Required libraries are Imported:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Data is imported using the following:

```
from google.colab import files
import io
data = files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving StudentsPerformance.csv to StudentsPerformance.csv
```

```
df=pd.read_csv(io.StringIO(data['StudentsPerformance.csv'].decode('utf-8')))
```

```
df.head
```

```
<bound method NDFrame.head of      gender race/ethnicity  ... reading score writing s
0    female        group B  ...             72            74
1    female        group C  ...             90            88
2    female        group B  ...             95            93
3      male        group A  ...             57            44
4      male        group C  ...             78            75
..      ...            ...  ...            ...           ...
995  female        group E  ...             99            95
996    male        group C  ...             55            55
997  female        group C  ...             71            65
998  female        group D  ...             78            77
999  female        group D  ...             86            86

[1000 rows x 8 columns]>
```

```
print(df.shape)
```

```
(1000, 8)
```

```
df.head()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writi sco
0	female	group B	bachelor's degree	standard	none	72	72	

```
df.tail()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	wri s
995	female	group E	master's degree	standard	completed	88	99	
996	male	group C	high school	free/reduced	none	62	55	
997	female	group C	high school	free/reduced	completed	59	71	

```
df.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
```

```
df.describe()
```

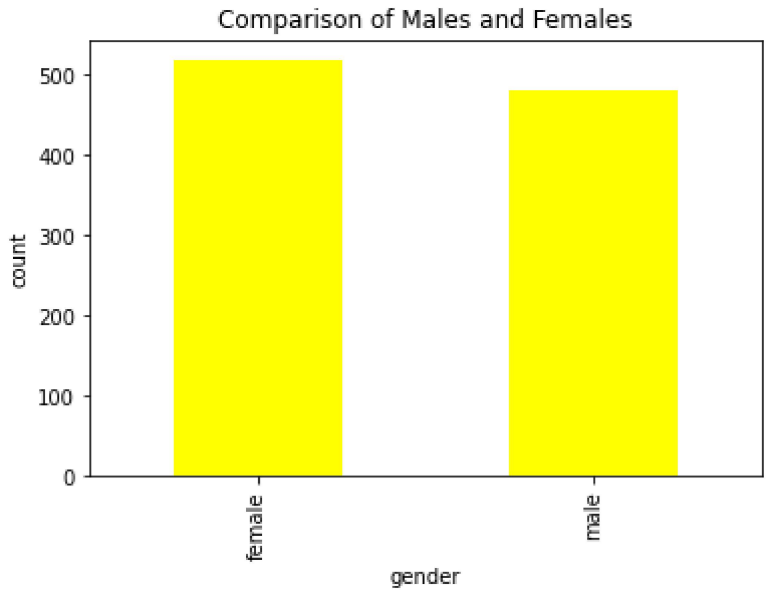
Missing values are checked:

```
df.isnull().sum()
```

```
gender                         0
race/ethnicity                 0
parental level of education    0
lunch                          0
test preparation course        0
math score                     0
reading score                  0
writing score                  0
dtype: int64
```

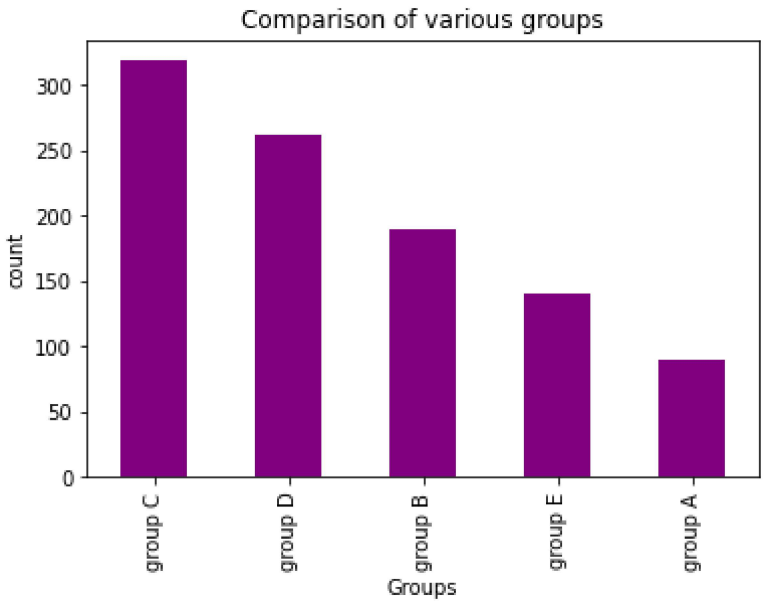
Data Visualization

```
# visualising the number of male and female in the dataset

df['gender'].value_counts(normalize = True)
df['gender'].value_counts(dropna = False).plot.bar(color = 'yellow')
plt.title('Comparison of Males and Females')
plt.xlabel('gender')
plt.ylabel('count')
plt.show()
```




```
# visualizing the different groups in the dataset

df['race/ethnicity'].value_counts(normalize = True)
df['race/ethnicity'].value_counts(dropna = False).plot.bar(color = 'purple')
```

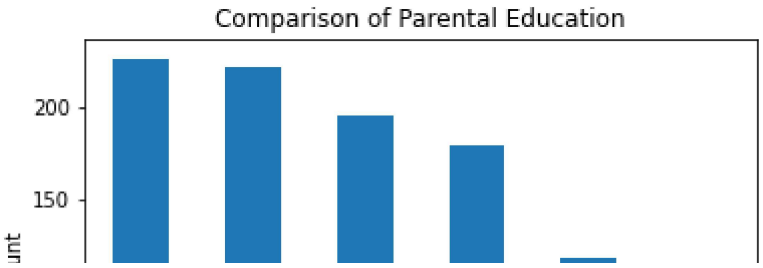
```
plt.title('Comparison of various groups')
plt.xlabel('Groups')
plt.ylabel('count')
plt.show()
```



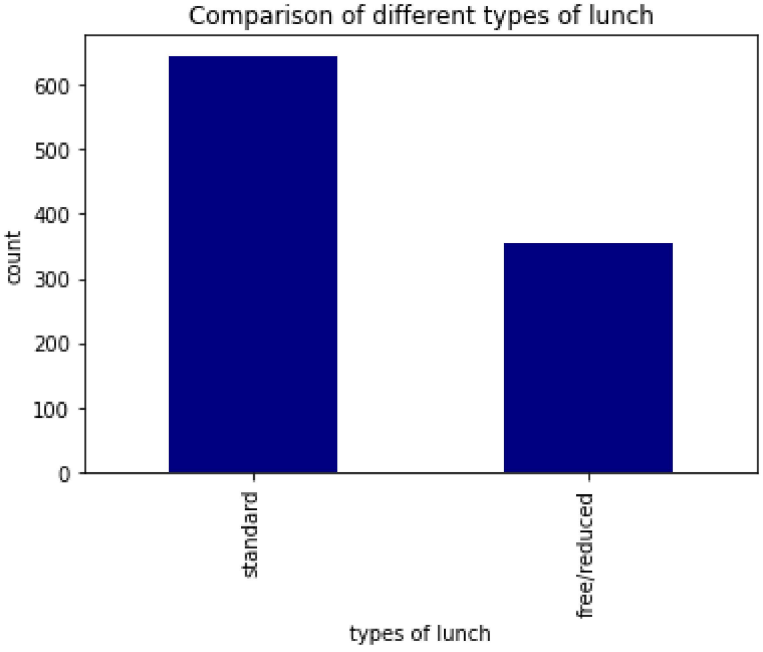

```
df['race/ethnicity'].value_counts()
```

```
group C    319
group D    262
group B    190
group E    140
group A     89
Name: race/ethnicity, dtype: int64
```

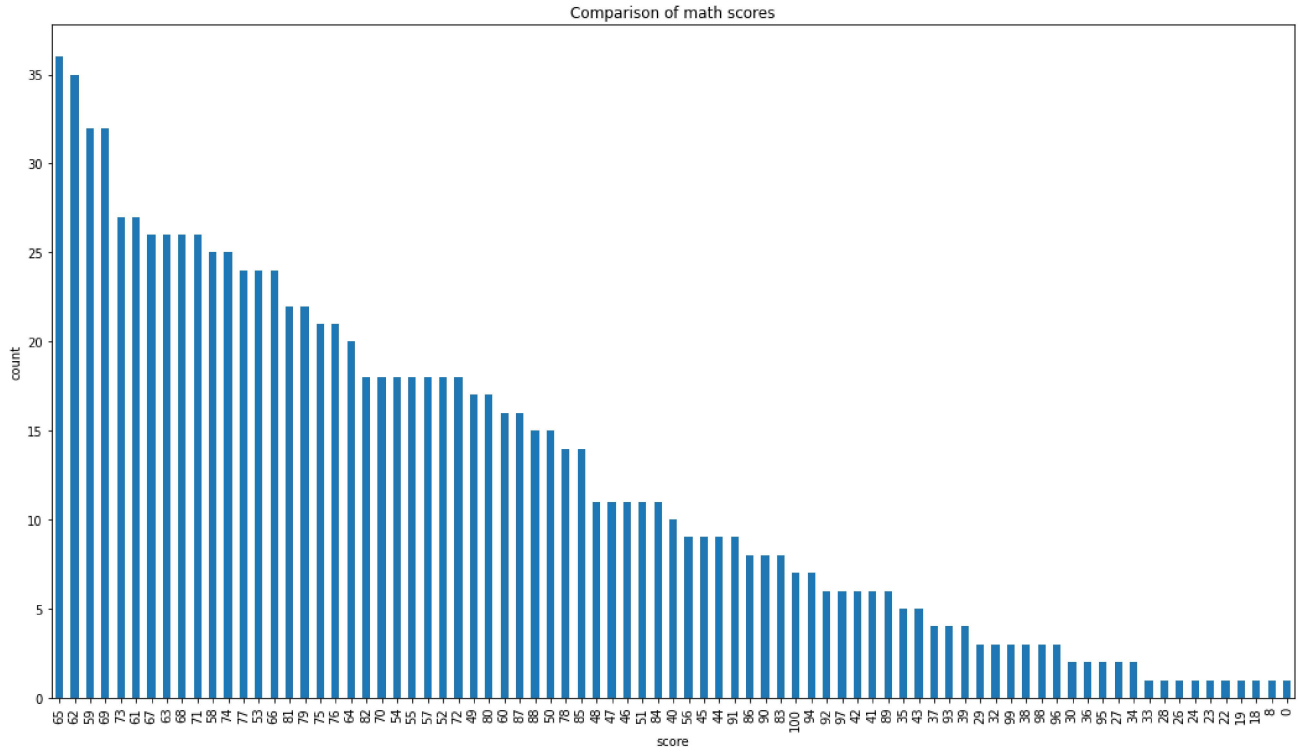
```
# visualizing the differnt parental education levels

df['parental level of education'].value_counts(normalize = True)
df['parental level of education'].value_counts(dropna = False).plot.bar()
plt.title('Comparison of Parental Education')
plt.xlabel('Degree')
plt.ylabel('count')
plt.show()
```



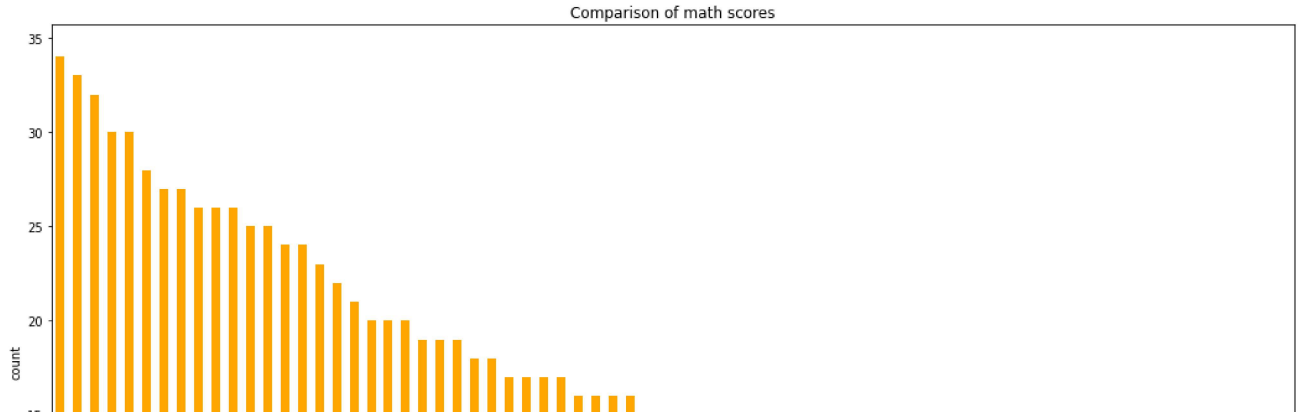

```
# visualizing different types of lunch

df['lunch'].value_counts(normalize = True)
df['lunch'].value_counts(dropna = False).plot.bar(color = 'navy')
plt.title('Comparison of different types of lunch')
plt.xlabel('types of lunch')
plt.ylabel('count')
plt.show()
```



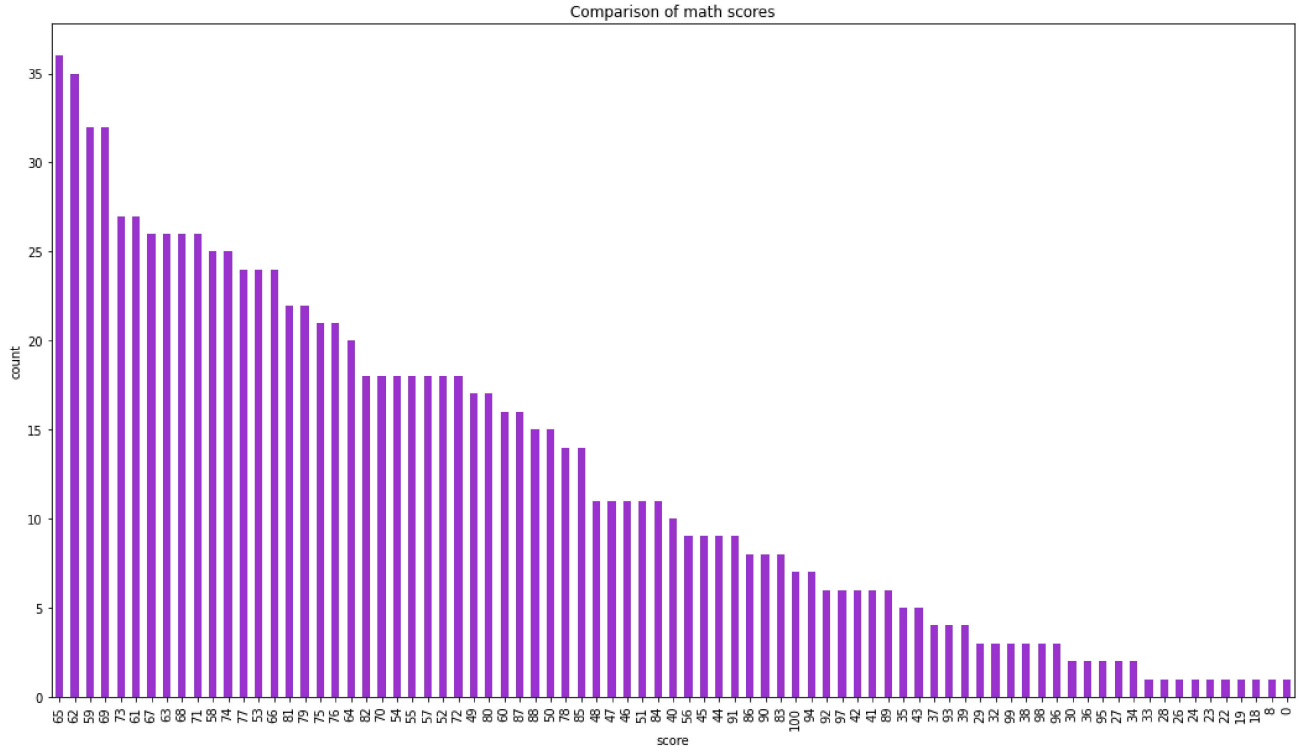

```
# visualizing maths score

df['math score'].value_counts(normalize = True)
df['math score'].value_counts(dropna = False).plot.bar(figsize = (18, 10))
plt.title('Comparison of math scores')
plt.xlabel('score')
plt.ylabel('count')
plt.show()
```




```
# visualizing reading score score

df['reading score'].value_counts(normalize = True)
df['reading score'].value_counts(dropna = False).plot.bar(figsize = (18, 10), color = 'ora
plt.title('Comparison of math scores')
plt.xlabel('score')
plt.ylabel('count')
plt.show()
```

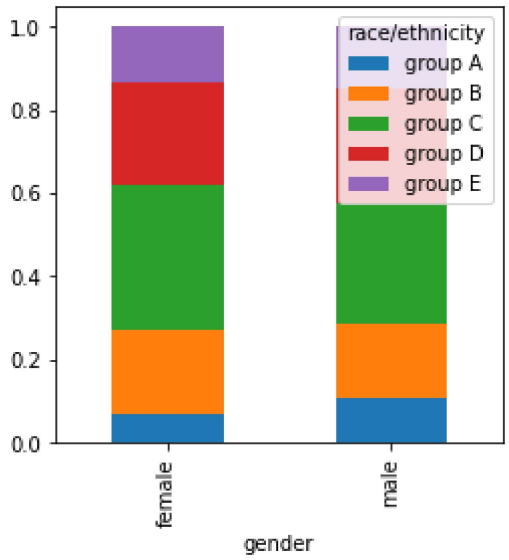



```
# visualizing writing score

df['math score'].value_counts(normalize = True)
df['math score'].value_counts(dropna = False).plot.bar(figsize = (18, 10), color = 'darkor
plt.title('Comparison of math scores')
plt.xlabel('score')
plt.ylabel('count')
plt.show()
```



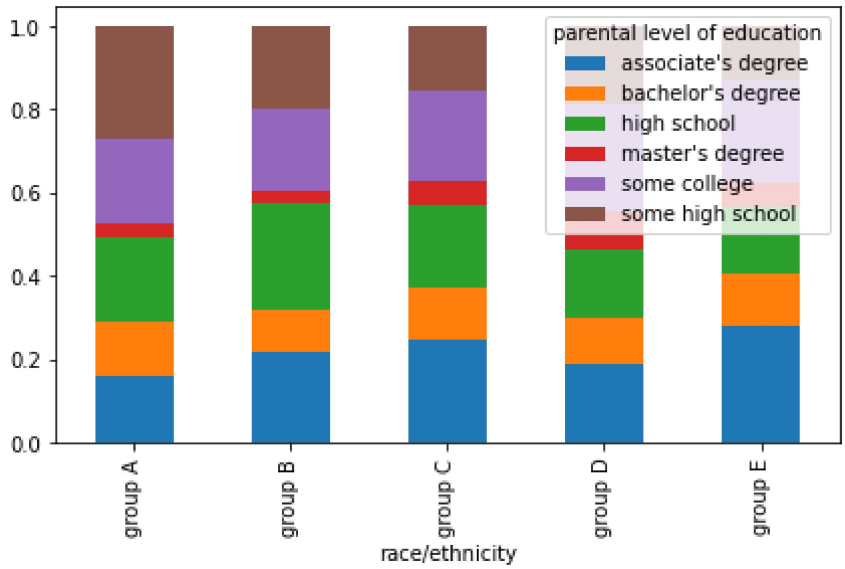
```
# gender vs race/etnicity

x = pd.crosstab(df['gender'], df['race/ethnicity'])
x.div(x.sum(1).astype(float), axis = 0).plot(kind = 'bar', stacked = True, figsize = (4, 4
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc47d8fa450>
```

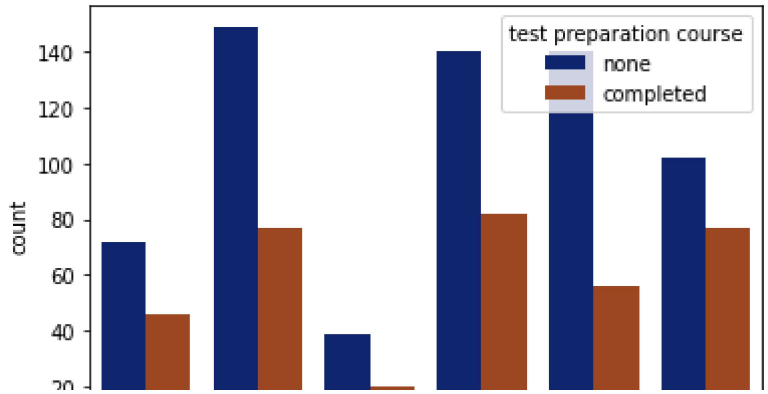



```
# comparison of race/ethnicity and parental level of education

x = pd.crosstab(df['race/ethnicity'], df['parental level of education'])
x.div(x.sum(1).astype(float), axis = 0).plot(kind = 'bar', stacked = 'True', figsize = (7,
```

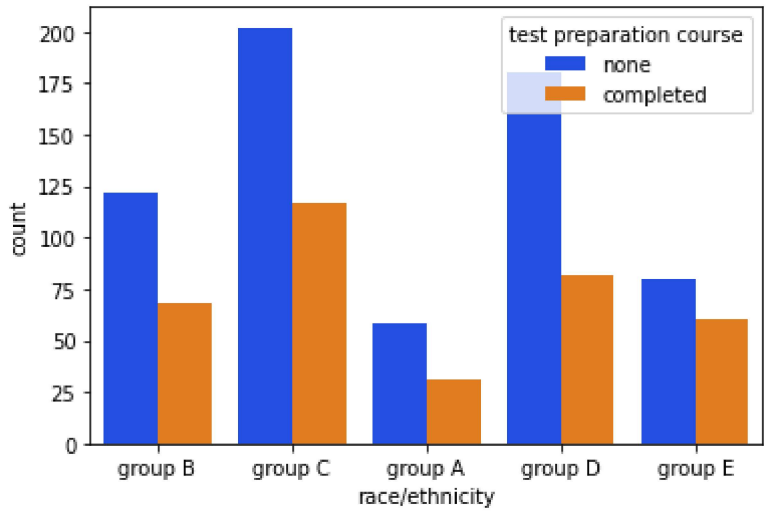
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc479d68710>
```



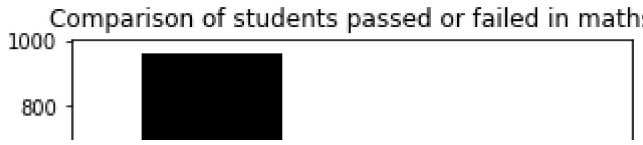

```
# comparison of parental degree and test course

sns.countplot(x = 'parental level of education', data = df, hue = 'test preparation course
plt.show()
```




```
# comparison of race/ethnicity and test preparation course

sns.countplot(x = 'race/ethnicity', data = df,  hue = 'test preparation course', palette =
plt.show()
```



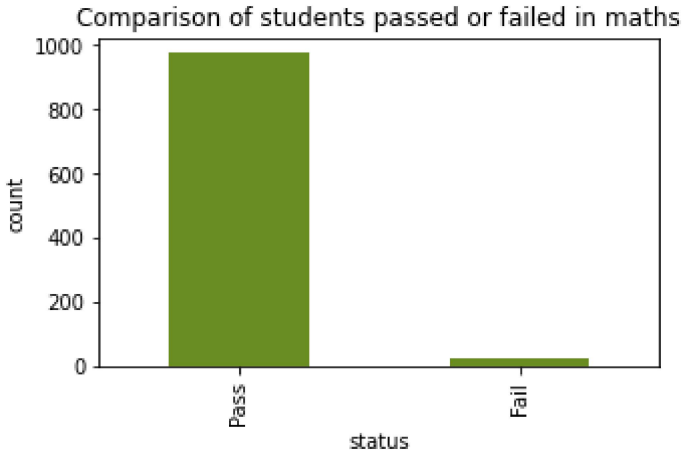

```
# feature engineering on the data to visualize and solve the dataset more accurately

# setting a passing mark for the students to pass on the three subjects individually
passmarks = 40

# creating a new column pass_math, this column will tell us whether the students are pass
df['pass_math'] = np.where(df['math score']< passmarks, 'Fail', 'Pass')
df['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()
```




```
df['pass_math'].value_counts()
```

```
Pass    960
Fail     40
Name: pass_math, dtype: int64
```

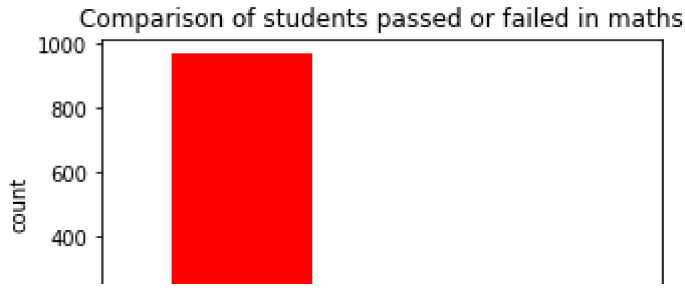

```
# creating a new column pass_math, this column will tell us whether the students are pass
df['pass_reading'] = np.where(df['reading score']< passmarks, 'Fail', 'Pass')
df['pass_reading'].value_counts(dropna = False).plot.bar(color = 'olivedrab', figsize = (5

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



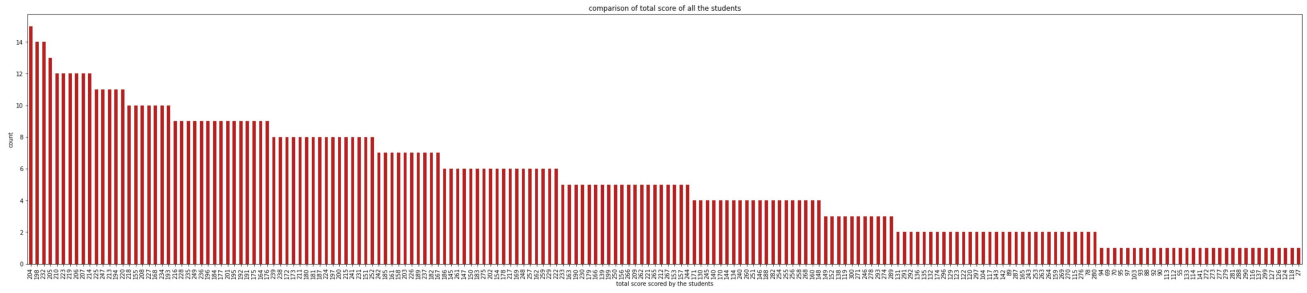

```
df['pass_reading'].value_counts(dropna = False)
```

```
Pass    974
Fail     26
Name: pass_reading, dtype: int64
```

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

df['total_score'] = df['math score'] + df['reading score'] + df['writing score']

df['total_score'].value_counts(normalize = True)
df['total_score'].value_counts(dropna = True).plot.bar(color = 'firebrick', figsize = (40,

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

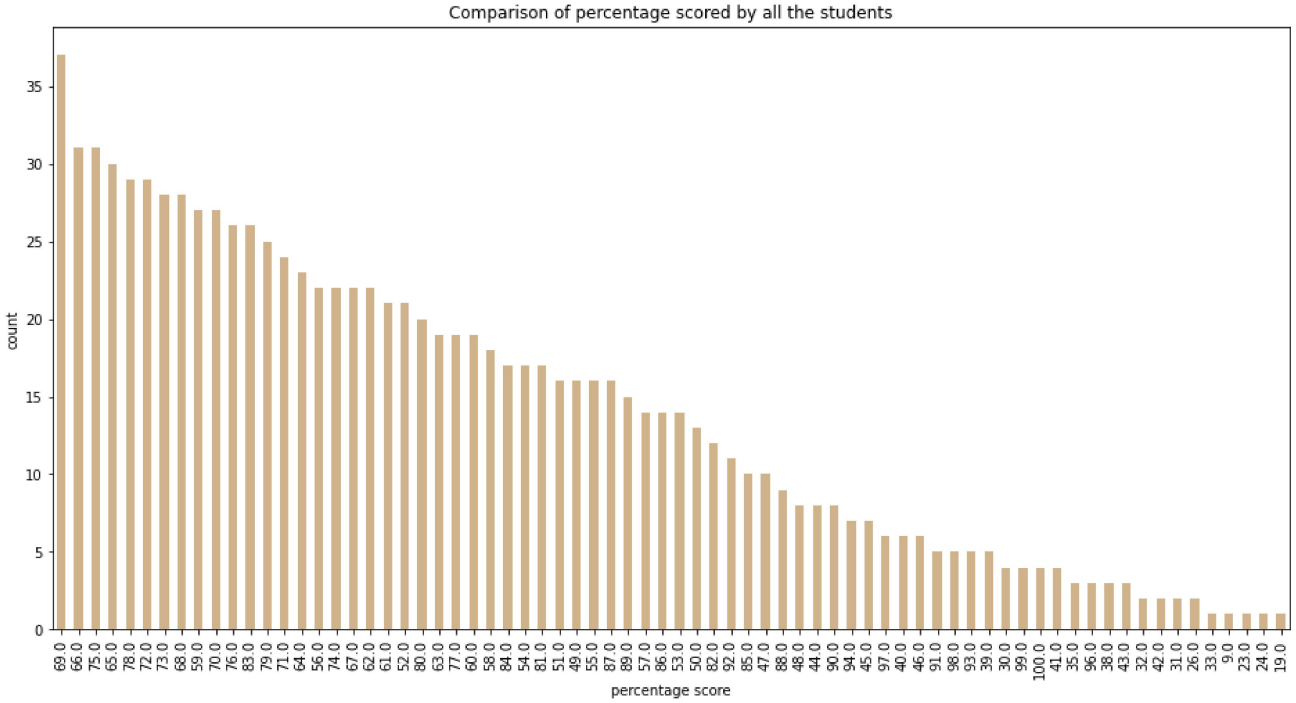


Double-click (or enter) to edit

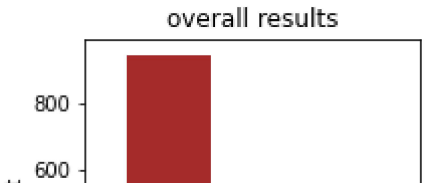
```
# computing percentage for each of the students
# importing math library to use ceil
from math import *

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

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

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

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

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
```




```
# checking which student is fail overall

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

df['status'].value_counts(dropna = False).plot.bar(color = 'brown', figsize = (3, 3))
plt.title('overall results')
plt.xlabel('status')
plt.ylabel('count')
plt.show()
```



Assigning grades to the grades according to the following criteria :

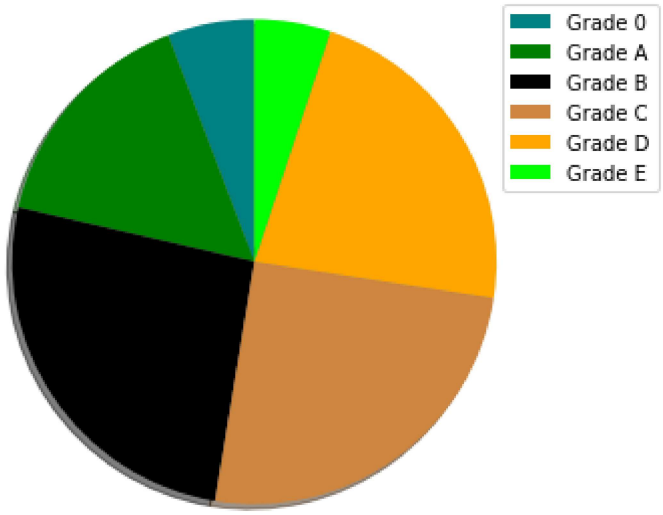
- 0 - 40 marks : grade E
- 41 - 60 marks : grade D
- 60 - 70 marks : grade C
- 70 - 80 marks : grade B
- 80 - 90 marks : grade A
- 90 - 100 marks : grade O

```
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'

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

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

```
B    260
C    252
D    223
A    156
O     58
E     51
Name: grades, dtype: int64
```

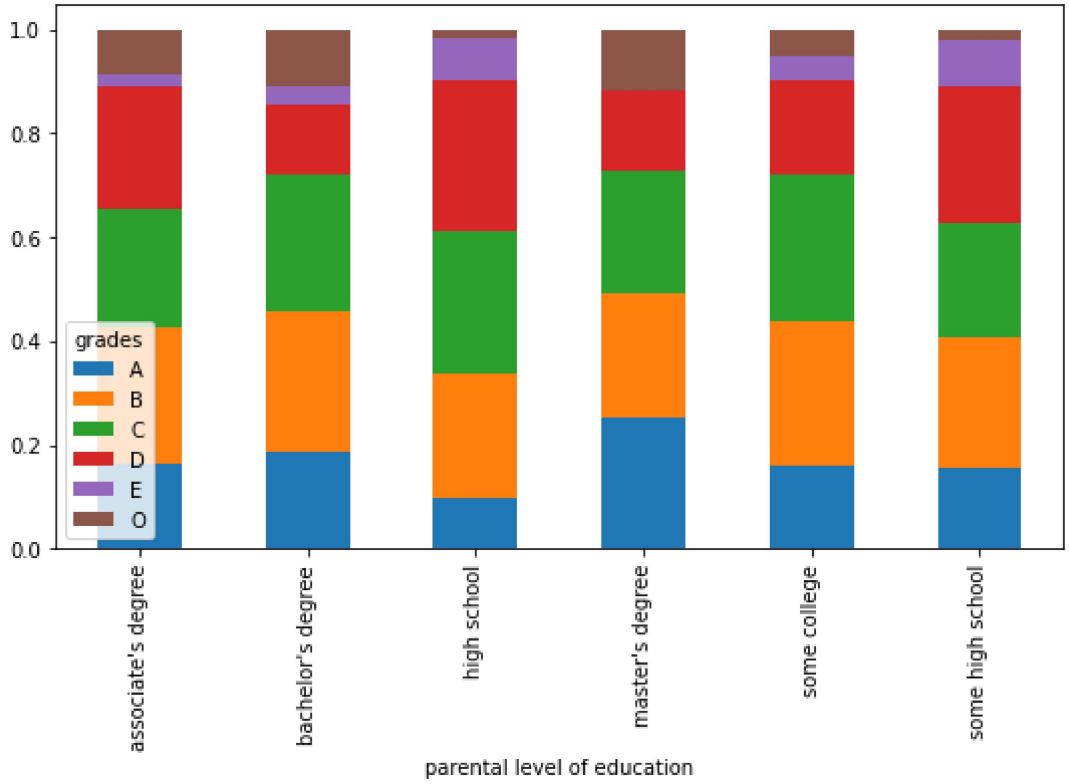
```
# plotting a pie chart for the distribution of various grades amongst the students

labels = ['Grade 0', 'Grade A', 'Grade B', 'Grade C', 'Grade D', 'Grade E']
sizes = [58, 156, 260, 252, 223, 51]
colors = ['teal', 'green', 'black', 'peru', 'orange', 'lime']
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()
```



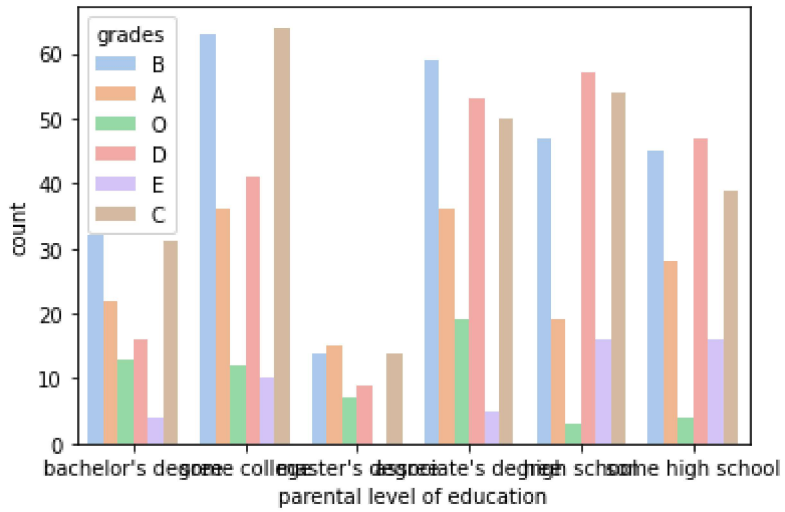

```
# comparison parent's degree and their corresponding grades

x = pd.crosstab(df['parental level of education'], df['grades'])
x.div(x.sum(1).astype(float), axis = 0).plot(kind = 'bar', stacked = True, figsize = (9, 5
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc479edb250>
```

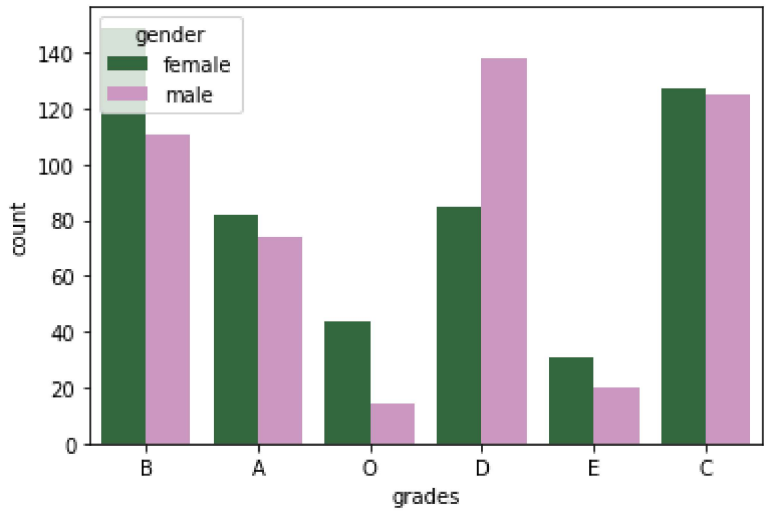



```
# for better visualization we will plot it again using seaborn
```

```
sns.countplot(x = df['parental level of education'], data = df, hue = df['grades'], palett
plt.show()
```




```
# comparing the distribution of grades among males and females

sns.countplot(x = df['grades'], data = df, hue = df['gender'], palette = 'cubehelix')
#sns.palplot(sns.dark_palette('purple'))
plt.show()
```




```
df.head()
```

gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writi sco
		bachelor's					

```
df.describe()
```

	math score	reading score	writing score	total_score	percentage
count	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000
mean	66.08900	69.169000	68.054000	203.312000	68.105000
std	15.16308	14.600192	15.195657	42.771978	14.258095
min	0.00000	17.000000	10.000000	27.000000	9.000000
25%	57.00000	59.000000	57.750000	175.000000	59.000000
50%	66.00000	70.000000	69.000000	205.000000	69.000000
75%	77.00000	79.000000	79.000000	233.000000	78.000000
max	100.00000	100.000000	100.000000	300.000000	100.000000

Data_Preprocessing:

```
from sklearn.preprocessing import LabelEncoder

# creating an encoder
le = LabelEncoder()

# label encoding for test preparation course
df['test preparation course'] = le.fit_transform(df['test preparation course'])
df['test preparation course'].value_counts()
```

```
1    642
0    358
Name: test preparation course, dtype: int64
```

```
# label encoding for lunch

df['lunch'] = le.fit_transform(df['lunch'])
df['lunch'].value_counts()
```

```
1    645
0    355
Name: lunch, dtype: int64
```

```
# label encoding for race/ethnicity
# we have to map values to each of the categories

df['race/ethnicity'] = df['race/ethnicity'].replace('group A', 1)
```

```
df['race/ethnicity'] = df['race/ethnicity'].replace('group B', 2)
df['race/ethnicity'] = df['race/ethnicity'].replace('group C', 3)
df['race/ethnicity'] = df['race/ethnicity'].replace('group D', 4)
df['race/ethnicity'] = df['race/ethnicity'].replace('group E', 5)

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

```
3    319
4    262
2    190
5    140
1     89
Name: race/ethnicity, dtype: int64
```

```
# label encoding for parental level of education

df['parental level of education'] = le.fit_transform(df['parental level of education'])
df['parental level of education'].value_counts()
```

```
4    226
0    222
2    196
5    179
1    118
3     59
Name: parental level of education, dtype: int64
```

```
# label encoding for gender

df['gender'] = le.fit_transform(df['gender'])
df['gender'].value_counts()
```

```
0    518
1    482
Name: gender, dtype: int64
```

```
# label encoding for pass_math

df['pass_math'] = le.fit_transform(df['pass_math'])
df['pass_math'].value_counts()
```

```
1    960
0     40
Name: pass_math, dtype: int64
```

```
# label encoding for pass_reading

df['pass_reading'] = le.fit_transform(df['pass_reading'])
df['pass_reading'].value_counts()
```

```
1    974
0     26
Name: pass_reading, dtype: int64
```

```
# label encoding for pass_writing

df['pass_writing'] = le.fit_transform(df['pass_writing'])
df['pass_writing'].value_counts()
```

```
1    968
0     32
Name: pass_writing, dtype: int64
```

```
# label encoding for status

df['status'] = le.fit_transform(df['status'])
df['status'].value_counts()
```

```
1    949
0     51
Name: status, dtype: int64
```

```
# label encoding for grades
# we have to map values to each of the categories

df['grades'] = df['grades'].replace('O', 0)
df['grades'] = df['grades'].replace('A', 1)
df['grades'] = df['grades'].replace('B', 2)
df['grades'] = df['grades'].replace('C', 3)
df['grades'] = df['grades'].replace('D', 4)
df['grades'] = df['grades'].replace('E', 5)

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

```
3    319
4    262
2    190
5    140
1     89
Name: race/ethnicity, dtype: int64
```

```
df.shape
```

```
(1000, 15)
```

```
# splitting the dependent and independent variables

x = df.iloc[:,:14]
y = df.iloc[:,14]

print(x.shape)
print(y.shape)
```

```
(1000, 14)
(1000,)
```

```
# splitting the dataset into training and test sets

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state =

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(750, 14)
(750,)
(250, 14)
(250,)
```

```
# importing the MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

# creating a scaler
mm = MinMaxScaler()

# feeding the independent variable into the scaler
x_train = mm.fit_transform(x_train)
x_test = mm.transform(x_test)
```

Model_Building:

```
from sklearn.linear_model import  LogisticRegression

# creating a model
model = LogisticRegression(penalty = 'l2', solver = 'lbfgs', multi_class = 'auto', max_ite

# feeding the training data to the model
model.fit(x_train, y_train)

# predicting the test set results
y_pred = model.predict(x_test)

# calculating the classification accuracies
print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))
```

```
Training Accuracy : 0.8546666666666667
Testing Accuracy : 0.82
```

Confusion Matrix:


```
# printing the confusion matrix
```

```
# printing the confusion matrix

from sklearn.metrics import confusion_matrix

# creating a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# printing the confusion matrix
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm, annot = True, cmap = 'Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc4798598d0>
```

Classification Report

```
from sklearn.metrics import classification_report

cr = classification_report(y_test, y_pred)
print(cr)
```

```
                precision    recall  f1-score   support

             0       0.60      0.27      0.37        11
             1       0.69      0.56      0.62        32
             2       0.80      0.83      0.82        72
             3       0.77      0.88      0.82        58
             4       0.92      0.95      0.94        61
             5       1.00      0.94      0.97        16

      accuracy                           0.82       250
     macro avg       0.80      0.74      0.76       250
  weighted avg       0.81      0.82      0.81       250
```