly')
best_svrPR_pca = GridSearchCV(estimator = svrP_R, param_grid = parms_svr, return_train_
best_svrPR_pca.fit(x_train_pca,y_trainset)

model_dataframe.at[8,'PCA'] = best_svrPR_pca.best_score_
print('Poly SVM Regression')
print('The Best params: {}'.format(best_svrPR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_svrPR_pca.best_score_))
```

```
Poly SVM Regression
The Best params: {'C': 10000, 'degree': 3}
The Best CVSscore: 0.1544
```

Regression Model 10: Decision Tree

In [99]:

```
#import DecisionTree package
from sklearn.tree import DecisionTreeRegressor
```

In [92]:

```
#creating datasets for DecisionTree Reressor
X_selected = x_trainset.to_numpy()[:50,3].reshape(-1,1)
Y_selected = y_trainset[:50]

tree_regressor = DecisionTreeRegressor()
tree_regressor.fit(X_selected, Y_selected)

X_selected_new=npy.linspace(X_selected.min(), X_selected.max(), 50).reshape(50, 1)
Y_predicted = tree_regressor.predict(X_selected_new)
print('Decision Tree Regression')
print('The Best params: {}'.format(best_svrR_pca.best_params_))
print('The Best CVSscore: {:.4f}'.format(best_svrR_pca.best_score_))
```

```
Decision Tree Regression
The Best params: {'C': 100, 'epsilon': 100}
The Best CVSscore: 0.1222
```

In [101]:

```
parmas = {'min_samples_leaf':range(4,31,4),
          'min_samples_split':range(4,31,4)}
```

In [102]:

```
grid_tree_regressor = GridSearchCV(tree_regressor, parmas, cv=5, return_train_score=Tru
grid_tree_regressor.fit(x_trainset, y_trainset)
```

Out[102]:

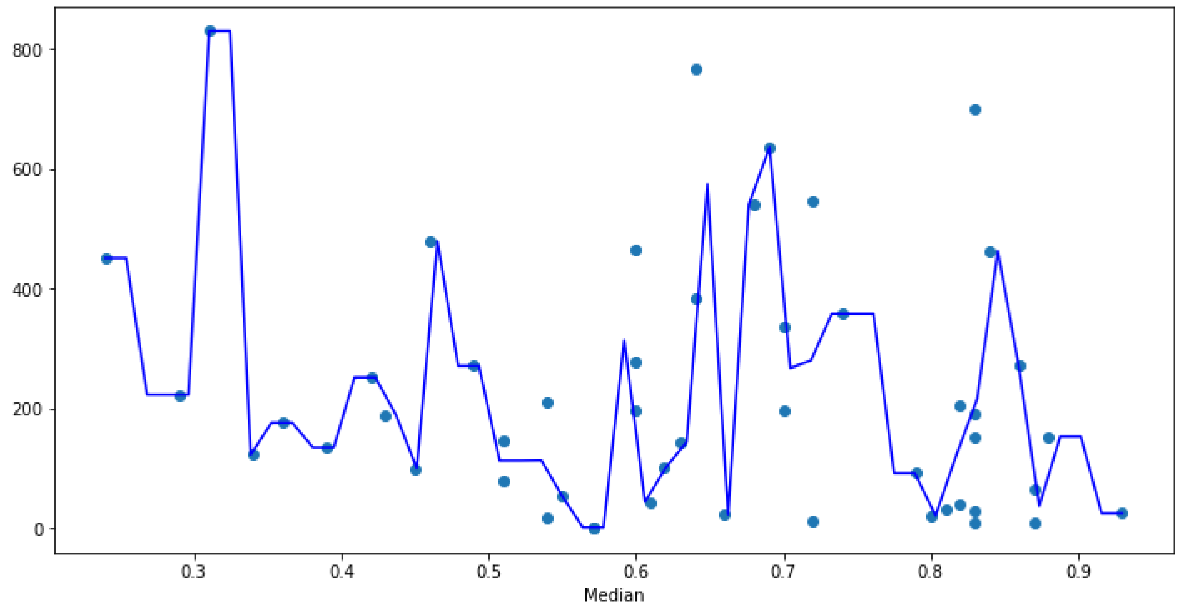
```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid={'min_samples_leaf': range(4, 31, 4),
                         'min_samples_split': range(4, 31, 4)},
             return_train_score=True)
```

In [104]:

```
#visualization
plot.subplots(figsize = (12,6))
plot.plot(X_selected_new, Y_predicted, c = 'b')
plot.xlabel('Median')
plot.scatter(X_selected, Y_selected)

```

Out[104]:

```
<matplotlib.collections.PathCollection at 0xac64c4250>
```

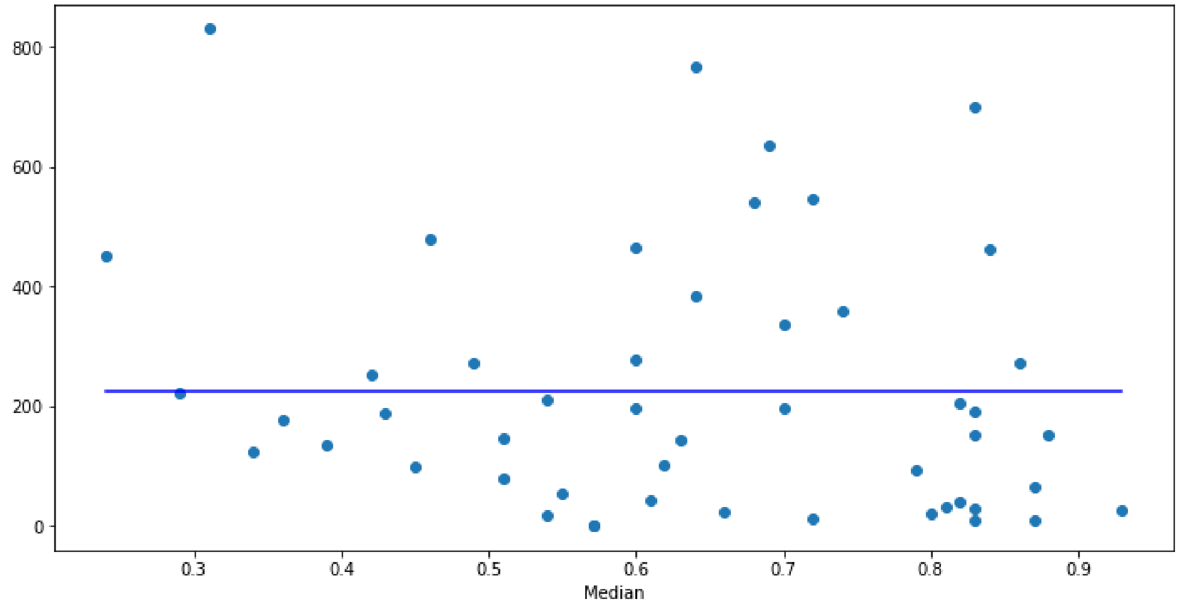


In [105]:

```
#Visualization

tree_regressor = DecisionTreeRegressor(min_samples_split=5,min_samples_leaf=25)
tree_regressor.fit(X_selected, Y_selected)

X_selected_new=npy.linspace(X_selected.min(), X_selected.max(), 50).reshape(50, 1)
Y_predicted = tree_regressor.predict(X_selected_new)

plot.subplots(figsize = (12,6))
plot.plot(X_selected_new, Y_predicted, c = 'b')
plot.xlabel('Median')
plot.scatter(X_selected, Y_selected)

```

Out[105]:

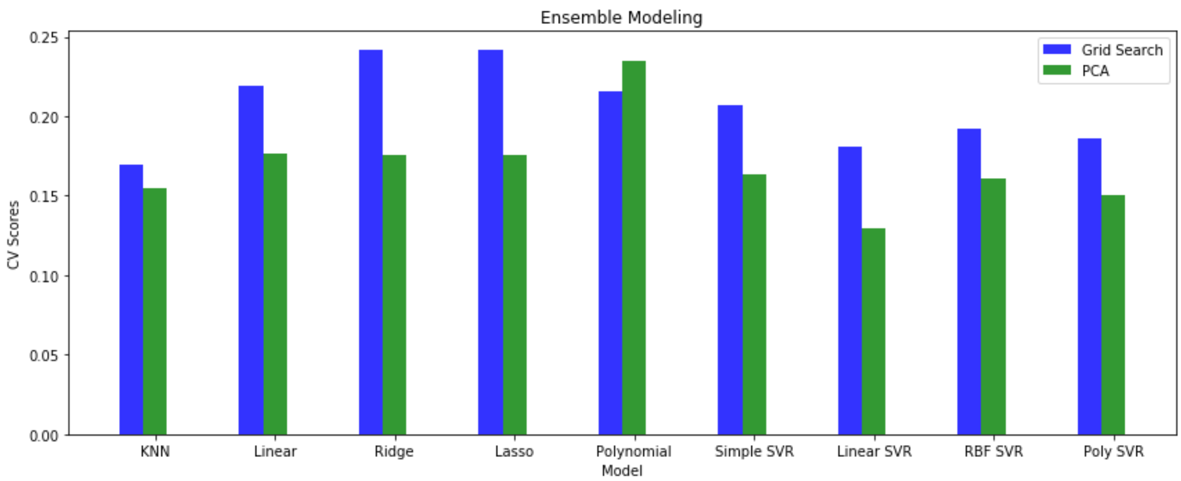
```
<matplotlib.collections.PathCollection at 0xac66ffca0>
```



PCA Models Comparison

In [500]:

```
# create plot
pca_df = model_dataframe.replace(r' Regression', '', regex = True)
pca_df['Model'].replace(' Regression', '')

fig, ax = plot.subplots(figsize=(12,5))
index = npy.arange(9)
bar_width = 0.2
opacity = 0.8

rects1 = plot.bar(index, pca_df['Grid Search'], bar_width, alpha=opacity, color='b', la
rects2 = plot.bar(index + bar_width, pca_df['PCA'], bar_width, alpha=opacity, color='g'

plot.xlabel('Model')
plot.ylabel('CV Scores')
plot.title('Ensemble Modeling')
plot.xticks(index + bar_width, pca_df['Model'])
plot.legend()

plot.tight_layout()
plot.show()
```



In [95]:

```
# print values for non-visual comparison
model_dataframe
```

Out[95]:

	Model	Grid Search	PCA	Bagging
0	KNNRegression	0.176396	0.127418	NaN
1	LinearRegression	0.206333	0.135710	NaN
2	RidgeRegression	0.228955	0.134282	NaN
3	LassoRegression	0.228949	0.134253	NaN
4	PolynomialRegression	0.199035	0.191203	NaN
5	Simple SVRRegression	0.177345	0.107605	0.182435
6	SVR LinearRegression	0.159569	0.088960	0.161030
7	SVR RBFRegression	0.168805	0.127953	NaN
8	SVR PolynomialRegression	0.175448	0.126137	NaN
9	Decision Tree Regression	0.149686	NaN	NaN

PCA Automation : Result

In [96]:

```
# Find indexes with increasing accuracy
inc_count = 0
inc_index = []
for index in range(9):
    if model_df['Grid Search'][index] < model_df['PCA'][index]:
        inc_count += 1
        inc_index.append(index)

# Automate result display
from colorama import Fore, Style
if inc_count > 4:
    print('By comapring with project 1 models, Since PCA increases accuracy for majorit
    print(f"{Fore.GREEN}\033[1m helps in getting better results.{Style.RESET_ALL}")
else:
    print('\033[1mBy comapring with project 1 models, Since PCA DOES NOT increase accur
    print('\033[1mWe can conclude that PCA ',end = '')
    print(f'{Fore.RED}\033[1mis not really helpful in getting better results.{Style.RES
    if inc_count > 0:
        print('\nHowever, PCA helps in getting better results for:')
        for index in inc_index:
            print(f'{Fore.GREEN}\t' + model_df['Model'][index])
            print(Style.RESET_ALL)
```

```
Since PCA DOES NOT increase accuracy for majority of models
We can say, PCA is not helpful getting better results.
```

Task 8: Final Step : Deep Learning

In [94]:

```
#deep learning
deep_learn = Sequential()
deep_learn.add(Dense(9, input_dim=9, kernel_initializer='normal', activation='relu'))
deep_learn.add(Dense(1, kernel_initializer='normal'))
deep_learn.compile(loss='mse', optimizer='sgd' , metrics = ['mse'])
deep_learn.fit(x_trainset, y_trainset, epochs = 100, batch_size = 20)
```

```
Epoch 1/100
61/61 [==============================] - 6s 2ms/step - loss: 75407.1677 -
mse: 75407.1677
Epoch 2/100
61/61 [==============================] - 0s 2ms/step - loss: 36668.3161 -
mse: 36668.3161
Epoch 3/100
61/61 [==============================] - 0s 2ms/step - loss: 36485.5984 -
mse: 36485.5986
Epoch 4/100
61/61 [==============================] - 0s 2ms/step - loss: 33020.5171 -
mse: 33020.5171
Epoch 5/100
61/61 [==============================] - 0s 2ms/step - loss: 34312.0615 -
mse: 34312.0588
Epoch 6/100
61/61 [==============================] - 0s 2ms/step - loss: 36102.1905 -
mse: 36102.1905
Epoch 7/100
```

In [95]:

```
#train and test models
y_pred_train = deep_learn.predict(x_trainset)
y_pred_test = deep_learn.predict(x_testset)

#model scores
print(f'The Train dataset score: ',r2_score(y_trainset,y_pred_train))
print(f'The Test dataset score: ',r2_score(y_testset,y_pred_test))
```

```
The Train dataset score:  -9.925009772060456e-05
The Test dataset score:  -0.008656217820685486
```

End of Project 2 : Regression

Initials:

-rp