In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

The dataset used is "Students Performance in Exams: Marks secured by the students in various subjects" by jaki link (https://www.kaggle.com/spscientist/students-performance-in-exams) that has scores for different exams and different factors for 1000 students. Futher description below.

Loading the Dataset and seeing

In [2]:
```
data=pd.read_csv('datasets_74977_169835_StudentsPerformance.csv')
data.head()
```

Out[2]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

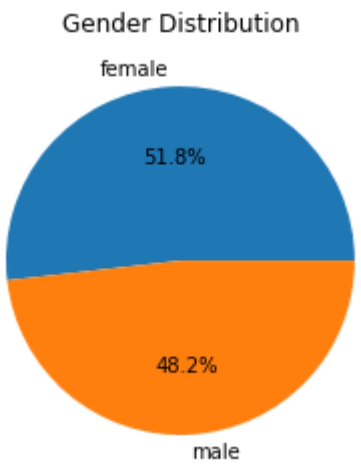
In [3]:
```
data.describe()
```

Out[3]:

	math score	reading score	writing score
count	1000.00000	1000.000000	1000.000000
mean	66.08900	69.169000	68.054000
std	15.16308	14.600192	15.195657
min	0.00000	17.000000	10.000000
25%	57.00000	59.000000	57.750000
50%	66.00000	70.000000	69.000000
75%	77.00000	79.000000	79.000000
max	100.00000	100.000000	100.000000

In [4]:
```
plt.title('Gender Distribution')
plt.pie([len(data.where(data['gender']==g).dropna()) for g in data['gender'].unique()],labels=data['gender'].
unique(),autopct='%1.1f%%')
```

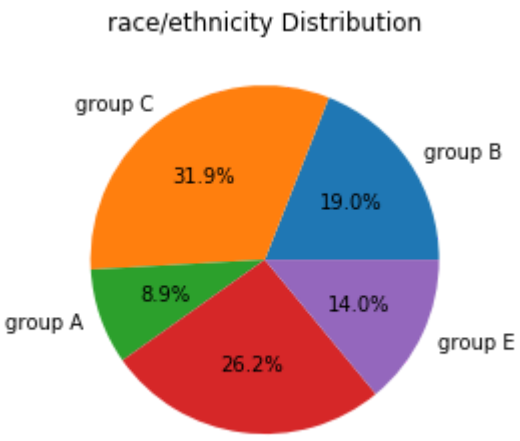
Out[4]:
```
([<matplotlib.patches.Wedge at 0x26f61eef908>,
  <matplotlib.patches.Wedge at 0x26f61ef4188>],
 [Text(-0.06217041096298424, 1.0982417038160106, 'female'),
  Text(0.06217041096298411, -1.0982417038160106, 'male')],
 [Text(-0.033911133252536856, 0.5990409293541875, '51.8%'),
  Text(0.033911133252536786, -0.5990409293541875, '48.2%')])
```



For our first categorical feature, gender, we have an even distribution almost split 50:50 between males and females in the dataset

In [5]:
```
plt.title('race/ethnicity Distribution')
plt.pie([len(data.where(data['race/ethnicity']==g).dropna()) for g in data['race/ethnicity'].unique()],labels
=data['race/ethnicity'].unique(),autopct='%1.1f%%')
```

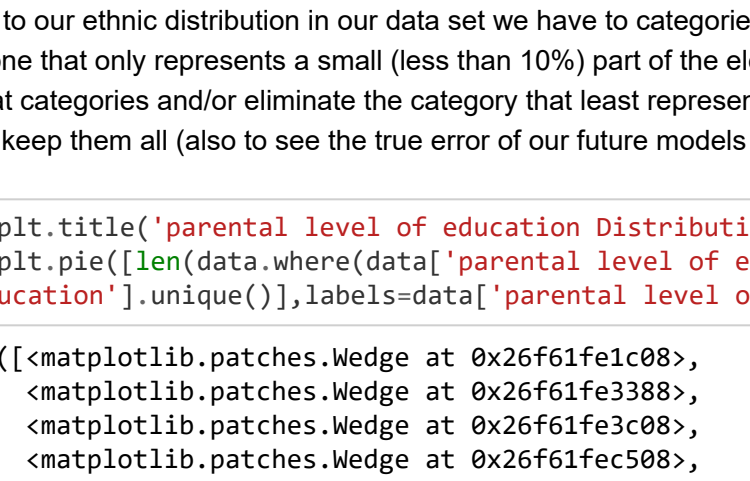
Out[5]:
```
([<matplotlib.patches.Wedge at 0x26f61f7a908>,
  <matplotlib.patches.Wedge at 0x26f61f7b448>,
  <matplotlib.patches.Wedge at 0x26f61f7bcc8>,
  <matplotlib.patches.Wedge at 0x26f61f845c8>,
  <matplotlib.patches.Wedge at 0x26f61f8d108>],
 [Text(0.9097886363331099, 0.618291708822899, 'group B'),
  Text(-0.6437648321003798, 0.8919455369868554, 'group C'),
  Text(-1.038434601178987, -0.362840983178911, 'group A'),
  Text(-0.14472078029994478, -1.0904383961276194, 'group D'),
  Text(0.9953097568356061, -0.4683572225853327, 'group E')],
 [Text(0.4962483470907872, 0.33725002299430845, '19.0%'),
  Text(-0.3511444538729344, 0.48651574744737563, '31.9%'),
  Text(-0.5664188733703563, -0.19791326355213326, '8.9%'),
  Text(-0.07893860743633352, -0.5947845797059742, '26.2%'),
  Text(0.5428962310012396, -0.255467575955636, '14.0%')])
```



When we come to our ethnic distribution in our data set we have to categories that predominate the dataset by representing more than 50% of the elements, and one that only represents a small (less than 10%) part of the elements in our data set. We can either decide to eliminate random elements from the dominat categories and/or eliminate the category that least represents our data to avoid biases. In this case, since our dataset isn´t that big, we are going to keep them all (also to see the true error of our future models and how well they behave with different data).

In [6]:
```
plt.title('parental level of education Distribution')
plt.pie([len(data.where(data['parental level of education']==g).dropna()) for g in data['parental level of ed
ucation'].unique()],labels=data['parental level of education'].unique(),autopct='%1.1f%%')
```

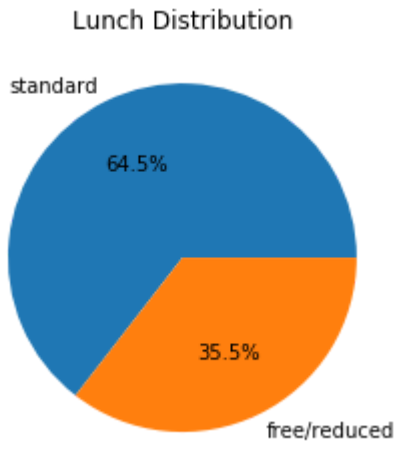
Out[6]:
```
([<matplotlib.patches.Wedge at 0x26f61fe1c08>,
  <matplotlib.patches.Wedge at 0x26f61fe3388>,
  <matplotlib.patches.Wedge at 0x26f61fe3c08>,
  <matplotlib.patches.Wedge at 0x26f61fec508>,
  <matplotlib.patches.Wedge at 0x26f61ff7048>,
  <matplotlib.patches.Wedge at 0x26f61ff7988>],
 [Text(1.0252782228085795, 0.3985029056788436, "bachelor's degree"),
  Text(0.13100688469389565, 1.092170863996472, 'some college'),
  Text(-0.7704522508666288, 0.7851135772195926, "master's degree"),
  Text(-1.0957469709260732, -0.09663630635705887, "associate's degree"),
  Text(-0.18571680664389534, -1.0842090516731513, 'high school'),
  Text(0.9306078067899267, -0.5864887977972321, 'some high school')],
 [Text(0.5592426669864978, 0.21736522127936922, '11.8%'),
  Text(0.07145830074212489, 0.5957295621798938, '22.6%'),
  Text(-0.4202466822908884, 0.428243769392505, '5.9%'),
  Text(-0.5976801659596762, -0.052710712558395746, '22.2%'),
  Text(-0.10130007635121563, -0.5913867554580825, '19.6%'),
  Text(0.5076042582490509, -0.319902980616672, '17.9%')])
```



Like we talked before, but in this case most of the categories represent almost a same portion of our data except for the master´s degree that only represents 6%. Nevertheless we are going to keep them to test for true error.

In [7]:
```
plt.title('Lunch Distribution')
plt.pie([len(data.where(data['lunch']==g).dropna()) for g in data['lunch'].unique()],labels=data['lunch'].uni
que(),autopct='%1.1f%%')
```

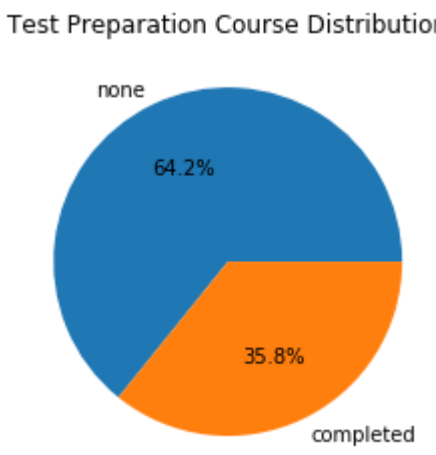
Out[7]:
```
([<matplotlib.patches.Wedge at 0x26f6204b508>,
  <matplotlib.patches.Wedge at 0x26f6204d688>],
 [Text(-0.48393302764960144, 0.9878303623344902, 'standard'),
  Text(0.48393293516224545, -0.9878304076435662, 'free/reduced')],
 [Text(-0.2639634696270553, 0.5388165612733582, '64.5%'),
  Text(0.2639634191794066, -0.5388165859873997, '35.5%')])
```



We can see the most dominant is standard lunch, but free/reduced still has a significant representation of our dataset

In [8]:
```
plt.title('Test Preparation Course Distribution')
plt.pie([len(data.where(data['test preparation course']==g).dropna()) for g in data['test preparation course'
].unique()],labels=data['test preparation course'].unique(),autopct='%1.1f%%')
```

Out[8]:
```
([<matplotlib.patches.Wedge at 0x26f620a0108>,
  <matplotlib.patches.Wedge at 0x26f620a1188>],
 [Text(-0.47460171119818767, 0.9923473261553901, 'none'),
  Text(0.4746018041084478, -0.9923472817199666, 'completed')],
 [Text(-0.2588736606535569, 0.5412803597211218, '64.2%'),
  Text(0.2588737113318806, -0.5412803354836181, '35.8%')])
```

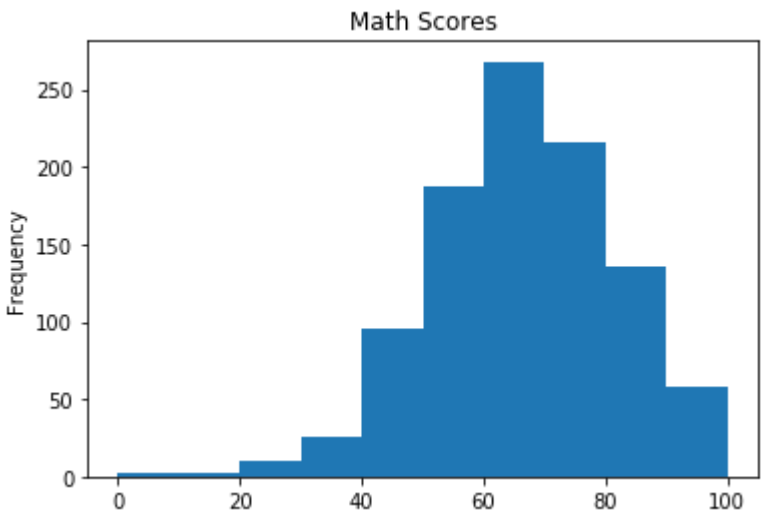


The same as explained above, most of the people didn´t take a course but there is still a significant amount of people that have taken them.

The following continous features have alredy been 'explained' above in data.describe()

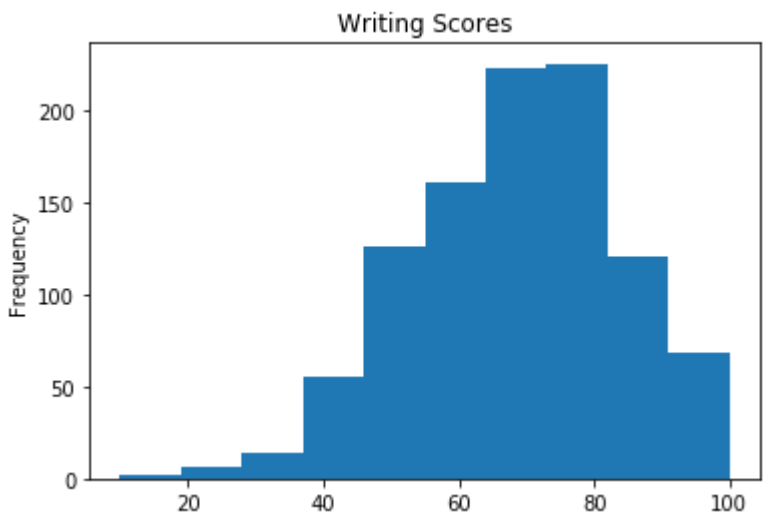
In [9]:
```
data['math score'].plot(kind='hist',title='Math Scores')
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26f620f33c8>
```



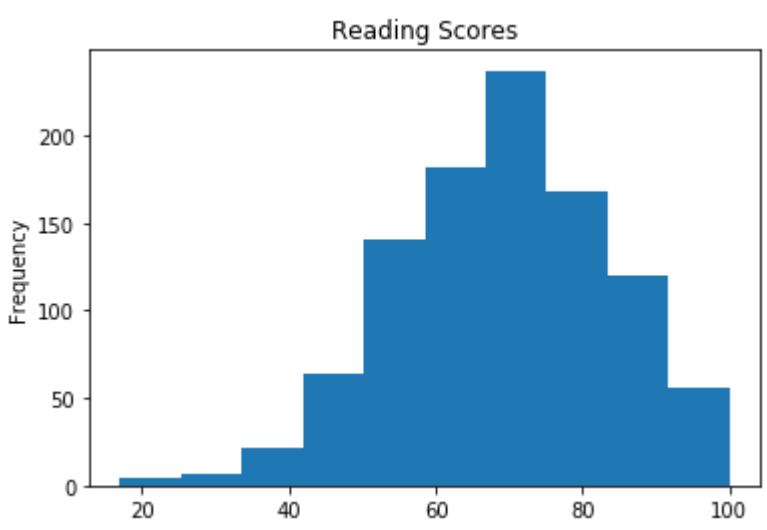
In [10]:
```
data['writing score'].plot(kind='hist',title='Writing Scores')
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x26f62184c08>



In [11]:
```
data['reading score'].plot(kind='hist',title='Reading Scores')
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x26f621fa748>



Cleaning and Preprocessing

we have 5 categorical features (gender,race/ethnicity,parental level of education,lunch,test preparation course) and 3 numeric/continous features (math score,reading score,writing score) so to facilitate the process to create models we are going to onehot encode the categorical features and normalize the continous features and putting all together in a temproal dataframe

In [12]:
```
new_data=pd.DataFrame()
```

Gender

it is our first categorical variable

In [13]:
```
print('Gender categories: ',data['gender'].unique())
```

```
Gender categories:  ['female' 'male']
```

Since gender is a categorical feature we have to onehot enconde it using LabelBinarizer from sklearn (female,male)->R^1

In [14]:
```
genderLabelBin=LabelBinarizer().fit(data['gender'].unique())
new_data['gender']=[genderLabelBin.transform([g])[0][0] for g in data['gender']]
new_data.head()
```

Out[14]:

	gender
0	0
1	0
2	0
3	1
4	1

Race/ethnic group

our second categorical feature

In [15]:
```
print('race/ethnicity categories: ',data['race/ethnicity'].unique())
```

```
race/ethnicity categories:  ['group B' 'group C' 'group A' 'group D' 'group E']
```

LabelBinarizer from sklearn ('group B' 'group C' 'group A' 'group D' 'group E')->R^5

In [16]:
```
raceLabelBin=LabelBinarizer().fit(data['race/ethnicity'].unique())
new_data['race']=[raceLabelBin.transform([r])[0]for r in data['race/ethnicity']]
new_data['group A']=[r[0] for r in new_data['race']]
new_data['group B']=[r[1] for r in new_data['race']]
new_data['group C']=[r[2] for r in new_data['race']]
new_data['group D']=[r[3] for r in new_data['race']]
new_data['group E']=[r[4] for r in new_data['race']]
del new_data['race']
new_data.head()
```

Out[16]:

	gender	group A	group B	group C	group D	group E
0	0	0	1	0	0	0
1	0	0	0	1	0	0
2	0	0	1	0	0	0
3	1	1	0	0	0	0
4	1	0	0	1	0	0

Parental Leve of education

In [17]:
```
print('Parental Level of Education: ',data['parental level of education'].unique())
```

```
Parental Level of Education:  ["bachelor's degree" 'some college' "master's degree" "associate's degree"
 'high school' 'some high school']
```

LabelBinarizer from sklearn ('group B' 'group C' 'group A' 'group D' 'group E')->R^6

In [18]:
```
parentLabelBin=LabelBinarizer().fit(data['parental level of education'].unique())
new_data['parental level of education']=[parentLabelBin.transform([r])[0] for r in data['parental level of ed
ucation']]
new_data['associate degree']=[p[0] for p in new_data['parental level of education']]
new_data['bachelor degree']=[p[1] for p in new_data['parental level of education']]
new_data['high school']=[p[2] for p in new_data['parental level of education']]
new_data['master degree']=[p[3] for p in new_data['parental level of education']]
new_data['some college']=[p[4] for p in new_data['parental level of education']]
new_data['some high school']=[p[5] for p in new_data['parental level of education']]
del new_data['parental level of education']
new_data.head()
```

Out[18]:

	gender	group A	group B	group C	group D	group E	associate degree	bachelor degree	high school	master degree	some college	some high school
0	0	0	1	0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	1	0
2	0	0	1	0	0	0	0	0	0	1	0	0
3	1	1	0	0	0	0	1	0	0	0	0	0
4	1	0	0	1	0	0	0	0	0	0	1	0

Lunch

In [19]:
```
print('Lunch: ',data['lunch'].unique())
```

```
Lunch:  ['standard' 'free/reduced']
```

In [20]:
```
lunchLabelBin=LabelBinarizer().fit(data['lunch'].unique())
new_data['lunch']=lunchLabelBin.transform(data['lunch'])
new_data.head()
```

Out[20]:

	gender	group A	group B	group C	group D	group E	associate degree	bachelor degree	high school	master degree	some college	some high school	lunch
0	0	0	1	0	0	0	0	1	0	0	0	0	1
1	0	0	0	1	0	0	0	0	0	0	1	0	1
2	0	0	1	0	0	0	0	0	0	1	0	0	1
3	1	1	0	0	0	0	1	0	0	0	0	0	0
4	1	0	0	1	0	0	0	0	0	0	1	0	1

Test preparation course

In [21]:
```
print('test preparation course: ',data['test preparation course'].unique())
```

```
test preparation course:  ['none' 'completed']
```

In [22]:
```
testLabelBin=LabelBinarizer().fit(data['test preparation course'].unique())
new_data['test preparation course']=testLabelBin.transform(data['test preparation course'])
new_data.head()
```

Out[22]:

	gender	group A	group B	group C	group D	group E	associate degree	bachelor degree	high school	master degree	some college	some high school	lunch	test preparation course
0	0	0	1	0	0	0	0	1	0	0	0	0	1	1
1	0	0	0	1	0	0	0	0	0	0	1	0	1	0
2	0	0	1	0	0	0	0	0	0	1	0	0	1	1
3	1	1	0	0	0	0	1	0	0	0	0	0	0	1
4	1	0	0	1	0	0	0	0	0	0	1	0	1	1

Normalizing continous features

In [23]:
```
new_data['math score']=data['math score']/100
new_data['reading score']=data['reading score']/100
new_data['writing score']=data['writing score']/100
new_data['average score']=[(new_data['math score'].iloc[i]+new_data['writing score'].iloc[i]+new_data['readin
g score'].iloc[i])/3 for i in range(len(new_data))]
new_data.head()
```

Out[23]:

	gender	group A	group B	group C	group D	group E	associate degree	bachelor degree	high school	master degree	some college	some high school	lunch	test preparation course	math score	readi sco
0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	0.72	0.
1	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0.69	0.
2	0	0	1	0	0	0	0	0	0	1	0	0	1	1	0.90	0.
3	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0.47	0.
4	1	0	0	1	0	0	0	0	0	0	1	0	1	1	0.76	0.

Now the dataset is splitted into labels that we can use as features for our model and are normalized

Analysis after processing and cleaning data

We are going to se the disrtibutions of scores corresponding to different features

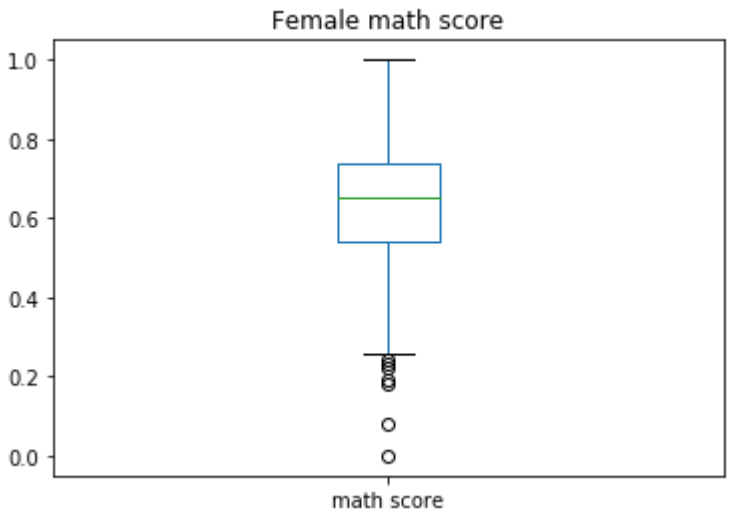
Gender

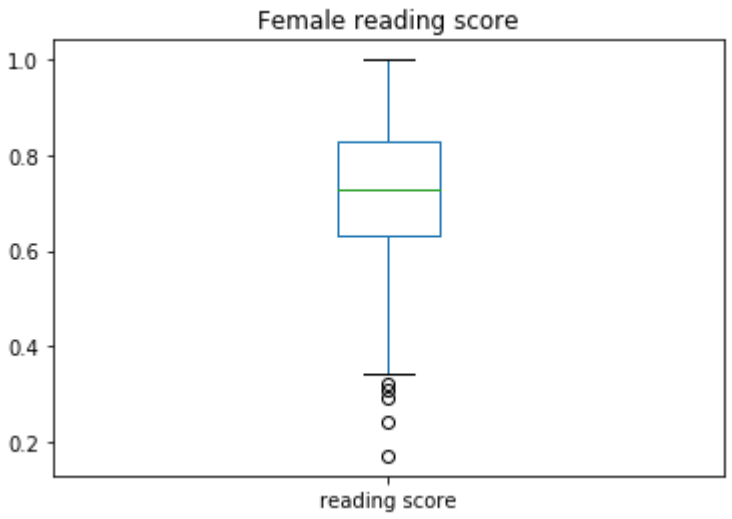
Female

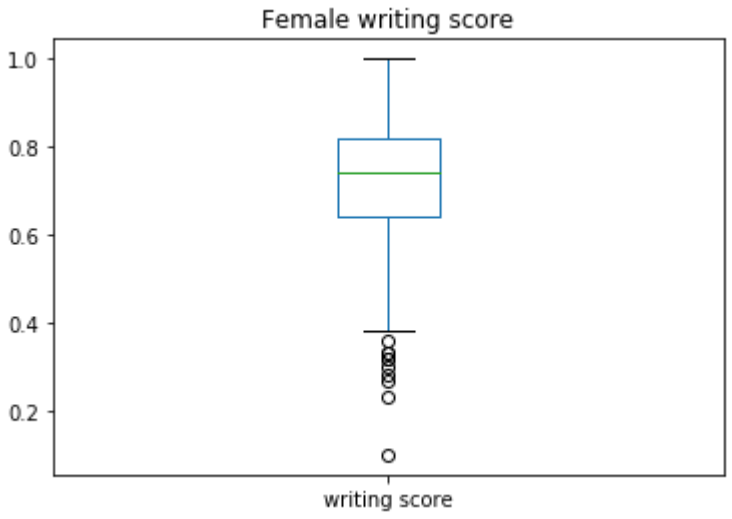
In [24]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['gender']==0).dropna().plot(y=c,kind='box',title='Female '+c)
new_data[new_data.columns[-4:]].where(new_data['gender']==0).dropna().describe()
```

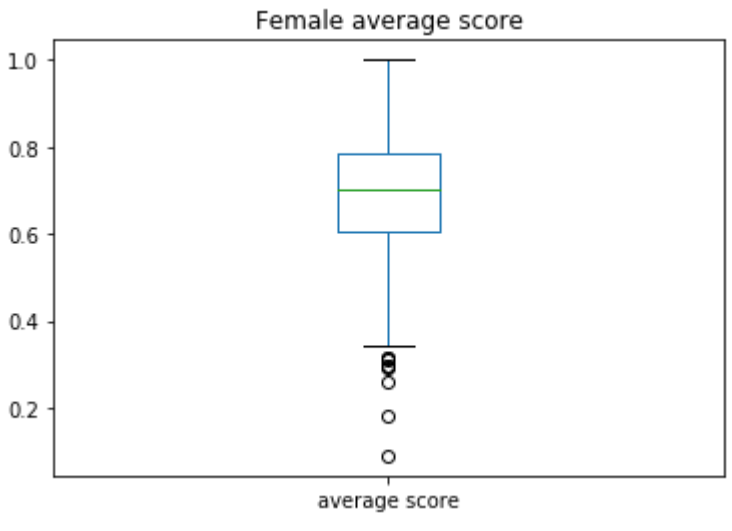
Out[24]:

	math score	reading score	writing score	average score
count	518.000000	518.000000	518.000000	518.000000
mean	0.636332	0.726081	0.724672	0.695695
std	0.154915	0.143782	0.148448	0.145418
min	0.000000	0.170000	0.100000	0.090000
25%	0.540000	0.632500	0.640000	0.606667
50%	0.650000	0.730000	0.740000	0.703333
75%	0.740000	0.830000	0.820000	0.786667
max	1.000000	1.000000	1.000000	1.000000







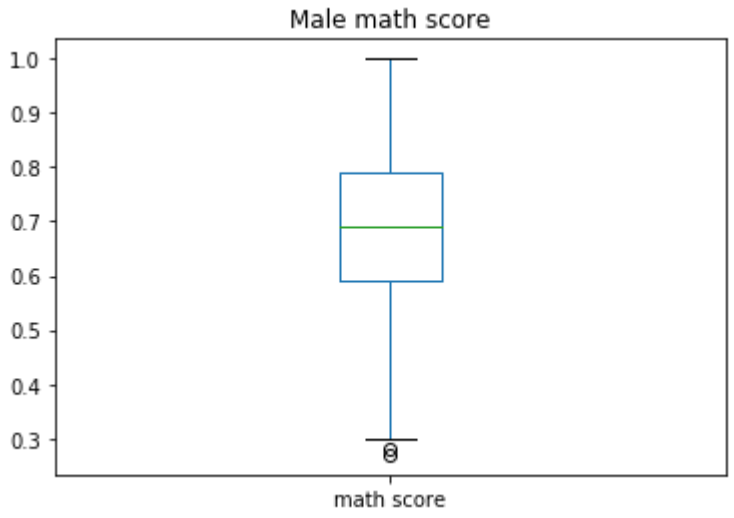


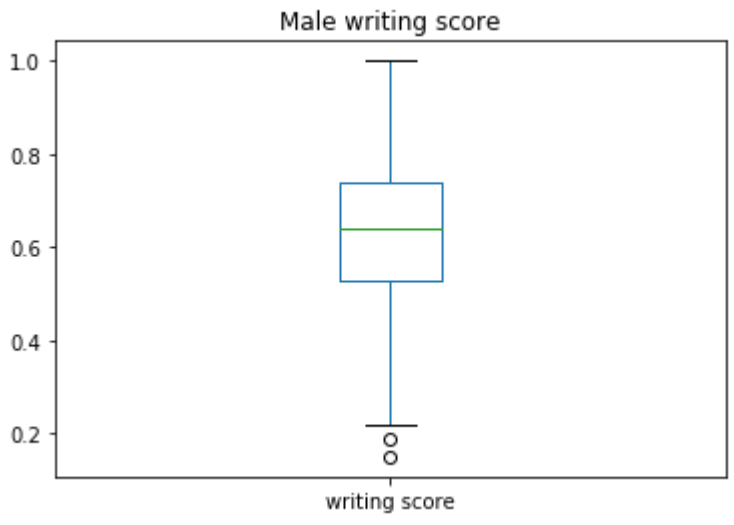
Males

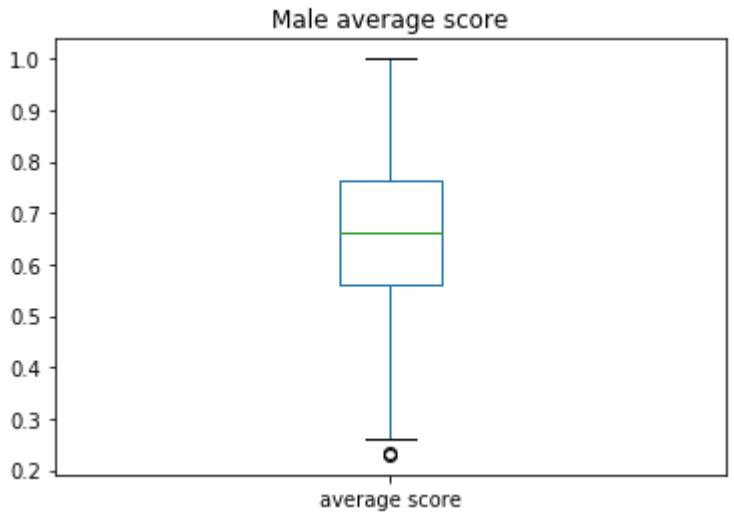
In [25]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['gender']==1).dropna().plot(y=c,kind='box',title='Male '+c)
new_data[new_data.columns[-4:]].where(new_data['gender']==1).dropna().describe()
```

Out[25]:

	math score	reading score	writing score	average score
count	482.000000	482.000000	482.000000	482.000000
mean	0.687282	0.654730	0.633112	0.658375
std	0.143563	0.139318	0.141138	0.136988
min	0.270000	0.230000	0.150000	0.230000
25%	0.590000	0.560000	0.530000	0.560000
50%	0.690000	0.660000	0.640000	0.663333
75%	0.790000	0.750000	0.737500	0.762500
max	1.000000	1.000000	1.000000	1.000000







In [26]:
```
new_data[['gender']+new_data.columns[-4:].tolist()].corr()[new_data.columns[-4:]].iloc[0]
```

Out[26]:
```
math score       0.167982
reading score   -0.244313
writing score   -0.301225
average score   -0.130861
Name: gender, dtype: float64
```

We can see in the 6 graphs above that the distributions for the scores depending on gender are quite different. In the case of females, there are more outliers in the lower bound in all the scores. Nevertheless females actually perform, on average, better than men by 4%.

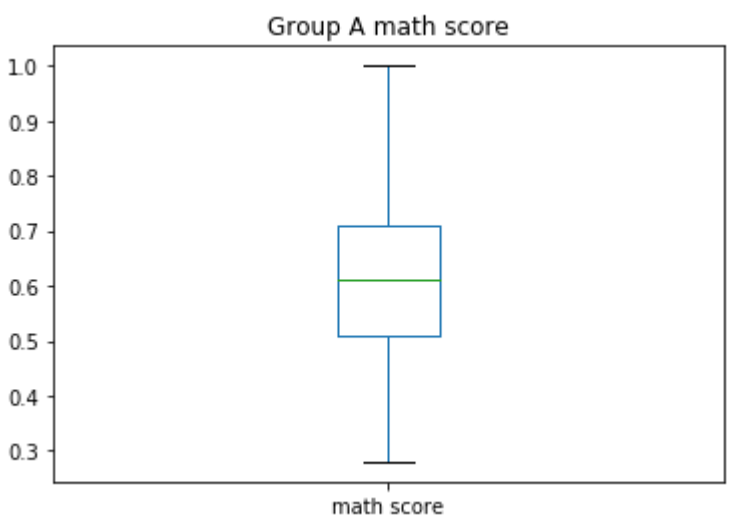
Groups

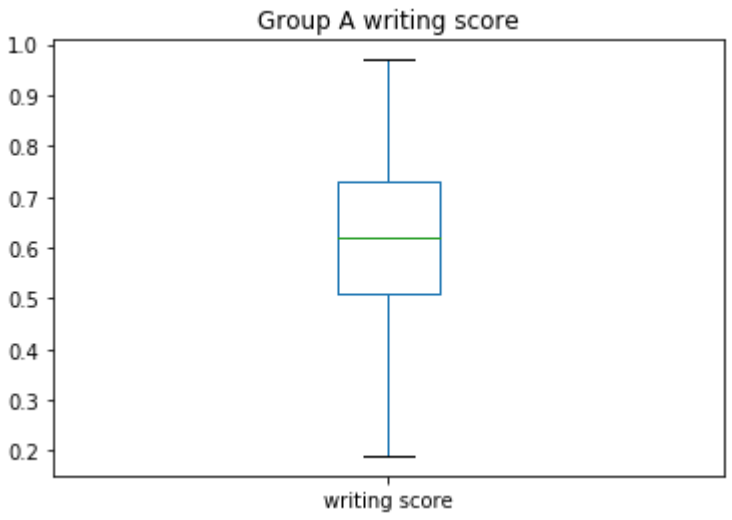
Group A

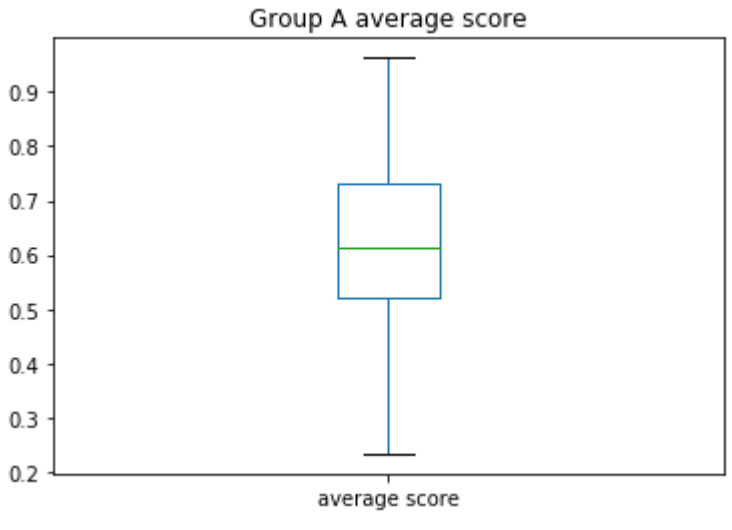
In [27]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['group A']==1).dropna().plot(y=c,kind='box',title='Group A '+c)
new_data[new_data.columns[-4:]].where(new_data['group A']==1).dropna().describe()
```

Out[27]:

	math score	reading score	writing score	average score
count	89.000000	89.000000	89.000000	89.000000
mean	0.616292	0.646742	0.626742	0.629925
std	0.145230	0.155438	0.154683	0.144446
min	0.280000	0.230000	0.190000	0.233333
25%	0.510000	0.530000	0.510000	0.520000
50%	0.610000	0.640000	0.620000	0.613333
75%	0.710000	0.740000	0.730000	0.730000
max	1.000000	1.000000	0.970000	0.963333





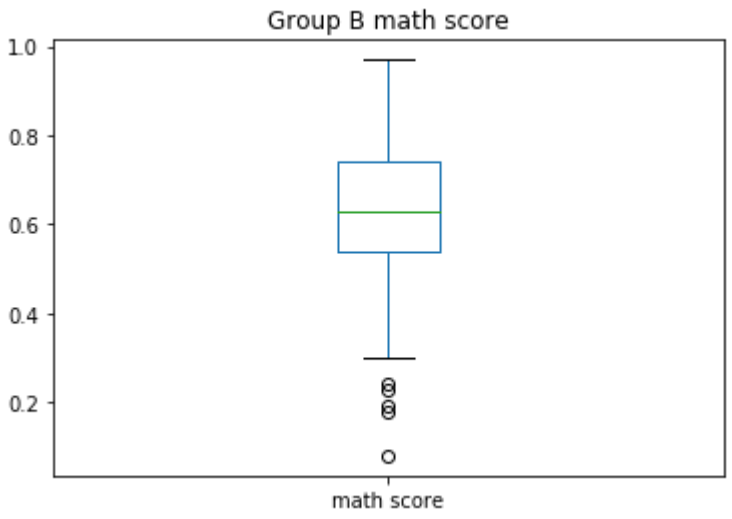


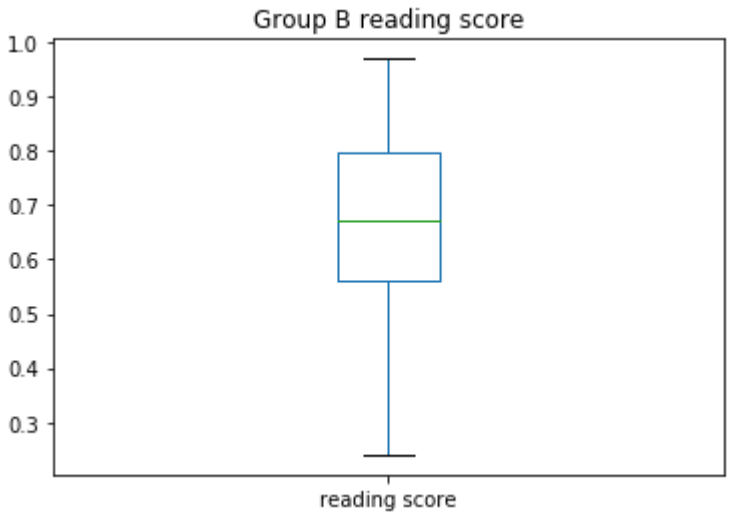
Group B

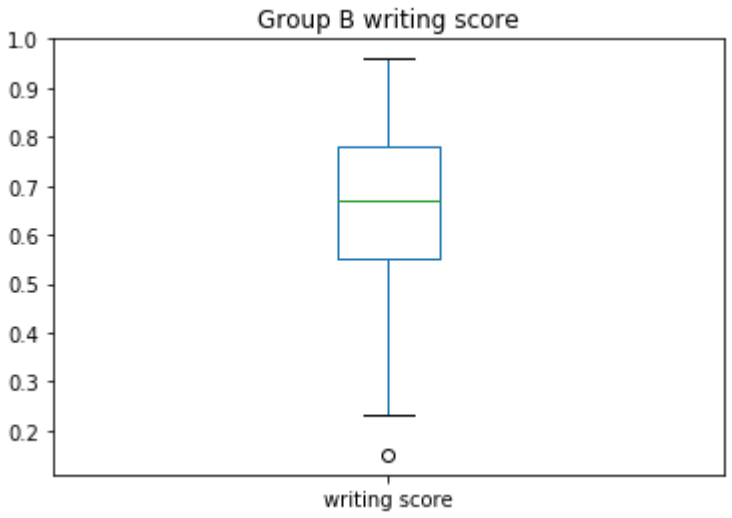
In [28]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['group B']==1).dropna().plot(y=c,kind='box',title='Group B '+c)
new_data[new_data.columns[-4:]].where(new_data['group B']==1).dropna().describe()
```

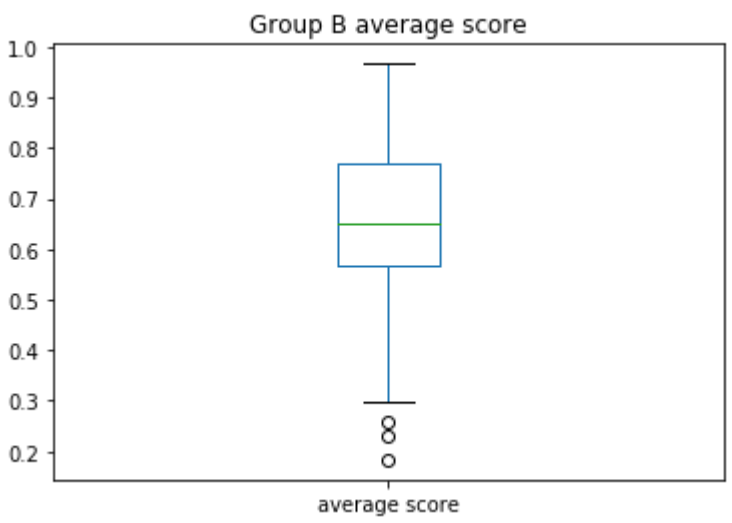
Out[28]:

	math score	reading score	writing score	average score
count	190.000000	190.000000	190.000000	190.000000
mean	0.634526	0.673526	0.656000	0.654684
std	0.154682	0.151775	0.156252	0.147321
min	0.080000	0.240000	0.150000	0.183333
25%	0.540000	0.560000	0.552500	0.566667
50%	0.630000	0.670000	0.670000	0.650000
75%	0.740000	0.797500	0.780000	0.768333
max	0.970000	0.970000	0.960000	0.966667







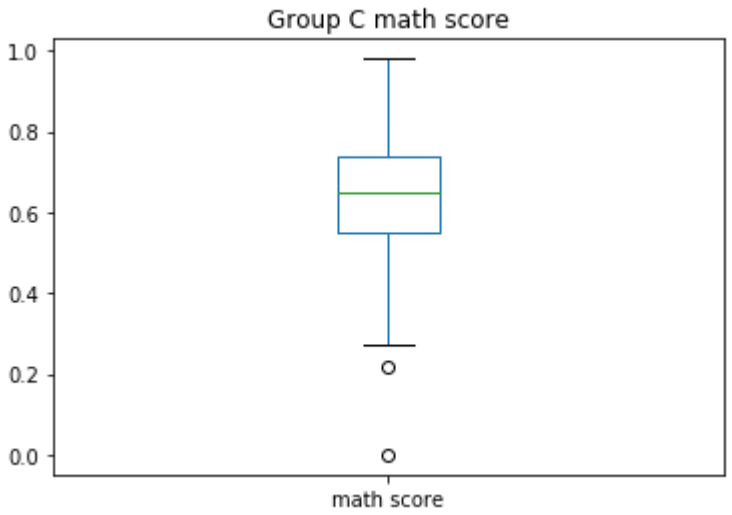


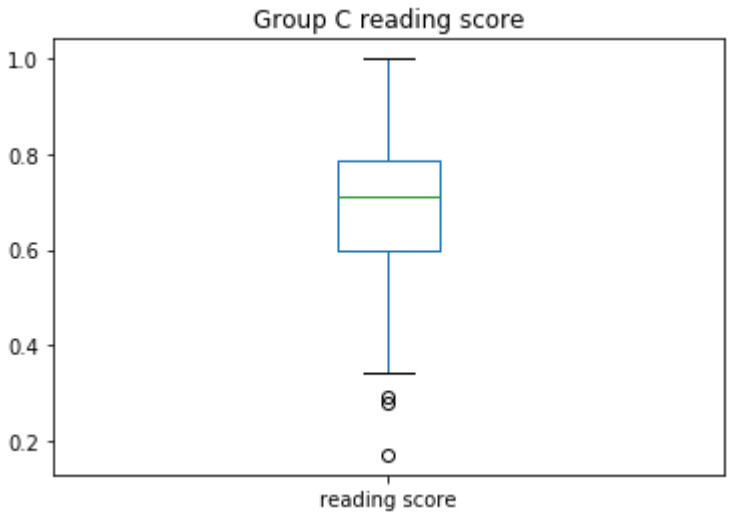
Group C

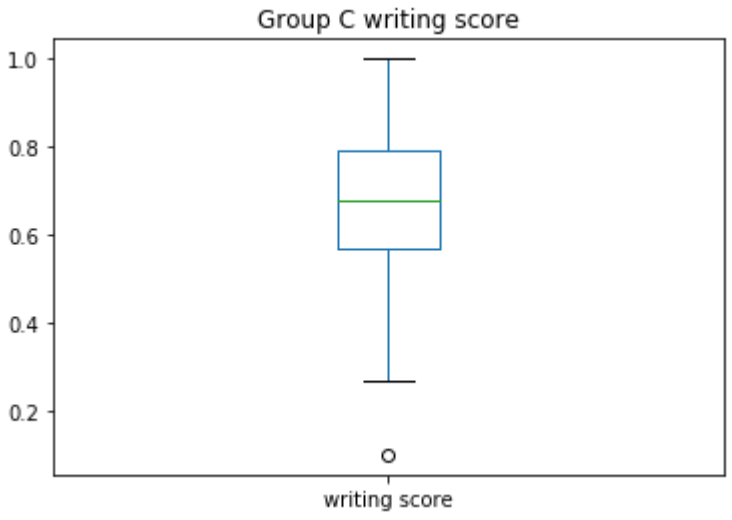
In [29]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['group C']==1).dropna().plot(y=c,kind='box',title='Group C '+c)
new_data[new_data.columns[-4:]].where(new_data['group C']==1).dropna().describe()
```

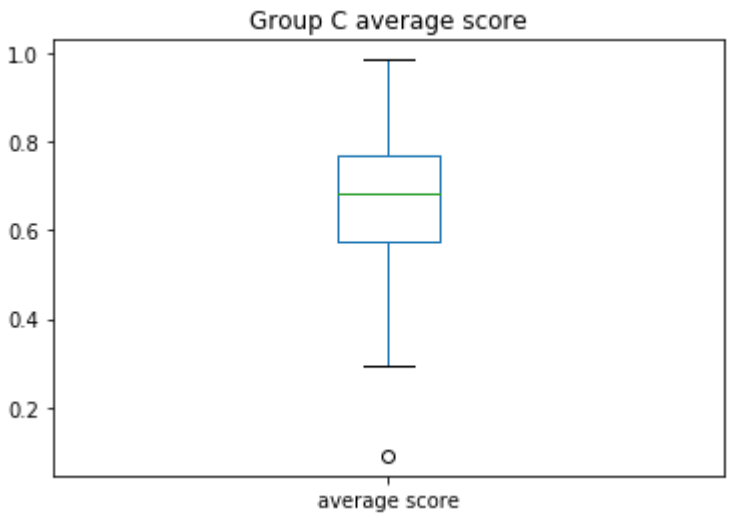
Out[29]:

	math score	reading score	writing score	average score
count	319.000000	319.000000	319.000000	319.000000
mean	0.644639	0.691034	0.678276	0.671317
std	0.148527	0.139970	0.149834	0.138722
min	0.000000	0.170000	0.100000	0.090000
25%	0.550000	0.600000	0.570000	0.576667
50%	0.650000	0.710000	0.680000	0.683333
75%	0.740000	0.785000	0.790000	0.770000
max	0.980000	1.000000	1.000000	0.986667







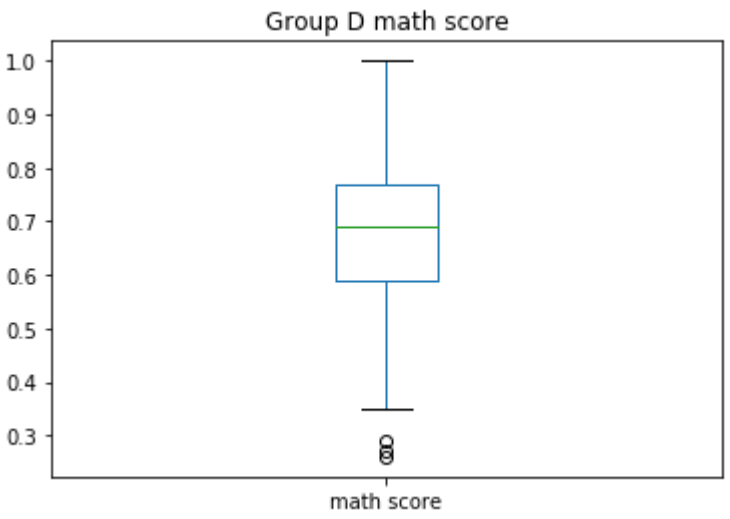


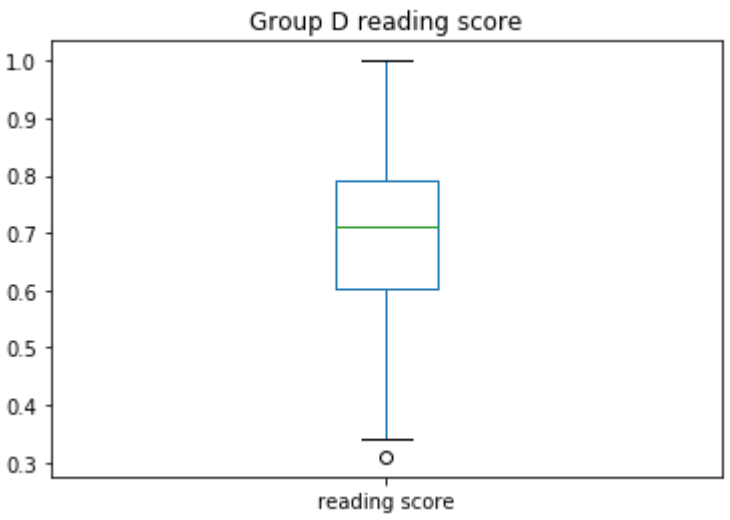
Group D

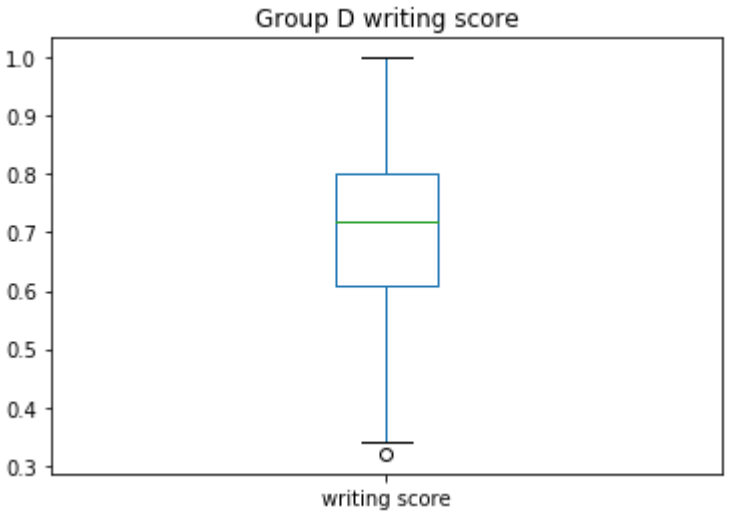
In [30]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['group D']==1).dropna().plot(y=c,kind='box',title='Group D '+c)
new_data[new_data.columns[-4:]].where(new_data['group D']==1).dropna().describe()
```

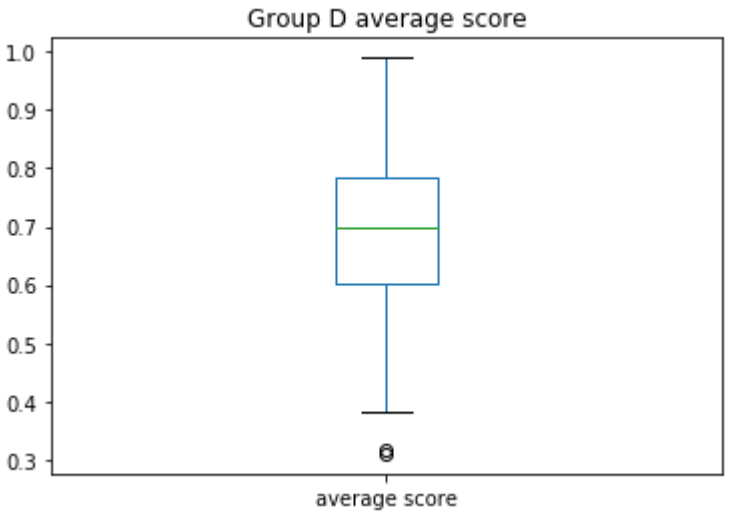
Out[30]:

	math score	reading score	writing score	average score
count	262.000000	262.000000	262.000000	262.000000
mean	0.673626	0.700305	0.701450	0.691794
std	0.137694	0.138953	0.143677	0.132528
min	0.260000	0.310000	0.320000	0.310000
25%	0.590000	0.602500	0.610000	0.603333
50%	0.690000	0.710000	0.720000	0.700000
75%	0.770000	0.790000	0.800000	0.785833
max	1.000000	1.000000	1.000000	0.990000







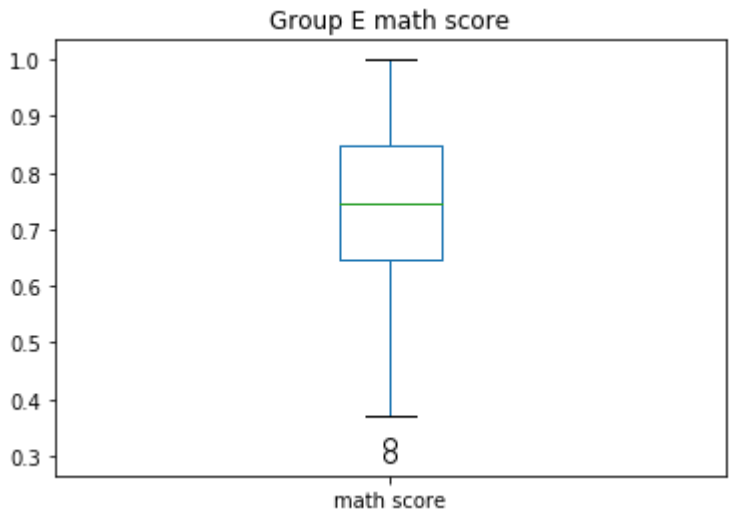


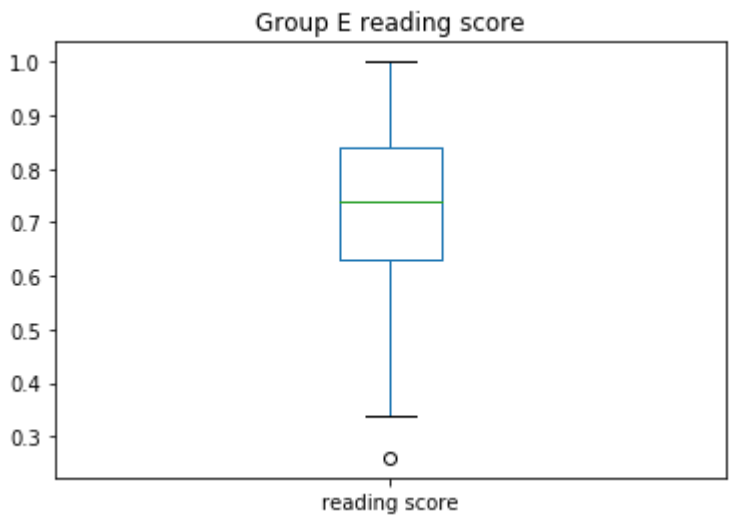
Group E

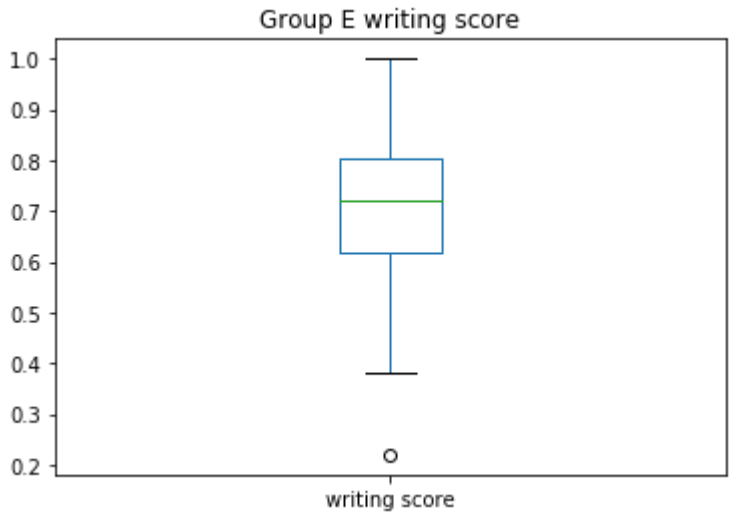
In [31]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['group E']==1).dropna().plot(y=c,kind='box',title='Group E '+c)
new_data[new_data.columns[-4:]].where(new_data['group E']==1).dropna().describe()
```

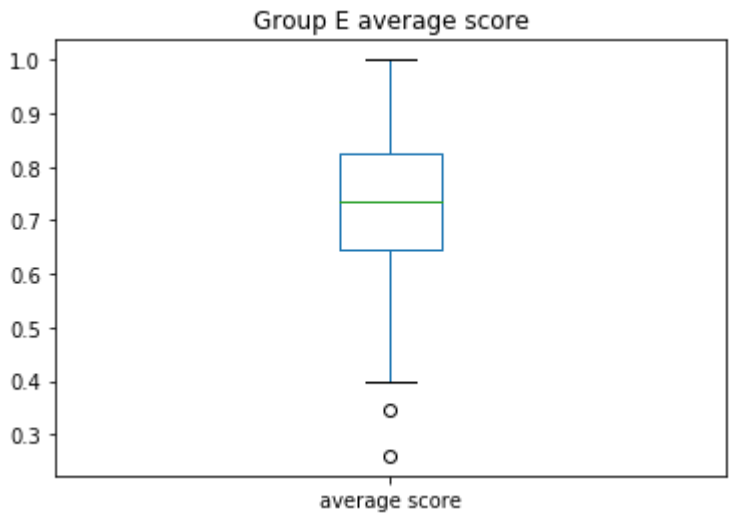
Out[31]:

	math score	reading score	writing score	average score
count	140.000000	140.000000	140.000000	140.000000
mean	0.738214	0.730286	0.714071	0.727524
std	0.155343	0.148740	0.151139	0.145650
min	0.300000	0.260000	0.220000	0.260000
25%	0.647500	0.630000	0.620000	0.646667
50%	0.745000	0.740000	0.720000	0.735000
75%	0.850000	0.840000	0.802500	0.824167
max	1.000000	1.000000	1.000000	1.000000









In [32]:
```
new_data[['group A','group B','group C','group D','group E']+new_data.columns[-4:].tolist()].corr()[new_data.columns[-4:]].iloc[:-4]
```

Out[32]:

	math score	reading score	writing score	average score
group A	-0.091977	-0.096274	-0.110714	-0.104803
group B	-0.084250	-0.060283	-0.078254	-0.078247
group C	-0.073387	-0.003074	-0.010203	-0.030691
group D	0.050071	0.035177	0.082032	0.058902
group E	0.205855	0.106712	0.089077	0.141050

In the case of groups we see many things. For example, the group performed best on average, has at least 1 one in each subject, few outliers, but didn´t had the smallest standard deviation of all. In another way the group that performed the worst on average, didn´t had any outliers and had at least 1 one in both reading and mathematics. Neverless 3 groups performed under average and the rest above average in the entire dataset

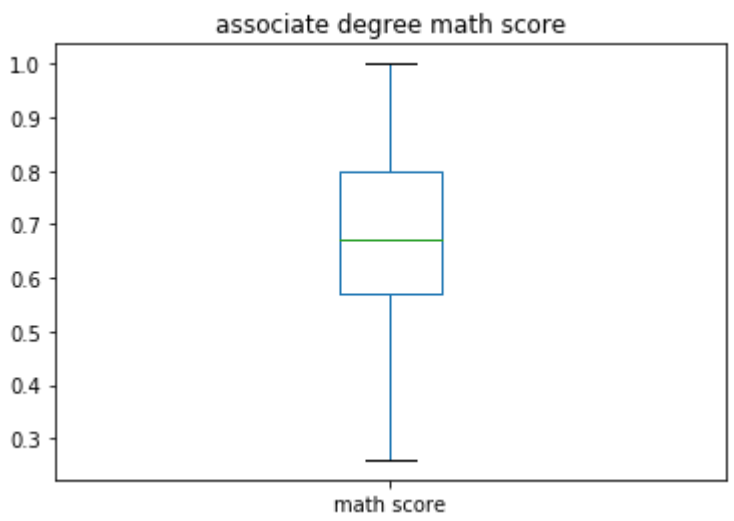
Parental Level Education

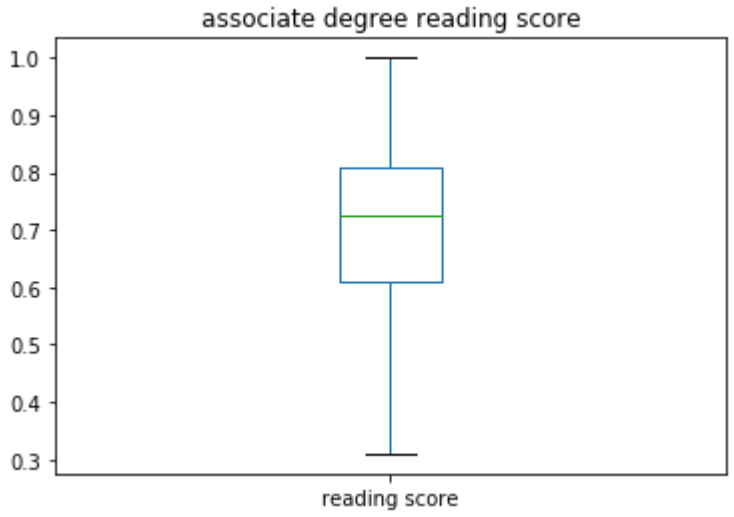
Associate Degree

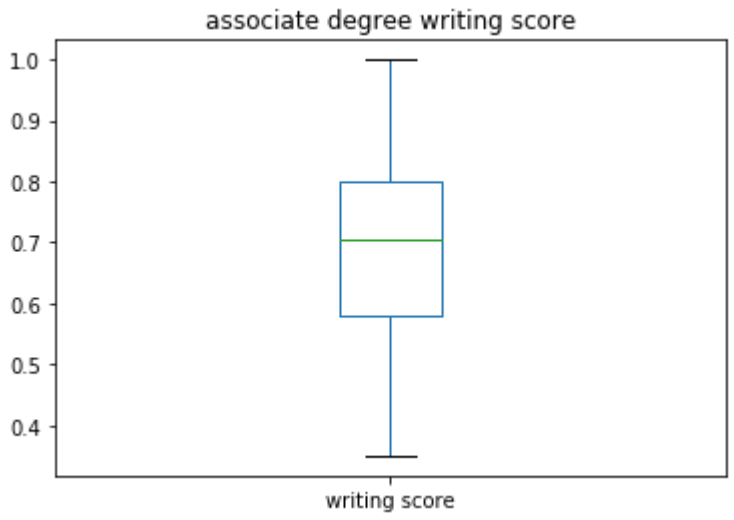
In [33]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['associate degree']==1).dropna().plot(y=c,kind='box',title='associate degree '+c)
new_data[new_data.columns[-4:]].where(new_data['associate degree']==1).dropna().describe()
```

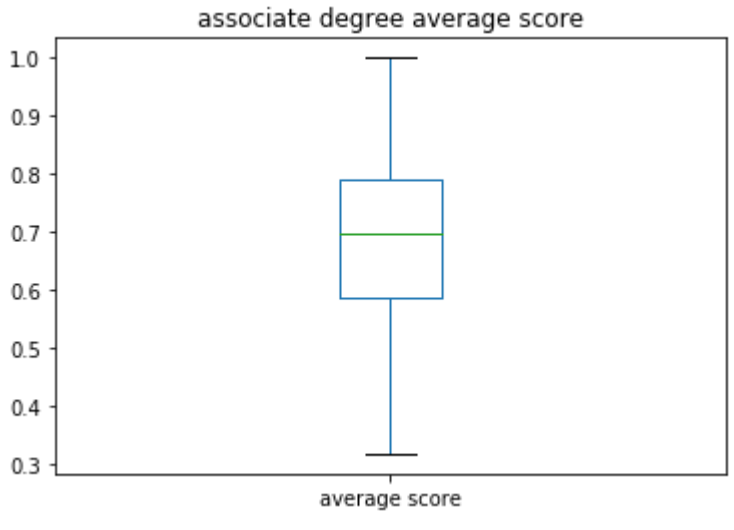
Out[33]:

	math score	reading score	writing score	average score
count	222.000000	222.000000	222.000000	222.000000
mean	0.678829	0.709279	0.698964	0.695691
std	0.151121	0.138689	0.143111	0.136709
min	0.260000	0.310000	0.350000	0.316667
25%	0.570000	0.610000	0.580000	0.586667
50%	0.670000	0.725000	0.705000	0.696667
75%	0.800000	0.810000	0.800000	0.790000
max	1.000000	1.000000	1.000000	1.000000







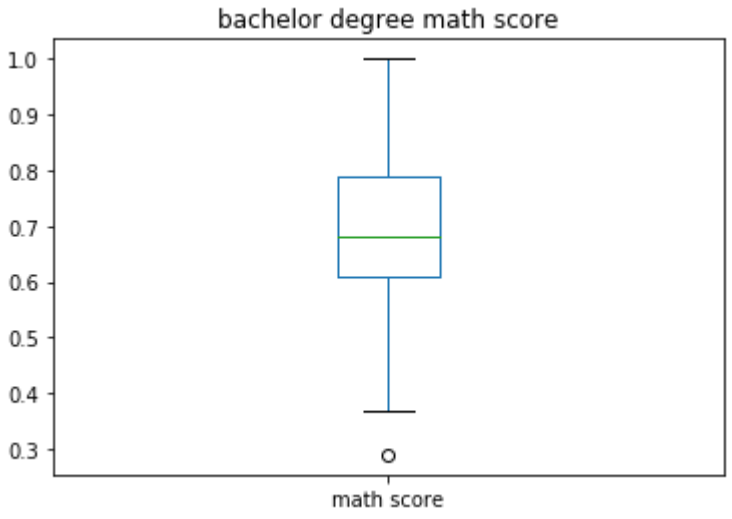


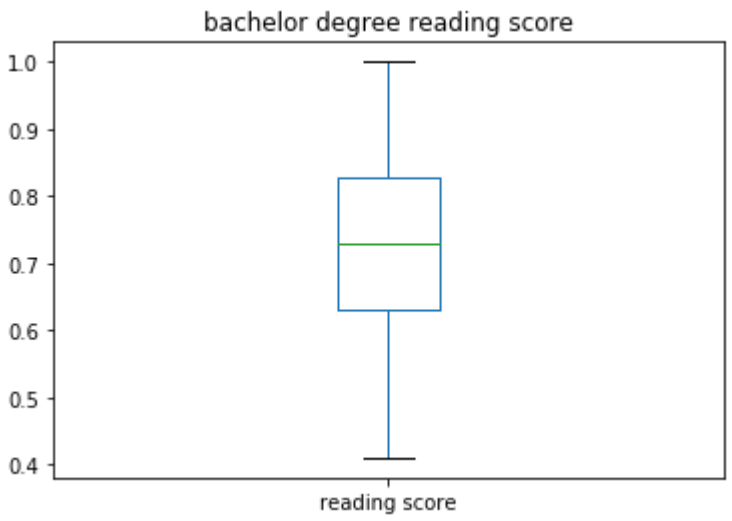
Bachelor Degree

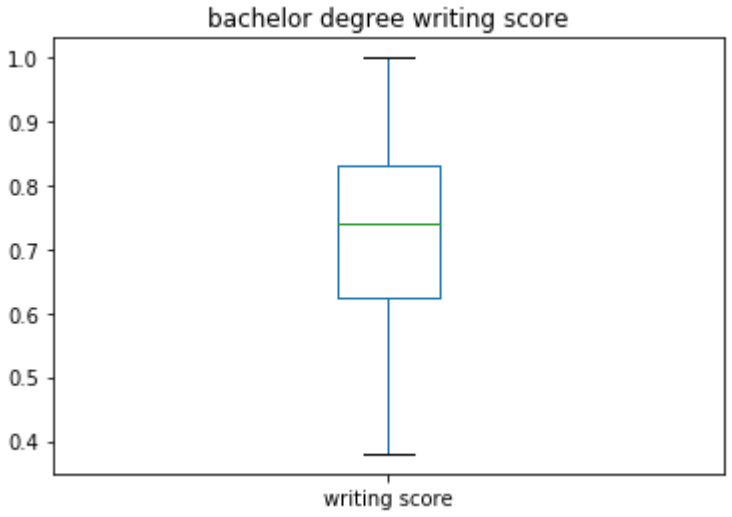
In [34]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['bachelor degree']==1).dropna().plot(y=c,kind='box',title='bachelor degree '+c)
new_data[new_data.columns[-4:]].where(new_data['bachelor degree']==1).dropna().describe()
```

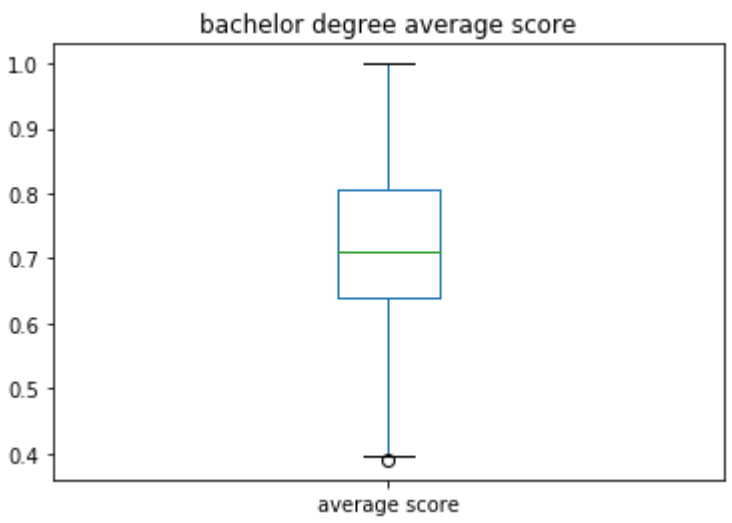
Out[34]:

	math score	reading score	writing score	average score
count	118.000000	118.000000	118.000000	118.000000
mean	0.693898	0.730000	0.733814	0.719237
std	0.149438	0.142853	0.147283	0.139466
min	0.290000	0.410000	0.380000	0.390000
25%	0.610000	0.630000	0.625000	0.640833
50%	0.680000	0.730000	0.740000	0.711667
75%	0.790000	0.827500	0.830000	0.806667
max	1.000000	1.000000	1.000000	1.000000







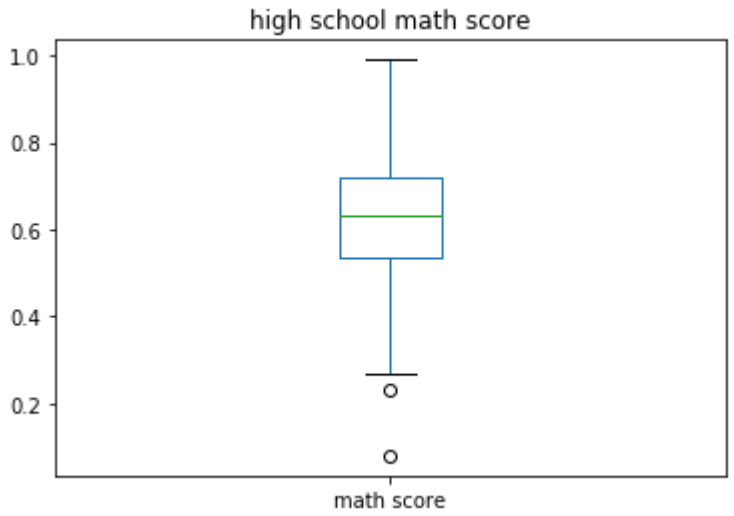


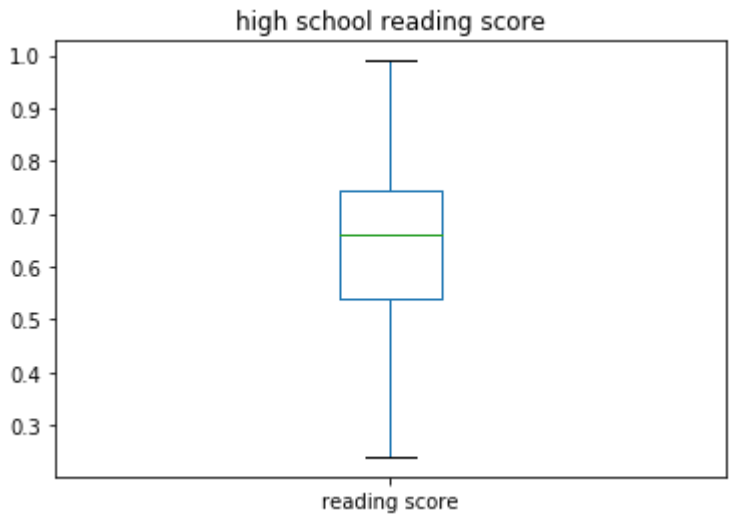
High School

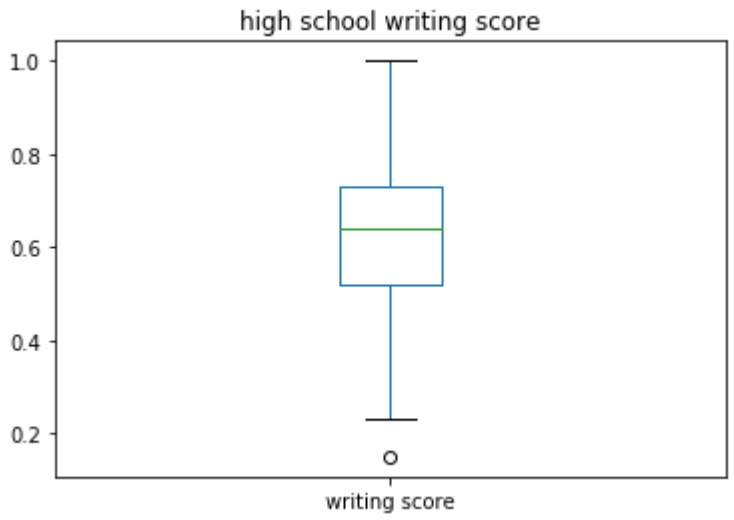
In [35]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['high school']==1).dropna().plot(y=c,kind='box',title='high school '+c)
new_data[new_data.columns[-4:]].where(new_data['high school']==1).dropna().describe()
```

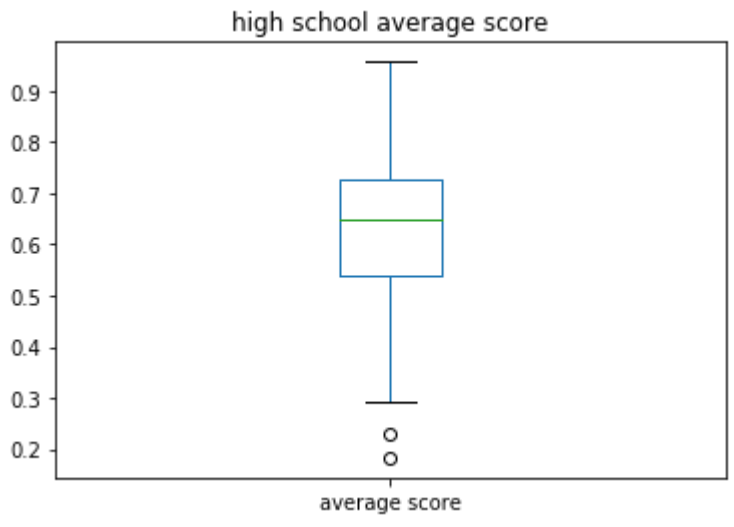
Out[35]:

	math score	reading score	writing score	average score
count	196.000000	196.000000	196.000000	196.000000
mean	0.621378	0.647041	0.624490	0.630969
std	0.145397	0.141321	0.140859	0.135106
min	0.080000	0.240000	0.150000	0.183333
25%	0.537500	0.540000	0.520000	0.539167
50%	0.630000	0.660000	0.640000	0.650000
75%	0.720000	0.742500	0.730000	0.726667
max	0.990000	0.990000	1.000000	0.956667







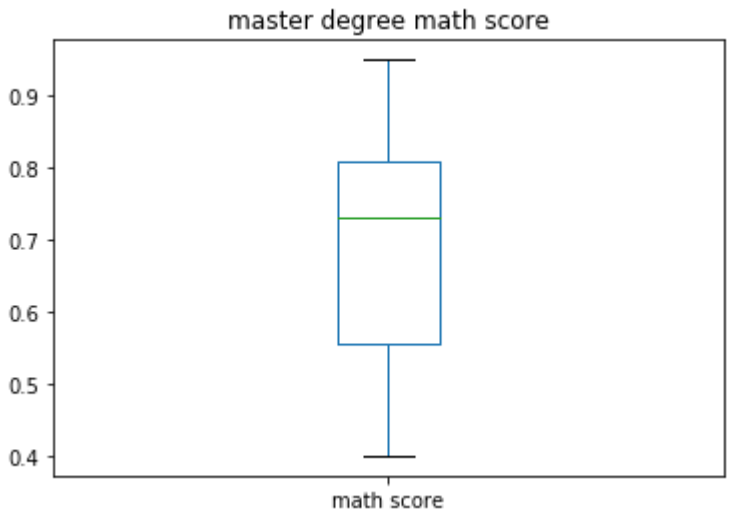


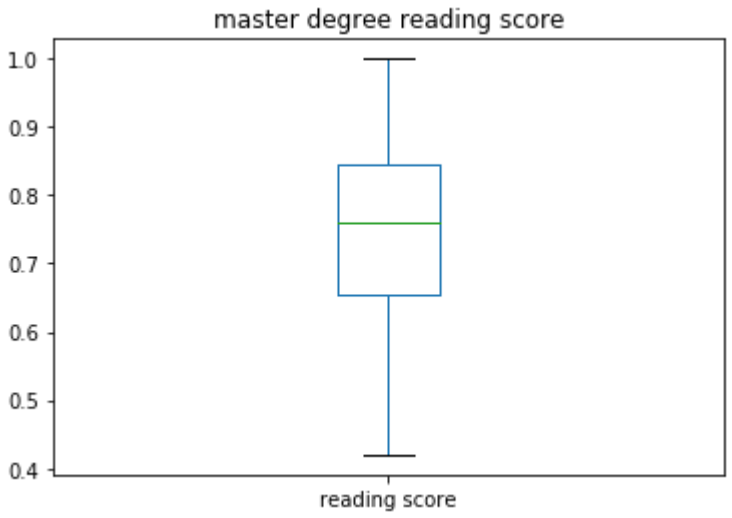
Master Degree

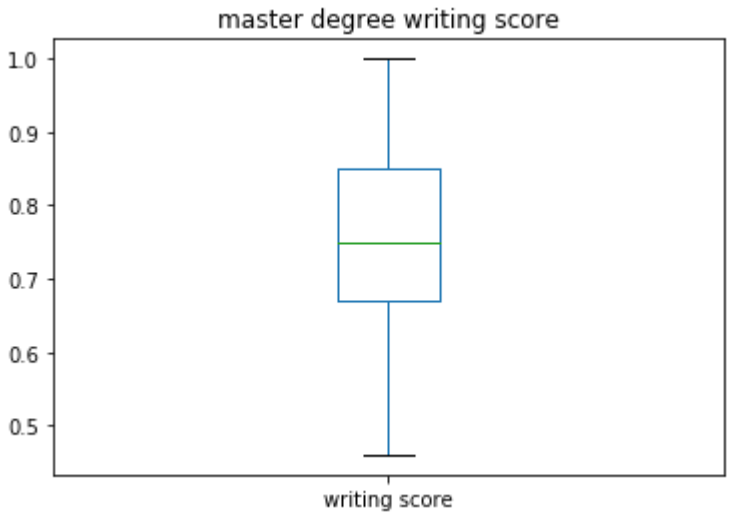
In [36]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['master degree']==1).dropna().plot(y=c,kind='box',title='master degree '+c)
new_data[new_data.columns[-4:]].where(new_data['master degree']==1).dropna().describe()
```

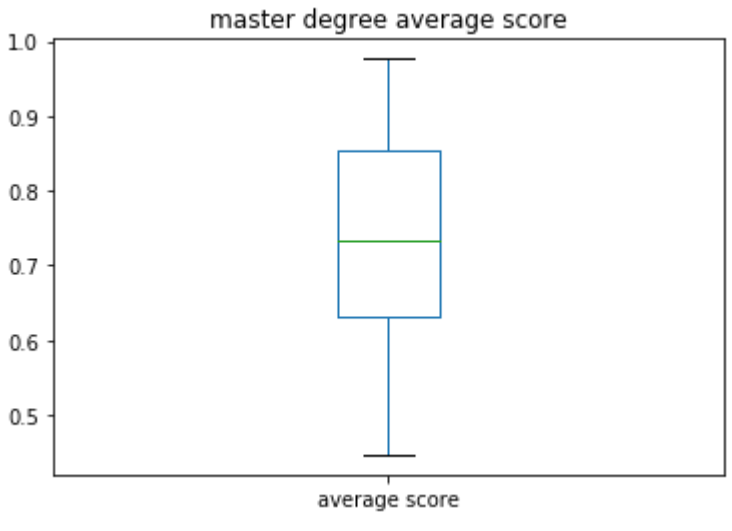
Out[36]:

	math score	reading score	writing score	average score
count	59.000000	59.000000	59.000000	59.000000
mean	0.697458	0.753729	0.756780	0.735989
std	0.151539	0.137752	0.137307	0.136010
min	0.400000	0.420000	0.460000	0.446667
25%	0.555000	0.655000	0.670000	0.631667
50%	0.730000	0.760000	0.750000	0.733333
75%	0.810000	0.845000	0.850000	0.855000
max	0.950000	1.000000	1.000000	0.976667







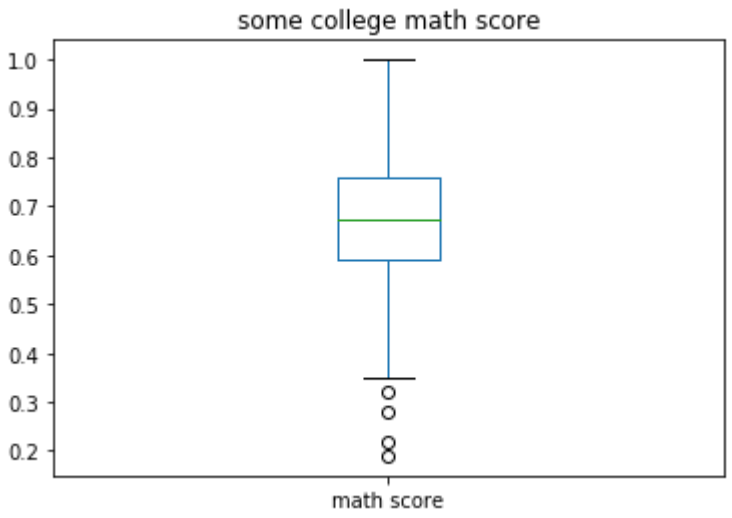


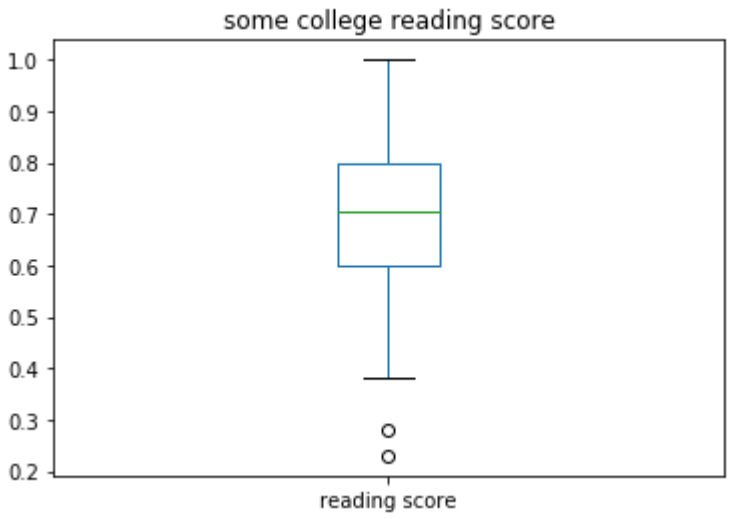
Some College

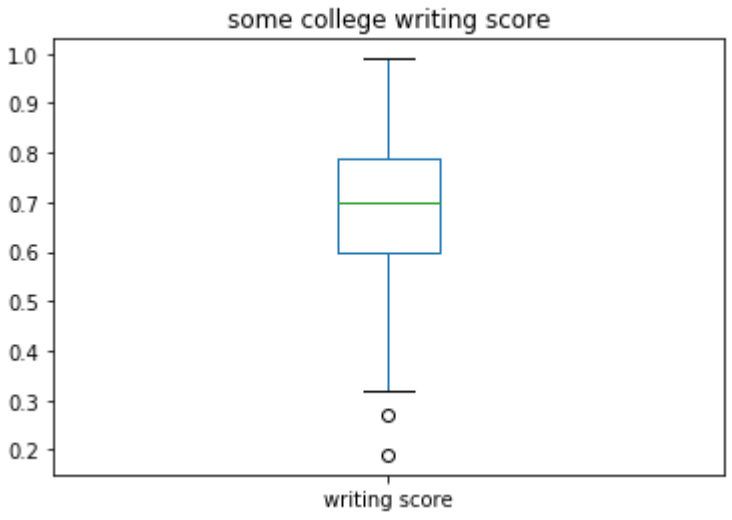
In [37]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['some college']==1).dropna().plot(y=c,kind='box',title='some college '+c)
new_data[new_data.columns[-4:]].where(new_data['some college']==1).dropna().describe()
```

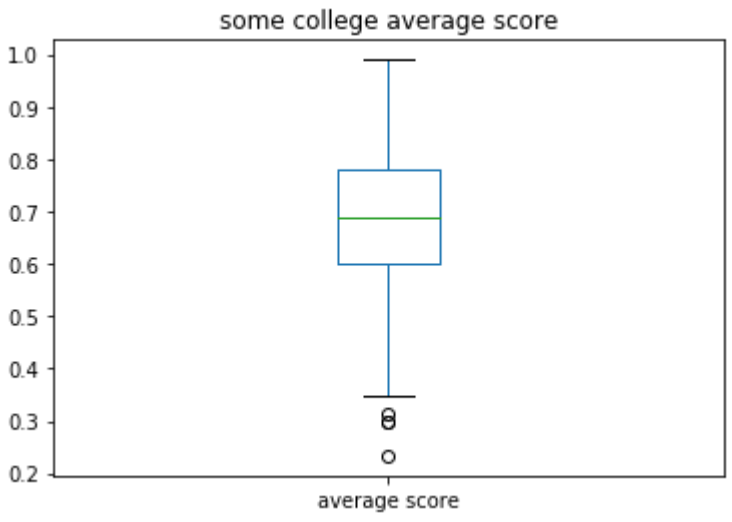
Out[37]:

	math score	reading score	writing score	average score
count	226.000000	226.000000	226.000000	226.000000
mean	0.671283	0.694602	0.688407	0.684764
std	0.143129	0.140570	0.150123	0.137110
min	0.190000	0.230000	0.190000	0.233333
25%	0.590000	0.600000	0.600000	0.600000
50%	0.675000	0.705000	0.700000	0.686667
75%	0.760000	0.797500	0.790000	0.780000
max	1.000000	1.000000	0.990000	0.990000







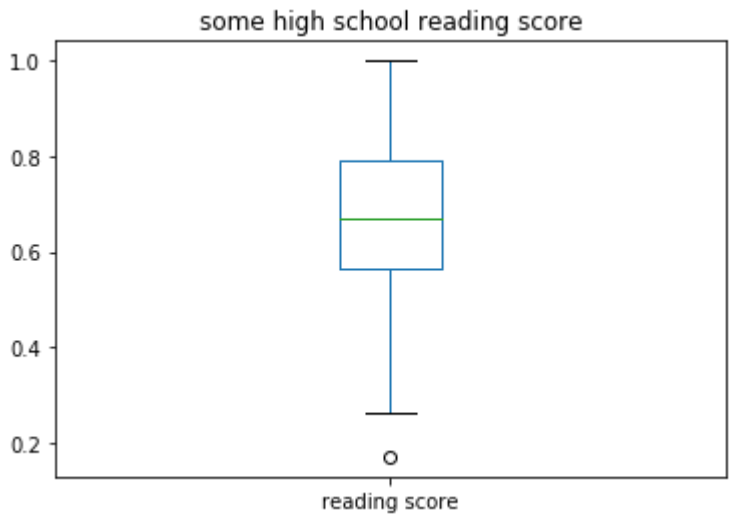


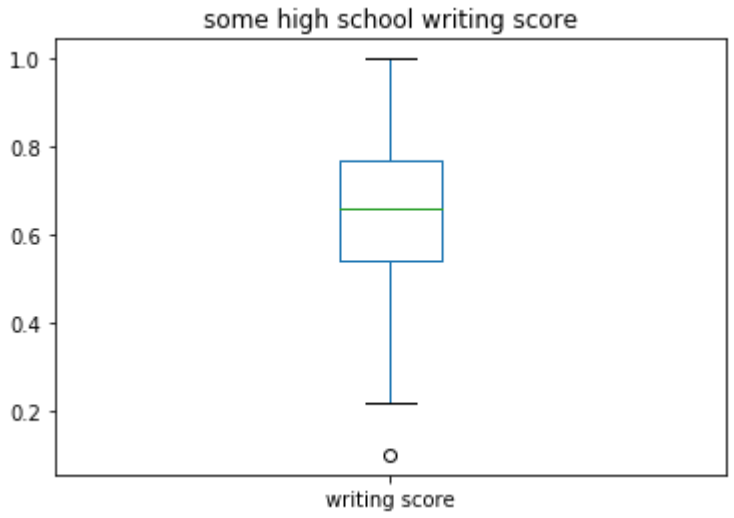
Some High School

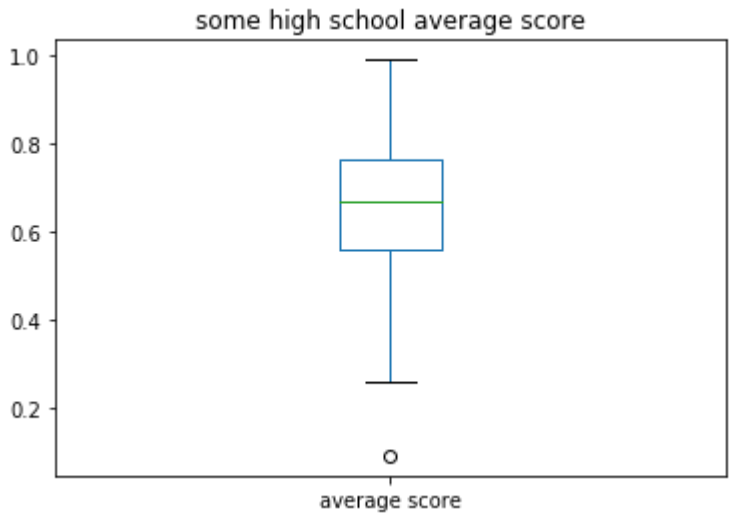
In [38]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['some high school']==1).dropna().plot(y=c,kind='box',title='some high school '+c)
new_data[new_data.columns[-4:]].where(new_data['some high school']==1).dropna().describe()
```

Out[38]:

	math score	reading score	writing score	average score
count	179.000000	179.000000	179.000000	179.000000
mean	0.634972	0.669385	0.648883	0.651080
std	0.159280	0.154793	0.157362	0.149841
min	0.000000	0.170000	0.100000	0.090000
25%	0.530000	0.565000	0.540000	0.556667
50%	0.650000	0.670000	0.660000	0.666667
75%	0.740000	0.790000	0.770000	0.765000
max	0.970000	1.000000	1.000000	0.990000







In [39]:
```
new_data[['associate degree', 'bachelor degree', 'high school',
          'master degree','some college',
          'some high school']+new_data.columns[-4:].tolist()].corr()[new_data.columns[-4:]].iloc[:-4]
```

Out[39]:

	math score	reading score	writing score	average score
associate degree	0.063228	0.064386	0.064799	0.067414
bachelor degree	0.079664	0.096024	0.128297	0.106599
high school	-0.128725	-0.151068	-0.182211	-0.161936
master degree	0.060417	0.106452	0.125693	0.102411
some college	0.037056	0.010782	0.027989	0.026761
some high school	-0.079852	-0.071369	-0.097326	-0.087247

We can see that the parents education can actually affect the mean scores of their kids, the better education they had it would be reflected positively on their kids.

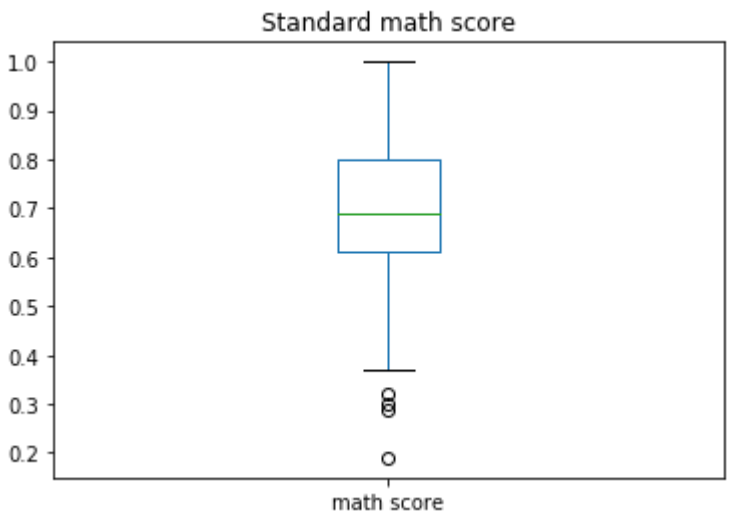
Lunch

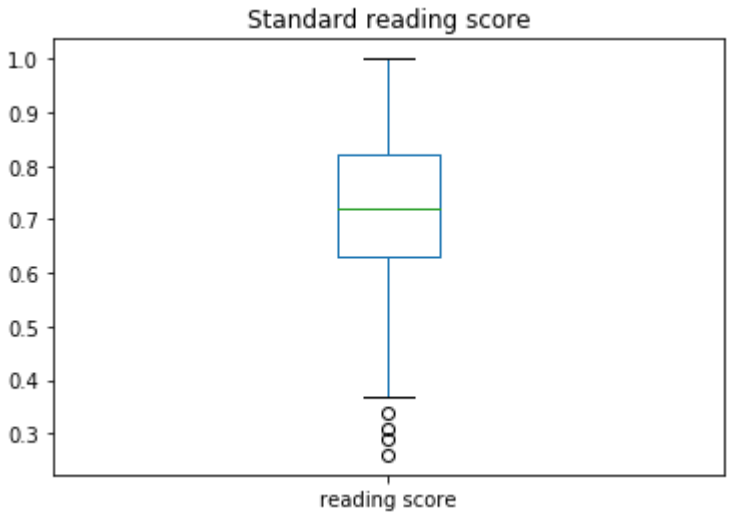
Standard

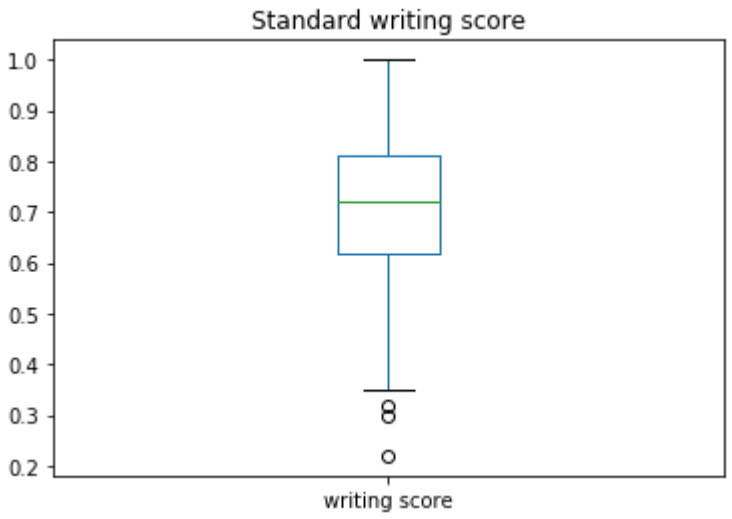
In [40]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['lunch']==1).dropna().plot(y=c,kind='box',title='Standard '+c)
new_data[new_data.columns[-4:]].where(new_data['lunch']==1).dropna().describe()
```

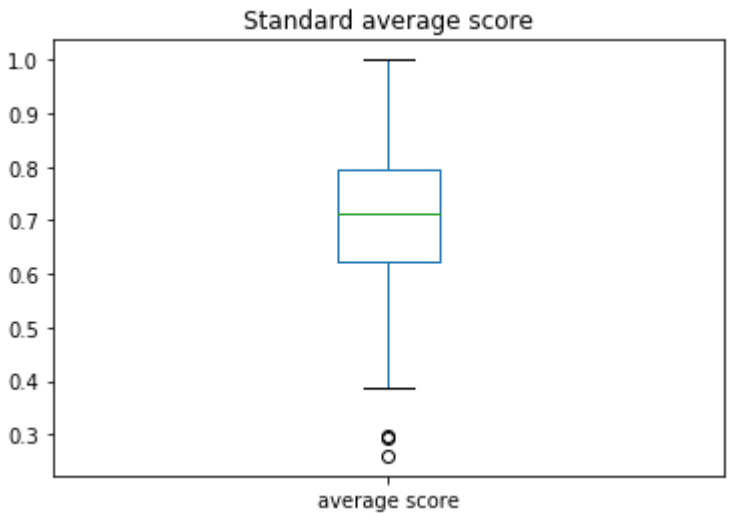
Out[40]:

	math score	reading score	writing score	average score
count	645.000000	645.000000	645.000000	645.000000
mean	0.700341	0.716543	0.708233	0.708372
std	0.136535	0.138306	0.143395	0.131865
min	0.190000	0.260000	0.220000	0.260000
25%	0.610000	0.630000	0.620000	0.623333
50%	0.690000	0.720000	0.720000	0.713333
75%	0.800000	0.820000	0.810000	0.796667
max	1.000000	1.000000	1.000000	1.000000







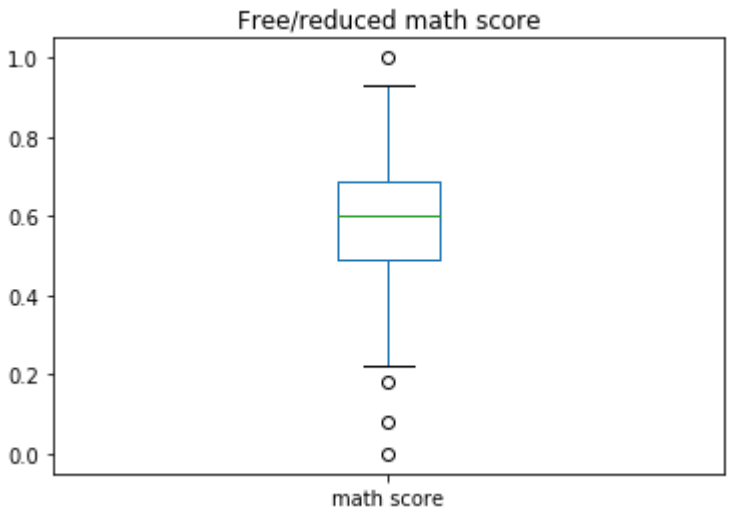


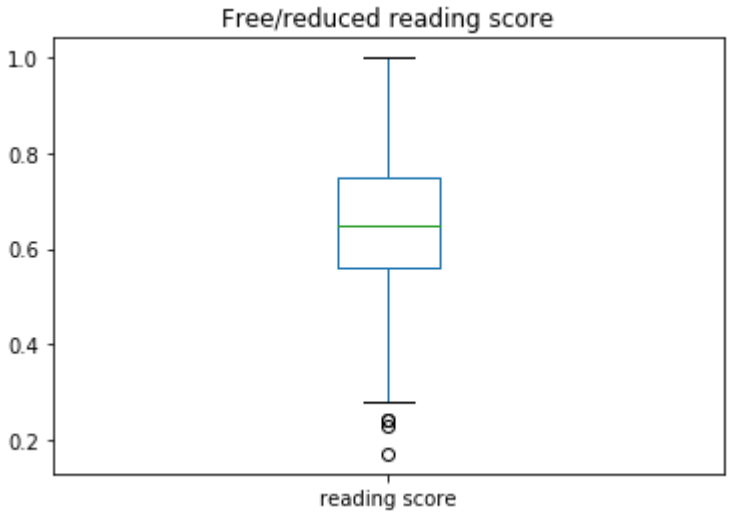
Free/reduced

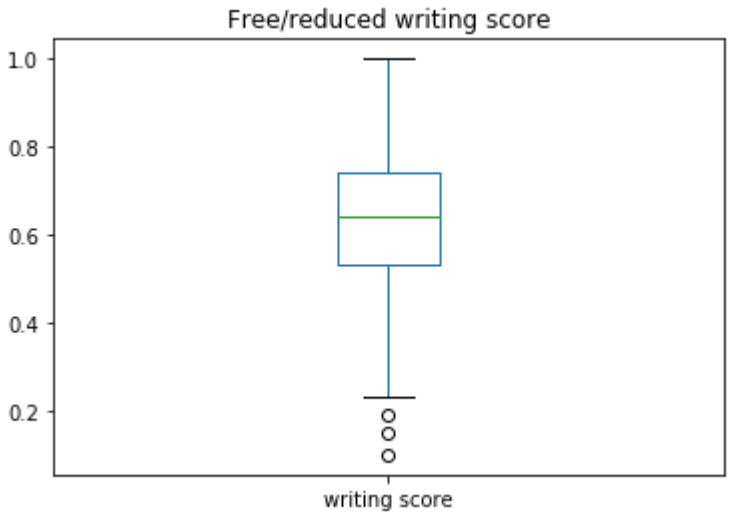
In [41]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['lunch']==0).dropna().plot(y=c,kind='box',title='Free/reduced '+c)
new_data[new_data.columns[-4:]].where(new_data['lunch']==0).dropna().describe()
```

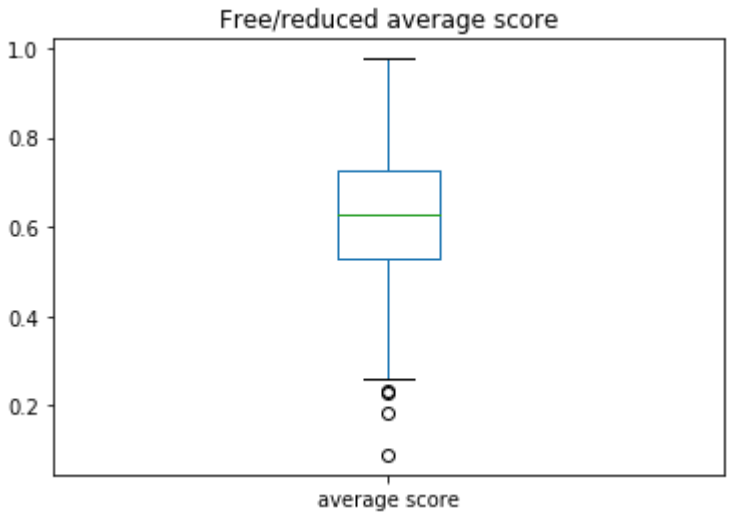
Out[41]:

	math score	reading score	writing score	average score
count	355.000000	355.000000	355.000000	355.000000
mean	0.589211	0.646535	0.630225	0.621991
std	0.151600	0.148953	0.154338	0.144583
min	0.000000	0.170000	0.100000	0.090000
25%	0.490000	0.560000	0.530000	0.528333
50%	0.600000	0.650000	0.640000	0.626667
75%	0.690000	0.750000	0.740000	0.725000
max	1.000000	1.000000	1.000000	0.976667









In [42]:
```
new_data[['lunch']+new_data.columns[-4:].tolist()].corr()[new_data.columns[-4:]].iloc[:-4]
```

Out[42]:

	math score	reading score	writing score	average score
lunch	0.350877	0.22956	0.245769	0.290064

We see that lunch has a relatively good correlation with the math score 1:3, and the people that had standard lunches perform a 5% better on average

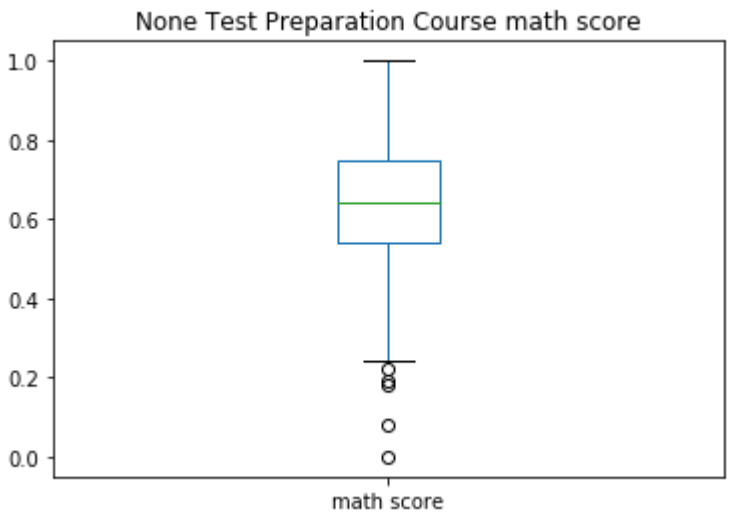
Test Preparation Course

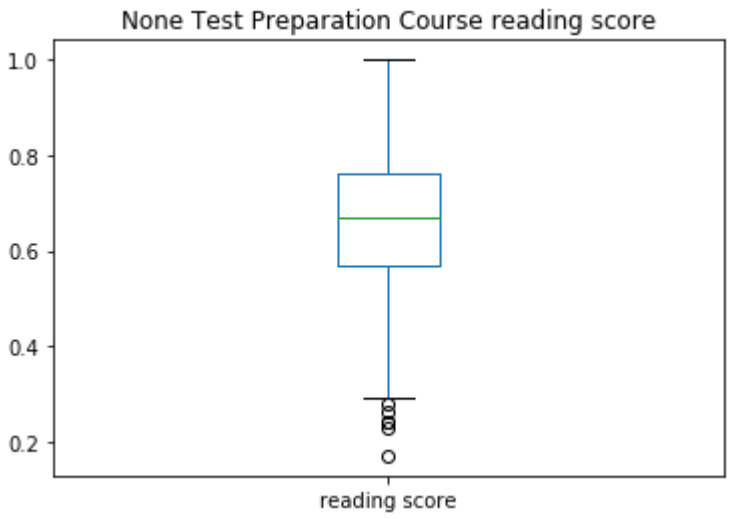
None

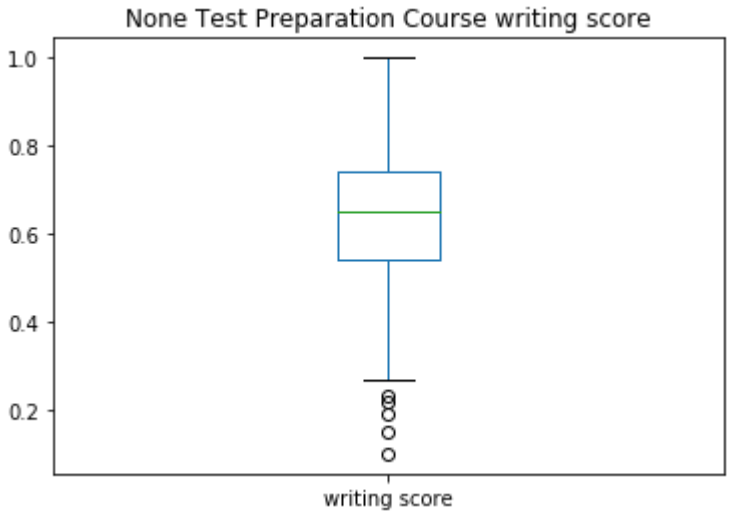
In [43]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['test preparation course']==1).dropna().plot(y=c,kind='box',title='None Test Preparation Course '+c)
new_data[new_data.columns[-4:]].where(new_data['test preparation course']==1).dropna().describe()
```

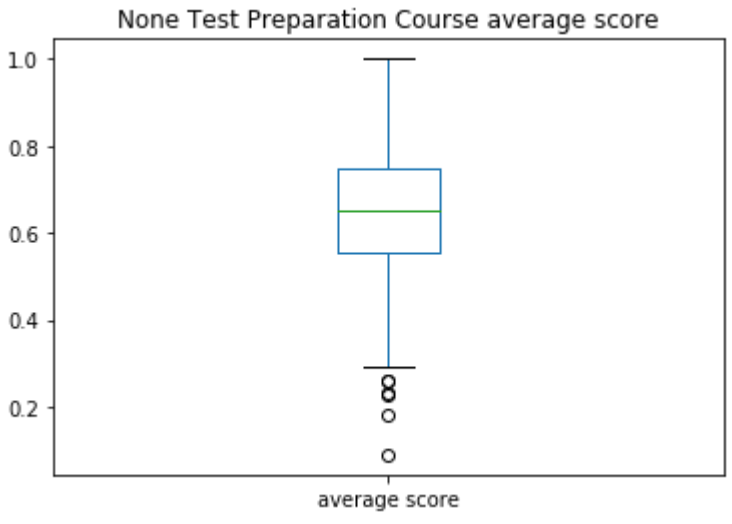
Out[43]:

	math score	reading score	writing score	average score
count	642.000000	642.000000	642.000000	642.000000
mean	0.640779	0.665343	0.645047	0.650389
std	0.151924	0.144639	0.149997	0.141867
min	0.000000	0.170000	0.100000	0.090000
25%	0.540000	0.570000	0.540000	0.554167
50%	0.640000	0.670000	0.650000	0.653333
75%	0.747500	0.760000	0.740000	0.750000
max	1.000000	1.000000	1.000000	1.000000







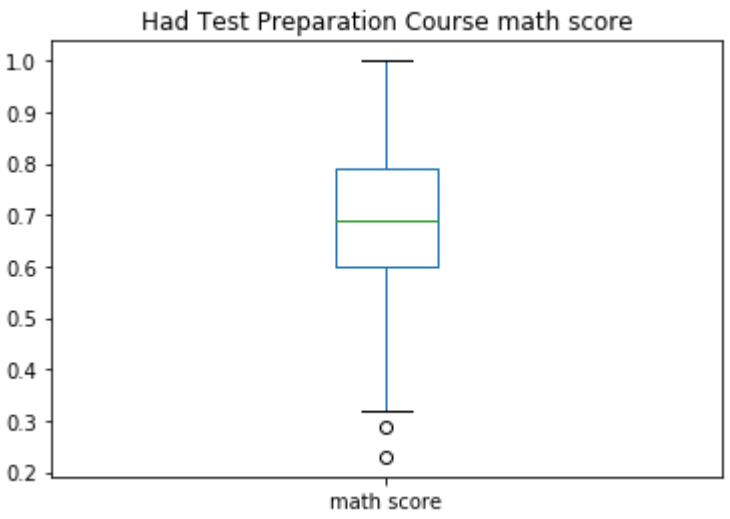


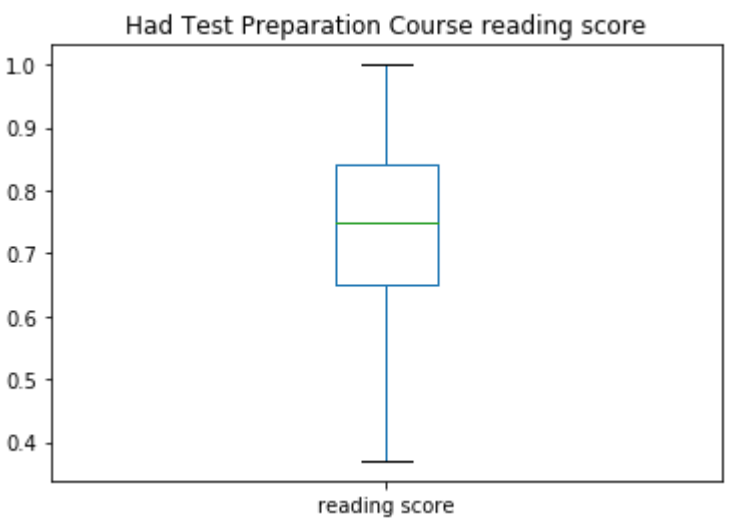
Did completed a course

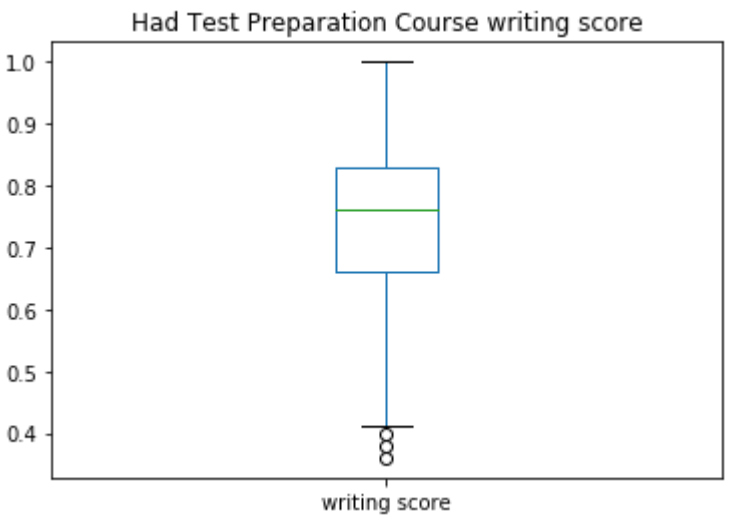
In [44]:
```
for c in new_data.columns[-4:]:
    new_data.where(new_data['test preparation course']==0).dropna().plot(y=c,kind='box',title='Had Test Preparation Course '+c)
new_data[new_data.columns[-4:]].where(new_data['test preparation course']==0).dropna().describe()
```

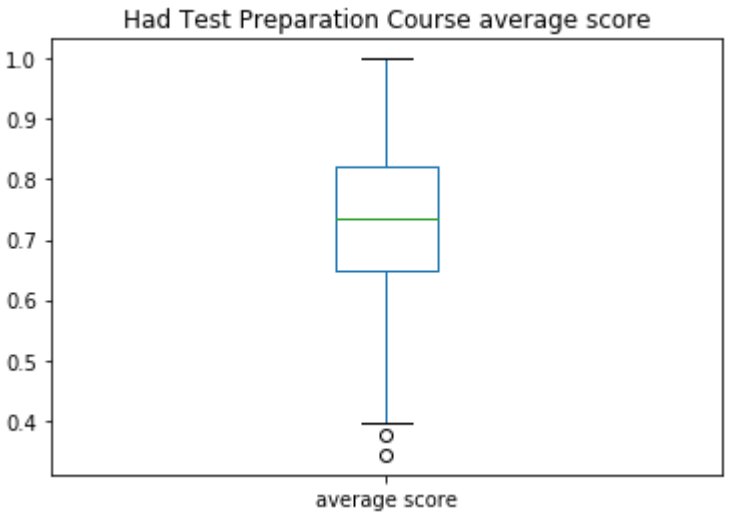
Out[44]:

	math score	reading score	writing score	average score
count	358.000000	358.000000	358.000000	358.000000
mean	0.696955	0.738939	0.744190	0.726695
std	0.144447	0.136384	0.133753	0.130370
min	0.230000	0.370000	0.360000	0.343333
25%	0.600000	0.650000	0.660000	0.650000
50%	0.690000	0.750000	0.760000	0.735000
75%	0.790000	0.840000	0.830000	0.821667
max	1.000000	1.000000	1.000000	1.000000









In [45]:
```
new_data[['test preparation course']+new_data.columns[-4:].tolist()].corr()[new_data.columns[-4:]].iloc[:-4]
```

Out[45]:

	math score	reading score	writing score	average score
test preparation course	-0.177702	-0.24178	-0.312946	-0.25671

We can see that studying actually increases the mean average score by 7%, but still we can se in both groups there were people that achive ones in their scores.

Splitting the Dataset into Train, Validation, and Test sets

60:20:20 split

600:200:200 split

In [46]:
```
train, testVal =train_test_split(new_data,test_size=0.4,random_state=1)
test,validation=train_test_split(testVal,test_size=0.5,random_state=1)
print('Train set:',len(train))
print('Validation set:',len(validation))
print('Test set:',len(test))
```

```
Train set: 600
Validation set: 200
Test set: 200
```

Refine and Evaluate Your Model

we are going to make models for each of the original categories (gender,race/ethnicity,parental level of education,lunch,test preparation course), see how it reflects on each of the different scores (math score,reading score,writing score), and in the end see how they reflect the average score.

this will be done using a ridge regression for alpha between [-9,9]

For calculating error we are going to use MSE, so we can increment our accurancy by penalizing the biggests mistakes.

In [47]:
```
lambdas=np.linspace(-9,10,30)
lambdas
```

Out[47]:
```
array([-9.        , -8.34482759, -7.68965517, -7.03448276, -6.37931034,
       -5.72413793, -5.06896552, -4.4137931 , -3.75862069, -3.10344828,
       -2.44827586, -1.79310345, -1.13793103, -0.48275862,  0.17241379,
        0.82758621,  1.48275862,  2.13793103,  2.79310345,  3.44827586,
        4.10344828,  4.75862069,  5.4137931 ,  6.06896552,  6.72413793,
        7.37931034,  8.03448276,  8.68965517,  9.34482759, 10.        ])
```

Gender Model

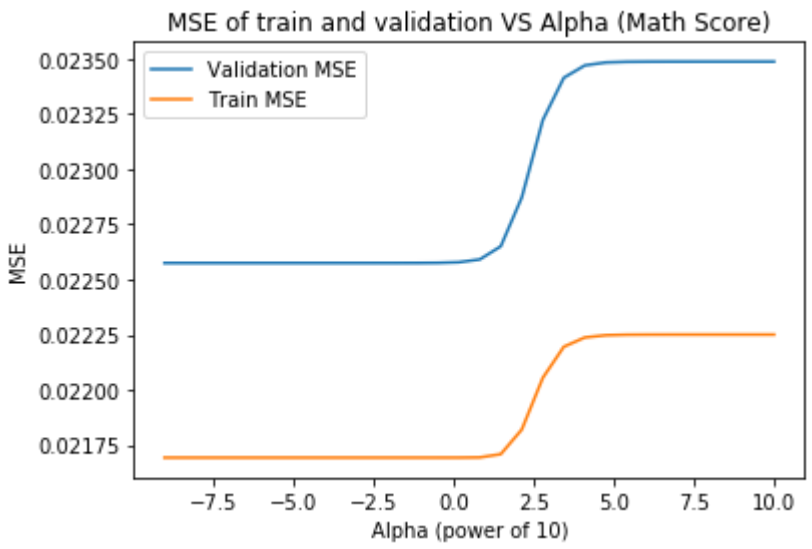
-Math Score

In [48]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['gender'].values.reshape((-1,1)),train['math score'])
    preds=model.predict(validation['gender'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['math score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['math score'],model.predict(train['gender'].values.re
shape((-1,1)))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-------------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Math Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.022574065227669474
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.02257406522767783
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.022574065227715514
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.022574065227885905
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.02257406522865621
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.022574065232138164
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.02257406524787794
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.022574065319027205
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.02257406564064733
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.02257406709448807
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.022574073666446468
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.02257410337573887
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.02257423770725469
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.022574845642703522
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.02257760769383895
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.02259033322404906
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.022649727554218013
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.02287110195738924
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.023221257084225636
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.023412777006881248
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.02346873537901534
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.02348198487911752
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.02348496142545981
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.02348562216035587
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.02348576844009752
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.02348580080577117
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.023485807966024053
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.023485809550039857
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.023485809900459044
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.02348580997797941
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[48]: `<matplotlib.legend.Legend at 0x26f637f9748>`



We see that the best model might have a high complexity for the gender feature since the best MSE is achivied with a low alpha.

Now we calculate the generalaization error

In [49]:
```
preds=bestModel.predict(test['gender'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['math score'],preds)])/len(preds)
print('MSE for test set Math Score: ', MSE)
```

```
MSE for test set Math Score:  0.02398411001483309
```

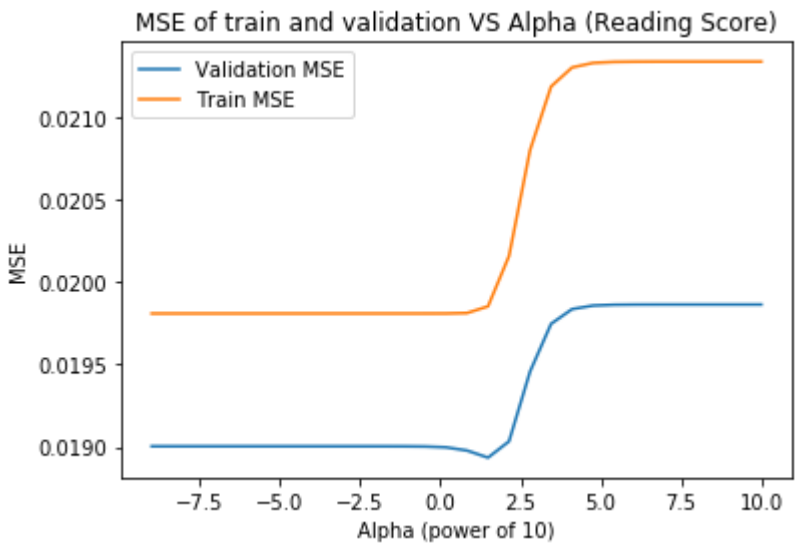
-Reading Score

In [50]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['gender'].values.reshape((-1,1)),train['reading score'])
    preds=model.predict(validation['gender'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['reading score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['reading score'],model.predict(train['gender'].values
.reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Reading Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.01900320990809008
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.01900320990807442
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.019003209908003634
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.01900320990768366
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.01900320990623731
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.019003209899699267
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.019003209870145026
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.01900320973654951
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.019003209132652736
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.019003206402870548
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.0190031940642066
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.01900313830781324
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.019002886651635093
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.019001756862625355
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.018996806298693415
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.0189773986017701
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.01893416479891168
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.019033366740272263
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.019456812469172385
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.01974632720185902
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.019835816366900877
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.019857283412885524
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.01986212016534067
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.01986319452526652
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.01986343241133946
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.019863485047385922
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.01986349669213191
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.01986349926822674
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.01986349983811586
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019863499964187716
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=30.39195382313195, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[50]: <matplotlib.legend.Legend at 0x26f637f0a08>



We see that the best model might have a high complexity for the gender feature since the best MSE is achivied with a low alpha.

Now we calculate the generalaization error

In [51]:
```
preds=bestModel.predict(test['gender'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['reading score'],preds)])/len(preds)
print('MSE for test set Reading Score: ', MSE)
```

```
MSE for test set Reading Score:  0.021682737817237428
```

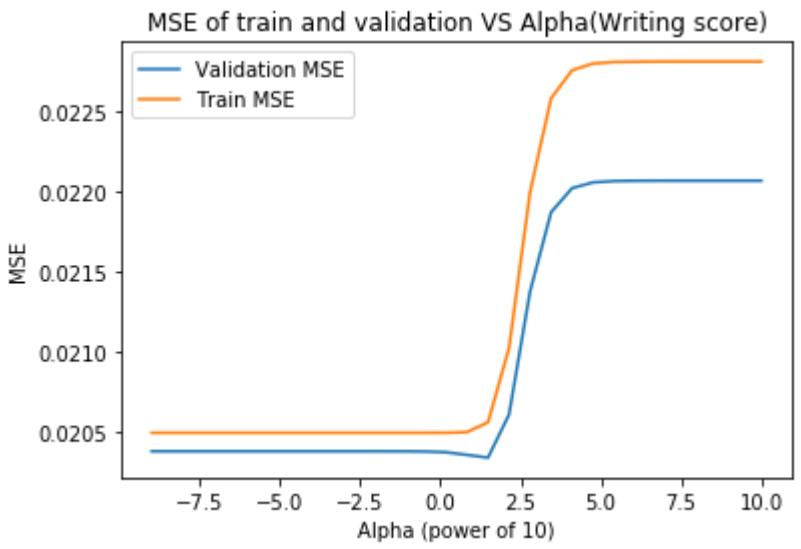
-Writing Score

In [52]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['gender'].values.reshape((-1,1)),train['writing score'])
    preds=model.predict(validation['gender'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['writing score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['writing score'],model.predict(train['gender'].values
.reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha(Writing score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.020383026847223897
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.02038302684720926
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.020383026847143007
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.020383026846843545
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.02038302684548989
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.020383026839370953
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.020383026811711096
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.02038302668667889
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.020383026121492027
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.020383023566707643
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.020383012019404574
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.02038295984655666
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.02038272451816836
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.020381671158646066
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.020377118594916057
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.02036049074856724
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.02034340405497416
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.020612490327959202
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.02137879916984015
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.02187195573803672
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.022022307887399572
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.022058263149023878
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.022066358653898985
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.022068156586499837
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.022068554673425833
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.022068642755778255
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.02206866224232767
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.022068666553214423
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.02206866750687777
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.022068667717848846
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=30.39195382313195, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[52]: <matplotlib.legend.Legend at 0x26f623ee148>



We see that the best model might have a high complexity for the gender feature since the best MSE is achivied with a low alpha.

Now we calculate the generalaization error

In [53]:
```
preds=bestModel.predict(test['gender'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['writing score'],preds)])/len(preds)
print('MSE for test set writing Score: ', MSE)
```

```
MSE for test set writing Score:  0.023013780061304216
```

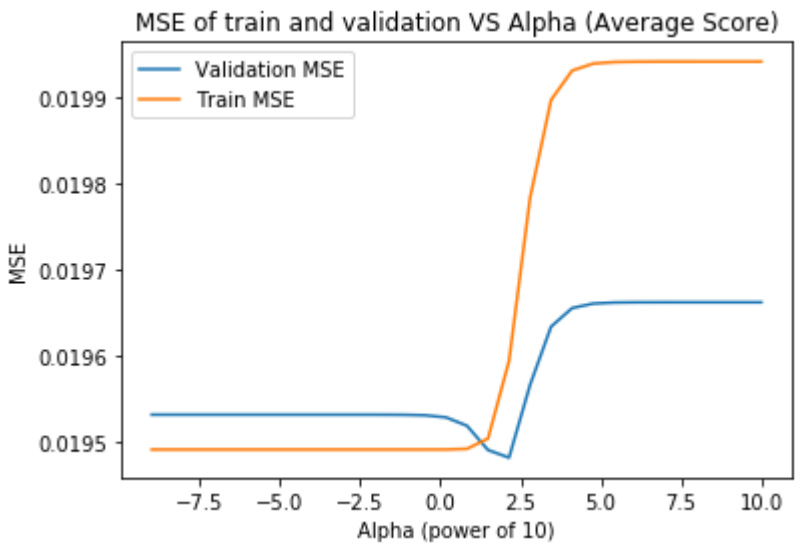
Average Score

In [54]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['gender'].values.reshape((-1,1)),train['average score'])
    preds=model.predict(validation['gender'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['average score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['average score'],model.predict(train['gender'].values
.reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Average Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.019532245887085492
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.019532245887078005
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.0195322458870442
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.019532245886891376
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.01953224588620056
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.01953224588307776
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.019532245868961615
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.019532245805151723
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.019532245516709304
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.01953224421286319
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.019532238319338296
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.01953221168507213
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.01953209142252217
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.019531550516935833
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.01952916023268754
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.01951940075334835
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.019491297625218392
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.01948248664260704
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.019566766637107876
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.019634202973911587
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.019655711669030497
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.019660906932439475
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.019662079264372447
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.019662339755449162
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.019662397437997036
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.019662410201381167
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.01966241302505274
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.01966241364971657
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019662413787906032
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.01966241381847656
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=137.3823795883261, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[54]: <matplotlib.legend.Legend at 0x26f62252b48>



We see that the best model might have a lower complexity than the other models seen for this feature because it penalizes more the complexity

Now we calculate the generalaization error

In [55]:
```
preds=bestModel.predict(test['gender'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['average score'],preds)])/len(preds)
print('MSE for test set average score: ', MSE)
```

```
MSE for test set average score:  0.021817606680576668
```

Race/Ethnicity Model

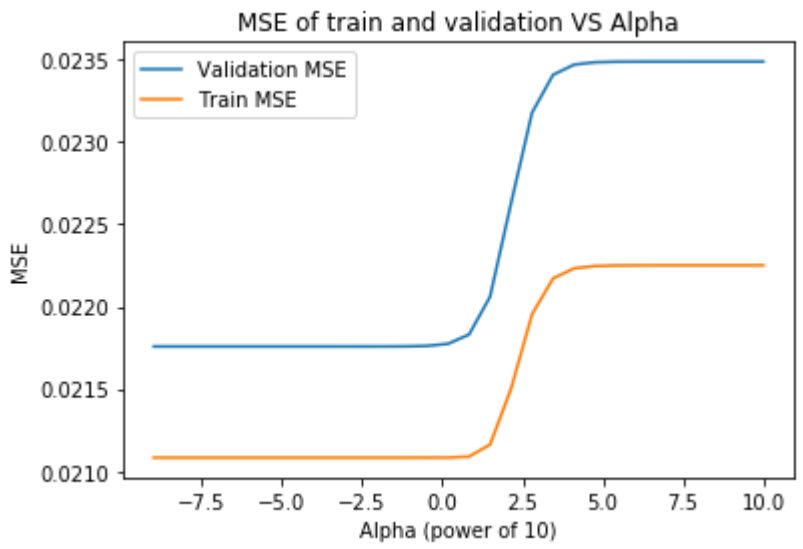
-Math Score

In [56]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[['group A','group B','group C','group D','group E']],train['math s
core'])
    preds=model.predict(validation[['group A','group B','group C','group D','group E']])
    MSE=sum([(r-p)**2 for r,p in zip(validation['math score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['math score'],model.predict(train[['group A','group
 B','group C','group D','group E']]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.021757852351217907
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.02175785235125594
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.02175785235142783
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.02175785235220492
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.02175785235571763
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.021757852371596272
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.021757852443373377
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.021757852767831282
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.021757854234495996
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.02175786086434 29
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.021757890833672088
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.021758026307193267
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.021758638726443215
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.021761407644394385
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.021773929748917027
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.021830199794665384
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.022059445873281208
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.02262632900541318
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.023176252425400654
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.023405761478912792
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.023467399200219362
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.023481700870688543
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.02348489917531622
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.023485608417726435
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.02348576540132431
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.02348580013359704
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.02348580781732793
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.023485809517145212
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.02348580989318205
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.02348580997636957
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[56]: <matplotlib.legend.Legend at 0x26f62629648>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [57]:
```
preds=bestModel.predict(test[['group A','group B','group C','group D','group E']])
MSE=sum([(r-p)**2 for r,p in zip(test['math score'],preds)])/len(preds)
print('MSE for test set Math Score: ', MSE)
```

```
MSE for test set Math Score:  0.02372989201447016
```

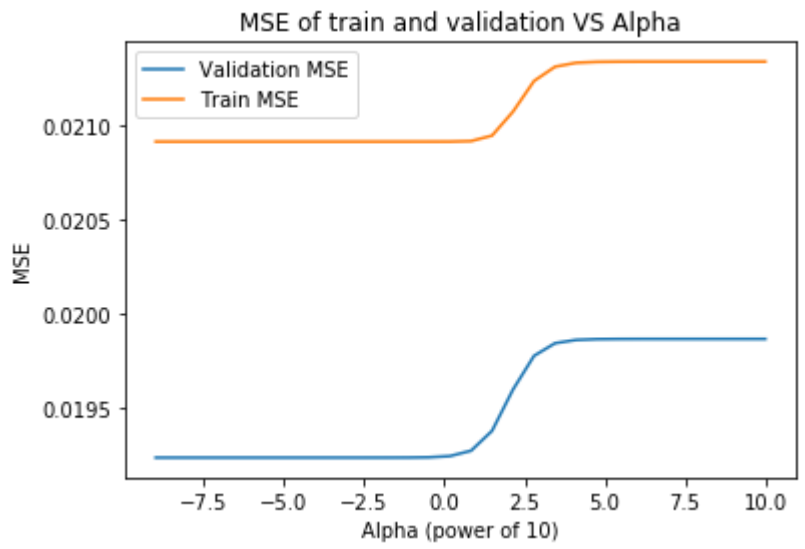
-Reading Score

In [58]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[['group A','group B','group C','group D','group E']],train['readin
g score'])
    preds=model.predict(validation[['group A','group B','group C','group D','group E']])
    MSE=sum([(r-p)**2 for r,p in zip(validation['reading score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['reading score'],model.predict(train[['group A','grou
p B','group C','group D','group E']]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.019232547068647646
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.019232547068667543
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.01923254706875758
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.019232547069164618
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.01923254707100445
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.019232547079321257
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.019232547116916167
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.019232547286858363
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.019232548055056656
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.019232551527572833
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.019232567224334057
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.019232638174339317
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.01923295879298533
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.019234406030779833
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.019240904088451544
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.0192692680679696
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.019374854996093146
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.019595124282644604
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.019773860442882346
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.019840988621088265
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.019858362327373365
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.019862355362484287
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.019863246382248335
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.019863443874687813
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.019863487582908294
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.01986349725302381
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.019863499392307173
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.019863499865565088
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019863499970260087
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019863499993420866
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[58]: <matplotlib.legend.Legend at 0x26f63c8aa08>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [59]:
```
preds=bestModel.predict(test[['group A','group B','group C','group D','group E']])
MSE=sum([(r-p)**2 for r,p in zip(test['reading score'],preds)])/len(preds)
print('MSE for test set Reading Score: ', MSE)
```

```
MSE for test set Reading Score:  0.022316533418641992
```

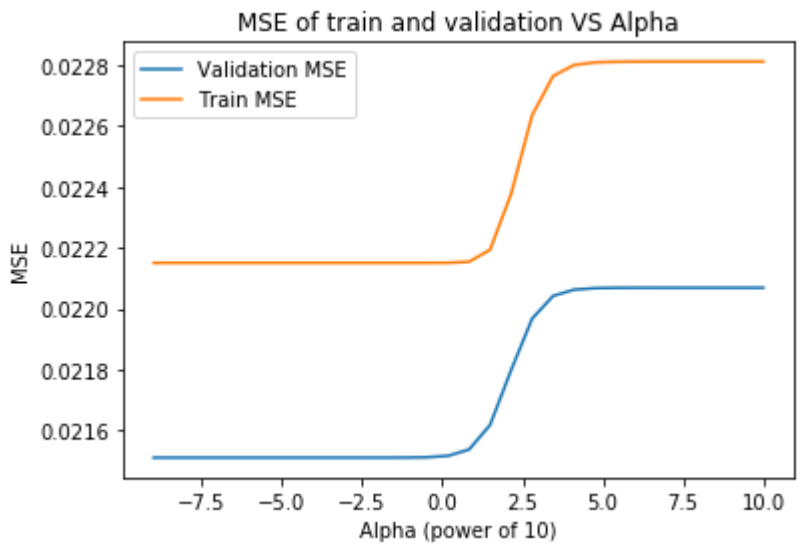
-Writing Score

In [60]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[['group A','group B','group C','group D','group E']],train['writin
g score'])
    preds=model.predict(validation[['group A','group B','group C','group D','group E']])
    MSE=sum([(r-p)**2 for r,p in zip(validation['writing score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['writing score'],model.predict(train[['group A','grou
p B','group C','group D','group E']]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.021511220390213474
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.021511220390227584
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.02151122039029133
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.02151122039057946
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.02151122039188194
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.021511220397769537
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.021511220424383633
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.021511220544688712
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.021511221088510318
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.021511223546776002
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.021511234659002735
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.02151128489011799
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.021511511950549744
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.02151253827153428
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.021517173314362207
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.021537833514179122
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.02161830089263854
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.021797654057981578
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.021966984192450162
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.022041738469888134
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.02206243281550564
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.022067274031816745
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.02206835873465069
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.022068599375587188
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.022068652644011567
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.022068664429777084
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.022068667037123468
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.022068667613928835
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.022068667741530823
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.022068667769759173
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[60]: <matplotlib.legend.Legend at 0x26f622bb308>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [61]:
```
preds=bestModel.predict(test[['group A','group B','group C','group D','group E']])
MSE=sum([(r-p)**2 for r,p in zip(test['writing score'],preds)])/len(preds)
print('MSE for test set writing Score: ', MSE)
```

```
MSE for test set writing Score:  0.024322818385711 52
```

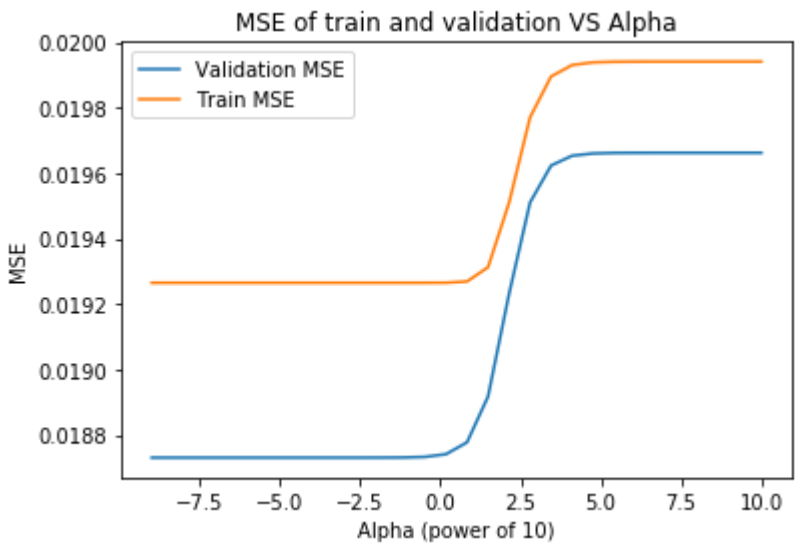
Average Score

In [62]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[['group A','group B','group C','group D','group E']],train['averag
e score'])
    preds=model.predict(validation[['group A','group B','group C','group D','group E']])
    MSE=sum([(r-p)**2 for r,p in zip(validation['average score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['average score'],model.predict(train[['group A','grou
p B','group C','group D','group E']]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.01873055826340837
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.01873055826343345
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.018730558263546818
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.018730558264059317
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.018730558266375916
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.01873055827684786
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.018730558324184743
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.018730558538164105
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.01873055950542617
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.018730563877785114
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.01873058364223763
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.018730672981356188
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.018731076760076334
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.018732900606260258
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.018741113931675376
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.018777391065638173
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.01891735279991355
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.01923092207350791
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.01951064980175255
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.01962345438525887
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.019653469448987892
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.019660418339357114
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.019661971550160036
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.01966231594509946
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.019662392171532497
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.01966240903636935
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.01966241276732908
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.01966241359270262
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019662413775293326
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019662413815686355
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[62]: <matplotlib.legend.Legend at 0x26f62275708>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [63]:
```
preds=bestModel.predict(test[['group A','group B','group C','group D','group E']])
MSE=sum([(r-p)**2 for r,p in zip(test['average score'],preds)])/len(preds)
print('MSE for test set average score: ', MSE)
```

```
MSE for test set average score:  0.021582651456023064
```

Parental Education Level Model

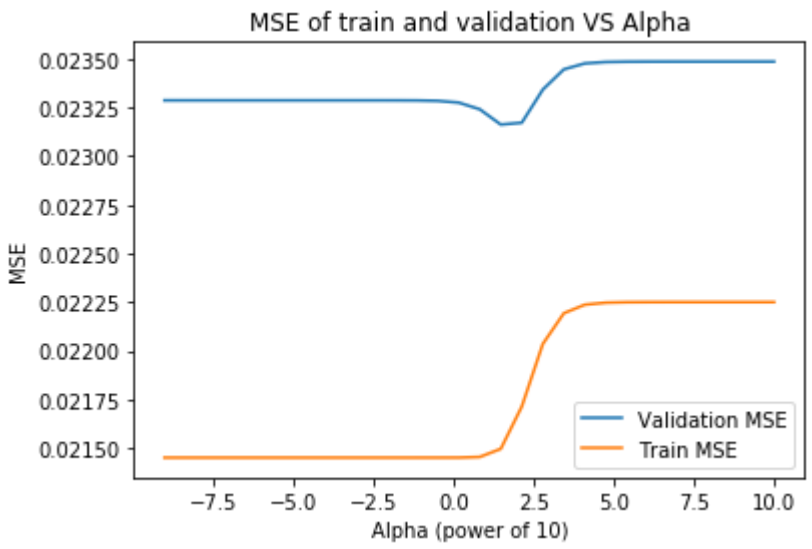
-Math Score

In [64]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[['associate degree', 'bachelor degree', 'high school', 'master deg
ree',
       'some college', 'some high school']],train['math score'])
    preds=model.predict(validation[['associate degree', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']])
    MSE=sum([(r-p)**2 for r,p in zip(validation['math score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['math score'],model.predict(train[['associate degree'
, 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-------------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.023286746779763737
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.02328674677973517
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.023286746779606182
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.02328674677902306
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.0232867467763871
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.02328674676447172
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.023286746710609933
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.02328674646713583
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.023286745366551776
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.02328674039162272
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.023286717905230604
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.02328661630059167
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.023286157864645093
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.023284102882589072
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.023275156744903266
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.023240866902108935
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.02316235280723738
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.023171800083685204
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.02334187672017418
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.023446011157217492
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.02347651617450725
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.02348372870062586
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.023485348317787383
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.023485707804381434
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.023485787389105846
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.02348580499783369
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.02348580889340539
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.023485809755196995
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.02348580994584426
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.02348580998801959
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=30.39195382313195, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[64]: <matplotlib.legend.Legend at 0x26f62313fc8>



We see that the best model might have high complexity but still smaller than the other models for the previous features

Now we calculate the generalaization error

In [65]:
```
preds=bestModel.predict(test[['associate degree', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']])
MSE=sum([(r-p)**2 for r,p in zip(test['math score'],preds)])/len(preds)
print('MSE for test set Math Score: ', MSE)
```

```
MSE for test set Math Score:  0.02435247199905342
```

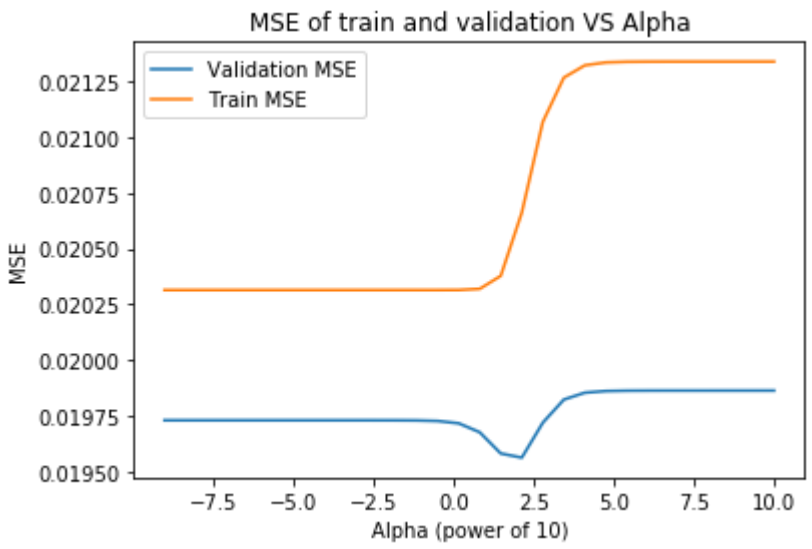
-Reading Score

In [66]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[['associate degree', 'bachelor degree', 'high school', 'master deg
ree',
       'some college', 'some high school']],train['reading score'])
    preds=model.predict(validation[['associate degree', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']])
    MSE=sum([(r-p)**2 for r,p in zip(validation['reading score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['reading score'],model.predict(train[['associate degr
ee', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.01973029817836148
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.019730298178328064
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.019730298178177008
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.01973029817749423
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.019730298174407714
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.019730298160455715
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.019730298097387664
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.019730297812298115
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.019730296523598263
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.019730290698338822
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.019730264368547818
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.019730145398459156
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.019729608629031302
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.019727202871500298
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.019716735285840115
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.01967660901580924
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.01958159697306496
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.019562914736196927
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.019719546996564766
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.019823171990068084
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.019854053186844065
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.019861382954708
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.01986303031529449
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.019863396029403145
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.019863476996214314
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.01986349491090643
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.019863498874174663
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.019863499750942707
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019863499944903124
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.01986349998781138
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=137.3823795883261, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[66]: <matplotlib.legend.Legend at 0x26f63b000c8>



It has the same complexity as the model before for the same feature

Now we calculate the generalaization error

In [67]:
```
preds=bestModel.predict(test[['associate degree', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']])
MSE=sum([(r-p)**2 for r,p in zip(test['reading score'],preds)])/len(preds)
print('MSE for test set Reading Score: ', MSE)
```

```
MSE for test set Reading Score:  0.02258436614749226
```

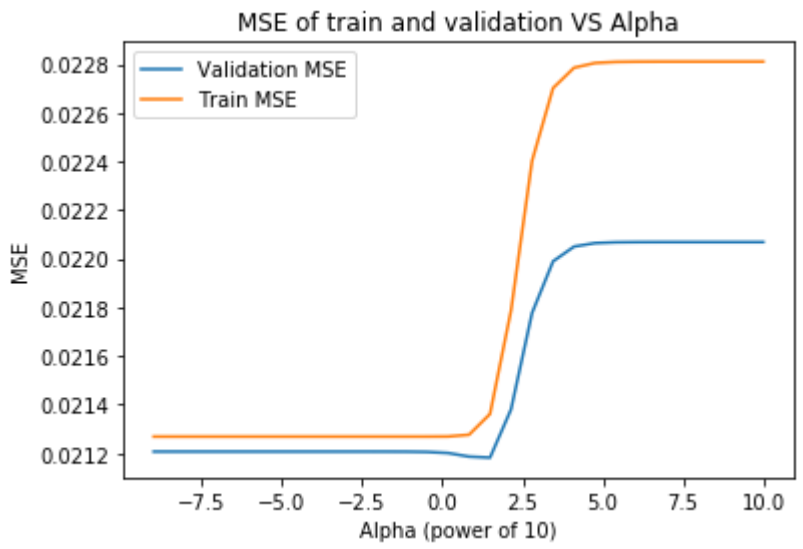
-Writing Score

In [68]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[['associate degree', 'bachelor degree', 'high school', 'master deg
ree',
       'some college', 'some high school']],train['writing score'])
    preds=model.predict(validation[['associate degree', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']])
    MSE=sum([(r-p)**2 for r,p in zip(validation['writing score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['writing score'],model.predict(train[['associate degr
ee', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-------------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.021207242531231005
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.021207242531216277
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.02120724253114963
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.021207242530848363
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.02120724252948654
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.021207242523330685
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.021207242495503875
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.02120724236971727
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.02120724180112167
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.021207239230959244
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.021207227614768995
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.021207175143327817
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.021206938725817935
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.021205885681939964
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.02120143394861412
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.02118671317578702
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.021182409579122048
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.021380971905232152
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.02177708461041001
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.021989526019803005
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.022050262996483574
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.02206454991157
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.022067754521531823
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.02206846563311504
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.02206862305349094
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.02206865788352825
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.02206866558894257
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.02206866729355943
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.022068667670658165
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.022068667754080604
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=30.39195382313195, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[68]: <matplotlib.legend.Legend at 0x26f626be0c8>



We see that the best model might have really high complexity,but smaller than the ones before.

Now we calculate the generalaization error

In [69]:
```
preds=bestModel.predict(test[['associate degree', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']])
MSE=sum([(r-p)**2 for r,p in zip(test['writing score'],preds)])/len(preds)
print('MSE for test set writing Score: ', MSE)
```

```
MSE for test set writing Score:  0.024678440200529853
```

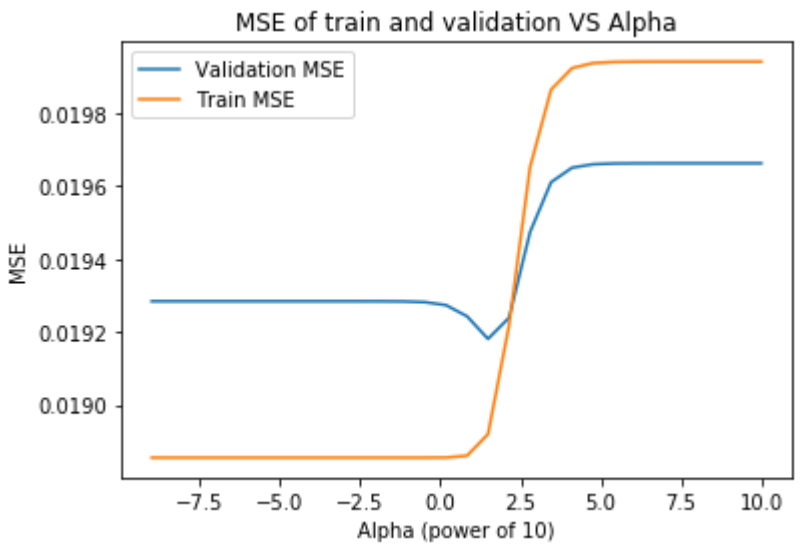
Average Score

In [70]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[['associate degree', 'bachelor degree', 'high school', 'master deg
ree',
       'some college', 'some high school']],train['average score'])
    preds=model.predict(validation[['associate degree', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']])
    MSE=sum([(r-p)**2 for r,p in zip(validation['average score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['average score'],model.predict(train[['associate degr
ee', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-------------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.01928405985614211
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.019284059856115855
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.019284059855997168
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.019284059855460674
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.019284059853035593
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.019284059842073313
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.019284059792519955
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.019284059568521486
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.019284058555974325
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.01928405397900813
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.01928403329164904
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.019283939821319575
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.019283518196892956
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.01928163044697266
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.019273455647386018
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.019242852365993643
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.019180860930539678
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.019238762838623014
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.01947355010323507
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.019610487702360197
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.01965030262524796
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.019659702305796067
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.01966181238147412
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.019662280695920843
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.019662384371809415
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.01966240731081171
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.019662412385594
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.01966241350825436
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.01966241375661155
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019662413811553543
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=30.39195382313195, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[70]: <matplotlib.legend.Legend at 0x26f63ab10c8>



It has a relative high complexity

Now we calculate the generalaization error

In [71]:
```
preds=bestModel.predict(test[['associate degree', 'bachelor degree', 'high school', 'master degree',
       'some college', 'some high school']])
MSE=sum([(r-p)**2 for r,p in zip(test['average score'],preds)])/len(preds)
print('MSE for test set average score: ', MSE)
```

```
MSE for test set average score:  0.02189094181095825
```

Lunch Model

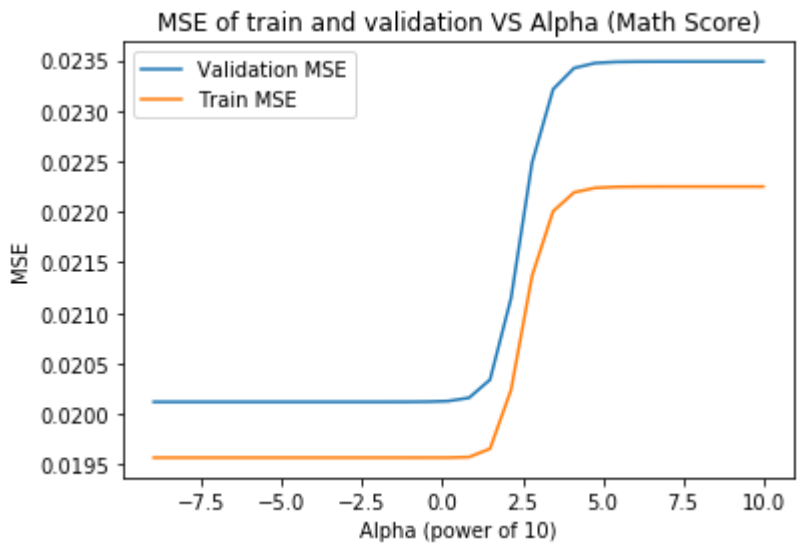
-Math Score

In [72]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['lunch'].values.reshape((-1,1)),train['math score'])
    preds=model.predict(validation['lunch'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['math score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['math score'],model.predict(train['lunch'].values.res
hape((-1,1)))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Math Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.020121483648474072
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.0201214836484929
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.02012148364857807
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.020121483648963098
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.020121483650703546
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.020121483658570933
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.020121483694134377
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.02012148385489376
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.02012148458158531
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.020121487866534275
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.02012150271660816
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.020121569863454845
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.020121873784025725
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.020123255599166542
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.020129660975194804
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.020161513775665208
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.020340954646104924
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.02114638686633111
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.02248639077940999
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.02321190591745903
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.023421919640972434
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.02347150491480879
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.02348263691697355
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.023485107628736426
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.023485654599880147
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.023485775621133582
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.023485802394602275
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.023485808317519136
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.023485809627798723
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.023485809917661013
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[72]: <matplotlib.legend.Legend at 0x26f63cb6dc8>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [73]:
```
preds=bestModel.predict(test['lunch'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['math score'],preds)])/len(preds)
print('MSE for test set Math Score: ', MSE)
```

```
MSE for test set Math Score:  0.021912342588019 95
```

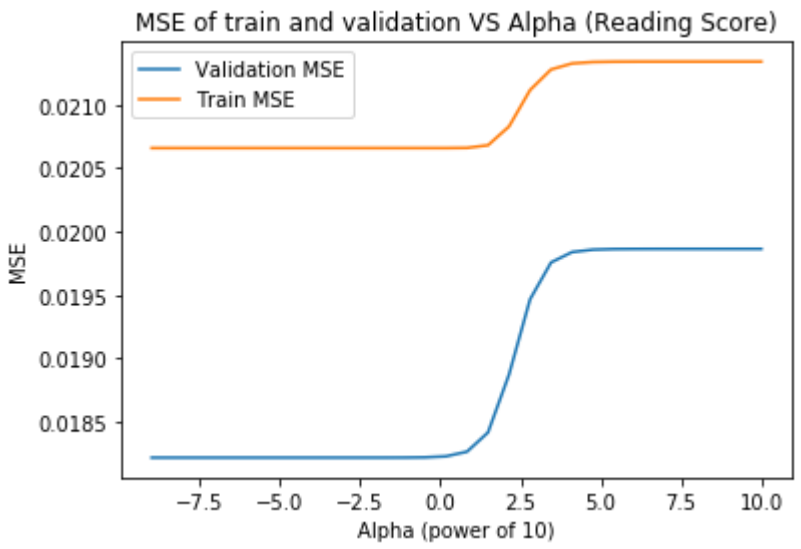
-Reading Score

In [74]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['lunch'].values.reshape((-1,1)),train['reading score'])
    preds=model.predict(validation['lunch'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['reading score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['reading score'],model.predict(train['lunch'].values.
reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Reading Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.018220409217750515
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.01822040921777554
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.01822040921788852
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.018220409218399264
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.018220409220707973
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.018220409231144145
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.01822040927831935
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.018220409491567984
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.018220410455526876
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.018220414812954513
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.018220434509915097
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.018220523544010336
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.018220925945560785
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.018222743630566817
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.018230933072824425
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.018267371939201905
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.01841901467527508
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.01887503648919585
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.019466818439923966
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.019757142513353358
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.019838827230959828
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.01985798266507993
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.0198622765132311
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.019863229194327897
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.01986344008487961
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.019863486745134475
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.01986349706772018
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.019863499351315452
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019863499856496904
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019863499968253994
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[74]: <matplotlib.legend.Legend at 0x26f63996a88>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [75]:
```
preds=bestModel.predict(test['lunch'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['reading score'],preds)])/len(preds)
print('MSE for test set Reading Score: ', MSE)
```

```
MSE for test set Reading Score:  0.020961393969171377
```

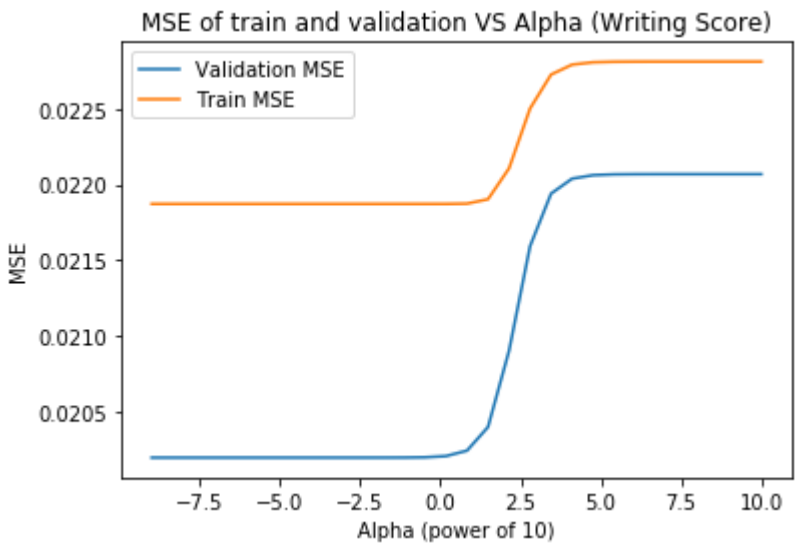
-Writing Score

In [76]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['lunch'].values.reshape((-1,1)),train['writing score'])
    preds=model.predict(validation['lunch'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['writing score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['writing score'],model.predict(train['lunch'].values.
reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Writing Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.020199689408218063
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.020199689408242367
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.020199689408352275
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.020199689408848993
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.020199689411094356
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.020199689421244282
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.020199689467125456
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.020199689674524565
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.020199690612041 98
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.0201996948499514
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.020199714006786 69
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.020199800602166685
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.020200192037878515
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.020201961333578 83
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.020209955669383474
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.020245953705397248
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.020401738105521 05
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.020904098218271917
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.021593590460294 7
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.021940533580202252
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.022038898616429173
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.022062008504229205
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.022067190949714637
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.022068340892199022
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.022068595454838778
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.022068651777949682
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.022068664238248704
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.022068666994756368
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.022068667604556454
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.022068667739457468
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[76]: <matplotlib.legend.Legend at 0x26f63c73e08>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [77]:
```
preds=bestModel.predict(test['lunch'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['writing score'],preds)])/len(preds)
print('MSE for test set writing Score: ', MSE)
```

```
MSE for test set writing Score:  0.022805815329092302
```

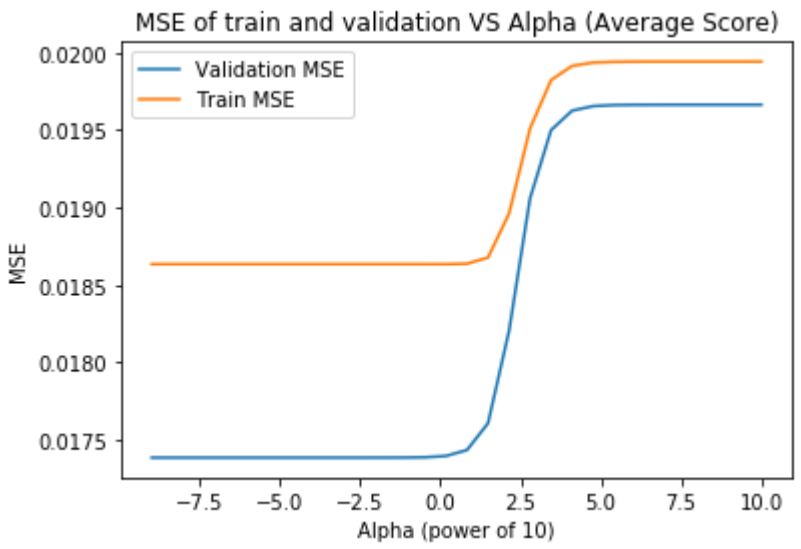
Average Score

In [78]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['lunch'].values.reshape((-1,1)),train['average score'])
    preds=model.predict(validation['lunch'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['average score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['average score'],model.predict(train['lunch'].values.
reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Average Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.017384335384640154
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.017384335384665665
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.017384335384780947
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.01738433538530207
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.017384335387657702
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.017384335398306038
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.017384335446440344
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.017384335664024378
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.01738433664758151
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.017384341093614548
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.017384361191399432
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.017384452043417885
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.017384862786230817
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.01738672069029647
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.01739514219674231
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.017433562312713747
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.01760671856209385
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.01820266427164576
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.01905945451346281
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.01949906798237221
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.01962442181230645
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.01965391297207962
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.01966052848491461
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.019661996514603267
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.01966232149713408
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.01966239340120058
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.019662409308468905
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.019662412827526822
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.01966241360601984
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019662413778239386
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[78]: <matplotlib.legend.Legend at 0x26f63b5cb88>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [79]:
```
preds=bestModel.predict(test['lunch'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['average score'],preds)])/len(preds)
print('MSE for test set average score: ', MSE)
```

```
MSE for test set average score:  0.019841617719585862
```

Test Preparation Model

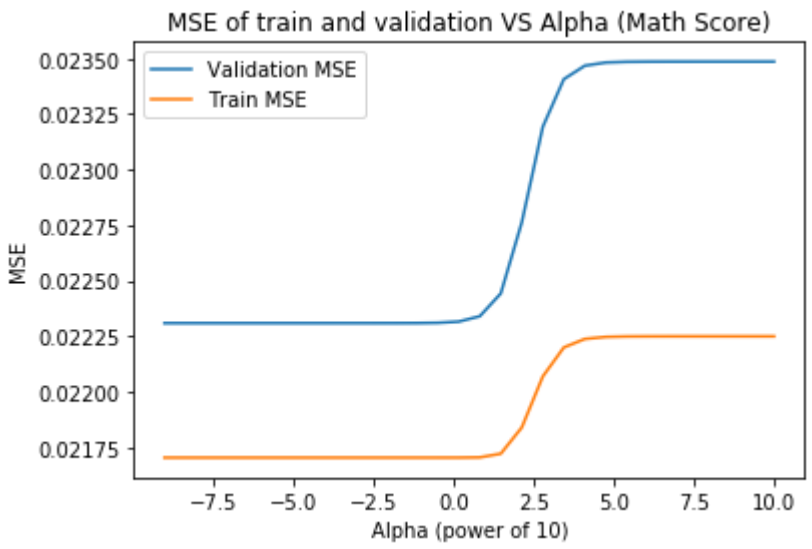
-Math Score

In [80]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['test preparation course'].values.reshape((-1,1)),train['math scor
e'])
    preds=model.predict(validation['test preparation course'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['math score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['math score'],model.predict(train['test preparation c
ourse'].values.reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Math Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.022309373362891485
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.022309373362907902
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.022309373362982093
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.022309373363317418
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.022309373364833275
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.022309373371685336
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.02230937340265931
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.022309373542672515
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.02230937417558152
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.022309377036551437
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.02230938996909306
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.022309448427646133
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.02230971265827917
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.022310906606140723
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.02231629373195376
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.022340411093510905
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.02244285648052731
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.022763034207087494
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.023192651218942126
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.02340689193543753
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.023467483643873806
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.023481710878975673
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.02348490095856822
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.02348560879106792
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.02348576548287959
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.023485800151588094
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.023485807821305466
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.02348580951802502
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.02348580989337669
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.023485809976412596
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[80]: <matplotlib.legend.Legend at 0x26f626f99c8>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [81]:
```
preds=bestModel.predict(test['test preparation course'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['math score'],preds)])/len(preds)
print('MSE for test set Math Score: ', MSE)
```

```
MSE for test set Math Score:  0.023887814844591557
```

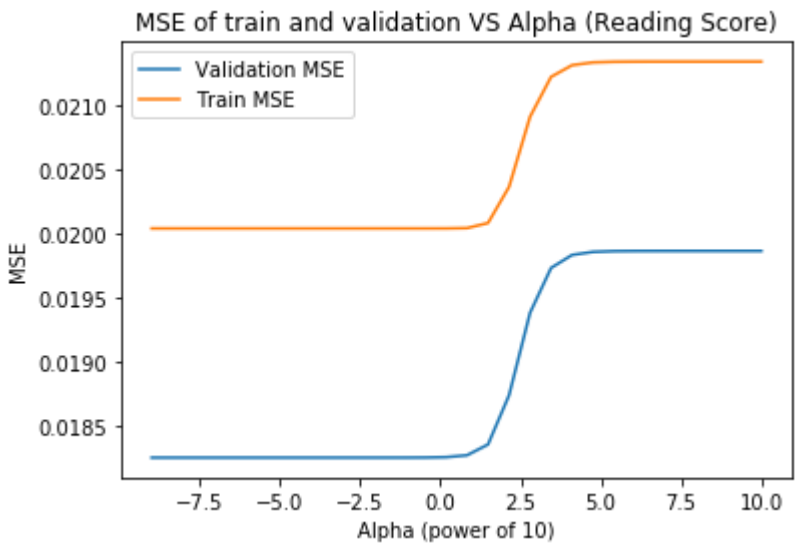
-Reading Score

In [82]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['test preparation course'].values.reshape((-1,1)),train['reading s
core'])
    preds=model.predict(validation['test preparation course'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['reading score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['reading score'],model.predict(train['test preparatio
n course'].values.reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Reading Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.018254189700397105
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.018254189700405772
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.01825418970044492
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.01825418970062195
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.018254189701422178
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.018254189705039475
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.01825418972139092
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.018254189795305367
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.01825419012942589
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.018254191639791216
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.018254198467636124
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.01825422934129355
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.01825436909265426
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.018255004709708332
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.01825795546393272
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.01827270893253915
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.01835686447336914
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.018739713677341376
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.019382121013910084
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.01973142254580171
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.019832682553430018
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.01985659947508283
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.01986196933588569
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.01986316118182127
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.01986342503619001
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.01986348341589924
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.01986349633121447
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.019863499188384138
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019863499820452938
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019863499960280303
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[82]: `<matplotlib.legend.Legend at 0x26f63bebc08>`



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [83]:
```
preds=bestModel.predict(test['test preparation course'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['reading score'],preds)])/len(preds)
print('MSE for test set Reading Score: ', MSE)
```

```
MSE for test set Reading Score:  0.02189865388154558
```

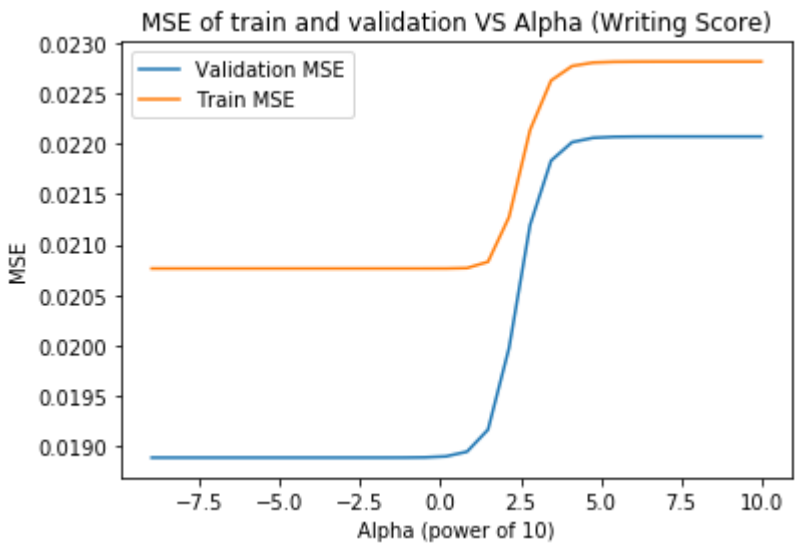
-Writing Score

In [84]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['test preparation course'].values.reshape((-1,1)),train['writing s
core'])
    preds=model.predict(validation['test preparation course'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['writing score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['writing score'],model.predict(train['test preparatio
n course'].values.reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Writing Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.018890831879488087
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.018890831879518108
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.018890831879653767
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.01889083188026705
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.0188908318830393
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.01889083189557084
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.01889083195221783
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.018890832208282363
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.018890833365785602
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.01889083859812955
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.01889086225058224
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.01889096917635843
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.01889145268849202
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.01889364177621344
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.01890360502157119
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.018949811779931644
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.01916840121754517
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.019976016197129402
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.021193331553040543
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.021830441536314762
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.02201319481962399
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.02205625217937627
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.02206591404936247
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.02206805824355987
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.022068532918488697
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.022068637943147378
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.022068661177671187
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.022068666317689445
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.02206866745477455
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.022068667706322462
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[84]: <matplotlib.legend.Legend at 0x26f638701c8>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [85]:
```
preds=bestModel.predict(test['test preparation course'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['writing score'],preds)])/len(preds)
print('MSE for test set writing Score: ', MSE)
```

```
MSE for test set writing Score:  0.022917551606776138
```

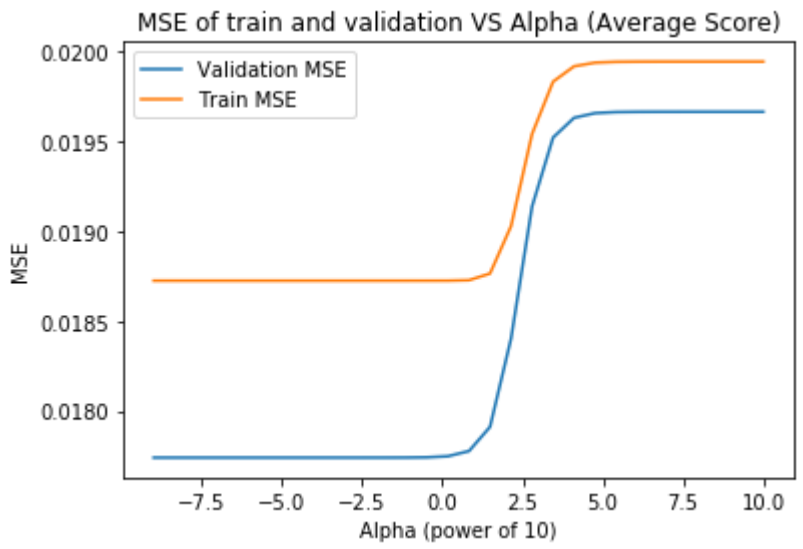
Average Score

In [86]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train['test preparation course'].values.reshape((-1,1)),train['average s
core'])
    preds=model.predict(validation['test preparation course'].values.reshape((-1,1)))
    MSE=sum([(r-p)**2 for r,p in zip(validation['average score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['average score'],model.predict(train['test preparatio
n course'].values.reshape((-1,1))))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Average Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.0177409813571112
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.017740981357129877
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.017740981357214306
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.017740981357595952
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.017740981359321083
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.01774098136711934
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.017740981402370203
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.017740981561716503
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.017740982282018834
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.017740985538051077
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.01774100025669433
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.01774106679478186
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.017741367664588205
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.017742729609422688
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.01774892356987625
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.017777562793928104
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.017911881517085136
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.01840231349054999
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.019136242763952382
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.019519300879702904
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.01962909382490401
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.019654956616415756
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.019660759861299643
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.019662047724580512
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.019662332827089145
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.019662395907690474
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.019662409862961566
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.01966241295019274
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019662413633156194
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019662413784242553
--------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[86]: <matplotlib.legend.Legend at 0x26f626c0a88>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [87]:
```
preds=bestModel.predict(test['test preparation course'].values.reshape((-1,1)))
MSE=sum([(r-p)**2 for r,p in zip(test['average score'],preds)])/len(preds)
print('MSE for test set average score: ', MSE)
```

```
MSE for test set average score:  0.02094556319124707
```

All Feature Model

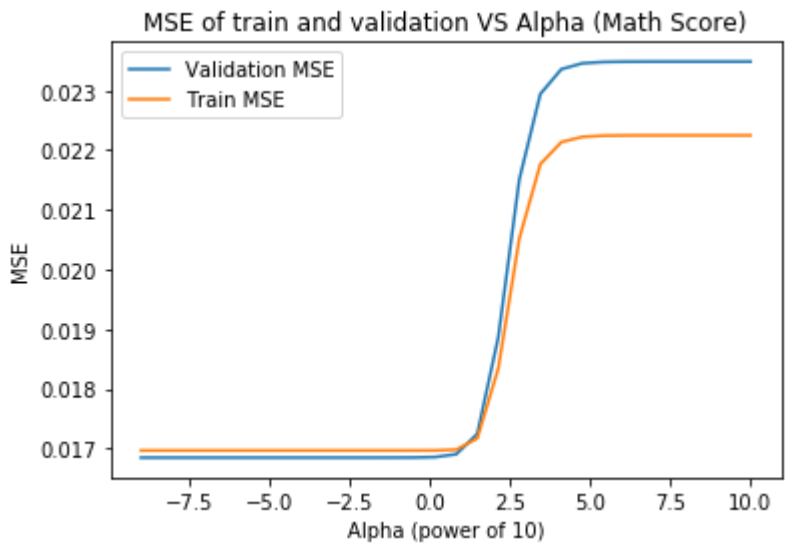
-Math Score

In [88]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[new_data.columns[:-4]],train['math score'])
    preds=model.predict(validation[new_data.columns[:-4]])
    MSE=sum([(r-p)**2 for r,p in zip(validation['math score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['math score'],model.predict(train[new_data.columns[:-
4]]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-------------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Math Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.016847738202133488
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.016847738202155026
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.01684773820225243
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.016847738202692638
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.016847738204682584
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.01684773821367783
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.01684773825433956
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.01684773843814543
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.016847739269022506
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.016847743025075718
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.01684776000774838
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.016847836856805253
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.016848185901426324
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.01684979734661592
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.016857744539511846
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.016905236500306283
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.017245556913417524
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.018866720860180188
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.0215113166436075
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.022942701286013764
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.023358953724910342
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.023457397231019413
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.023479507137696812
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.02348441481910125
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.02348550131411145
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.02348574170996392
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.02348579489266707
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.02348580665792632
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.023485809260660778
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.023485809836442148
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=1e-09, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[88]: <matplotlib.legend.Legend at 0x26f6265fb88>



We see that the best model might have really high complexity because it has the smallest alpha possible.

Now we calculate the generalaization error

In [89]:
```
preds=bestModel.predict(test[new_data.columns[:-4]])
MSE=sum([(r-p)**2 for r,p in zip(test['math score'],preds)])/len(preds)
print('MSE for test set Math Score: ', MSE)
```

```
MSE for test set Math Score:  0.018738439375397105
```

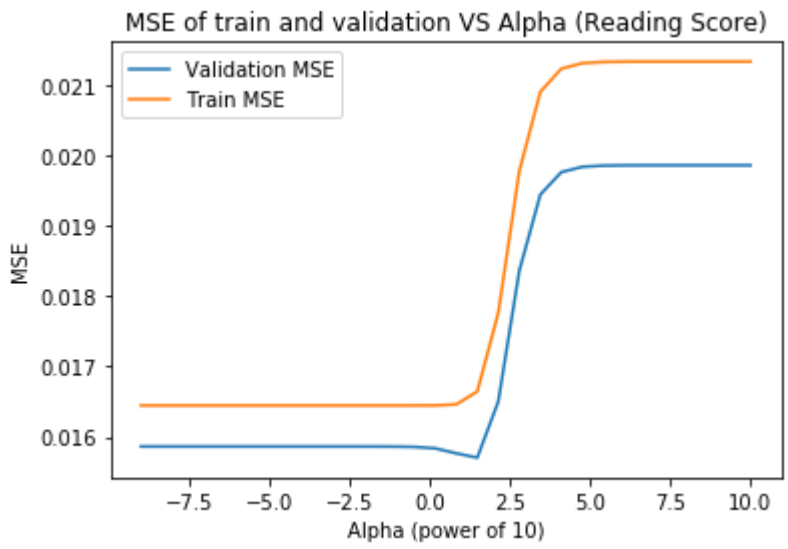
-Reading Score

In [90]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[new_data.columns[:-4]],train['reading score'])
    preds=model.predict(validation[new_data.columns[:-4]])
    MSE=sum([(r-p)**2 for r,p in zip(validation['reading score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['reading score'],model.predict(train[new_data.columns
[:-4]]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Reading Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.015863204519644944
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.01586320451957793
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.015863204519275115
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.01586320451790624
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.015863204511718423
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.015863204483747294
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.01586320435730792
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.015863203785758225
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.015863201202168584
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.015863189523781622
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.015863136740532486
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.015862898288446548
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.015861823396934732
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.015857025292883575
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.015836539032546268
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.01576535614291117
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.015704336734886085
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.016502938030734514
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.018363182360044448
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.019446231132205158
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.019765803659230242
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.019841606954674174
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.019858642863505784
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.0198624248129272
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.019863262111894906
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.019863447372367643
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.01986348835754925
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.01986349742443416
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019863499430228932
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019863499873954308
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=30.39195382313195, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[90]: <matplotlib.legend.Legend at 0x26f637c4448>



has a lower complexity than the one before

Now we calculate the generalaization error

In [91]:
```
preds=bestModel.predict(test[new_data.columns[:-4]])
MSE=sum([(r-p)**2 for r,p in zip(test['reading score'],preds)])/len(preds)
print('MSE for test set Reading Score: ', MSE)
```

```
MSE for test set Reading Score:  0.01844106300098884
```

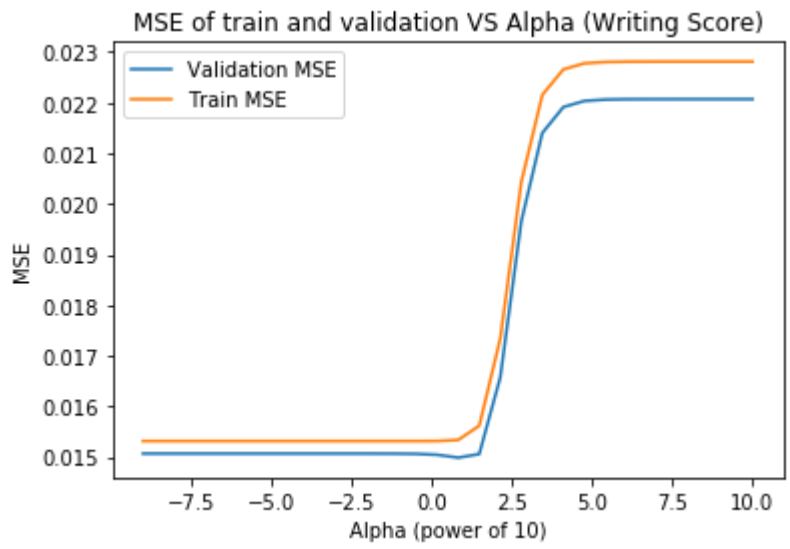
-Writing Score

In [92]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[new_data.columns[:-4]],train['writing score'])
    preds=model.predict(validation[new_data.columns[:-4]])
    MSE=sum([(r-p)**2 for r,p in zip(validation['writing score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['writing score'],model.predict(train[new_data.columns
[:-4]]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-----------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Writing Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.015067299634438515
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.015067299634376828
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.015067299634097983
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.015067299632837539
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.0150672996271399
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.015067299601384547
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.015067299484961287
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.015067298958688026
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.01506729657976821
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.015067285826647903
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.015067237227701934
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.015067017726315309
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.015066029236679228
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.0150616366831569
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.015043281090692796
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.014987079108406768
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.015058080859156728
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.016574669124297295
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.019655629520912585
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.02140085525282514
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.02191249597352203
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.02203368014437435
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.022060905968948715
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.02206694962822031
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.022068287633554395
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.02206858367908961
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.0220686491732035
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.022068663662037413
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.02206866686728669
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.02206866757635744
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=6.723357536499335, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[92]: <matplotlib.legend.Legend at 0x26f651ce8c8>



Has the same complexity as the one shown before.

Now we calculate the generalaization error

In [93]:
```
preds=bestModel.predict(test[new_data.columns[:-4]])
MSE=sum([(r-p)**2 for r,p in zip(test['writing score'],preds)])/len(preds)
print('MSE for test set writing Score: ', MSE)
```

```
MSE for test set writing Score:  0.01693878437160637
```

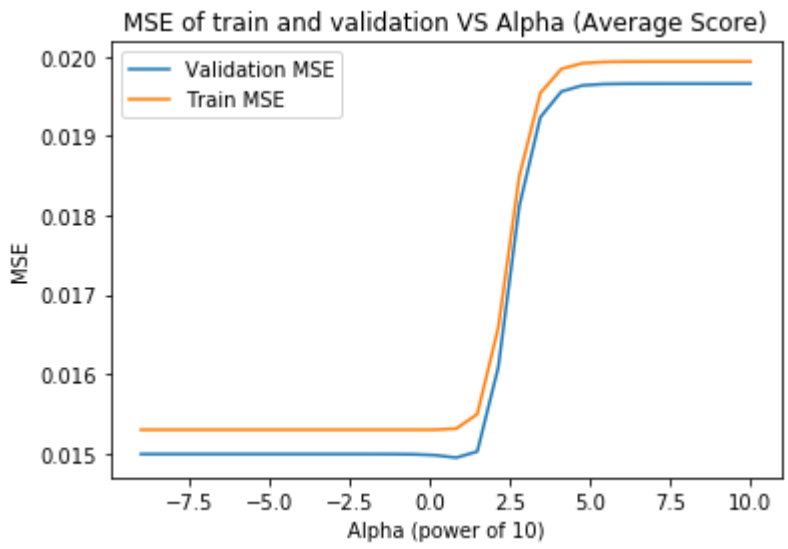
Average Score

In [94]:
```
valError=float('inf')
bestModel=None
mses=[]
msesTrain=[]
for power in lambdas:
    print('-------------------------------------------------')
    print('Alpha of: ',10**power)
    model=Ridge(alpha=10**power).fit(train[new_data.columns[:-4]],train['average score'])
    preds=model.predict(validation[new_data.columns[:-4]])
    MSE=sum([(r-p)**2 for r,p in zip(validation['average score'],preds)])/len(preds)
    print('Validation MSE: ',MSE)
    mses.append(MSE)
    msesTrain.append(sum([(r-p)**2 for r,p in zip(train['average score'],model.predict(train[new_data.columns
[:-4]]))])/len(train))
    if(MSE<valError):
        bestModel=model
        valError=MSE
print('-------------------------------------------------')
print('Best Model: ',bestModel)
plt.title('MSE of train and validation VS Alpha (Average Score)')
plt.xlabel('Alpha (power of 10)')
plt.ylabel('MSE')
plt.plot(lambdas,mses, label='Validation MSE')
plt.plot(lambdas,msesTrain, label='Train MSE')
plt.legend()
```

```
--------------------------------------------------
Alpha of:  1e-09
Validation MSE:  0.014988043138041083
--------------------------------------------------
Alpha of:  4.520353656360241e-09
Validation MSE:  0.01498804313800411
--------------------------------------------------
Alpha of:  2.0433597178569395e-08
Validation MSE:  0.01498804313783701
--------------------------------------------------
Alpha of:  9.236708571873865e-08
Validation MSE:  0.014988043137081644
--------------------------------------------------
Alpha of:  4.1753189365604003e-07
Validation MSE:  0.014988043133667122
--------------------------------------------------
Alpha of:  1.8873918221350957e-06
Validation MSE:  0.014988043118232217
--------------------------------------------------
Alpha of:  8.531678524172814e-06
Validation MSE:  0.014988043048461113
--------------------------------------------------
Alpha of:  3.856620421163472e-05
Validation MSE:  0.014988042733071688
--------------------------------------------------
Alpha of:  0.00017433288221999874
Validation MSE:  0.014988041307414773
--------------------------------------------------
Alpha of:  0.0007880462815669905
Validation MSE:  0.014988034863242728
--------------------------------------------------
Alpha of:  0.003562247890262437
Validation MSE:  0.014988005739465729
--------------------------------------------------
Alpha of:  0.01610262027560939
Validation MSE:  0.014987874215511638
--------------------------------------------------
Alpha of:  0.07278953843983146
Validation MSE:  0.014987282247415237
--------------------------------------------------
Alpha of:  0.3290344562312671
Validation MSE:  0.01498465840212653
--------------------------------------------------
Alpha of:  1.4873521072935119
Validation MSE:  0.014973826849581347
--------------------------------------------------
Alpha of:  6.723357536499335
Validation MSE:  0.01494310178791182
--------------------------------------------------
Alpha of:  30.39195382313195
Validation MSE:  0.01502001869892487
--------------------------------------------------
Alpha of:  137.3823795883261
Validation MSE:  0.016084147687417942
--------------------------------------------------
Alpha of:  621.0169418915604
Validation MSE:  0.01811774784346582
--------------------------------------------------
Alpha of:  2807.21620394117
Validation MSE:  0.019238185909733318
--------------------------------------------------
Alpha of:  12689.610031679182
Validation MSE:  0.01956341341170717
--------------------------------------------------
Alpha of:  57361.52510448681
Validation MSE:  0.01964024530140032
--------------------------------------------------
Alpha of:  259294.3797404667
Validation MSE:  0.019657496402677454
--------------------------------------------------
Alpha of:  1172102.2975334793
Validation MSE:  0.019661325335292797
--------------------------------------------------
Alpha of:  5298316.906283702
Validation MSE:  0.019662172997325554
--------------------------------------------------
Alpha of:  23950266.199874908
Validation MSE:  0.019662360548831147
--------------------------------------------------
Alpha of:  108263673.38740563
Validation MSE:  0.019662402040765282
--------------------------------------------------
Alpha of:  489390091.8477499
Validation MSE:  0.019662411219751198
--------------------------------------------------
Alpha of:  2212216291.0704503
Validation MSE:  0.019662413250344978
--------------------------------------------------
Alpha of:  10000000000.0
Validation MSE:  0.019662413699556413
--------------------------------------------------
```

```
Best Model:  Ridge(alpha=6.723357536499335, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Out[94]: `<matplotlib.legend.Legend at 0x26f652505c8>`



Has the same complesxity as the one before

Now we calculate the generalaization error

In [95]:
```
preds=bestModel.predict(test[new_data.columns[:-4]])
MSE=sum([(r-p)**2 for r,p in zip(test['average score'],preds)])/len(preds)
print('MSE for test set average score: ', MSE)
```

```
MSE for test set average score:  0.01707478235435576
```

Conclusions

In the end we can conclude that for this dataset the more accurate models tend to have a relative high complexity. Also, in the end the best model was the one that took into account all of the features to predict the scores and the average score, but the true error (Test error) is still too high which could mean a lot of noise that may come due to the many outliers.