## Importing Library

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

import cufflinks as cf

from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

'''%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))'''
```

## Loading Dataset

```
data = pd.read_csv("/content/StudentsPerformance.csv")
data.head()
```

| | 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 |

```
data.tail()
```

| | gender | race/ethnicity | parental level of education | lunch | test preparation course | math score | reading score | writing score |
|---|---|---|---|---|---|---|---|---|
| 995 | female | group E | master's degree | standard | completed | 88 | 99 | 95 |
| 996 | male | group C | high school | free/reduced | none | 62 | 55 | 55 |
| 997 | female | group C | high school | free/reduced | completed | 59 | 71 | 65 |
| 998 | female | group D | some college | standard | completed | 68 | 78 | 77 |

```
data.info()
```

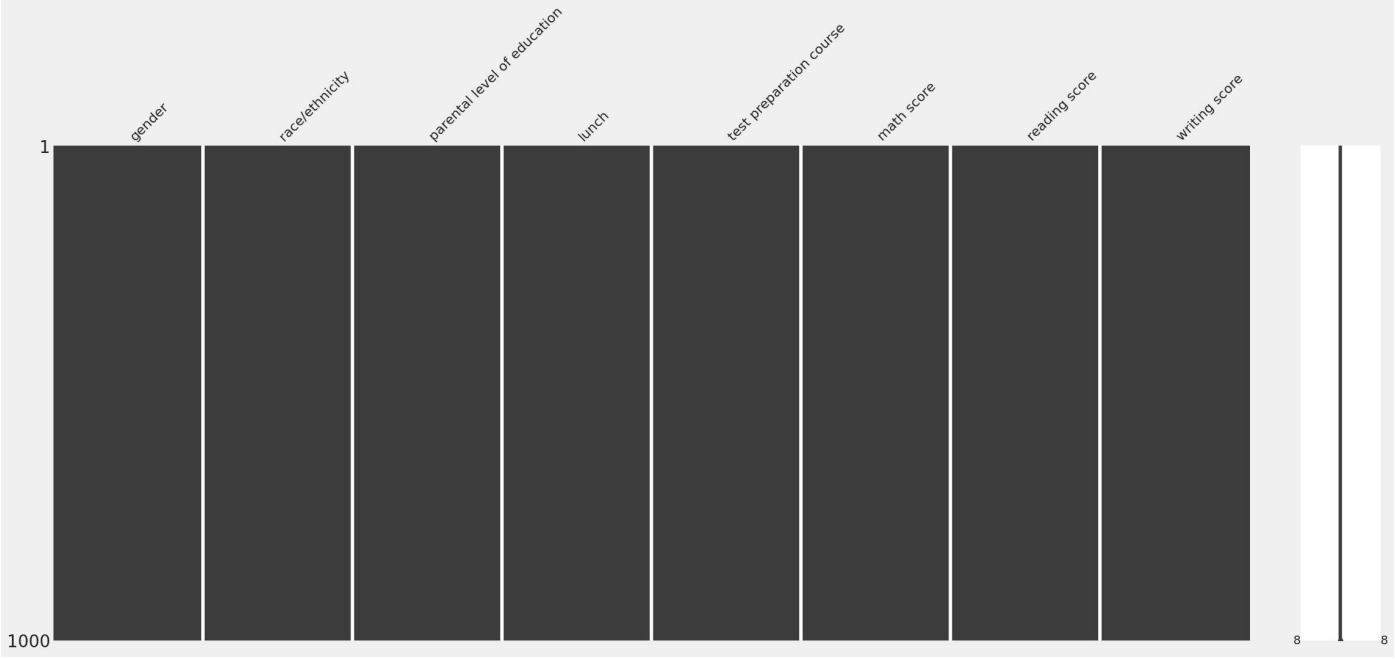
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64 
 6   reading score                1000 non-null   int64 
 7   writing score                1000 non-null   int64 
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

```
data.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **math score** | 1000.0 | 66.089 | 15.163080 | 0.0 | 57.00 | 66.0 | 77.0 | 100.0 |
| **reading score** | 1000.0 | 69.169 | 14.600192 | 17.0 | 59.00 | 70.0 | 79.0 | 100.0 |
| **writing score** | 1000.0 | 68.054 | 15.195657 | 10.0 | 57.75 | 69.0 | 79.0 | 100.0 |

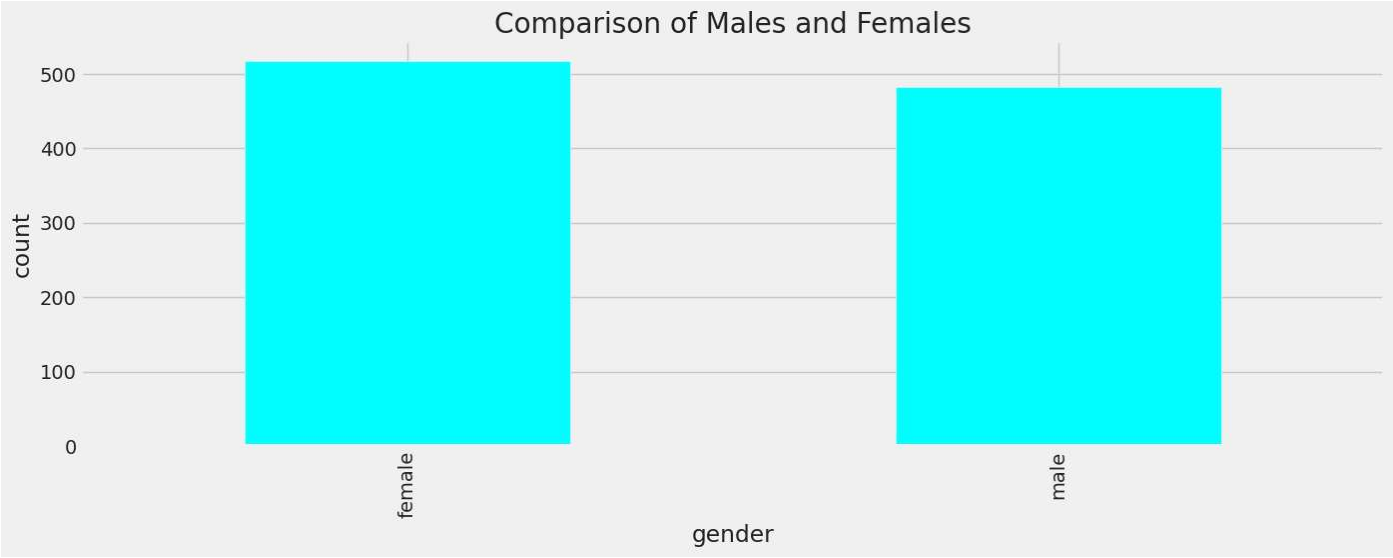
```
#### Visualizing the null values using missingo function

import missingno as msno
msno.matrix(data)
```

```
<Axes: >
```




## Data Visualization

```
# visualising the number of male and female in the dataset
plt.subplots(figsize=(15,5))
data['gender'].value_counts(normalize = True)
data['gender'].value_counts(dropna = False).plot.bar(color = 'cyan')
plt.title('Comparison of Males and Females')
plt.xlabel('gender')
plt.ylabel('count')
plt.show()
```

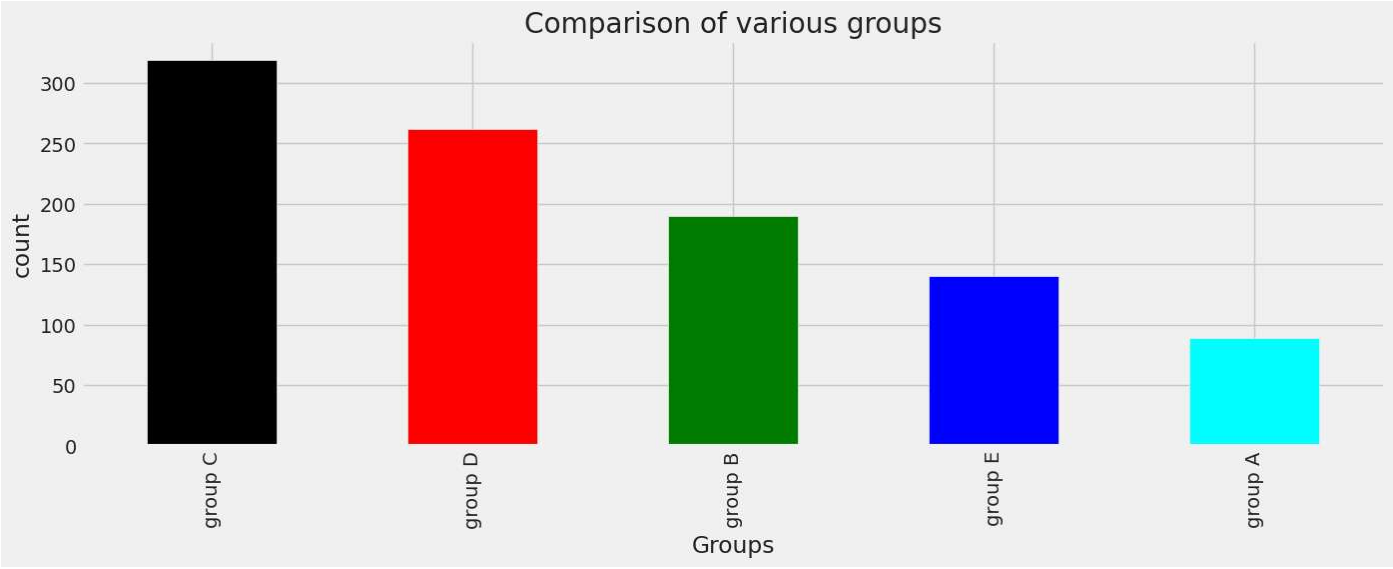



```
data['race/ethnicity'].value_counts()
```

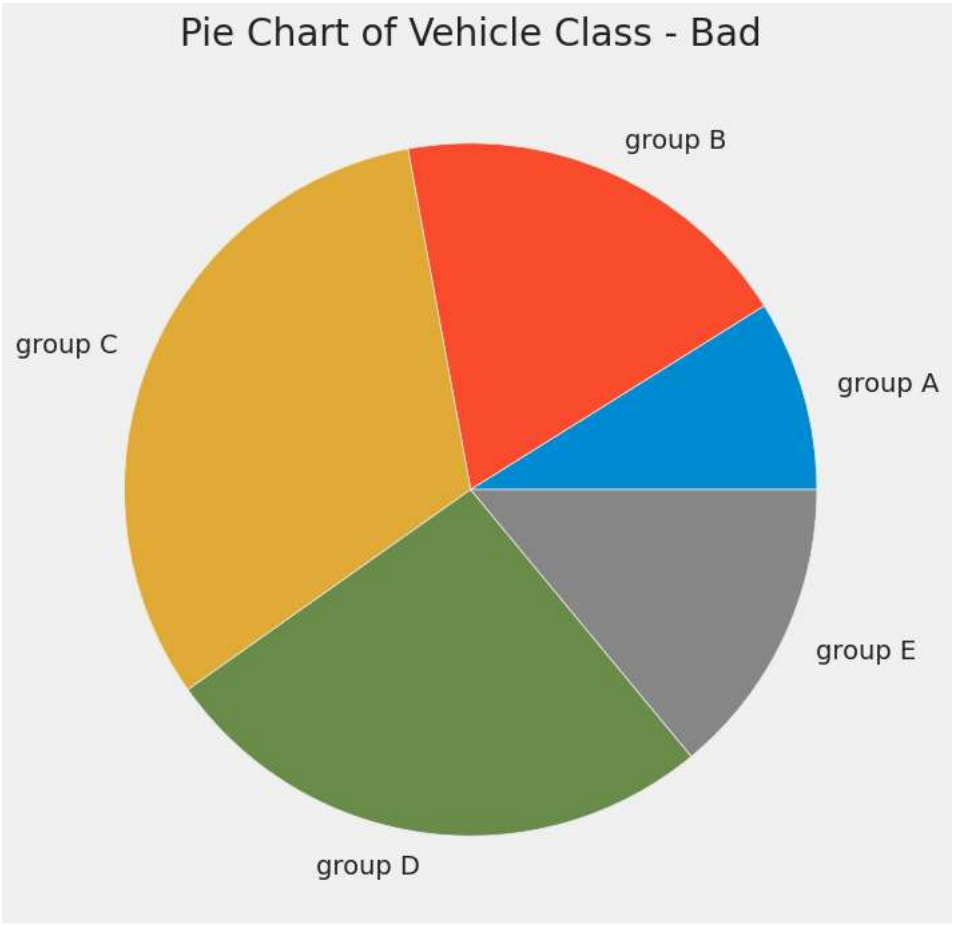
| race/ethnicity | count |
|---|---|
| group C | 319 |
| group D | 262 |
| group B | 190 |
| group E | 140 |
| group A | 89 |

dtype: int64

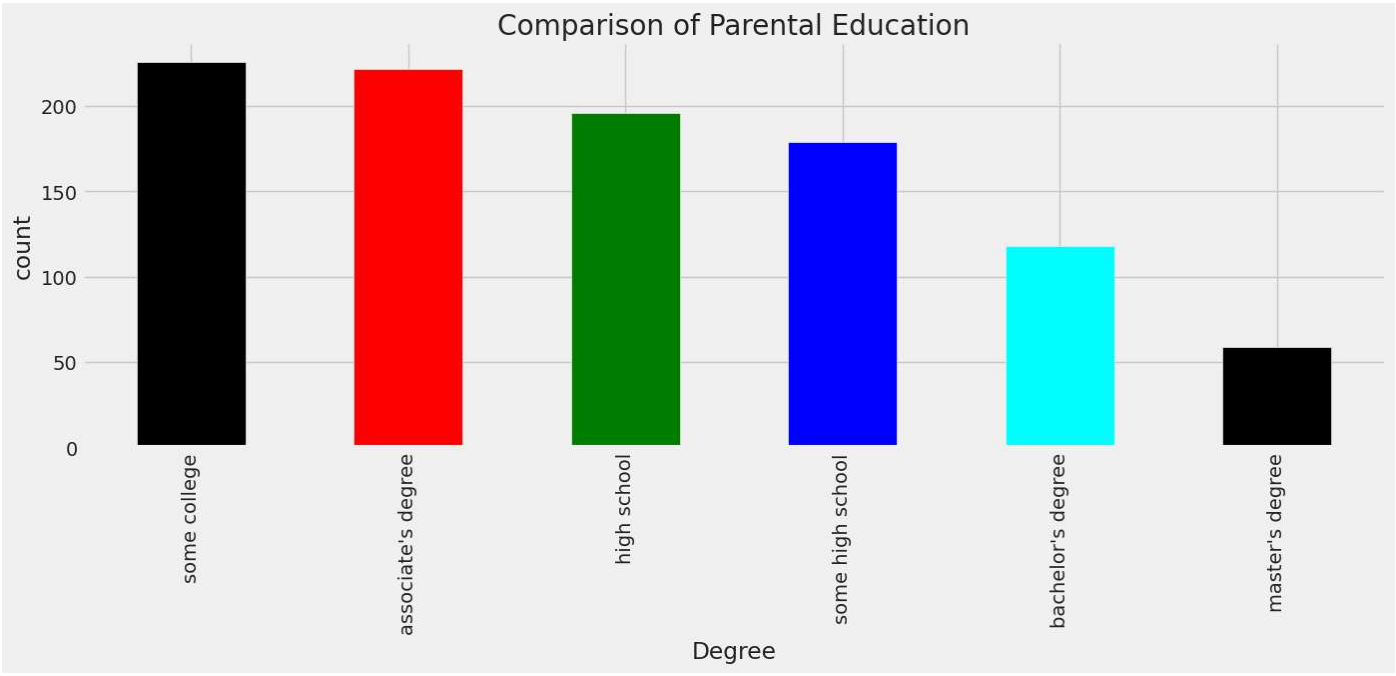
```
# visualizing the different groups in the dataset
plt.subplots(figsize=(15,5))
data['race/ethnicity'].value_counts(normalize = True)
data['race/ethnicity'].value_counts(dropna = False).plot.bar(color=['black', 'red', 'green', 'blue', 'cyan'])
plt.title('Comparison of various groups')
plt.xlabel('Groups')
plt.ylabel('count')
plt.show()
```



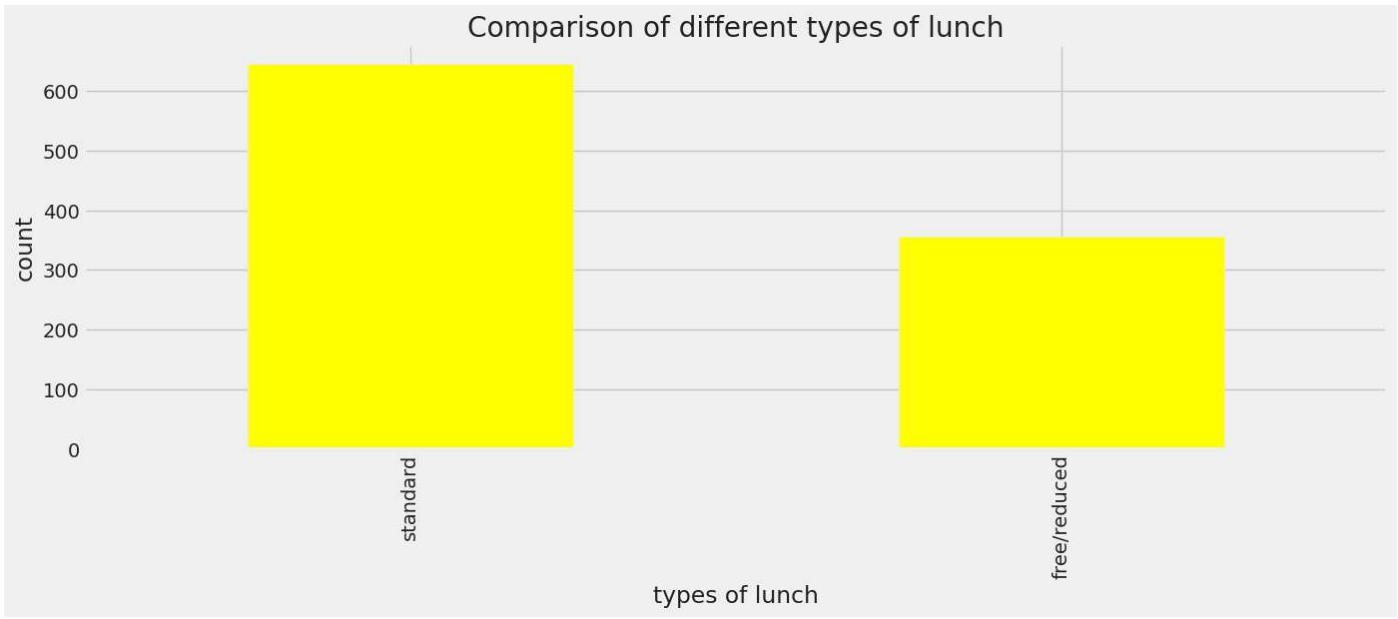

```
# Prepare Data
df = data.groupby('race/ethnicity').size()

# Make the plot with pandas
df.plot(kind='pie', subplots=True, figsize=(15, 8))
plt.title("Pie Chart of Vehicle Class - Bad")
plt.ylabel("")
plt.show()
```



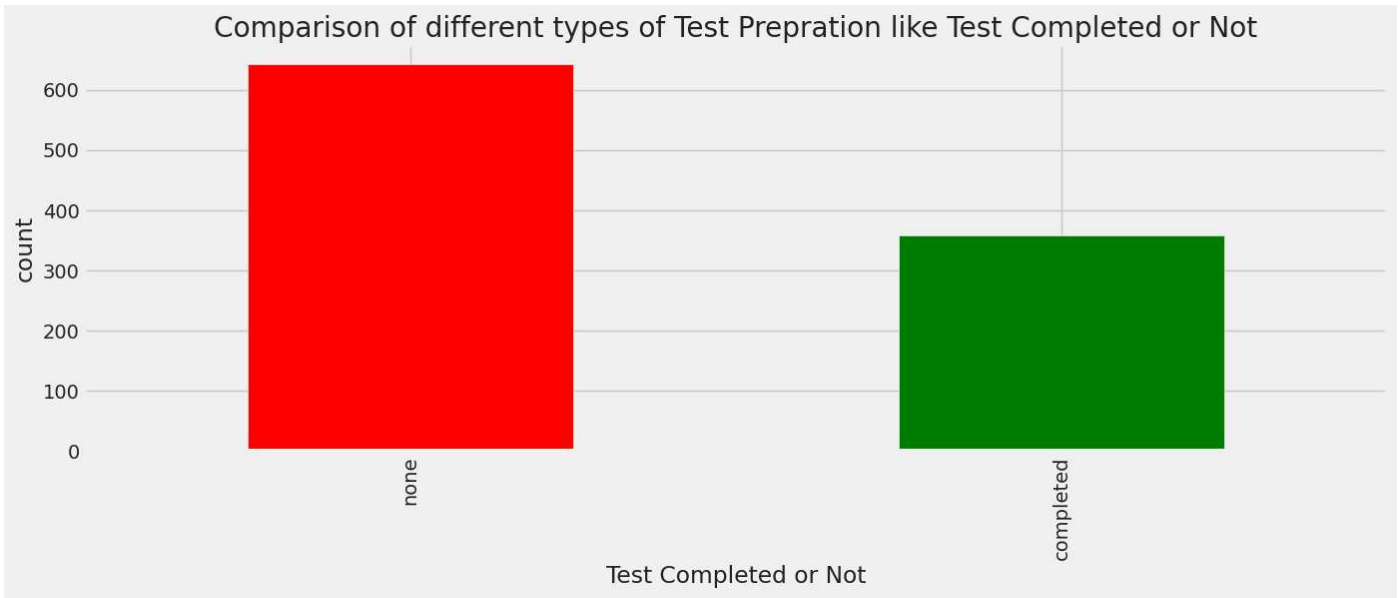

```
# visualizing the differnt parental education levels
plt.subplots(figsize=(15,5))
data['parental level of education'].value_counts(normalize = True)
data['parental level of education'].value_counts(dropna = False).plot.bar(color=['black', 'red', 'green', 'blue', 'cyan'])
plt.title('Comparison of Parental Education')
plt.xlabel('Degree')
plt.ylabel('count')
plt.show()
```



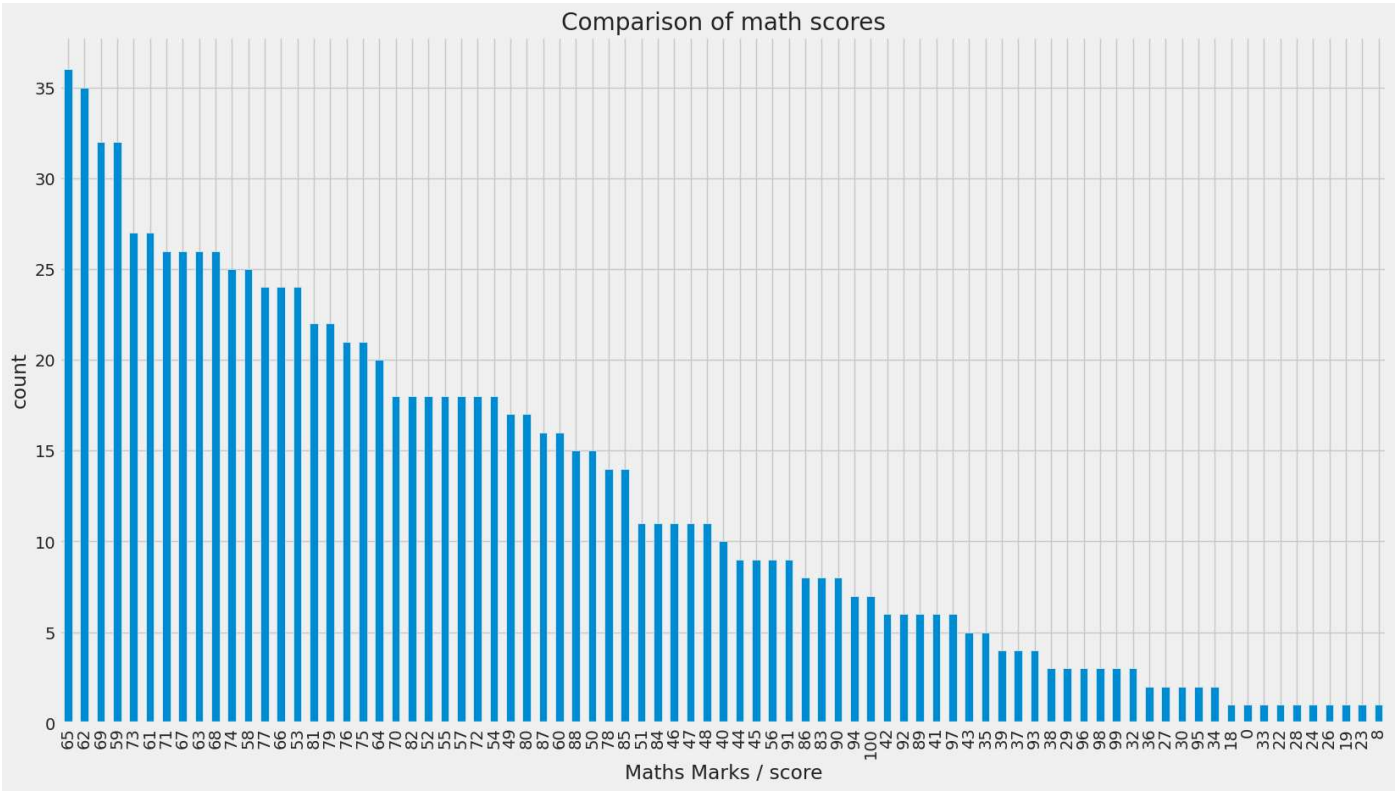

```
# visualizing different types of lunch
plt.subplots(figsize=(15,5))
data['lunch'].value_counts(normalize = True)
data['lunch'].value_counts(dropna = False).plot.bar(color = 'yellow')
plt.title('Comparison of different types of lunch')
plt.xlabel('types of lunch')
plt.ylabel('count')
plt.show()
```



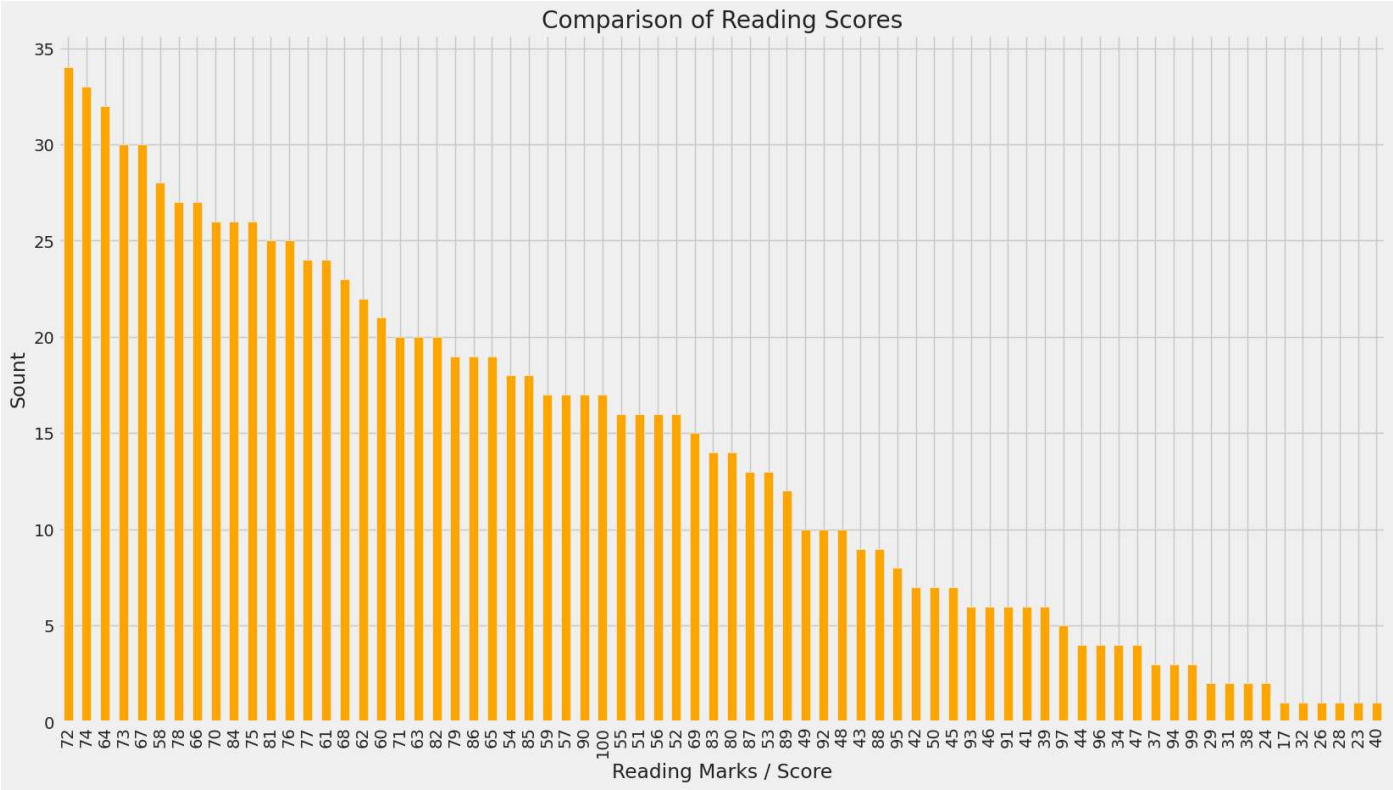

```
# visualizing different types of lunch
plt.subplots(figsize=(15,5))
data['test preparation course'].value_counts(normalize = True)
data['test preparation course'].value_counts(dropna = False).plot.bar(color = ['red', 'green'])
plt.title('Comparison of different types of Test Prepration like Test Completed or Not')
plt.xlabel('Test Completed or Not')
plt.ylabel('count')
plt.show()
```



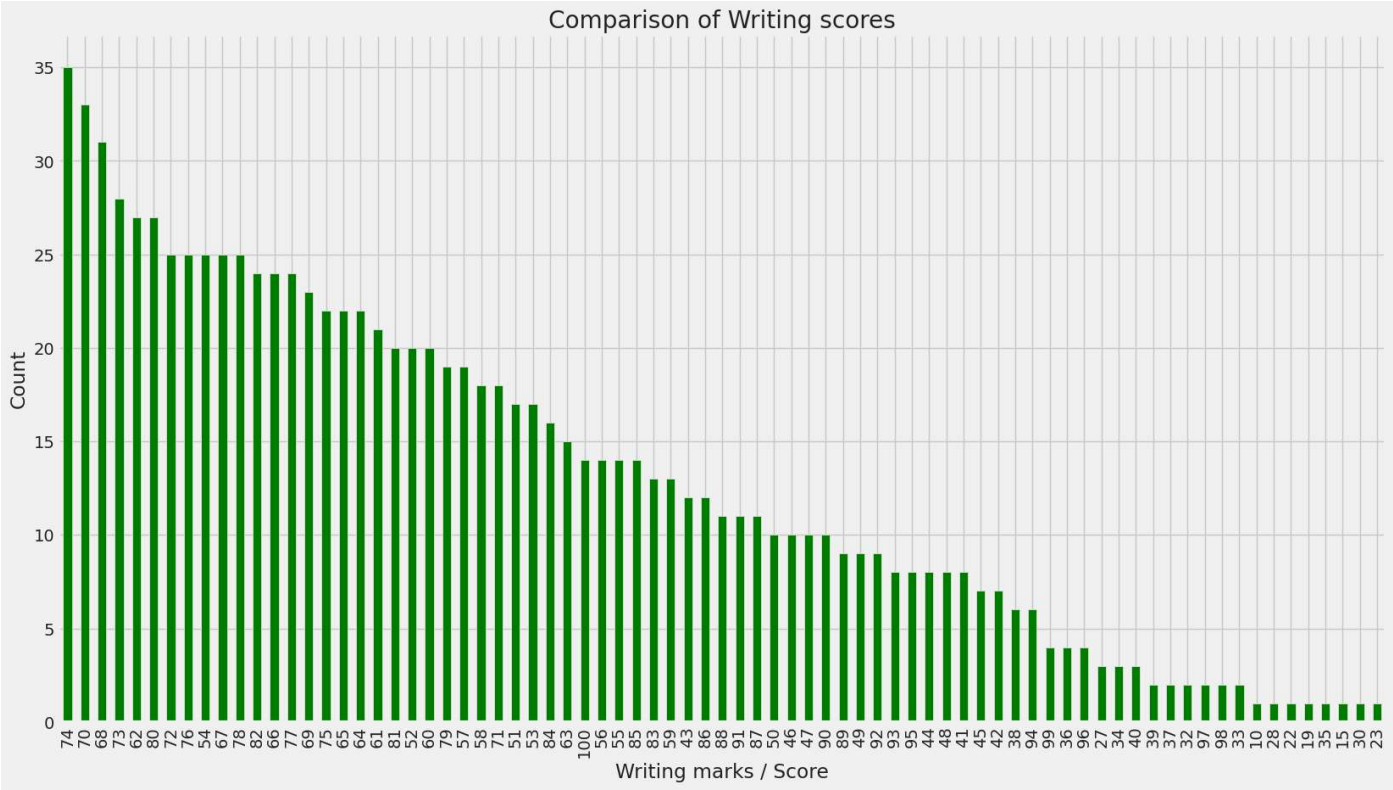
```
# visualizing maths score
plt.subplots(figsize=(15,5))
data['math score'].value_counts(normalize = True)
data['math score'].value_counts(dropna = False).plot.bar(figsize = (18, 10))
plt.title('Comparison of math scores')
plt.xlabel('Maths Marks / score')
plt.ylabel('count')
plt.show()
```




```
# visualizing reading score score
plt.subplots(figsize=(15,5))
data['reading score'].value_counts(normalize = True)
data['reading score'].value_counts(dropna = False).plot.bar(figsize = (18, 10), color = 'orange')
plt.title('Comparison of Reading Scores')
plt.xlabel('Reading Marks / Score')
plt.ylabel('Sount')
plt.show()
```



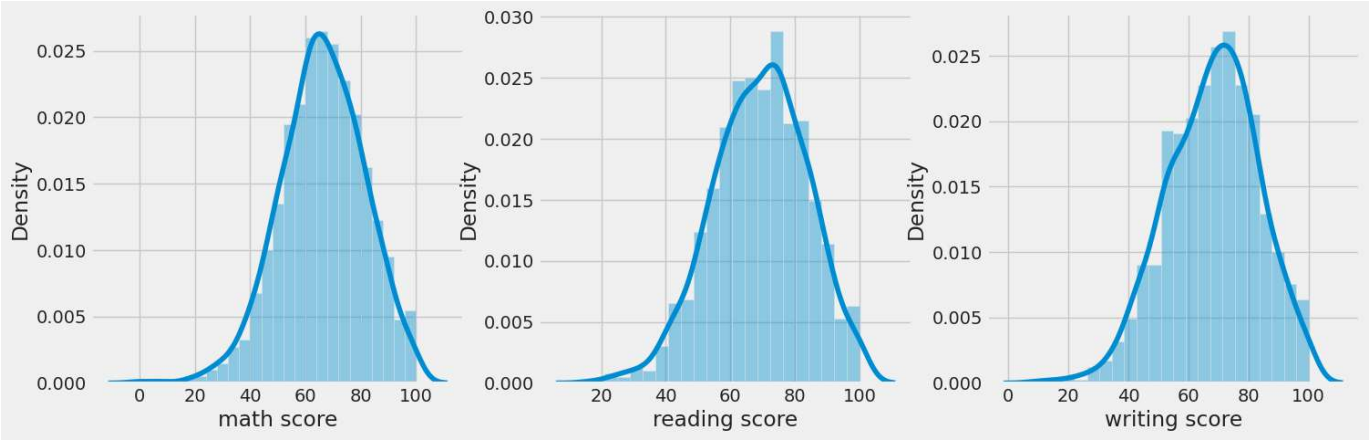

```
# visualizing writing score
plt.subplots(figsize=(15,5))
data['writing score'].value_counts(normalize = True)
data['writing score'].value_counts(dropna = False).plot.bar(figsize = (18, 10), color = 'green')
plt.title('Comparison of Writing scores')
plt.xlabel('Writing marks / Score')
plt.ylabel('Count')
plt.show()
```




## Detecting Outliers

```
plt.figure(figsize = (16,5))
#sns.distplot(data['writing score'])

plt.subplot(1, 3, 1)
sns.distplot(data['math score'])

plt.subplot(1, 3, 2)
sns.distplot(data['reading score'])

plt.subplot(1, 3, 3)
sns.distplot(data['writing score'])

plt.show()
```

```
/tmp/ipython-input-30-138431772.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


/tmp/ipython-input-30-138431772.py:8: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


/tmp/ipython-input-30-138431772.py:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```




- It is very much clear, that there is no skewness in the Target Columns,

**Lets check the Inference**

```
# lets take seed so that everytime the random values come out to be constant
np.random.seed(6)

# lets take 100 sample values from the dataset of 1000 values
sample_math_marks = np.random.choice(a= data['math score'], size=100)

# getting the sample mean
print ("Sample mean for Math Scores:", sample_math_marks.mean() )

# getting the population mean
print("Population mean for Math Scores:", data['math score'].mean())
```

```
# lets take 100 sample values from the dataset of 1000 values
sample_reading_marks = np.random.choice(a= data['reading score'], size=100)

# getting the sample mean
print ("\nSample mean for Reading Scores:", sample_reading_marks.mean() )

# getting the population mean
print("Population mean for Reading Scores:", data['reading score'].mean())

# lets take 100 sample values from the dataset of 1000 values
sample_writing_marks = np.random.choice(a= data['writing score'], size=100)

# getting the sample mean
print ("\nSample mean for Writing Scores:", sample_math_marks.mean() )

# getting the population mean
print("Population mean for Writing Scores:", data['writing score'].mean())
```

```
Sample mean for Math Scores: 63.12
Population mean for Math Scores: 66.089

Sample mean for Reading Scores: 68.5
Population mean for Reading Scores: 69.169

Sample mean for Writing Scores: 63.12
Population mean for Writing Scores: 68.054
```

**Let check the Confidence Interval for Math Score**

```
# lets import the scipy package
import scipy.stats as stats
import math

# lets seed the random values
np.random.seed(10)

# lets take a sample size
sample_size = 1000
sample = np.random.choice(a= data['math score'],
                          size = sample_size)
sample_mean = sample.mean()

# Get the z-critical value*
z_critical = stats.norm.ppf(q = 0.95)

 # Check the z-critical value
print("z-critical value: ",z_critical)

# Get the population standard deviation
pop_stdev = data['math score'].std()

# checking the margin of error
margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))

# defining our confidence interval
confidence_interval = (sample_mean - margin_of_error,
                       sample_mean + margin_of_error)

# lets print the results
print("Confidence interval:",end=" ")
print(confidence_interval)
print("True mean: {}".format(data['math score'].mean()))
```

```
z-critical value:  1.6448536269514722
Confidence interval: (np.float64(64.82729483328328), np.float64(66.40470516671672))
True mean: 66.089
```

**Let check the Confidence Interval for Reading Score**

```
# lets import the scipy package
import scipy.stats as stats
import math

# lets seed the random values
np.random.seed(10)

# lets take a sample size
```

```
sample_size = 1000
sample = np.random.choice(a= data['reading score'],
                          size = sample_size)
sample_mean = sample.mean()

# Get the z-critical value*
z_critical = stats.norm.ppf(q = 0.95)

 # Check the z-critical value
print("z-critical value: ",z_critical)

# Get the population standard deviation
pop_stdev = data['reading score'].std()

# checking the margin of error
margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))

# defining our confidence interval
confidence_interval = (sample_mean - margin_of_error,
                       sample_mean + margin_of_error)

# lets print the results
print("Confidence interval:",end=" ")
print(confidence_interval)
print("True mean: {}".format(data['reading score'].mean()))
```

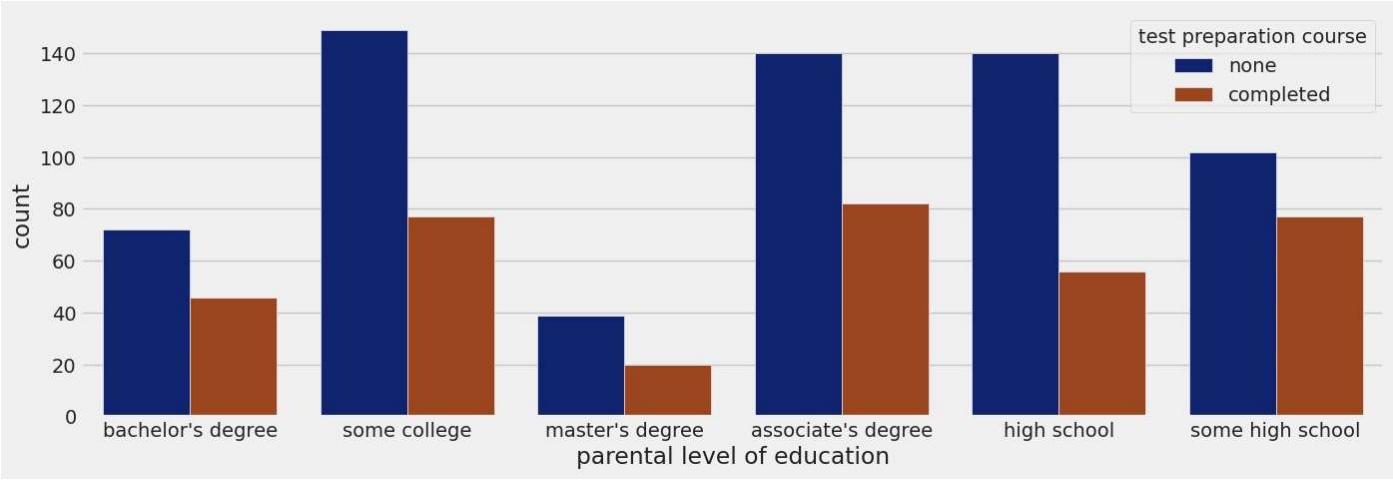
```
z-critical value:  1.6448536269514722
Confidence interval: (np.float64(67.75757337011645), np.float64(69.27642662988355))
True mean: 69.169
```

**Let check the Confidence Interval for Writing Score**

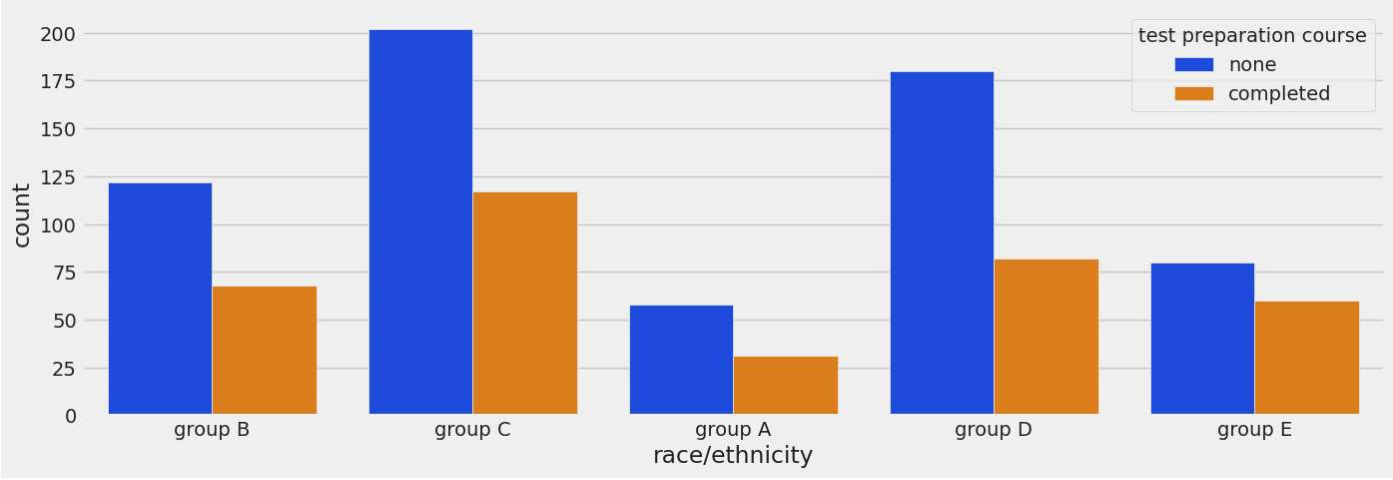
```
# lets take a sample size
sample_size = 1000
sample = np.random.choice(a= data['writing score'],
                          size = sample_size)
sample_mean = sample.mean()

# Get the z-critical value*
z_critical = stats.norm.ppf(q = 0.95)

 # Check the z-critical value
print("z-critical value: ",z_critical)

# Get the population standard deviation
pop_stdev = data['writing score'].std()

# checking the margin of error
margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))

# defining our confidence interval
confidence_interval = (sample_mean - margin_of_error,
                       sample_mean + margin_of_error)

# lets print the results
print("Confidence interval:",end=" ")
print(confidence_interval)
print("True mean: {}".format(data['writing score'].mean()))
```

```
z-critical value:  1.6448536269514722
Confidence interval: (np.float64(67.59660035030862), np.float64(69.17739964969138))
True mean: 68.054
```

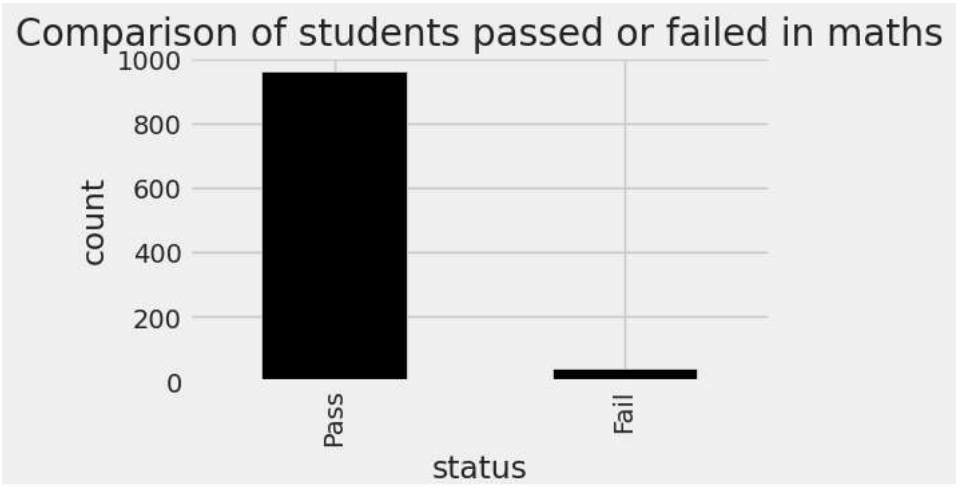
```
# comparison of parental degree and test course
plt.subplots(figsize=(15,5))
sns.countplot(x = 'parental level of education', data = data, hue = 'test preparation course', palette = 'dark')
plt.show()
```



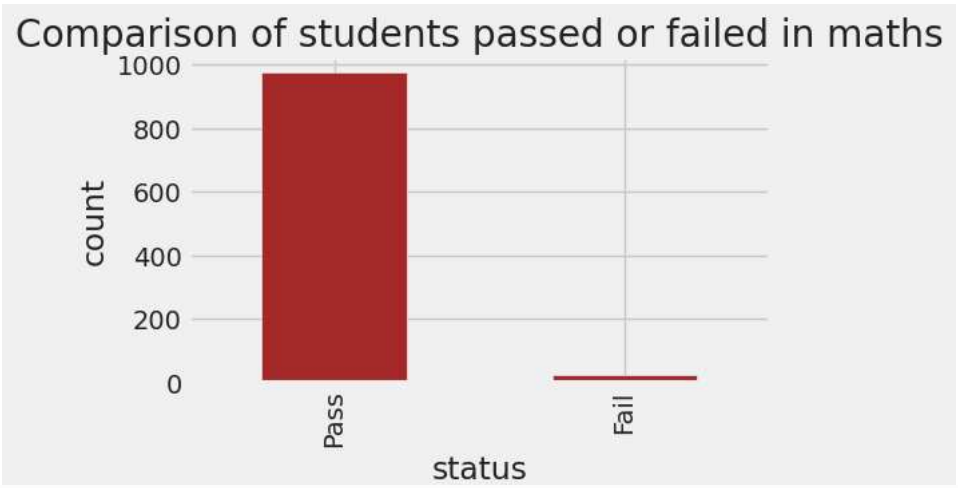

```
# comparison of race/ethnicity and test preparation course
plt.subplots(figsize=(15,5))
sns.countplot(x = 'race/ethnicity', data = data,  hue = 'test preparation course', palette = 'bright')
plt.show()
```



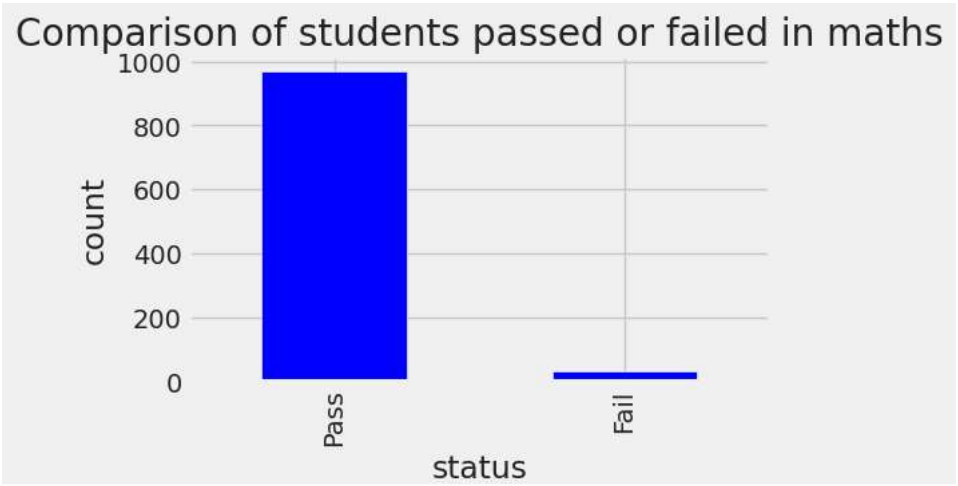

```
# feature engineering on the data to visualize and solve the dataset more accurately

# setting a passing mark for the students to pass on the three subjects individually
plt.subplots(figsize=(15,8))
passmarks = 40

# creating a new column pass_math, this column will tell us whether the students are pass or fail
data['pass_math'] = np.where(data['math score']< passmarks, 'Fail', 'Pass')
data['pass_math'].value_counts(dropna = False).plot.bar(color = 'black', figsize = (5, 3))

plt.title('Comparison of students passed or failed in maths')
plt.xlabel('status')
plt.ylabel('count')
plt.show()
```



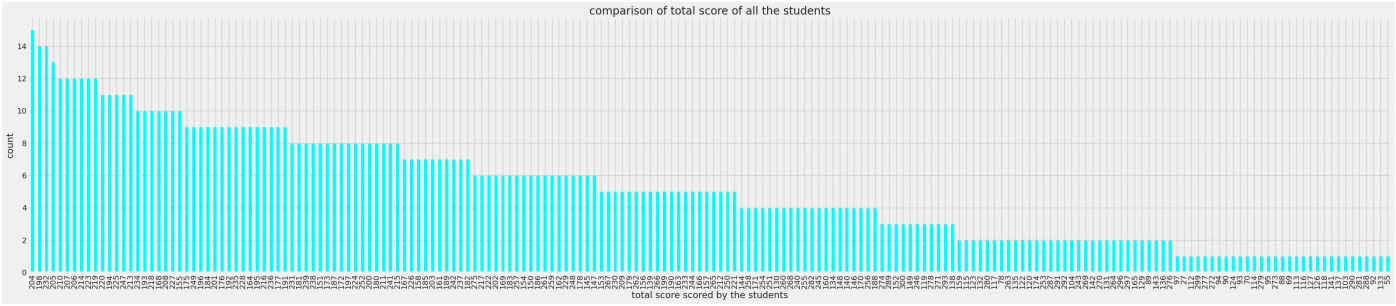

```
# creating a new column pass_math, this column will tell us whether the students are pass or fail
data['pass_reading'] = np.where(data['reading score']< passmarks, 'Fail', 'Pass')
data['pass_reading'].value_counts(dropna = False).plot.bar(color = 'brown', figsize = (5, 3))

plt.title('Comparison of students passed or failed in maths')
plt.xlabel('status')
plt.ylabel('count')
plt.show()
```




```
# creating a new column pass_math, this column will tell us whether the students are pass or fail
data['pass_writing'] = np.where(data['writing score']< passmarks, 'Fail', 'Pass')
data['pass_writing'].value_counts(dropna = False).plot.bar(color = 'blue', figsize = (5, 3))

plt.title('Comparison of students passed or failed in maths')
plt.xlabel('status')
plt.ylabel('count')
plt.show()
```

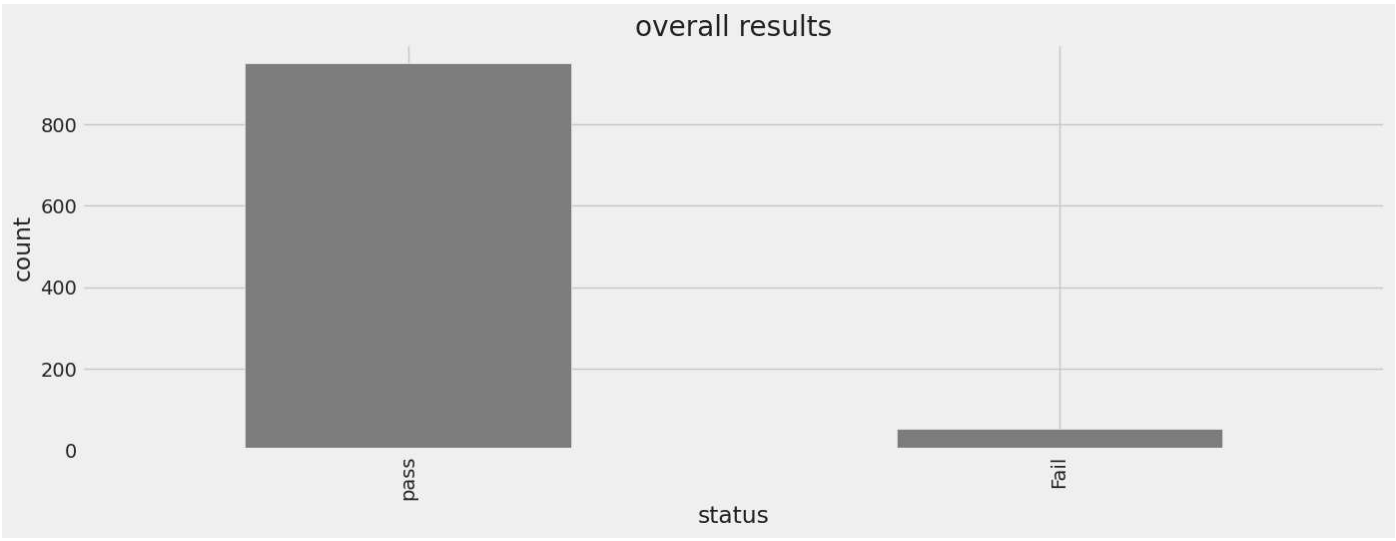



```
# computing the total score for each student
plt.subplots(figsize=(15,8))
data['total_score'] = data['math score'] + data['reading score'] + data['writing score']

data['total_score'].value_counts(normalize = True)
data['total_score'].value_counts(dropna = True).plot.bar(color = 'cyan', figsize = (40, 8))

plt.title('comparison of total score of all the students')
plt.xlabel('total score scored by the students')
plt.ylabel('count')
plt.show()
```

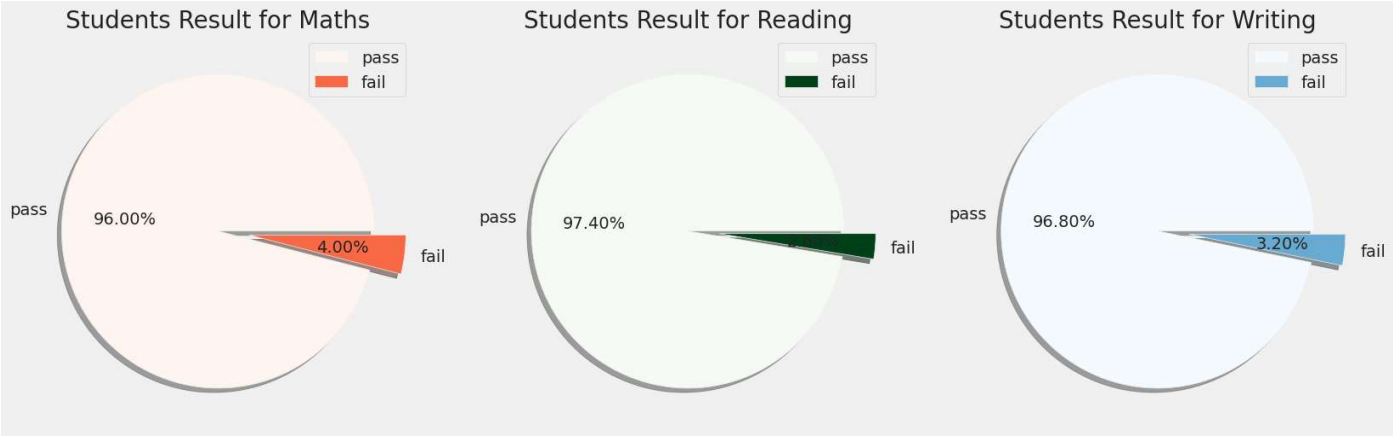



```
# computing percentage for each of the students
# importing math library to use ceil
from math import *

data['percentage'] = data['total_score']/3

for i in range(0, 1000):
  data['percentage'][i] = ceil(data['percentage'][i])

data['percentage'].value_counts(normalize = True)
data['percentage'].value_counts(dropna = False).plot.bar(figsize = (16, 8), color = 'red')

plt.title('Comparison of percentage scored by all the students')
plt.xlabel('percentage score')
plt.ylabel('count')
plt.show()


# checking which student is fail overall

data['status'] = data.apply(lambda x : 'Fail' if x['pass_math'] == 'Fail' or
                            x['pass_reading'] == 'Fail' or x['pass_writing'] == 'Fail'
                            else 'pass', axis = 1)

data['status'].value_counts(dropna = False).plot.bar(color = 'gray', figsize = (15, 5))
plt.title('overall results')
plt.xlabel('status')
```

```
plt.ylabel('count')
plt.show()
```




```
# setting a passing mark for the students to pass on the three subjects individually
passmarks = 40
plt.rcParams['figure.figsize'] = (18, 12)

# creating a new column pass_math, this column will tell us whether the students are pass or fail
data['pass_math'] = np.where(data['math score']< passmarks, 'Fail', 'Pass')
data['pass_reading'] = np.where(data['reading score']< passmarks, 'Fail', 'Pass')
data['pass_writing'] = np.where(data['writing score']< passmarks, 'Fail', 'Pass')

# pie chart to represent the ratio of pass and fail status between the students

size = data['pass_math'].value_counts()
colors = plt.cm.Reds(np.linspace(0, 1, 3))
labels = "pass", "fail"
explode = [0, 0.2]

plt.subplot(1, 3, 1)
plt.pie(size, colors = colors, labels = labels, autopct = '%.2f%%', explode = explode, shadow = True)
plt.title('Students Result for Maths', fontsize = 20)
plt.legend()

size = data['pass_reading'].value_counts()
colors = plt.cm.Greens(np.linspace(0, 1, 2))
labels = "pass", "fail"
explode = [0, 0.2]

plt.subplot(1, 3, 2)
plt.pie(size, colors = colors, labels = labels, autopct = '%.2f%%', explode = explode, shadow = True)
plt.title('Students Result for Reading', fontsize = 20)
plt.legend()

size = data['pass_writing'].value_counts()
colors = plt.cm.Blues(np.linspace(0, 1, 3))
labels = "pass", "fail"
explode = [0, 0.2]

plt.subplot(1, 3, 3)
plt.pie(size, colors = colors, labels = labels, autopct = '%.2f%%', explode = explode, shadow = True)
plt.title('Students Result for Writing', fontsize = 20)
plt.legend()

plt.show()
```




```
# Assigning grades to the grades according to the following criteria :
# 0  - 40 marks : grade E
# 41 - 60 marks : grade D
# 60 - 70 marks : grade C
# 70 - 80 marks : grade B
# 80 - 90 marks : grade A
# 90 - 100 marks : grade O

def getgrade(percentage, status):
  if status == 'Fail':
    return 'E'
  if(percentage >= 90):
    return 'O'
  if(percentage >= 80):
    return 'A'
  if(percentage >= 70):
    return 'B'
  if(percentage >= 60):
    return 'C'
  if(percentage >= 40):
    return 'D'
  else :
    return 'E'

data['grades'] = data.apply(lambda x: getgrade(x['percentage'], x['status']), axis = 1 )

data['grades'].value_counts()
```

Show hidden output

```
# plotting a pie chart for the distribution of various grades amongst the students


plt.subplots(figsize=(15,8))
labels = ['Grade 0', 'Grade A', 'Grade B', 'Grade C', 'Grade D', 'Grade E']
sizes = [58, 156, 260, 252, 223, 51]
colors = ['yellow', 'gold', 'lightskyblue', 'lightcoral', 'pink', 'cyan']
explode = (0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001)

patches, texts = plt.pie(sizes, colors=colors, shadow=True, startangle=90)
plt.legend(patches, labels)
plt.axis('equal')
plt.tight_layout()
plt.show()
```