Analyzing The University Dataset

First let's import the necessary libraries.

In [1]:
```
import numpy as np
import pandas as pd
import os
import random
import scipy.stats as st

random.seed(42)
```

Also import the visualization libraries.

In [2]:
```
%matplotlib inline

import matplotlib as mlt
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('ggplot')
```

Let's define a function so that we can easily load the datasets.

In [3]:
```
DATASET_PATH = 'Workable Datasets'

def load_the_dataset(file_name, dataset_path=DATASET_PATH):
    csv_path = os.path.join(dataset_path, file_name)
    return pd.read_csv(csv_path)
```

Let's import the dataset.

In [4]:
```
university_df = load_the_dataset('UNIVERSITY_N.csv')
```

Let's check the data.

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

Out[5]:

	Gender	Age	Popular Website	Proficiency	Medium	Location	Household Internet Facilities	Browse Time	Browsing Status	Resider
0	Female	23	Instagram	Not at all	Desktop	Library	Connected	Night	Daily	Remo
1	Female	23	Youtube	Good	Mobile	University	Connected	Morning	Daily	Remo

	Gender	Age	Popular Website	Proficiency	Medium	Location	Household Internet Facilities	Browse Time	Browsing Status	Resider
2	Female	23	Whatsapp	Good	Mobile	University	Connected	Midnight	Daily	To
3	Female	23	Whatsapp	Average	Laptop and Mobile	University	Connected	Morning	Daily	Villa

Check the dataset using `info()`.

In [6]:
```
university_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 20 columns):
 #   Column                                 Non-Null Count  Dtype 
---  ------                                 --------------  ----- 
 0   Gender                                 301 non-null    object
 1   Age                                    301 non-null    int64 
 2   Popular Website                        301 non-null    object
 3   Proficiency                            301 non-null    object
 4   Medium                                 301 non-null    object
 5   Location                               301 non-null    object
 6   Household Internet Facilities          301 non-null    object
 7   Browse Time                            301 non-null    object
 8   Browsing Status                        301 non-null    object
 9   Residence                              301 non-null    object
 10  Total Internet Usage(hrs/day)          301 non-null    int64 
 11  Time Spent in Academic(hrs/day)        301 non-null    int64 
 12  Purpose of Use                         301 non-null    object
 13  Years of Internet Use                  301 non-null    int64 
 14  Browsing Purpose                       301 non-null    object
 15  Webinar                                301 non-null    object
 16  Priority of Learning                   301 non-null    object
 17  Internet Usage For Educational Purpose 301 non-null    object
 18  Academic Performance                   301 non-null    object
 19  Obstacles                              301 non-null    object
dtypes: int64(4), object(16)
memory usage: 47.2+ KB
```

Let's check the `shape`.

In [7]:
```
university_df.shape
```

Out[7]: (301, 20)

Now let's check all the categorical attributes individually. Start with `Gender` first.

In [8]:
```
university_df['Gender'].value_counts()
```

Out[8]:
```
Male      174
Female    127
Name: Gender, dtype: int64
```

Check Age

In [9]:
```
university_df['Age'].value_counts()
```

Out[9]:
```
23    189
24     76
25     30
22      4
26      1
20      1
Name: Age, dtype: int64
```

Check Frequently Visited Website

In [10]:
```
university_df['Popular Website'].value_counts()
```

Out[10]:
```
Google       129
Youtube       60
Facebook      30
Whatsapp      25
Gmail         25
Instagram     17
Twitter       15
Name: Popular Website, dtype: int64
```

In [11]:
```
university_df.rename(columns={
    'Popular Website':'Frequently Visited Website',
}, inplace=True)

university_df.columns
```

Out[11]:
```
Index(['Gender', 'Age', 'Frequently Visited Website', 'Proficiency', 'Medium',
       'Location', 'Household Internet Facilities', 'Browse Time',
       'Browsing Status', 'Residence', 'Total Internet Usage(hrs/day)',
       'Time Spent in Academic(hrs/day)', 'Purpose of Use',
       'Years of Internet Use', 'Browsing Purpose', 'Webinar',
       'Priority of Learning', 'Internet Usage For Educational Purpose',
       'Academic Performance', 'Obstacles'],
      dtype='object')
```

Check Effectiveness Of Internet Usage

In [12]:
```
university_df['Proficiency'].value_counts()
```

Out[12]:
```
Good          127
Average        92
Very Good      56
Not at all     26
Name: Proficiency, dtype: int64
```

In [13]:
```
university_df.rename(columns={
    'Proficiency':'Effectiveness Of Internet Usage'
}, inplace=True)

university_df.columns
```

Out[13]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Medium', 'Location',
       'Household Internet Facilities', 'Browse Time', 'Browsing Status',
       'Residence', 'Total Internet Usage(hrs/day)',
       'Time Spent in Academic(hrs/day)', 'Purpose of Use',
       'Years of Internet Use', 'Browsing Purpose', 'Webinar',
       'Priority of Learning', 'Internet Usage For Educational Purpose',
       'Academic Performance', 'Obstacles'],
      dtype='object')
```

In [14]:
```
university_df.replace({'Effectiveness Of Internet Usage': {'Very Good':'Very E
                                                          'Average':'Somewhat E
```

In [15]:
```
university_df['Effectiveness Of Internet Usage'].value_counts()
```

Out[15]:
```
Effective             127
Somewhat Effective     92
Very Effective         56
Not at all             26
Name: Effectiveness Of Internet Usage, dtype: int64
```

Check Devices Used For Internet Browsing

In [16]:
```
university_df['Medium'].value_counts()
```

Out[16]:
```
Laptop and Mobile    164
Mobile                91
Desktop               33
Laptop                13
Name: Medium, dtype: int64
```

In [17]:
```
university_df.rename(columns={
    'Medium':'Devices Used For Internet Browsing',
}, inplace=True)

university_df.columns
```

Out[17]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location', 'Household Internet Facilities', 'Browse Time',
       'Browsing Status', 'Residence', 'Total Internet Usage(hrs/day)',
       'Time Spent in Academic(hrs/day)', 'Purpose of Use',
       'Years of Internet Use', 'Browsing Purpose', 'Webinar',
       'Priority of Learning', 'Internet Usage For Educational Purpose',
       'Academic Performance', 'Obstacles'],
      dtype='object')
```

Check Location Of Internet Use

In [18]:
```
university_df['Location'].value_counts()
```

Out[18]:
```
University    119
```

```
Library        67
Home           61
Cyber Cafe     48
Others          6
Name: Location, dtype: int64
```

In [19]:
```
university_df.rename(columns={
    'Location':'Location Of Internet Use'
}, inplace=True)

university_df.columns
```

Out[19]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location Of Internet Use', 'Household Internet Facilities',
       'Browse Time', 'Browsing Status', 'Residence',
       'Total Internet Usage(hrs/day)', 'Time Spent in Academic(hrs/day)',
       'Purpose of Use', 'Years of Internet Use', 'Browsing Purpose',
       'Webinar', 'Priority of Learning',
       'Internet Usage For Educational Purpose', 'Academic Performance',
       'Obstacles'],
      dtype='object')
```

Check Household Internet Facilities

In [20]:
```
university_df['Household Internet Facilities'].value_counts()
```

Out[20]:
```
Connected        270
Not Connected     31
Name: Household Internet Facilities, dtype: int64
```

Check Time Of Internet Browsing

In [21]:
```
university_df['Browse Time'].value_counts()
```

Out[21]:
```
Night       106
Day          67
Morning      65
Midnight     63
Name: Browse Time, dtype: int64
```

In [22]:
```
university_df.rename(columns={
       'Browse Time':'Time Of Internet Browsing',
}, inplace=True)

university_df.columns
```

Out[22]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location Of Internet Use', 'Household Internet Facilities',
       'Time Of Internet Browsing', 'Browsing Status', 'Residence',
       'Total Internet Usage(hrs/day)', 'Time Spent in Academic(hrs/day)',
       'Purpose of Use', 'Years of Internet Use', 'Browsing Purpose',
       'Webinar', 'Priority of Learning',
       'Internet Usage For Educational Purpose', 'Academic Performance',
       'Obstacles'],
      dtype='object')
```

Check Frequency Of Internet Usage

In [23]:
```
university_df['Browsing Status'].value_counts()
```

Out[23]:
```
Daily      269
Weekly      27
Monthly      5
Name: Browsing Status, dtype: int64
```

In [24]:
```
university_df.rename(columns={
    'Browsing Status':'Frequency Of Internet Usage',
}, inplace=True)

university_df.columns
```

Out[24]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location Of Internet Use', 'Household Internet Facilities',
       'Time Of Internet Browsing', 'Frequency Of Internet Usage', 'Residence
',
       'Total Internet Usage(hrs/day)', 'Time Spent in Academic(hrs/day)',
       'Purpose of Use', 'Years of Internet Use', 'Browsing Purpose',
       'Webinar', 'Priority of Learning',
       'Internet Usage For Educational Purpose', 'Academic Performance',
       'Obstacles'],
      dtype='object')
```

Check Place Of Student's Residence

In [25]:
```
university_df['Residence'].value_counts()
```

Out[25]:
```
Town       213
Village     80
Remote       8
Name: Residence, dtype: int64
```

In [26]:
```
university_df.rename(columns={
    'Residence':'Place Of Student\'s Residence',
}, inplace=True)

university_df.columns
```

Out[26]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location Of Internet Use', 'Household Internet Facilities',
       'Time Of Internet Browsing', 'Frequency Of Internet Usage',
       'Place Of Student's Residence', 'Total Internet Usage(hrs/day)',
       'Time Spent in Academic(hrs/day)', 'Purpose of Use',
       'Years of Internet Use', 'Browsing Purpose', 'Webinar',
       'Priority of Learning', 'Internet Usage For Educational Purpose',
       'Academic Performance', 'Obstacles'],
      dtype='object')
```

Check Purpose Of Internet Use'

In [27]:
```
university_df['Purpose of Use'].value_counts()
```

Out[27]:
```
Education           148
Social Media         43
Entertainment        34
News                 34
Online Shopping      31
Blog                 11
Name: Purpose of Use, dtype: int64
```

In [28]:
```
university_df.rename(columns={
    'Purpose of Use':'Purpose Of Internet Use',
}, inplace=True)

university_df.columns
```

Out[28]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location Of Internet Use', 'Household Internet Facilities',
       'Time Of Internet Browsing', 'Frequency Of Internet Usage',
       'Place Of Student's Residence', 'Total Internet Usage(hrs/day)',
       'Time Spent in Academic(hrs/day)', 'Purpose Of Internet Use',
       'Years of Internet Use', 'Browsing Purpose', 'Webinar',
       'Priority of Learning', 'Internet Usage For Educational Purpose',
       'Academic Performance', 'Obstacles'],
      dtype='object')
```

Check Browsing Purpose

In [29]:
```
university_df['Browsing Purpose'].value_counts()
```

Out[29]:
```
Academic        200
Non-academic    101
Name: Browsing Purpose, dtype: int64
```

Check Webinar

In [30]:
```
university_df['Webinar'].value_counts()
```

Out[30]:
```
Yes    209
No      92
Name: Webinar, dtype: int64
```

Check Priority Of Learning On The Internet

In [31]:
```
university_df['Priority of Learning'].value_counts()
```

Out[31]:
```
Academic Learning                    89
Communication Skills                 53
Creativity and Innovative Skills     47
Non-academic Learning                42
Leadership Development               42
Career Opportunity                   28
Name: Priority of Learning, dtype: int64
```

In [32]:
```
university_df.rename(columns={
    'Priority of Learning':'Priority Of Learning On The Internet',
}, inplace=True)

university_df.columns
```

Out[32]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location Of Internet Use', 'Household Internet Facilities',
       'Time Of Internet Browsing', 'Frequency Of Internet Usage',
       'Place Of Student's Residence', 'Total Internet Usage(hrs/day)',
       'Time Spent in Academic(hrs/day)', 'Purpose Of Internet Use',
       'Years of Internet Use', 'Browsing Purpose', 'Webinar',
       'Priority Of Learning On The Internet',
       'Internet Usage For Educational Purpose', 'Academic Performance',
       'Obstacles'],
      dtype='object')
```

Check Internet Usage For Educational Purpose

In [33]:
```
university_df['Internet Usage For Educational Purpose'].value_counts()
```

Out[33]:
```
Articles or Blogs related to academical studies        64
E-books or other Media files                           52
Research/Journal/Conference Papers                     49
Notes or lectures for academical purpose               48
Articles or Blogs related to non-academical studies    48
Courses Available on specific topics                   40
Name: Internet Usage For Educational Purpose, dtype: int64
```

Check Academic Performance

In [34]:
```
university_df['Academic Performance'].value_counts()
```

Out[34]:
```
Good                144
Satisfactory        100
Average              33
Not Satisfactory     24
Name: Academic Performance, dtype: int64
```

In [35]:
```
university_df.replace({'Academic Performance': {'Good':'Excellent', 'Satisfact
```

In [36]:
```
university_df['Academic Performance'].value_counts()
```

Out[36]:
```
Excellent           144
Good                100
Average              33
Not Satisfactory     24
Name: Academic Performance, dtype: int64
```

Check Barriers To Internet Access

In [37]:
```
university_df['Obstacles'].value_counts()
```

Out[37]:
```
High Price        190
Bad Service        89
Unavailability     22
```

In [38]:
```
university_df.rename(columns={
    'Obstacles':'Barriers To Internet Access',
}, inplace=True)

university_df.columns
```

Out[38]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location Of Internet Use', 'Household Internet Facilities',
       'Time Of Internet Browsing', 'Frequency Of Internet Usage',
       'Place Of Student's Residence', 'Total Internet Usage(hrs/day)',
       'Time Spent in Academic(hrs/day)', 'Purpose Of Internet Use',
       'Years of Internet Use', 'Browsing Purpose', 'Webinar',
       'Priority Of Learning On The Internet',
       'Internet Usage For Educational Purpose', 'Academic Performance',
       'Barriers To Internet Access'],
      dtype='object')
```

Plot the data

Now we can plot the data. Let's write a couple of functions so that we easily plot the data.

This function saves the figures.

In [39]:
```
# Write a function to save the figures
PROJECT_ROOT_DIR = "."
DATASET_ID = "University"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "Figures", DATASET_ID)
os.makedirs(IMAGES_PATH, exist_ok = True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

This function plots histogram and box plot of the given non-categorical data.

In [40]:
```
def numerical_data_plot(dataframe, fig_id, hist_alpha=0.6, color='crimson',
                        title='Image Title', xlabel='X Label', ylabel='Y Label

#     plt.figure(figsize=(10, 6))
#     sns.set(font_scale=1.5)

#     plt.subplot(121)

    count, bin_edges = np.histogram(dataframe)
    dataframe.plot(kind='hist', alpha=hist_alpha,
                   xticks=bin_edges, color=color)

    # Let's add a KDE plot
#     mn, mx = plt.xlim()
#     plt.xlim(mn, mx)
#     kde_x = np.linspace(mn, mx, 300)
#     kde = st.gaussian_kde(dataframe)
#     plt.plot(kde_x, kde.pdf(kde_x) * kde_mul, 'k--', color=color)
#     kde_mul=1000,

#     plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

#     plt.subplot(122)
#     red_circle = dict(markerfacecolor='r', marker='o')
#     dataframe.plot(kind='box', color=color, flierprops=red_circle)

#     save_fig(fig_id)
```

This function plots histograms of the given categorical data.

In [41]:
```
def categorical_bar_plot(dataframe, rot=0, alpha=0.80, color = ['steelblue',
                         title='Distribution', xlabel = 'Column name', ylabel=

    dataframe.value_counts().plot(kind='bar', rot=rot, alpha=alpha, color=colo

    plt.title(title, fontweight='bold')
    plt.xlabel(xlabel, fontweight='bold')
    plt.ylabel(ylabel, fontweight='bold')
```

let's define a function to create scatter plots of the numerical values and check the distribution of the attribute values against the target column, `Academic Performance`

In [42]:
```
def categorical_scatter_plot(dataframe, x_column, y_column, title, legend_titl
                             y_label, x_label = 'Number of students'):

    plt.figure(figsize=(15, 7))
    sns.set(font_scale=1.5)
    sns.set_style("whitegrid", {'axes.grid' : False})

    plt.plot(dataframe[x_column].loc[dataframe[y_column] == 'Excellent'].inde
             dataframe[x_column].loc[dataframe[y_column] == 'Excellent'],
             'bo', label = 'Excellent')

    plt.plot(dataframe[x_column].loc[dataframe[y_column] == 'Good'].index,
             dataframe[x_column].loc[dataframe[y_column] == 'Good'],
             'yo', label = 'Good')

    plt.plot(dataframe[x_column].loc[dataframe[y_column] == 'Average'].index,
             dataframe[x_column].loc[dataframe[y_column] == 'Average'],
             'go', label = 'Average')

    plt.plot(dataframe[x_column].loc[dataframe[y_column] == 'Not Satisfactory
             dataframe[x_column].loc[dataframe[y_column] == 'Not Satisfactory
             'ro', label = 'Not Satisfactory')

#     plt.title(title, fontweight='bold')
    plt.xlabel(x_label, fontweight='bold')
    plt.ylabel(y_label, fontweight='bold')
    plt.legend(title = legend_title, title_fontsize=14, loc='lower right', fo
```

A modification of the previous function to create scatter plots of the numerical values vs numerical values and check the distribution of the attribute values against the target column, `Academic Performance`

In [43]:
```
def categorical_scatter_plot_wrt_academic_performance(dataframe, x_column, y_c
                              y_label, x_label, legend_title):

    plt.figure(figsize=(15, 7))
    sns.set(font_scale=1.2)
    sns.set_style("whitegrid", {'axes.grid' : False})

    plt.plot(dataframe[x_column].loc[dataframe['Academic Performance'] == 'Exc
             dataframe[y_column].loc[dataframe['Academic Performance'] == 'Exc
             'bo', label = 'Excellent')

    plt.plot(dataframe[x_column].loc[dataframe['Academic Performance'] == 'Goo
             dataframe[y_column].loc[dataframe['Academic Performance'] == 'Goo
             'yo', label = 'Good')

    plt.plot(dataframe[x_column].loc[dataframe['Academic Performance'] == 'Ave
             dataframe[y_column].loc[dataframe['Academic Performance'] == 'Ave
             'go', label = 'Average')

    plt.plot(dataframe[x_column].loc[dataframe['Academic Performance'] == 'Not
             dataframe[y_column].loc[dataframe['Academic Performance'] == 'Not
             'ro', label = 'Not Satisfactory')

#     plt.title(title, fontweight='bold')
    plt.xlabel(x_label, fontweight='bold')
    plt.ylabel(y_label, fontweight='bold')
    plt.legend(title = legend_title, loc='upper right', fontsize=14)
```

This function plot histograms of the categorical values against the `'Academic Performance'` column.

These are helper functions.

In [44]:
```
def init_dictionary(dictionary, labels):
    for label in labels:
        dictionary[label] = []

def append_to_dict(dictionary, indexes, values):
    x = 0
    for index in indexes:
        dictionary[index].append(values[x])
        x += 1

def furnish_the_lists(labels, indexes, values):
    list_dif = [i for i in labels + indexes if i not in labels or i not in inc

    indexes.extend(list_dif)
    for i in range(len(list_dif)):
        values.append(0)

def append_dataframe_to_dict(dataframe, column_name, labels, dictionary):
    values = dataframe[column_name].value_counts().tolist()
    indexes = dataframe[column_name].value_counts().index.tolist()
    furnish_the_lists(labels, indexes, values)
    append_to_dict(dictionary, indexes, values)

    return dictionary
```

This is the main function.

In [45]:
```
def cat_vs_cat_bar_plot(dataframe, column_name, column_cat_list):
    excellent_result_df = dataframe.loc[dataframe['Academic Performance'] ==
    good_result_df = dataframe.loc[dataframe['Academic Performance'] == 'Good
    average_result_df = dataframe.loc[dataframe['Academic Performance'] == 'Av
    unsatisfactory_result_df = dataframe.loc[dataframe['Academic Performance']

    labels = column_cat_list
    dictionary = {}

    init_dictionary(dictionary, labels)

    dictionary = append_dataframe_to_dict(excellent_result_df, column_name, la
    dictionary = append_dataframe_to_dict(good_result_df, column_name, labels,
    dictionary = append_dataframe_to_dict(average_result_df, column_name, labe
    dictionary = append_dataframe_to_dict(unsatisfactory_result_df, column_nam

    return dictionary
```

The following function does the same thing with respect to `'Browsing Purpose'`

In [46]:
```
def cat_vs_cat_bar_plot_browsing_purpose(dataframe, column_name, column_cat_li
    academic_df = dataframe.loc[dataframe['Browsing Purpose'] == 'Academic']
    non_academic_df = dataframe.loc[dataframe['Browsing Purpose'] == 'Non-acad

    labels = column_cat_list
    dictionary = {}

    init_dictionary(dictionary, labels)

    dictionary = append_dataframe_to_dict(academic_df, column_name, labels, di
    dictionary = append_dataframe_to_dict(non_academic_df, column_name, labels

    return dictionary
```

This function add value counts on top of each bar in the histogram.

In [47]:
```
def autolabel(rects):

    total_height = 0

    for rect in rects:
        total_height += rect.get_height()

    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format("{:.2f}".format((height/total_height)*100)) +
                    xy = (rect.get_x() + rect.get_width()/2, height),
                    xytext = (0, 3), # 3 points vertical offset
                    textcoords = "offset points",
                    ha = 'center', va = 'bottom')
```

Now let's start plotting the data.

Plotting Non-Categorical Values

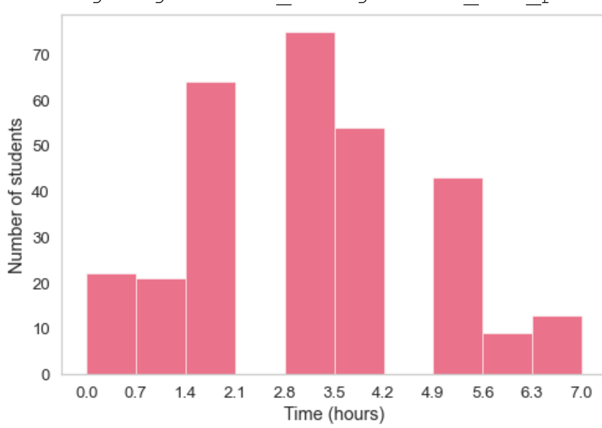
Only `'Total Internet Usage(hrs/day)'`, `'Time Spent in Academic(hrs/day)'`, `'Duration Of Internet Usage(In Years)'` are the non-categorical values in the dataset.

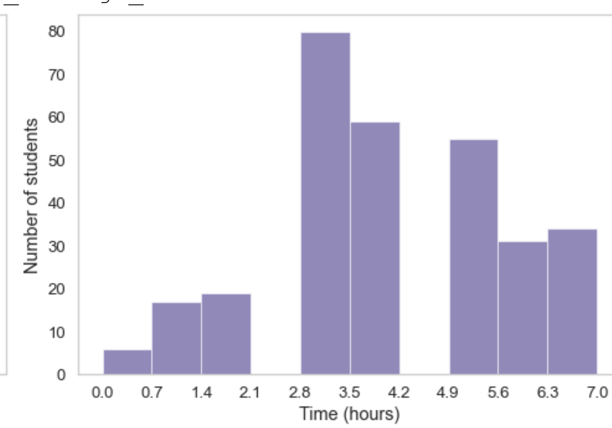
Let's plot the bar plot for each of the non-categorical attributes together.

In [48]:

```
plt.figure(figsize=(14, 5))
plt.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)
sns.set(font_scale=1.2)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.subplot(121)
numerical_data_plot(university_df['Total Internet Usage(hrs/day)'], 'Total_Int
                    title = 'Total internet usage in a day',
                    xlabel = 'Time (hours)', ylabel = 'Number of students')

plt.subplot(122)
numerical_data_plot(university_df['Time Spent in Academic(hrs/day)'], 'Time_Sp
                    hist_alpha=0.6, color='darkslateblue',
                    title='Total Time spent in academic studies in a day',
                    xlabel='Time (hours)', ylabel='Number of students')


save_fig('Non_Categorical_Bar_plot_collage_1')

plt.show()
```

```
Saving figure Non_Categorical_Bar_plot_collage_1
```





In [49]:

```
# plt.figure(figsize=(7, 5))
# plt.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)
# sns.set(font_scale=1.2)
# sns.set_style("whitegrid", {'axes.grid' : False})

# numerical_data_plot(university_df['Duration Of Internet Usage(In Years)'],
#                     hist_alpha=0.6, color='salmon',
#                     title='How long have the students been using internet?',
#                     ylabel='Number of students')


# save_fig('Non_Categorical_Bar_plot_2')

# plt.show()
```

Plotting `Total Internet Usage(hrs/day)`

In [50]:
```
university_df['Total Internet Usage(hrs/day)'].value_counts()
```

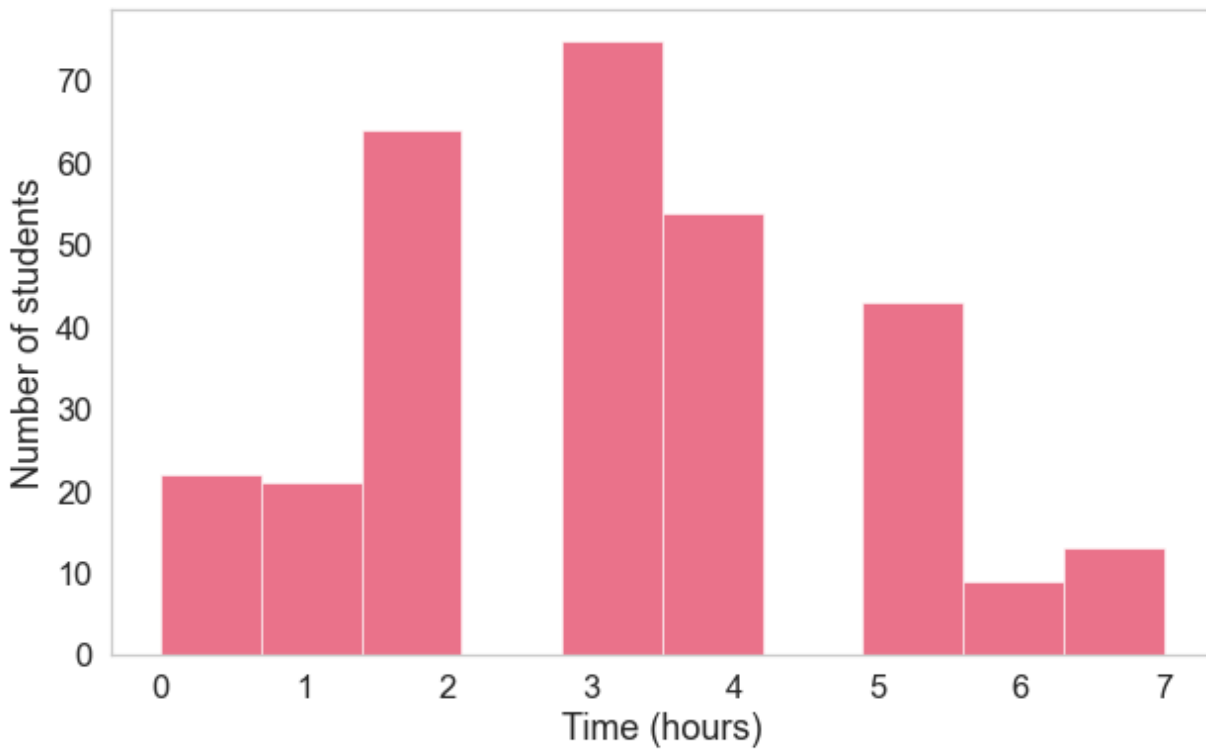
Out[50]:
```
3    75
2    64
4    54
5    43
0    22
1    21
7    13
6     9
Name: Total Internet Usage(hrs/day), dtype: int64
```

First let's check the histogram and the boxplot of this column.

In [51]:
```
# numerical_data_plot(university_df['Total Internet Usage(hrs/day)'], 'Total_
#                     title = 'Total internet usage in a day',
#                     xlabel = 'Time (hours)', ylabel = 'Number of Students')
```

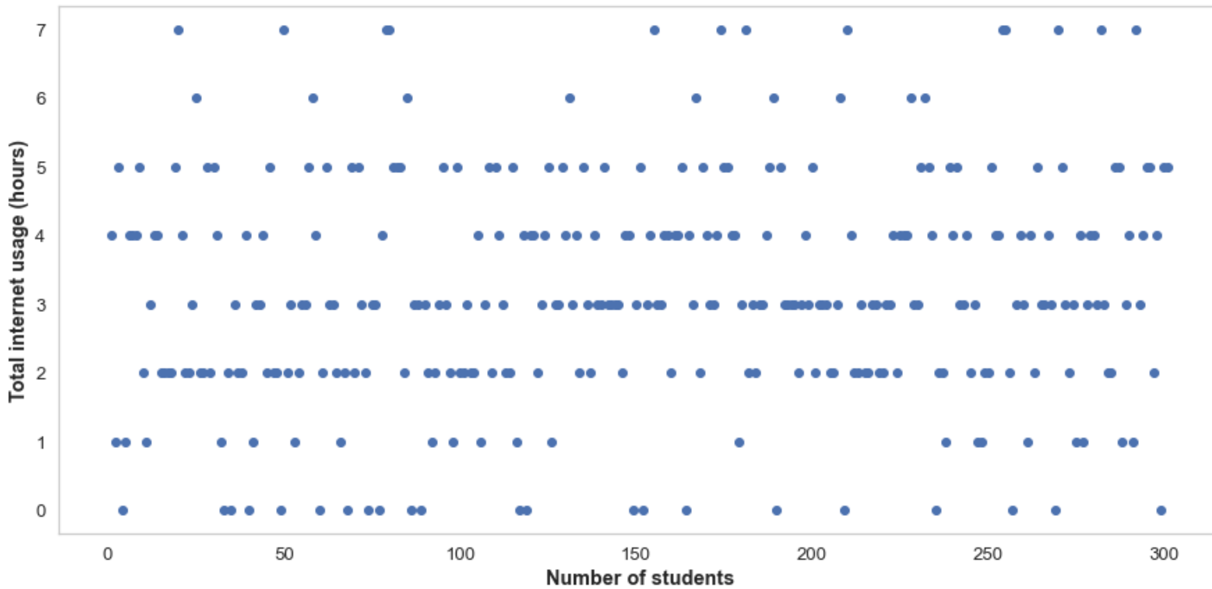
Now let's do it without the box plot.

In [52]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

university_df['Total Internet Usage(hrs/day)'].plot(kind='hist', alpha=0.6, co

plt.xlabel('Time (hours)')
plt.ylabel('Number of students')

plt.show()
```



Now let's check the scatter plot.

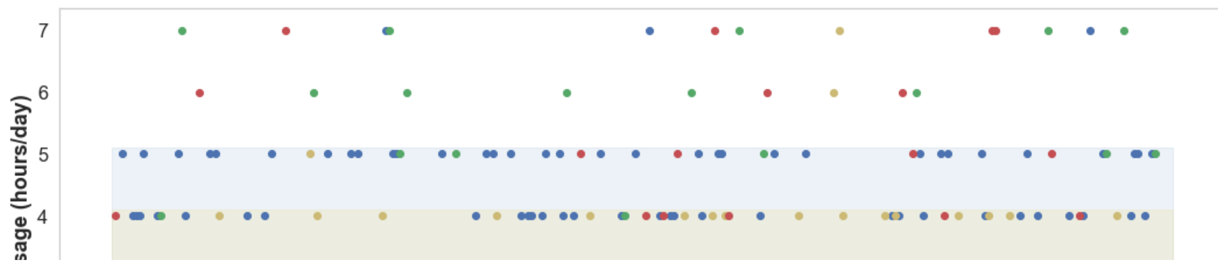
In [53]:
```
plt.figure(figsize=(15,7))
sns.set(font_scale=1.2)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.plot(np.linspace(1, len(university_df.index), len(university_df.index)),
         university_df['Total Internet Usage(hrs/day)'], 'bo')

plt.ylabel('Total internet usage (hours)', fontweight='bold')
plt.xlabel('Number of students', fontweight='bold')


plt.show()
```



Now let's try plotting `Total Internet Usage(hrs/day)` against the target column `'Academic Performance'`.

In [54]:
```
categorical_scatter_plot(university_df, 'Total Internet Usage(hrs/day)', 'Acad
                        'Total Internet Usage In A Day W.R.T. Academic Perforn
                         'Total internet usage (hours/day)')

plt.fill_between([-1, 305], [4.1, 4.1], -0.2, color='gold', alpha=0.1, interpo
plt.fill_between([-1, 305], [5.1, 5.1], 1.9, color='steelblue', alpha=0.1, int
# plt.fill_between([-1, 305], [8.1, 8.1], 3.8, color='red', alpha=0.1, interpo

save_fig('Total_Internet_Usage_In_A_Day_WRT_Academic_Performance_Scatter_Plot

plt.show()
```

```
Saving figure Total_Internet_Usage_In_A_Day_WRT_Academic_Performance_Scatter_P
lot
```



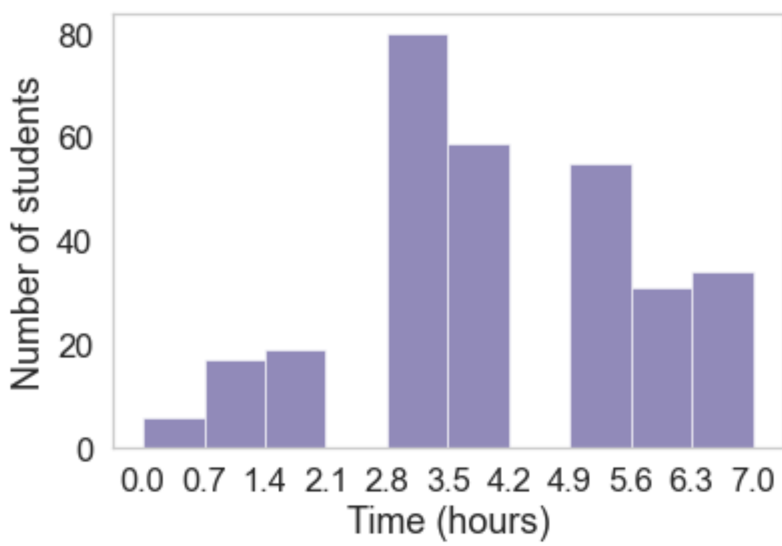
Plotting Time Spent in Academic(hrs/day)

In [55]:
```
university_df['Time Spent in Academic(hrs/day)'].value_counts()
```

Out[55]:
```
3    80
4    59
5    55
7    34
6    31
2    19
1    17
0     6
Name: Time Spent in Academic(hrs/day), dtype: int64
```

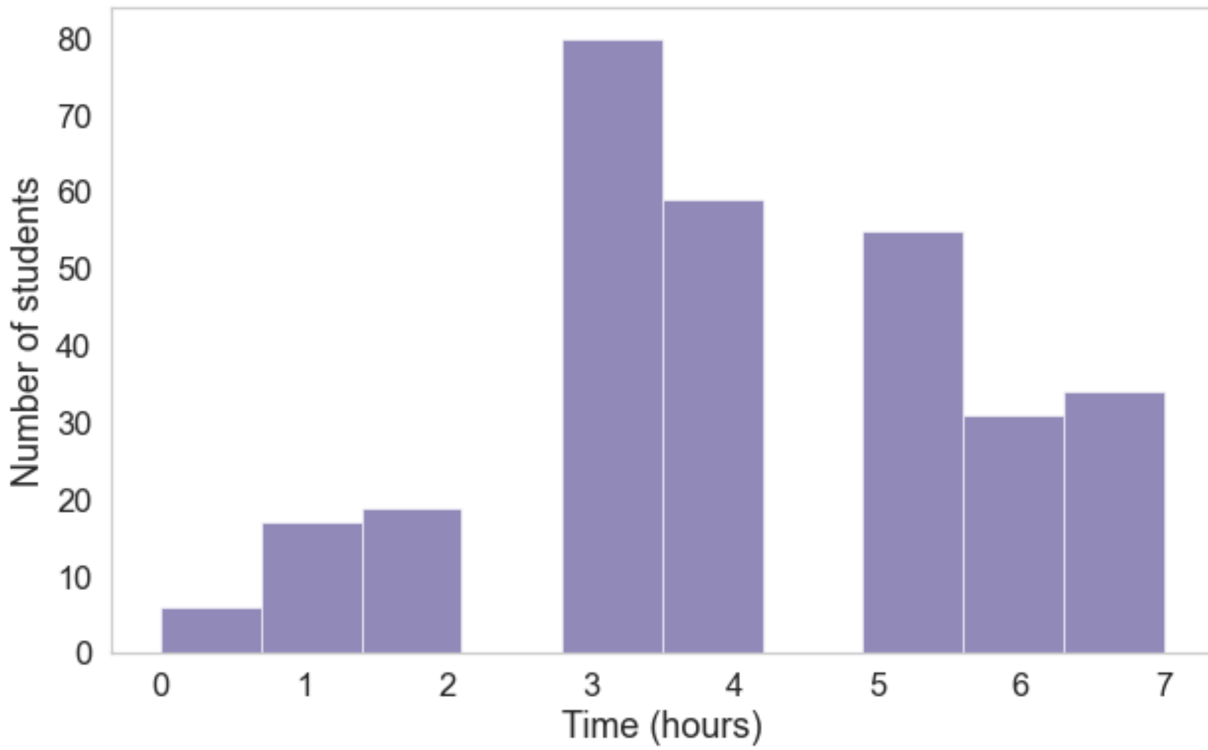
First let's check the histogram and the boxplot of this column.

In [56]:
```
numerical_data_plot(university_df['Time Spent in Academic(hrs/day)'], 'Time_Sp
                    hist_alpha=0.6, color='darkslateblue',
                    title='Total time spent in academic studies in a day',
                    xlabel='Time (hours)', ylabel='Number of students')

plt.show()
```



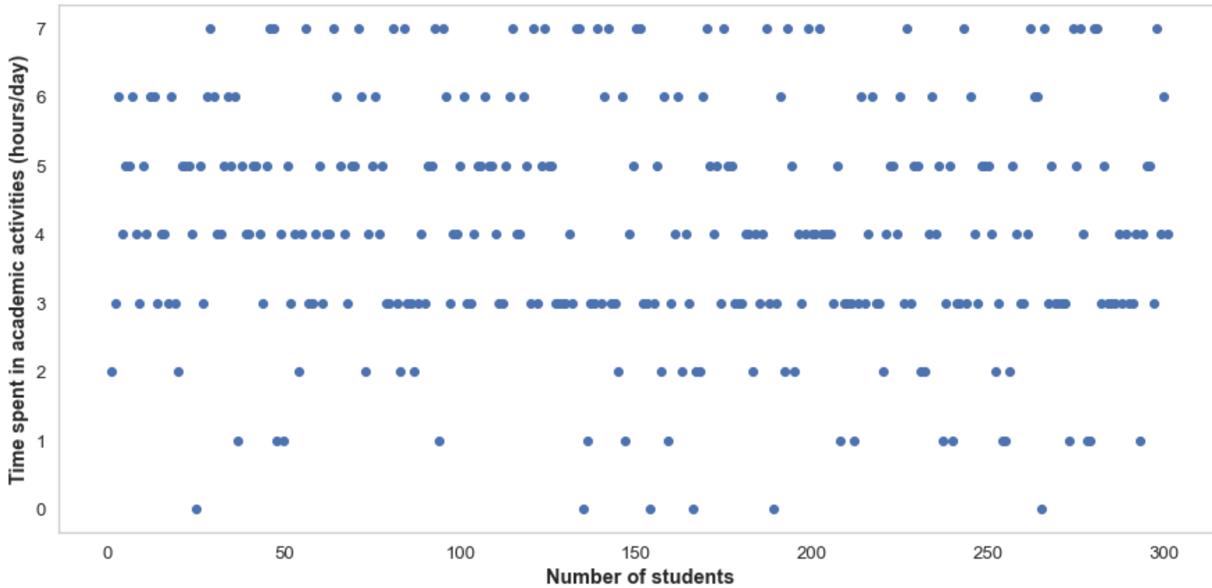
Now let's do it without the box plot.

In [57]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

university_df['Time Spent in Academic(hrs/day)'].plot(kind='hist', alpha=0.6,

# plt.title('Total time spent in academic studies in a day')
plt.xlabel('Time (hours)')
plt.ylabel('Number of students')

plt.show()
```



Now let's check the scatter plot.

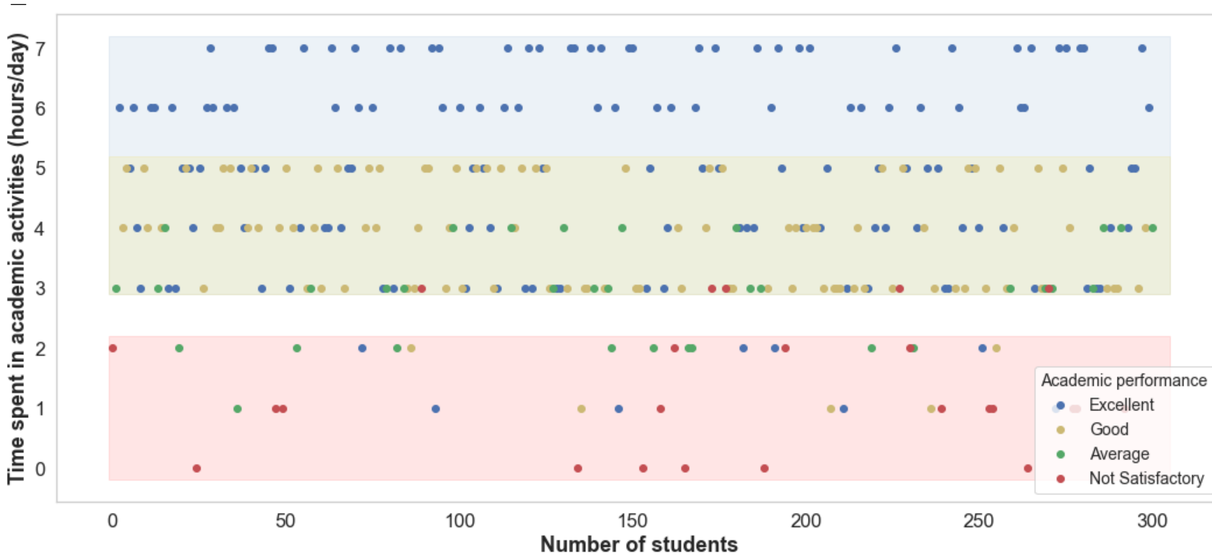
In [58]:
```
plt.figure(figsize=(15,7))
sns.set(font_scale=1.2)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.plot(np.linspace(1, len(university_df.index), len(university_df.index)),
         university_df['Time Spent in Academic(hrs/day)'], 'bo')

# plt.title('Total time spent in academic in a day', fontweight='bold')
plt.ylabel('Time spent in academic activities (hours/day)', fontweight='bold')
plt.xlabel('Number of students', fontweight='bold')

plt.show()
```



Now let's try plotting `Time Spent in Academic(hrs/day)` against the target column `'Academic Performance'` .

In [59]:
```
categorical_scatter_plot(university_df, 'Time Spent in Academic(hrs/day)', 'Ac
                         'Time Spent In Academic In A Day W.R.T. Academic Per
                         'Time spent in academic activities (hours/day)')

plt.fill_between([-1, 305], [7.2, 7.2], 2.9, color='steelblue', alpha=0.1, int
plt.fill_between([-1, 305], [5.2, 5.2], 2.9, color='gold', alpha=0.1, interpol
plt.fill_between([-1, 305], [2.2, 2.2], -0.2, color='red', alpha=0.1, interpol

save_fig('Time_Spent_In_Academic_In_A_Day_WRT_Academic_Performance_Scatter_Plo
plt.show()
```

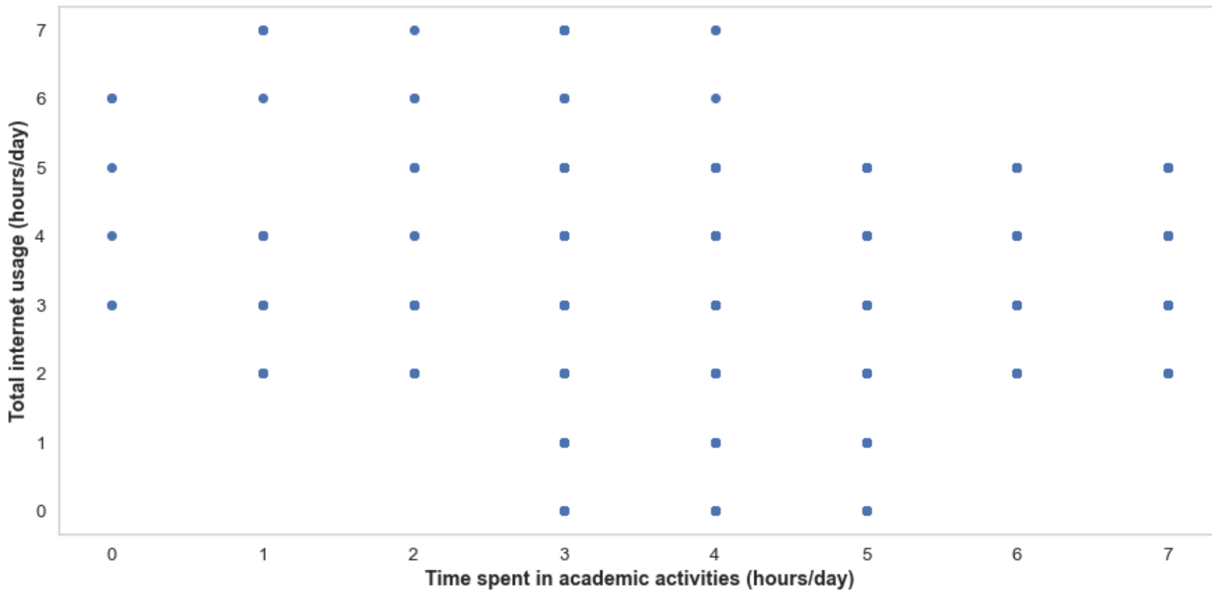
```
Saving figure Time_Spent_In_Academic_In_A_Day_WRT_Academic_Performance_Scatter
_Plot
```



Plotting `Time Spent in Academic(hrs/day)` vs `Total Internet Usage(hrs/day)`

Let's use scatter plot.

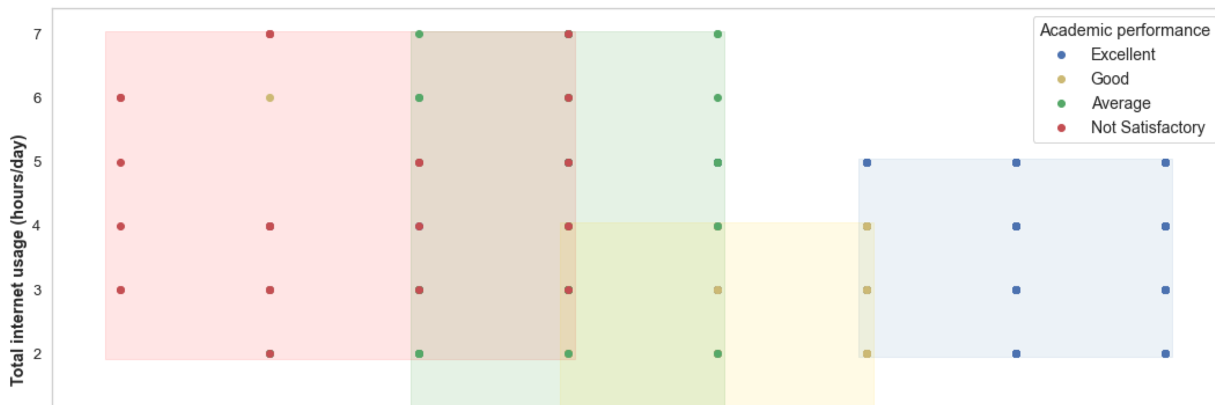
In [60]:
```
plt.figure(figsize=(15, 7))
sns.set(font_scale=1.2)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.plot(university_df['Time Spent in Academic(hrs/day)'],
         university_df['Total Internet Usage(hrs/day)'], 'bo')

# plt.title('Time Spent in Academic(hrs/day) vs Total Internet Usage(hrs/day)
plt.xlabel('Time spent in academic activities (hours/day)', fontweight='bold')
plt.ylabel('Total internet usage (hours/day)', fontweight='bold')

plt.show()
```



Now let's try plotting `Time Spent in Academic(hrs/day)` **vs** `'Total Internet Usage(hrs/day)'` **against the target** `'Academic Performance'`**.**

In [61]:
```
categorical_scatter_plot_wrt_academic_performance(university_df,  'Time Spent
                                                  'Total Internet Usage(hrs/da
                                                  'Time Spent in Academic(hrs,
                                                  'Total internet usage (hours
                                                  'Time spent in academic acti
                                                  'Academic performance')


plt.fill_between([-0.1, 3.05], [7.05, 7.05], 1.9, color='red', alpha=0.1, inte
plt.fill_between([1.95, 4.05], [7.05, 7.05], 0.9, color='green', alpha=0.1, in
plt.fill_between([4.95, 7.05], [5.05, 5.05], 1.95, color='steelblue', alpha=0
plt.fill_between([2.95, 5.05], [4.05, 4.05], -0.1, color='gold', alpha=0.1, in

save_fig('Time_Spent_in_Academic_vs_Total_Internet_Usage_Scatter_Plot')

plt.show()
```

```
Saving figure Time_Spent_in_Academic_vs_Total_Internet_Usage_Scatter_Plot
```



Plotting Duration Of Internet Usage(In Years)

In [62]:
```
university_df.rename(columns={
    'Years of Internet Use':'Duration Of Internet Usage(In Years)',
}, inplace=True)

university_df.columns
```

Out[62]:
```
Index(['Gender', 'Age', 'Frequently Visited Website',
       'Effectiveness Of Internet Usage', 'Devices Used For Internet Browsing
',
       'Location Of Internet Use', 'Household Internet Facilities',
       'Time Of Internet Browsing', 'Frequency Of Internet Usage',
       'Place Of Student's Residence', 'Total Internet Usage(hrs/day)',
       'Time Spent in Academic(hrs/day)', 'Purpose Of Internet Use',
       'Duration Of Internet Usage(In Years)', 'Browsing Purpose', 'Webinar',
       'Priority Of Learning On The Internet',
       'Internet Usage For Educational Purpose', 'Academic Performance',
       'Barriers To Internet Access'],
      dtype='object')
```

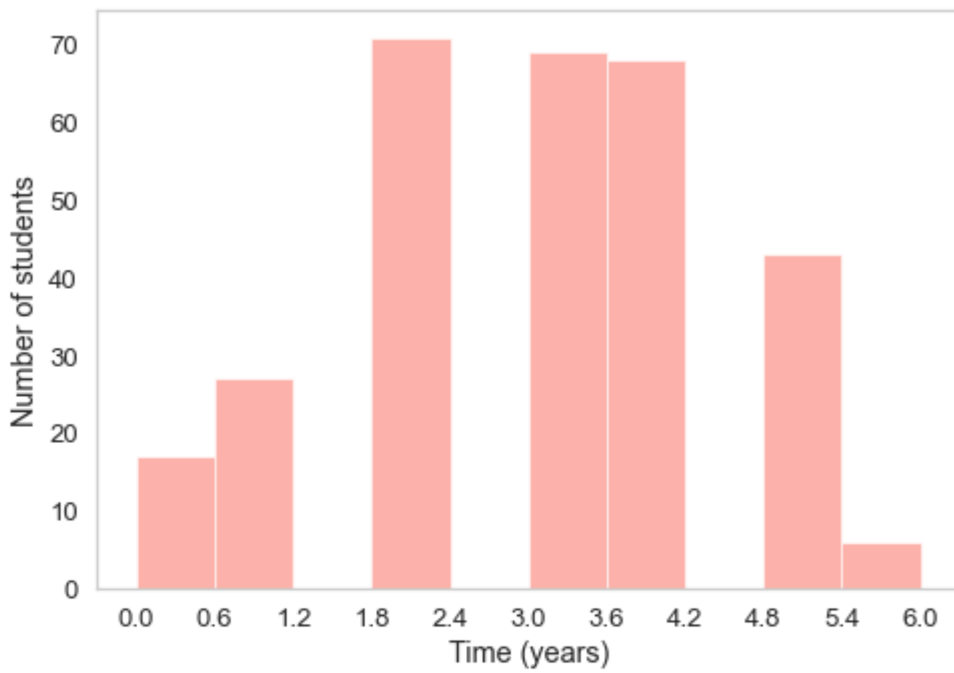
In [63]:
```
university_df['Duration Of Internet Usage(In Years)'].value_counts()
```

Out[63]:
```
2    71
3    69
4    68
5    43
1    27
0    17
6     6
Name: Duration Of Internet Usage(In Years), dtype: int64
```

First let's check the histogram and the boxplot of this column.

In [64]:
```
plt.figure(figsize=(7, 5))
plt.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)
sns.set(font_scale=1.2)
sns.set_style("whitegrid", {'axes.grid' : False})

numerical_data_plot(university_df['Duration Of Internet Usage(In Years)'], 'Du
                    hist_alpha=0.6, color='salmon',
                    title='How long have the students been using internet?', x
                    ylabel='Number of students')


save_fig('Non_Categorical_Bar_plot_2')

plt.show()
```

```
Saving figure Non_Categorical_Bar_plot_2
```



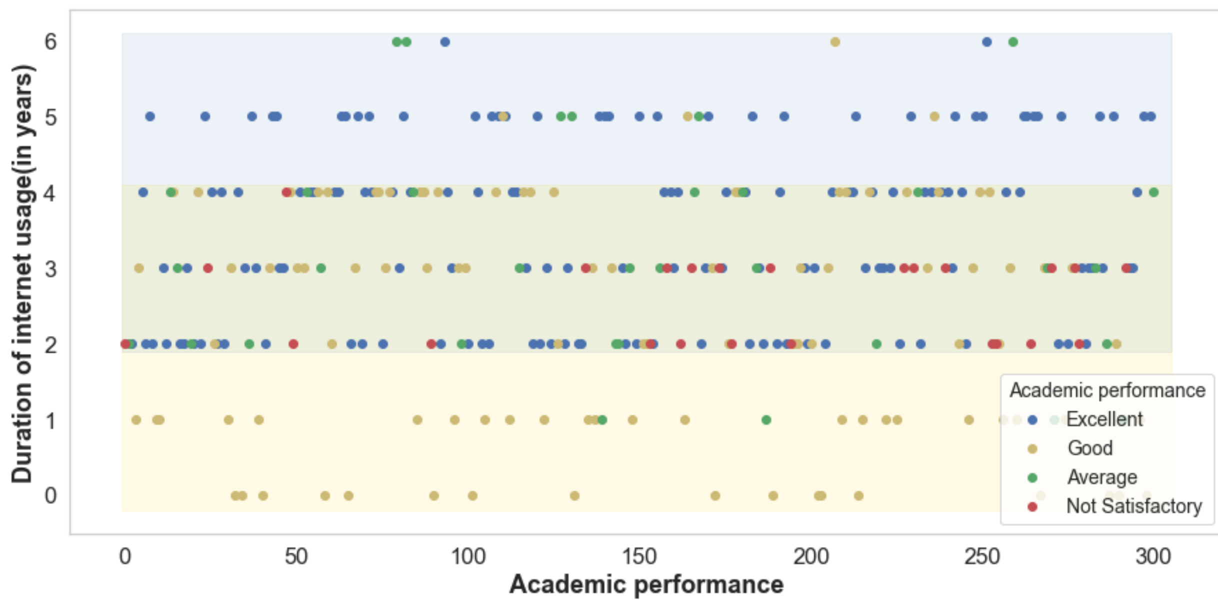
Now let's check the scatter plot.

In [65]:
```
plt.figure(figsize=(15, 7))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.plot(np.linspace(1, len(university_df.index), len(university_df.index)),
         university_df['Duration Of Internet Usage(In Years)'], 'bo')

# plt.title('Duration Of Internet Usage (In Years)', fontweight='bold')
plt.ylabel('Years', fontweight='bold')
plt.xlabel('Number of students', fontweight='bold')

save_fig('Duration_Of_Internet_Usage_In_Years_Scatter_Plot')
plt.show()
```

```
Saving figure Duration_Of_Internet_Usage_In_Years_Scatter_Plot
```

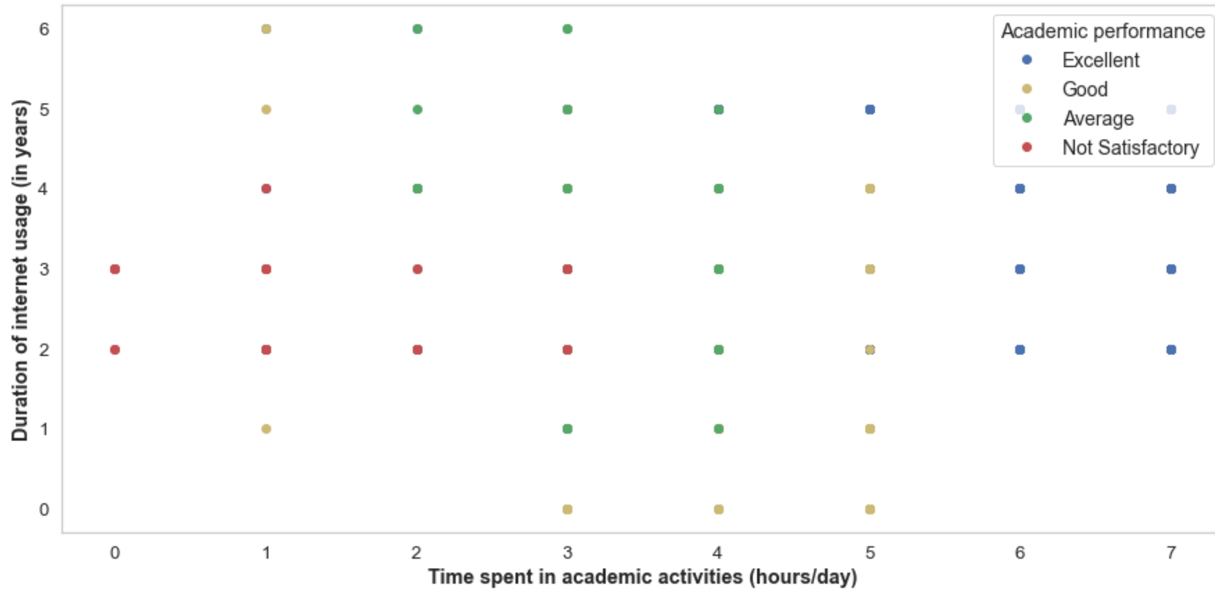
Now let's try plotting `'Duration Of Internet Usage(In Years)'` against the target column `'Academic Performance'`.

In [66]:
```
categorical_scatter_plot(university_df, 'Duration Of Internet Usage(In Years)
                         'Duration Of Internet Usage(In Years) vs Academic Pe
                         'Duration of internet usage(in years)', 'Academic pe

plt.fill_between([-1, 305], [6.1, 6.1], 1.9, color='steelblue', alpha=0.1, int
plt.fill_between([-1, 305], [4.1, 4.1], -0.2, color='gold', alpha=0.1, interpo

plt.show()
```



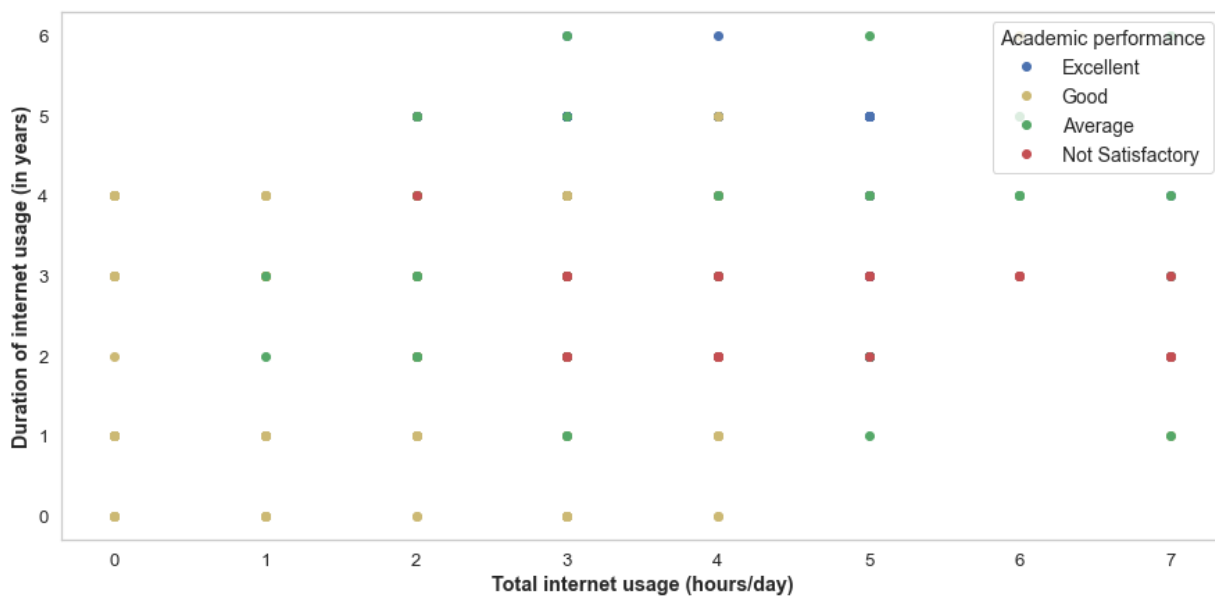
Now let's try plotting `Time Spent in Academic(hrs/day)` vs `'Duration Of Internet Usage(In Years)'` against the target `'Academic Performance'`.

In [67]:
```
categorical_scatter_plot_wrt_academic_performance(university_df, 'Time Spent 
                                                   'Duration Of Internet Usage
                                                   'Time Spent in Academic (hr
                                                   'Duration of internet usage
                                                   'Time spent in academic act


plt.show()
```



Now let's try plotting `'Total Internet Usage(hrs/day)'` **vs** `'Duration Of Internet Usage(In Years)'` **against the target** `'Academic Performance'` **.**

In [68]:
```
categorical_scatter_plot_wrt_academic_performance(university_df, 'Total Intern
                                                  'Duration Of Internet Usage
                                                  'Total Internet Usage (hrs/c
                                                  'Duration of internet usage
                                                  'Total internet usage (hours

plt.show()
```



Plotting Categorical Values

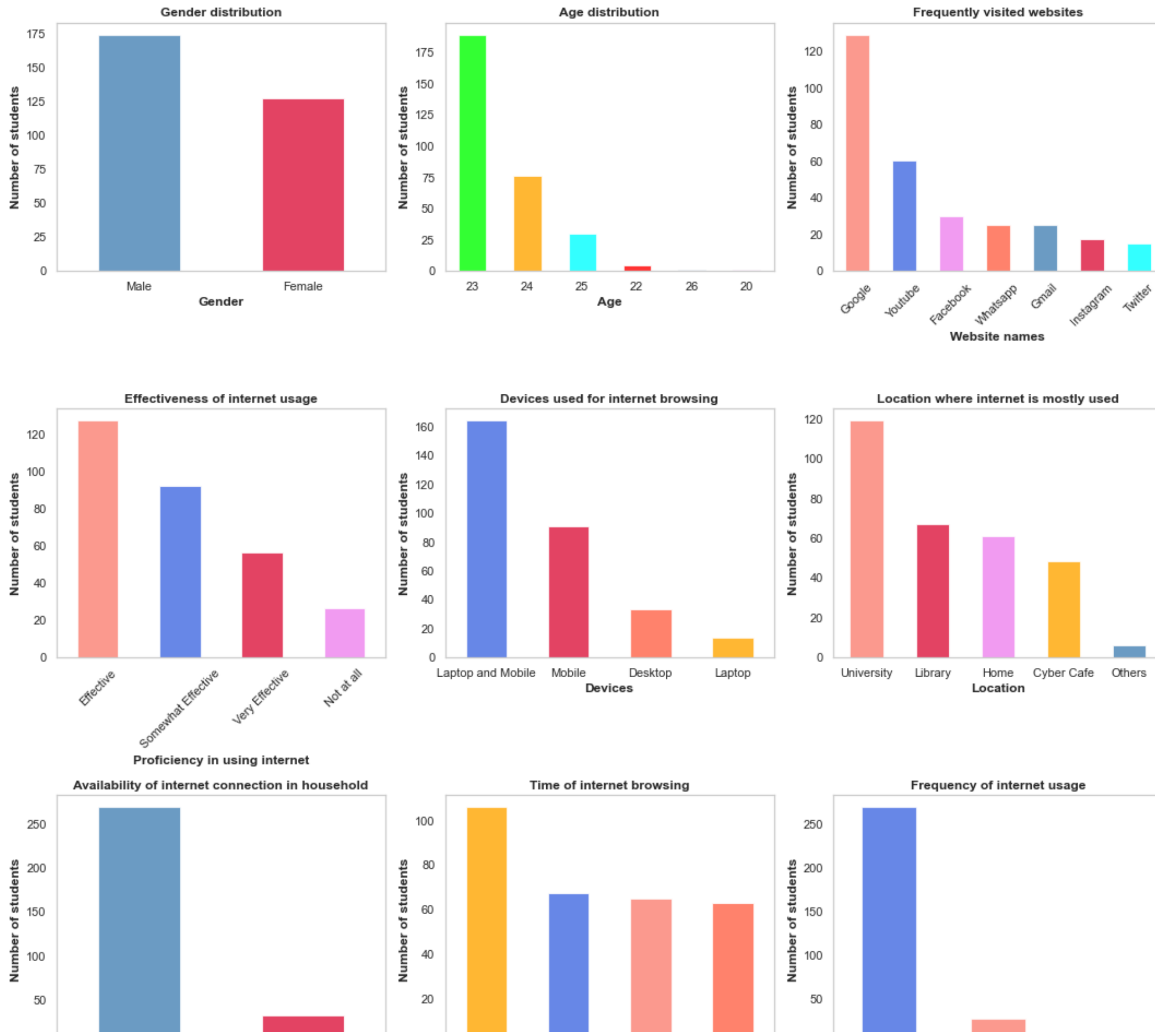
`'Gender'` , `'Age'` , `'Frequently Visited Website'` , `'Effectiveness Of Internet Usage'` , `'Devices Used For Internet Browsing'` , `'Location Of Internet Use'` , `'Household Internet Facilities'` , `'Time Of Internet Browsing'` , `'Frequency Of Internet Usage'` , `'Place Of Student's Residence'` , `'Purpose Of Internet Use'` , `'Browsing Purpose'` , `'Webinar'` , `'Priority Of Learning On The Internet'` , 'Academic Performance' , 'Barriers To Internet Access' are the categorical values in the dataset.

Let's plot the bar plot for each of the categorical attributes together.

In [69]:

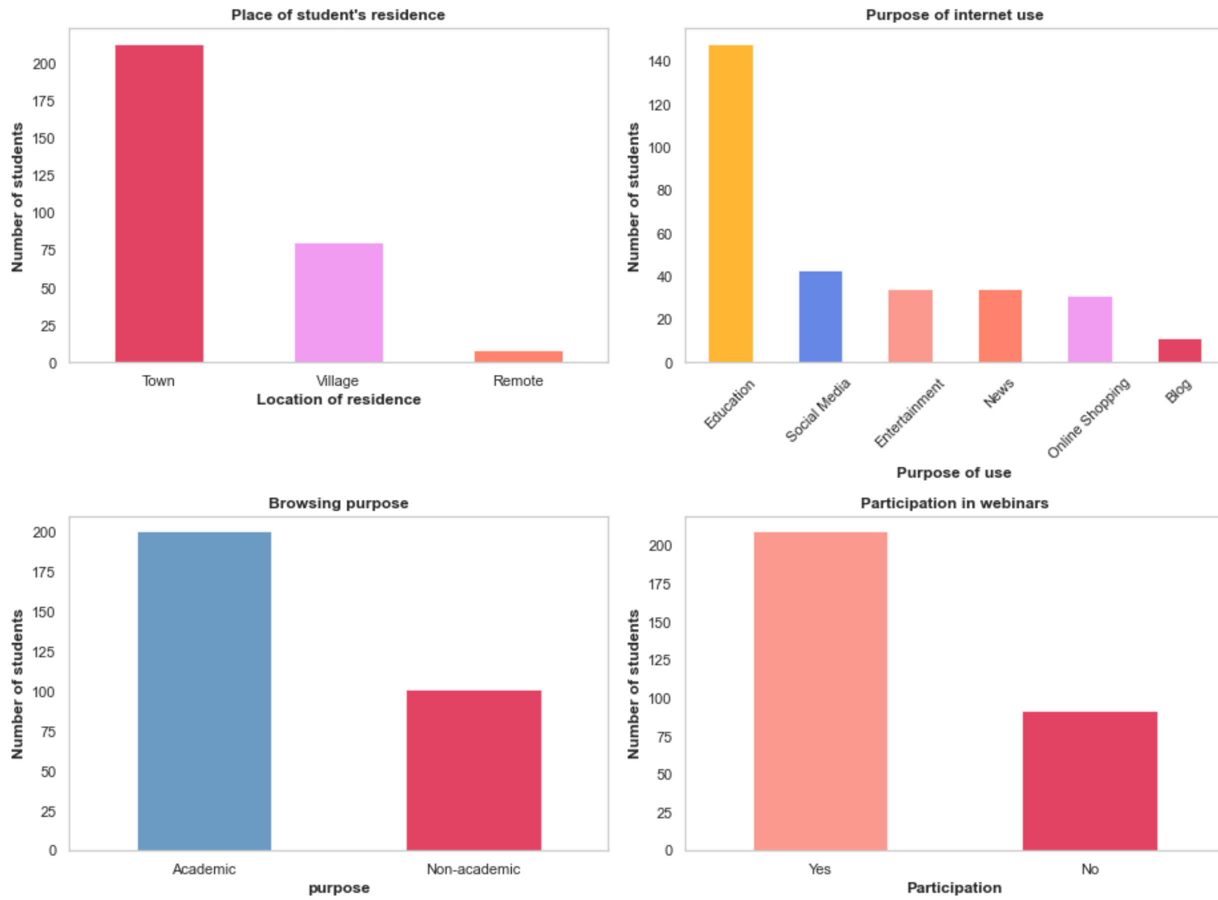
```
plt.figure(figsize=(15, 14))
plt.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)
sns.set(font_scale=1)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.subplot(331)
categorical_bar_plot(university_df['Gender'], title='Gender distribution', xla

plt.subplot(332)
categorical_bar_plot(university_df['Age'],
                     color=['lime', 'orange', 'cyan', 'red', 'steelblue', 'vi
                     title='Age distribution', xlabel='Age')

plt.subplot(333)
categorical_bar_plot(university_df['Frequently Visited Website'], rot=45,
                     color=['salmon', 'royalblue', 'violet', 'tomato', 'steel
                     title='Frequently visited websites', xlabel='Website name

plt.subplot(334)
categorical_bar_plot(university_df['Effectiveness Of Internet Usage'], rot=45,
                     color=['salmon', 'royalblue', 'crimson', 'violet'],
                     title='Effectiveness of internet usage', xlabel='Proficie

plt.subplot(335)
categorical_bar_plot(university_df['Devices Used For Internet Browsing'],
                     color=['royalblue', 'crimson', 'tomato', 'orange'],
                     title='Devices used for internet browsing', xlabel='Devic

plt.subplot(336)
categorical_bar_plot(university_df['Location Of Internet Use'],
                     color=['salmon', 'crimson', 'violet', 'orange', 'steelblu
                     title='Location where internet is mostly used', xlabel='I

plt.subplot(337)
categorical_bar_plot(university_df['Household Internet Facilities'],
                     title='Availability of internet connection in household',
                     xlabel='Household internet facilities')

plt.subplot(338)
categorical_bar_plot(university_df['Time Of Internet Browsing'], color=['orang
                     title='Time of internet browsing', xlabel='Browsing time

plt.subplot(339)
categorical_bar_plot(university_df['Frequency Of Internet Usage'], color=['roy
                     title='Frequency of internet usage', xlabel='Browsing sta


save_fig('Bar_plot_collage_1')

plt.show()
```

```
Saving figure Bar_plot_collage_1
```


Gender distribution
Number of students
Male
Female
Gender
Age distribution
Number of students
23
24
25
22
26
20
Age
Frequently visited websites
Number of students
Google
Youtube
Facebook
Whatsapp
Gmail
Instagram
Twitter
Website names
Effectiveness of internet usage
Number of students
Effective
Somewhat Effective
Very Effective
Not at all
Proficiency in using internet
Devices used for internet browsing
Number of students
Laptop and Mobile
Mobile
Desktop
Laptop
Devices
Location where internet is mostly used
Number of students
University
Library
Home
Cyber Cafe
Others
Location
Availability of internet connection in household
Number of students
Time of internet browsing
Number of students
Frequency of internet usage
Number of students

In [70]:
```
plt.figure(figsize=(13, 18))
plt.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)
sns.set(font_scale=1)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.subplot(321)
categorical_bar_plot(university_df['Place Of Student\'s Residence'], color=['c
                     title='Place of student\'s residence', xlabel='Location o

plt.subplot(322)
categorical_bar_plot(university_df['Purpose Of Internet Use'], rot=45,
                     color = ['orange', 'royalblue', 'salmon', 'tomato', 'viol
                     title='Purpose of internet use', xlabel='Purpose of use')

plt.subplot(323)
categorical_bar_plot(university_df['Browsing Purpose'], title='Browsing purpos
                     xlabel='purpose')

plt.subplot(324)
categorical_bar_plot(university_df['Webinar'], color=['salmon', 'crimson'],
                     title='Participation in webinars', xlabel='Participation

plt.subplot(325)
categorical_bar_plot(university_df['Priority Of Learning On The Internet'], ro
                     color = ['orange', 'royalblue', 'salmon', 'steelblue', 'v
                     title='Priority of learning on the internet', xlabel='Pri

plt.subplot(326)
categorical_bar_plot(university_df['Internet Usage For Educational Purpose'],
                     color=['orange', 'royalblue', 'salmon', 'steelblue', 'vio
                     title='Different reasons for internet browsing for educat
                     xlabel='Internet usage for educational purpose')

save_fig('Bar_plot_collage_2')
plt.show()
```

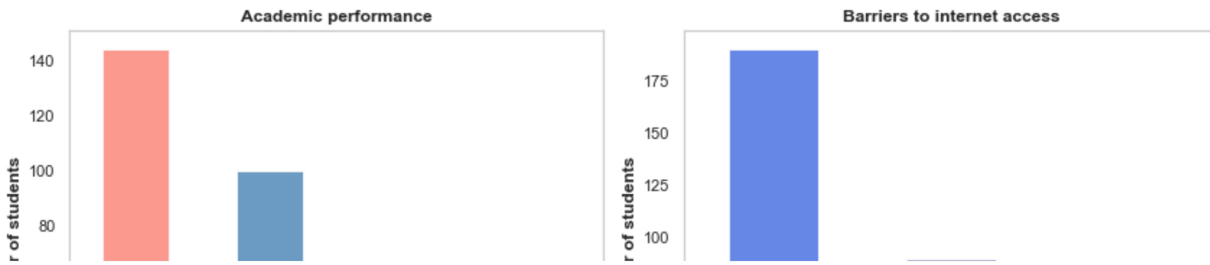
```
Saving figure Bar_plot_collage_2
```



In [71]:

```
plt.figure(figsize=(12, 5))
plt.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)
sns.set(font_scale=1)
sns.set_style("whitegrid", {'axes.grid' : False})


plt.subplot(121)
categorical_bar_plot(university_df['Academic Performance'], color=['salmon',
                     title='Academic performance', xlabel='Performance')

plt.subplot(122)
categorical_bar_plot(university_df['Barriers To Internet Access'],
                     color=['royalblue', 'darkslateblue', 'coral', 'crimson'],
                     title='Barriers to internet access', xlabel='Obstacles')


save_fig('Bar_plot_collage_3')
plt.show()
```

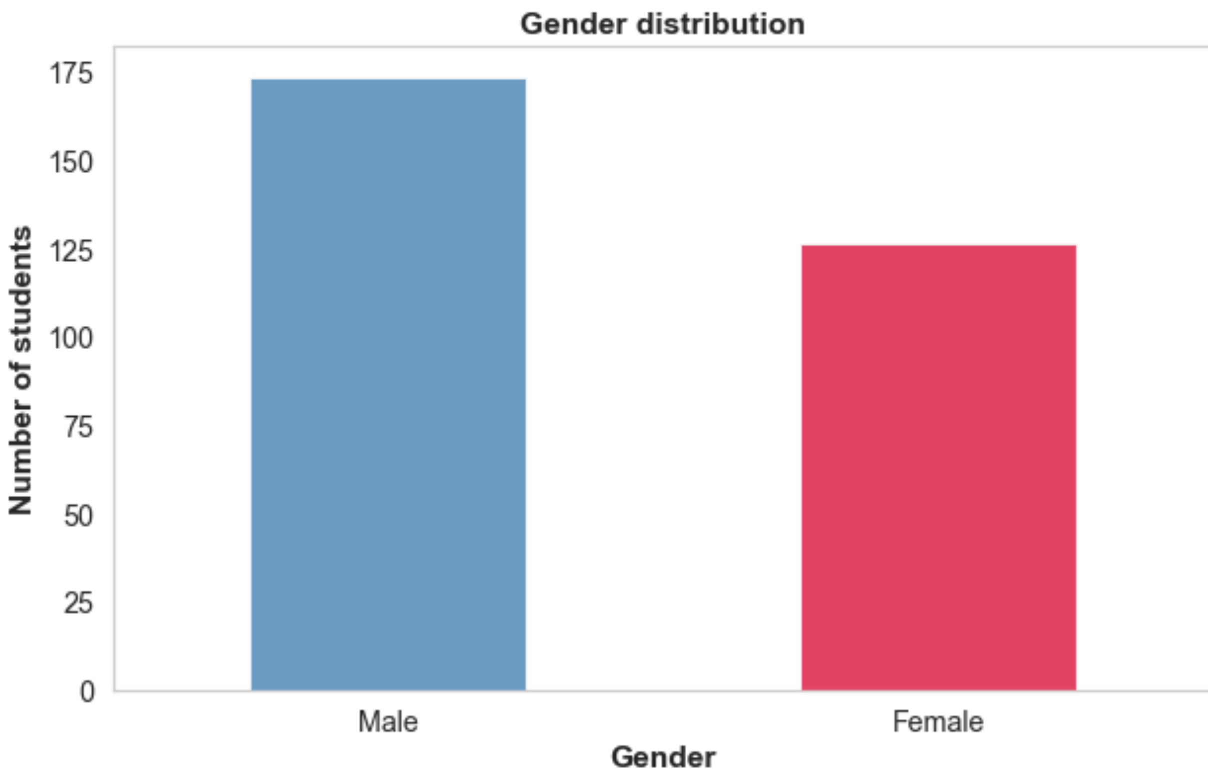
```
Saving figure Bar_plot_collage_3
```



Plotting 'Gender'

Let's check the histogram.

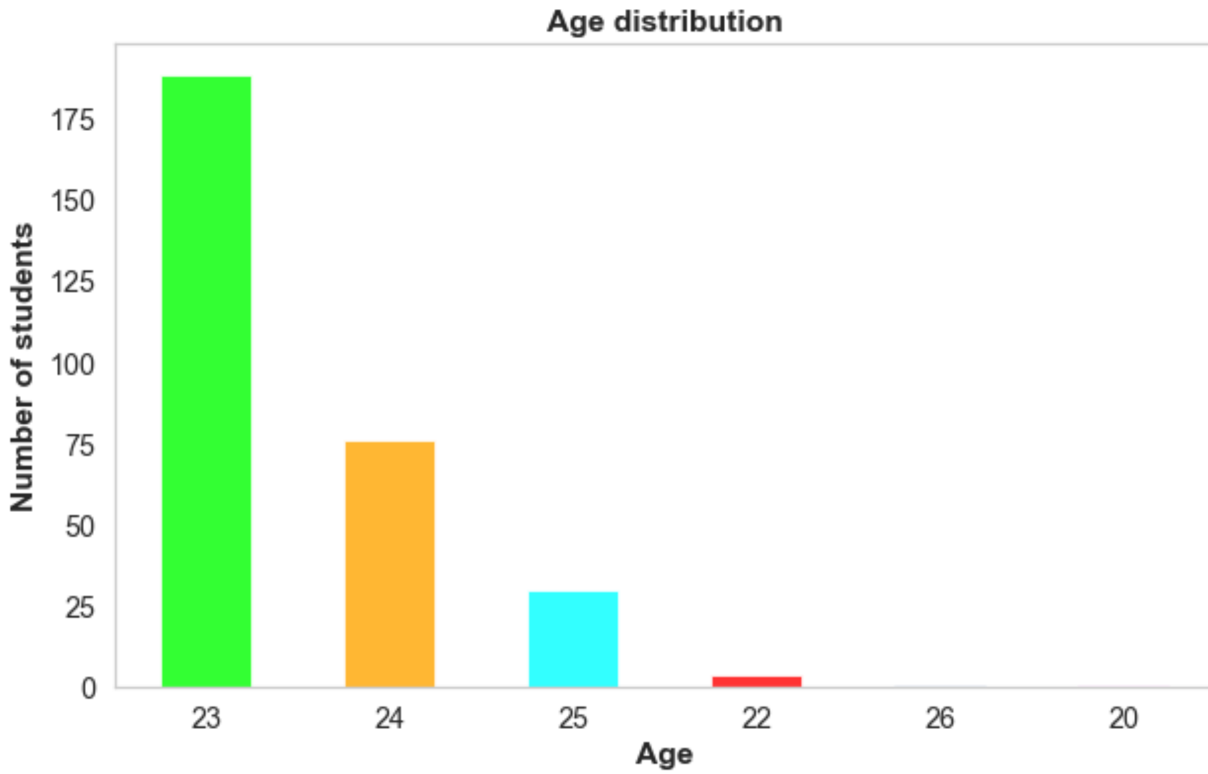
In [72]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Gender'], title='Gender distribution', xla

plt.show()
```



Plotting 'Age'

Let's check the histogram.

In [73]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Age'],
                     color=['lime', 'orange', 'cyan', 'red', 'steelblue', 'vio
                     title='Age distribution', xlabel='Age')

plt.show()
```



Plotting Frequently Visited Website'

Let's check the histogram.

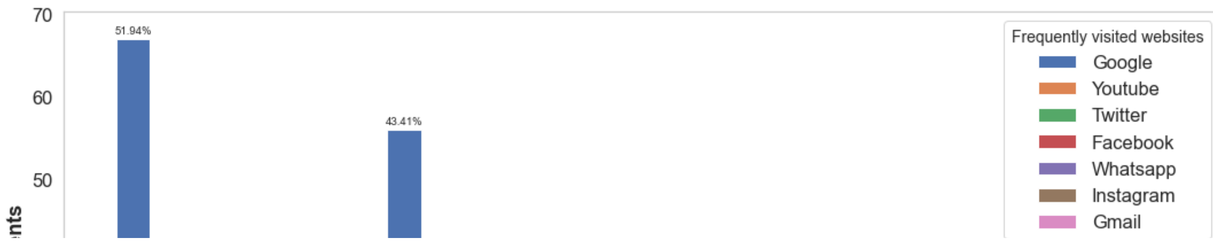
In [74]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Frequently Visited Website'], rot=45,
                     color=['salmon', 'royalblue', 'violet', 'tomato', 'steelb
                     title='Frequently visited websites', xlabel='Website name

plt.show()
```

Let's check the distribution of this feature against the target i.e. `'Academic Performance'`.

In [75]:

```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Frequently Visited Website',
                              university_df['Frequently Visited Website'].valu

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - (width + 0.13), dictionary['Google'], width/2, label = 'Go
rects2 = ax.bar(x - width, dictionary['Youtube'], width/2, label = 'Youtube')
rects3 = ax.bar(x - width/2, dictionary['Twitter'], width/2, label = 'Twitter
rects4 = ax.bar(x, dictionary['Facebook'], width/2, label = 'Facebook')
rects5 = ax.bar(x + width/2, dictionary['Whatsapp'], width/2, label = 'Whatsap
rects6 = ax.bar(x + width, dictionary['Instagram'], width/2, label = 'Instagra
rects7 = ax.bar(x + (width + 0.13), dictionary['Gmail'], width/2, label = 'Gma

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Frequently Visited Websites W.R.T. Academic Performance', font
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Frequently visited websites', title_fontsize=14)

sns.set(font_scale=0.8)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)
autolabel(rects6)
autolabel(rects7)

fig.tight_layout()

save_fig('Frequently_Visited_Websites_WRT_Academic_Performance_Frequency_Dist

plt.show()
```

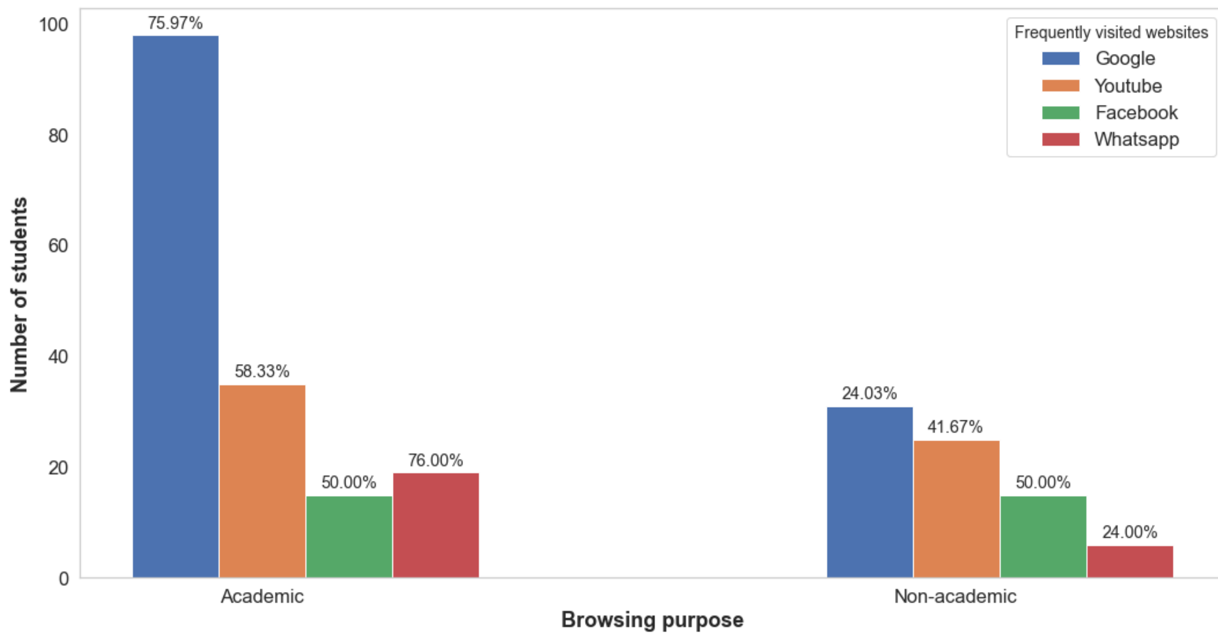
```
Saving figure Frequently_Visited_Websites_WRT_Academic_Performance_Frequency_D
istribution
```



Let's check the distribution of this feature against the target i.e. `'Browsing Purpose'` **.**

In [76]:

```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot_browsing_purpose(university_df, 'Frequently V
                              university_df['Frequently Visited Website'].valu

labels = ['Academic', 'Non-academic']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - width, dictionary['Google'], width/2, label = 'Google')
rects2 = ax.bar(x - width/2, dictionary['Youtube'], width/2, label = 'Youtube
rects3 = ax.bar(x, dictionary['Facebook'], width/2, label = 'Facebook')
rects4 = ax.bar(x + width/2, dictionary['Whatsapp'], width/2, label = 'Whatsap

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Browsing purpose', fontweight = 'bold')
# ax.set_title('Frequently Visited Websites vs Browsing Purpose', fontweight =
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Frequently visited websites', title_fontsize=14 ,loc='upper r

sns.set(font_scale=1.2)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)

fig.tight_layout()

plt.show()
```

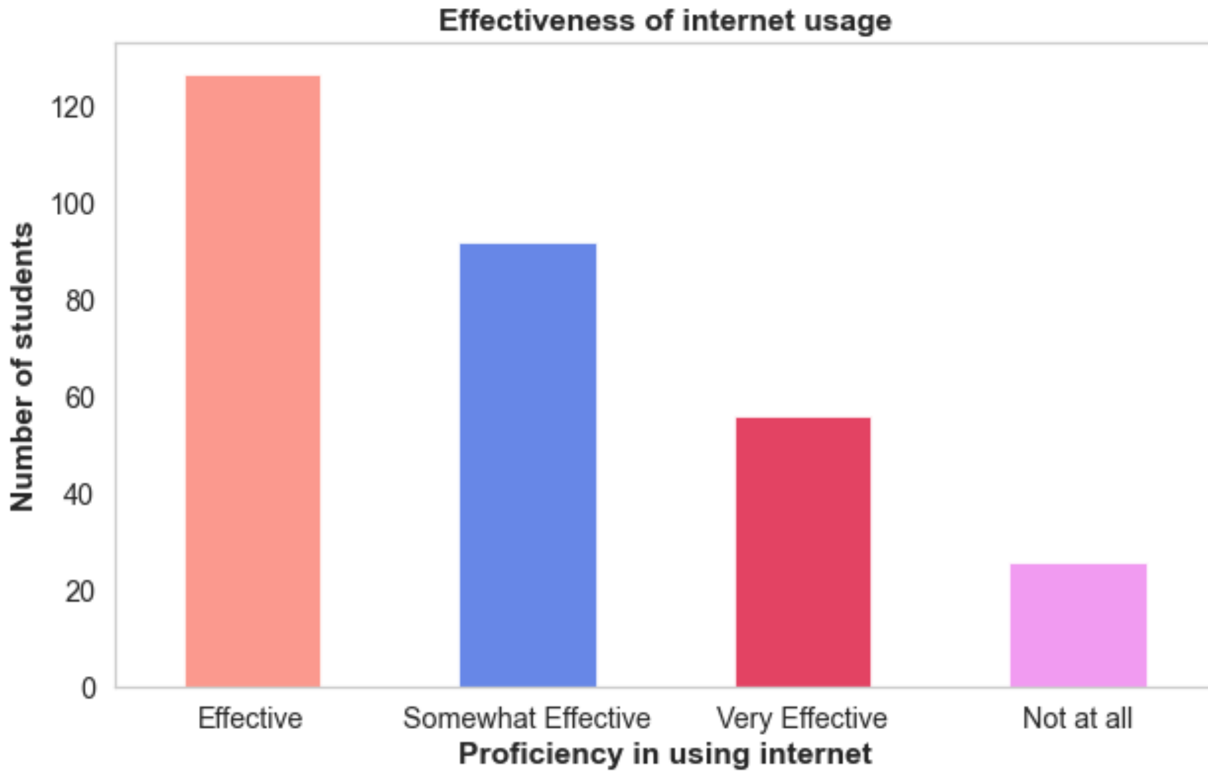


Plotting 'Effectiveness Of Internet Usage'

Let's check the histogram.

In [77]:

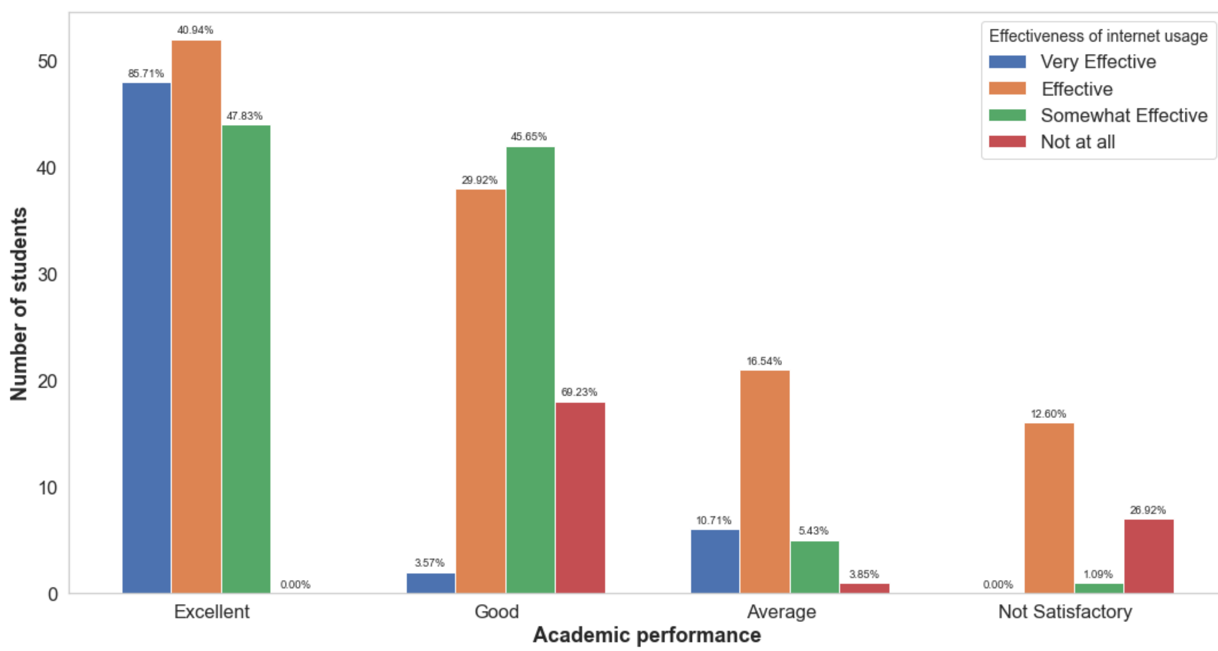
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Effectiveness Of Internet Usage'],
                     color=['salmon', 'royalblue', 'crimson', 'violet'],
                     title='Effectiveness of internet usage', xlabel='Proficie

plt.show()
```



Let's check the distribution of this feature against the target i.e. `'Academic Performance'` .

In [78]:

```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Effectiveness Of Internet Usa
                                  ['Very Effective', 'Effective', 'Somewhat Effect

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.35

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - width, dictionary['Very Effective'], width/2, label = 'Ve
rects2 = ax.bar(x - width/2, dictionary['Effective'], width/2, label = 'Effect
rects3 = ax.bar(x, dictionary['Somewhat Effective'], width/2, label = 'Somewha
rects4 = ax.bar(x + width/2, dictionary['Not at all'], width/2, label = 'Not a

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Effectiveness Of Internet Usage vs Academic Performance', font
ax.set_xticks(x - width/3)
ax.set_xticklabels(labels)
ax.legend(title='Effectiveness of internet usage', title_fontsize=14)

sns.set(font_scale=0.8)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)

fig.tight_layout()

plt.show()
```

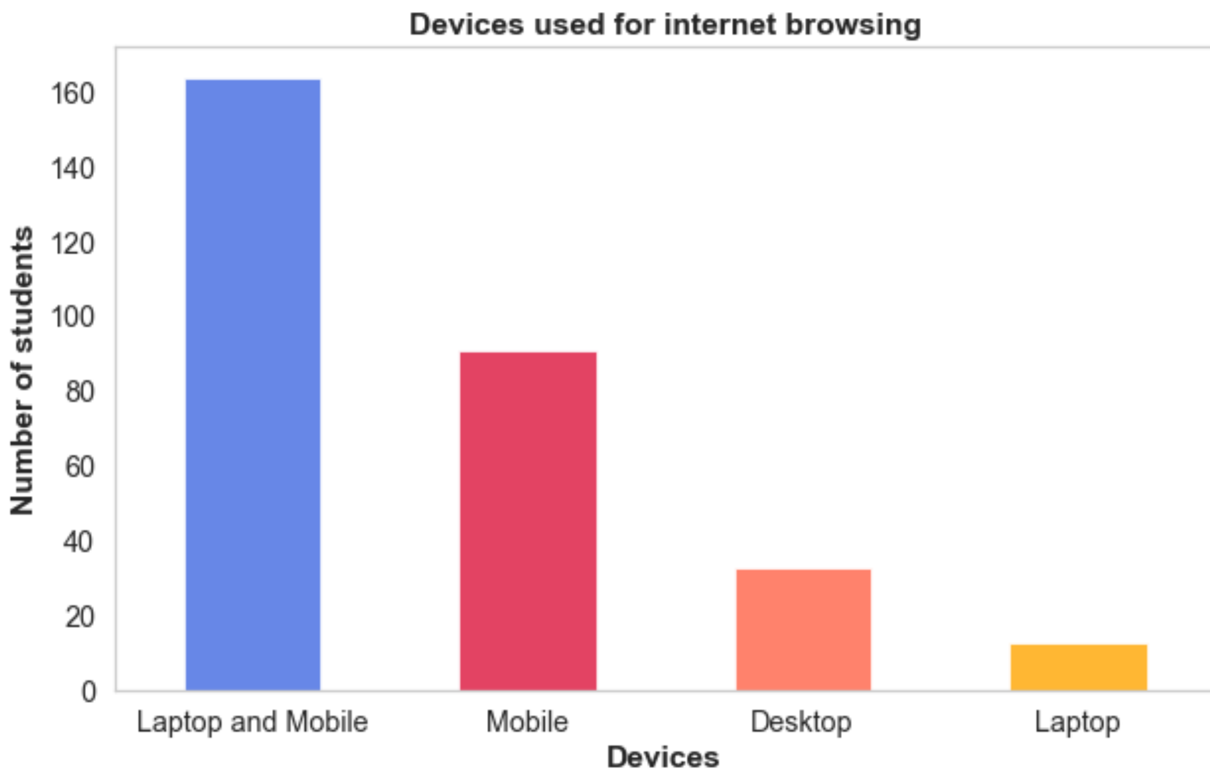


Plotting 'Devices Used For Internet Browsing'

Let's check the histogram.

In [79]:

```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Devices Used For Internet Browsing'],
                     color=['royalblue', 'crimson', 'tomato', 'orange'],
                     title='Devices used for internet browsing', xlabel='Devic

plt.show()
```

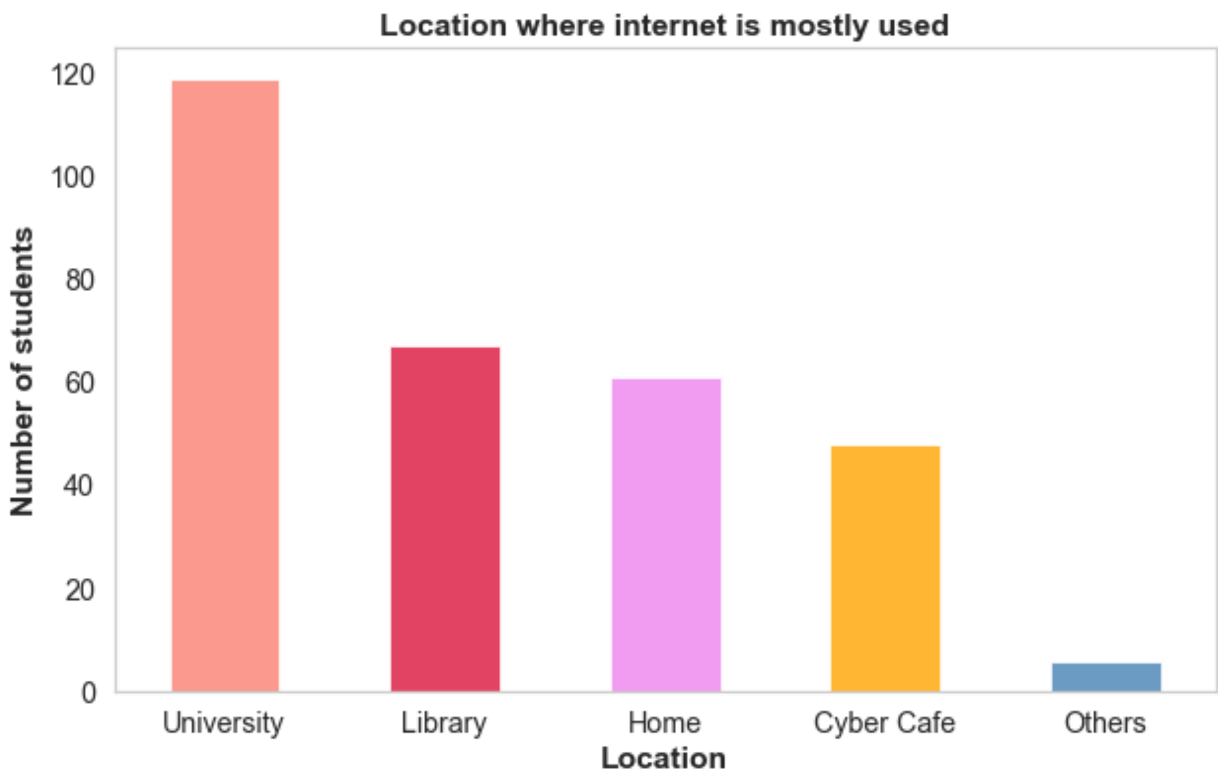


Plotting 'Location Of Internet Use'

Let's check the histogram.

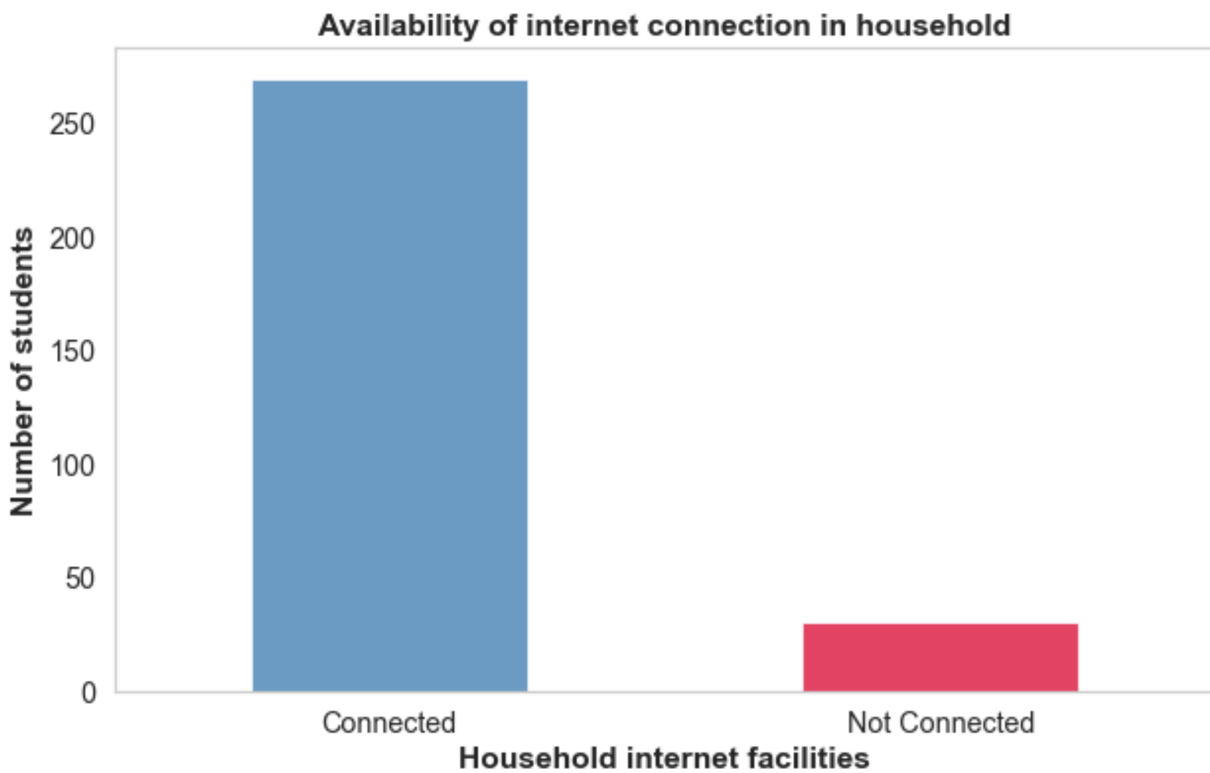
In [80]:

```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Location Of Internet Use'],
                     color=['salmon', 'crimson', 'violet', 'orange', 'steelblu
                     title='Location where internet is mostly used', xlabel='I

plt.show()
```



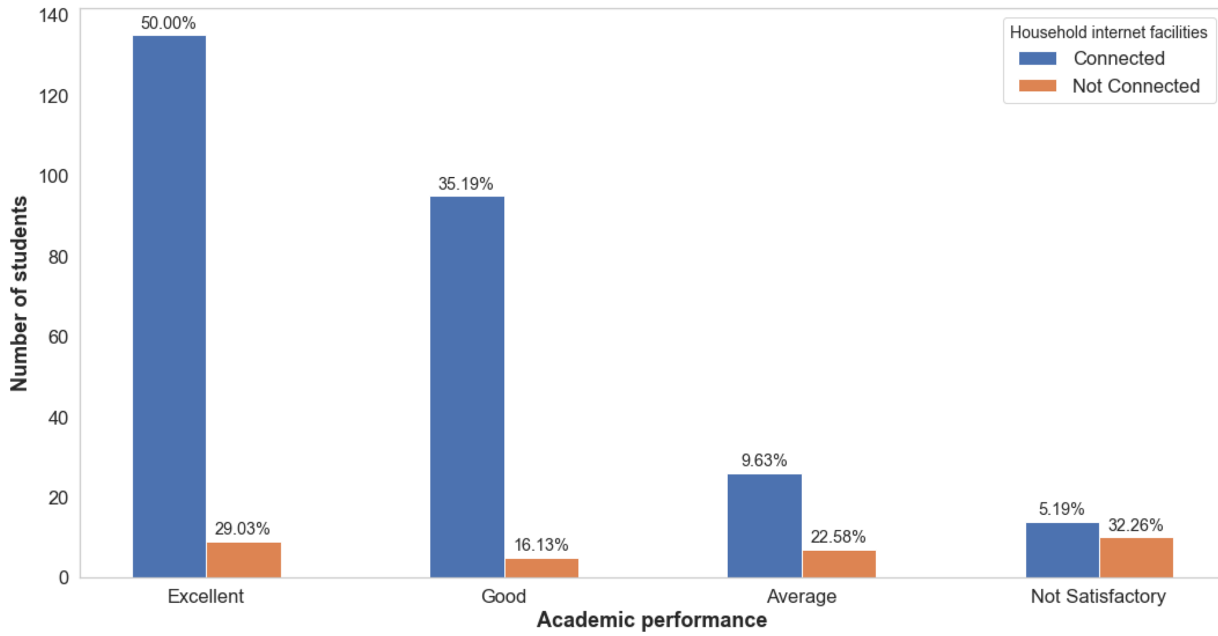
Plotting 'Household Internet Facilities'

In [81]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Household Internet Facilities'],
                     title='Availability of internet connection in household',
                     xlabel='Household internet facilities')

plt.show()
```



Let's check the distribution of this feature against the target i.e. `'Academic Performance'` .

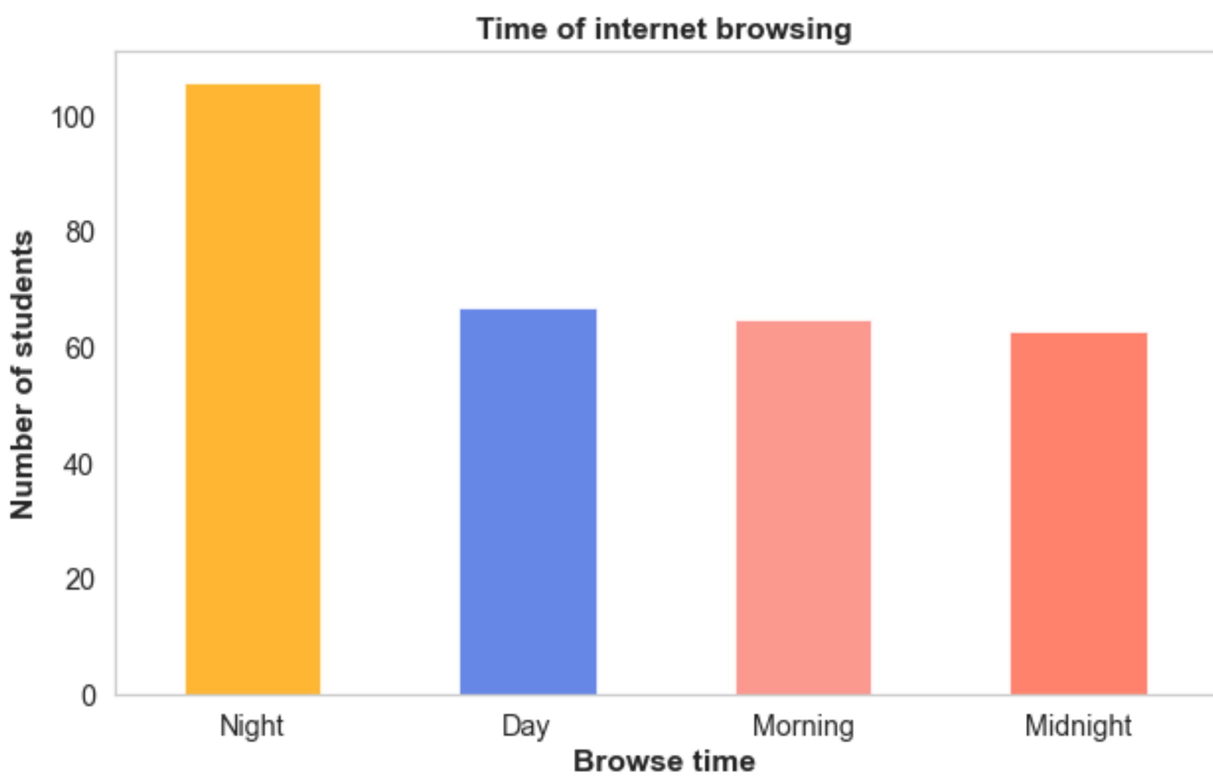
In [82]:
```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Household Internet Facilities
                              university_df['Household Internet Facilities'].v

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - width, dictionary['Connected'], width, label = 'Connected
rects2 = ax.bar(x, dictionary['Not Connected'], width, label = 'Not Connected

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Availability Of Internet Connection In Household vs Academic
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Household internet facilities', title_fontsize=14)

sns.set(font_scale=1.2)

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

plt.show()
```



Plotting 'Time Of Internet Browsing'

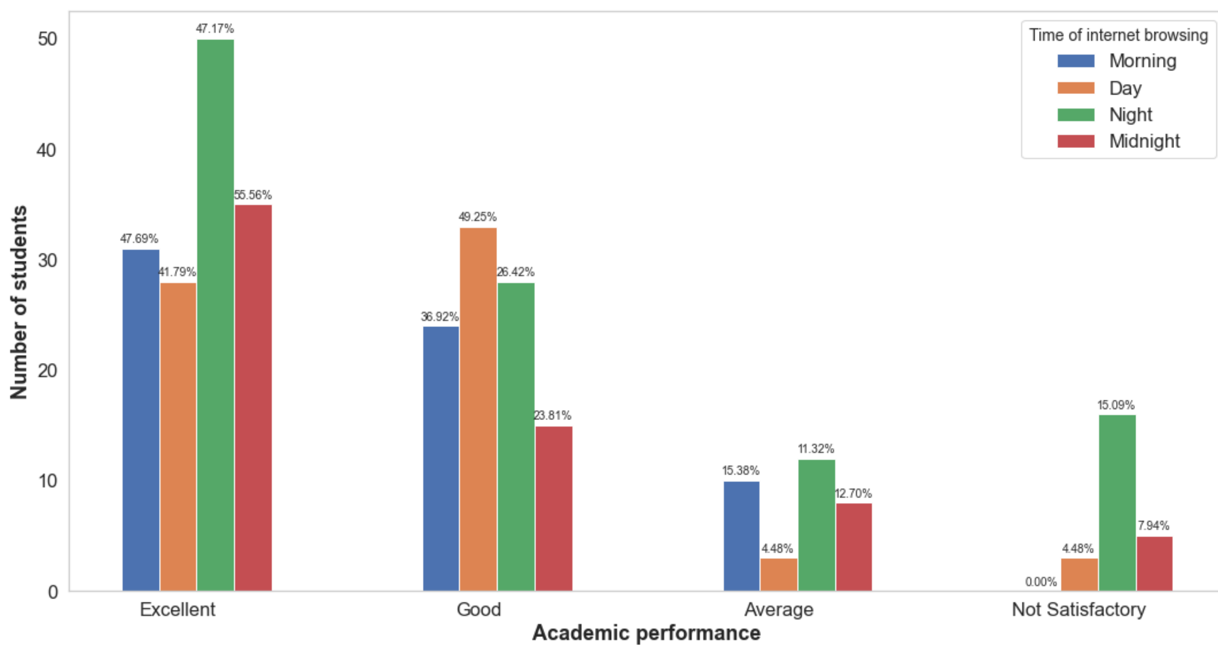
Let's check the histogram.

In [83]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Time Of Internet Browsing'], color=['oranc
                     title='Time of internet browsing', xlabel='Browse time')

plt.show()
```



Let's check the distribution of this feature against the target i.e. `'Academic Performance'` .

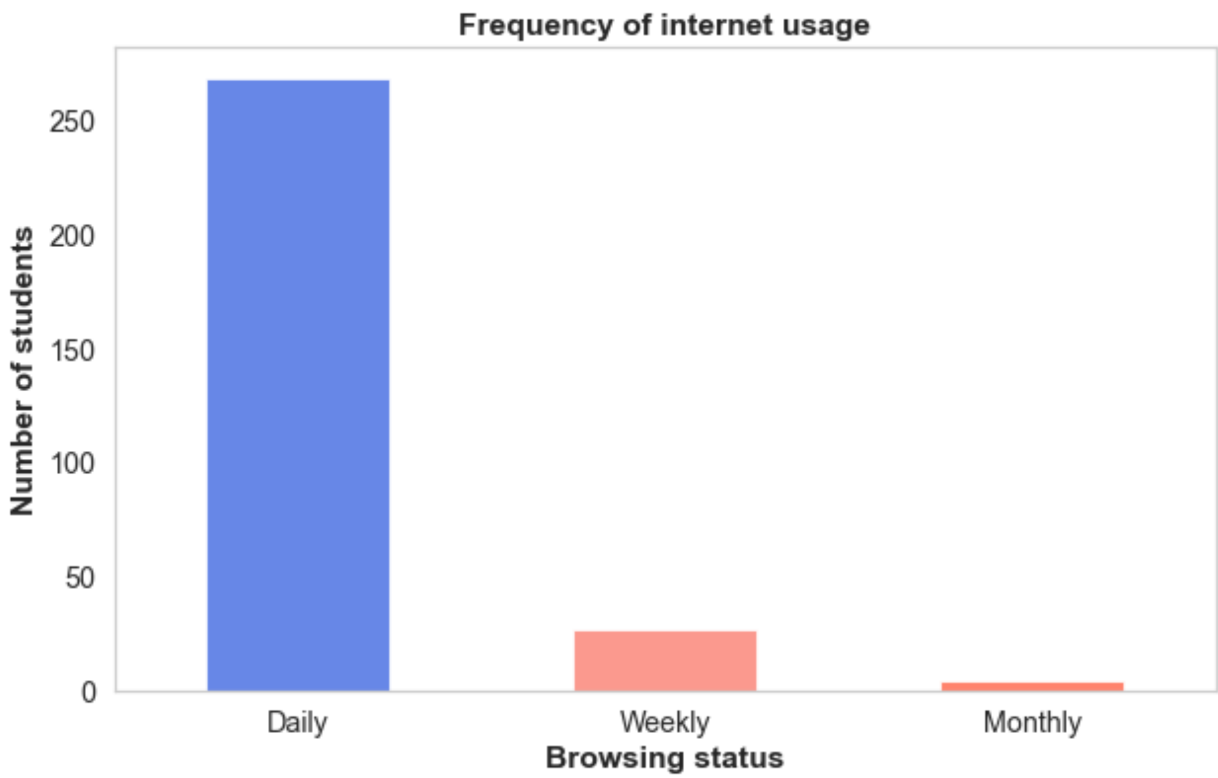
In [84]:
```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Time Of Internet Browsing',
                                  ['Morning', 'Day', 'Night', 'Midnight'])

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - width, dictionary['Morning'], width/2, label = 'Morning')
rects2 = ax.bar(x - width/2, dictionary['Day'], width/2, label = 'Day')
rects3 = ax.bar(x, dictionary['Night'], width/2, label = 'Night')
rects4 = ax.bar(x + width/2, dictionary['Midnight'], width/2, label = 'Midnigh

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Time Of Internet Browsing vs Academic Performance', fontweigh
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Time of internet browsing', title_fontsize=14)

sns.set(font_scale=0.85)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)

fig.tight_layout()

plt.show()
```



Plotting `'Frequency Of Internet Usage'`

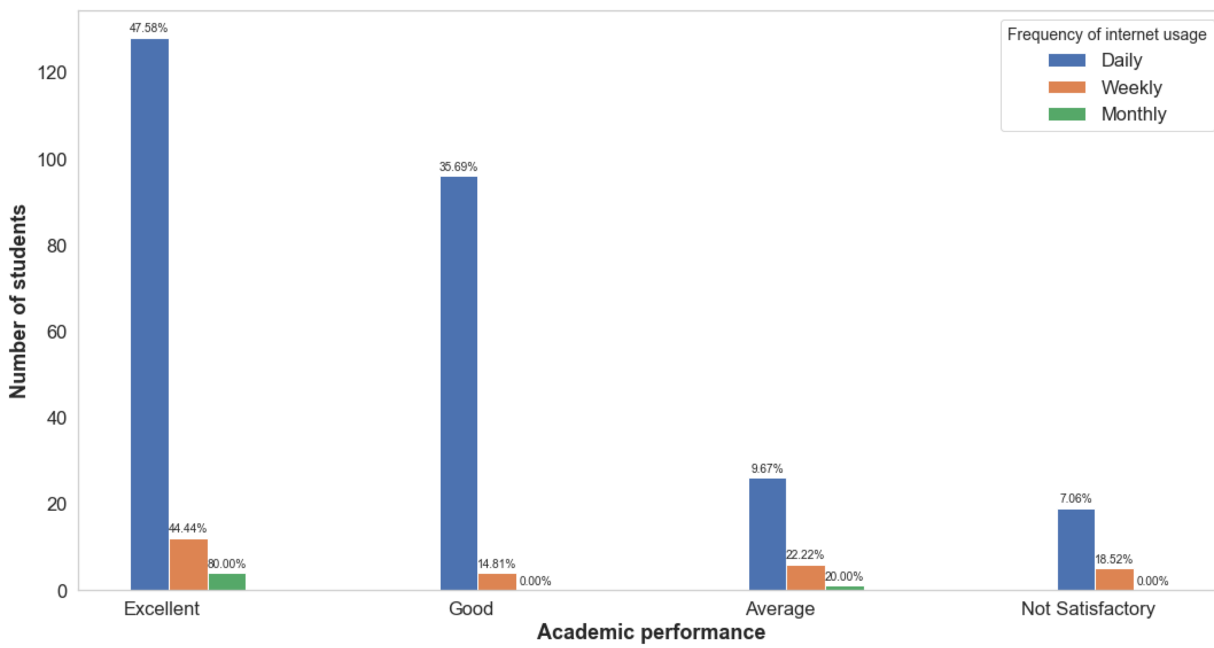
Let's check the histogram.

In [85]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Frequency Of Internet Usage'], color=['ro
                     title='Frequency of internet usage', xlabel='Browsing st

plt.show()
```



Let's check the distribution of this feature against the target i.e. `'Academic Performance'` .

In [86]:
```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Frequency Of Internet Usage',
                              ['Daily', 'Weekly', 'Monthly'])

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - width/2, dictionary['Daily'], width/2, label = 'Daily')
rects2 = ax.bar(x, dictionary['Weekly'], width/2, label = 'Weekly')
rects3 = ax.bar(x + width/2, dictionary['Monthly'], width/2, label = 'Monthly

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Frequency Of Internet Usage vs Academic Performance', fontwei
ax.set_xticks(x - width/3)
ax.set_xticklabels(labels)
ax.legend(title='Frequency of internet usage', title_fontsize=14)

sns.set(font_scale=0.85)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)

fig.tight_layout()

plt.show()
```



Plotting 'Place Of Student's Residence'

Let's check the histogram.

In [87]:

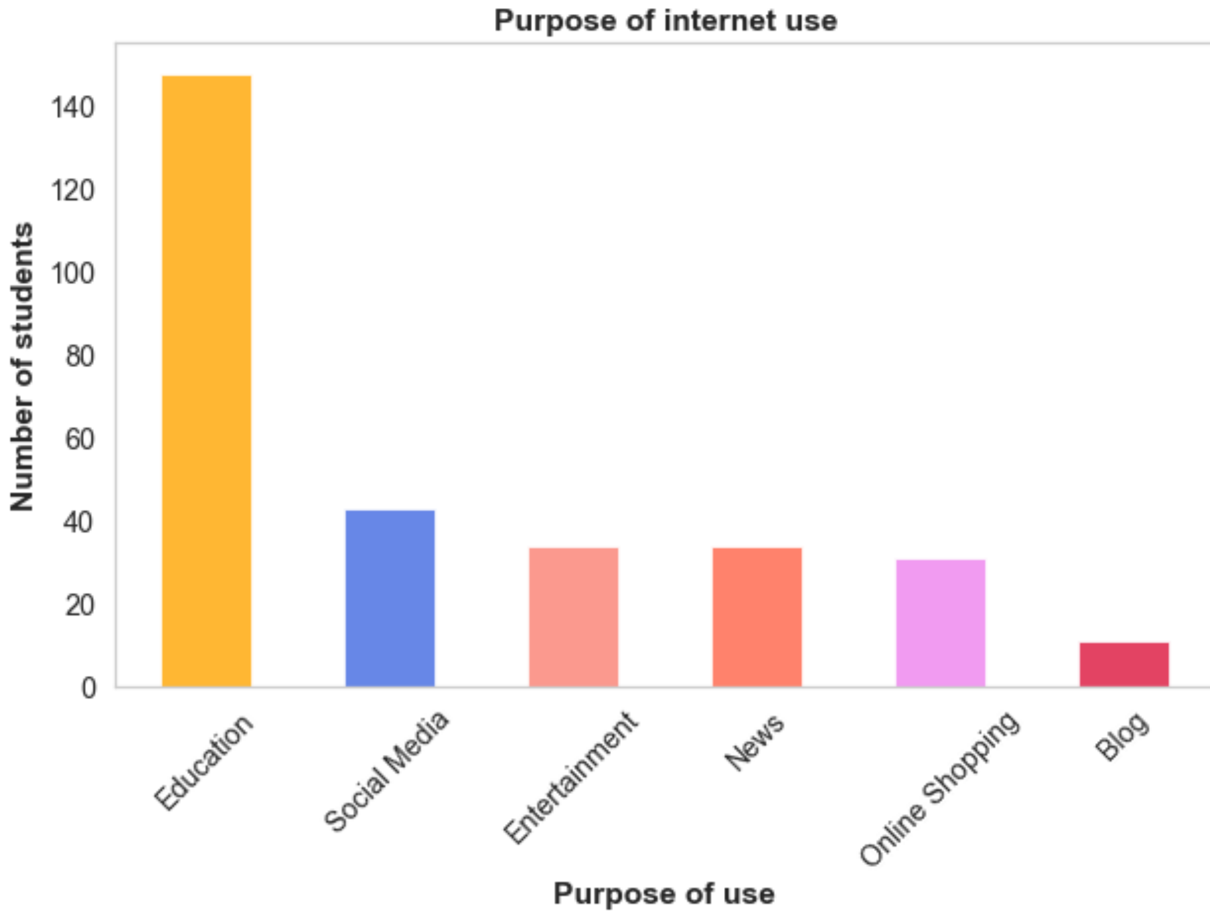
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Place Of Student\'s Residence'], color=['c
                     title='Place of student\'s residence', xlabel='Location c

plt.show()
```

Plotting 'Purpose Of Internet Use'

Let's check the histogram.

In [88]:

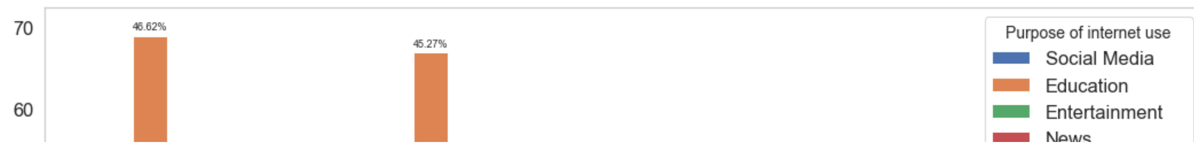
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Purpose Of Internet Use'], rot=45,
                     color = ['orange', 'royalblue', 'salmon', 'tomato', 'vio
                     title='Purpose of internet use', xlabel='Purpose of use')

plt.show()
```



Let's check the distribution of this feature against the target i.e. `'Academic Performance'` .

In [89]:
```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Purpose Of Internet Use',
                                 university_df['Purpose Of Internet Use'].value_c

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - (width + 0.125), dictionary['Social Media'], width/2, labe
rects2 = ax.bar(x - width, dictionary['Education'], width/2, label = 'Educatio
rects3 = ax.bar(x - width/2, dictionary['Entertainment'], width/2, label = 'En
rects4 = ax.bar(x, dictionary['News'], width/2, label = 'News')
rects5 = ax.bar(x + width/2, dictionary['Online Shopping'], width/2, label =
rects6 = ax.bar(x + width, dictionary['Blog'], width/2, label = 'Blog')

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Purpose Of Internet Use W.R.T. Academic Performance', fontwei
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Purpose of internet use', title_fontsize=14)

sns.set(font_scale=0.75)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)
autolabel(rects6)

fig.tight_layout()

save_fig('Purpose_Of_Internet_Use_WRT_Academic_Performance_Frequency_Distribut
plt.show()
```

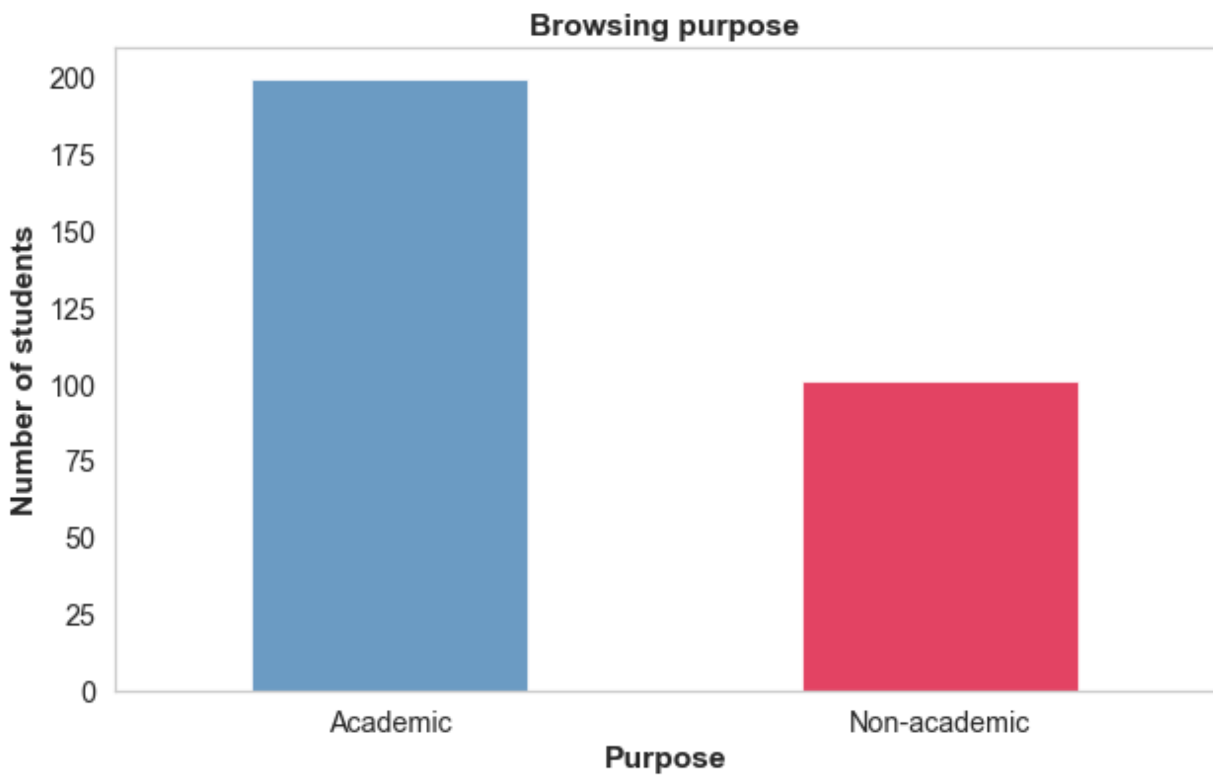
```
Saving figure Purpose_Of_Internet_Use_WRT_Academic_Performance_Frequency_Distr
ibution
```



Plotting 'Browsing Purpose'

Let's check the histogram.

In [90]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Browsing Purpose'], title='Browsing purpos
                     xlabel='Purpose')

plt.show()
```

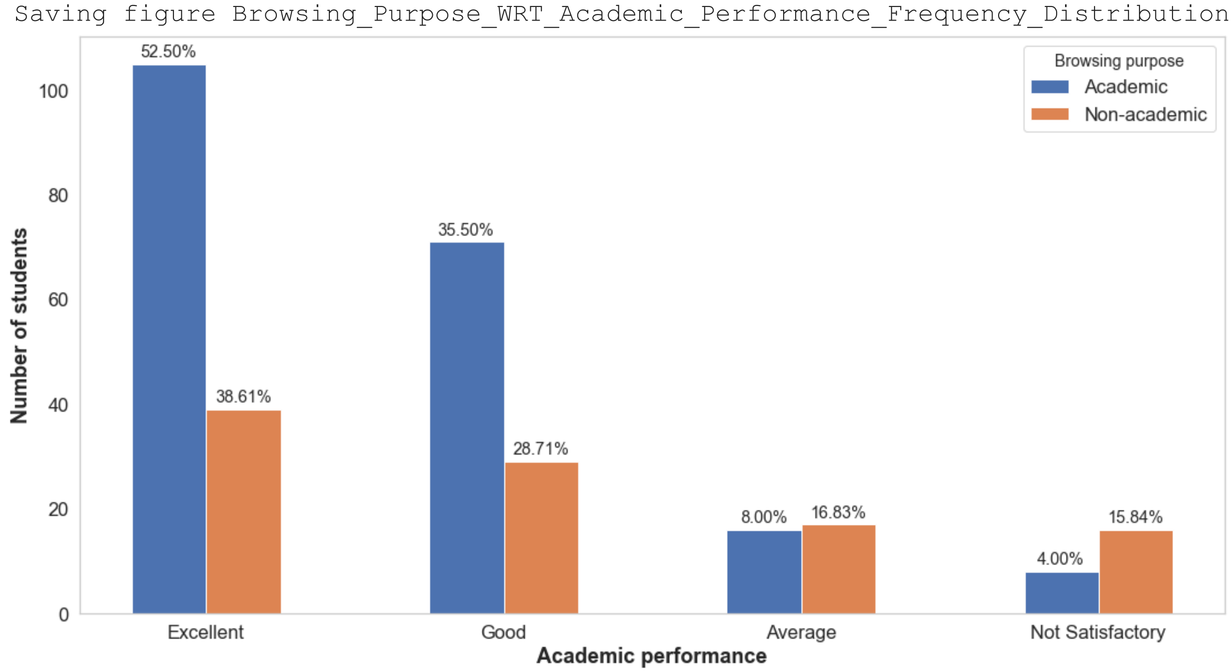


Let's check the distribution of this feature against the target i.e. `'Academic Performance'` .

In [91]:

```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Browsing Purpose',
                              university_df['Browsing Purpose'].value_counts()

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - width, dictionary['Academic'], width, label = 'Academic')
rects2 = ax.bar(x, dictionary['Non-academic'], width, label = 'Non-academic')

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Browsing Purpose W.R.T. Academic Performance', fontweight = 'l
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Browsing purpose', title_fontsize=14, loc='upper right')

sns.set(font_scale=1.2)

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

save_fig('Browsing_Purpose_WRT_Academic_Performance_Frequency_Distribution')

plt.show()
```

```
Saving figure Browsing_Purpose_WRT_Academic_Performance_Frequency_Distribution
```



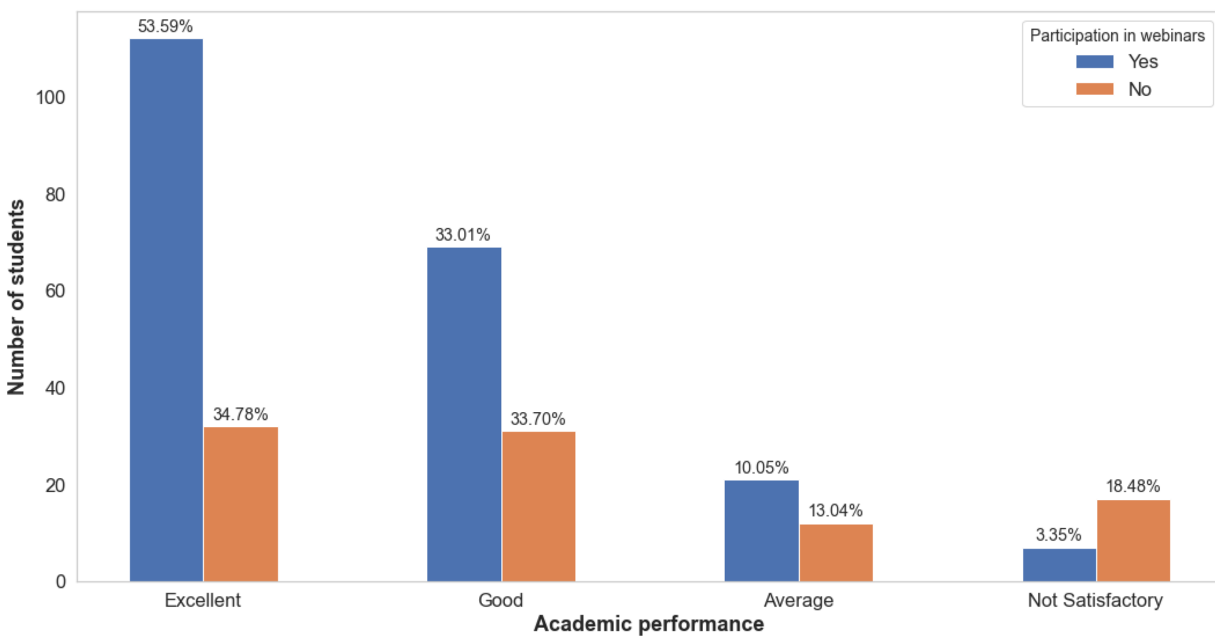
Plotting 'Webinar'

Let's check the histogram.

In [92]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Webinar'], color=['salmon', 'crimson'],
                     title='Participation in webinars', xlabel='Participation

plt.show()
```

Let's check the distribution of this feature against the target i.e. `'Academic Performance'` .

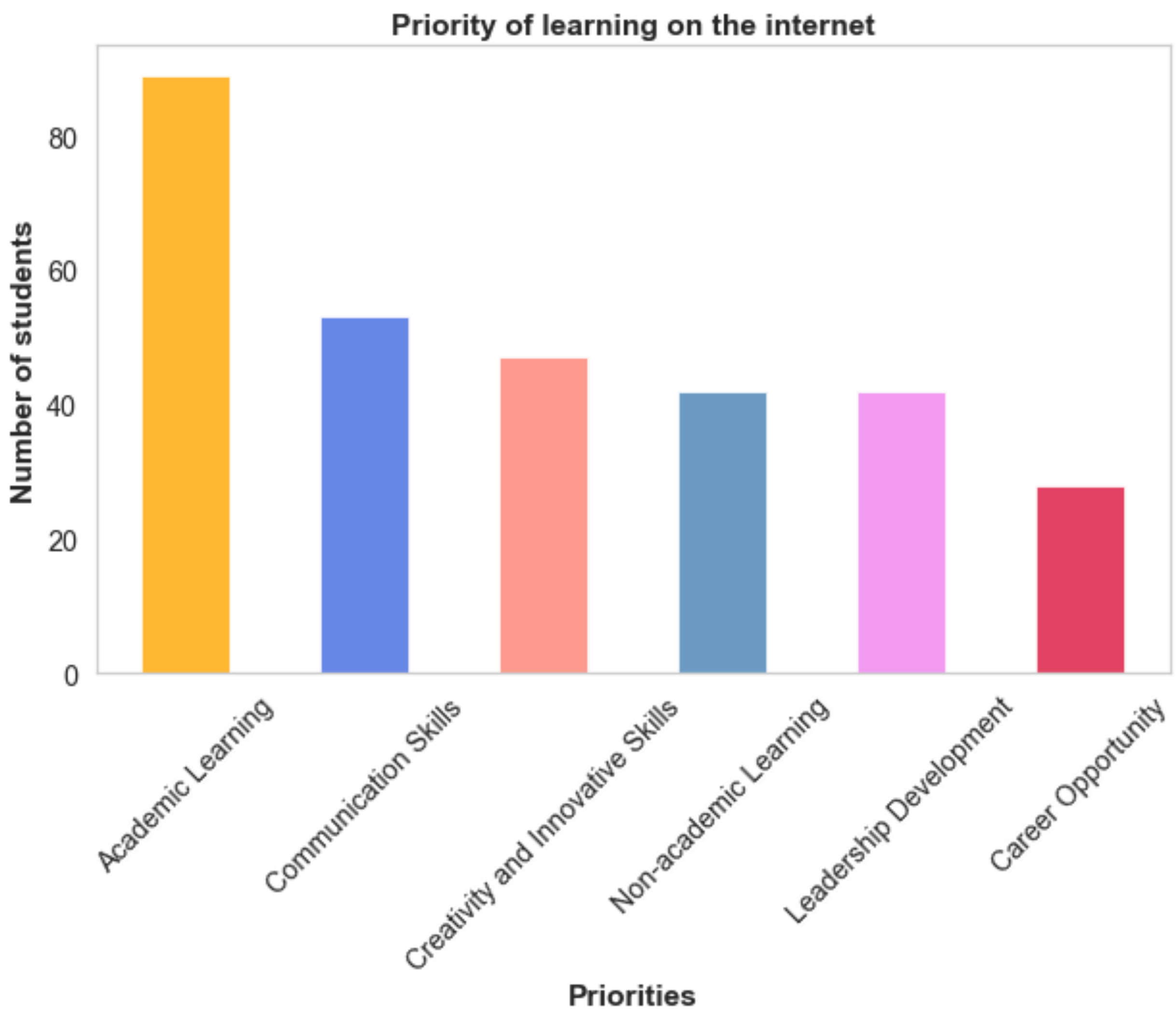
In [93]:
```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Webinar',
                                university_df['Webinar'].value_counts().index.to

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - width, dictionary['Yes'], width, label = 'Yes')
rects2 = ax.bar(x, dictionary['No'], width, label = 'No')

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Participation In Webinars vs Academic Performance', fontweigh
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Participation in webinars', title_fontsize=14, loc='upper ri

sns.set(font_scale=1.2)

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

plt.show()
```



Plotting 'Priority Of Learning On The Internet'

Let's check the histogram.

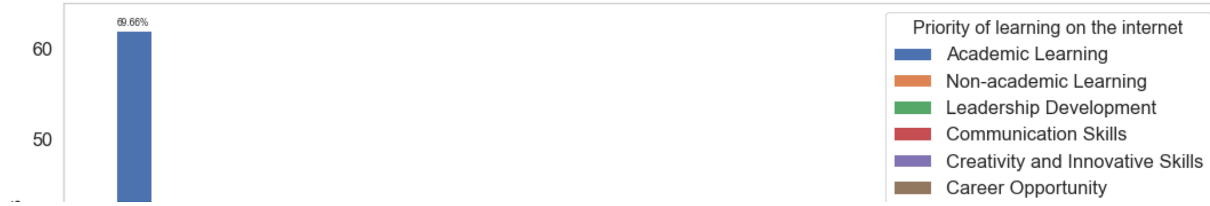
In [94]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Priority Of Learning On The Internet'], ro
                     color = ['orange', 'royalblue', 'salmon', 'steelblue', 'v
                     title='Priority of learning on the internet', xlabel='Pri

plt.show()
```



Let's check the distribution of this feature against the target i.e. `'Academic Performance'` .

In [95]:
```
sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Priority Of Learning On The I
                              ['Academic Learning', 'Non-academic Learning',
                               'Communication Skills', 'Creativity and Innovat

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - (width + 0.12), dictionary['Academic Learning'], width/2,
rects2 = ax.bar(x - width, dictionary['Non-academic Learning'], width/2, label
rects3 = ax.bar(x - width/2, dictionary['Leadership Development'], width/2, la
rects4 = ax.bar(x, dictionary['Communication Skills'], width/2, label = 'Commu
rects5 = ax.bar(x + width/2, dictionary['Creativity and Innovative Skills'], w
                label = 'Creativity and Innovative Skills')
rects6 = ax.bar(x + width, dictionary['Career Opportunity'], width/2, label =

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Priority Of Learning On The Internet W.R.T. Academic Performa
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Priority of learning on the internet', title_fontsize=16, loc

sns.set(font_scale=0.7)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)
autolabel(rects6)

fig.tight_layout()

save_fig('Priority_Of_Learning_On_The_Internet_W.R.T._Academic_Performance_Fre
plt.show()
```

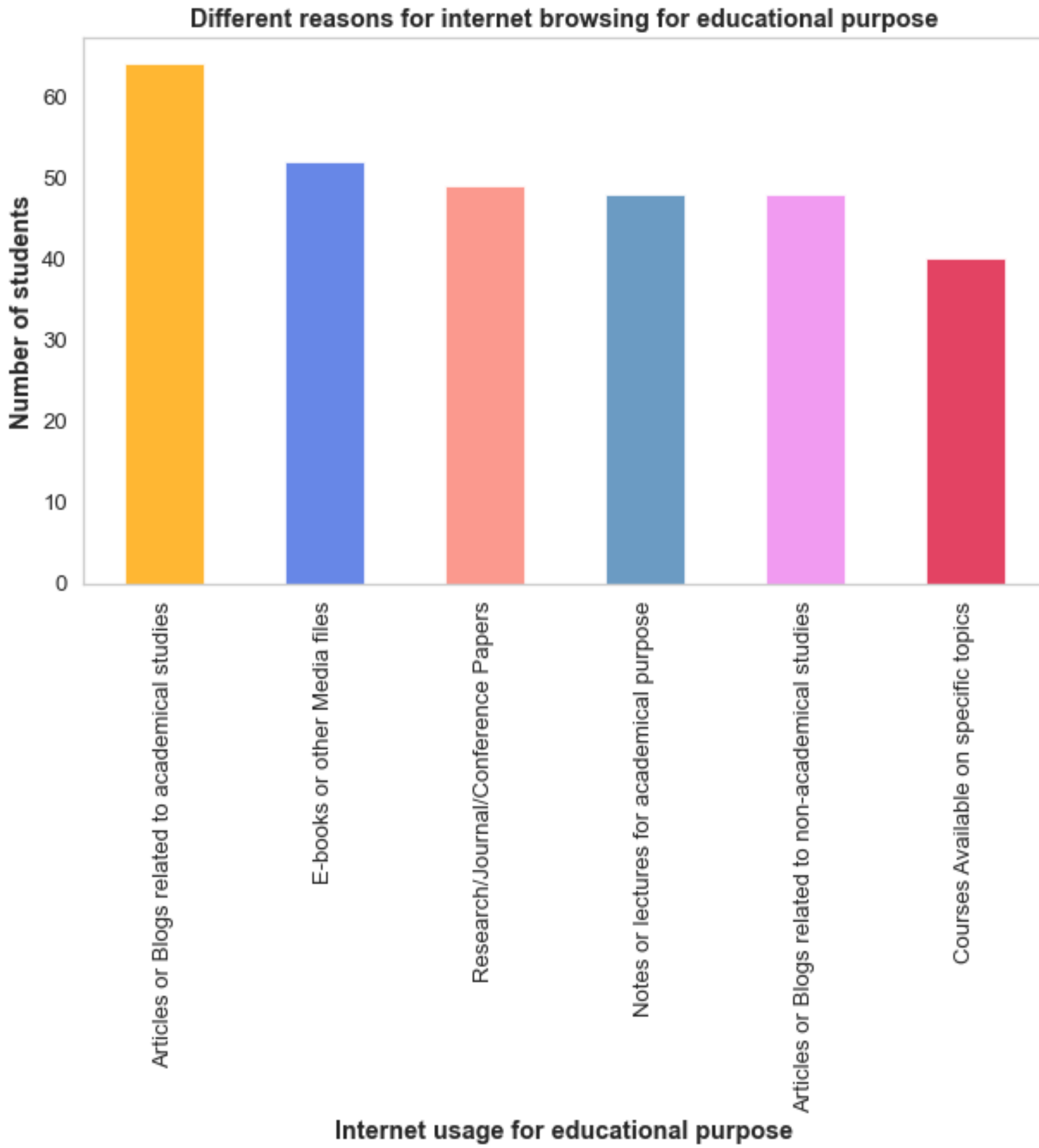
```
Saving figure Priority_Of_Learning_On_The_Internet_W.R.T._Academic_Performance
_Frequency_Distribution
```



Plotting `'Internet Usage For Educational Purpose'`

Let's check the histogram.

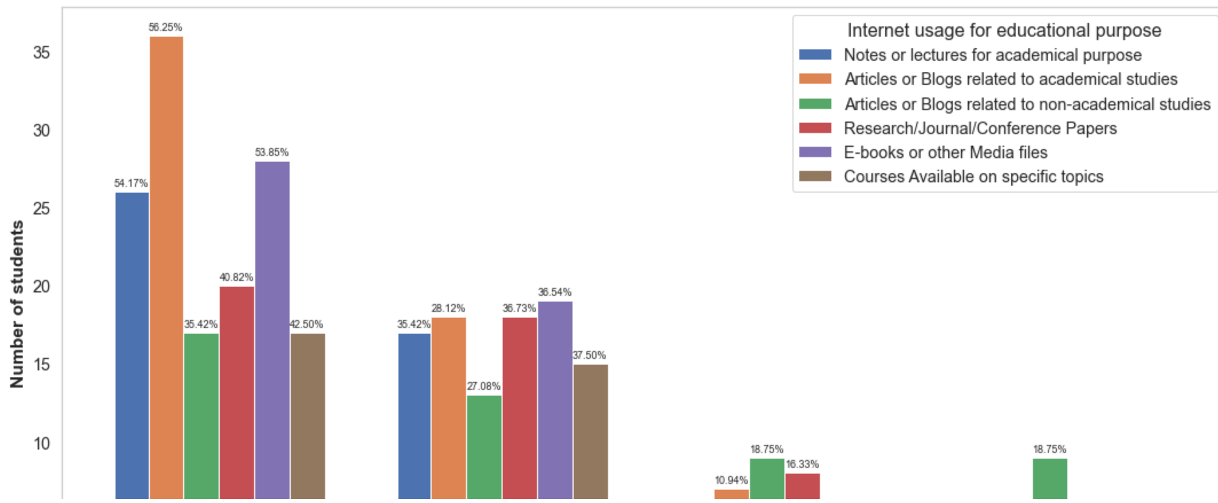
In [96]:
```
plt.figure(figsize=(10, 11))
plt.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)
sns.set(font_scale=1.2)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Internet Usage For Educational Purpose'],
                     color=['orange', 'royalblue', 'salmon', 'steelblue', 'vic
                     title='Different reasons for internet browsing for educat
                     xlabel='Internet usage for educational purpose')

plt.show()
```



Let's check the distribution of this feature against the target i.e. `'Academic Performance'`.

In [97]:
```
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

dictionary = cat_vs_cat_bar_plot(university_df, 'Internet Usage For Educationa
                              ['Notes or lectures for academical purpose',
                               'Articles or Blogs related to academical studie
                               'Articles or Blogs related to non-academical st
                               'Research/Journal/Conference Papers','E-books o
                               'Courses Available on specific topics'])

labels = ['Excellent', 'Good', 'Average', 'Not Satisfactory']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - (width + 0.12), dictionary['Notes or lectures for academic
                width/2, label = 'Notes or lectures for academical purpose')
rects2 = ax.bar(x - width, dictionary['Articles or Blogs related to academical
                width/2, label = 'Articles or Blogs related to academical stud
rects3 = ax.bar(x - width/2, dictionary['Articles or Blogs related to non-acad
                width/2, label = 'Articles or Blogs related to non-academical 
rects4 = ax.bar(x, dictionary['Research/Journal/Conference Papers'],
                width/2, label = 'Research/Journal/Conference Papers')
rects5 = ax.bar(x + width/2, dictionary['E-books or other Media files'],
                width/2, label = 'E-books or other Media files')
rects6 = ax.bar(x + width, dictionary['Courses Available on specific topics'],
                width/2, label = 'Courses Available on specific topics')

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Academic performance', fontweight = 'bold')
# ax.set_title('Internet Usage For Educational Purpose W.R.T. Academic Perforn
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Internet usage for educational purpose', title_fontsize=16, l

sns.set(font_scale=0.75)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)
autolabel(rects6)

fig.tight_layout()

save_fig('Internet_Usage_For_Educational_Purpose_WRT_Academic_Performance_Freq

plt.show()
```

```
Saving figure Internet_Usage_For_Educational_Purpose_WRT_Academic_Performance_
Frequency_Distribution
```

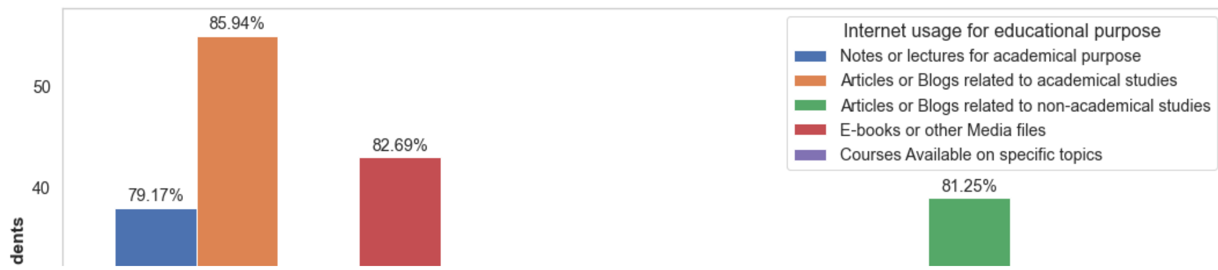


Let's check the distribution of this feature against the target i.e. `'Browsing Purpose'`.

In [98]:

```
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})


dictionary = cat_vs_cat_bar_plot_browsing_purpose(university_df, 'Internet Usa
                              university_df['Internet Usage For Educational Pu

labels = ['Academic', 'Non-academic']
x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots(figsize=(15, 8))
fig.subplots_adjust(top=0.5, bottom=0.1, hspace=0.5, wspace=0.2)

rects1 = ax.bar(x - width, dictionary['Notes or lectures for academical purpos
                width/2, label = 'Notes or lectures for academical purpose')
rects2 = ax.bar(x - width/2, dictionary['Articles or Blogs related to academic
                width/2, label = 'Articles or Blogs related to academical stu
rects3 = ax.bar(x, dictionary['Articles or Blogs related to non-academical stu
                width/2, label = 'Articles or Blogs related to non-academical
rects4 = ax.bar(x + width/2, dictionary['E-books or other Media files'],
                width/2, label = 'E-books or other Media files')
rects5 = ax.bar(x + width, dictionary['Courses Available on specific topics'],
                width/2, label = 'Courses Available on specific topics')

ax.set_ylabel('Number of students', fontweight = 'bold')
ax.set_xlabel('Browsing purpose', fontweight = 'bold')
# ax.set_title('Internet Usage For Educational Purpose W.R.T. Browsing Purpose
ax.set_xticks(x - width/2)
ax.set_xticklabels(labels)
ax.legend(title='Internet usage for educational purpose', title_fontsize=16, l

sns.set(font_scale=1.2)

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)

fig.tight_layout()

save_fig('Internet_Usage_For_Educational_Purpose_WRT_Browsing_Purpose_Frequenc
plt.show()
```

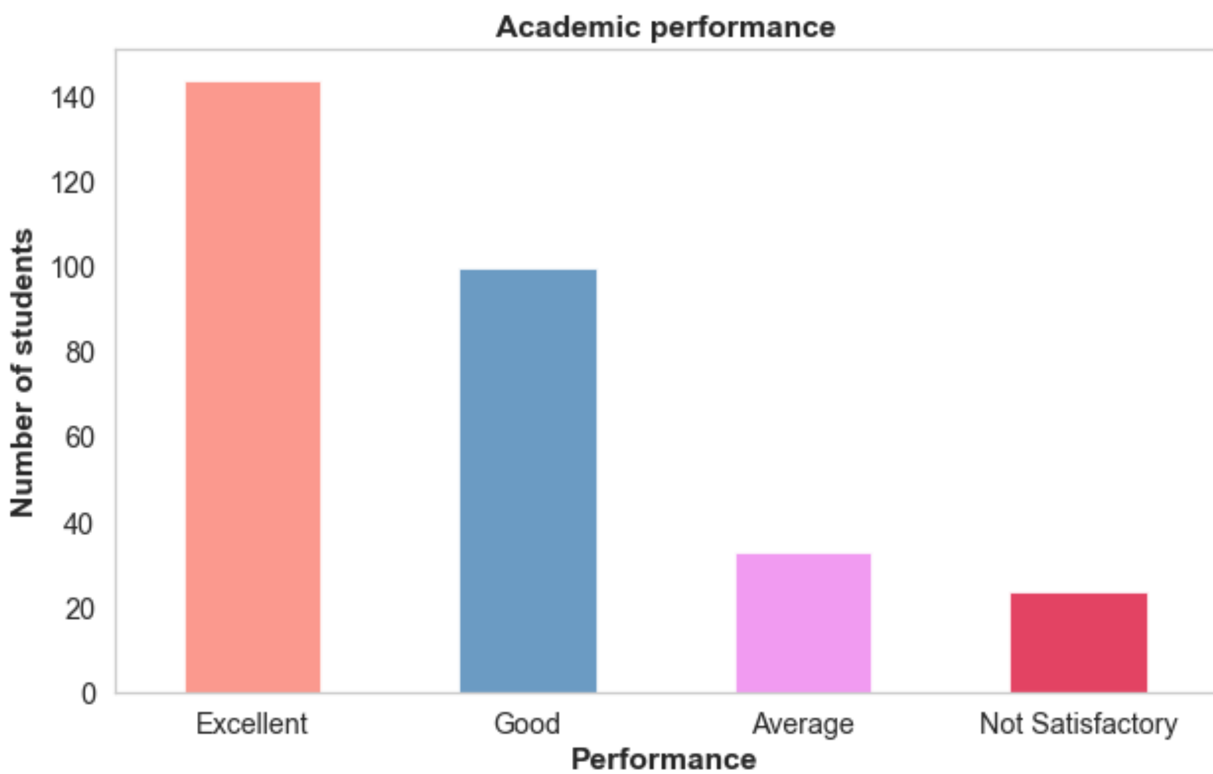
```
Saving figure Internet_Usage_For_Educational_Purpose_WRT_Browsing_Purpose_Freq
uency_Distribution
```



Plotting 'Academic Performance'

Let's check the histogram.

In [99]:
```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Academic Performance'], color=['salmon',
                     title='Academic performance', xlabel='Performance')

plt.show()
```

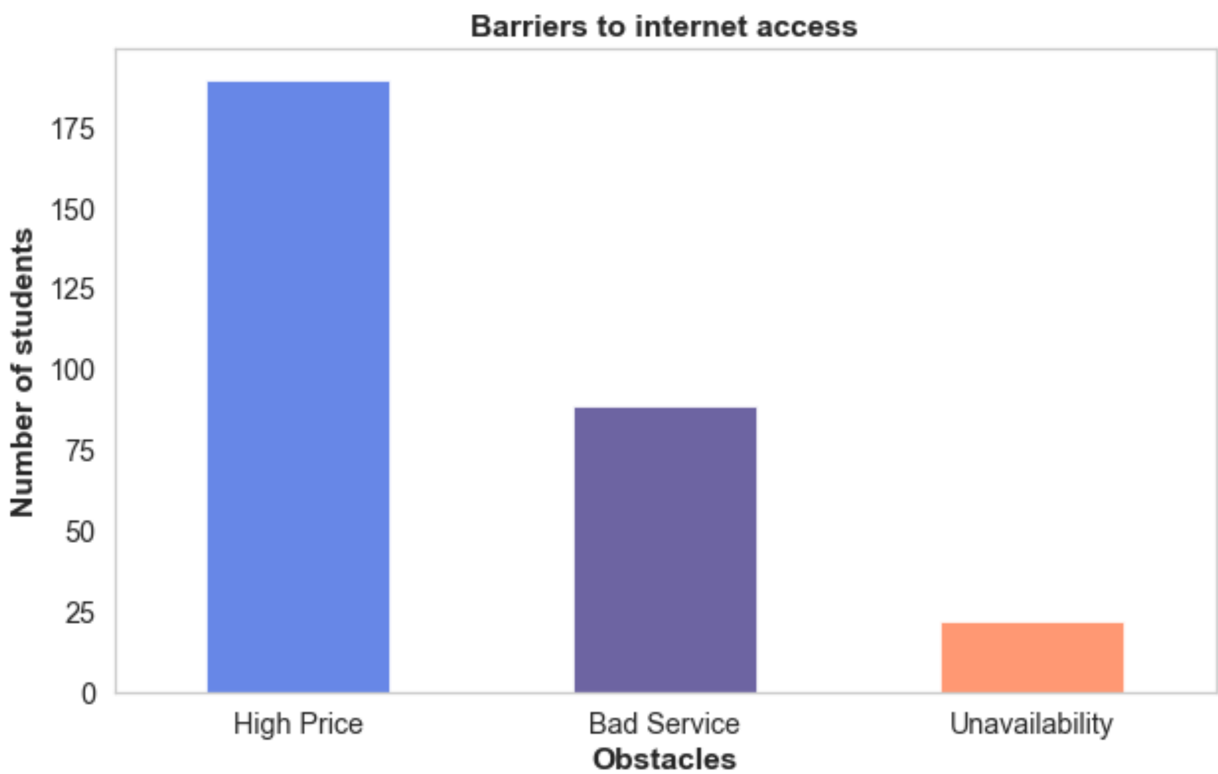


Plotting 'Barriers To Internet Access'

Let's check the histogram.

In [100…

```
plt.figure(figsize=(10, 6))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

categorical_bar_plot(university_df['Barriers To Internet Access'],
                     color=['royalblue', 'darkslateblue', 'coral', 'crimson'],
                     title='Barriers to internet access', xlabel='Obstacles')

plt.show()
```



Inspecting Age Closer

Let's define a function to make this process easier.

In [101…

```
# For Styling:
cust_palt = [
    '#111d5e', '#c70039', '#f37121', '#ffbd69', '#ffc93c'
]
```

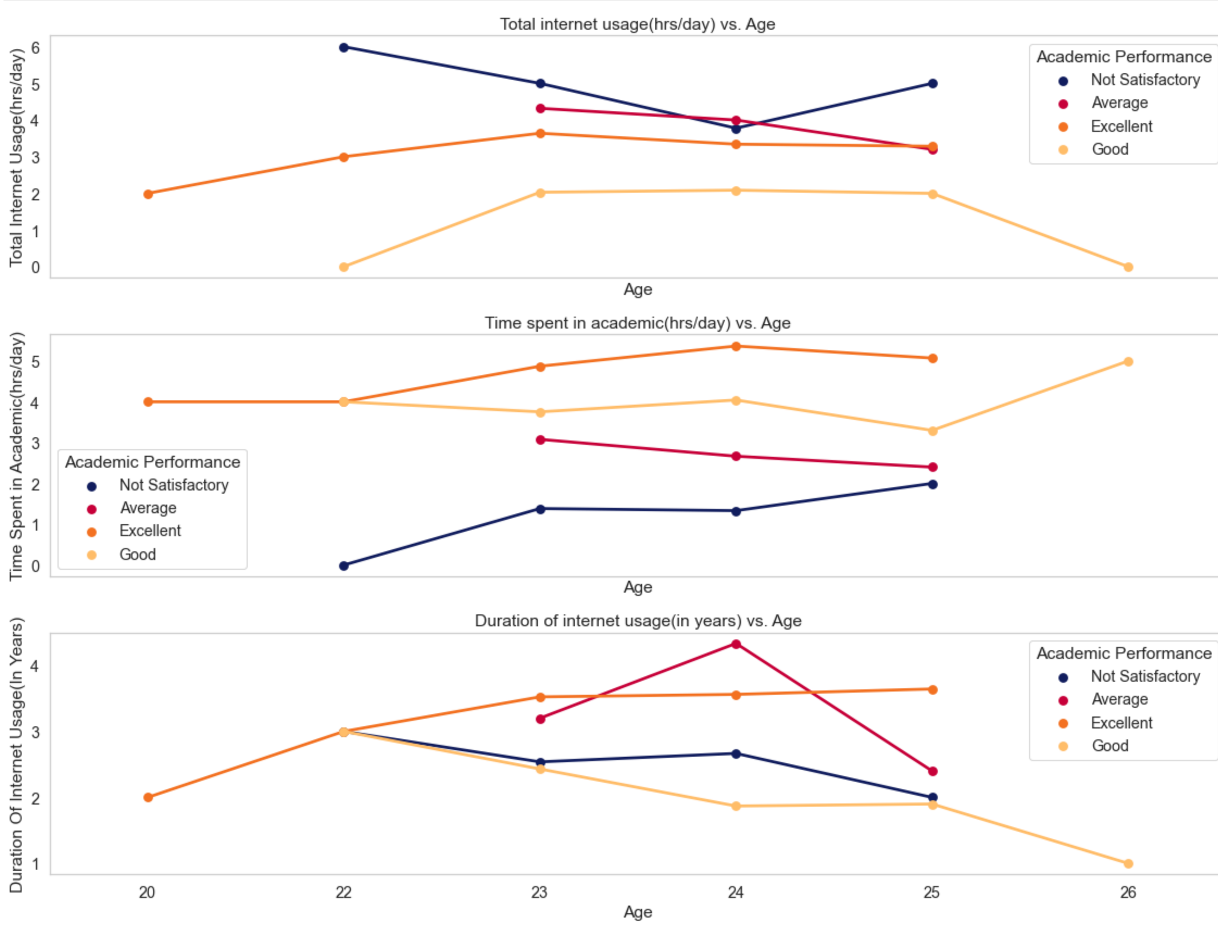
In [102…

```
def ctn_freq(dataframe, cols, xax, hue = None, rows = 3, columns = 1):

    sns.set_style("whitegrid", {'axes.grid' : False})

    ''' A function for displaying numerical data frequency vs age and conditi

    fig, axes = plt.subplots(rows, columns, figsize=(16, 12), sharex=True)
    axes = axes.flatten()

    for i, j in zip(dataframe[cols].columns, axes):
        sns.pointplot(x = xax,
                      y = i,
                      data = dataframe,
                      palette = cust_palt[:4],
                      hue = hue,
                      ax = j, ci = False)
        j.set_title(f'{str(i).capitalize()} vs. Age')


        plt.tight_layout()
```

Now let's inspect the columns `'Total Internet Usage(hrs/day)'`, `'Duration Of Internet Usage(In Years)'`, `'Time Spent in Academic(hrs/day)'` against the column `'Age'` and also segment the distribution by the target `'Academic Performance'`.

In [103…

```
ctn_freq(university_df,
         ['Total Internet Usage(hrs/day)', 'Time Spent in Academic(hrs/day)',
         'Age', hue='Academic Performance',rows=3, columns=1)
```

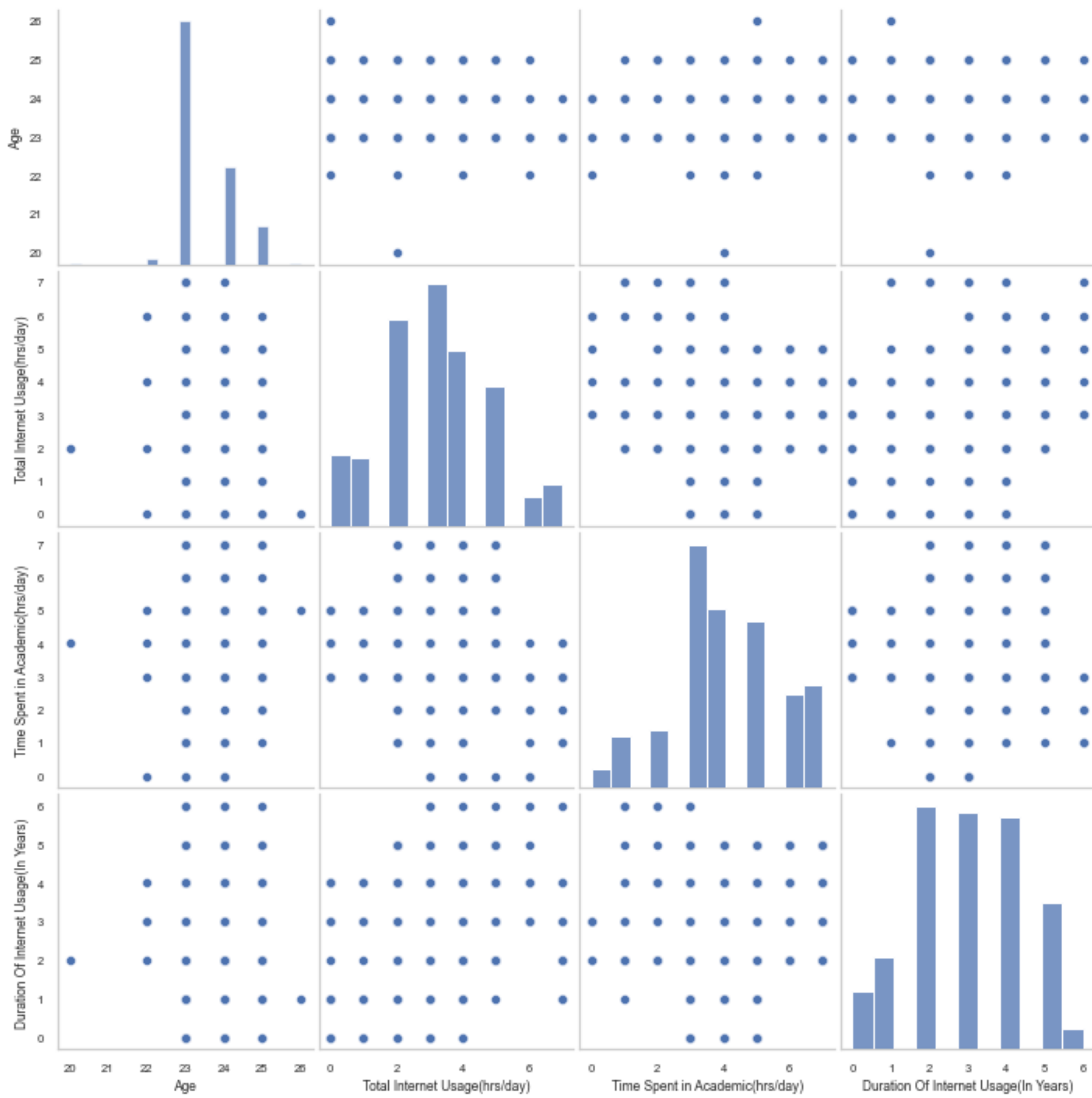


Multivariate Analysis

Multivariate analysis (MVA) is based on the principles of multivariate statistics, which involves observation and analysis of more than one statistical outcome variable at a time. Typically, MVA is used to address the situations where multiple measurements are made on each experimental unit and the relations among these measurements and their structures are important.

In [104…

```
# Numeric data vs each other and condition:

sns.set(font_scale = 0.7)
sns.set_style("whitegrid", {'axes.grid' : False})

sns.pairplot(university_df)

plt.show()
```



Let's add `hue = "Academic Performance"` **in the** `pairplot`

In [105…

```
sns.set(font_scale = 0.7)
sns.set_style("whitegrid", {'axes.grid' : False})

sns.pairplot(university_df, hue = "Academic Performance")

plt.show()
```

Correlations

We are going to use pearson correlation for to find linear relations between features, heatmap is decent way to show these relations.

In [106…

```
university_df.corr(method='pearson', min_periods=1)
```

Out[106…

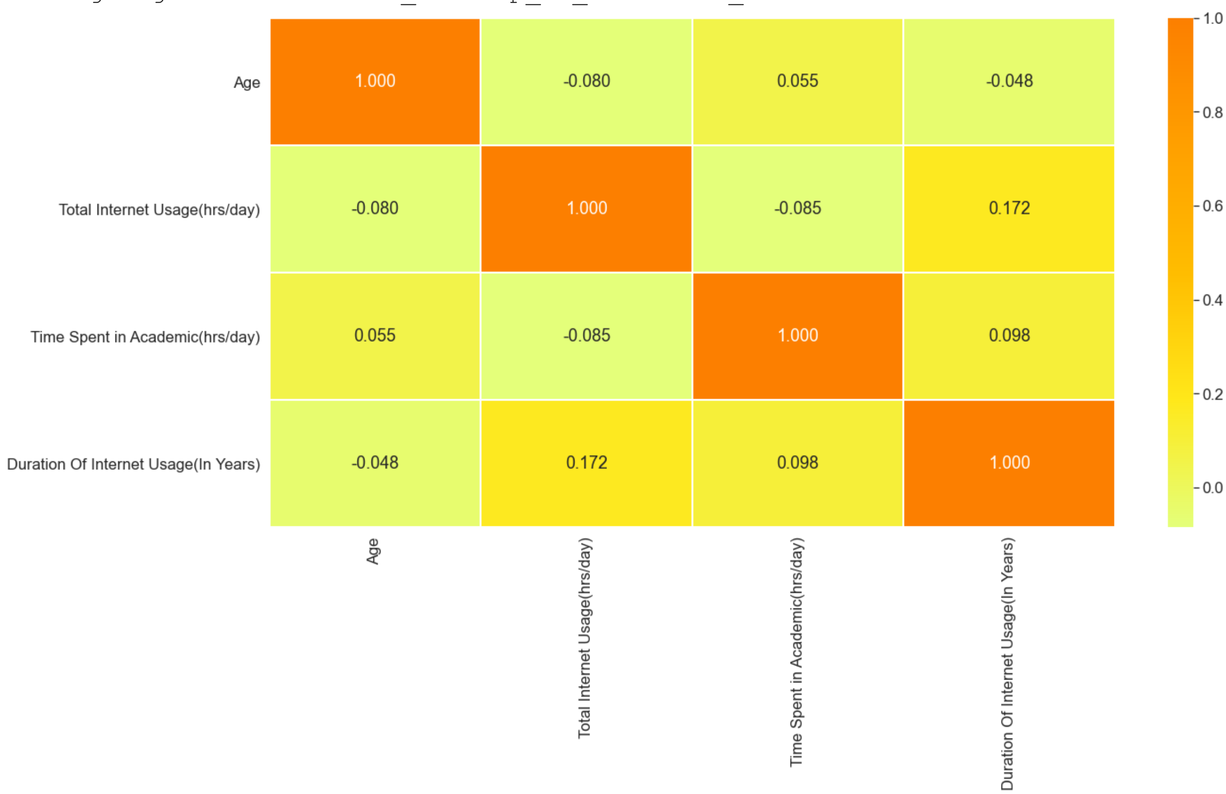
	Age	Total Internet Usage(hrs/day)	Time Spent in Academic(hrs/day)	Duration Of Internet Usage(In Years)
Age	1.000000	-0.080094	0.054634	-0.048158
Total Internet Usage(hrs/day)	-0.080094	1.000000	-0.085073	0.171855

	Age	Total Internet Usage(hrs/day)	Time Spent in Academic(hrs/day)	Duration Of Internet Usage(In Years)

In [107…

```
# Correlation heatmap between variables:

sns.set(font_scale=1.5)

correlation_df = university_df.corr(method='pearson', min_periods=1)
mask = np.triu(correlation_df.corr())

plt.figure(figsize=(20, 12))

sns.heatmap(correlation_df,
            annot = True,
            fmt = '.3f',
            cmap = 'Wistia',
            linewidths = 1,
            cbar = True)

save_fig('Correlation_heatmap_of_numerical_variables')

plt.show()
```

```
Saving figure Correlation_heatmap_of_numerical_variables
```



Start Predicting the Models

Let's drop the target column `'Academic Performance'` from the main dataframe. Store the target column on a separate column first.

In [108…
```
university_labels = university_df["Academic Performance"].copy()

university_df.drop("Academic Performance", axis = 1, inplace=True)

university_df.head()
```

Out[108…

	Gender	Age	Frequently Visited Website	Effectiveness Of Internet Usage	Devices Used For Internet Browsing	Location Of Internet Use	Household Internet Facilities	Time Of Internet Browsing	Frequency Of Internet Usage	S R
0	Female	23	Instagram	Not at all	Desktop	Library	Connected	Night	Daily	
1	Female	23	Youtube	Effective	Mobile	University	Connected	Morning	Daily	
2	Female	23	Whatsapp	Effective	Mobile	University	Connected	Midnight	Daily	
3	Female	23	Whatsapp	Somewhat Effective	Laptop and Mobile	University	Connected	Morning	Daily	
4	Male	24	Facebook	Somewhat Effective	Laptop and Mobile	Cyber Cafe	Connected	Night	Daily	

In [109…
```
university_labels.head()
```

Out[109…
```
0    Not Satisfactory
1             Average
2           Excellent
3                Good
4                Good
Name: Academic Performance, dtype: object
```

Let's separate the numerical and categorical columns for preprocessing. Let's check which columns are numerical and which are categorical.

In [110…
```
university_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 19 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
```

```
 0   Gender                                     301 non-null    object
 1   Age                                        301 non-null    int64
 2   Frequently Visited Website                 301 non-null    object
 3   Effectiveness Of Internet Usage            301 non-null    object
 4   Devices Used For Internet Browsing         301 non-null    object
 5   Location Of Internet Use                   301 non-null    object
 6   Household Internet Facilities              301 non-null    object
 7   Time Of Internet Browsing                  301 non-null    object
 8   Frequency Of Internet Usage                301 non-null    object
 9   Place Of Student's Residence               301 non-null    object
 10  Total Internet Usage(hrs/day)              301 non-null    int64
 11  Time Spent in Academic(hrs/day)            301 non-null    int64
 12  Purpose Of Internet Use                    301 non-null    object
 13  Duration Of Internet Usage(In Years)       301 non-null    int64
 14  Browsing Purpose                           301 non-null    object
 15  Webinar                                    301 non-null    object
 16  Priority Of Learning On The Internet       301 non-null    object
 17  Internet Usage For Educational Purpose     301 non-null    object
 18  Barriers To Internet Access                301 non-null    object
dtypes: int64(4), object(15)
memory usage: 44.8+ KB
```

The columns `'Age'` , `'Total Internet Usage(hrs/day)'` , `'Time Spent in Academic(hrs/day)'` , `'Duration Of Internet Usage(In Years)'` contain numerical values. Let's separate them from the main dataframe.

In [111…

```
university_cat = university_df.drop(['Age', 'Total Internet Usage(hrs/day)','
                                     'Duration Of Internet Usage(In Years)'], axis = 1)

university_cat.head()
```

Out[111…

	Gender	Frequently Visited Website	Effectiveness Of Internet Usage	Devices Used For Internet Browsing	Location Of Internet Use	Household Internet Facilities	Time Of Internet Browsing	Frequency Of Internet Usage	Place Studen Residen
0	Female	Instagram	Not at all	Desktop	Library	Connected	Night	Daily	Remo
1	Female	Youtube	Effective	Mobile	University	Connected	Morning	Daily	Remo
2	Female	Whatsapp	Effective	Mobile	University	Connected	Midnight	Daily	To
3	Female	Whatsapp	Somewhat Effective	Laptop and Mobile	University	Connected	Morning	Daily	Villa
4	Male	Facebook	Somewhat Effective	Laptop and Mobile	Cyber Cafe	Connected	Night	Daily	To

Gender	Frequently Visited	Effectiveness Of Internet	Devices Used For ...	Location Of ...	Household Internet	Time Of Internet	Frequency Of ...	Place Studen

In [112…

```
university_cat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 15 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Gender                                  301 non-null    object
 1   Frequently Visited Website              301 non-null    object
 2   Effectiveness Of Internet Usage         301 non-null    object
 3   Devices Used For Internet Browsing      301 non-null    object
 4   Location Of Internet Use                301 non-null    object
 5   Household Internet Facilities           301 non-null    object
 6   Time Of Internet Browsing               301 non-null    object
 7   Frequency Of Internet Usage             301 non-null    object
 8   Place Of Student's Residence            301 non-null    object
 9   Purpose Of Internet Use                 301 non-null    object
 10  Browsing Purpose                        301 non-null    object
 11  Webinar                                 301 non-null    object
 12  Priority Of Learning On The Internet    301 non-null    object
 13  Internet Usage For Educational Purpose  301 non-null    object
 14  Barriers To Internet Access             301 non-null    object
dtypes: object(15)
memory usage: 35.4+ KB
```

Store the numerical attributes in a separate variable.

In [113…

```
university_num = university_df[['Age', 'Total Internet Usage(hrs/day)','Time S
                       'Duration Of Internet Usage(In Years)']].copy()

university_num.head()
```

Out[113…

	Age	Total Internet Usage(hrs/day)	Time Spent in Academic(hrs/day)	Duration Of Internet Usage(In Years)
0	23	4	2	2
1	23	1	3	2
2	23	5	6	2
3	23	0	4	1
4	24	1	5	3

In [114…

```
university_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 4 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Age                                   301 non-null    int64
 1   Total Internet Usage(hrs/day)         301 non-null    int64
 2   Time Spent in Academic(hrs/day)       301 non-null    int64
 3   Duration Of Internet Usage(In Years)  301 non-null    int64
dtypes: int64(4)
memory usage: 9.5 KB
```

Let's integerize the categorical values in the dataset `university_cat` . We'll use the

`LabelEncoder` from the `sklearn.preprocessing`.

In [115…

```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

temp_df_cat = university_cat.apply(preprocessing.LabelEncoder().fit_transform)

temp_df_cat.head()
```

Out[115…

	Gender	Frequently Visited Website	Effectiveness Of Internet Usage	Devices Used For Internet Browsing	Location Of Internet Use	Household Internet Facilities	Time Of Internet Browsing	Frequency Of Internet Usage	Place (Student Residen
0	0	3	1	0	2	0	3	0	
1	0	6	0	3	4	0	2	0	
2	0	5	0	3	4	0	1	0	
3	0	5	2	2	4	0	2	0	
4	1	0	2	2	0	0	3	0	

Let's Normalize the dataset using `sklearn`'s `normalize` function. But the dataset seems to perform better without normalization.

In [116…

```
# from sklearn.preprocessing import normalize


# temp_df_normalized = normalize(college_num)
# temp_df_num = pd.DataFrame(temp_df_normalized, columns = list(college_num))

# temp_df_num.head()
```

Let's combine the preprocessed numerical and categorical part of the dataset.

In [117…

```
# Place the DataFrames side by side

X = pd.concat([university_num, temp_df_cat], axis=1)
y = university_labels

X.head()
```

Out[117…

	Age	Total Internet Usage(hrs/day)	Time Spent in Academic(hrs/day)	Duration Of Internet Usage(In Years)	Gender	Frequently Visited Website	Effectiveness Of Internet Usage	Devices Used For Internet Browsing
0	23	4	2	2	0	3	1	0
1	23	1	3	2	0	6	0	3
2	23	5	6	2	0	5	0	3

Age	Total Internet	Time Spent in	Duration Of Internet	Gender	Frequently Visited	Effectiveness Of Internet	Devices Used For

Split the dataset for training and testing purposes. We'll use `sklearn` 's `train_test_split` function to do this.

In [118…

```
# split a dataset into train and test sets
from sklearn.model_selection import train_test_split


# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)

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

```
(210, 19) (91, 19) (210,) (91,)
```

Implementing Machine Learning Algorithms For Classification

Stochastic Gradient Descent

Let's start with Stochastic Gradient Descent classifier. We'll use `sklearn` 's `SGDClassifier` to do this. After training the classifier, we'll check the model accuracy score.

In [119…

```
from sklearn.linear_model import SGDClassifier
from sklearn import metrics


sgd_clf = SGDClassifier(max_iter=100, tol=1e-3, random_state=42)

sgd_clf.fit(X_train, y_train)

score = sgd_clf.score(X_train, y_train)
print("Training score: ", score)
```

```
Training score:  0.580952380952381
```

Let's check the confusion matrix and classification report of this model.

In [120…
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


y_pred_sgd = sgd_clf.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred_sgd)
class_report = classification_report(y_test, y_pred_sgd)


print("Accuracy:", metrics.accuracy_score(y_test, y_pred_sgd))

print(conf_mat)
print(class_report)
```

```
Accuracy: 0.6263736263736264
[[10  1  0  0]
 [ 7 35  0  0]
 [13  5 12  0]
 [ 8  0  0  0]]
                  precision    recall  f1-score   support

         Average       0.26      0.91      0.41        11
       Excellent       0.85      0.83      0.84        42
            Good       1.00      0.40      0.57        30
Not Satisfactory       0.00      0.00      0.00         8

        accuracy                           0.63        91
       macro avg       0.53      0.54      0.46        91
    weighted avg       0.76      0.63      0.63        91

E:\Users\MSI\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:12
21: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Let's perform cross validation using this model. We'll `KFold` for this purpose.

In [121…
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


cv_sgd = KFold(n_splits=10, shuffle=True, random_state=42)
cross_val_score(sgd_clf, X_train, y_train, cv=cv_sgd, scoring="accuracy", n_jo
```

Out[121…
```
array([0.61904762, 0.33333333, 0.42857143, 0.71428571, 0.71428571,
       0.71428571, 0.80952381, 0.66666667, 0.38095238, 0.80952381])
```

In [122…
```
scores = cross_val_score(sgd_clf, X_test, y_test, cv=cv_sgd, scoring="accuracy
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.672 (0.133)
```

Let's check the score.

In [123…
```
scores = cross_val_score(sgd_clf, X_test, y_test, cv=3, scoring="accuracy", n_
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.605 (0.091)
```

Let's plot the training accuracy curve. But first we'll train and predict the model with

`max_iter` in the range of (5, 300)

In [124…

```
m_iter = []
training = []
test = []
scores = {}

max_i = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 70, 80, 90, 100, 130,

for i in range(len(max_i)):
    clf = SGDClassifier(max_iter=max_i[i], tol=1e-3, random_state=42)

    clf.fit(X_train, y_train)

    training_score = clf.score(X_train, y_train)
    test_score = clf.score(X_test, y_test)
    m_iter.append(max_i[i])

    training.append(training_score)
    test.append(test_score)
    scores[i] = [training_score, test_score]
```

```
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_grad
ient.py:570: ConvergenceWarning: Maximum number of iteration reached before co
nvergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_grad
ient.py:570: ConvergenceWarning: Maximum number of iteration reached before co
nvergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_grad
ient.py:570: ConvergenceWarning: Maximum number of iteration reached before co
nvergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_grad
ient.py:570: ConvergenceWarning: Maximum number of iteration reached before co
nvergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_grad
ient.py:570: ConvergenceWarning: Maximum number of iteration reached before co
nvergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_grad
ient.py:570: ConvergenceWarning: Maximum number of iteration reached before co
nvergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_grad
ient.py:570: ConvergenceWarning: Maximum number of iteration reached before co
nvergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
```

Let's check the `scores` variable.

In [125…

```
for keys, values in scores.items():
    print(keys, ':', values)
```

```
0 : [0.680952380952381, 0.6483516483516484]
1 : [0.5047619047619047, 0.5494505494505495]
2 : [0.7476190476190476, 0.6703296703296703]
3 : [0.3761904761904762, 0.3626373626373626]
4 : [0.7476190476190476, 0.6373626373626373]
5 : [0.6571428571428571, 0.5934065934065934]
```

```
6 : [0.7380952380952381, 0.6263736263736264]
7 : [0.580952380952381, 0.6263736263736264]
8 : [0.580952380952381, 0.6263736263736264]
9 : [0.580952380952381, 0.6263736263736264]
10 : [0.580952380952381, 0.6263736263736264]
11 : [0.580952380952381, 0.6263736263736264]
12 : [0.580952380952381, 0.6263736263736264]
13 : [0.580952380952381, 0.6263736263736264]
14 : [0.580952380952381, 0.6263736263736264]
15 : [0.580952380952381, 0.6263736263736264]
16 : [0.580952380952381, 0.6263736263736264]
17 : [0.580952380952381, 0.6263736263736264]
18 : [0.580952380952381, 0.6263736263736264]
19 : [0.580952380952381, 0.6263736263736264]
20 : [0.580952380952381, 0.6263736263736264]
```

Finally, let's plot the training score.

In [126…

```
# plt.figure(figsize=(10, 4))
# sns.set(font_scale=1.3)
# sns.set_style("whitegrid", {'axes.grid' : False})

# ax = sns.stripplot(m_iter, training);
# ax.set(xlabel ='max iteration', ylabel ='Training Score')

# plt.show()
```

Testing score.

In [127…

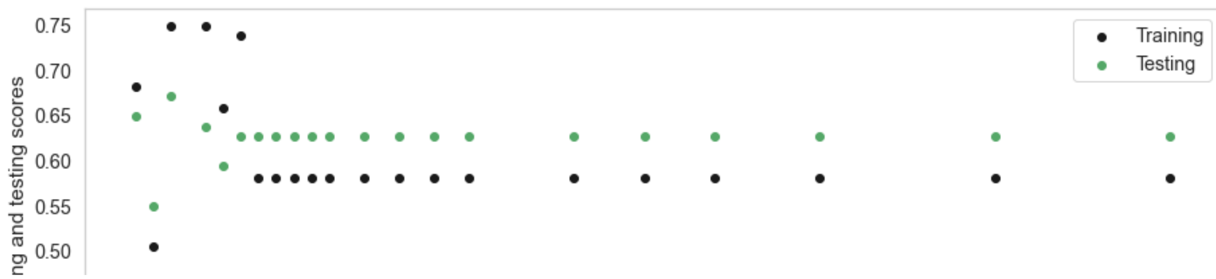
```
# plt.figure(figsize=(10, 4))
# sns.set(font_scale=1.3)
# sns.set_style("whitegrid", {'axes.grid' : False})

# ax = sns.stripplot(m_iter, test);
# ax.set(xlabel ='max iteration', ylabel ='Testing Score')

# plt.show()
```

Let's combine the two scores together to compare the two.

In [128…

```
plt.figure(figsize=(13, 5))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.scatter(m_iter, training, color ='k')
plt.scatter(m_iter, test, color ='g')

plt.ylabel('Training and testing scores')
plt.xlabel('Max iteration')
plt.legend(labels=['Training', 'Testing'])

save_fig('SGDClassifier_training_testing_scores')
plt.show()
```

```
Saving figure SGDClassifier_training_testing_scores
```



Decision Tree

Let's start with Decision Tree classifier. We'll use `sklearn` 's `DecisionTreeClassifier` to do this. After training the classifier, we'll check the model accuracy score.

In [129…

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

dec_tree_clf = DecisionTreeClassifier(max_depth=12, max_leaf_nodes = 50, rand

dec_tree_clf.fit(X_train, y_train)

score = dec_tree_clf.score(X_train, y_train)
print("Training score: ", score)
```

```
Training score:  0.9952380952380953
```

Let's check the confusion matrix and classification report of this model.

In [130…

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


y_pred_dec_tree = dec_tree_clf.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred_dec_tree)
class_report = classification_report(y_test, y_pred_dec_tree)


print("Accuracy:", metrics.accuracy_score(y_test, y_pred_dec_tree))

print(conf_mat)
print(class_report)
```

```
Accuracy: 0.6923076923076923
[[ 3  5  3  0]
 [ 2 35  4  1]
 [ 5  3 21  1]
 [ 2  1  1  4]]
                  precision    recall  f1-score   support

         Average       0.25      0.27      0.26        11
       Excellent       0.80      0.83      0.81        42
            Good       0.72      0.70      0.71        30
Not Satisfactory       0.67      0.50      0.57         8

        accuracy                           0.69        91
       macro avg       0.61      0.58      0.59        91
    weighted avg       0.69      0.69      0.69        91
```

Let's perform cross validation using this model. We'll `KFold` for this purpose.

In [131…
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


cv_dec_tree = KFold(n_splits=5, shuffle=True, random_state=42)
cross_val_score(dec_tree_clf, X_train, y_train, cv=cv_dec_tree, scoring="accu
```

Out[131…
```
array([0.71428571, 0.66666667, 0.61904762, 0.66666667, 0.64285714])
```

In [132…
```
scores = cross_val_score(dec_tree_clf, X_test, y_test, cv=cv_dec_tree, scorin
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.704 (0.024)
```

Let's check the score.

In [133…
```
scores = cross_val_score(dec_tree_clf, X_test, y_test, cv=3, scoring="accurac
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.595 (0.115)
```

Let's plot the training accuracy curve. But first we'll train and predict the model with `max_depth` in the range of (1, 27)

In [134…
```
m_depth = []
training = []
test = []
scores = {}

max_d = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2

for i in range(len(max_d)):
    clf = DecisionTreeClassifier(max_depth=max_d[i], max_leaf_nodes = 50, ran

    clf.fit(X_train, y_train)

    training_score = clf.score(X_train, y_train)
    test_score = clf.score(X_test, y_test)
    m_depth.append(max_d[i])

    training.append(training_score)
    test.append(test_score)
    scores[i] = [training_score, test_score]
```

Let's check the `scores` variable.

In [135…
```
for keys, values in scores.items():
    print(keys, ':', values)
```

```
0 : [0.6333333333333333, 0.5714285714285714]
1 : [0.638095238095238, 0.5604395604395604]
2 : [0.7047619047619048, 0.6593406593406593]
3 : [0.7380952380952381, 0.6483516483516484]
4 : [0.7952380952380952, 0.6483516483516484]
5 : [0.8333333333333334, 0.6703296703296703]
6 : [0.8904761904761904, 0.6703296703296703]
7 : [0.9380952380952381, 0.6593406593406593]
8 : [0.9523809523809523, 0.6593406593406593]
```

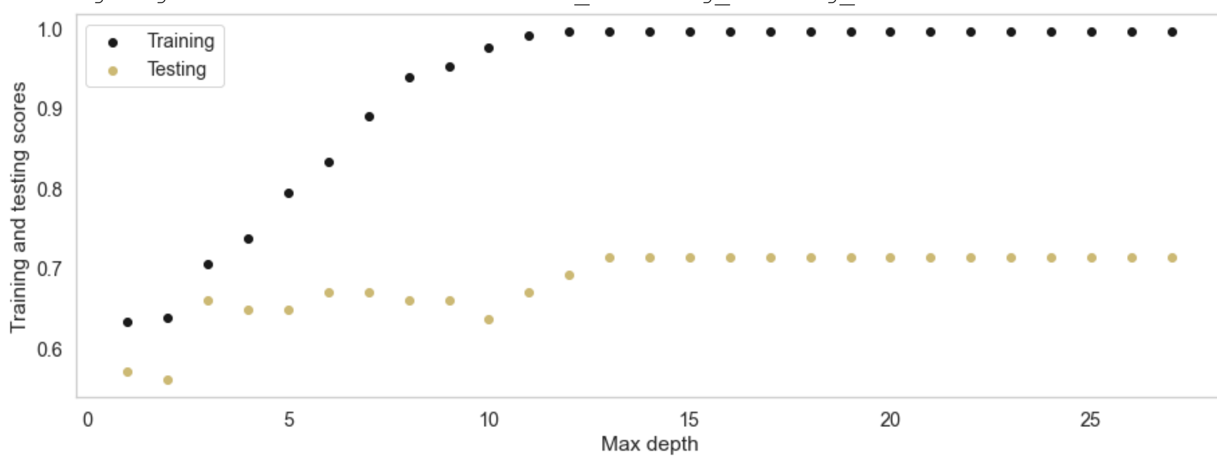
```
9 : [0.9761904761904762, 0.6373626373626373]
10 : [0.9904761904761905, 0.6703296703296703]
11 : [0.9952380952380953, 0.6923076923076923]
12 : [0.9952380952380953, 0.7142857142857143]
13 : [0.9952380952380953, 0.7142857142857143]
14 : [0.9952380952380953, 0.7142857142857143]
15 : [0.9952380952380953, 0.7142857142857143]
16 : [0.9952380952380953, 0.7142857142857143]
17 : [0.9952380952380953, 0.7142857142857143]
18 : [0.9952380952380953, 0.7142857142857143]
19 : [0.9952380952380953, 0.7142857142857143]
20 : [0.9952380952380953, 0.7142857142857143]
21 : [0.9952380952380953, 0.7142857142857143]
22 : [0.9952380952380953, 0.7142857142857143]
23 : [0.9952380952380953, 0.7142857142857143]
24 : [0.9952380952380953, 0.7142857142857143]
25 : [0.9952380952380953, 0.7142857142857143]
26 : [0.9952380952380953, 0.7142857142857143]
```

Finally, let's plot the training and testing scores together so that we can compare the two.

In [136…

```
plt.figure(figsize=(13, 5))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.scatter(m_depth, training, color ='k')
plt.scatter(m_depth, test, color ='y')

plt.ylabel('Training and testing scores')
plt.xlabel('Max depth')
plt.legend(labels=['Training', 'Testing'])

save_fig('DecisionTreeClassifier_training_testing_scores')
plt.show()
```

```
Saving figure DecisionTreeClassifier_training_testing_scores
```



Logistic Regression

Let's start with Logistic Regression classifier. We'll use `sklearn` 's `LogisticRegression` to do this. After training the classifier, we'll check the model accuracy score.

In [137…

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

log_reg = LogisticRegression(max_iter=5000, multi_class='multinomial', random_

log_reg.fit(X_train, y_train)

score = log_reg.score(X_train, y_train)
print("Training score: ", score)
```

```
Training score:  0.8333333333333334
```

Let's check the confusion matrix and classification report of this model.

In [138…

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


y_pred_log = log_reg.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred_log)
class_report = classification_report(y_test, y_pred_log)


print("Accuracy:", metrics.accuracy_score(y_test, y_pred_log))

print(conf_mat)
print(class_report)
```

```
Accuracy: 0.7142857142857143
[[ 2  4  5  0]
 [ 1 36  4  1]
 [ 1  8 20  1]
 [ 1  0  0  7]]
                  precision    recall  f1-score   support

         Average       0.40      0.18      0.25        11
       Excellent       0.75      0.86      0.80        42
            Good       0.69      0.67      0.68        30
Not Satisfactory       0.78      0.88      0.82         8

        accuracy                           0.71        91
       macro avg       0.65      0.65      0.64        91
    weighted avg       0.69      0.71      0.70        91
```

Let's perform cross validation using this model. We'll `KFold` for this purpose.

In [139…

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


cv_log_reg = KFold(n_splits=10, shuffle=True, random_state=42)
cross_val_score(log_reg, X_train, y_train, cv=cv_log_reg, scoring="accuracy",
```

Out[139…

```
array([0.52380952, 0.9047619 , 0.47619048, 0.85714286, 0.80952381,
       0.71428571, 0.66666667, 0.71428571, 0.71428571, 0.80952381])
```

In [140…

```
scores = cross_val_score(log_reg, X_test, y_test, cv=cv_log_reg, scoring="accu
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.658 (0.138)
```

Let's check the score.

In [141…

```
scores = cross_val_score(log_reg, X_test, y_test, cv=4, scoring="accuracy", n_
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.639 (0.121)
```

Let's plot the training accuracy curve. But first we'll train and predict the model with `max_iter` in the range of (50, 200)

In [142…

```
m_iter = []
training = []
test = []
scores = {}

max_i = [50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 100(
         1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000]
#          22, 23, 24, 25, 26, 27]

for i in range(len(max_i)):
    clf = LogisticRegression(max_iter=max_i[i], multi_class='multinomial', ra

    clf.fit(X_train, y_train)

    training_score = clf.score(X_train, y_train)
    test_score = clf.score(X_test, y_test)
    m_iter.append(max_i[i])

    training.append(training_score)
    test.append(test_score)
    scores[i] = [training_score, test_score]
```

```
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
```

```
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
```

```
ion
  n_iter_i = _check_optimize_result(
E:\Users\MSI\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
```

Let's check the `scores` variable.

In [143…

```
for keys, values in scores.items():
    print(keys, ':', values)
```

```
0 : [0.8142857142857143, 0.7032967032967034]
1 : [0.8333333333333334, 0.7032967032967034]
2 : [0.8095238095238095, 0.6923076923076923]
3 : [0.8380952380952381, 0.7032967032967034]
4 : [0.8333333333333334, 0.6923076923076923]
5 : [0.8380952380952381, 0.7032967032967034]
6 : [0.8428571428571429, 0.6923076923076923]
7 : [0.8380952380952381, 0.7252747252747253]
8 : [0.8428571428571429, 0.7362637362637363]
9 : [0.8428571428571429, 0.7362637362637363]
10 : [0.8333333333333334, 0.7142857142857143]
11 : [0.8333333333333334, 0.7142857142857143]
12 : [0.8333333333333334, 0.7142857142857143]
13 : [0.8333333333333334, 0.7142857142857143]
14 : [0.8333333333333334, 0.7252747252747253]
15 : [0.8333333333333334, 0.7362637362637363]
16 : [0.8333333333333334, 0.7142857142857143]
17 : [0.8333333333333334, 0.7142857142857143]
18 : [0.8333333333333334, 0.7142857142857143]
19 : [0.8333333333333334, 0.7142857142857143]
20 : [0.8333333333333334, 0.7142857142857143]
21 : [0.8333333333333334, 0.7142857142857143]
22 : [0.8333333333333334, 0.7142857142857143]
23 : [0.8333333333333334, 0.7142857142857143]
24 : [0.8333333333333334, 0.7142857142857143]
```

Finally, let's plot the training and testing scores together so that we can compare the two.

In [144…

```
plt.figure(figsize=(13, 5))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.scatter(m_iter, training, color ='k')
plt.scatter(m_iter, test, color ='r')

plt.ylabel('Training and testing scores')
plt.xlabel('Max iteration')
plt.legend(labels=['Training', 'Testing'])

save_fig('LogisticRegression_training_testing_scores')
plt.show()
```

```
Saving figure LogisticRegression_training_testing_scores
```

Random Forest

Let's start with Random Forest classifier. We'll use `sklearn` 's `RandomForestClassifier` to do this. After training the classifier, we'll check the model accuracy score.

In [145…

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

random_for_clf = RandomForestClassifier(n_estimators=14, max_depth=50, random_

random_for_clf.fit(X_train, y_train)

score = random_for_clf.score(X_train, y_train)
print("Training score: ", score)
```

```
Training score:  0.9952380952380953
```

Let's check the confusion matrix and classification report of this model.

In [146…

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


y_pred_rand = random_for_clf.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred_rand)
class_report = classification_report(y_test, y_pred_rand)


print("Accuracy:", metrics.accuracy_score(y_test, y_pred_rand))

print(conf_mat)
print(class_report)
```

```
Accuracy: 0.7582417582417582
[[ 3  6  2  0]
 [ 1 38  3  0]
 [ 0  5 25  0]
 [ 0  4  1  3]]
                  precision    recall  f1-score   support

         Average       0.75      0.27      0.40        11
       Excellent       0.72      0.90      0.80        42
            Good       0.81      0.83      0.82        30
Not Satisfactory       1.00      0.38      0.55         8

        accuracy                           0.76        91
       macro avg       0.82      0.60      0.64        91
    weighted avg       0.78      0.76      0.74        91
```

Let's perform cross validation using this model. We'll `KFold` for this purpose.

In [147…
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


cv_rand_for = KFold(n_splits=10, shuffle=True, random_state=42)
cross_val_score(random_for_clf, X_train, y_train, cv=cv_rand_for, scoring="ac
```

Out[147…
```
array([0.61904762, 0.9047619 , 0.66666667, 0.85714286, 0.76190476,
       0.71428571, 0.76190476, 0.76190476, 0.66666667, 0.76190476])
```

In [148…
```
scores = cross_val_score(random_for_clf, X_test, y_test, cv=cv_rand_for, scor
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.702 (0.142)
```

Let's check the score.

In [149…
```
scores = cross_val_score(random_for_clf, X_test, y_test, cv=4, scoring="accura
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.715 (0.087)
```

Let's plot the training accuracy curve. But first we'll train and predict the model with `n_estimators` in the range of (1, 35)

In [150…
```
n_estimate = []
training = []
test = []
scores = {}

for i in range(1, 35):
    clf = RandomForestClassifier(n_estimators=i, max_depth=50, random_state=4

    clf.fit(X_train, y_train)

    training_score = clf.score(X_train, y_train)
    test_score = clf.score(X_test, y_test)
    n_estimate.append(i)

    training.append(training_score)
    test.append(test_score)
    scores[i] = [training_score, test_score]
```

Let's check the `scores` variable.

In [151…
```
for keys, values in scores.items():
    print(keys, ':', values)
```

```
1 : [0.8666666666666667, 0.5824175824175825]
2 : [0.861904761904762, 0.5824175824175825]
3 : [0.9619047619047619, 0.6373626373626373]
4 : [0.9714285714285714, 0.6923076923076923]
5 : [0.9857142857142858, 0.6813186813186813]
6 : [0.9809523809523809, 0.7362637362637363]
7 : [0.9857142857142858, 0.6813186813186813]
8 : [0.9904761904761905, 0.7142857142857143]
9 : [0.9952380952380953, 0.7032967032967034]
```

```
10 : [0.9857142857142858, 0.7142857142857143]
11 : [0.9952380952380953, 0.7252747252747253]
12 : [0.9952380952380953, 0.7692307692307693]
13 : [0.9952380952380953, 0.7582417582417582]
14 : [0.9952380952380953, 0.7582417582417582]
15 : [0.9952380952380953, 0.7472527472527473]
16 : [0.9952380952380953, 0.7472527472527473]
17 : [0.9952380952380953, 0.7252747252747253]
18 : [0.9952380952380953, 0.7362637362637363]
19 : [1.0, 0.7032967032967034]
20 : [1.0, 0.7362637362637363]
21 : [1.0, 0.7142857142857143]
22 : [1.0, 0.7362637362637363]
23 : [1.0, 0.7692307692307693]
24 : [1.0, 0.7252747252747253]
25 : [1.0, 0.7472527472527473]
26 : [1.0, 0.7582417582417582]
27 : [1.0, 0.7582417582417582]
28 : [1.0, 0.7472527472527473]
29 : [1.0, 0.7582417582417582]
30 : [1.0, 0.7692307692307693]
31 : [1.0, 0.7582417582417582]
32 : [1.0, 0.7582417582417582]
33 : [1.0, 0.7692307692307693]
```

Finally, let's plot the training and testing scores together so that we can compare the two.

In [152…

```
plt.figure(figsize=(13, 5))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

plt.scatter(n_estimate, training, color ='k')
plt.scatter(n_estimate, test, color ='b')

plt.ylabel('Training and testing scores')
plt.xlabel('N estimators')
plt.legend(labels=['Training', 'Testing'])

save_fig('RandomForestClassifier_training_testing_scores')
plt.show()
```

```
Saving figure RandomForestClassifier_training_testing_scores
```

Naive Bayes

Let's start with Naive Bayes classifier. We'll use `sklearn` **'s** `GaussianNB` **,** `MultinomialNB`

and `CategoricalNB` to do this. After training the classifier, we'll check the model accuracy score.

In [153…

```
### 1.GaussianNB
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics


gaussNB_clf = GaussianNB()

gaussNB_clf.fit(X_train, y_train)

score = gaussNB_clf.score(X_train, y_train)
print("Training score: ", score)
```

```
Training score:  0.7952380952380952
```

In [154…

```
### 2.MultinomialNB
from sklearn.naive_bayes import MultinomialNB


multinomNB_clf = MultinomialNB()

multinomNB_clf.fit(X_train, y_train)

score = multinomNB_clf.score(X_train, y_train)
print("Training score: ", score)
```

```
Training score:  0.7285714285714285
```

`GaussianNB` performs the best among the naive bayes classifiers.

Let's check the confusion matrix and classification report of this model.

In [155…

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


y_pred_nb = gaussNB_clf.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred_nb)
class_report = classification_report(y_test, y_pred_nb)


print("Accuracy:", metrics.accuracy_score(y_test, y_pred_nb))

print(conf_mat)
print(class_report)
```

```
Accuracy: 0.7692307692307693
[[ 5  1  3  2]
 [ 1 35  5  1]
 [ 2  3 25  0]
 [ 0  2  1  5]]
                  precision    recall  f1-score   support

         Average       0.62      0.45      0.53        11
       Excellent       0.85      0.83      0.84        42
            Good       0.74      0.83      0.78        30
Not Satisfactory       0.62      0.62      0.62         8

        accuracy                           0.77        91
```

```
   macro avg       0.71      0.69      0.69        91
weighted avg       0.77      0.77      0.77        91
```

Let's perform cross validation using this model. We'll `KFold` for this purpose.

In [156…

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


cv_gauss_nb = KFold(n_splits=5, shuffle=True, random_state=42)
cross_val_score(gaussNB_clf, X_train, y_train, cv=cv_gauss_nb, scoring="accura
```

Out[156…

```
array([0.71428571, 0.71428571, 0.80952381, 0.30952381, 0.66666667])
```

In [157…

```
scores = cross_val_score(gaussNB_clf, X_test, y_test, cv=cv_gauss_nb, scoring=
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.647 (0.106)
```

Let's check the confusion matrix and classification report of this model.

In [158…

```
scores = cross_val_score(gaussNB_clf, X_test, y_test, cv=4, scoring="accuracy"
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.504 (0.167)
```

Check Feature Importance

Univariate Selection

Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features. The code below uses the chi-squared (chi²) statistical test for non-negative features to select 10 of the best features from the Mobile Price Range Prediction Dataset.

In [159…

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2


bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X, y)

dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Specs', 'Score']  #naming the dataframe columns

print(featureScores.nlargest(10, 'Score'))  #print 10 best features
```

```
                              Specs      Score
2       Time Spent in Academic(hrs/day)  87.603189
1         Total Internet Usage(hrs/day)  72.972743
```

```
16  Priority Of Learning On The Internet  60.909090
6        Effectiveness Of Internet Usage  35.498851
3   Duration Of Internet Usage(In Years)  34.511385
9          Household Internet Facilities  31.763502
11           Frequency Of Internet Usage  19.424903
14                      Browsing Purpose  13.429896
13                 Purpose Of Internet Use  10.424208
```

Feature Importance

We can get the feature importance of each feature of our dataset by using the feature importance property of the model. Feature importance gives a score for each feature of the data, the higher the score more important or relevant is the feature towards our output variable. Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top 10 features for the dataset.

In [160…

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt


model = ExtraTreesClassifier()
model.fit(X, y)
print(model.feature_importances_) #use inbuilt class feature_importances of t

#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index = X.columns)
```

```
[0.03086515 0.10962564 0.14348648 0.10953231 0.02633029 0.05047774
 0.08123822 0.03671965 0.04048805 0.02531903 0.04295743 0.01816403
 0.03659215 0.0512274  0.02984827 0.03051738 0.07160146 0.03272851
 0.0322808 ]
```

Let's plot the top 10 most important features.

In [161…

```python
plt.figure(figsize=(13, 5))
sns.set(font_scale=1.3)
sns.set_style("whitegrid", {'axes.grid' : False})

feat_importances.nlargest(10).plot(kind='barh')

plt.xlabel('Important features')

save_fig('top_ten_important_features')
plt.show()
```

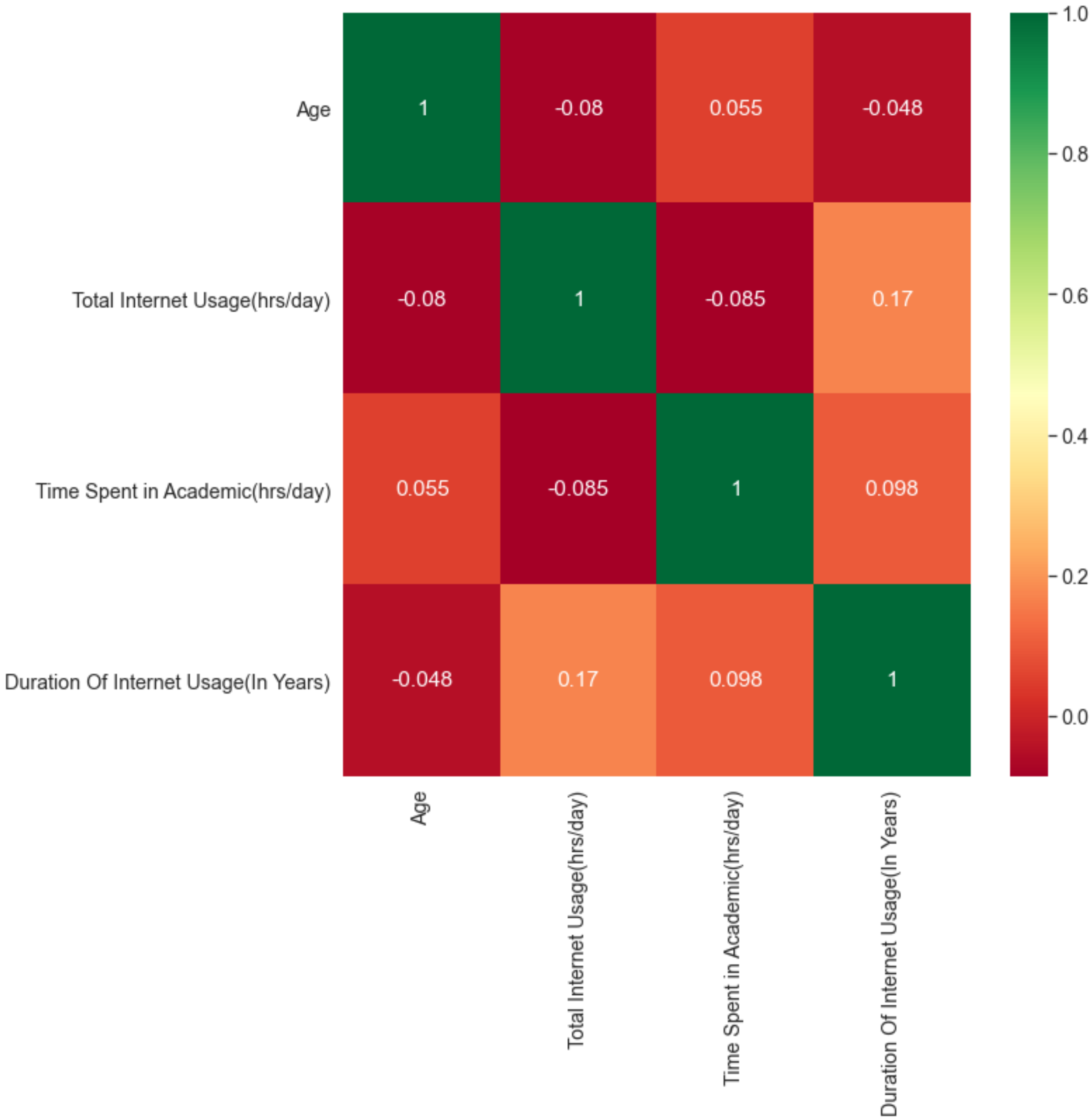
```
Saving figure top_ten_important_features
```

Correlation Matrix with Heatmap

Correlation states how the features are related to each other or the target variable. Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable) Heatmap makes it easy to identify which features are most related to the target variable, we will plot heatmap of correlated features using the seaborn library.

In [162…

```
import pandas as pd
import numpy as np
import seaborn as sns

#get correlations of each features in dataset
corrmat = university_df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(10,10))

#plot heat map
g=sns.heatmap(university_df[top_corr_features].corr(),annot=True,cmap="RdYlGn'
```



Hyperparameter Optimization

hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

We'll perform hyperparameter optimization using the following optimization techniques:

1. `GridSearchCV` - Exhaustive search over specified parameter values for an estimator.

2. `RandomizedSearchCV` - Randomized search on hyper parameters. The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings.

3. `BayesSearchCV` - Bayesian Optimization of model hyperparameters provided by the Scikit-Optimize library.

4. **Genetic Algorithm using the TPOT library** - TPOT is an open-source library for performing AutoML in Python. It makes use of the popular Scikit-Learn machine learning library for data transforms and machine learning algorithms and uses a Genetic Programming stochastic global search procedure to efficiently discover a top-performing model pipeline for a given dataset.

Let's start with `GridSearchCV`.

Hyperparameter Optimization using `GridSearchCV`

As we saw, the algorithms that performs the best is the `RandomForestClassifier`. Let's try and optimize the algorithm more to get a better result. First let's see the parameters that we'll try and tune in the `RandomForestClassifier`.

In [163…

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics


random_for_clf = RandomForestClassifier()

random_for_clf.get_params().keys()
```

Out[163…

```
dict_keys(['bootstrap', 'ccp_alpha', 'class_weight', 'criterion', 'max_depth',
'max_features', 'max_leaf_nodes', 'max_samples', 'min_impurity_decrease', 'min
_impurity_split', 'min_samples_leaf', 'min_samples_split', 'min_weight_fractio
n_leaf', 'n_estimators', 'n_jobs', 'oob_score', 'random_state', 'verbose', 'wa
rm_start'])
```

Let's create a dictionary that defines the parameters that we want to optimize.

In [164…

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 50, stop = 250, num = 5)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 50, num = 10)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]# Create the random grid


random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap
              }

print(random_grid)
```

```
{'n_estimators': [50, 100, 150, 200, 250], 'max_features': ['auto', 'sqrt'], '
max_depth': [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, None], 'min_samples_split
': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

Now, let's optimize the model using `GridSearchCV`. The method we'll use for cross validation is `RepeatedStratifiedKFold`.

In [165…

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# define the search
gs_rand_for = GridSearchCV(random_for_clf, param_grid=random_grid, scoring='ac

gs_rand_for.fit(X_train, y_train)

gs_rand_for.best_params_
```

Out[165…

```
{'bootstrap': True,
 'max_depth': None,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 200}
```

Let's check the training score. It should be performing much better now.

In [166…

```
gs_rand_for.score(X_train, y_train)
```

Out[166…

```
1.0
```

Let's put the model to use and predict our test set.

In [167…

```
y_pred_gs_rand = gs_rand_for.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred_gs_rand)
class_report = classification_report(y_test, y_pred_gs_rand)


print("Accuracy:", metrics.accuracy_score(y_test, y_pred_gs_rand))

print(conf_mat)
print(class_report)
```

```
Accuracy: 0.8571428571428571
[[ 2  1  3  0]
 [ 0 47  3  1]
 [ 0  3 25  0]
 [ 0  0  2  4]]
                  precision    recall  f1-score   support

         Average       1.00      0.33      0.50         6
       Excellent       0.92      0.92      0.92        51
            Good       0.76      0.89      0.82        28
Not Satisfactory       0.80      0.67      0.73         6

        accuracy                           0.86        91
       macro avg       0.87      0.70      0.74        91
    weighted avg       0.87      0.86      0.85        91
```

Hyperparameter Optimization using `RandomizedSearchCV`

As we saw, the algorithms that performs the best is the `RandomForestClassifier`. Let's try and optimize the algorithm more to get a better result. First let's see the parameters that we'll try and tune in the `RandomForestClassifier`.

We'll use the same dictionary that we created before as the parameters that we want to optimize. Now, let's optimize the model using `RandomizedSearchCV`. The method we'll use for cross validation is `RepeatedStratifiedKFold`.

In [168…

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform

# define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

rs_rand_for = RandomizedSearchCV(random_for_clf, random_grid, scoring='accura

rs_rand_for.fit(X_train, y_train)

rs_rand_for.best_params_
```

Out[168…

```
{'n_estimators': 150,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
```

```
'max_depth': 10,
```

Let's check the training score. It should be performing much better now.

In [169…

```
rs_rand_for.score(X_train, y_train)
```

Out[169…

```
0.9904761904761905
```

Let's put the model to use and predict our test set.

In [170…

```
y_pred_rs_rand = rs_rand_for.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred_rs_rand)
class_report = classification_report(y_test, y_pred_rs_rand)


print("Accuracy:", metrics.accuracy_score(y_test, y_pred_rs_rand))

print(conf_mat)
print(class_report)
```

```
Accuracy: 0.8461538461538461
[[ 1  2  3  0]
 [ 0 47  3  1]
 [ 0  3 25  0]
 [ 1  0  1  4]]
                  precision    recall  f1-score   support

         Average       0.50      0.17      0.25         6
       Excellent       0.90      0.92      0.91        51
            Good       0.78      0.89      0.83        28
Not Satisfactory       0.80      0.67      0.73         6

        accuracy                           0.85        91
       macro avg       0.75      0.66      0.68        91
    weighted avg       0.83      0.85      0.83        91
```

Hyperparameter Optimization using `BayesSearchCV`

The algorithms that performs the best is the `RandomForestClassifier` . Let's try and optimize the algorithm more to get a better result. First let's see the parameters that we'll try and tune in the `RandomForestClassifier` .

We'll use the same dictionary that we created before as the parameters that we want to optimize. Now, let's optimize the model using **Bayesian Optimization** implemented in `BayesSearchCV` . `skopt` library contains this class. The method we'll use for cross validation is `RepeatedStratifiedKFold` .

In [171…
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from skopt import BayesSearchCV


# define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# define the search
bs_rand_for = BayesSearchCV(estimator=random_for_clf, search_spaces=random_gr

# perform the search
bs_rand_for.fit(X, y)

# report the best result
print(bs_rand_for.best_score_)
print(bs_rand_for.best_params_)
```

```
E:\Users\MSI\anaconda3\lib\site-packages\skopt\optimizer\optimizer.py:449: Use
rWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
E:\Users\MSI\anaconda3\lib\site-packages\skopt\optimizer\optimizer.py:449: Use
rWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
0.7986738351254481
OrderedDict([('bootstrap', False), ('max_depth', None), ('max_features', 'sqrt
'), ('min_samples_leaf', 1), ('min_samples_split', 2), ('n_estimators', 200)])
```

Let's check the training score. It should be performing much better now.

In [172…
```
bs_rand_for.score(X_train, y_train)
```

Out[172… 1.0

Let's put the model to use and predict our test set.

In [173…
```
y_pred_bs_rand = bs_rand_for.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred_bs_rand)
class_report = classification_report(y_test, y_pred_bs_rand)


print("Accuracy:", metrics.accuracy_score(y_test, y_pred_bs_rand))

print(conf_mat)
print(class_report)
```

```
Accuracy: 1.0
[[ 6  0  0  0]
 [ 0 51  0  0]
 [ 0  0 28  0]
 [ 0  0  0  6]]
                  precision    recall  f1-score   support

         Average       1.00      1.00      1.00         6
       Excellent       1.00      1.00      1.00        51
            Good       1.00      1.00      1.00        28
Not Satisfactory       1.00      1.00      1.00         6

        accuracy                           1.00        91
```

```
   macro avg       1.00      1.00      1.00        91
weighted avg       1.00      1.00      1.00        91
```

Hyperparameter Optimization using `Genetic Algorithm`

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate "survival of the fittest" among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

To implement genetic algorithm we'll use **TPOT** which is an open-source library for performing AutoML in Python. It makes use of the popular Scikit-Learn machine learning library for data transforms and machine learning algorithms and uses a Genetic Programming stochastic global search procedure to efficiently discover a top-performing model pipeline for a given dataset.

We'll first have to numberize the training and test label set. Here we use `sklearn` 's `LabelEncoder` class to implement this.

In [186…

```
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

y_train_n = label_encoder.fit_transform(y_train)
y_test_n = label_encoder.fit_transform(y_test)

y_train_n
```

Out[186…

```
array([1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 0, 2, 1, 2, 3, 2, 3, 1, 0, 2, 1,
       1, 2, 1, 1, 1, 0, 2, 2, 2, 2, 1, 2, 3, 0, 2, 1, 2, 2, 3, 1, 1, 1,
       2, 1, 1, 1, 2, 1, 2, 2, 2, 1, 3, 3, 2, 2, 1, 1, 0, 1, 0, 2, 2, 2,
       2, 0, 0, 2, 0, 1, 2, 2, 1, 1, 3, 1, 1, 1, 2, 2, 3, 0, 2, 1, 2, 2,
       1, 2, 3, 1, 2, 0, 1, 2, 2, 1, 2, 2, 1, 2, 1, 3, 1, 2, 1, 1, 1, 2,
       1, 3, 2, 1, 0, 0, 1, 2, 0, 1, 2, 2, 1, 0, 1, 1, 1, 1, 1, 3, 1, 0,
       1, 2, 2, 0, 0, 3, 1, 1, 2, 1, 2, 1, 1, 0, 0, 2, 3, 0, 1, 2, 2, 1,
       0, 1, 1, 2, 0, 1, 1, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 2, 2,
       1, 1, 0, 2, 2, 1, 1, 2, 2, 1, 3, 1, 0, 1, 1, 2, 2, 1, 3, 0, 2, 1,
       1, 2, 1, 1, 3, 1, 3, 1, 1, 1, 2, 0])
```

In [187…

```
y_train.head(20)
```

Out[187…

```
140          Excellent
92           Excellent
113          Excellent
124          Excellent
```

```
14                 Good
223           Excellent
285           Excellent
7             Excellent
268                Good
146           Excellent
38            Excellent
166             Average
196                Good
220           Excellent
34                 Good
89     Not Satisfactory
87                 Good
292    Not Satisfactory
106           Excellent
144             Average
Name: Academic Performance, dtype: object
```

Here we see our labels are encoded according to the following:

1. **Excellent** - **1**
2. **Good** - **2**
3. **Average** - **0**
4. **Not Satisfactory** - **3**

Let's finally train the Genetic Algorithm using `TPOTClassifier`. We are currently using 15 `generations`, 100 `population_size` and 150 `offspring_size`.

In [191…

```python
from tpot import TPOTClassifier


tpot = TPOTClassifier(generations=15, population_size=100, offspring_size=150,
                      verbosity=2, early_stop=8, cv = 10, scoring = 'accuracy
                      random_state=42)

tpot.fit(X_train, y_train_n)
print(tpot.score(X_test, y_test_n))
tpot.export('tpot_digits_pipeline_uni.py')
```

```
Generation 1 - Current best internal CV score: 0.7523809523809524

Generation 2 - Current best internal CV score: 0.7523809523809524

Generation 3 - Current best internal CV score: 0.7523809523809524

Generation 4 - Current best internal CV score: 0.7523809523809524

Generation 5 - Current best internal CV score: 0.7619047619047619

Generation 6 - Current best internal CV score: 0.7619047619047619

Generation 7 - Current best internal CV score: 0.7619047619047619

Generation 8 - Current best internal CV score: 0.7714285714285715

Generation 9 - Current best internal CV score: 0.7714285714285715

Generation 10 - Current best internal CV score: 0.7714285714285715
```

```
Generation 11 - Current best internal CV score: 0.7714285714285715

Generation 12 - Current best internal CV score: 0.7714285714285715

Generation 13 - Current best internal CV score: 0.7714285714285715

Generation 14 - Current best internal CV score: 0.7714285714285715

Generation 15 - Current best internal CV score: 0.7714285714285715

Best pipeline: ExtraTreesClassifier(ExtraTreesClassifier(input_matrix, bootstr
ap=False, criterion=gini, max_features=0.05, min_samples_leaf=10, min_samples_
split=12, n_estimators=100), bootstrap=False, criterion=entropy, max_features=
0.45, min_samples_leaf=1, min_samples_split=19, n_estimators=100)
0.7912087912087912
```

Genetic algorithm showed us that the most optimized algorithm is the `ExtraTreeClassifier` with the following parameter :

`ExtraTreesClassifier(ExtraTreesClassifier(input_matrix, bootstrap=False, criterion=gini, max_features=0.05, min_samples_leaf=10, min_samples_split=12, n_estimators=100), bootstrap=False, criterion=entropy, max_features=0.45, min_samples_leaf=1, min_samples_split=19, n_estimators=100) 0.7912087912087912`

Let's fit this algorithm to our dataset and check the training score.

In [204…

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline, make_union
from tpot.builtins import StackingEstimator
from tpot.export_utils import set_param_recursive


# Average CV score on the training set was: 0.7714285714285715
exported_pipeline = make_pipeline(
    StackingEstimator(estimator=ExtraTreesClassifier(bootstrap=False, criteri
    ExtraTreesClassifier(bootstrap=False, criterion="entropy", max_features=0
)
# Fix random state for all the steps in exported pipeline
set_param_recursive(exported_pipeline.steps, 'random_state', 42)

exported_pipeline.fit(X_train, y_train_n)
results = exported_pipeline.predict(X_test)

score = exported_pipeline.score(X_train, y_train_n)
print("Training score: ", score)
```

```
Training score:  0.8904761904761904
```

Let's check the accuracy on the test set and check the confusion matrix, precision, recall and f1 scores.

In [206…

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


conf_mat = confusion_matrix(y_test_n, results)
class_report = classification_report(y_test_n, results)


print("Accuracy:", metrics.accuracy_score(y_test_n, results))

print(conf_mat)
print(class_report)
```

```
Accuracy: 0.7912087912087912
[[ 4  1  0  1]
 [ 1 46  4  0]
 [ 2  8 18  0]
 [ 1  0  1  4]]
              precision    recall  f1-score   support

           0       0.50      0.67      0.57         6
           1       0.84      0.90      0.87        51
           2       0.78      0.64      0.71        28
           3       0.80      0.67      0.73         6

    accuracy                           0.79        91
   macro avg       0.73      0.72      0.72        91
weighted avg       0.80      0.79      0.79        91
```

Finally, let's perform `KFold` cross validation.

In [208…

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


cv_ga = KFold(n_splits=10, shuffle=True, random_state=42)

scores = cross_val_score(exported_pipeline, X_train, y_train_n, cv=cv_ga, sco
print('Training Accuracy On KFold Cross Validation: %.3f (%.3f)' % (np.mean(sc

scores = cross_val_score(exported_pipeline, X_test, y_test_n, cv=cv_ga, scori
print('Testing Accuracy On KFold Cross Validation: %.3f (%.3f)' % (np.mean(sc
```

```
Training Accuracy On KFold Cross Validation: 0.743 (0.065)
Testing Accuracy On KFold Cross Validation: 0.758 (0.120)
```

This model givess us a 76% accuracy on KFold cross validation.

In [ ]: