# Project Part 2

Project Description:

- Use same datasets as Project 1.
- Preprocess data: Explore data and apply data scaling.

Regression Task:

- Apply any two models with bagging and any two models with pasting.
- Apply any two models with adaboost boosting
- Apply one model with gradient boosting
- Apply PCA on data and then apply all the models in project 1 again on data you get from PCA. Compare your results with results in project 2. You don't need to apply all the models twice. Just copy the result table from project 1, prepare similar table for all the models after PCA and compare both tables. Does PCA help in getting better results?
- Apply deep learning models covered in class

Classification Task:

- Apply two voting classifiers - one with hard voting and one with soft voting
- Apply any two models with bagging and any two models with pasting.
- Apply any two models with adaboost boosting
- Apply one model with gradient boosting
- Apply PCA on data and then apply all the models in project 1 again on data you get from PCA. Compare your results with results in project 1. You don't need to apply all the models twice. Just copy the result table from project 1, prepare similar table for all the models after PCA and compare both tables. Does PCA help in getting better results?
- Apply deep learning models covered in class

## Dataset: [Airbnb Listings (https://www.kaggle.com/rudymizrahi/airbnb-listings-in-major-us-cities-deloitte-ml#train.csv)](https://www.kaggle.com/rudymizrahi/airbnb-listings-in-major-us-cities-deloitte-ml#train.csv)

# Project Description and Requirements:

## Dataset requirements:

For this projects in this class, you will pick your datasets. The datasets should satisfy the following conditions:

- At least 15 features (columns)
- At least 1000 instances (rows)
- 
  - Shape of train data:
- 
  - Shape of test data:
- At least two categorical/ordinal columns.
- Between 5 to 10 percent missing values across the dataset.

#### Importing the libraries

In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
import seaborn as sns
import sklearn
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import scale
```

In [2]:
```
%matplotlib inline
```

# Project Description:

Read data into Jupyter notebook, use pandas to import data into a data frame Preprocess data: Explore data, check for missing data and apply data scaling. Justify the type of scaling used.

# The first part is "Preprocess data" and same as the first project.

Loading the dataset

In [3]:
```
df = pd.read_csv("train.csv")
```

In [4]:
```
print('Shape of train data: ', df.shape)
```

```
Shape of train data:  (74111, 29)
```

29 features and 74111 instances are avalable in this data set

In [5]:
```
df.head()
```

Out[5]:

| | id | log_price | property_type | room_type | amenities | accommodates | bathroon |
|---|---|---|---|---|---|---|---|
| 0 | 6901257 | 5.010635 | Apartment | Entire home/apt | {"Wireless Internet","Air conditioning",Kitche... | 3 | 1 |
| 1 | 6304928 | 5.129899 | Apartment | Entire home/apt | {"Wireless Internet","Air conditioning",Kitche... | 7 | 1 |
| 2 | 7919400 | 4.976734 | Apartment | Entire home/apt | {TV,"Cable TV","Wireless Internet","Air condit... | 5 | 1 |
| 3 | 13418779 | 6.620073 | House | Entire home/apt | {TV,"Cable TV",Internet,"Wireless Internet",Ki... | 4 | 1 |
| 4 | 3808709 | 4.744932 | Apartment | Entire home/apt | {TV,Internet,"Wireless Internet","Air conditio... | 2 | 1 |

5 rows × 29 columns

Few changes in df...

In [6]:
```
df['host_response_rate']= pd.to_numeric(df['host_response_rate'].str.strip('%'
))
df['room_type']= df['room_type'].map({'Entire home/apt':'Entire home/apt','Pri
vate room':'Private room','Shared room':'Private room'})
```

# Dataset Description and goal

This dataset has many features that are important in the Airbnb price of a house or Apt or ... and might affect its Price. The aim of this competition was to predict the price of AirBnB listings in major U.S. cities.

# Data Description

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74111 entries, 0 to 74110
Data columns (total 29 columns):
id                          74111 non-null int64
log_price                   74111 non-null float64
property_type               74111 non-null object
room_type                   74111 non-null object
amenities                   74111 non-null object
accommodates                74111 non-null int64
bathrooms                   73911 non-null float64
bed_type                    74111 non-null object
cancellation_policy         74111 non-null object
cleaning_fee                74111 non-null bool
city                        74111 non-null object
description                 74111 non-null object
first_review                58247 non-null object
host_has_profile_pic        73923 non-null object
host_identity_verified      73923 non-null object
host_response_rate          55812 non-null float64
host_since                  73923 non-null object
instant_bookable            74111 non-null object
last_review                 58284 non-null object
latitude                    74111 non-null float64
longitude                   74111 non-null float64
name                        74111 non-null object
neighbourhood               67239 non-null object
number_of_reviews           74111 non-null int64
review_scores_rating        57389 non-null float64
thumbnail_url               65895 non-null object
zipcode                     73145 non-null object
bedrooms                    74020 non-null float64
beds                        73980 non-null float64
dtypes: bool(1), float64(8), int64(3), object(17)
memory usage: 15.9+ MB
```

In [8]:
```
df.isnull().sum()
# number of nulls
```

Out[8]:
```
id                            0
log_price                     0
property_type                 0
room_type                     0
amenities                     0
accommodates                  0
bathrooms                   200
bed_type                      0
cancellation_policy           0
cleaning_fee                  0
city                          0
description                   0
first_review              15864
host_has_profile_pic        188
host_identity_verified      188
host_response_rate        18299
host_since                  188
instant_bookable              0
last_review               15827
latitude                      0
longitude                     0
name                          0
neighbourhood              6872
number_of_reviews             0
review_scores_rating      16722
thumbnail_url              8216
zipcode                     966
bedrooms                     91
beds                        131
dtype: int64
```

In [9]:
```
print('missing values across the dataset % {:.2f}'.format(df.isnull().sum().su
m()/(len(df)*29)*100))
```

```
missing values across the dataset % 3.90
```

The number of nulls are didn't meet the requirments, we have to generate them.

In [10]:
```
df.columns
```

Out[10]:
```
Index(['id', 'log_price', 'property_type', 'room_type', 'amenities',
       'accommodates', 'bathrooms', 'bed_type', 'cancellation_policy',
       'cleaning_fee', 'city', 'description', 'first_review',
       'host_has_profile_pic', 'host_identity_verified', 'host_response_rat
e',
       'host_since', 'instant_bookable', 'last_review', 'latitude',
       'longitude', 'name', 'neighbourhood', 'number_of_reviews',
       'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'bed
s'],
      dtype='object')
```

### ***Target variable***

- SalePrice: the property's sale price in dollars. This is the target variable that we're trying to predict.

### ***Features***

- id: The Property id
- log price: the log price of the property for the night in dollar
- property type: Apartment or House
  - Apartment = 1 and House = 0 **(since the apartment has more instanses)**
- room_type: Entire or Private
  - Entire = 1 and Private = 0
- amenities: list of all amenities
  - because all of them are unique the column will be dropped
- accommodates: Number of guests that a house can accommodate
- bathroom: Number of bathrooms
- bed_type: Airbed, couch, Futon, Pull out sofa, Real bed
  - there is no difference between them so we will use one hot-vector
- cancellation_policy: strict, moderate, flexible
- cleaning fee: True, they have cleaning fee False otherwise
  - we will use True = 1 and False = 0 **(since we have more instances with TRUE values)**
- city: NYC is a new york city and LA is Los Angeles, Boston, Chicago, DC, SF
  - we will use one-hot vector for this feature
- description: Description of the house
  - we will drop this column since all values are unique
- first_review: the date of the first review
  - we have to see the correlation matrix but, even with high numbers in there it is still not believable to find a relation between the date and price, So I'll drop this feature later.
- host_has_profile_pic: True means it has, False otherwise
  - we will use True = 1 and False = 0 **(since we have more instanses with TRUE vlaues)**
- host_identity_verified: True means it has, False otherwise
  - we will use True = 1 and False = 0 **(since we have more instanses with TRUE vlaues)**
- host_response_rate: % of respond rate, between 0 and 1
- host_since: the date of the host account
  - we have to see the correlation matrix but, even with high numbers in there it is still not believable to find a relation between the date and price, So I'll drop this feature later.
- instant_bookable: True means it has, False otherwise
  - we will use True = 1 and False = 0 **(since we have more instanses with TRUE vlaues)**
- last_review: the date of the last review
  - we have to see the correlation matrix but, even with high numbers in there it is still not believable to find a relation between the date and price, So I'll drop this feature later.
- latitude: related to the location of the house, great for the visualization but not useful for regression, however, the location can be scored base on the location, and neighborhood but for simplification, we will ignore this part
- longitude: related to the location of the house, great for the visualization but not useful for regression, however, the location can be scored base on the location and neighborhood but for simplification, we will ignore this part

- name: All unique values for the place
  - since all values are unique we will drop this feature
- neighborhood: related to the location of the house, great for the visualization but not useful for regression, however, the location can be scored base on the location, and neighborhood but for simplification, we will ignore this part
- number_of_reviews: number of views
- review_scores_rating: review score between 0 - 100
- thumbnail_url:
  - not related to the regression
- zipcode: related to the location of the house, great for the visualization but not useful for regression, however, the location can be scored base on the location, and neighborhood but for simplification, we will ignore this part
- bedrooms: number of bedrooms a house has
- beds: number of beds a house have

In [11]:
```
df.drop(columns= ['id', 'amenities', 'description', 'first_review',
       'host_since','last_review', 'latitude',
       'longitude', 'name', 'neighbourhood', 'thumbnail_url', 'zipcode'], inpl
ace= True)
```

so we droped the features with mostly uniuqe instances so the we can draw smaller correlation matrix.

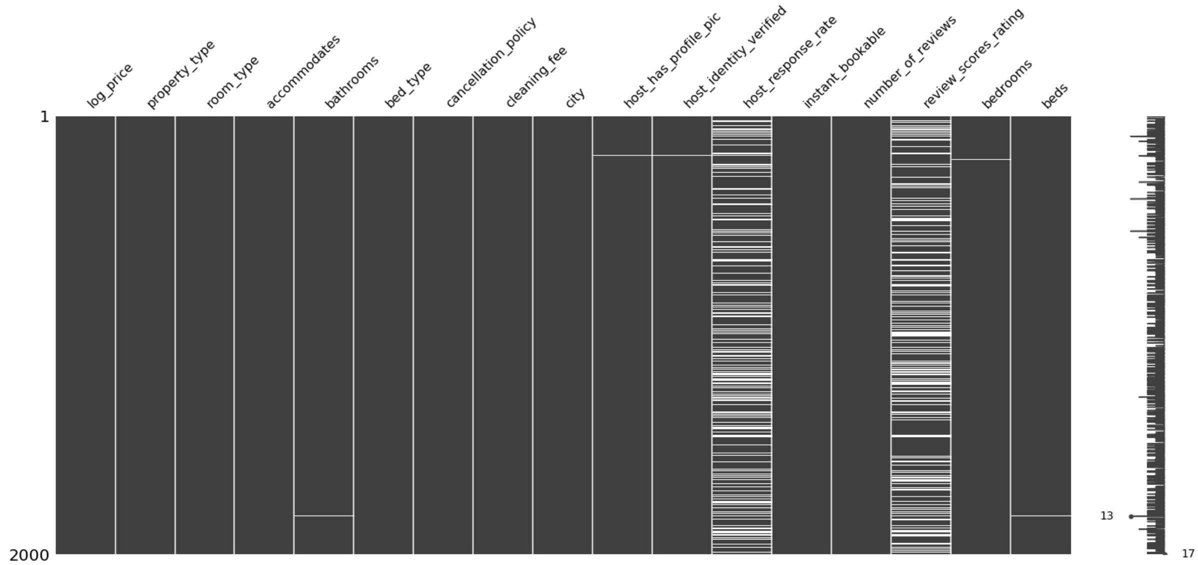
the data set has around 75,000 instanses and it will too long in the ML models to run each model. I will use only 2000 of them.

In [12]:
```
df=df[0:2000]
```

## Visualizing the missing portion of the dataset

In [13]:
```
# !pip install missingno
import missingno as msno
msno.matrix(df)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1a9c2f4ffc8>




In [14]:
```
print('missing values across the dataset % {:.2f}'.format(df.isnull().sum().su
m()/(len(df)*17)*100))
```

```
missing values across the dataset % 3.12
```

So we have to generate missed values so we are going to generate them. now they are only in two columns.

## Creating missing data

We will create a mask here that randomly will null out values in our dataset , we will pass p to the mask, which the first element, will determine the % of the missing data we want.

In [15]:
```
df = df.mask(np.random.choice([True, False], size=df.shape, p=[.02,.98]))
```

In [16]:
```
all_data_na = (df.isnull().sum() / len(df)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_value
s(ascending=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data
```

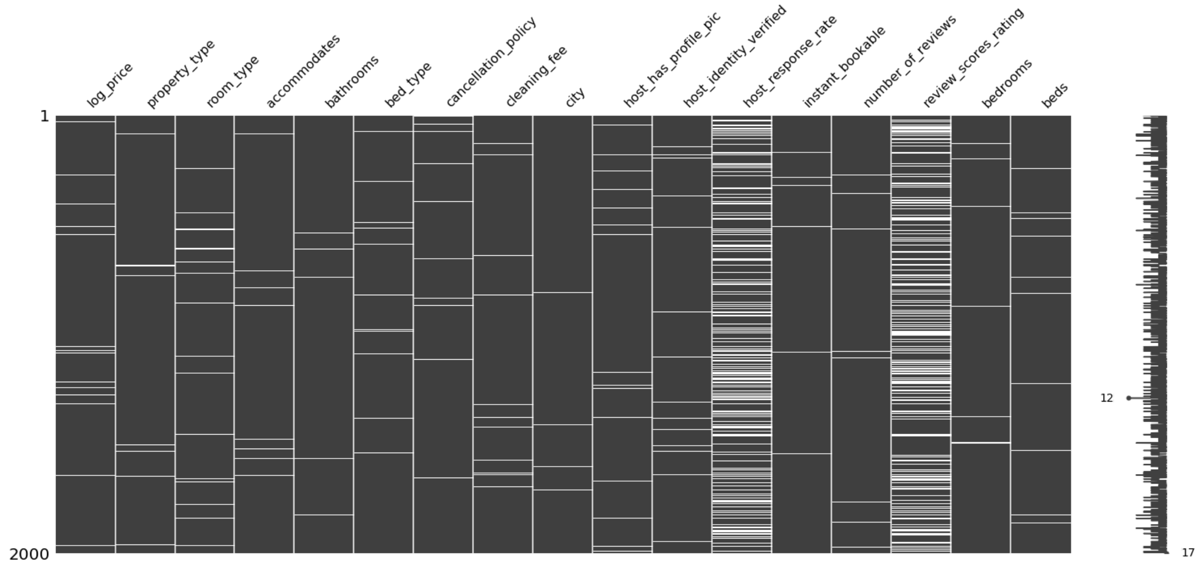
Out[16]:

| | Missing Ratio |
|---|---|
| host_response_rate | 27.10 |
| review_scores_rating | 27.00 |
| room_type | 2.50 |
| beds | 2.40 |
| bedrooms | 2.35 |
| cleaning_fee | 2.30 |
| host_has_profile_pic | 2.30 |
| bed_type | 2.15 |
| log_price | 2.15 |
| number_of_reviews | 2.15 |
| host_identity_verified | 2.05 |
| cancellation_policy | 1.85 |
| bathrooms | 1.85 |
| property_type | 1.80 |
| accommodates | 1.75 |
| instant_bookable | 1.65 |
| city | 1.10 |

visualizing the dataset after nulling out values at random

In [17]:
```
import missingno as msno
msno.matrix(df)
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1a9c2e1bf08>




In [18]:
```
print('missing values across the dataset % {:.2f}'.format(df.isnull().sum().su
m()/(len(df)/17)))
```

```
missing values across the dataset % 14.36
```

Now it is okay.

In [19]:
```
df.columns
```

Out[19]:
```
Index(['log_price', 'property_type', 'room_type', 'accommodates', 'bathroom
s',
       'bed_type', 'cancellation_policy', 'cleaning_fee', 'city',
       'host_has_profile_pic', 'host_identity_verified', 'host_response_rat
e',
       'instant_bookable', 'number_of_reviews', 'review_scores_rating',
       'bedrooms', 'beds'],
      dtype='object')
```

## Dropping and imputing missing data

So we have two general course of action regarding the missing data, either dropping them or replacing them with the mean or the median. Here in this case, most of our variables are catagorical and we will be dropping those because random assignment doesnt make sense.on other features we are going to use mean for normal distribution and median for skewed ones.

**Dropping rows:**

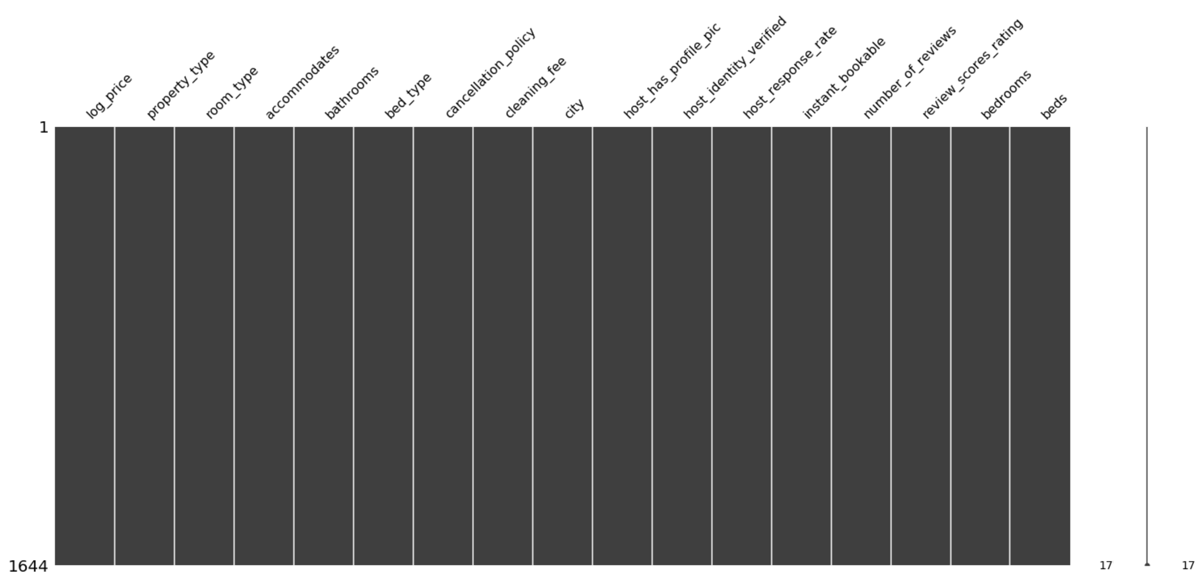
At first you might think it is best to impute the missing data however the features with missing data that remain are mostly categorical and mode isnt a correct representation of the missing values, so I've decided to drop the rows in order notto misrepresent the data. Also we are going to drop instances in target column with missing values\

In [20]:
```
rows_drop=['log_price', 'property_type', 'room_type',
       'bed_type', 'cancellation_policy', 'cleaning_fee', 'city',
       'host_has_profile_pic', 'host_identity_verified',
       'instant_bookable']
for i in rows_drop:
    df=df[df[i].notnull()]
```

In [21]:
```
df['accommodates'].fillna(df['accommodates'].median(),inplace=True)
df['bathrooms'].fillna(df[ 'bathrooms'].median(),inplace=True)
df['host_response_rate'].fillna(df['host_response_rate'].mean(),inplace=True)
df['number_of_reviews'].fillna(round(df['number_of_reviews'].mean(),0),inplace
=True)
df['review_scores_rating'].fillna(round(df[ 'review_scores_rating'].mean()),in
place=True)
df['bedrooms'].fillna(df['bedrooms'].median(),inplace=True)
df['beds'].fillna(df['beds'].median(),inplace=True)
```

In [22]:
```
msno.matrix(df)
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1a9c2f0ab48>




In [23]:
```
df.shape
# row and columns after cleaning
```

Out[23]: (1644, 17)

## Datatypes and Transformations

In [24]:
```
df.dtypes
```

Out[24]:
```
log_price                  float64
property_type               object
room_type                   object
accommodates               float64
bathrooms                  float64
bed_type                    object
cancellation_policy         object
cleaning_fee               float64
city                        object
host_has_profile_pic        object
host_identity_verified      object
host_response_rate          float64
instant_bookable            object
number_of_reviews          float64
review_scores_rating       float64
bedrooms                   float64
beds                       float64
dtype: object
```

In [25]:
```
df.bed_type.unique()
```

Out[25]:
```
array(['Real Bed', 'Futon', 'Pull-out Sofa', 'Couch', 'Airbed'],
      dtype=object)
```

In [26]:
```
df.head(10)
```

Out[26]:

| | log_price | property_type | room_type | accommodates | bathrooms | bed_type | cancellation_poli |
|---|---|---|---|---|---|---|---|
| 1 | 5.129899 | Apartment | Entire home/apt | 7.0 | 1.0 | Real Bed | str |
| 2 | 4.976734 | Apartment | Entire home/apt | 5.0 | 1.0 | Real Bed | modera |
| 3 | 6.620073 | House | Entire home/apt | 4.0 | 1.0 | Real Bed | flexib |
| 4 | 4.744932 | Apartment | Entire home/apt | 2.0 | 1.0 | Real Bed | modera |
| 5 | 4.442651 | Apartment | Private room | 2.0 | 1.0 | Real Bed | str |
| 7 | 4.787492 | Condominium | Entire home/apt | 2.0 | 1.0 | Real Bed | modera |
| 8 | 4.787492 | House | Private room | 2.0 | 1.0 | Real Bed | modera |
| 9 | 3.583519 | House | Private room | 2.0 | 1.0 | Real Bed | modera |
| 10 | 4.605170 | Apartment | Private room | 2.0 | 1.0 | Real Bed | str |
| 11 | 5.010635 | House | Entire home/apt | 4.0 | 1.5 | Real Bed | str |

In [27]:
```
df['room_type'] = df['room_type'].map({'Entire home/apt':1,'Private room':0}).
astype(int)
df['host_has_profile_pic'] = df['host_has_profile_pic'].map({'t':1,'f':0}).ast
ype(int)
df['host_identity_verified'] = df['host_identity_verified'].map({'t':1,'f':0})
.astype(int)
df['instant_bookable'] = df['instant_bookable'].map({'t':1,'f':0}).astype(int)
```

Now one hot-vectors

In [28]:
```
emb=pd.get_dummies(df['property_type'],columns='property_type',prefix='propert
y_type ')
df=pd.concat([df, emb], axis=1)
df.drop(['property_type'],axis=1,inplace= True)
```

In [29]:
```
emb=pd.get_dummies(df['bed_type'],columns='bed_type',prefix='bed_type ')
df=pd.concat([df, emb], axis=1)
df.drop(['bed_type'],axis=1,inplace= True)
```

In [30]:
```
emb=pd.get_dummies(df['cancellation_policy'],columns='cancellation_policy',pre
fix='cancellation_policy ')
df=pd.concat([df, emb], axis=1)
df.drop(['cancellation_policy'],axis=1,inplace= True)
```

In [31]:
```
emb=pd.get_dummies(df['city'],columns='city',prefix='city ')
df=pd.concat([df, emb], axis=1)
df.drop(['city'],axis=1,inplace= True)
```

In [32]:
```
df.head()
```

Out[32]:

| | log_price | room_type | accommodates | bathrooms | cleaning_fee | host_has_profile_pic | host_ide |
|---|---|---|---|---|---|---|---|
| 1 | 5.129899 | 1 | 7.0 | 1.0 | 1.0 | 1 | |
| 2 | 4.976734 | 1 | 5.0 | 1.0 | 1.0 | 1 | |
| 3 | 6.620073 | 1 | 4.0 | 1.0 | 1.0 | 1 | |
| 4 | 4.744932 | 1 | 2.0 | 1.0 | 1.0 | 1 | |
| 5 | 4.442651 | 0 | 2.0 | 1.0 | 1.0 | 1 | |

5 rows × 48 columns

In [33]:
```
df.describe()
```

Out[33]:

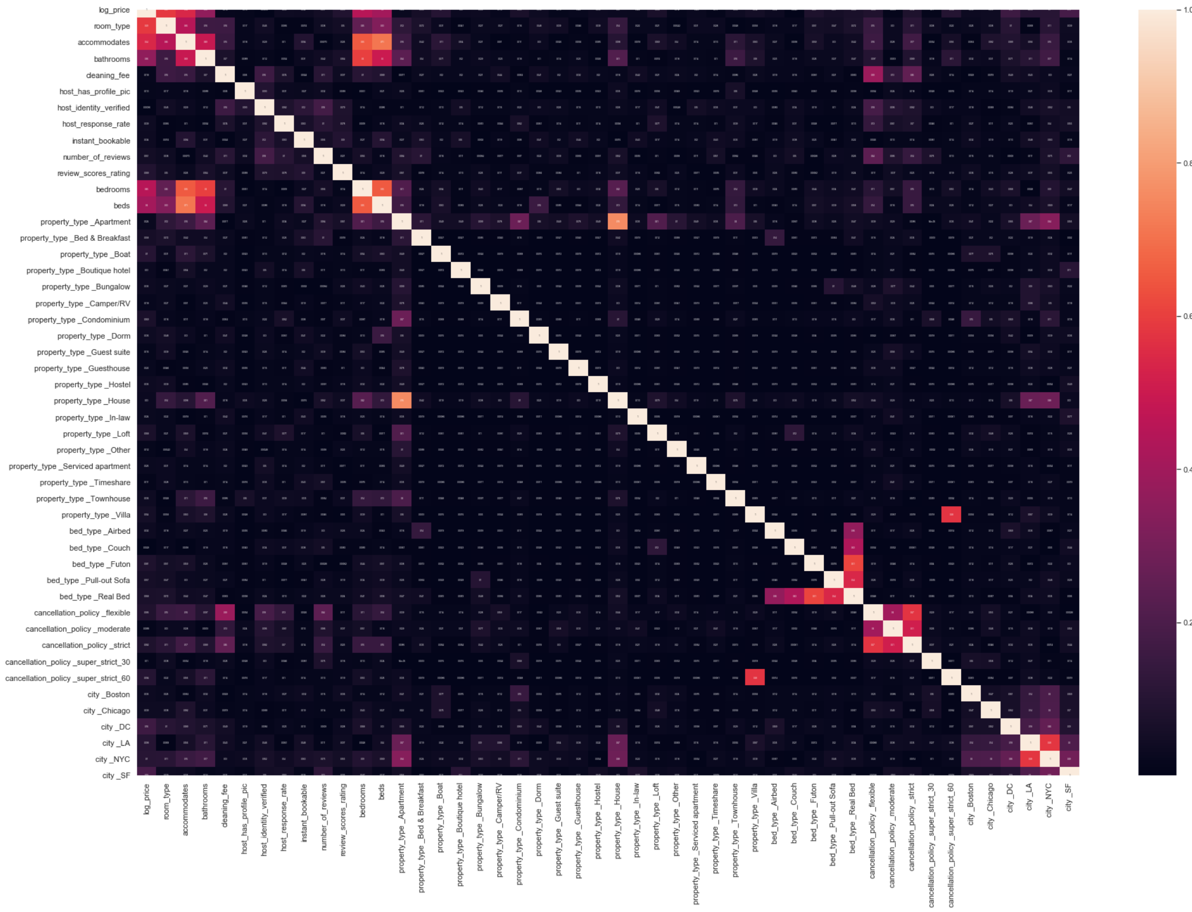
| | log_price | room_type | accommodates | bathrooms | cleaning_fee | host_has_profile_pic |
|---|---|---|---|---|---|---|
| count | 1644.000000 | 1644.000000 | 1644.000000 | 1644.000000 | 1644.000000 | 1644.000000 |
| mean | 4.810536 | 0.558394 | 3.159976 | 1.211375 | 0.747567 | 0.995742 |
| std | 0.721868 | 0.496730 | 2.155959 | 0.511759 | 0.434541 | 0.065133 |
| min | 2.890372 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 4.317488 | 0.000000 | 2.000000 | 1.000000 | 0.000000 | 1.000000 |
| 50% | 4.779123 | 1.000000 | 2.000000 | 1.000000 | 1.000000 | 1.000000 |
| 75% | 5.252233 | 1.000000 | 4.000000 | 1.000000 | 1.000000 | 1.000000 |
| max | 7.569412 | 1.000000 | 16.000000 | 5.500000 | 1.000000 | 1.000000 |

8 rows × 48 columns

# Correlation and highly correlated features

In [34]:
```
# Correlation of all the variables with respect to each other
sns.set(font_scale=1.8)
sns.set(rc={'figure.figsize':(30,20)})
sns.heatmap(df.corr().abs(),annot=True, annot_kws={"size": 3})
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1a9c36103c8>

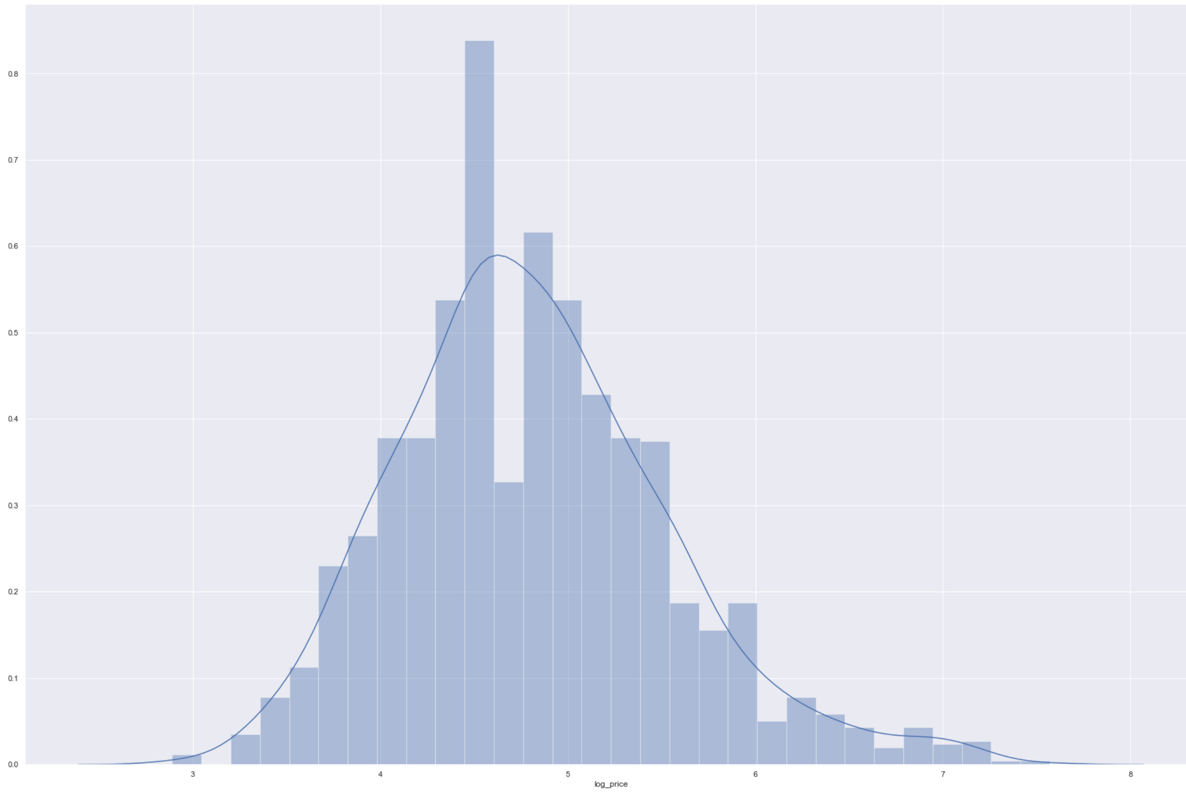



As it shows most of the variables don't have a correlation to each other more than 70% so there is no need to be worried about collinearity. on the other hand, log_price has a relation with bedrooms, beds, room_type, and accommodates.

## Scaling and transformation

In [35]:
```
#histogram of target variable :
sns.distplot(df['log_price']);

print(df.log_price.var())
```

0.521092971293666




since the data already logged the histogram looks just fine

## Dropping index + target features

the data set has around 40,000 instanses and it will too long in the ML models to run each model. I will use only 10,000 of them.

In [36]:
```
df_org = df
y_org = df['log_price']
X_org = df.drop(columns=['log_price'],axis=1)
```

`python MinMaxScaler(feature_range = (0, 1))` will transform each value in the column proportionally within the range [0,1]. Use this as the first scaler choice to transform a feature, as it will preserve the shape of the dataset (no distortion).

`python StandardScaler()` will transform each value in the column to range about the mean 0 and standard deviation 1, ie, each value will be normalised by subtracting the mean and dividing by standard deviation. Use StandardScaler if you know the data distribution is normal.

If there are outliers, use `pyhton RobustScaler()` . Alternatively you could remove the outliers and use either of the above 2 scalers (choice depends on whether data is normally distributed)

based on the explanation above I will use the MinMaxScaler. However, our data are mostly less than 10 and exept for one or two featuers there is no need for scaleing

In [37]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

X_train_org, X_test_org, y_train, y_test = train_test_split(X_org,y_org, test_
size=0.25, random_state=0)
```

In [38]:
```
X_train_org.describe()
```

Out[38]:

| | room_type | accommodates | bathrooms | cleaning_fee | host_has_profile_pic | host_identity |
|---|---|---|---|---|---|---|
| count | 1233.000000 | 1233.000000 | 1233.000000 | 1233.000000 | 1233.000000 | 123 |
| mean | 0.562855 | 3.152474 | 1.196269 | 0.747770 | 0.995945 | |
| std | 0.496235 | 2.143272 | 0.492643 | 0.434469 | 0.063577 | |
| min | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| 25% | 0.000000 | 2.000000 | 1.000000 | 0.000000 | 1.000000 | |
| 50% | 1.000000 | 2.000000 | 1.000000 | 1.000000 | 1.000000 | |
| 75% | 1.000000 | 4.000000 | 1.000000 | 1.000000 | 1.000000 | |
| max | 1.000000 | 16.000000 | 5.500000 | 1.000000 | 1.000000 | |

8 rows × 47 columns

In [39]:
```
scaler = MinMaxScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train_org),columns=X_train_org.c
olumns)
X_test = pd.DataFrame(scaler.transform(X_test_org),columns=X_test_org.columns)

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(1233, 47) (1233,) (411, 47) (411,)
```

In [40]:
```
X_train.describe()
```

Out[40]:

| | room_type | accommodates | bathrooms | cleaning_fee | host_has_profile_pic | host_identity |
|---|---|---|---|---|---|---|
| count | 1233.000000 | 1233.000000 | 1233.000000 | 1233.000000 | 1233.000000 | 123 |
| mean | 0.562855 | 0.143498 | 0.217504 | 0.747770 | 0.995945 | |
| std | 0.496235 | 0.142885 | 0.089571 | 0.434469 | 0.063577 | |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| 25% | 0.000000 | 0.066667 | 0.181818 | 0.000000 | 1.000000 | |
| 50% | 1.000000 | 0.066667 | 0.181818 | 1.000000 | 1.000000 | |
| 75% | 1.000000 | 0.200000 | 0.181818 | 1.000000 | 1.000000 | |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | |

8 rows × 47 columns

## Regression Task:

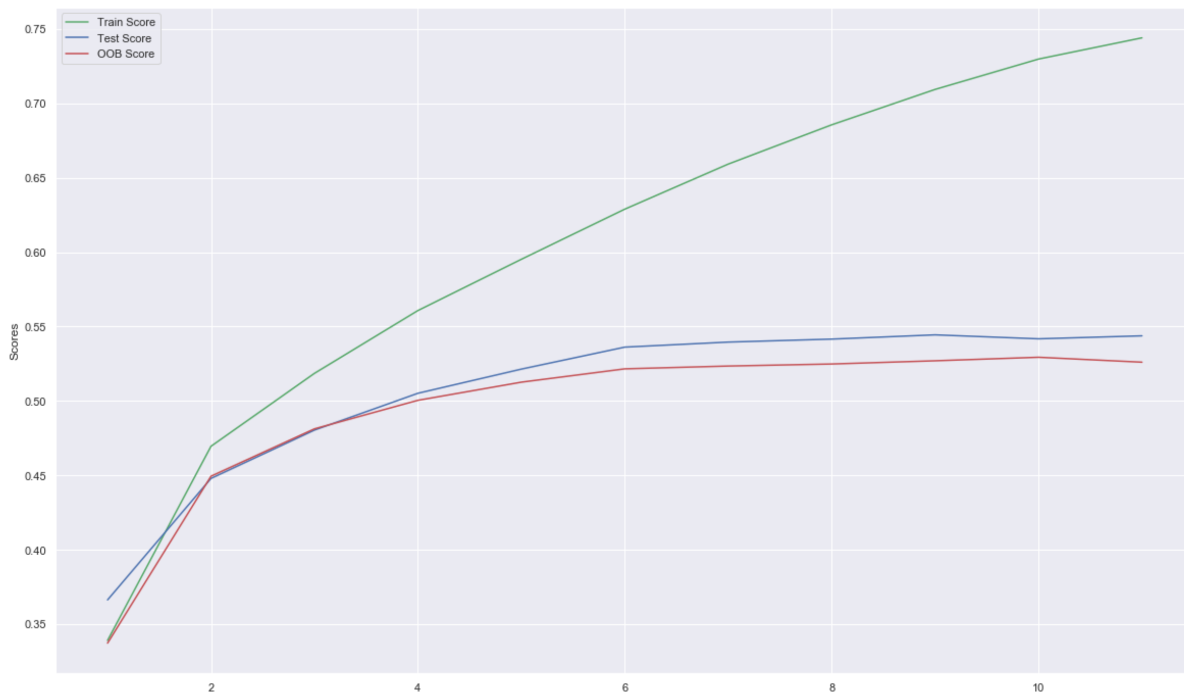
- Apply any two models with bagging and any two models with pasting.
- Apply any two models with AdaBoost boosting.
- Apply one model with gradient boosting.
- Apply PCA on data and then apply all the models in project 1 again on data you get from PCA. Compare your results with results in project 1. You don't need to apply all the models twice. Just copy the result table from project 1, prepare a similar table for all the models after PCA and compare both tables. Does PCA help in getting better results?
- Apply deep learning models covered in class

## Bagging

#### Decision tree with bagging

In [41]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
sns.set(rc={'figure.figsize':(20,12)})


train_score_array = []
test_score_array = []
oob_score_array=[]

for n in range(1,12):
    dt_reg = DecisionTreeRegressor(max_depth=n,random_state=0)
    bag_reg_dt = BaggingRegressor(dt_reg, n_estimators=50, max_samples=500, bo
otstrap=True,random_state=0,oob_score=True)
    bag_reg_dt.fit(X_train, y_train)
    train_score_array.append(bag_reg_dt.score(X_train, y_train))
    test_score_array.append(bag_reg_dt.score(X_test, y_test))
    oob_score_array.append(bag_reg_dt.oob_score_)

x_axis = range(1,12)
plt.plot(x_axis, train_score_array, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_array, c = 'b', label = 'Test Score')
plt.plot(x_axis, oob_score_array, c = 'r', label = 'OOB Score')
plt.legend()
plt.xlabel('Max Depth')
plt.ylabel('Scores')
```

Out[41]:
```
Text(0, 0.5, 'Scores')
```




# Model 1

### Random Forest

In [42]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
rf_reg = RandomForestRegressor(n_estimators=100,random_state=0)
bag_reg_rf = BaggingRegressor(rf_reg, n_estimators=100, max_samples=500, boots
trap=True,random_state=0,oob_score=True)
bag_reg_rf.fit(X_train, y_train)

y_pred=bag_reg_rf.predict(X_test)

print('Train score: {:.4f} %'.format(bag_reg_rf.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(bag_reg_rf.score(X_test, y_test)*100))

bag_reg_rf_trainscore = bag_reg_rf.score(X_train, y_train)*100
bag_reg_rf_testscore = bag_reg_rf.score(X_test, y_test)*100

print('MAE: {:.4f}'.format(mean_absolute_error(y_test,y_pred)))
print('MSE: {:.4f}'.format(mean_squared_error(y_test,y_pred)))
print('RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
Train score: 70.6632 %
Test score: 55.2276 %
MAE: 0.3898
MSE: 0.2664
RMSE: 0.5161
```

SVM with RBF kernel

In [43]:
```
from sklearn.svm import SVR, LinearSVR
from sklearn.ensemble import BaggingRegressor
svm_reg =  SVR(kernel='rbf',C=10,gamma=0.01)
bag_reg_rf = BaggingRegressor(svm_reg, n_estimators=100, max_samples=500, boot
strap=True,random_state=0,oob_score=True)
bag_reg_rf.fit(X_train, y_train)

y_pred=bag_reg_rf.predict(X_test)

print('Train score: {:.4f} %'.format(bag_reg_rf.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(bag_reg_rf.score(X_test, y_test)*100))

print('MAE: {:.4f}'.format(mean_absolute_error(y_test,y_pred)))
print('MSE: {:.4f}'.format(mean_squared_error(y_test,y_pred)))
print('RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
Train score: 54.0936 %
Test score: 52.4472 %
MAE: 0.3932
MSE: 0.2829
RMSE: 0.5319
```

| Bagging Regressor Scores | Random Forest | SVM with RBF kernel |
|---|---|---|
| Train | 70.6632 | 54.0936 |
| Test | 55.2276 | 52.4472 |

For the SVM (kernel=rbf) model and the Random forest regressor both test scores are shown above.

# Pasting

## Model 1

### SVR with linear kernel

In [44]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.svm import SVR
svm_reg=SVR(C=100,gamma=0.1, kernel='linear')

bag_reg_svm = BaggingRegressor(svm_reg, n_estimators=100, bootstrap=False,rand
om_state=0)
bag_reg_svm.fit(X_train, y_train)

y_pred=bag_reg_svm.predict(X_test)

print('Train score: {:.4f} %'.format(bag_reg_svm.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(bag_reg_svm.score(X_test, y_test)*100))

print('MAE: {:.4f}'.format(mean_absolute_error(y_test,y_pred)))
print('MSE: {:.4f}'.format(mean_squared_error(y_test,y_pred)))
print('RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(y_test,y_pred))))
```

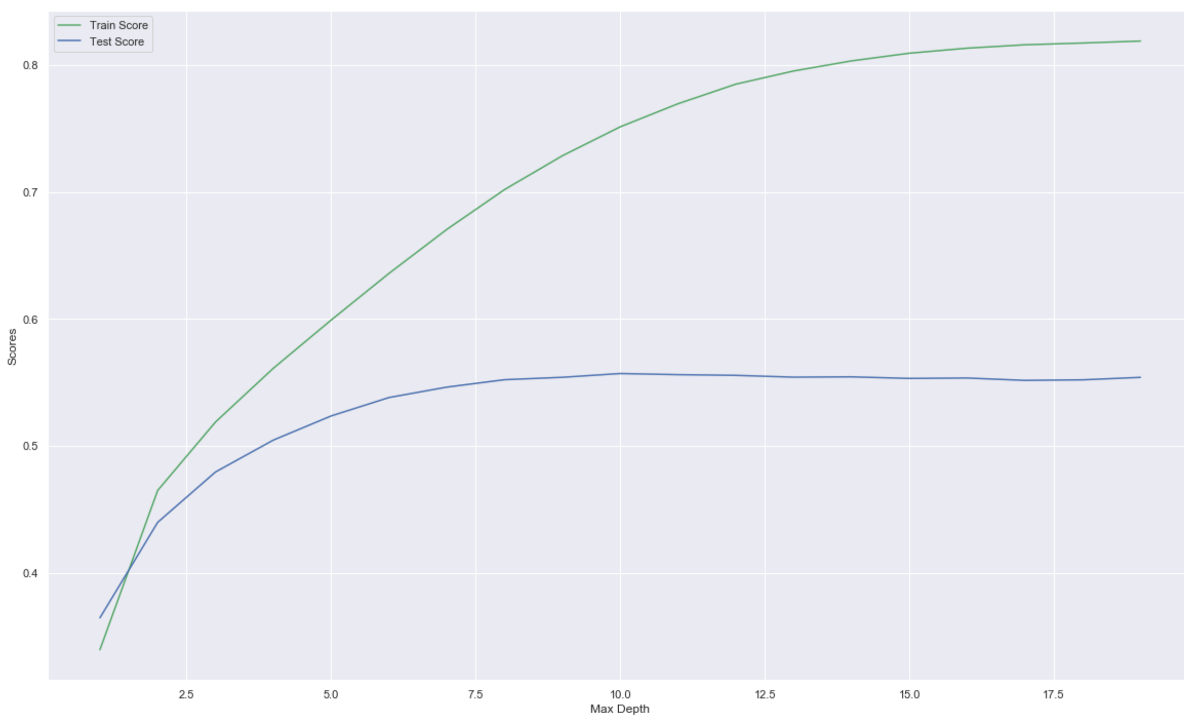
```
Train score: 54.5026 %
Test score: 51.4413 %
MAE: 0.3966
MSE: 0.2889
RMSE: 0.5375
```

## Model 2

### Decision Tree

In [45]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor

sns.set(rc={'figure.figsize':(20,12)})


train_score_array = []
test_score_array = []

for n in range(1,20):
    dt_reg = DecisionTreeRegressor(max_depth=n,random_state=0)
    bag_reg_dt = BaggingRegressor(dt_reg, n_estimators=500, max_samples=500, b
ootstrap=False,random_state=0)
    bag_reg_dt.fit(X_train, y_train)
    train_score_array.append(bag_reg_dt.score(X_train, y_train))
    test_score_array.append(bag_reg_dt.score(X_test, y_test))

x_axis = range(1,20)
plt.plot(x_axis, train_score_array, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_array, c = 'b', label = 'Test Score')
plt.legend()
plt.xlabel('Max Depth')
plt.ylabel('Scores')
```

Out[45]:
```
Text(0, 0.5, 'Scores')
```




AS we can see the Max_depth=5 will give us the least amount of over fitting and the est train score however the scores is still 55% so changing 3 to 5 will give us the least overfiting.

In [46]:
```
dt_reg = DecisionTreeRegressor(max_depth=5,random_state=0)
bag_reg_dt = BaggingRegressor(dt_reg, n_estimators=500, max_samples=500, boots
trap=False,random_state=0)
bag_reg_dt.fit(X_train, y_train)

y_pred=bag_reg_dt.predict(X_test)

print('Train score: {:.4f} %'.format(bag_reg_dt.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(bag_reg_dt.score(X_test, y_test)*100))

print('MAE: {:.4f}'.format(mean_absolute_error(y_test,y_pred)))
print('MSE: {:.4f}'.format(mean_squared_error(y_test,y_pred)))
print('RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
Train score: 59.9205 %
Test score: 52.3648 %
MAE: 0.3975
MSE: 0.2834
RMSE: 0.5324
```

| Pasting Regressor Scores | SVR linear | decision tree |
|---|---|---|
| Train | 54.5026 | 59.9205 |
| Test | 51.4413 | 52.3648 |

Using pasting on SVR linear will give us 50% test score while lower training score but for Decsion tree we have test score is 52%.

# AdaBoost

## Model 1

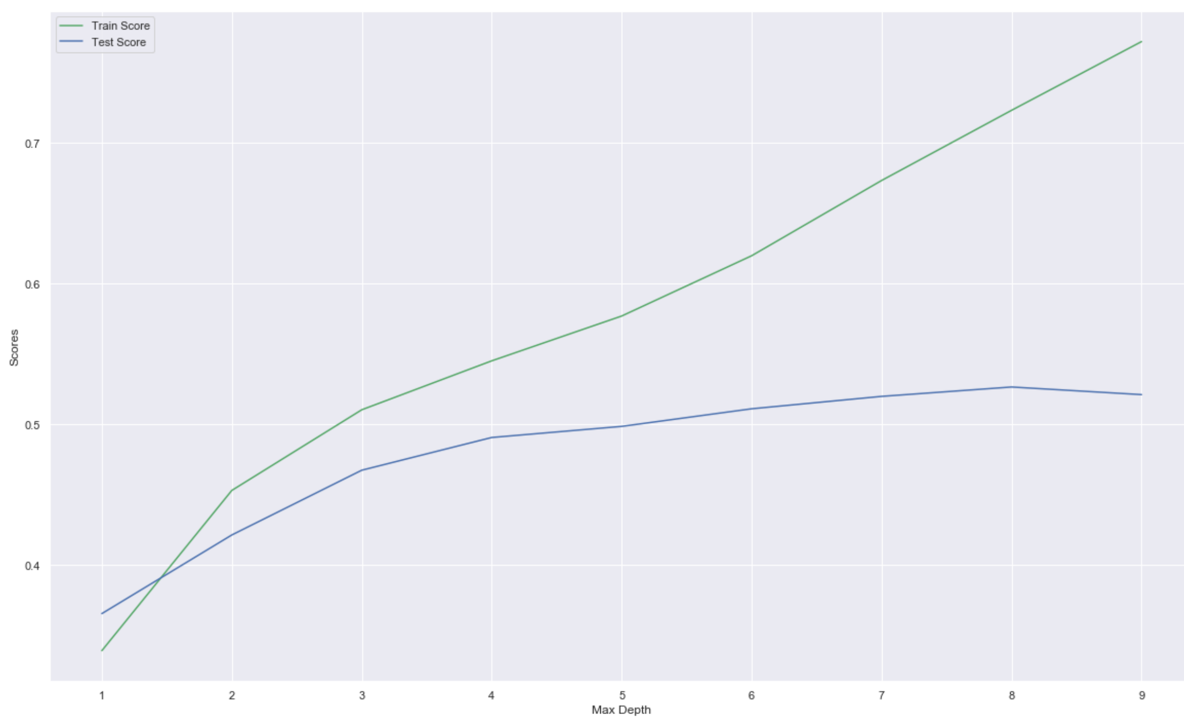
### Decisiontree

In [47]:
```
from sklearn.ensemble import AdaBoostRegressor

train_score_array = []
test_score_array = []
best_score=0


for n in range(1,10):
    for learning_rate in [0.001,0.01,0.1,1,10,100]:
        for n_estimators in [50,100,150,200,250,500]:
            dtree_reg=DecisionTreeRegressor(max_depth=n)
            ada_reg_dtree = AdaBoostRegressor(dtree_reg, n_estimators=n_estima
tors,learning_rate=learning_rate,random_state=0)
            ada_reg_dtree.fit(X_train, y_train)
            train_score_array.append(ada_reg_dtree.score(X_train, y_train))
            test_score_array.append(ada_reg_dtree.score(X_test, y_test))
            score=ada_reg_dtree.score(X_test, y_test)
            if(score>best_score):
                best_score=score
                best_parameters = {'learning_rate': learning_rate, 'max_depth'
: n,'n_estimators':n_estimators}

print(best_parameters)
```

```
{'learning_rate': 0.1, 'max_depth': 8, 'n_estimators': 100}
```

In [48]:
```
from sklearn.ensemble import AdaBoostRegressor
train_score_array = []
test_score_array = []

for n in range(1,10):
    dtree_reg=DecisionTreeRegressor(max_depth=n)
    ada_reg_dtree = AdaBoostRegressor(dtree_reg, n_estimators=50,learning_rate
=0.01,random_state=0)
    ada_reg_dtree.fit(X_train, y_train)
    train_score_array.append(ada_reg_dtree.score(X_train, y_train))
    test_score_array.append(ada_reg_dtree.score(X_test, y_test))


x_axis = range(1,10)
plt.plot(x_axis, train_score_array, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_array, c = 'b', label = 'Test Score')
plt.legend()
plt.xlabel('Max Depth')
plt.ylabel('Scores')
```

Out[48]: 
```
Text(0, 0.5, 'Scores')
```



In [49]:
```
dt_reg = DecisionTreeRegressor(max_depth=5,random_state=0)
ada_reg_dt = AdaBoostRegressor(dt_reg, n_estimators=150,learning_rate=0.1,rand
om_state=0)

ada_reg_dt.fit(X_train,y_train)

y_pred=ada_reg_dt.predict(X_test)

print('Train score: {:.4f} %'.format(ada_reg_dt.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(ada_reg_dt.score(X_test, y_test)*100))

print('MAE: {:.4f}'.format(mean_absolute_error(y_test,y_pred)))
print('MSE: {:.4f}'.format(mean_squared_error(y_test,y_pred)))
print('RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
Train score: 63.2459 %
Test score: 52.0595 %
MAE: 0.4076
MSE: 0.2852
RMSE: 0.5341
```

# Model 2

### SVM with linear kernel

In [50]:
```
from sklearn.svm import SVR, LinearSVR
from sklearn.ensemble import AdaBoostRegressor
svr_reg=SVR(C=10)
ada_reg_svm = AdaBoostRegressor(svr_reg, n_estimators=100,learning_rate=0.1,ra
ndom_state=0)
ada_reg_svm.fit(X_train, y_train)

y_pred=ada_reg_svm.predict(X_test)

print('Train score: {:.4f} %'.format(ada_reg_svm.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(ada_reg_svm.score(X_test, y_test)*100))

print('MAE: {:.4f}'.format(mean_absolute_error(y_test,y_pred)))
print('MSE: {:.4f}'.format(mean_squared_error(y_test,y_pred)))
print('RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
Train score: 75.0771 %
Test score: 48.6931 %
MAE: 0.4217
MSE: 0.3053
RMSE: 0.5525
```

| Adaboost Regressor Scores | decision tree | SVC linear |
|---|---|---|
| Train | 63.2459 | 75.0771 |
| Test | 52.0595 | 48.6931 |

We have overfitting in both models used with Adabooster. specialy SVC

## Gradient boosting

In [51]:
```
from sklearn.ensemble import GradientBoostingRegressor

gbt_rf = GradientBoostingRegressor(n_estimators=500,learning_rate=0.1,random_s
tate=0)
gbt_rf.fit(X_train, y_train)

y_pred=gbt_rf.predict(X_test)

print('Train score: {:.4f} %'.format(gbt_rf.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(gbt_rf.score(X_test, y_test)*100))

print('RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
Train score: 78.6488 %
Test score: 52.2661 %
RMSE: 0.5329
```

In [52]:
```
from pprint import pprint
pprint(gbt_rf.get_params())
```

```
{'alpha': 0.9,
 'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'ls',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': 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': 500,
 'n_iter_no_change': None,
 'presort': 'deprecated',
 'random_state': 0,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

In [53]:
```
from sklearn.model_selection import GridSearchCV
param_grid_gbt = {
            'max_depth': range(1,5),
            'alpha': [0.5,0.6,0.7,0.8,0.9],
            'n_estimators': [50,100,150,200],
            'learning_rate': [0.01,0.1,1]
            }

CV_gbt = GridSearchCV(estimator =gbt_rf, param_grid = param_grid_gbt , return_
train_score=True, verbose = 1, n_jobs = -1)
CV_gbt.fit(X_train, y_train)

best_parameters_gbt=CV_gbt.best_params_
print(best_parameters_gbt)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed:   11.7s
[Parallel(n_jobs=-1)]: Done 776 tasks      | elapsed:   19.7s

{'alpha': 0.5, 'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 150}

[Parallel(n_jobs=-1)]: Done 1200 out of 1200 | elapsed:   29.5s finished
```

In [54]:
```
gbt_rf = GradientBoostingRegressor(n_estimators=50,learning_rate=0.1,random_st
ate=0,max_depth=3,alpha=0.5)
gbt_rf.fit(X_train, y_train)

y_pred=gbt_rf.predict(X_test)

print('Train score: {:.4f} %'.format(gbt_rf.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(gbt_rf.score(X_test, y_test)*100))

print('MAE: {:.4f}'.format(mean_absolute_error(y_test,y_pred)))
print('MSE: {:.4f}'.format(mean_squared_error(y_test,y_pred)))
print('RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
Train score: 63.0679 %
Test score: 54.0212 %
MAE: 0.3910
MSE: 0.2736
RMSE: 0.5230
```

We were able to achieve 60% trainscore and 55% testscore with gradientboosting regressor.

# PCA

In [55]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 0.90)
x_c = pca.fit_transform(X_train)
```

In [56]:
```
pca.n_components_
```

Out[56]:
```
12
```

In [57]:
```
from sklearn.decomposition import KernelPCA

linear_pca = KernelPCA(n_components =12, kernel="linear", gamma=0.04)

X_train_reduced= linear_pca.fit_transform(X_train)
X_test_reduced = linear_pca.transform(X_test)

print(X_train_reduced.shape)
print(y_train.shape)
print(X_test_reduced.shape)
print(y_test.shape)
```

```
(1233, 12)
(1233,)
(411, 12)
(411,)
```

# Models before and after PCA:

In [58]:
```
columns = ['Model', 'Regressor', 'Train Score', 'Test Score', 'MSE','MAE','RMS
E']
models = pd.DataFrame(columns=columns)
```

## Linear regression

original dataset

In [59]:
```
from sklearn.linear_model import LinearRegression

lreg = LinearRegression()
lreg.fit(X_train, y_train)

y_pred = lreg.predict(X_test)

print('Train score: {:.4f} %'.format(lreg.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(lreg.score(X_test, y_test)*100))
```

```
Train score: 55.8957 %
Test score: -867619030585644442750976.0000 %
```

In [60]:
```
models = models.append({'Model' : 'Linear regression',
                                                'Regressor' : '(Multiple)Linea
r Regressor without PCA',
                                        'Train Score' : lreg.score(X_train, y_
train),
                                       'Test Score' : lreg.score(X_test, y_tes
t),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

**Reduced dataset**

In [61]:
```
lreg = LinearRegression()
lreg.fit(X_train_reduced, y_train)

y_pred = lreg.predict(X_test_reduced)

print('Train score: {:.4f} %'.format(lreg.score(X_train_reduced, y_train)*100
))
print('Test score: {:.4f} %'.format(lreg.score(X_test_reduced, y_test)*100))
```

```
Train score: 43.3180 %
Test score: 44.6065 %
```

In [62]:
```
models = models.append({'Model' : 'Linear regression',
                                                'Regressor' : '(Multiple)Linea
r RegressorWith PCA',
                                        'Train Score' : lreg.score(X_train_red
uced, y_train),
                                       'Test Score' : lreg.score(X_test_reduce
d, y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

# KNN

### original

Gridsearch

In [63]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
param_grid_knn = {
            'leaf_size' : range(1,50),
            'n_neighbors' : range(1,50),
            'p': [1,2],
            'weights': ['distance','uniform'],
            'algorithm': ['auto', 'ball_tree']
            }

CV_knn = GridSearchCV(estimator =KNeighborsRegressor(), param_grid = param_gri
d_knn , return_train_score=True, verbose = 1, n_jobs = -1)
CV_knn.fit(X_train, y_train)

best_parameters_knn=CV_knn.best_params_
print(best_parameters_knn)
```

```
Fitting 5 folds for each of 648 candidates, totalling 3240 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Done 261 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done 511 tasks      | elapsed:   14.8s
[Parallel(n_jobs=-1)]: Done 861 tasks      | elapsed:   21.3s
[Parallel(n_jobs=-1)]: Done 1311 tasks      | elapsed:   29.4s
[Parallel(n_jobs=-1)]: Done 1861 tasks      | elapsed:   37.6s
[Parallel(n_jobs=-1)]: Done 2511 tasks      | elapsed:   46.4s

{'algorithm': 'auto', 'leaf_size': 2, 'n_neighbors': 9, 'p': 1, 'weights': 'd
istance'}

[Parallel(n_jobs=-1)]: Done 3240 out of 3240 | elapsed:   55.5s finished
```

In [64]:
```
from sklearn.neighbors import KNeighborsRegressor
knn_reg1=KNeighborsRegressor(n_neighbors=9,leaf_size=2,weights='distance',algo
rithm='auto',p=1)

knn_reg1.fit(X_train, y_train)

y_pred_knn=knn_reg1.predict(X_test)

print('Train score: {:.4f} %'.format(knn_reg1.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(knn_reg1.score(X_test, y_test)*100))
```

```
Train score: 98.1628 %
Test score: 46.3466 %
```

In [65]:
```
models = models.append({'Model' : 'KNN',
                                                   'Regressor' : 'KNN Regressor w
ithout PCA',
                                           'Train Score' : knn_reg1.score(X_train
, y_train),
                                          'Test Score' : knn_reg1.score(X_test, y
_test),
                                          'MSE' : mean_squared_error(y_test,y_pre
d),
                                       'MAE' : mean_absolute_error(y_test,y_pred
),
                                        'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                                  ignore_index=True)
```

In [66]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
knn_reg=KNeighborsRegressor(n_neighbors=9,leaf_size=1,weights='distance',algor
ithm='auto',p=1)
scores_train = cross_val_score(knn_reg, X_train, y_train)
scores_test = cross_val_predict(knn_reg, X_test, y_test)
print("Cross-validation scores_train: {}".format(scores_train))
print("Cross-validation scores_test: {}".format(scores_test))
```

```
Cross-validation scores_train: [0.41119701 0.28312801 0.44376087 0.39254236
0.48554263]
Cross-validation scores_test: [4.81508732 4.7371705  4.44519719 5.07422123 5.
25131618 4.91346301
 5.44756054 5.13491592 5.04940961 4.78213404 4.7156523  4.5830416
 5.64956008 4.32438434 5.19519823 4.55387689 5.56592859 4.35168064
 5.13296259 4.56727796 4.84892842 5.51905316 4.27129863 4.78483926
 4.71518104 3.97708115 4.40814812 4.77071929 4.94949667 4.57795542
 5.21707179 5.35305444 4.52216984 4.04570079 5.05036831 5.0860979
 5.32929632 4.48221603 4.22336824 5.57746709 4.96267609 5.16239045
 4.92786148 5.48463619 4.23599307 4.14645263 4.84611012 4.37660016
 4.52761399 4.56940593 4.47329595 4.95399953 4.040248   4.92852115
 5.04307854 4.85397386 5.15389034 4.68739262 4.14226541 5.13531744
 4.47429814 4.91886574 4.38269382 5.20953743 5.1304509  5.20359994
 4.94468153 3.97825833 4.47838015 5.35144583 5.84762412 5.21088754
 4.74967873 3.98591459 4.01338902 4.01120265 5.08825792 4.19718767
 5.0337784  5.16192915 5.47746225 5.01069837 5.07241105 5.32467906
 4.39144256 4.75784517 5.36734876 5.57520232 4.20576303 5.67616324
 5.1903891  4.4678607  5.5896447  4.67780172 4.70256836 5.14475558
 4.98019979 5.23842515 5.89092678 4.0681206  4.60115717 4.72406568
 5.20624947 4.8292514  5.43008178 5.25507108 4.11370697 5.04690671
 4.32879017 4.61205821 4.01928732 4.51123829 4.3959282  5.66610978
 5.26977713 3.9487446  4.56808401 4.19015598 4.37351917 4.49418175
 5.19535554 5.08240808 4.76892515 5.26348001 3.97899716 4.52221604
 4.84929486 4.85732151 5.29051403 4.02392543 4.41158857 5.13944983
 5.30015163 4.47183124 4.84682408 5.10114658 4.60548189 5.10973334
 4.07832561 5.50751623 5.21666326 4.79410497 5.33758893 4.9842919
 5.58100452 4.51545303 4.48477963 4.9854961  5.08960947 5.01293743
 4.80483994 4.83108935 3.68887945 3.99432265 4.98951854 4.30715721
 4.42800141 4.92301289 5.19368894 4.97122351 4.9317734  4.73247554
 4.34969724 4.991115   4.98035052 5.11688436 5.05833589 5.18882202
 4.80162346 5.35782279 4.85845172 4.66881705 4.99553155 5.26927771
 5.71002986 5.03779151 4.99583808 4.86899093 4.92822393 4.27229686
 5.37935919 4.39157681 5.22486496 4.92915825 5.2973116  4.44919937
 4.36944785 4.4086453  5.42560211 4.24462136 4.75303671 5.20317204
 5.16718213 5.2682439  6.4546297  4.30630398 5.02457094 5.45332788
 4.46350425 5.18640231 4.50697836 5.27908614 5.19020856 4.12181702
 5.45169984 4.25329539 5.13394813 4.77370393 5.26240076 5.01182356
 4.36441436 4.19541491 4.42938556 4.73355951 6.40026488 4.62359097
 5.34106279 5.48898196 4.93206617 4.51342385 4.4046953  4.07547896
 4.73703502 5.47751646 5.4961296  4.38423721 4.07437356 4.72698615
 5.41857002 4.98189542 4.45934572 5.24247168 3.99330346 4.48777392
 5.04400505 4.96282301 4.60517019 5.38558316 4.51398181 4.14378674
 4.7045778  4.77183836 5.6512236  4.39509699 5.48232155 5.22703519
 4.12166235 5.25335069 4.36129639 5.31551474 4.22061263 3.94583738
 3.73205463 5.43713397 4.0119396  4.1615458  4.85983369 4.50097212
 4.86970526 4.94088606 5.86606219 4.89833595 4.82687527 5.17245631
 4.94980806 4.34992616 4.12436002 4.23386782 4.34339469 4.1700325
 4.16601647 5.84406521 4.86374013 5.20436954 4.51250276 5.00621144
 4.57400363 4.32253485 4.90156839 4.11079931 4.55297691 5.18193924
 4.79582412 4.68745532 4.12771984 4.07833368 4.15711746 4.2035451
 5.33223639 4.82325757 4.50096226 5.19636251 4.2224839  4.47533741
 4.85007486 5.10928009 5.43253662 4.10748637 5.30371848 4.0635369
 4.52306624 4.30234344 5.322154   4.69523553 5.4615347  5.2482229
 5.59313304 5.41611095 4.87673337 4.65396035 4.09822271 4.91793538
 4.91041402 4.92921788 4.57036837 5.03576986 4.91926465 4.98892448
 4.13079513 5.45133565 4.31878972 4.24535004 5.25714915 4.27060394
```

```
 4.1371438  3.95999756 5.61105859 4.60250726 4.89248076 4.43972813
 5.48492249 4.21601103 5.26612001 4.13477088 5.24004953 5.17823404
 5.31778305 5.59848821 5.1112082  5.48866791 5.10412262 5.22375328
 5.15270605 4.04475915 5.09296973 4.19891153 4.24508329 5.17708884
 5.10590756 4.07387996 4.00582466 4.63741    5.17964504 4.57628979
 4.30246978 4.16914092 4.03950529 4.78115179 4.48797756 3.93089459
 6.03192092 5.19917875 5.41952849 5.13411778 4.59547066 5.04954927
 5.36105181 5.0417274  5.04960516 4.55447616 4.70352341 4.39509509
 4.56347527 4.50240072 5.38099269 5.24350555 5.31507384 4.54065434
 4.38657406 4.50127321 4.24298504 5.0869764  5.18699538 5.04531273
 5.19711766 4.61850416 3.95063204 4.35245088 5.38239367 3.806754
 4.73750064 4.04227642 5.0237597  4.41602193 6.90775528 5.25566263
 4.9529344  4.77136281 4.67892796 5.11263219 5.22739078 4.54265346
 4.22527539 4.44634523 4.71258468 5.41561846 4.08939005 5.4316176
 4.78529854 4.32473901 4.27282967]
```

In [67]:
```
print("Average cross-validation score_train: {:.4f}".format(scores_train.mean
()))
print("Average cross-validation score_test: {:.2f}".format(scores_test.mean
()))
```

```
Average cross-validation score_train: 0.4032
Average cross-validation score_test: 4.81
```

## Reduced dataset

In [68]:
```
from sklearn.model_selection import GridSearchCV
param_grid_knn = {
            'leaf_size' : range(1,50),
            'n_neighbors' : range(1,50),
            'p': [1,2],
            'weights': ['distance','uniform'],
            'algorithm': ['auto', 'ball_tree']
            }

CV_knn = GridSearchCV(estimator =knn_reg, param_grid = param_grid_knn , return
_train_score=True, verbose = 1, n_jobs = -1)
CV_knn.fit(X_train_reduced, y_train)

best_parameters_knn=CV_knn.best_params_
print(best_parameters_knn)
```

```
Fitting 5 folds for each of 2888 candidates, totalling 14440 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 1208 tasks      | elapsed:    8.4s
[Parallel(n_jobs=-1)]: Done 2208 tasks      | elapsed:   13.6s
[Parallel(n_jobs=-1)]: Done 3608 tasks      | elapsed:   19.6s
[Parallel(n_jobs=-1)]: Done 5408 tasks      | elapsed:   26.8s
[Parallel(n_jobs=-1)]: Done 7608 tasks      | elapsed:   35.6s
[Parallel(n_jobs=-1)]: Done 10208 tasks      | elapsed:   47.3s
[Parallel(n_jobs=-1)]: Done 13208 tasks      | elapsed:   59.8s

{'algorithm': 'auto', 'leaf_size': 4, 'n_neighbors': 15, 'p': 1, 'weights':
'uniform'}

[Parallel(n_jobs=-1)]: Done 14440 out of 14440 | elapsed:  1.1min finished
```

In [69]:
```
knn_reg1=KNeighborsRegressor(n_neighbors=15,leaf_size=4,weights='uniform',algo
rithm='auto',p=1)

knn_reg1.fit(X_train_reduced, y_train)

y_pred_knn=knn_reg1.predict(X_test_reduced)

print('Train score: {:.4f} %'.format(knn_reg1.score(X_train_reduced, y_train)*
100))
print('Test score: {:.4f} %'.format(knn_reg1.score(X_test_reduced, y_test)*100
))
```

```
Train score: 46.7906 %
Test score: 42.9709 %
```

In [70]:
```
models = models.append({'Model' : 'KNN',
                                                'Regressor' : 'KNN Regressor w
ith PCA',
                                        'Train Score' : knn_reg1.score(X_train
_reduced, y_train),
                                       'Test Score' : knn_reg1.score(X_test_re
duced, y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

In [71]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
knn_reg=KNeighborsRegressor(n_neighbors=15,leaf_size=4,weights='uniform',algor
ithm='auto',p=1)
scores_train = cross_val_score(knn_reg, X_train_reduced, y_train)
scores_test = cross_val_predict(knn_reg, X_test_reduced, y_test)
print("Cross-validation scores_train: {}".format(scores_train))
print("Cross-validation scores_test: {}".format(scores_test))
```

```
Cross-validation scores_train: [0.40082646 0.30252931 0.4066898  0.36241474
0.42376937]
Cross-validation scores_test: [4.74629137 4.80957254 4.42327814 5.13400024 5.
27888044 4.65643826
 5.34498506 4.98025219 4.87926796 5.08233323 5.02682283 4.42327814
 5.50298278 4.33702744 4.79817393 4.55440977 5.34845893 4.33060174
 4.99006697 4.64450762 4.80028003 5.4072258  4.41989087 5.04607909
 5.01946725 4.03034438 4.33702744 5.14495886 4.95483382 4.33853009
 5.28232333 5.34498506 4.52305518 4.3125887  4.99452128 4.89672442
 5.17407219 4.38623338 4.12892169 5.26011832 5.21182436 5.20332965
 4.78528866 5.16639575 4.55359469 4.20411475 4.95977309 4.48758983
 4.91107678 4.51391873 4.42327814 5.21182436 4.25197073 4.84584569
 5.0511726  4.65643826 5.24251543 4.69119139 4.2151666  5.24251543
 4.47216436 5.08819519 4.45932656 5.16639575 5.19216459 5.24251543
 5.31667942 4.50060929 4.60967461 5.45355056 5.54709365 5.31667942
 4.78348518 4.45521221 4.20411475 4.35059317 5.22520499 4.3125887
 5.12702497 5.29587225 5.16639575 5.17918503 5.20908709 5.11271262
 4.45813744 4.46830189 5.01019485 5.50941386 4.39284556 5.30832622
 5.03194605 4.50274647 5.33298828 4.72022648 4.69037003 4.70764873
 5.23211323 5.60123896 5.73317243 4.19587787 4.76302272 4.83500514
 4.70764873 4.74432216 5.11271262 5.09073779 4.47825891 5.0898249
 4.73234255 4.95324637 4.21569175 4.70764873 4.32329046 5.30832622
 5.00261479 4.37014776 4.44830057 4.35799052 4.35146178 4.81008436
 5.16266878 5.10137025 4.74177309 5.17406258 4.13892654 4.44830057
 4.91895572 4.93057352 5.14320888 4.59587956 4.43338731 5.17028971
 5.25863442 4.65139849 4.66961804 5.12463246 4.79343127 4.82558241
 4.02244906 5.30832622 5.15824474 5.09073779 5.09073779 5.3130236
 5.32146857 4.37649796 4.6484682  5.13447182 4.99518238 5.17199355
 5.05000025 4.91895572 4.59587956 3.96973043 5.19110573 4.32329046
 4.32329046 5.10186569 5.16266878 5.27710464 5.09073779 4.76434805
 4.24394604 5.27710464 5.13447182 5.04542802 5.14291667 5.3038515
 5.03291419 5.36863809 4.70804387 4.82710643 4.77434454 5.23107913
 5.59869891 4.83974024 5.31678683 4.43612511 5.13163548 4.27083241
 5.26982801 4.15484897 5.18493759 5.02100344 5.25939423 4.33504011
 4.31222998 4.12212083 5.32444815 4.30940976 4.41813572 5.26982801
 4.64925413 5.16425314 5.04731243 5.09416601 5.54305106 4.9830679
 4.42365108 5.26021909 4.27828206 5.10316394 4.86704287 4.22982095
 5.11413815 4.28738728 5.07458327 4.82233849 5.23107913 4.71519543
 4.28738728 4.53919336 4.5144494  5.22299779 5.04731243 4.39036961
 5.20714708 5.53941415 4.93685624 4.47428124 4.47443813 4.12212083
 5.24691985 5.14398665 5.35834836 4.28068052 4.46423144 4.56503295
 5.26021909 5.2628003  4.54210587 5.07458327 4.47529911 4.28068052
 4.75424073 4.91424832 4.58443754 5.34681162 4.35734244 4.24154028
 4.55947205 4.91176552 5.28700997 4.4137739  5.14289565 5.36863809
 4.31793801 5.28333057 4.18677445 5.18014204 4.11350436 4.1406534
 4.39904885 5.1937626  4.01931655 4.96365638 4.88538865 4.546453
 4.58836296 5.00202518 5.52547282 4.97488339 5.07843537 5.1632097
 5.08005747 4.26397416 4.36365631 4.32485038 4.26397416 4.34773707
 4.36365631 5.68801552 5.28183292 5.24064856 4.34477192 5.18749354
 4.31587105 4.36225099 5.28175001 4.30253615 4.77102425 5.03537387
 4.52948153 4.95156477 4.4651013  4.42365182 4.46019704 4.39703052
 5.19814746 4.97488339 4.95254057 5.04811089 4.30253615 4.46699978
 4.57745681 5.13899727 5.28333057 4.19940618 5.22526348 4.71972261
 4.57807086 4.32485038 5.31340483 4.91244843 5.53444854 5.18486051
 5.15746691 5.18669048 5.28333057 4.42838104 4.49416418 4.46699978
 5.19197775 4.9910574  4.46699978 5.19814746 4.92191155 5.03537387
 4.18677445 5.47963145 4.26397416 4.13809468 5.04811089 4.3308617
```

```
4.46019704 3.96677477 5.4349766  4.90058187 4.82544581 4.61355208
5.07332787 4.00930688 5.21808894 4.61406282 5.18329979 4.79278481
5.13965516 5.58812767 5.27324094 5.28588452 5.36763129 5.28630422
5.38415219 4.14846945 5.21645392 4.29533583 4.14779678 5.04585097
5.13080543 4.14846945 4.21230702 5.16241777 5.08912468 4.35673494
4.24139114 4.29059709 4.03418052 4.96714141 4.95400139 4.25524134
5.64997547 4.99384315 5.20234434 5.04815237 4.95087645 5.43380488
5.05262349 5.28459852 5.08276292 4.8028166  4.94336651 4.48626477
4.43206341 4.42138866 5.20856801 5.5244278  5.24483177 5.18619005
4.37634314 4.68408265 4.39810346 5.28015411 5.13965516 4.98832912
5.16353249 4.85700412 4.15535685 4.24139114 5.16582012 4.04427518
4.99289728 4.30625105 5.06637646 4.89219654 4.66795507 5.48040777
4.95644149 4.98673585 5.18619005 5.30109135 5.21351291 4.49540058
4.457289   4.36886503 5.06965866 5.16582012 4.17880776 5.1998538
4.92813995 4.30032437 4.30032437]
```

In [72]:
```
print("Average cross-validation score_train: {:.4f}".format(scores_train.mean
()))
print("Average cross-validation score_test: {:.2f}".format(scores_test.mean
()))
```

```
Average cross-validation score_train: 0.3792
Average cross-validation score_test: 4.83
```

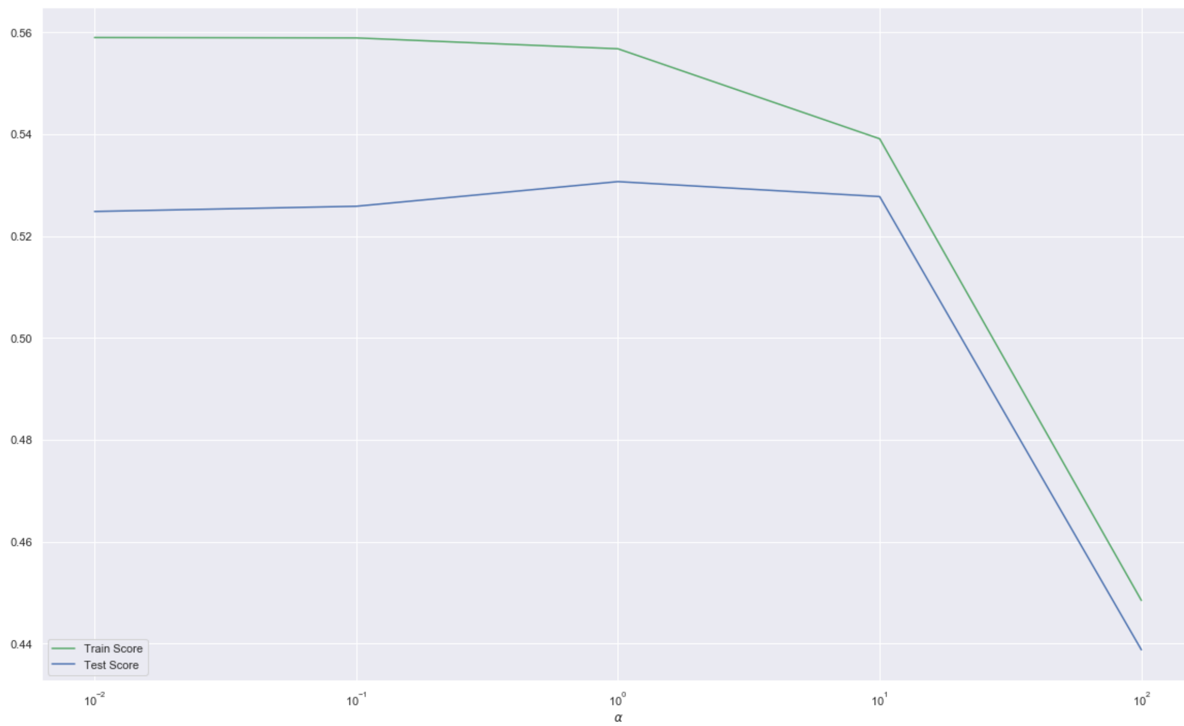
# Ridge

### Original dataset

In [73]:
```
from  sklearn.linear_model import Ridge

x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    train_score_list.append(ridge.score(X_train,y_train))
    test_score_list.append(ridge.score(X_test, y_test))
```

In [74]:
```
%matplotlib inline
import matplotlib.pyplot as plt

sns.set(rc={'figure.figsize':(20,12)})

plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[74]:
```
Text(0.5, 0, '$\\alpha$')
```

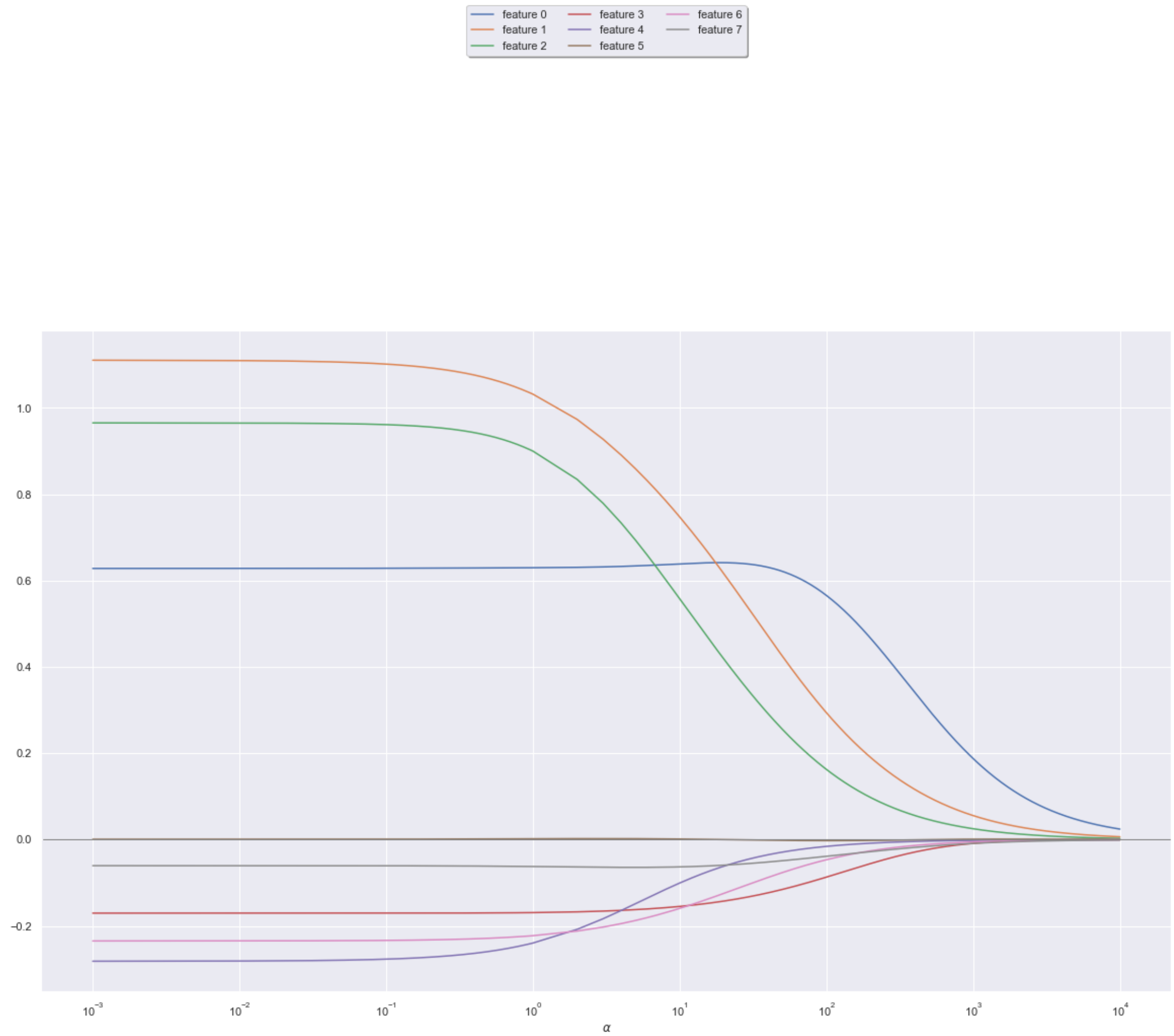



In [75]:
```
from  sklearn.linear_model import Ridge
ridge = Ridge(alpha = 10)
ridge.fit(X_train,y_train)

y_pred=ridge.predict(X_test)

print('Train score: {:.4f} %'.format(ridge.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(ridge.score(X_test, y_test)*100))
```

```
Train score: 53.9079 %
Test score: 52.7730 %
```

In [76]:
```
models = models.append({'Model' : 'Ridge',
                                                                   'Regressor' : 'Ridge Regressor
without PCA',
                                                           'Train Score' : ridge.score(X_train, y
_train),
                                                          'Test Score' : ridge.score(X_test, y_te
st),
                                                          'MSE' : mean_squared_error(y_test,y_pre
d),
                                                        'MAE' : mean_absolute_error(y_test,y_pred
),
                                                         'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                                                  ignore_index=True)
```

In [77]:
```
%matplotlib inline
sns.set(rc={'figure.figsize':(20,12)})

x_range1 = np.linspace(0.001, 1, 100).reshape(-1,1)
x_range2 = np.linspace(1, 10000, 10000).reshape(-1,1)

x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    coeff.append(ridge.coef_ )

coeff = np.array(coeff)

for i in range(0,8):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```




In [78]:
```
print("Average cross-validation score_train: {:.4f}".format(scores_train.mean
()))
print("Average cross-validation score_test: {:.2f}".format(scores_test.mean
()))
```

```
Average cross-validation score_train: 0.3792
Average cross-validation score_test: 4.83
```

In [79]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
ridge1 = Ridge(alpha = 10)
scores_train = cross_val_score(ridge1, X_train, y_train)
scores_test = cross_val_score(ridge1, X_test, y_test)
print("Cross-validation scores_train: {}".format(scores_train))
print("Cross-validation scores_test: {}".format(scores_test))
```

```
Cross-validation scores_train: [0.54491048 0.44755707 0.50951331 0.48023248
0.53442068]
Cross-validation scores_test: [0.5428013  0.50693039 0.47579163 0.56717695 0.
24672741]
```

### Reduced dataset

In [80]:
```
from  sklearn.linear_model import Ridge

x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train_reduced,y_train)
    train_score_list.append(ridge.score(X_train_reduced,y_train))
    test_score_list.append(ridge.score(X_test_reduced, y_test))
```

In [81]:
```
%matplotlib inline
import matplotlib.pyplot as plt

sns.set(rc={'figure.figsize':(20,12)})

plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[81]:
```
Text(0.5, 0, '$\\alpha$')
```

In [82]:
```
ridge = Ridge(alpha = 1)
ridge.fit(X_train_reduced,y_train)

y_pred=ridge.predict(X_test_reduced)

print('Train score: {:.4f} %'.format(ridge.score(X_train_reduced, y_train)*100
))
print('Test score: {:.4f} %'.format(ridge.score(X_test_reduced, y_test)*100))
```

```
Train score: 43.3172 %
Test score: 44.5676 %
```

In [83]:
```
models = models.append({'Model' : 'Ridge',
                                                'Regressor' : 'Ridge Regressor
with PCA',
                                        'Train Score' :ridge.score(X_train_red
uced, y_train),
                                       'Test Score' : ridge.score(X_test_reduc
ed, y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

In [84]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
ridge1 = Ridge(alpha = 1)
scores_train = cross_val_score(ridge1, X_train_reduced, y_train)
scores_test = cross_val_score(ridge1, X_test_reduced, y_test)
print("Cross-validation scores_train: {}".format(scores_train))
print("Cross-validation scores_test: {}".format(scores_test))
```

```
Cross-validation scores_train: [0.40230564 0.35951638 0.44342576 0.38703968
0.45908939]
Cross-validation scores_test: [0.49376484 0.4758514  0.46857454 0.46897962 0.
05399436]
```

In [85]:
```
print("Average cross-validation score_train: {:.4f}".format(scores_train.mean
()))
print("Average cross-validation score_test: {:.2f}".format(scores_test.mean
()))
```

```
Average cross-validation score_train: 0.4103
Average cross-validation score_test: 0.39
```

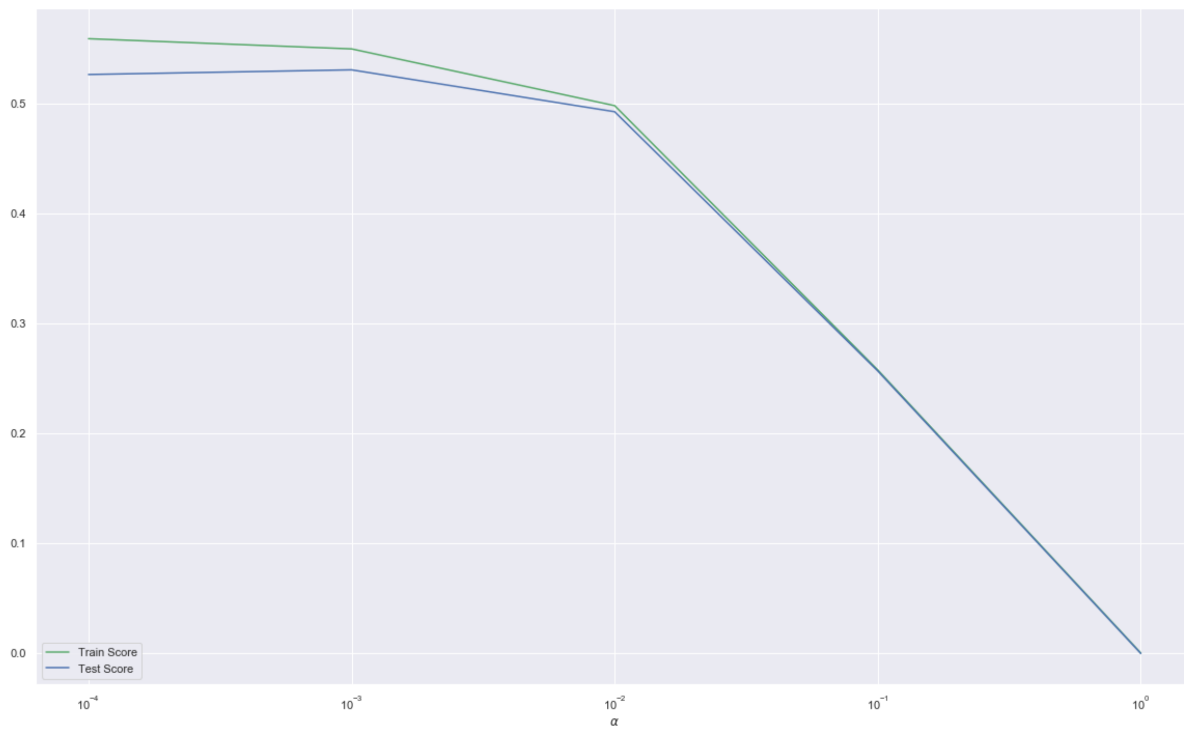
# Lasso

### Original set

In [86]:
```
from sklearn.linear_model import Lasso
x_range = [0.0001,0.001,0.01, 0.1, 1]
train_score_list = []
test_score_list = []

for alpha in x_range:
    lasso = Lasso(alpha)
    lasso.fit(X_train,y_train)
    train_score_list.append(lasso.score(X_train,y_train))
    test_score_list.append(lasso.score(X_test, y_test))
```

In [87]:
```
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[87]:
```
Text(0.5, 0, '$\\alpha$')
```

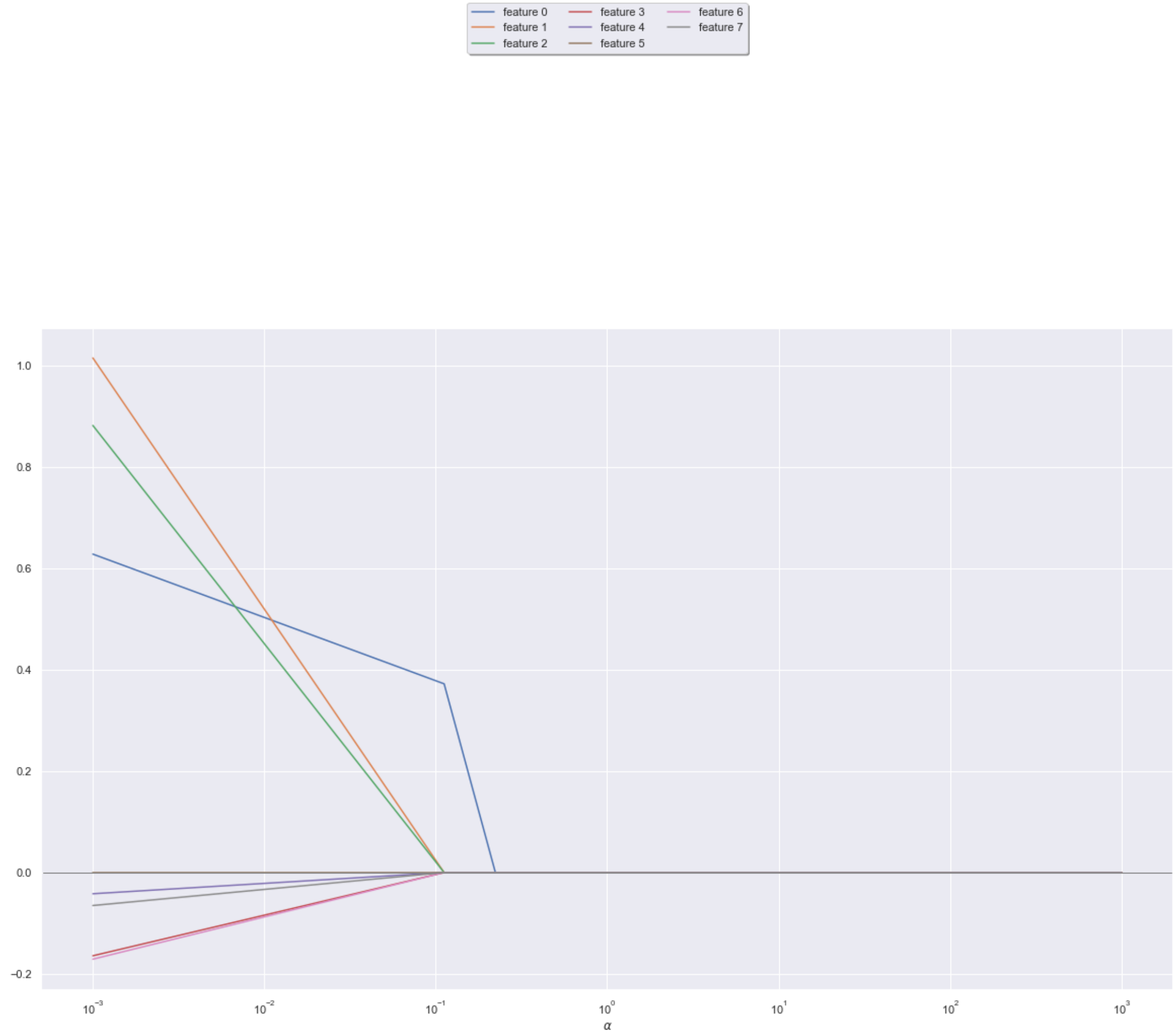



In [88]:
```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha = 0.0001)
lasso.fit(X_train,y_train)

y_pred=lasso.predict(X_test)

print('Train score: {:.4f} %'.format(lasso.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(lasso.score(X_test, y_test)*100))
```

```
Train score: 55.8741 %
Test score: 52.6178 %
```

In [89]:
```
models = models.append({'Model' : 'Lasso',
                                                'Regressor' : 'Lasso Regressor
without PCA',
                                        'Train Score' : lasso.score(X_train, y
_train),
                                       'Test Score' : lasso.score(X_test, y_te
st),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

In [90]:
```
%matplotlib inline
sns.set(rc={'figure.figsize':(20,12)})


x_range1 = np.linspace(0.001, 1, 10).reshape(-1,1)
x_range2 = np.linspace(1, 1000, 1000).reshape(-1,1)

x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    lasso = Lasso(alpha)
    lasso.fit(X_train,y_train)
    coeff.append(lasso.coef_ )

coeff = np.array(coeff)

for i in range(0,8):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```




In [91]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
lasso1 = Lasso(alpha = 0.0001)
scores_train = cross_val_score(lasso1, X_train, y_train)
scores_test = cross_val_predict(lasso1, X_train, y_train)

print("Cross-validation scores_train: {}".format(scores_train))
print("Cross-validation scores_test: {}".format(scores_test))
```

```
Cross-validation scores_train: [0.53337997 0.42372337 0.51808729 0.49258532
0.54179471]
Cross-validation scores_test: [5.77072229 4.94110854 4.49356912 ... 4.2330455
2 5.14348041 4.2336031 ]
```

In [92]:
```
print("Average cross-validation score: {:.4f}".format(scores_train.mean()))
print("Average cross-validation score: {:.2f}".format(scores_test.mean()))
```

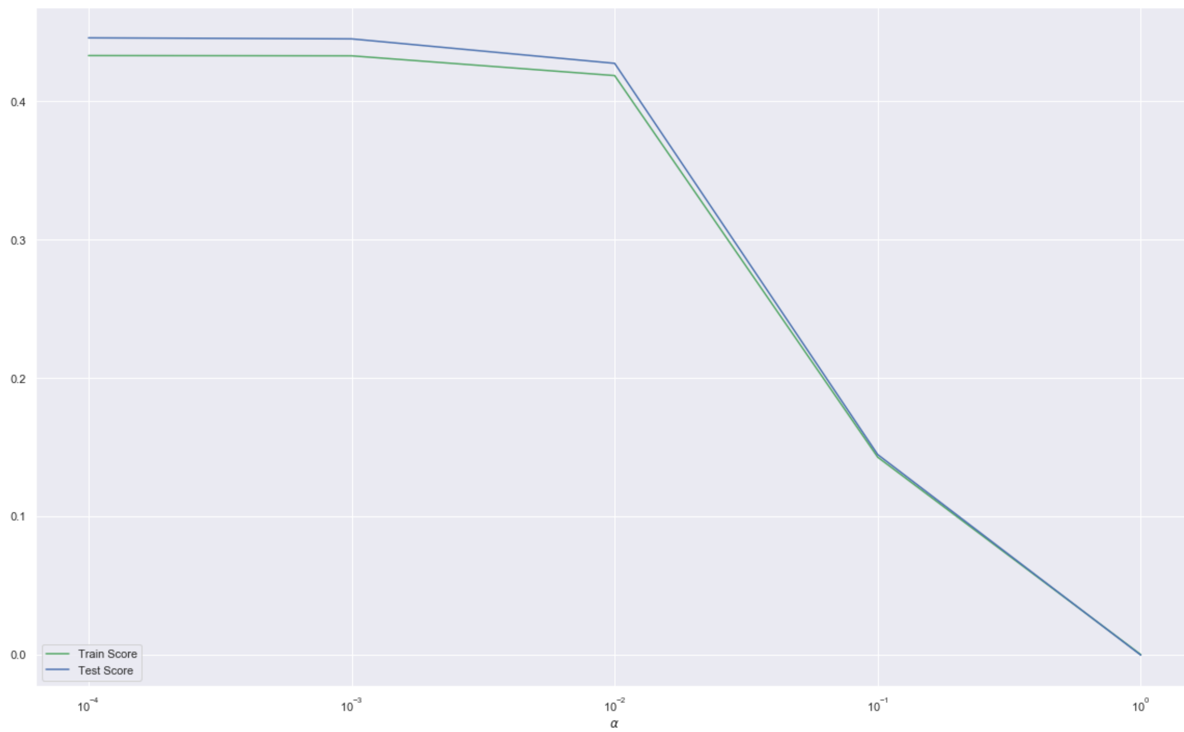
```
Average cross-validation score: 0.5019
Average cross-validation score: 4.80
```

### Reduced dataset

In [93]:
```
from sklearn.linear_model import Lasso
x_range = [0.0001,0.001,0.01, 0.1, 1]
train_score_list = []
test_score_list = []

for alpha in x_range:
    lasso = Lasso(alpha)
    lasso.fit(X_train_reduced,y_train)
    train_score_list.append(lasso.score(X_train_reduced,y_train))
    test_score_list.append(lasso.score(X_test_reduced, y_test))
```

In [94]:
```
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[94]:
```
Text(0.5, 0, '$\\alpha$')
```

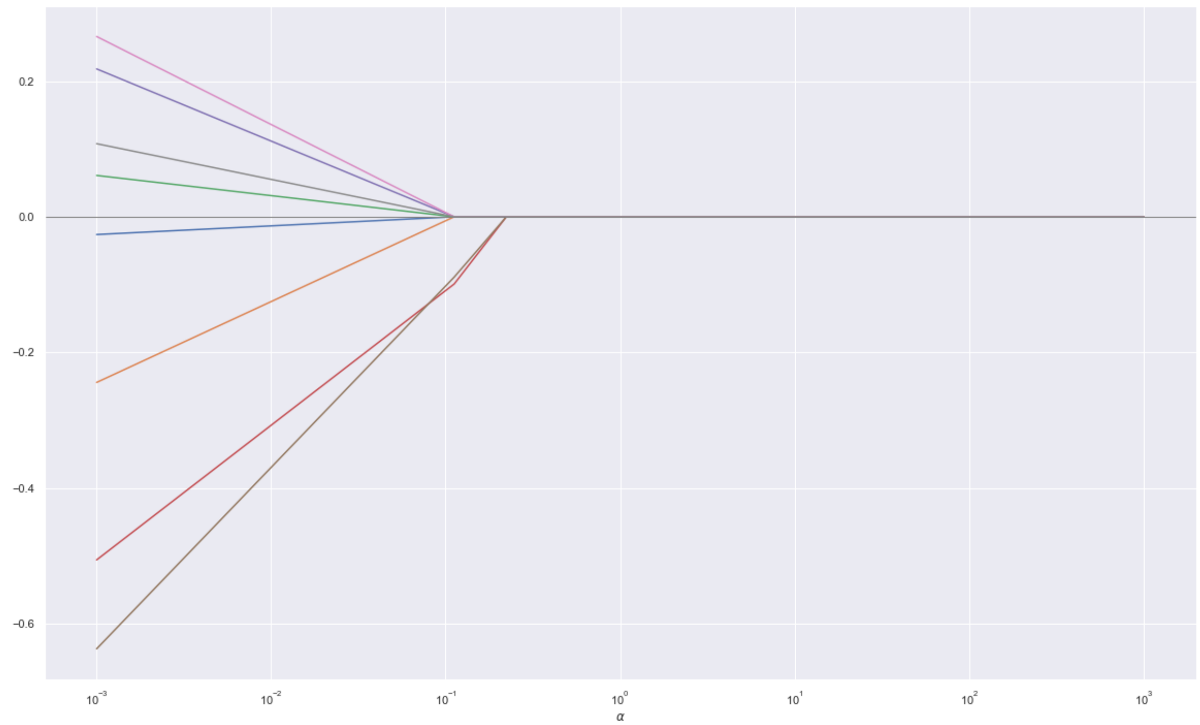



In [95]:
```
lasso = Lasso(alpha = 0.001)
lasso.fit(X_train_reduced,y_train)

y_pred=lasso.predict(X_test_reduced)

print('Train score: {:.4f} %'.format(lasso.score(X_train_reduced, y_train)*100
))
print('Test score: {:.4f} %'.format(lasso.score(X_test_reduced, y_test)*100))
```

```
Train score: 43.3006 %
Test score: 44.5255 %
```

In [96]:
```
models = models.append({'Model' : 'Lasso',
                                                'Regressor' : 'Lasso Regressor
with PCA',
                                        'Train Score' : lasso.score(X_train_re
duced, y_train),
                                       'Test Score' : lasso.score(X_test_reduc
ed, y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

In [97]:
```
%matplotlib inline
sns.set(rc={'figure.figsize':(20,12)})


x_range1 = np.linspace(0.001, 1, 10).reshape(-1,1)
x_range2 = np.linspace(1, 1000, 1000).reshape(-1,1)

x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    lasso = Lasso(alpha)
    lasso.fit(X_train_reduced,y_train)
    coeff.append(lasso.coef_ )

coeff = np.array(coeff)

for i in range(0,8):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```


In [98]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
lasso1 = Lasso(alpha = 0.001)
scores_train = cross_val_score(lasso1, X_train_reduced, y_train)
scores_test = cross_val_predict(lasso1, X_train_reduced, y_train)

print("Cross-validation scores_train: {}".format(scores_train))
print("Cross-validation scores_test: {}".format(scores_test))
```

```
Cross-validation scores_train: [0.40309495 0.36209414 0.44177038 0.3846773
0.45864527]
Cross-validation scores_test: [5.41527283 5.00019473 4.34538228 ... 4.3418989
1 5.10810034 4.36479338]
```

In [99]:
```
print("Average cross-validation score: {:.4f}".format(scores_train.mean()))
print("Average cross-validation score: {:.2f}".format(scores_test.mean()))
```

```
Average cross-validation score: 0.4101
Average cross-validation score: 4.80
```

# Polynomial regression

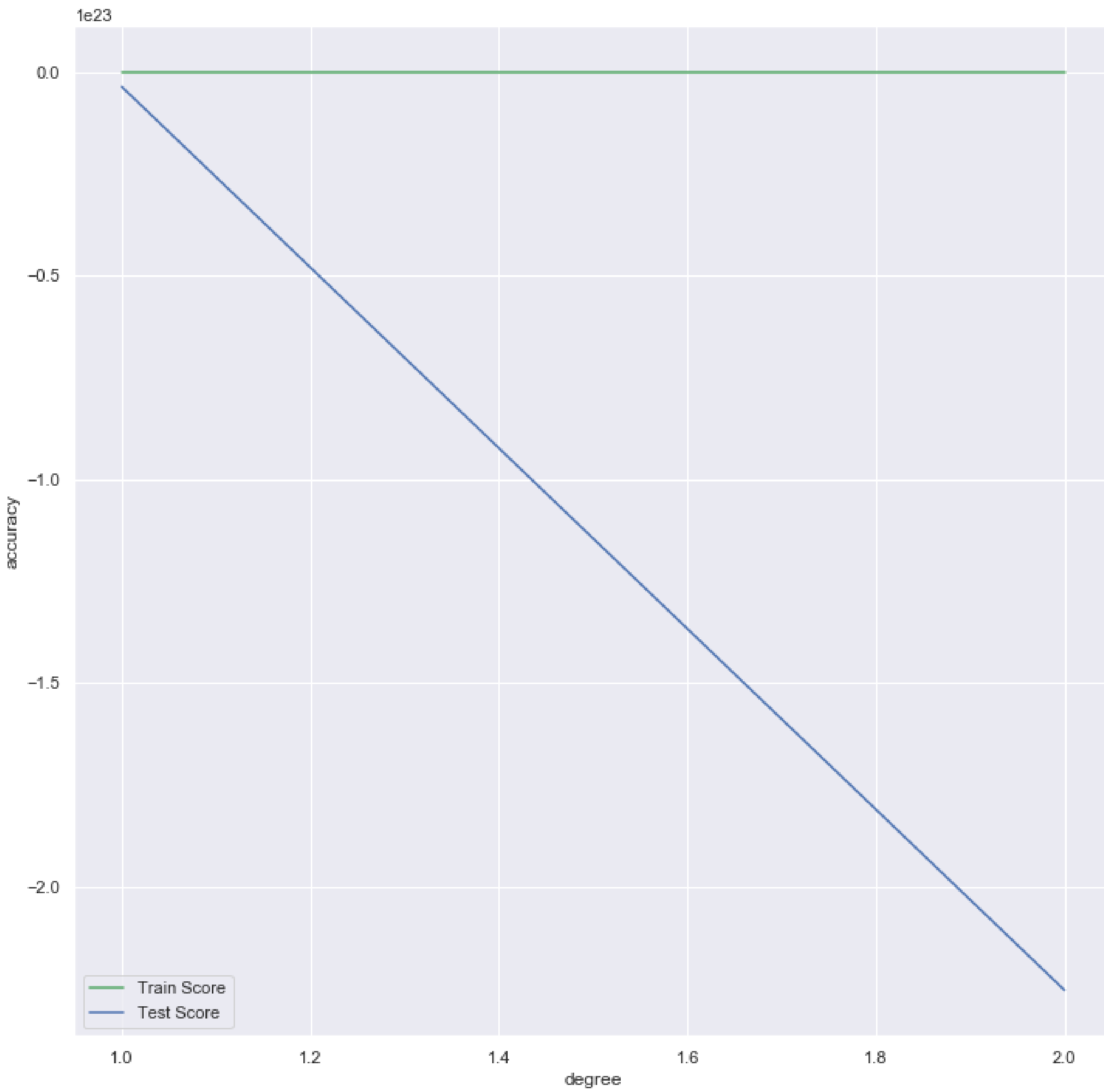
In [100]:
```
from sklearn.preprocessing import PolynomialFeatures
train_score_list = []
test_score_list = []

for n in range(1,3):
    poly = PolynomialFeatures(n)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)
    lreg.fit(X_train_poly, y_train)
    train_score_list.append(lreg.score(X_train_poly, y_train))
    test_score_list.append(lreg.score(X_test_poly, y_test))
```

In [101]:
```
print(train_score_list)
print(test_score_list)
```

```
[0.5558362875278833, 0.7076483605050513]
[-3.641465039358819e+21, -2.2540018789607715e+23]
```

In [102]:
```
%matplotlib inline
sns.set(rc={'figure.figsize':(12,12)})


x_axis = range(1,3)
plt.plot(x_axis, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel('degree')
plt.ylabel('accuracy')
plt.legend()
```

Out[102]: <matplotlib.legend.Legend at 0x1a9e57cec88>



In [103]:
```
poly = PolynomialFeatures(1)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
lreg.fit(X_train_poly, y_train)

y_pred=lreg.predict(X_test_poly)

print('Train score: {:.4f} %'.format(lreg.score(X_train_poly, y_train)*100))
print('Test score: {:.4f} %'.format(lreg.score(X_test_poly, y_test)*100))
```

```
Train score: 55.5836 %
Test score: -3641465039358818873507 84.0000 %
```

In [104]:
```
models = models.append({'Model' : 'Polynomial',
                                                'Regressor' : 'Polynomial Regr
essor without PCA',
                                        'Train Score' : lreg.score(X_train_pol
y, y_train),
                                       'Test Score' : lreg.score(X_test_poly,
y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

#### reduced dataset

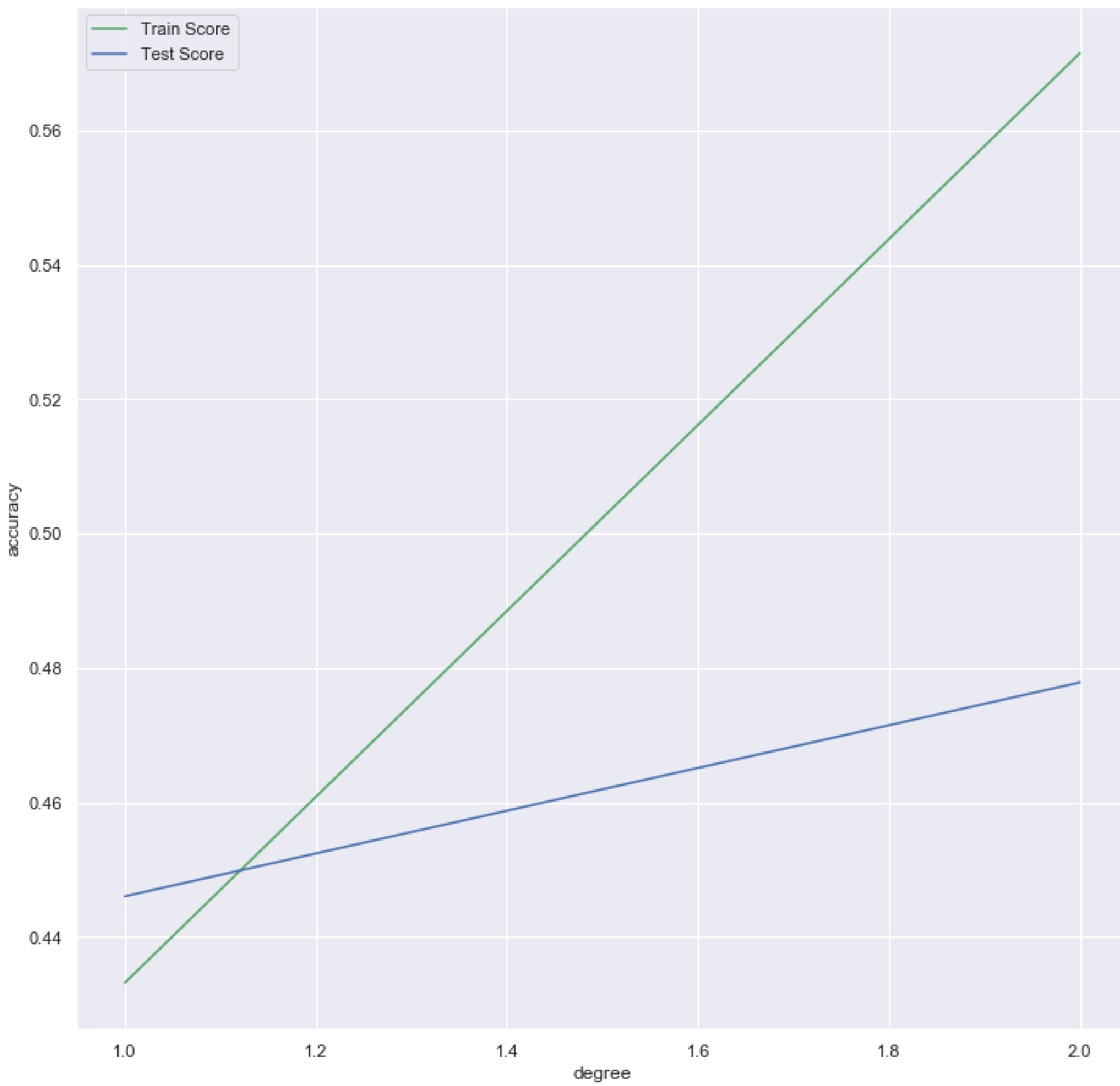
In [105]:
```
from sklearn.preprocessing import PolynomialFeatures
train_score_list = []
test_score_list = []

for n in range(1,3):
    poly = PolynomialFeatures(n)
    X_train_poly = poly.fit_transform(X_train_reduced)
    X_test_poly = poly.transform(X_test_reduced)
    lreg.fit(X_train_poly, y_train)
    train_score_list.append(lreg.score(X_train_poly, y_train))
    test_score_list.append(lreg.score(X_test_poly, y_test))
```

In [106]:
```
print(train_score_list)
print(test_score_list)
```

```
[0.433180109949559, 0.5714778814722447]
[0.4460649605261857, 0.47786468430308415]
```

In [107]:
```
%matplotlib inline
sns.set(rc={'figure.figsize':(12,12)})


x_axis = range(1,3)
plt.plot(x_axis, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel('degree')
plt.ylabel('accuracy')
plt.legend()
```

Out[107]: <matplotlib.legend.Legend at 0x1a9e559b388>



In [108]:
```
poly = PolynomialFeatures(2)
X_train_poly = poly.fit_transform(X_train_reduced)
X_test_poly = poly.transform(X_test_reduced)
lreg.fit(X_train_poly, y_train)

y_pred_poly=lreg.predict(X_test_poly)

print('Train score: {:.4f} %'.format(lreg.score(X_train_poly, y_train)*100))
print('Test score: {:.4f} %'.format(lreg.score(X_test_poly, y_test)*100))
```

```
Train score: 57.1478 %
Test score: 47.7865 %
```

In [109]:
```
models = models.append({'Model' : 'Polynomial',
                                                'Regressor' : 'Polynomial Regr
essor with PCA',
                                        'Train Score' : lreg.score(X_train_pol
y, y_train),
                                       'Test Score' : lreg.score(X_test_poly,
y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

# SVMs

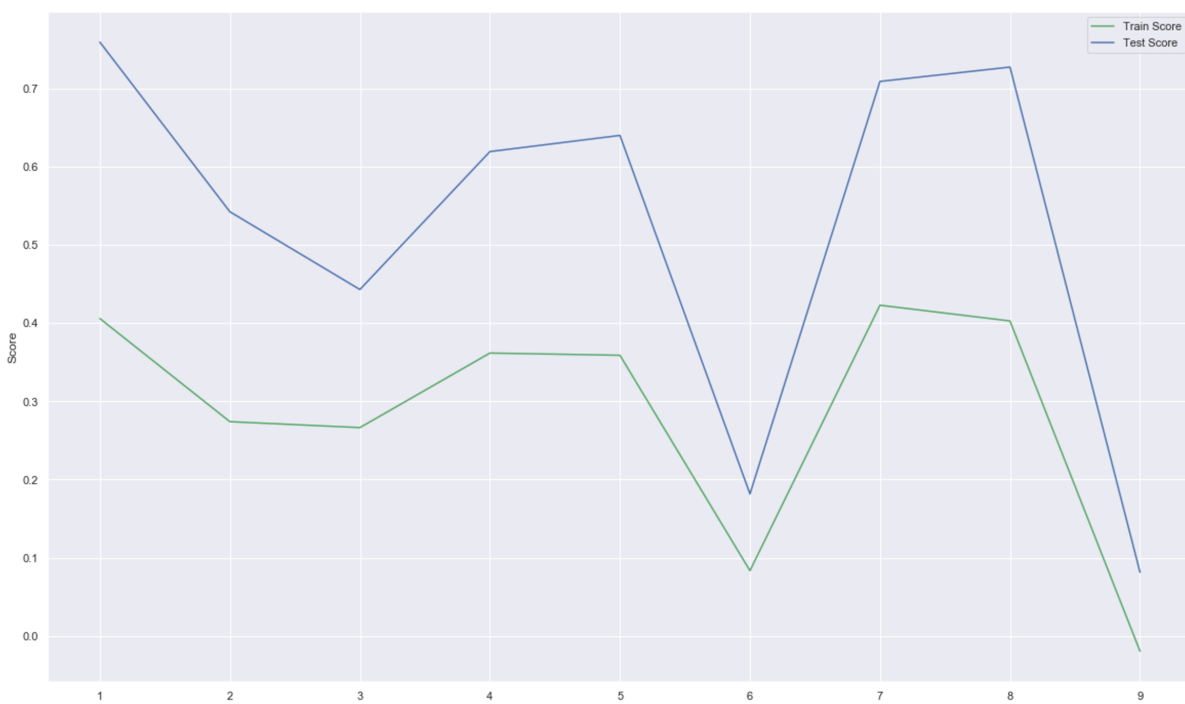
## original dataset

### linear SVM

In [110]:
```
from sklearn.svm import LinearSVR

sns.set(rc={'figure.figsize':(20,12)})
linear_svm = LinearSVR()

linear_svm.fit(X_train, y_train)
y_pred=linear_svm.predict(X_test)

train_score_array = []
test_score_array = []

for n in range(1,10):
    linear_svm = LinearSVR(max_iter=n)
    linear_svm.fit(X_train, y_train)
    train_score_array.append(linear_svm.score(X_train, y_train))
    test_score_array.append(linear_svm.score(X_test, y_pred))

x_axis = range(1,10)
plt.plot(x_axis, train_score_array, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_array, c = 'b', label = 'Test Score')
plt.legend()
plt.xlabel('Max Iterations')
plt.ylabel('Score')
```

```
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[110]: `Text(0, 0.5, 'Score')`



In [111]:
```
#9 iter_max
linear_svm = LinearSVR(max_iter=23)
linear_svm.fit(X_train, y_train)

y_pred=linear_svm.predict(X_test)

print('Train score: {:.4f} %'.format(linear_svm.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(linear_svm.score(X_test, y_test)*100))
```

```
Train score: 31.0240 %
Test score: 32.9820 %

C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [112]:
```
models = models.append({'Model' : 'SVM',
                                                'Regressor' : 'LinearSVR witho
ut PCA',
                                        'Train Score' : linear_svm.score(X_tra
in, y_train),
                                       'Test Score' : linear_svm.score(X_test,
y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

cross validation -SVM

In [113]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
linear_svr = LinearSVR(max_iter=9)
scores_train = cross_val_score(linear_svr, X_train, y_train)
scores_test = cross_val_predict(linear_svr, X_test, y_test)
print("Cross-validation scores_train: {}".format(scores_train))
print("Cross-validation scores_test: {}".format(scores_test))
```

```
Cross-validation scores_train: [-0.22127176  0.28819164  0.40146772  0.468733
52  0.34501129]
Cross-validation scores_test: [4.98289895 4.93638456 4.67535861 5.91285934 6.
37834842 5.18457428
 6.47732179 6.14182075 4.82817264 5.11907592 5.87861664 4.86847518
 5.63322633 4.76457599 5.53716697 4.82095112 6.63789041 4.94296976
 6.74178821 5.24125405 4.83634042 6.75055529 4.88642789 5.0220984
 5.45344954 4.80286799 4.84865663 5.35374683 5.80998285 4.79238656
 5.58480969 5.77541731 4.73125007 4.87600891 5.05305762 6.95256194
 5.79189275 4.84706746 4.62980172 6.43370581 5.52993404 4.56287103
 4.72409895 5.93815079 4.65057803 4.87083325 5.93795492 4.76125383
 5.19908697 5.04908548 4.9871358  5.4815768  4.88009689 5.93625112
 5.0251967  5.07362038 5.14051353 4.99197432 4.57951324 5.19068391
 4.52012173 5.33541205 4.61988591 5.56117906 5.55031558 5.45262148
 5.78412672 4.8547688  5.19121559 5.8064066  7.06406111 6.17513448
 5.25975475 4.81922453 4.83934923 5.08840521 6.31420748 4.85280056
 6.30549321 5.90629582 5.74702614 5.72733325 5.62138275 5.87161782
 5.34737287 5.314308   5.65375084 6.54934622 4.93152975 6.13799897
 5.44781161 5.20447204 6.55813845 5.01613698 5.34866288 5.43494505
 6.02669687 6.49412861 5.3849017  4.70654798 5.27060159 5.8196927
 5.37073376 5.92471044 5.93326149 6.10656229 5.294952   6.01489462
 5.32319823 5.7772259  4.43107552 5.31816846 5.31728002 5.88222969
 5.86765027 3.94077767 5.25290597 4.98873605 5.19347682 5.29112084
 6.21177026 5.96920524 5.93174291 6.30887444 4.83447095 5.01365618
 5.99212398 5.75685325 6.10772772 5.39351122 5.36807476 6.25771025
 6.23228241 6.11649844 6.46267462 5.95652941 5.29910887 6.15746093
 4.70961909 5.64667989 6.48579261 5.63269567 5.89068293 6.87147393
 6.05960122 5.17540868 5.34377852 5.96089978 6.06793197 5.62613785
 5.97654388 6.43160983 5.39573136 4.78319845 6.00837349 5.31399484
 5.3505826  5.64936374 6.33262254 5.92726285 5.55031158 5.9898304
 4.01714546 5.91592689 5.84065113 6.12058132 5.33934426 5.41012075
 5.30485011 5.28193774 4.22625291 4.90979815 5.72229695 5.23645549
 5.51789673 5.09161135 4.73238002 4.82755003 4.56615445 4.70390673
 5.02808451 4.69154861 5.89885193 4.96538513 5.87217858 3.94618581
 4.63010628 4.27493982 6.0278435  4.08881581 4.52470202 4.88152071
 5.58594384 5.08479127 5.23772191 4.56847509 5.14381949 6.44317829
 4.35103897 4.63534266 4.9203485  5.00279746 4.94596749 4.88095248
 5.11624511 4.31666736 5.15142802 4.62117691 5.18686679 4.8520087
 4.318258   4.54065344 4.94384381 4.33694659 5.38174783 4.35233545
 4.70981164 5.39787227 5.82165248 4.11874531 4.30831805 3.93774438
 4.62256531 6.5737396  5.81974738 4.56287056 4.75203601 4.69545727
 4.87261181 4.61742661 4.51713359 5.08347146 4.19297099 4.75267239
 5.19100298 5.18660546 4.80999309 5.84320249 4.65099248 4.63196126
 5.36987856 5.32869127 5.41488116 4.44992585 5.05831508 4.90620477
 4.49882013 4.74051946 4.16239037 5.57527539 3.90083236 3.91810186
 3.95639749 5.35396332 3.66882371 4.08559432 4.9627206  4.16830345
 3.54620934 4.83804226 6.41201516 4.83992567 4.60732696 5.35063021
 5.03098595 4.0118738  3.52150776 4.20163926 4.16711556 3.60360401
 3.49130271 5.58853883 4.59496213 5.36905895 4.0280645  4.94445528
 3.46327402 3.74919835 4.95317886 3.96744008 4.27273334 4.72711581
 4.72624727 3.75713559 4.01956016 4.05609483 3.76772915 4.1038569
 5.35165874 4.74068941 4.10197212 4.76358608 3.99928832 4.1974373
 4.09322312 4.92795465 5.42488003 4.30257072 5.93794182 3.84541691
 3.87138608 3.94538435 5.34563425 3.34007287 4.33357036 4.65717265
 5.89356954 5.41032814 4.55602141 3.415885   3.6254142  4.11670539
 4.96354328 5.15259485 4.16046203 4.97742607 5.20084061 4.59179167
 4.05171745 5.52247064 4.06107349 4.32573883 4.8293782  3.92277851
```

```
 3.75447908 3.97086086 5.674236   4.58117569 4.64220341 4.80974725
 6.15838188 4.84640731 5.5201443  4.5974373  5.42144748 4.35762171
 5.55601248 5.97551663 5.4774732  5.82552926 6.10638184 6.02146809
 4.82807792 4.27688359 5.58686073 4.11407181 4.04468123 5.77578681
 5.83133658 4.94916159 4.50398998 5.17902294 5.64788091 4.80886152
 4.69949679 4.66735477 4.18699423 5.5585569  5.25770895 4.19167578
 6.38359009 6.06629169 6.04582637 5.7972642  5.70419285 4.85800137
 6.02687557 5.87759546 6.07736981 4.30321471 5.1161544  4.31559876
 4.26852679 4.60856429 5.90898773 5.03719735 5.79167069 5.07906295
 4.3566236  4.83188952 4.89963063 5.89583619 5.61071771 4.96190943
 5.59539811 5.32293865 4.8589624  4.62496817 5.38360163 4.8270216
 4.91935305 4.12249643 4.98972642 5.0831475  4.69561113 5.76129864
 4.72224075 4.66940628 5.18641376 5.80196418 6.19572653 3.59543092
 4.81610536 4.28502209 4.79519156 5.74785374 4.5523271  6.04227837
 5.51455171 4.74478302 4.71082656]
```

```
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [114]:

```
print("Average cross-validation score_train: {:.2f}".format(scores_train.mean
()))
print("Average cross-validation score_test: {:.2f}".format(scores_test.mean
()))
```

```
Average cross-validation score_train: 0.26
Average cross-validation score_test: 5.12
```

### Reduced set

In [115]:
```
#9 iter_max
linear_svm = LinearSVR(max_iter=23)
linear_svm.fit(X_train_reduced, y_train)

y_pred=linear_svm.predict(X_test_reduced)

print('Train score: {:.4f} %'.format(linear_svm.score(X_train_reduced, y_train
)*100))
print('Test score: {:.4f} %'.format(linear_svm.score(X_test_reduced, y_test)*1
00))
```

```
Train score: 41.5965 %
Test score: 44.0624 %

C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [116]:
```
models = models.append({'Model' : 'SVM',
                                                'Regressor' : 'LinearSVR with
 PCA',
                                        'Train Score' : linear_svm.score(X_tra
in_reduced, y_train),
                                       'Test Score' : linear_svm.score(X_test_
reduced, y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

# SVR with linear Kernel

In [117]:
```
from sklearn.svm import SVR, LinearSVR
clf2 = SVR(kernel='linear', C=1)
clf2.fit(X_train,y_train)
clf2.score(X_train,y_train)
clf2.score(X_test,y_test)
```

Out[117]: 0.5154887478578933

In [118]:
```
clf2 = SVR(kernel='linear', C=10)
clf2.fit(X_train,y_train)
clf2.score(X_train,y_train)
clf2.score(X_test,y_test)
```

Out[118]: 0.5148180344285818

In [119]:
```
clf2 = SVR(kernel='linear', C=100)
clf2.fit(X_train,y_train)
clf2.score(X_train,y_train)
clf2.score(X_test,y_test)
```

Out[119]: 0.5144131581879268

In [120]:
```
clf2 = SVR(kernel='linear', C=1000)
clf2.fit(X_train,y_train)
clf2.score(X_train,y_train)
clf2.score(X_test,y_test)
```

Out[120]: 0.5136599032033538

Gridsearch

In [121]:
```
from sklearn import svm
from sklearn.svm import SVR

parameters = {'kernel': ['rbf','linear','poly'], 'C':[0.1,1,10],'gamma': [0.1,
1,10],'epsilon':[0.1,1,10,100]}
svr = svm.SVR()
clf = GridSearchCV(svr, parameters, return_train_score=True, verbose = 1, n_jo
bs = -1)
clf.fit(X_train,y_train)
clf.best_params_
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:    2.5s finished
```

Out[121]: {'C': 1, 'epsilon': 1, 'gamma': 1, 'kernel': 'rbf'}

CV for SVM

In [122]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
clf2 = SVR(kernel='rbf', C=1,gamma=1)
scores_train = cross_val_score(clf2, X_train, y_train)
scores_test = cross_val_predict(clf2, X_test, y_test)
print("Cross-validation scores_train: {}".format(scores_train.mean()))
print("Cross-validation scores_test: {}".format(scores_test.mean()))
```

```
Cross-validation scores_train: 0.3782384744046878
Cross-validation scores_test: 4.787258523827444
```

In [123]:
```
clf2.fit(X_train, y_train)
y_pred=clf2.predict(X_test)
```

In [124]:
```
models = models.append({'Model' : 'SVR',
                                                'Regressor' : 'SVR with rbf wi
thout PCA',
                                        'Train Score' : clf2.score(X_train, y_
train),
                                       'Test Score' : clf2.score(X_test, y_tes
t),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                              ignore_index=True)
```

## Reduced dataset

Gridsearch

In [125]:
```
from sklearn import svm
from sklearn.svm import SVR

parameters = {'kernel': ['rbf','linear','poly'], 'C':[0.1,1,10],'gamma': [0.1,
1,10],'epsilon':[0.1,1,10,100]}
svr = svm.SVR()
clf = GridSearchCV(svr, parameters, return_train_score=True, verbose = 1, n_jo
bs = -1)
clf.fit(X_train_reduced,y_train)
clf.best_params_
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 out of  20 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    0.0s finished
```

Out[125]:
```
{'C': 10, 'epsilon': 1, 'gamma': 0.01, 'kernel': 'rbf'}
```

{'C': 10, 'epsilon': 1, 'gamma': 0.01, 'kernel': 'rbf'}

In [126]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
clf2 = SVR(kernel='rbf', C=10,gamma=0.01,epsilon=1)
clf2.fit(X_train_reduced, y_train)
y_pred=clf2.predict(X_test_reduced)
```

In [127]:
```
models = models.append({'Model' : 'SVR',
                                        'Regressor' : 'SVR with rbf wi
th PCA',
                                'Train Score' : clf2.score(X_train_red
uced, y_train),
                               'Test Score' : clf2.score(X_test_reduce
d, y_test),
                               'MSE' : mean_squared_error(y_test,y_pre
d),
                            'MAE' : mean_absolute_error(y_test,y_pred
),
                             'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                              ignore_index=True)
```

CV for SVR with rbf kernel

In [128]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
clf2 = SVR(kernel='rbf', C=10,gamma=0.01,epsilon=1)
scores_train = cross_val_score(clf2, X_train_reduced, y_train)
scores_test = cross_val_predict(clf2, X_test_reduced, y_test)
print("Cross-validation scores_train: {}".format(scores_train.mean()))
print("Cross-validation scores_test: {}".format(scores_test.mean()))
```

```
Cross-validation scores_train: 0.35915327286645365
Cross-validation scores_test: 4.948347745069806
```

CV for SVM

In [129]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
clf2 = SVR(kernel='rbf', C=10,gamma=0.01)
scores_train = cross_val_score(clf2, X_train_reduced, y_train)
scores_test = cross_val_predict(clf2, X_test_reduced, y_test)
print("Cross-validation scores_train: {}".format(scores_train.mean()))
print("Cross-validation scores_test: {}".format(scores_test.mean()))
```

```
Cross-validation scores_train: 0.4175578485287785
Cross-validation scores_test: 4.778027701747669
```

SVM with poly kernel

original dataset

In [130]:
```
from sklearn.svm import SVR
model = SVR(kernel='poly')
parameters = {'C':[0.1,1,10]}
clf = GridSearchCV(model, parameters, cv=5, return_train_score=True)
clf.fit(X_train, y_train)
clf.best_params_
```

Out[130]:
```
{'C': 1}
```

In [131]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
clf2 = SVR(kernel='poly', C=1)
scores_train = cross_val_score(clf2, X_train, y_train)
scores_test = cross_val_predict(clf2, X_test, y_test)
print("Cross-validation scores_train: {}".format(scores_train.mean()))
print("Cross-validation scores_test: {}".format(scores_test.mean()))
```

```
Cross-validation scores_train: 0.49655784224724614
Cross-validation scores_test: 4.7984134626006005
```

In [132]:
```
clf2 = SVR(kernel='poly', C=1)
clf2.fit(X_train, y_train)
y_pred=clf2.predict(X_test)
```

In [133]:
```
models = models.append({'Model' : 'SVR',
                                                'Regressor' : 'SVR with poly w
ithout PCA',
                                        'Train Score' : clf2.score(X_train, y_
train),
                                       'Test Score' : clf2.score(X_test, y_tes
t),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

### reduced dataset

In [134]:
```
from sklearn.svm import SVR
model = SVR(kernel='poly')
parameters = {'C':[0.1,1,10]}
clf = GridSearchCV(model, parameters, cv=5, return_train_score=True)
clf.fit(X_train_reduced, y_train)
clf.best_params_
```

Out[134]:
```
{'C': 0.1}
```

In [135]:
```
clf2 = SVR(kernel='poly', C=0.1)
clf2.fit(X_train_reduced, y_train)
y_pred=clf2.predict(X_test_reduced)
```

In [136]:
```
models = models.append({'Model' : 'SVR',
                                           'Regressor' : 'SVR with poly without PCA',
                                     'Train Score' : clf2.score(X_train_reduced, y_train),
                                    'Test Score' : clf2.score(X_test_reduced, y_test),
                                    'MSE' : mean_squared_error(y_test,y_pred),
                                 'MAE' : mean_absolute_error(y_test,y_pred),
                                  'RMSE' : np.sqrt(mean_squared_error(y_test,y_pred))},
                                         ignore_index=True)
```

In [137]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
clf2 = SVR(kernel='poly', C=0.1)
scores_train = cross_val_score(clf2, X_train_reduced, y_train)
scores_test = cross_val_predict(clf2, X_test_reduced, y_test)
print("Cross-validation scores_train: {}".format(scores_train.mean()))
print("Cross-validation scores_test: {}".format(scores_test.mean()))
```

```
Cross-validation scores_train: 0.38769293827281304
Cross-validation scores_test: 4.767453695173047
```

# Decision tree regressor

## Original dataset

In [138]:
```
tree = DecisionTreeRegressor(max_depth=4)
tree.fit(X_train, y_train)

y_pred=tree.predict(X_test)

print('Train score: {:.4f} %'.format(tree.score(X_train, y_train)*100))
print('Test score: {:.4f} %'.format(tree.score(X_test, y_test)*100))
```

```
Train score: 52.5432 %
Test score: 46.9497 %
```

In [139]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
param_grid_tree = {
            'max_depth' : range(1,5),
            'min_samples_leaf' : range(1,5)
            }

CV_tree = GridSearchCV(estimator =tree, param_grid = param_grid_tree , return_
train_score=True, verbose = 1, n_jobs = -1)
CV_tree.fit(X_train, y_train)

best_parameters_tree=CV_tree.best_params_
print(best_parameters_tree)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.1s finished

{'max_depth': 4, 'min_samples_leaf': 4}
```

In [140]:
```
tree1 = DecisionTreeRegressor(max_depth=4,min_samples_leaf=4)
tree1.fit(X_train, y_train)
y_pred=tree1.predict(X_test)
```

In [141]:
```
models = models.append({'Model' : 'Decision tree',
                                                 'Regressor' : 'decision tree w
ithout PCA',
                                         'Train Score' : tree1.score(X_train, y
_train),
                                        'Test Score' : tree1.score(X_test, y_te
st),
                                        'MSE' : mean_squared_error(y_test,y_pre
d),
                                     'MAE' : mean_absolute_error(y_test,y_pred
),
                                      'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                               ignore_index=True)
```

cv for decision tree regressor

In [142]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
tree1 = DecisionTreeRegressor(max_depth=4,min_samples_leaf=4)
scores_train = cross_val_score(tree1, X_train, y_train)
scores_test = cross_val_predict(tree1, X_test, y_test)
print("Cross-validation scores_train: {}".format(scores_train.mean()))
print("Cross-validation scores_test: {}".format(scores_test.mean()))
```

```
Cross-validation scores_train: 0.44742829097129927
Cross-validation scores_test: 4.8253115215317655
```

## Reduced dataset

In [143]:
```
tree = DecisionTreeRegressor(max_depth=4)
tree.fit(X_train_reduced, y_train)

y_pred=tree.predict(X_test_reduced)

print('Train score: {:.4f} %'.format(tree.score(X_train_reduced, y_train)*100
))
print('Test score: {:.4f} %'.format(tree.score(X_test_reduced, y_test)*100))
```

```
Train score: 44.4487 %
Test score: 37.4900 %
```

In [144]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
param_grid_tree = {
            'max_depth' : range(1,10),
            'min_samples_leaf' : range(1,10)
            }

CV_tree = GridSearchCV(estimator =tree, param_grid = param_grid_tree , return_
train_score=True, verbose = 1, n_jobs = -1)
CV_tree.fit(X_train_reduced, y_train)

best_parameters_tree=CV_tree.best_params_
print(best_parameters_tree)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.

{'max_depth': 5, 'min_samples_leaf': 9}

[Parallel(n_jobs=-1)]: Done 405 out of 405 | elapsed:    0.4s finished
```

In [145]:
```
tree1 = DecisionTreeRegressor(max_depth=5,min_samples_leaf=9)

tree1.fit(X_train_reduced, y_train)
y_pred=tree1.predict(X_test_reduced)
```

In [146]:
```
models = models.append({'Model' : 'Decsion tree',
                                                'Regressor' : 'Decision tree w
ith PCA',
                                        'Train Score' : tree1.score(X_train_re
duced, y_train),
                                       'Test Score' : tree1.score(X_test_reduc
ed, y_test),
                                       'MSE' : mean_squared_error(y_test,y_pre
d),
                                    'MAE' : mean_absolute_error(y_test,y_pred
),
                                     'RMSE' : np.sqrt(mean_squared_error(y_tes
t,y_pred))},
                                                ignore_index=True)
```

cv for decision tree regressor

In [147]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
tree1 = DecisionTreeRegressor(max_depth=5,min_samples_leaf=9)
scores_train = cross_val_score(tree1, X_train_reduced, y_train)
scores_test = cross_val_predict(tree1, X_test_reduced, y_test)
print("Cross-validation scores_train: {}".format(scores_train.mean()))
print("Cross-validation scores_test: {}".format(scores_test.mean()))
```

```
Cross-validation scores_train: 0.33828407417212797
Cross-validation scores_test: 4.815133668120479
```

**Comparison of results with and without using PCA**

## Deep learning model

Neural nets :

In [152]:
```
# pip install keras
# pip install tensorflow
```

In [153]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
import numpy

# fix random seed for reproducibility
numpy.random.seed(10)
```

Perceptron

In [154]:
```
# create model
model = Sequential()
model.add(Dense(10, input_dim=47, kernel_initializer='normal', activation='rel
u'))
model.add(Dense(1, kernel_initializer='normal'))
```

In [155]:
```
# Compile model
model.compile(loss='mse', optimizer='sgd' , metrics = ['mse'])
```

In [156]:
```
X_train.shape
```

Out[156]:
```
(1233, 47)
```

In [157]:
```
model.fit(X_train, y_train,epochs=10)
```

```
Epoch 1/10
1233/1233 [==============================] - 0s 55us/step - loss: 6.6410 - ms
e: 6.6410
Epoch 2/10
1233/1233 [==============================] - 0s 19us/step - loss: 0.4056 - ms
e: 0.4056
Epoch 3/10
1233/1233 [==============================] - 0s 19us/step - loss: 0.3627 - ms
e: 0.3627
Epoch 4/10
1233/1233 [==============================] - 0s 20us/step - loss: 0.3370 - ms
e: 0.3370
Epoch 5/10
1233/1233 [==============================] - 0s 20us/step - loss: 0.3167 - ms
e: 0.3167
Epoch 6/10
1233/1233 [==============================] - 0s 19us/step - loss: 0.3034 - ms
e: 0.3034
Epoch 7/10
1233/1233 [==============================] - 0s 20us/step - loss: 0.2925 - ms
e: 0.2925
Epoch 8/10
1233/1233 [==============================] - 0s 19us/step - loss: 0.2856 - ms
e: 0.2856
Epoch 9/10
1233/1233 [==============================] - 0s 19us/step - loss: 0.2811 - ms
e: 0.2811
Epoch 10/10
1233/1233 [==============================] - 0s 19us/step - loss: 0.2753 - ms
e: 0.2753
```

Out[157]:
```
<keras.callbacks.callbacks.History at 0x1a9ec0b49c8>
```

In [158]:
```
model.evaluate(X_test, y_test)
```

```
411/411 [==============================] - 0s 49us/step
```

Out[158]:
```
[0.31068396985240804, 0.3106839954853058]
```

In [159]:
```
from sklearn.metrics import r2_score, recall_score, precision_score

y_train_predict = model.predict(X_train)
y_test_predict = model.predict(X_test)

print('Train score: {:.2f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.2f}'.format(r2_score(y_test, y_test_predict)))
```

```
Train score: 0.45
Test score: 0.48
```

In [160]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV
```

In [161]:
```
def create_model_reg():
    #create model
    model = Sequential()
    model.add(Dense(20, input_dim=45, kernel_initializer='normal', activation=
'relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    #compile model
    model.compile(loss='mse', optimizer='sgd' , metrics = ['mse'])
    return model
```

In [162]:
```
seed = 10
np.random.seed(10)
```

In [163]:
```
from keras.wrappers.scikit_learn import KerasRegressor
model =KerasRegressor(build_fn = create_model_reg, verbose = 0)

param_grid = {'batch_size':[10,20,30] , 'epochs':[10, 50, 100]}
grid_search = GridSearchCV(estimator= model, param_grid = param_grid, cv = 5)
```

In [164]:
```
grid_search_result = grid_search.fit(X_train, y_train)
```

```
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_5_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_7_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_9_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_11_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_13_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_15_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_17_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_19_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
```

```
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_21_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_23_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_25_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_27_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_29_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_31_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_33_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_35_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
```

```
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_37_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_39_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_41_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_43_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_45_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_47_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_49_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_51_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
```

```
ValueError: Error when checking input: expected dense_53_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_55_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_57_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_59_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_61_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_63_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_65_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_67_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_69_input to have shape
```

```
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_71_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_73_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_75_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_77_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_79_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_81_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_83_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_85_input to have shape
(45,) but got array with shape (47,)
```

```
  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_87_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_89_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_91_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
C:\Users\mahdi\Anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:536: FitFailedWarning: Estimator fit failed. The score on this train-tes
t partition for these parameters will be set to nan. Details:
ValueError: Error when checking input: expected dense_93_input to have shape
(45,) but got array with shape (47,)

  FitFailedWarning)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-164-0c5d0e51f109> in <module>
----> 1 grid_search_result = grid_search.fit(X_train, y_train)

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit(self, X, y, groups, **fit_params)
    737             refit_start_time = time.time()
    738             if y is not None:
--> 739                 self.best_estimator_.fit(X, y, **fit_params)
    740             else:
    741                 self.best_estimator_.fit(X, **fit_params)

~\Anaconda3\lib\site-packages\keras\wrappers\scikit_learn.py in fit(self, x, y, **kwargs)
    149         fit_args.update(kwargs)
    150 
--> 151         history = self.model.fit(x, y, **fit_args)
    152 
    153         return history

~\Anaconda3\lib\site-packages\keras\engine\training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_freq, max_queue_size, workers, use_multiprocessing, **kwargs)
   1152             sample_weight=sample_weight,
   1153             class_weight=class_weight,
-> 1154             batch_size=batch_size)
   1155 
   1156         # Prepare validation data.

~\Anaconda3\lib\site-packages\keras\engine\training.py in _standardize_user_data(self, x, y, sample_weight, class_weight, check_array_lengths, batch_size)
    577             feed_input_shapes,
    578             check_batch_axis=False,  # Don't enforce the batch size.
--> 579             exception_prefix='input')
    580 
    581         if y is not None:

~\Anaconda3\lib\site-packages\keras\engine\training_utils.py in standardize_input_data(data, names, shapes, check_batch_axis, exception_prefix)
    143                             ': expected ' + names[i] + ' to have shape ' +
    144                             str(shape) + ' but got array with shape ' +
--> 145                             str(data_shape))
    146     return data
    147 

ValueError: Error when checking input: expected dense_95_input to have shape (45,) but got array with shape (47,)
```

In [166]:
```
model1 = Sequential()
model1.add(Dense(20, input_dim=47, kernel_initializer='normal', activation='re
lu'))
model1.add(Dense(1, kernel_initializer='normal'))
    #compile model
model1.compile(loss='mse', optimizer='sgd' , metrics = ['mse'])
model1.fit(X_train, y_train, batch_size= 10, epochs=10)
#model.evaluate(X_test, y_test)

y_train_predict = model1.predict(X_train)
y_test_predict = model1.predict(X_test)

print('Train score: {:.2f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.2f}'.format(r2_score(y_test, y_test_predict)))
```

```
Epoch 1/10
1233/1233 [==============================] - 0s 80us/step - loss: 1.8206 - ms
e: 1.8206
Epoch 2/10
1233/1233 [==============================] - 0s 58us/step - loss: 0.3320 - ms
e: 0.3320
Epoch 3/10
1233/1233 [==============================] - 0s 59us/step - loss: 0.2893 - ms
e: 0.2893
Epoch 4/10
1233/1233 [==============================] - 0s 61us/step - loss: 0.2773 - ms
e: 0.2773
Epoch 5/10
1233/1233 [==============================] - 0s 58us/step - loss: 0.2667 - ms
e: 0.2667
Epoch 6/10
1233/1233 [==============================] - 0s 60us/step - loss: 0.2601 - ms
e: 0.2601
Epoch 7/10
1233/1233 [==============================] - 0s 62us/step - loss: 0.2582 - ms
e: 0.2582
Epoch 8/10
1233/1233 [==============================] - 0s 60us/step - loss: 0.2515 - ms
e: 0.2515
Epoch 9/10
1233/1233 [==============================] - 0s 61us/step - loss: 0.2497 - ms
e: 0.2497
Epoch 10/10
1233/1233 [==============================] - 0s 60us/step - loss: 0.2439 - ms
e: 0.2439
Train score: 0.52
Test score: 0.52
```

Multi layer perceptron

In [170]:
```
from keras import optimizers
model2 = Sequential()
model2.add(Dense(32, input_dim =47, activation = 'relu'))
model2.add(Dense(64, activation = 'relu'))
model2.add(Dense(64, activation = 'relu'))
model2.add(Dense(32, activation = 'relu'))
model2.add(Dense(1))
```

In [171]:
```
model2.compile(loss='mean_absolute_error' , optimizer = 'adam',metrics=['mae']
)
```

In [172]:
```
model2.fit(X_train, y_train,epochs = 300, batch_size = 50,verbose=0)
y_train_predict = model2.predict(X_train)
y_test_predict = model2.predict(X_test)

print('Train score: {:.2f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.2f}'.format(r2_score(y_test, y_test_predict)))
```

```
Train score: 0.78
Test score: 0.36
```

In [173]:
```
model3 = Sequential()
model3.add(Dense(8, input_dim =47, activation = 'relu'))
model3.add(Dense(16, activation = 'relu'))
model3.add(Dense(32, activation = 'relu'))
model3.add(Dense(64, activation = 'relu'))
model3.add(Dense(64, activation = 'relu'))
model3.add(Dense(32, activation = 'relu'))
model3.add(Dense(16, activation = 'relu'))
model3.add(Dense(8, activation = 'relu'))
model3.add(Dense(1))
```

In [174]:
```
model3.compile(loss='mean_absolute_error' , optimizer = 'adam',metrics=['mae']
)
```

In [175]:
```
model3.fit(X_train, y_train,epochs = 300, batch_size = 50,verbose=0)
y_train_predict = model3.predict(X_train)
y_test_predict = model3.predict(X_test)

print('Train score: {:.2f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.2f}'.format(r2_score(y_test, y_test_predict)))
```

```
Train score: 0.66
Test score: 0.45
```

In [176]:
```
model4 = Sequential()
model4.add(Dense(8, input_dim =47,kernel_initializer='normal', activation = 'r
elu'))
model4.add(Dense(16, activation = 'relu'))
model4.add(Dense(32, activation = 'relu'))
model4.add(Dense(64, activation = 'relu'))
model4.add(Dense(64, activation = 'relu'))
model4.add(Dense(32, activation = 'relu'))
model4.add(Dense(16, activation = 'relu'))
model4.add(Dense(8, activation = 'relu'))
model4.add(Dense(1,kernel_initializer='normal'))
```

In [177]:
```
model4.compile(loss='mean_absolute_error' , optimizer = 'adam',metrics=['mae']
)
```

In [178]:
```
model4.fit(X_train, y_train,epochs = 300, batch_size = 50,verbose=0)
y_train_predict = model4.predict(X_train)
y_test_predict = model4.predict(X_test)

print('Train score: {:.2f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.2f}'.format(r2_score(y_test, y_test_predict)))
```

```
Train score: 0.58
Test score: 0.50
```

### Best NN model :

# Comparison and conclusions:

In [179]:
```
# models=models.drop(models.index[[12,13]]).sort_values(by =['Test Score','Train Score' ,'MSE'],ascending=False)
models=models.sort_values(by =['Test Score','Train Score' ,'MSE'],ascending=False)

models
```

Out[179]:

| | Model | Regressor | Train Score | Test Score | MSE | MAE | RMS |
|---|---|---|---|---|---|---|---|
| 4 | Ridge | Ridge Regressor without PCA | 0.539079 | 5.277302e-01 | 2.809984e-01 | 3.960557e-01 | 5.300928e-( |
| 6 | Lasso | Lasso Regressor without PCA | 0.558741 | 5.261778e-01 | 2.819221e-01 | 3.977008e-01 | 5.309633e-( |
| 14 | SVR | SVR with poly without PCA | 0.638890 | 5.259943e-01 | 2.820313e-01 | 4.027400e-01 | 5.310661e-( |
| 9 | Polynomial | Polynomial Regressor with PCA | 0.571478 | 4.778647e-01 | 2.166655e+21 | 3.976808e+09 | 4.654735e+1 |
| 16 | Decision tree | decision tree without PCA | 0.521606 | 4.719236e-01 | 3.142031e-01 | 4.156628e-01 | 5.605382e-( |
| 2 | KNN | KNN Regressor without PCA | 0.981628 | 4.634659e-01 | 3.295889e-01 | 4.335480e-01 | 5.740983e-( |
| 1 | Linear regression | (Multiple)Linear Regressorwith PCA | 0.433180 | 4.460650e-01 | 3.295889e-01 | 4.335480e-01 | 5.740983e-( |
| 5 | Ridge | Ridge Regressor with PCA | 0.433172 | 4.456761e-01 | 3.298203e-01 | 4.335965e-01 | 5.742998e-( |
| 7 | Lasso | Lasso Regressorwith PCA | 0.433006 | 4.452546e-01 | 3.300711e-01 | 4.337480e-01 | 5.745181e-( |
| 11 | SVM | LinearSVR with PCA | 0.415965 | 4.406242e-01 | 3.328261e-01 | 4.405064e-01 | 5.769108e-( |
| 15 | SVR | SVR with poly without PCA | 0.481356 | 4.300453e-01 | 3.391205e-01 | 4.435272e-01 | 5.823405e-( |
| 3 | KNN | KNN Regressor with PCA | 0.467906 | 4.297088e-01 | 3.295889e-01 | 4.335480e-01 | 5.740983e-( |
| 12 | SVR | SVR with rbf without PCA | 0.736722 | 4.277631e-01 | 3.404784e-01 | 4.437660e-01 | 5.835053e-( |
| 13 | SVR | SVR with rbf with PCA | 0.403555 | 4.195902e-01 | 3.453413e-01 | 4.520964e-01 | 5.876574e-( |
| 17 | Decsion tree | Decision tree with PCA | 0.507121 | 3.845197e-01 | 3.662080e-01 | 4.623581e-01 | 6.051512e-( |
| 10 | SVM | LinearSVR without PCA | 0.310240 | 3.298198e-01 | 3.987542e-01 | 5.039332e-01 | 6.314699e-( |
| 0 | Linear regression | (Multiple)Linear Regressor without PCA | 0.558957 | -8.676190e+20 | 5.162294e+20 | 1.941159e+09 | 2.272068e+1 |
| 8 | Polynomial | Polynomial Regressor without PCA | 0.555836 | -3.641465e+21 | 2.166655e+21 | 3.976808e+09 | 4.654735e+1 |

# Top 5 models

In [180]:
```
models.head()
```

Out[180]:

| | Model | Regressor | Train Score | Test Score | MSE | MAE | RMSE |
|---|---|---|---|---|---|---|---|
| 4 | Ridge | Ridge Regressor without PCA | 0.539079 | 0.527730 | 2.809984e-01 | 3.960557e-01 | 5.300928e-01 |
| 6 | Lasso | Lasso Regressor without PCA | 0.558741 | 0.526178 | 2.819221e-01 | 3.977008e-01 | 5.309633e-01 |
| 14 | SVR | SVR with poly without PCA | 0.638890 | 0.525994 | 2.820313e-01 | 4.027400e-01 | 5.310661e-01 |
| 9 | Polynomial | Polynomial Regressor with PCA | 0.571478 | 0.477865 | 2.166655e+21 | 3.976808e+09 | 4.654735e+10 |
| 16 | Decision tree | decision tree without PCA | 0.521606 | 0.471924 | 3.142031e-01 | 4.156628e-01 | 5.605382e-01 |

Among all the models included models SVM with rbf kernel run on the reduced dataset gives the best accuracy and Test score and is the better predictor.

In [ ]: