In [1]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import os
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans
```

In [2]:

```
data = pd.read_csv('/home/praveen/Desktop/SEM/ML/StudentsPerformance.csv')
data.rename(columns={"race/ethnicity":"ethnicity","parental level of education":"pa
data.head()
```

Out[2]:

	gender	ethnicity	parent_education	lunch	pre	math	reading	writing
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.dtypes
```

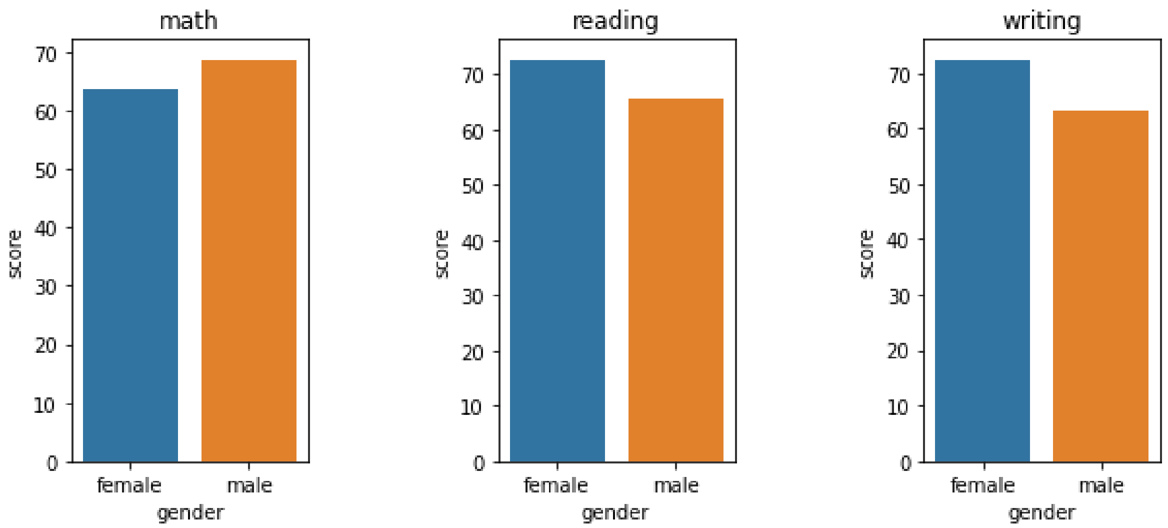
Out[3]:

```
gender              object
ethnicity           object
parent_education    object
lunch               object
pre                 object
math                 int64
reading              int64
writing              int64
dtype: object
```

In [4]:

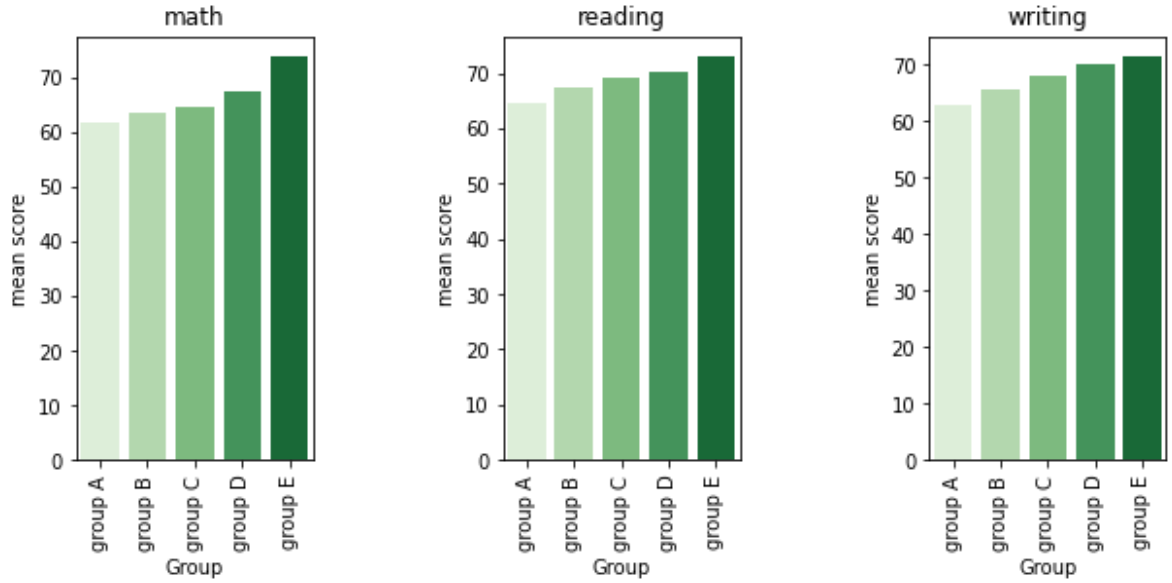
```
fig, ax = plt.subplots()
fig.subplots_adjust(hspace=0.8, wspace=0.8, left = 0.2, right = 1.5)
for idx in range(3):
    plt.subplot(1,3, idx+1)
    gender_df = data.groupby("gender")[list(data.columns[-3:])[idx]].describe()
    sns.barplot(gender_df.index, gender_df.loc[:,"mean"].values)
    plt.ylabel("score")
    plt.title(list(data.columns[-3:])[idx])

plt.show()
```

```
/home/praveen/anaconda3/lib/python3.8/site-packages/seaborn/_decorator
s.py:36: FutureWarning: Pass the following variables as keyword args:
x, y. From version 0.12, the only valid positional argument will be `d
ata`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
/home/praveen/anaconda3/lib/python3.8/site-packages/seaborn/_decorator
s.py:36: FutureWarning: Pass the following variables as keyword args:
x, y. From version 0.12, the only valid positional argument will be `d
ata`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
/home/praveen/anaconda3/lib/python3.8/site-packages/seaborn/_decorator
s.py:36: FutureWarning: Pass the following variables as keyword args:
x, y. From version 0.12, the only valid positional argument will be `d
ata`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```



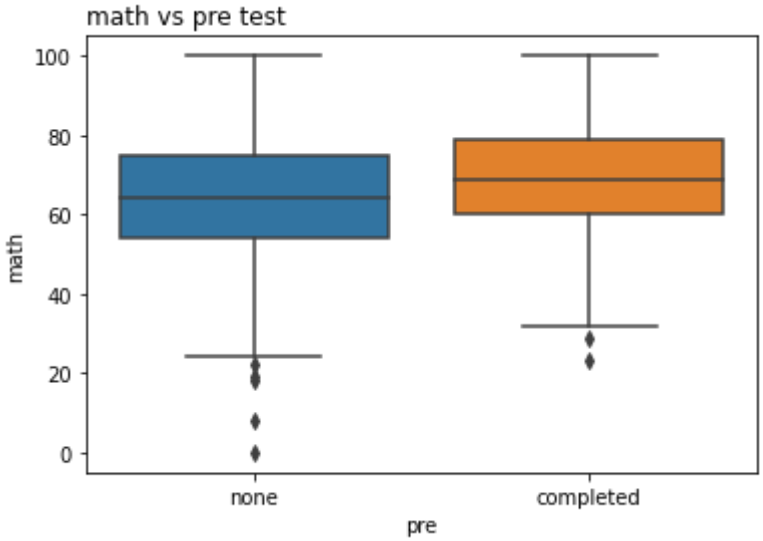
In [5]:

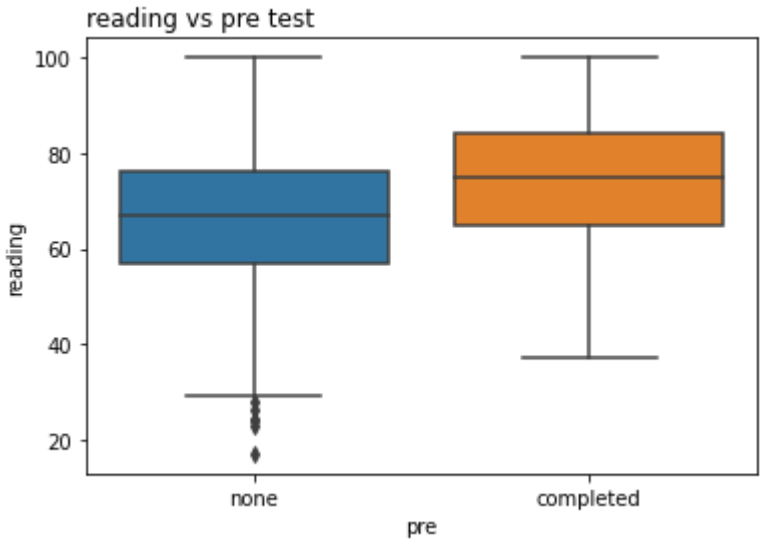
```
fig, ax = plt.subplots()
fig.subplots_adjust(hspace=0.8, wspace=0.8, left = 0.2, right = 1.5)
for idx in range(3):
    plt.subplot(1,3, idx+1)
    ethn_df = data.groupby("ethnicity")[list(data.columns[-3:])[idx]].mean()
    sns.barplot(x=ethn_df.index, y = ethn_df.values, palette = "Greens")
    plt.xlabel("Group")
    plt.ylabel("mean score")
    plt.xticks(rotation=90)
    plt.title(list(data.columns[-3:])[idx])
plt.show()
```

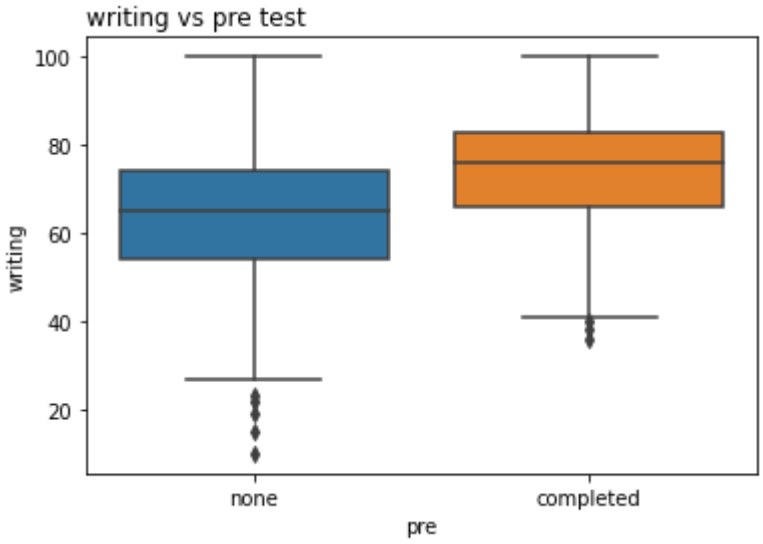


In [6]:

```
for item in data.columns[-3:]:
    sns.boxplot(x=data["pre"], y=data[item])
    plt.title(item+" vs pre test", loc="left")
    plt.show()
```

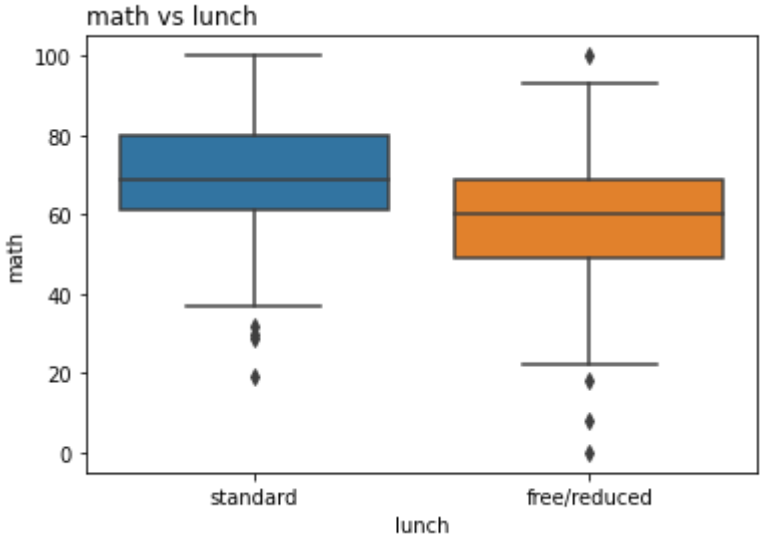


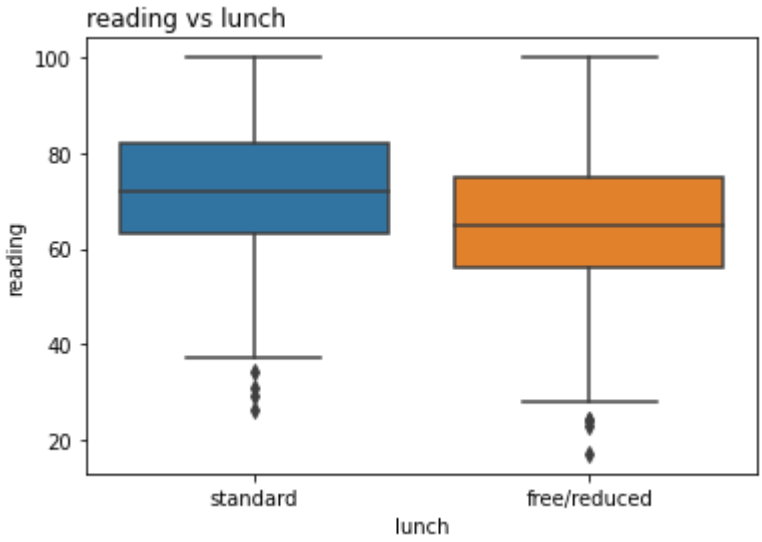


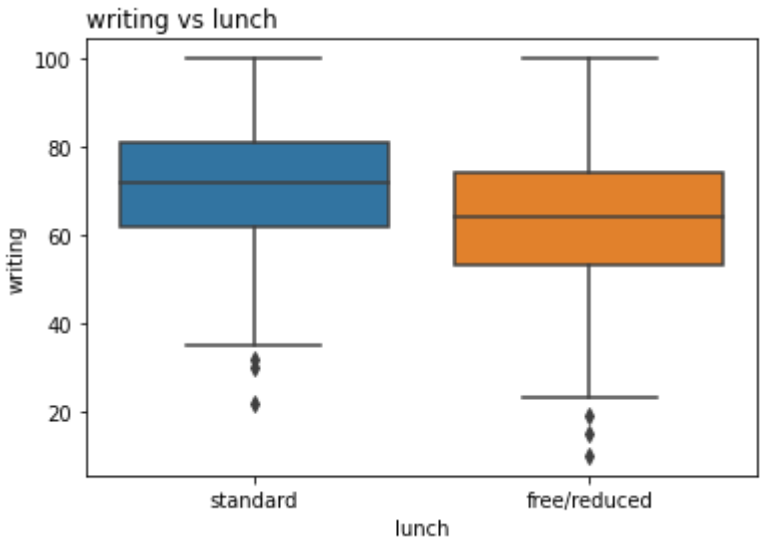


In [7]:

```
for item in data.columns[-3:]:
    sns.boxplot(x=data["lunch"], y=data[item])
    plt.title(item+" vs lunch", loc="left")
    plt.show()
```







In [8]:

```
labelencoder = LabelEncoder()
train_df = data.copy()
train_df["parent_education"] = labelencoder.fit_transform(train_df["parent_educatio
train_df["pre"] = labelencoder.fit_transform(train_df["pre"])
train_df["lunch"] = labelencoder.fit_transform(train_df["lunch"])
train_df.head()
```

Out[8]:

	gender	ethnicity	parent_education	lunch	pre	math	reading	writing
0	female	group B	1	1	1	72	72	74
1	female	group C	4	1	0	69	90	88
2	female	group B	3	1	1	90	95	93
3	male	group A	0	0	1	47	57	44
4	male	group C	4	1	1	76	78	75

In [27]:

```
kmeans = KMeans(init = "k-means++", n_clusters = 8)
kmeans.fit_transform(train_df.iloc[:, 2:])
kmeans_label = kmeans.labels_
data["classification"] = kmeans_label
data.head(10)
```

Out[27]:

	gender	ethnicity	parent_education	lunch	pre	math	reading	writing	classific
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	
5	female	group B	associate's degree	standard	none	71	83	78	
6	female	group B	some college	standard	completed	88	95	92	
7	male	group B	some college	free/reduced	none	40	43	39	
8	male	group D	high school	free/reduced	completed	64	64	67	
9	female	group B	high school	free/reduced	none	38	60	50	

In [26]:

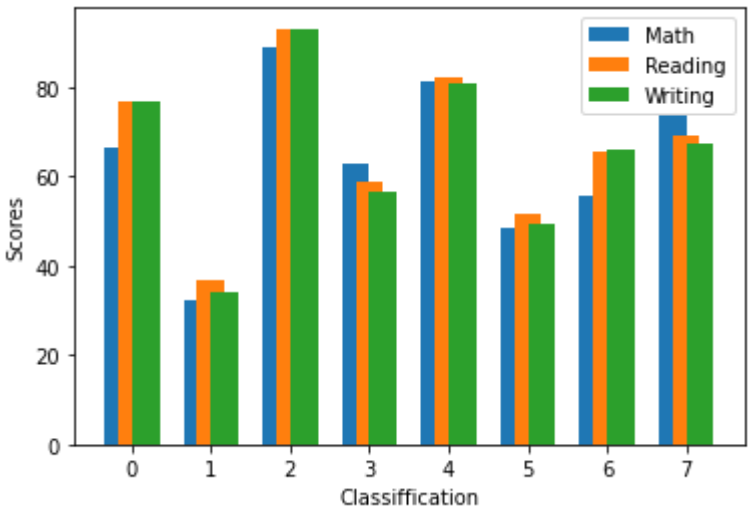
```
class_df = data.groupby("classification")[data.columns[-4:-1]].mean()
class_df
```

Out[26]:

classification	math	reading	writing
0	66.631579	76.801170	76.555556
1	32.358974	36.589744	34.025641
2	88.967391	93.076087	92.836957
3	62.991525	58.601695	56.338983
4	81.358025	82.043210	80.851852
5	48.673611	51.437500	49.284722
6	55.845070	65.584507	65.859155
7	73.454545	69.090909	67.424242

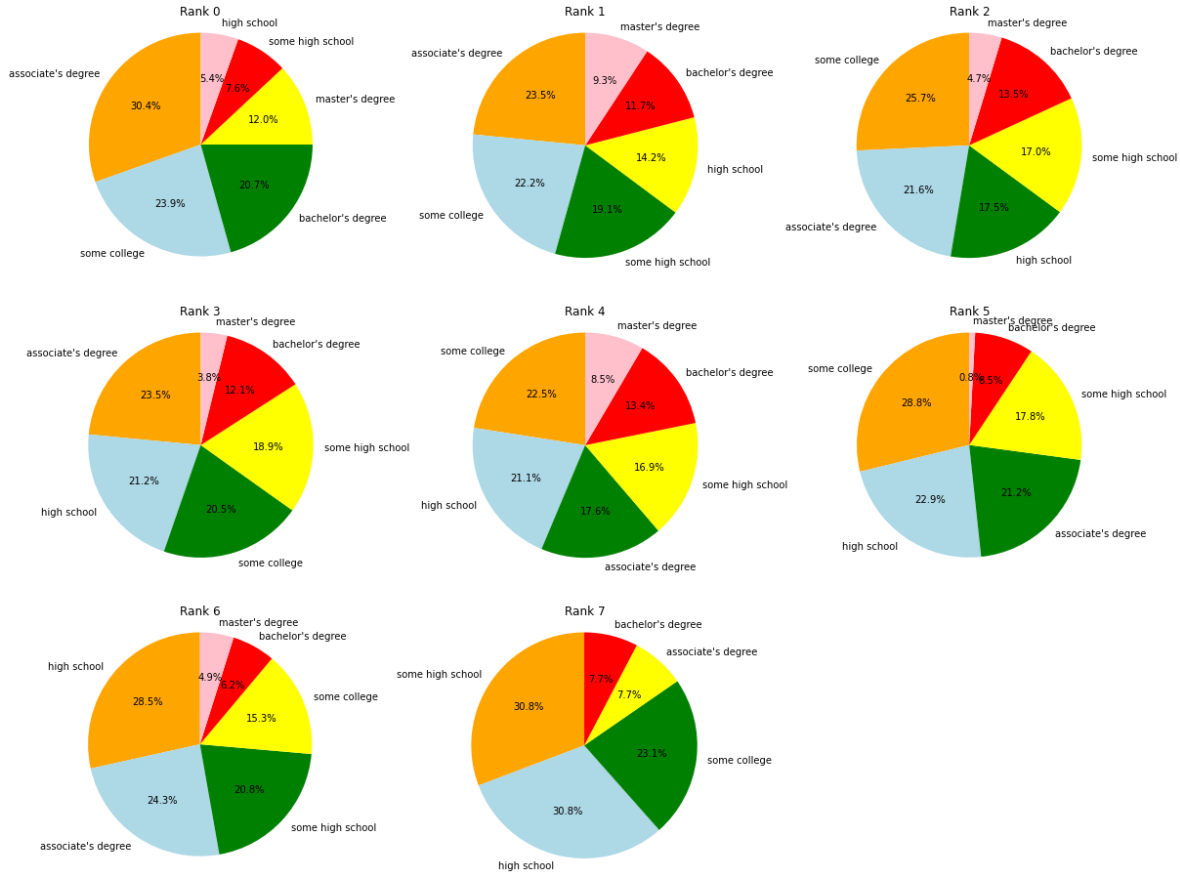
In [12]:

```
ind = np.arange(8)
width = 0.35
fig, ax = plt.subplots()
rects1 = ax.bar(ind - width/2, class_df.math, width, label='Math')
rects2 = ax.bar(ind, class_df.reading, width, label='Reading')
rects3 = ax.bar(ind + width/2, class_df.writing, width, label='Writing')

ax.set_xlabel('Classiffication')
ax.set_ylabel('Scores')
ax.set_xticks(ind)
ax.legend()
plt.show()
```



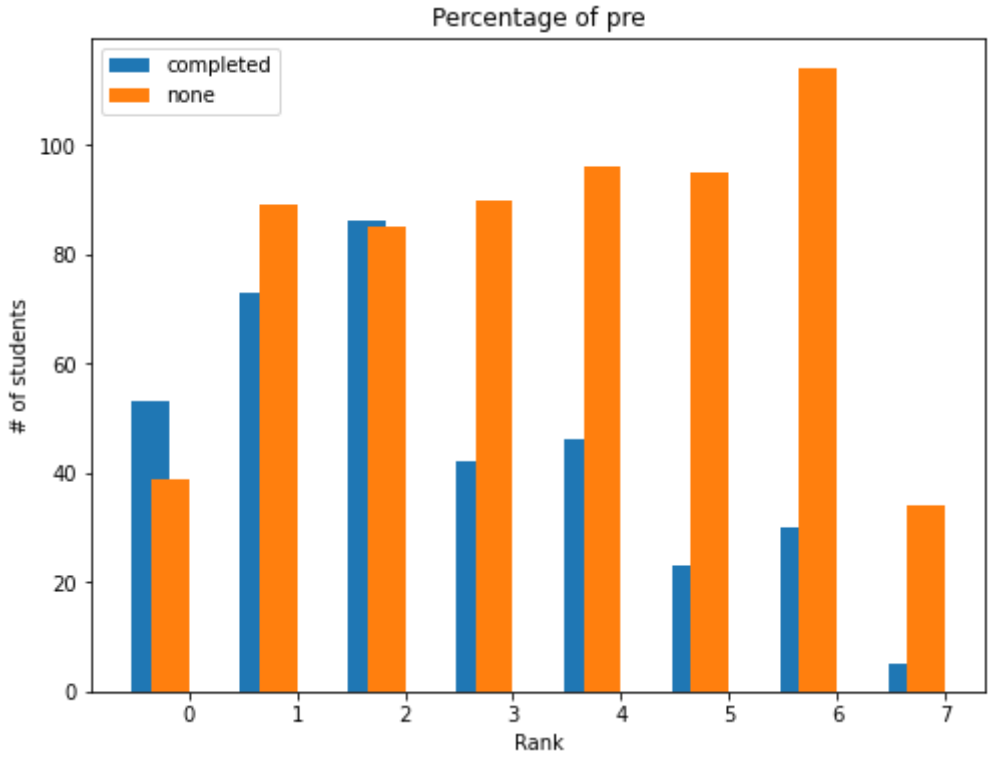
In [13]:

```
class_df["total_ave_score"] = (class_df.math + class_df.reading + class_df.writing)
rank = class_df["total_ave_score"].sort_values(ascending = False)
rank.index

def plot_pie_chart(column):
    fig, ax = plt.subplots(figsize=(20,16))
    color = ["orange","lightblue","green","yellow","red","pink","brown","gray"]
    for idx in range(8):
        plt.subplot(3, 3, idx+1)
        num = "class"+ str(idx)
        num = data[data["classification"]==rank.index[idx]]
        percentage_of_parent_edu = num[column].value_counts()
        percentage_of_parent_edu.sort_index()
        label = percentage_of_parent_edu.index
        value = percentage_of_parent_edu.values
        plt.pie(value, labels = label, autopct = "%1.1f%%",startangle=90, radius =
        plt.axis("equal")
        plt.title("Rank "+str(idx))
    plt.show()
plot_pie_chart("parent_education")
```



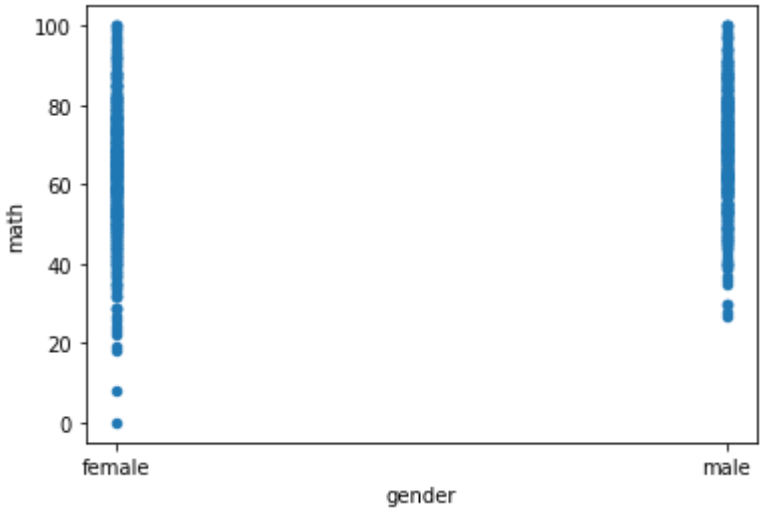
In [14]:

```
def plot_bar_chart(column):
    fig, ax = plt.subplots(figsize=(8,6))
    index_dict = dict()
    width = 0.35
    ind = np.arange(8)
    for idx in range(8):
        num = "class"+ str(idx)
        num = data[data["classification"]==rank.index[idx]]
        percentage_of_column = num[column].value_counts()
        percentage_of_column = percentage_of_column.sort_index()
        for key in percentage_of_column.index:
            if key not in index_dict.keys():
                index_dict[key] = []
                index_dict[key].append(percentage_of_column[key])
            else:
                index_dict[key].append(percentage_of_column[key])

    percentage_of_column = data[data["classification"]==rank.index[4]][column].valu
    for i in range(len(percentage_of_column.index)):
        rects = ax.bar(ind - width/(i+1),
                       index_dict[percentage_of_column.index[i]],
                       width, label=percentage_of_column.index[i])

    ax.set_xlabel('Rank')
    ax.set_ylabel('# of students')
    ax.set_title("Percentage of " + column)
    ax.set_xticks(ind)
    ax.legend()
    plt.show()

plot_bar_chart("pre")
```



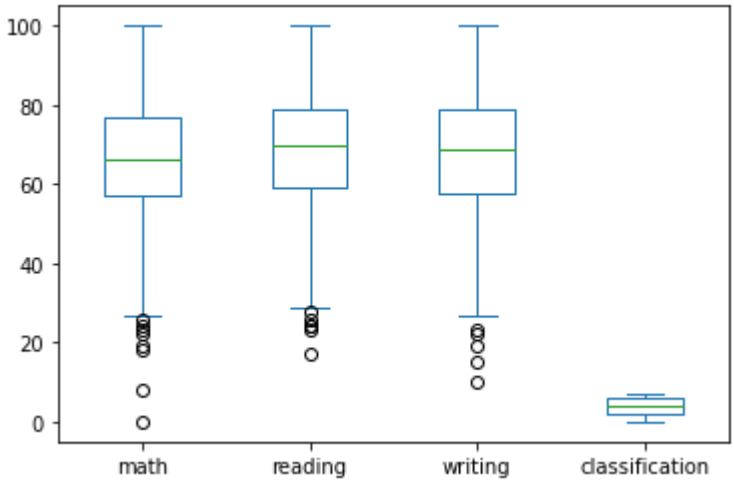
In [15]:

```
data.plot(kind='scatter',x='gender',y='math')
plt.show()
```



In [16]:

```
data.plot(kind='box')
plt.show()
```



In [17]:

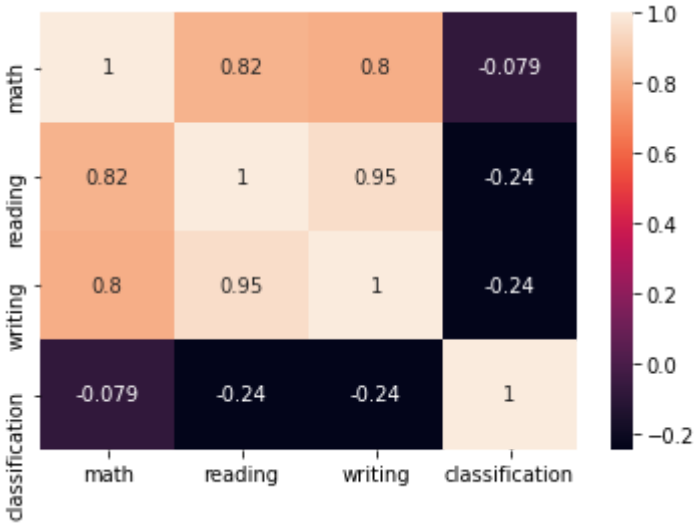
```
import seaborn as s
```

In [18]:

```
s.heatmap(data.corr(),annot=True)
```

Out[18]:

<AxesSubplot:>

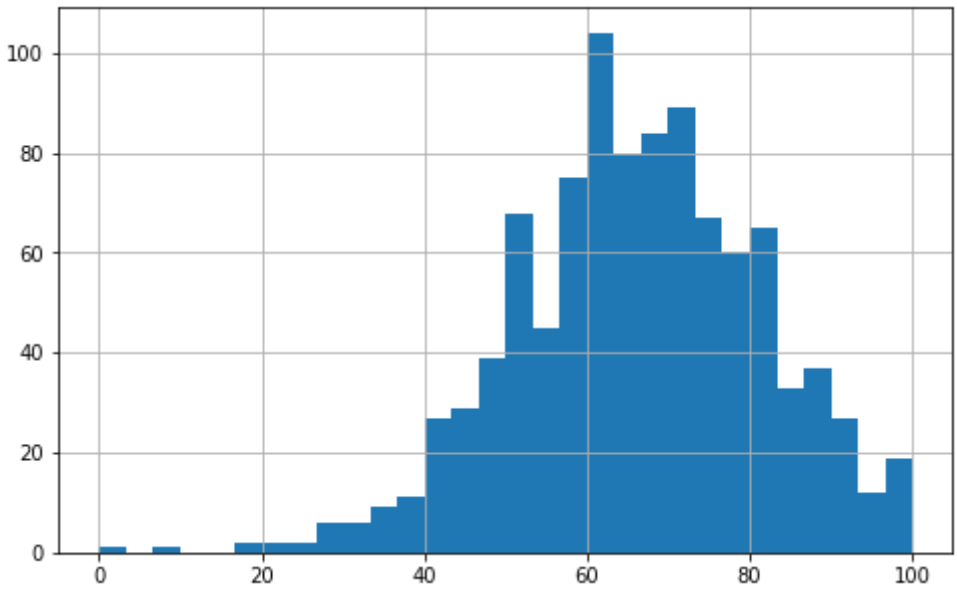


In [ ]:

```

```

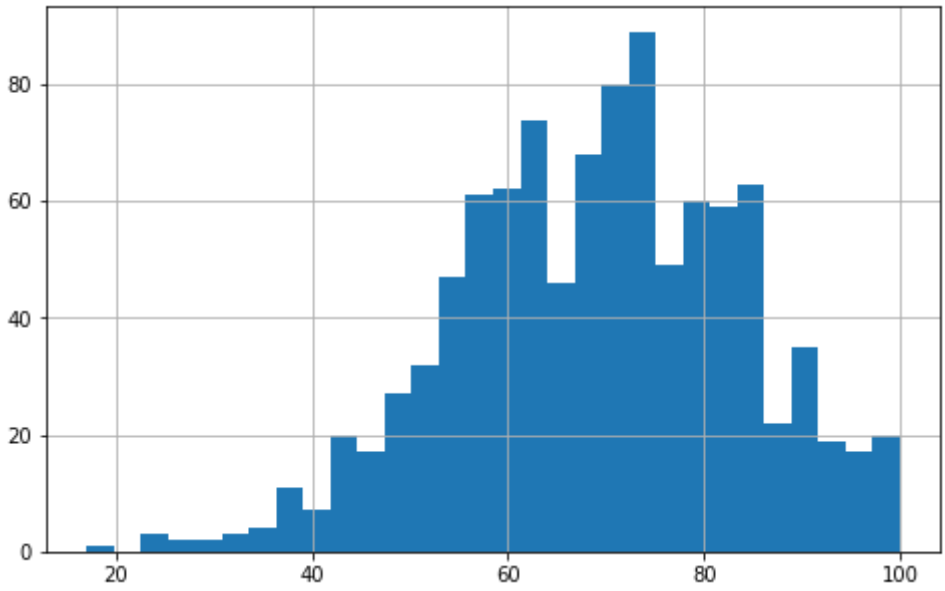
In [20]:

```
data['math'].hist(bins=30, figsize=(8, 5));
```



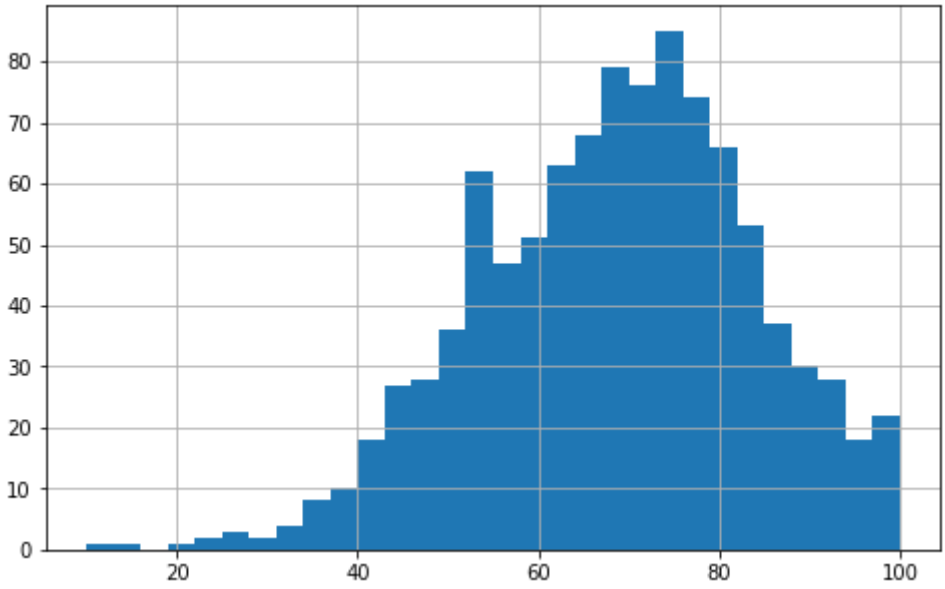
In [21]:

```
data['reading'].hist(bins=30, figsize=(8, 5));
```



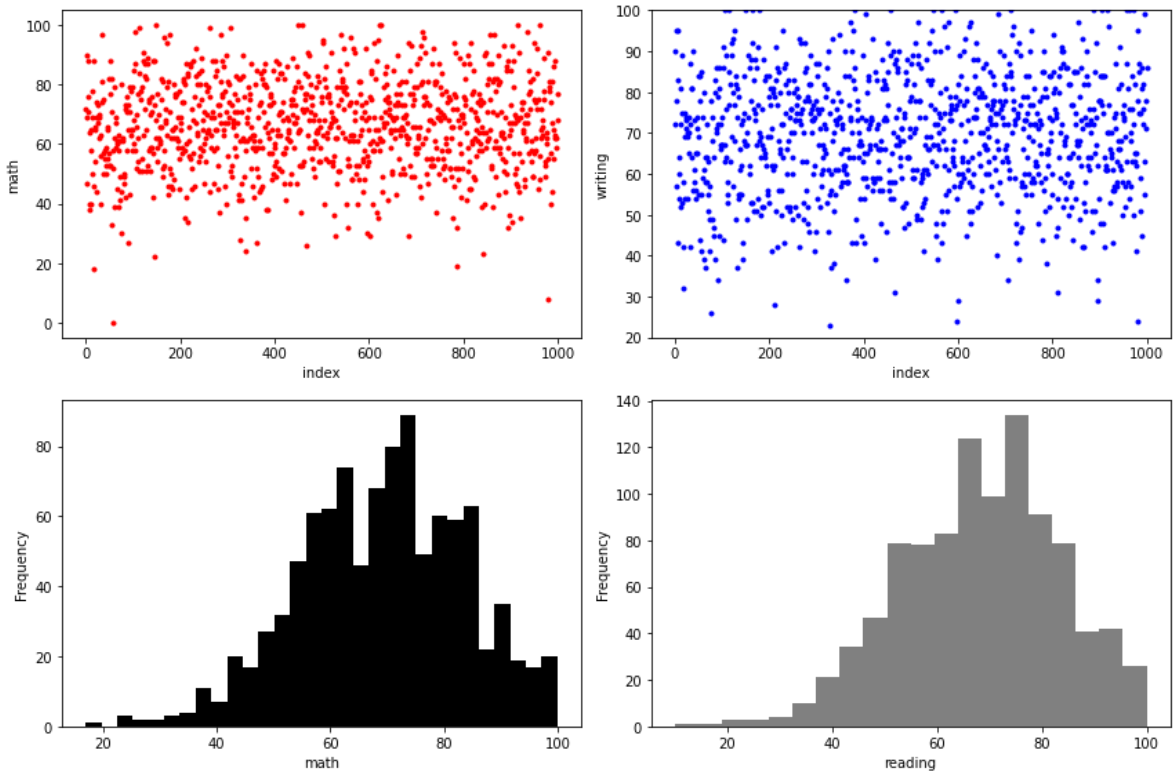
In [22]:

```
data['writing'].hist(bins=30, figsize=(8, 5));
```



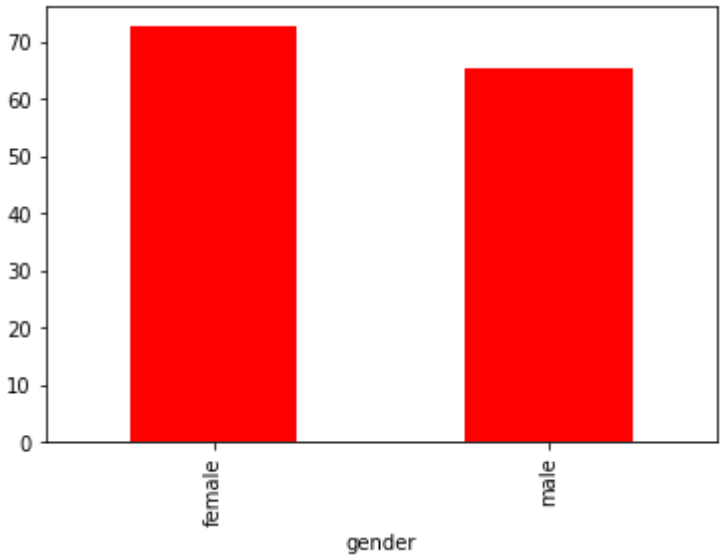
In [23]:

```
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
# or fig, (ax1, ax2, ax3, ax4) = plt.subplots(2, 2, figsize=(12, 8))

# axes is the axes object(s). It can be a single object or an array of objects.
# In this case, it is an array of dimension 2-by-2

data['math'].plot(ax = axes[0][0], style='.', color='red') # top left
data['reading'].plot(ax = axes[0][1], style='.', color='blue') # top right

data['reading'].plot.hist(bins=30, ax = axes[1][0], color='black') # bottom left
data['writing'].plot.hist(bins=20, ax = axes[1][1], color='gray') # bottom right

axes[0][0].set_xlabel('index')
axes[0][1].set_xlabel('index')
axes[1][0].set_xlabel('math')
axes[1][1].set_xlabel('reading')

axes[0][0].set_ylabel('math')
axes[0][1].set_ylabel('writing')

axes[0][1].set_ylim(20, 100)


fig.tight_layout()
```



In [24]:

```
data_avg_rate = data.groupby('gender')['reading'].mean()
data_avg_rate[:5].plot.bar(color='red');
```

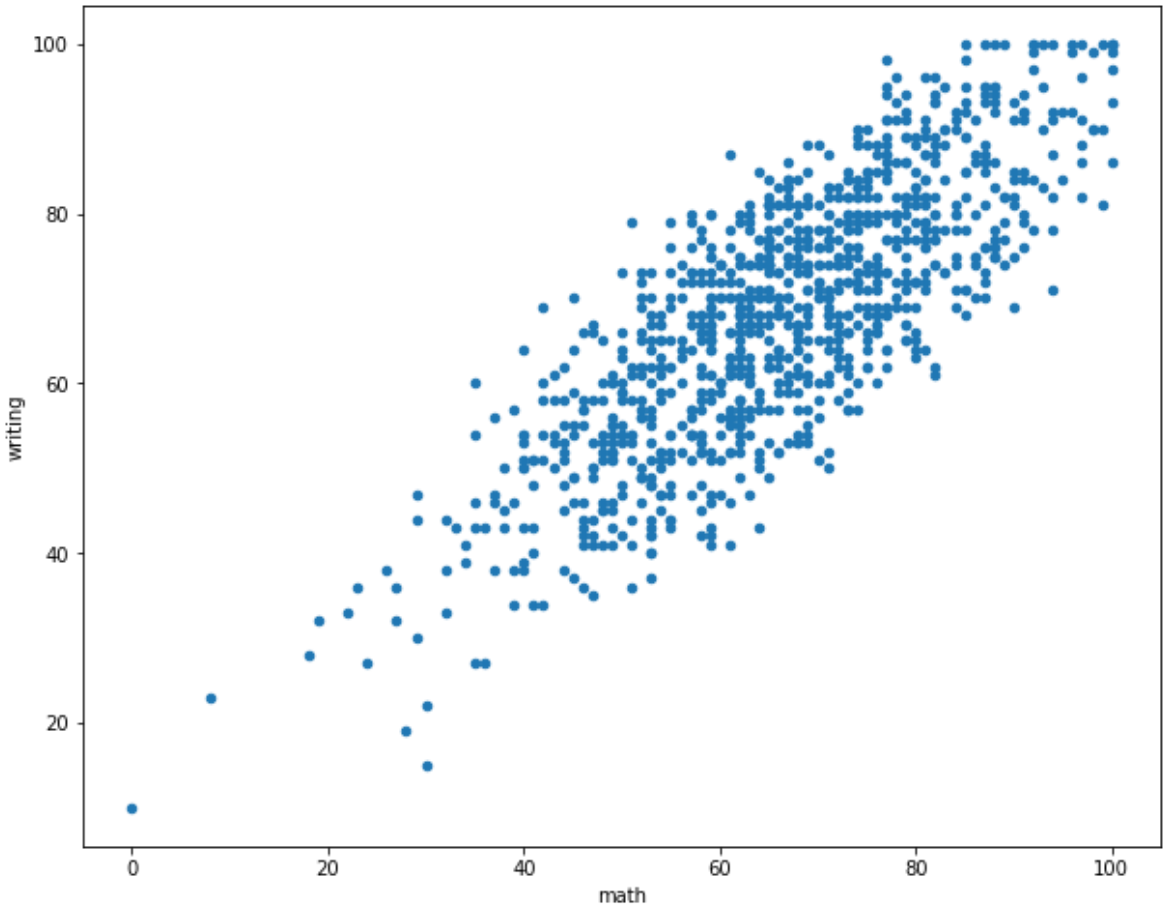


In [25]:

```
data.plot.scatter('math', 'writing', figsize=(10, 8))
```

Out[25]:

```
<AxesSubplot:xlabel='math', ylabel='writing'>
```



In [64]:

```
data.groupby(['gender']).mean()
```

Out[64]:

gender	math	reading	writing	classification	average
female	63.633205	72.608108	72.467181	4.162162	69.569498
male	68.728216	65.473029	63.311203	3.520747	65.837483

In [65]:

```
data['average'] = data[['math', 'reading','writing']].mean(axis=1)

SP_csv_clean = data.copy()
math_score = SP_csv_clean["math"]
reading_score = SP_csv_clean["reading"]
writing_score = SP_csv_clean["writing"]
average_score = SP_csv_clean["average"]
X_features = SP_csv_clean.drop(["math","reading","writing","average"],axis = 'colum
```

In [66]:

```
X_features_encoded = X_features.apply(lambda x: x.astype('category'))
X_features_encoded = pd.get_dummies(X_features_encoded,drop_first= True)
```

In [67]:

```
target = average_score
train_X, valid_X, train_y, valid_y = train_test_split(X_features_encoded, target, t
```

In [73]:

```
from sklearn.linear_model import LinearRegression
LinearReg = LinearRegression(normalize = True).fit(train_X,train_y)
```

In [74]:

```
LinearReg = LinearRegression(normalize = True).fit(train_X,train_y)
def evaluateRegressor(true,predicted,message = "Test set"):
    MSE = mean_squared_error(true,predicted,squared = True)
    MAE = mean_absolute_error(true,predicted)
    RMSE = mean_squared_error(true,predicted,squared = False)
    R_squared = r2_score(true,predicted)
    print(message)
    print("MSE:", MSE)
    print("MAE:", MAE)
    print("RMSE:", RMSE)
    print("R-squared:", R_squared)
```

In [75]:

```
print("Linear Regression")
predicted_train_y = LinearReg.predict(train_X)
evaluateRegressor(train_y,predicted_train_y,"    Training Set")
predicted_valid_y = LinearReg.predict(valid_X)
evaluateRegressor(valid_y,predicted_valid_y,"    Test Set")
print("\n")
```

```
Linear Regression
    Training Set
MSE: 10.3811147817894
MAE: 2.617732002611774
RMSE: 3.2219737400837705
R-squared: 0.9483268074354274
    Test Set
MSE: 14.528023976441272
MAE: 2.9401063622363517
RMSE: 3.811564505087284
R-squared: 0.9313232136430748
```

In [85]:

```
kf =KFold(n_splits=5, shuffle=True, random_state=42)

cnt = 1
for train_index, test_index in kf.split(train_X, train_y):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test set:{len(test_index)}')
    cnt += 1
```

```
Fold:1, Train set: 640, Test set:160
Fold:2, Train set: 640, Test set:160
Fold:3, Train set: 640, Test set:160
Fold:4, Train set: 640, Test set:160
Fold:5, Train set: 640, Test set:160
```

In [86]:

```
def rmse(score):
    rmse = np.sqrt(-score)
    print(f'rmse= {"{:.2f}".format(rmse)}')
```

In [88]:

```
score = cross_val_score(linear_model.LinearRegression(), train_X, train_y, cv= kf,
print(f'Scores for each fold: {score}')
rmse(score.mean())
```

```
Scores for each fold: [-11.22664229 -11.62608391 -10.80317186 -12.3510
8408  -9.79973225]
rmse= 3.34
```

In [90]:

```
max_depth = [1,2,3,4,5,6,7,8,9,10]

for val in max_depth:
    score = cross_val_score(tree.DecisionTreeRegressor(max_depth= val, random_state
    print(f'For max depth: {val}')
    rmse(score.mean())
```

```
For max depth: 1
rmse= 12.17
For max depth: 2
rmse= 9.88
For max depth: 3
rmse= 8.10
For max depth: 4
rmse= 6.62
For max depth: 5
rmse= 5.42
For max depth: 6
rmse= 4.27
For max depth: 7
rmse= 3.65
For max depth: 8
rmse= 3.81
For max depth: 9
rmse= 3.97
For max depth: 10
rmse= 4.11
```

In [ ]:

```
_
```