WMU102 ASSIGNMENT

GROUP MEMBERS

G2

1. Yeemadalee Roseneena (140694)
2. Nor Islahiah Binti Muhammed (149891)
3. Annis Afifah Mohd Azhar (145219)
4. Yasmin Binti Rafid (143889)

DUE DATE

Original: Thursday, 10 June 2021

New : Wednesday, 16 June 2021, 0600 AM

INSTRUCTION

Steps:

1. Pick a domain of your interest. Then, you may try the following: a. Pick a problem which already exists in your domain. b. Find a client / stakeholder and understand their domain problem.
2. Define the problem statement.
3. Detail a plan on how to:

   a. Find / collect data (you may collect your own or use any available dataset). Some notable dataset repositories are as follows:

   ```
   i. Kaggle (https://www.kaggle.com/datasets)

   ii. UCI Dataset (https://archive.ics.uci.edu/ml/index.php)

   iii. Google Dataset (https://datasetsearch.research.google.com/)
   ```

   b. Clean the data / perform exploratory data analysis (EDA) to get valuable insights, which includes: i. Data preprocessing (data cleaning) ii. Data visualisation

   c. Pick several machine learning methods which you think is suitable to solve your problem.

   d. Perform the training.

   e. Evaluate your model using performance metrics.

1. Pitch your solution to your client / stakeholder (in this case, it's your panel) and get them onboard. Revise your plan if necessary. Then pitch again.
2. Execute your plan.
3. Document everything in your Jupyter / Colaboratory notebook using markdown and figures.

Example notebook: https://www.kaggle.com/nadintamer/titanic-survival-predictions-beginner/comments#Titanic-Survival-Predictions-(Beginner)

STEP 1

Chosen domain: Education

STEP 2

Problem statement: Student's grade prediction

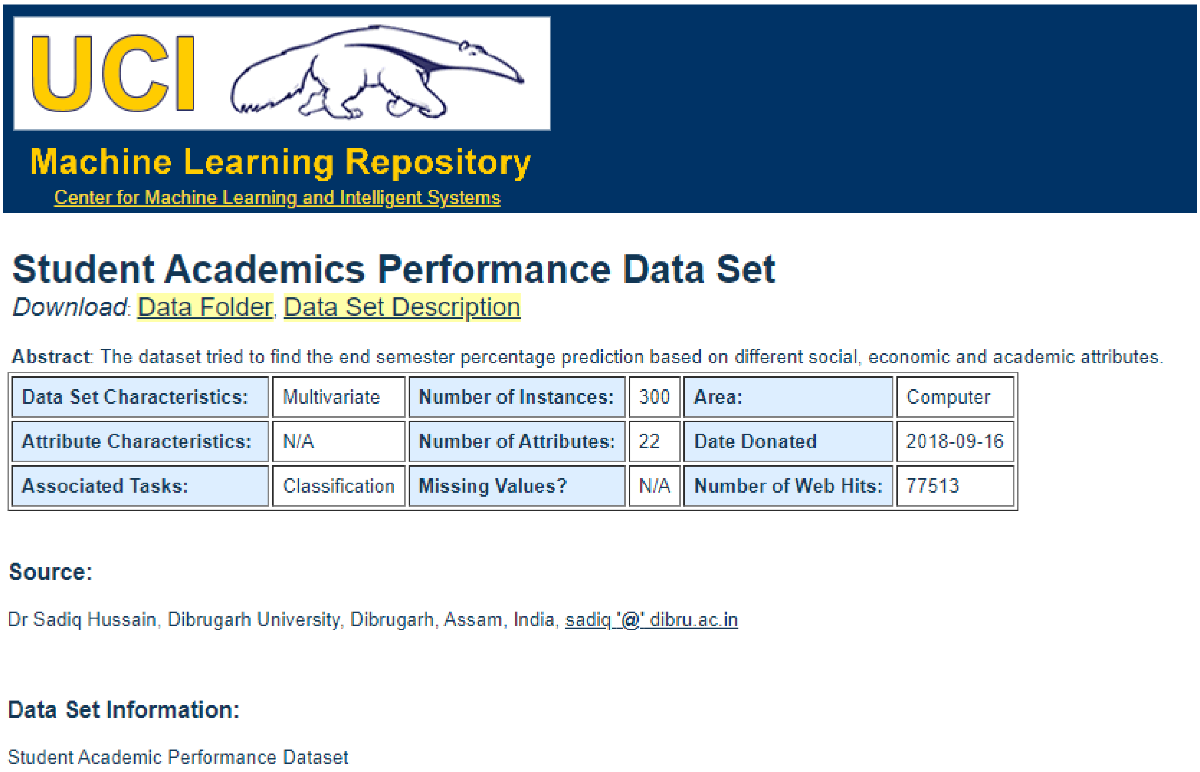
STEP 3 (a)

We choose to use available dataset.

Chosen dataset: https://archive.ics.uci.edu/ml/datasets/Student+Academics+Performance#

We chose this one because:

- It's neither too large nor too small (for people like us).
- Seemingly to be the only dataset we could kind of understand the content at the moment



UCI

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Student Academics Performance Data Set

Download: Data Folder, Data Set Description

Abstract: The dataset tried to find the end semester percentage prediction based on different social, economic and academic attributes.

Data Set Characteristics:	Multivariate	**Number of Instances:**	300	**Area:**	Computer
Attribute Characteristics:	N/A	**Number of Attributes:**	22	**Date Donated**	2018-09-16
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	77513

Source:

Dr Sadiq Hussain, Dibrugarh University, Dibrugarh, Assam, India, sadiq '@' dibru.ac.in

Data Set Information:

Student Academic Performance Dataset

STEP 3 (b)(i)

We follow the steps in Titanic Survival Predictions (Beginner) for reference.

In [1]:

```
#1) Import Necessary Libraries
#
# First, we need to import several Python libraries
# such as numpy, pandas, matplotlib and seaborn.
#


#data analysis libraries
import numpy as np
import pandas as pd

#visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

#ignore warnings
```

```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#2) Read in and Explore the Data
#
# It's time to read in our training and testing data using pd.read_csv, and
# take a first look at the training data using the describe() function.
#

# The file is in arff format. So we used Weka to convert it to csv format.

#import train and test CSV files
train = pd.read_csv('D:\\01. nim\\04. belajar\\USM\\THN_2\\Sem_2\\WMU102\\tugasan\\S
test = pd.read_csv('D:\\01. nim\\04. belajar\\USM\\THN_2\\Sem_2\\WMU102\\tugasan\\Sa

#take a look at the training data
print(train.describe(include='all'))
```

```
        ge  cst   tnp   twp  iap   esp  arr         ms   ls    as  ...   fq  \
count  131  131   131   131  131   131  131        131  131   131  ...  131
unique   2    5     4     4    4     4    2          1    2     2  ...    6
top      M  OBC  Good  Good   Vg  Good    N  Unmarried    V  Paid  ...   Um
freq    72   57    59    65   63    54   78        131   92    76  ...   40

        mq       fo         mo     nf       sh    ss   me     tt   atd
count  131      131        131    131      131   131  131    131   131
unique   6        5          5      3        3     2    4      3     3
top     Um  Service  Housewife  Large  Average  Govt  Eng  Small  Good
freq    52       38        115     58       59    91   62     78    56

[4 rows x 22 columns]
```

From above codes, we can see that we have 131 records in the dataset, the number of features, number of data inside each feature, the number of unique data inside each feature, top unique data for each feature, and the frequency of the top unique data for each feature.

In [3]:

```
#3) Data Analysis
#
# We're going to consider the features in the dataset and how complete they are.
#


#get a list of the features within the dataset
print(train.columns, '\n')

#see a sample of the dataset to get an idea of the variables
print(train.sample(5), '\n')
```

```
Index(['ge', 'cst', 'tnp', 'twp', 'iap', 'esp', 'arr', 'ms', 'ls', 'as', 'fmi',
       'fs', 'fq', 'mq', 'fo', 'mo', 'nf', 'sh', 'ss', 'me', 'tt', 'atd'],
      dtype='object')

   ge  cst   tnp twp   iap   esp arr         ms ls    as  ...      fq      mq  \
48  F  OBC  Good  Vg  Pass  Pass   Y  Unmarried  V  Free  ...      12      10
10  M   ST    Vg  Vg  Good    Vg   N  Unmarried  V  Paid  ...      Pg      Pg
51  F    G  Best  Vg  Best  Best   N  Unmarried  V  Free  ...  Degree      10
6   F  OBC  Good  Vg  Good  Good   N  Unmarried  V  Paid  ...      12  Degree
64  F  OBC    Vg  Vg    Vg    Vg   N  Unmarried  V  Paid  ...      12      Um

          fo         mo       nf       sh       ss   me       tt   atd
48    Others  Housewife    Large  Average     Govt  Asm  Average  Good
10   Retired    Retired    Large     Good  Private  Eng    Large  Good
51  Business  Housewife  Average     Good  Private  Eng    Small  Good
6    Service    Service  Average     Poor     Govt  Asm    Small  Good
```

```
64    Service  Housewife     Large      Good  Private  Eng     Small   Good

[5 rows x 22 columns]
```

Each columns are not labelled with their full names. There is no comments or further explanation anywhere. Looking at the content, however, we could kind of guess some of them.

Edited: Turns out, the full name for each features are available in Indonesian Journal of Electrical Engineering and Computer Science. 2018; Vol. 9, No. 2. February. pp. 447~459. The PDF is freely available. some of the original guesses were wrong. Example: esp is not the nickname for Espanol (which refers to Spain language).

452 ❐ ISSN: 2502-4752

Table 1: Dataset Description

Attribute	Description	Values
GE	Gender	(Male, Female)
CST	Caste	(General,SC,ST,OBC,MOBC)
TNP	Class X Percentage	(Best, Very Good, Good, Pass, Fail) If percentage >=80 then Best If percentage >= 60 but less than 80 then Very Good If percentage >= 45 but less than 60 then Good If Percentage >= 30 but less than 45 then Pass If Percentage < 30 then Fail
TWP	Class XII Percentage	(Best, Very Good, Good, Pass, Fail) Same as TNP
IAP	Internal Assessment Percentage	(Best, Very Good, Good, Pass, Fail) Same as TNP
ESP	End Semester Percentage	(Best, Very Good, Good, Pass, Fail) Same as TNP
ARR	Whether the student has back or arrear papers	(Yes, No)
MS	Marital Status	(Married, Unmarried)
LS	Lived in Town or Village	(Town, Village)
AS	Admission Category	(Free, Paid)
FMI	Family Monthly Income (in INR)	(Very High, High, Above Medium, Medium, Low) If FMI >= 30000 then Very High If FMI >= 20000 but less than 30000 then High If FMI >= 10000 but less than 20000 then Above Medium If FMI >= 5000 but less than 10000 then Medium If FMI is less than 5000 then Low The figures are expressed in INR.
FS	Family Size	(Large, Average, Small) If FS > 12 then Large If FS >= 6 but less than 12 then Average If FS < 6 then Small
FQ	Father Qualification	(IL, UM, 10, 12 , Degree, PG) IL= Illiterate UM= Under Class X
MQ	Mother Qualification	(IL, UM, 10, 12 , Degree, PG) IL= Illiterate UM= Under Class X
FO	Father Occupation	(Service, Business, Retired, Farmer, Others)
MO	Mother Occupation	(Service, Business, Retired, Farmer, Others)
NF	Number of Friends	(Large, Average, Small) Same as Family Size
SH	Study Hours	(Good, Average, Poor) >= 6 hours Good >= 4 hours Average < 2 hours Poor
SS	Student School attended at Class X level	(Govt., Private)
ME	Medium	(Eng,Asm,Hin,Ben)
TT	Home to College Travel Time	(Large, Average, Small) >= 2 hours Large >=1 hours Average < 1 hour Small
ATD	Class Attendance Percentage	(Good, Average, Poor) If percentage >= 80 then Good If percentage >= 60 but less than 80 then Average If Percentage < 60 then poor

Descriptions of some of the attributes of the dataset

Numerical Features: None Categorical Features: All of them Alphanumeric Features: None

What are the data types for each feature?

All of them are type string.

Now that we have an idea of what kinds of features we're working with, we can see how much information we have about each of them.

In [4]:

```
#3) Data Analysis - continued
```

```
#
# We're going to consider the features in the dataset and how complete they are.
#


print(train.describe(include = 'all'))
```

```
         ge  cst   tnp   twp  iap   esp  arr         ms   ls    as  ...   fq  \
count   131  131   131   131  131   131  131        131  131   131  ...  131
unique    2    5     4     4    4     4    2          1    2     2  ...    6
top       M  OBC  Good  Good   Vg  Good    N  Unmarried    V  Paid  ...   Um
freq     72   57    59    65   63    54   78        131   92    76  ...   40

         mq       fo         mo     nf       sh    ss   me     tt   atd
count   131      131        131    131      131   131  131    131   131
unique    6        5          5      3        3     2    4      3     3
top      Um  Service  Housewife  Large  Average  Govt  Eng  Small  Good
freq     52       38        115     58       59    91   62     78    56

[4 rows x 22 columns]
```

Some Observations:

- There are a total of 131 students in our training set.
- No features with missing values.

In [5]:

```
#3) Data Analysis - continued
#
# We're going to consider the features in the dataset and how complete they are.
#


#check for any other unusable values
print(pd.isnull(train).sum(), '\n')
```

```
ge     0
cst    0
tnp    0
twp    0
iap    0
esp    0
arr    0
ms     0
ls     0
as     0
fmi    0
fs     0
fq     0
mq     0
fo     0
mo     0
nf     0
sh     0
ss     0
me     0
tt     0
atd    0
dtype: int64
```

No NaN values exist.

Some Predictions:

- Lived in Town or Village: Students living in a town are more likely to have better grades.
- Family Monthly Income (in INR): Students with above medium family income are more likely to have better grades.
- Family Size: Students from a family of Small size are more likely to have better grades.
- Study Hours: Students who study for a Good length of time are more likely to have better grades
- Home to College Travel Time: Students who need a short time to travel to school are more likely to have better grades.
- Class Attendance Percentage: Students with good attendance are more likely to have better grades.

In [6]:

```
train.ls.count()
train.groupby('ls').count()
```

Out[6]:

ls	ge	cst	tnp	twp	iap	esp	arr	ms	as	fmi	...	fq	mq	fo	mo	nf	sh	ss	me	tt	atd
T	39	39	39	39	39	39	39	39	39	39	...	39	39	39	39	39	39	39	39	39	39
V	92	92	92	92	92	92	92	92	92	92	...	92	92	92	92	92	92	92	92	92	92

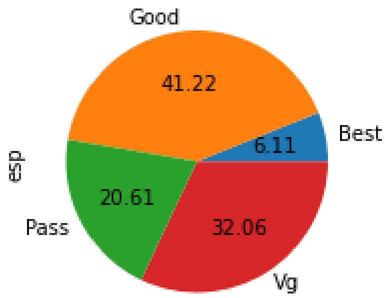
2 rows × 21 columns

In [7]:

```
# 4) Data Visualization - Also known as Step 3b(ii) in the WMU102 project instructio
#
# It's time to visualize our data so we can see whether our predictions were accurat
#

# Before that, let's check the grade.
# End Semester Percentage (esp)

print('Total of Students with esp(Best)      :', train['esp'].tolist().count('Best')
print('Total of Students with esp(Very Good) :', train['esp'].tolist().count('Vg'),
print('Total of Students with esp(Good)      :', train['esp'].tolist().count('Good')
print('Total of Students with esp(Pass)      :', train['esp'].tolist().count('Pass')
print('Total of Students with esp(Fail)      :', train['esp'].tolist().count('Fail')

train.groupby('esp')['esp'].count().plot.pie(autopct='%.2f',figsize=(3,3))
```

```
Total of Students with esp(Best)      : 8

Total of Students with esp(Very Good) : 42

Total of Students with esp(Good)      : 54

Total of Students with esp(Pass)      : 27

Total of Students with esp(Fail)      : 0
```

Out[7]: `<AxesSubplot:ylabel='esp'>`



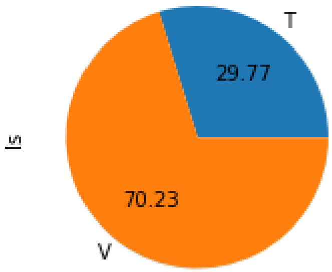
So, no one failed. Good for them.

In [8]:

```
# Lived in Town or Village (ls) Feature
# Prediction: Students living in a town are more likely to have better grades

print('Total of Students with ls(Town)    :', train['ls'].tolist().count('T'), '\n')

print('Total of Students with ls(Village) :', train['ls'].tolist().count('V'), '\n')

train.groupby('ls')['ls'].count().plot.pie(autopct='%.2f',figsize=(3,3))
```

```
Total of Students with ls(Town)    : 39

Total of Students with ls(Village) : 92
```

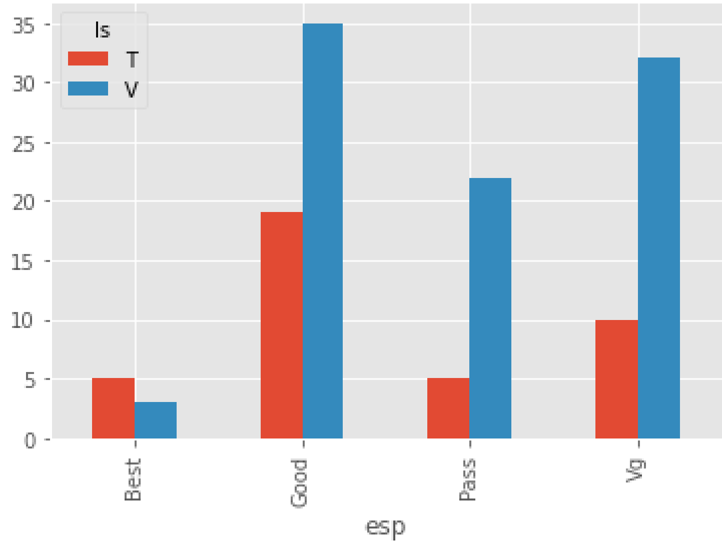
Out[8]: <AxesSubplot:ylabel='ls'>



So, the number of students who live in a village is more than 2 times higher than those who live in town.

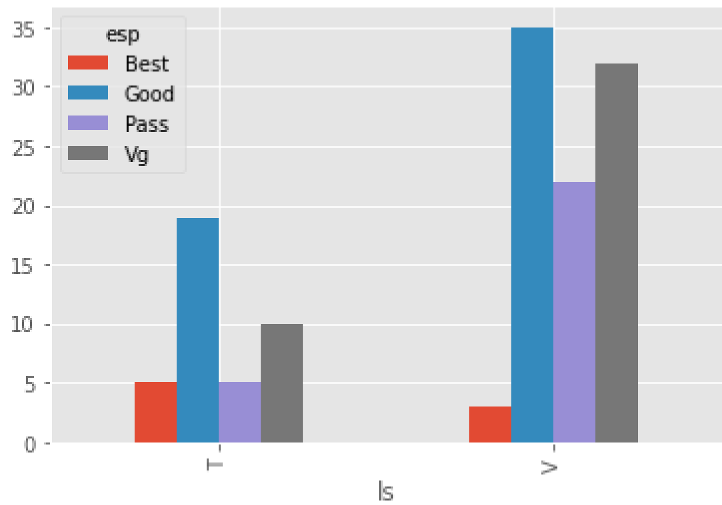
In [9]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['esp', 'ls'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [10]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['ls', 'esp'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [11]:

```
#print percentages of students living in Town vs. those living in Village that get B
print('% of Students living in Town with Best result          :',
      train['esp'][train['ls'] == 'T'].value_counts(normalize = True)['Best']*100)

print('% of Students living in Village with Best result       :',
      train['esp'][train['ls'] == 'V'].value_counts(normalize = True)['Best']*100, '


#print percentages of students living in Town vs. those living in Village that get V
print('% of Students living in Town with Very Good result     :',
      train['esp'][train['ls'] == 'T'].value_counts(normalize = True)['Vg']*100)

print('% of Students living in Village with Very Good result :',
      train['esp'][train['ls'] == 'V'].value_counts(normalize = True)['Vg']*100, '\n


#print percentages of students living in Town vs. those living in Village that get G
print('% of Students living in Town with Good result          :',
```

```
      train['esp'][train['ls'] == 'T'].value_counts(normalize = True)['Good']*100)

print('% of Students living in Village with Good result      :',
      train['esp'][train['ls'] == 'V'].value_counts(normalize = True)['Good']*100, '


#print percentages of students living in Town vs. those living in Village that get P
print('% of Students living in Town with Passable result         :',
      train['esp'][train['ls'] == 'T'].value_counts(normalize = True)['Pass']*100)

print('% of Students living in Village with Passable result      :',
      train['esp'][train['ls'] == 'V'].value_counts(normalize = True)['Pass']*100)
```

```
% of Students living in Town with Best result         : 12.82051282051282
% of Students living in Village with Best result      : 3.260869565217391

% of Students living in Town with Very Good result    : 25.64102564102564
% of Students living in Village with Very Good result : 34.78260869565217

% of Students living in Town with Good result         : 48.717948717948715
% of Students living in Village with Good result      : 38.04347826086957

% of Students living in Town with Passable result         : 12.82051282051282
% of Students living in Village with Passable result      : 23.91304347826087
```

LEGEND

ls = Lived in Town or Village T = Town V = Village

esp = End Semester Percentage Best = Best Vg = Very Good Good = Good Pass = Pass

PREDICTION RESULT

As predicted, even though the number of students who live in Town is more than 2 times lower than those who live in Village, the group has a higher percentage of those with Best result and a lower percentage of those with less than Good result compared to the other group.

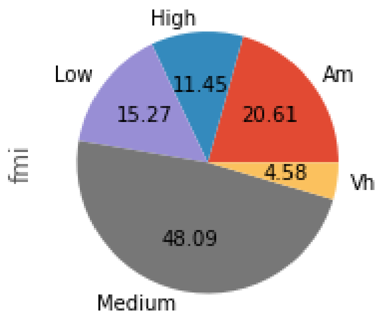
Based on the result, Nadin Tamer would probably say The Lived in Town or Village (ls) feature is essential in our predictions. But, honestly, I have no idea what he's talking about.

In [12]:

```
# Family Monthly Income (fmi) Feature
# Prediction: Students with above medium family income are more likely to have bette


print('Total of Students with fmi(Very High)    :', train['fmi'].tolist().count('Vh'
print('Total of Students with fmi(High)         :', train['fmi'].tolist().count('Hig
print('Total of Students with fmi(Above Medium) :', train['fmi'].tolist().count('Am'
print('Total of Students with fmi(Medium)       :', train['fmi'].tolist().count('Med
print('Total of Students with fmi(Low)          :', train['fmi'].tolist().count('Low

train.groupby('fmi')['fmi'].count().plot.pie(autopct='%.2f',figsize=(3,3))
```

```
Total of Students with fmi(Very High)    : 6

Total of Students with fmi(High)         : 15

Total of Students with fmi(Above Medium) : 27

Total of Students with fmi(Medium)       : 63

Total of Students with fmi(Low)          : 20
```

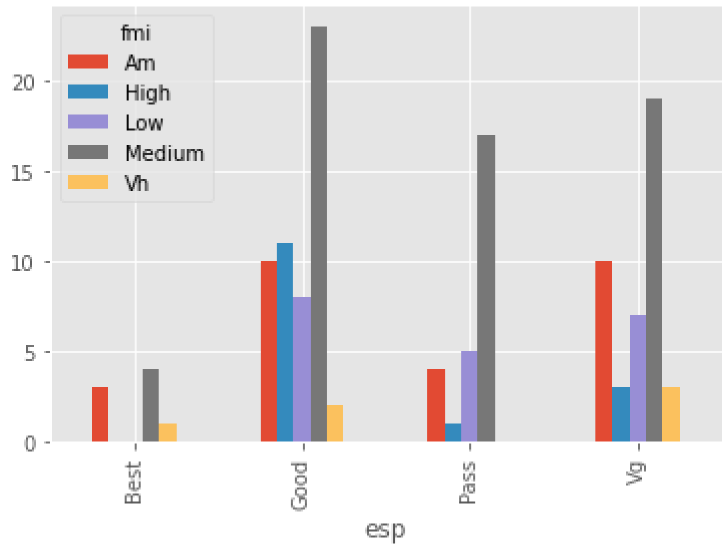
Out[12]: <AxesSubplot:ylabel='fmi'>



So, most students are from a family with medium income.

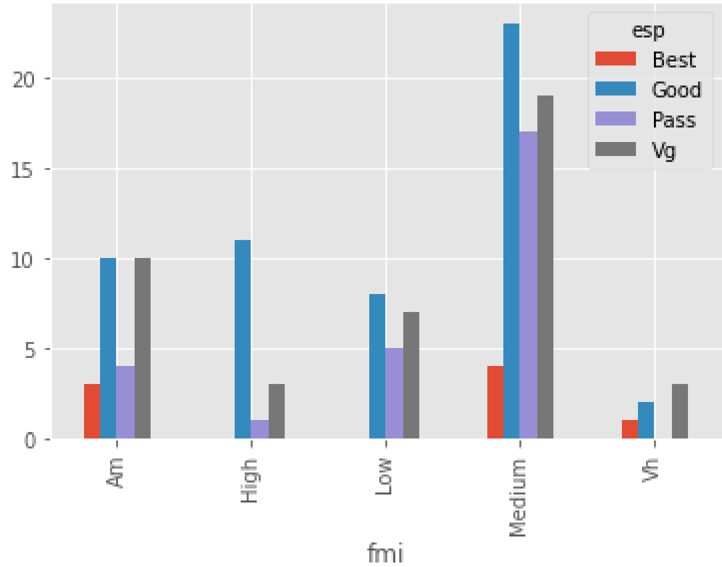
In [13]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['esp', 'fmi'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [14]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['fmi', 'esp'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [15]:

```
#print percentages of students with Best result
print('% of Students with fmi(Very High) with Best result    :',
      train['esp'][train['fmi'] == 'Vh'].value_counts(normalize = True)['Best']*100)
print('% of Students with fmi(Above Medium) with Best result :',
      train['esp'][train['fmi'] == 'Am'].value_counts(normalize = True)['Best']*100)
print('% of Students with fmi(Medium) with Best result       :',
      train['esp'][train['fmi'] == 'Medium'].value_counts(normalize = True)['Best']*


#print percentages of students with Very Good result
print('% of Students with fmi(Very High) with Very Good result    :',
      train['esp'][train['fmi'] == 'Vh'].value_counts(normalize = True)['Vg']*100)
print('% of Students with fmi(High) with Very Good result         :',
      train['esp'][train['fmi'] == 'High'].value_counts(normalize = True)['Vg']*100)
print('% of Students with fmi(Above Medium) with Very Good result :',
      train['esp'][train['fmi'] == 'Am'].value_counts(normalize = True)['Vg']*100)
print('% of Students with fmi(Medium) with Very Good result       :',
      train['esp'][train['fmi'] == 'Medium'].value_counts(normalize = True)['Vg']*10
print('% of Students with fmi(Low) with Very Good result          :',
      train['esp'][train['fmi'] == 'Low'].value_counts(normalize = True)['Vg']*100,


#print percentages of students with Good result
print('% of Students with fmi(Very High) with Good result    :',
      train['esp'][train['fmi'] == 'Vh'].value_counts(normalize = True)['Good']*100)
print('% of Students with fmi(High) with Good result         :',
      train['esp'][train['fmi'] == 'High'].value_counts(normalize = True)['Good']*10
print('% of Students with fmi(Above Medium) with Good result :',
      train['esp'][train['fmi'] == 'Am'].value_counts(normalize = True)['Good']*100)
print('% of Students with fmi(Medium) with Good result       :',
      train['esp'][train['fmi'] == 'Medium'].value_counts(normalize = True)['Good']*
print('% of Students with fmi(Low) with Good result          :',
      train['esp'][train['fmi'] == 'Low'].value_counts(normalize = True)['Good']*100


#print percentages of students with Passable result
print('% of Students with fmi(High) with Passable result         :',
      train['esp'][train['fmi'] == 'High'].value_counts(normalize = True)['Pass']*10
print('% of Students with fmi(Above Medium) with Passable result :',
      train['esp'][train['fmi'] == 'Am'].value_counts(normalize = True)['Pass']*100)
print('% of Students with fmi(Medium) with Passable result       :',
      train['esp'][train['fmi'] == 'Medium'].value_counts(normalize = True)['Pass']*
print('% of Students with fmi(Low) with Passable result          :',
      train['esp'][train['fmi'] == 'Low'].value_counts(normalize = True)['Pass']*100
```

```
% of Students with fmi(Very High) with Best result    : 16.666666666666664
% of Students with fmi(Above Medium) with Best result : 11.11111111111111
% of Students with fmi(Medium) with Best result       : 6.349206349206349

% of Students with fmi(Very High) with Very Good result    : 50.0
% of Students with fmi(High) with Very Good result         : 20.0
% of Students with fmi(Above Medium) with Very Good result : 37.03703703703704
% of Students with fmi(Medium) with Very Good result       : 30.158730158730158
% of Students with fmi(Low) with Very Good result          : 35.0

% of Students with fmi(Very High) with Good result    : 33.33333333333333
% of Students with fmi(High) with Good result         : 73.33333333333333
% of Students with fmi(Above Medium) with Good result : 37.03703703703704
% of Students with fmi(Medium) with Good result       : 36.507936507936506
% of Students with fmi(Low) with Good result          : 40.0

% of Students with fmi(High) with Passable result         : 6.666666666666667
% of Students with fmi(Above Medium) with Passable result : 14.814814814814813
% of Students with fmi(Medium) with Passable result       : 26.984126984126984
% of Students with fmi(Low) with Passable result          : 25.0
```

LEGEND

fmi = Family Monthly Income (in INR) Vh = Very High High = High Am = Above Medium Medium = Medium Low = Low

esp = End Semester Percentage Best = Best Vg = Very Good Good = Good Pass = Pass

PREDICTION RESULT

As predicted, students from above medium (which includes Above Medium, High, and Very High) have better grades in general.

Those from fmi(Very High), don't even have grade below than Good.

Those from fmi(High), have a much lower frequency for grade below than Good.

Those from fmi(Above Medium), have a much lower frequency for grade below than Good.

Those from fmi(Medium) and fmi(Low), have an about similar frequency for grade below than Good.

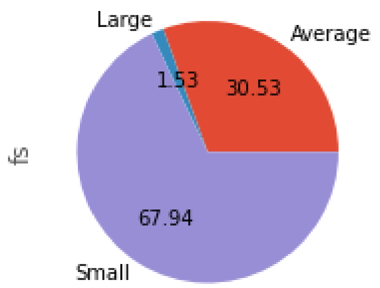
So, The Family Monthly Income (fmi) feature is essential in our predictions. Probably.

In [16]:

```
# Family Size (fs) Feature
# Prediction: Students from a family of Small size are more likely to have better gr


print('Total of Students with fs(Large)    :', train['fs'].tolist().count('Large'),
print('Total of Students with fs(Average)  :', train['fs'].tolist().count('Average')
print('Total of Students with fs(Small)    :', train['fs'].tolist().count('Small'),

train.groupby('fs')['fs'].count().plot.pie(autopct='%.2f',figsize=(3,3))
```

```
Total of Students with fs(Large)    : 2

Total of Students with fs(Average)  : 40

Total of Students with fs(Small)    : 89
```

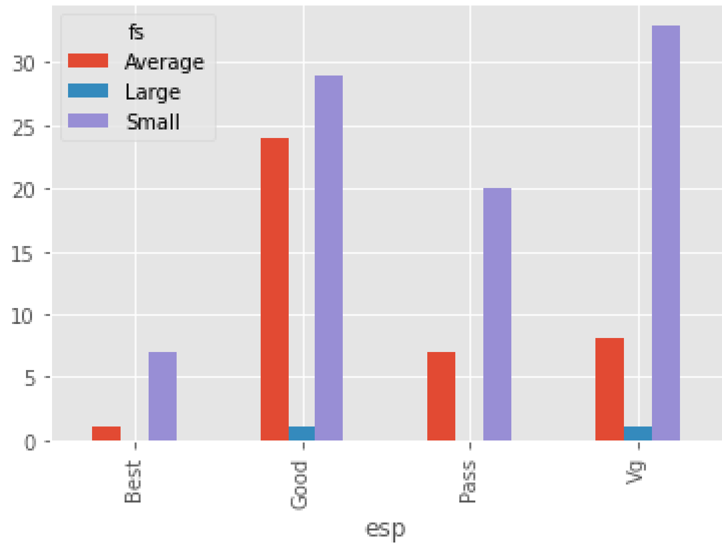
Out[16]: <AxesSubplot:ylabel='fs'>



So, about two third of the students come from a family of Small size. And, those from a family of Large size are less than 2 percent.

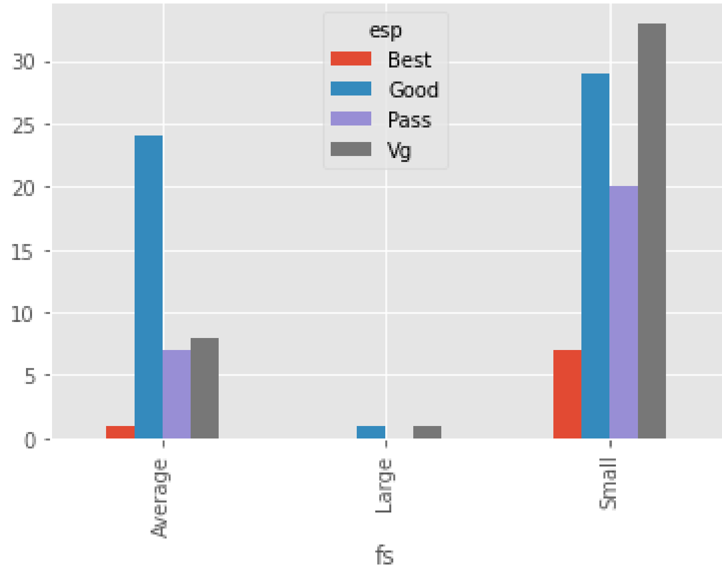
In [17]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['esp', 'fs'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [18]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['fs', 'esp'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [19]:

```
#print percentages of students with Best result
print('% of Students with fs(Small) with Best result         :',
      train['esp'][train['fs'] == 'Small'].value_counts(normalize = True)['Best']*10

print('% of Students with fs(Average) with Best result       :',
      train['esp'][train['fs'] == 'Average'].value_counts(normalize = True)['Best']*


#print percentages of students with Very Good result
print('% of Students with fs(Small) with Very Good result    :',
      train['esp'][train['fs'] == 'Small'].value_counts(normalize = True)['Vg']*100)

print('% of Students with fs(Average) with Very Good result  :',
      train['esp'][train['fs'] == 'Average'].value_counts(normalize = True)['Vg']*10


#print percentages of students with Good result
print('% of Students with fs(Small) with Good result         :',
      train['esp'][train['fs'] == 'Small'].value_counts(normalize = True)['Good']*10

print('% of Students with fs(Average) with Good result       :',
      train['esp'][train['fs'] == 'Average'].value_counts(normalize = True)['Good']*


#print percentages of students with Passable result
print('% of Students with fs(Small) with Passable result     :',
      train['esp'][train['fs'] == 'Small'].value_counts(normalize = True)['Pass']*10

print('% of Students with fs(Average) with Passable result   :',
      train['esp'][train['fs'] == 'Average'].value_counts(normalize = True)['Pass']*
```

```
% of Students with fs(Small) with Best result         : 7.865168539325842
% of Students with fs(Average) with Best result       : 2.5

% of Students with fs(Small) with Very Good result    : 37.07865168539326
% of Students with fs(Average) with Very Good result  : 20.0

% of Students with fs(Small) with Good result         : 32.58426966292135
% of Students with fs(Average) with Good result       : 60.0

% of Students with fs(Small) with Passable result     : 22.47191011235955
% of Students with fs(Average) with Passable result   : 17.5
```

LEGEND

fs = Family Size Large = Large Average = Average Small = Small

esp = End Semester Percentage Best = Best Vg = Very Good Good = Good Pass = Pass

PREDICTION RESULT

The number of those from fs(Large) is too low to compare with others.

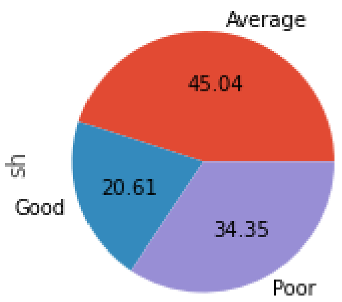
But, if we compare between fs(Small) and fs(Average), unexpectedly, students with fs(Average) have better result based on the fact that the frequency of grade below than Good for the latter is much higher than the other group.

In [20]:

```
# Study Hours (sh) Feature
# Prediction: Students who study for a Good length of time are more likely to have b

print('Total of Students with sh(Good)    :', train['sh'].tolist().count('Good'), '\
print('Total of Students with sh(Average) :', train['sh'].tolist().count('Average'),
print('Total of Students with sh(Poor)    :', train['sh'].tolist().count('Poor'), '\

train.groupby('sh')['sh'].count().plot.pie(autopct='%.2f',figsize=(3,3))
```

```
Total of Students with sh(Good)    : 27

Total of Students with sh(Average) : 59

Total of Students with sh(Poor)    : 45
```

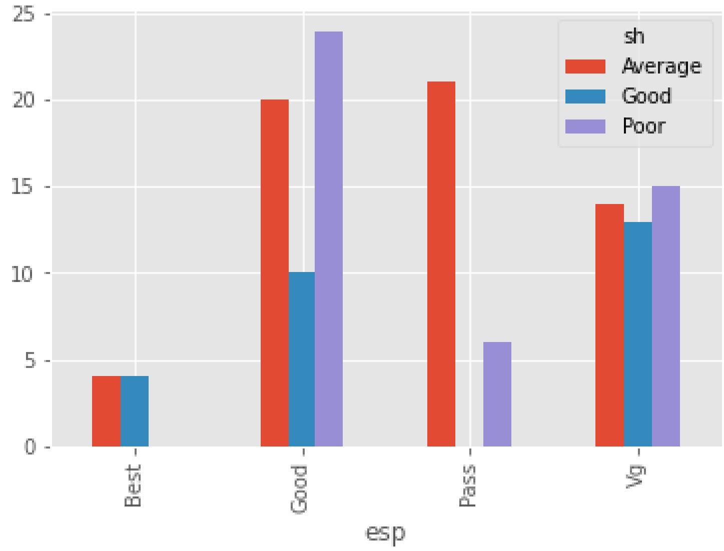
Out[20]: <AxesSubplot:ylabel='sh'>



So, it's pretty even but most of the students study for an Average length of time (<= 4 hours, >= 2 hours).

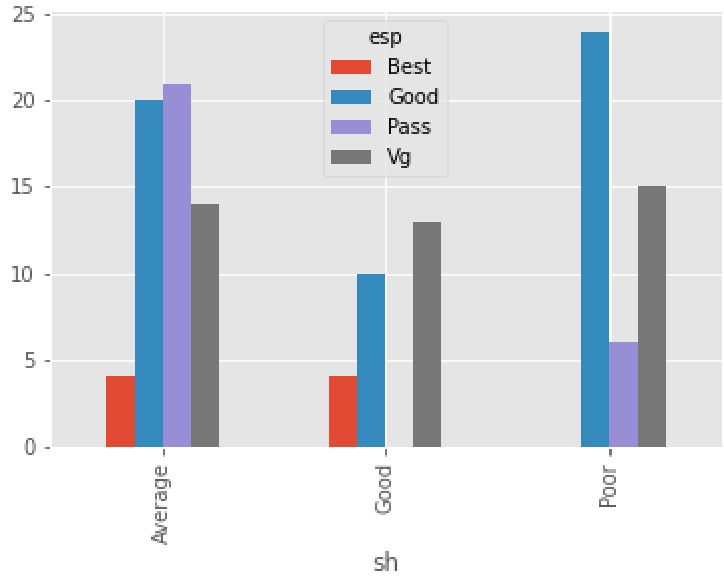
In [21]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['esp', 'sh'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [22]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['sh', 'esp'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [23]:

```
#print percentages of students with Best result
print('% of Students with sh(Good) with Best result    :',
      train['esp'][train['sh'] == 'Good'].value_counts(normalize = True)['Best']*100
print('% of Students with sh(Average) with Best result :',
      train['esp'][train['sh'] == 'Average'].value_counts(normalize = True)['Best']*


#print percentages of students with Very Good result
print('% of Students with sh(Good) with Very Good result    :',
      train['esp'][train['sh'] == 'Good'].value_counts(normalize = True)['Vg']*100)
print('% of Students with sh(Average) with Very Good result :',
      train['esp'][train['sh'] == 'Average'].value_counts(normalize = True)['Vg']*10
print('% of Students with sh(Poor) with Very Good result    :',
      train['esp'][train['sh'] == 'Poor'].value_counts(normalize = True)['Vg']*100,
```

```
#print percentages of students with Good result
print('% of Students with sh(Good) with Good result    :',
      train['esp'][train['sh'] == 'Good'].value_counts(normalize = True)['Good']*100
print('% of Students with sh(Average) with Good result :',
      train['esp'][train['sh'] == 'Average'].value_counts(normalize = True)['Good']*
print('% of Students with sh(Poor) with Good result    :',
      train['esp'][train['sh'] == 'Poor'].value_counts(normalize = True)['Good']*100


#print percentages of students with Passable result
print('% of Students with sh(Average) with Passable result :',
      train['esp'][train['sh'] == 'Average'].value_counts(normalize = True)['Pass']*
print('% of Students with sh(Poor) with Passable result    :',
      train['esp'][train['sh'] == 'Poor'].value_counts(normalize = True)['Pass']*100
```

```
% of Students with sh(Good) with Best result    : 14.814814814814813
% of Students with sh(Average) with Best result : 6.779661016949152

% of Students with sh(Good) with Very Good result    : 48.148148148148145
% of Students with sh(Average) with Very Good result : 23.728813559322035
% of Students with sh(Poor) with Very Good result    : 33.33333333333333

% of Students with sh(Good) with Good result    : 37.03703703703704
% of Students with sh(Average) with Good result : 33.89830508474576
% of Students with sh(Poor) with Good result    : 53.333333333333336

% of Students with sh(Average) with Passable result : 35.59322033898305
% of Students with sh(Poor) with Passable result    : 13.333333333333334
```

LEGEND

sh = Study Hours Good = Good Average = Average Poor = Poor

esp = End Semester Percentage Best = Best Vg = Very Good Good = Good Pass = Pass

PREDICTION RESULT

As expected, those who study for a Good period of time (6 hours or more) have the best results compared to others.

Unexpectedly, however, those who study for a Poor period of time (less than 2 hours) seems to have better result compared to those who study for an Average period of time. Their percentage of grades lower than Good is much lower compared to the former.

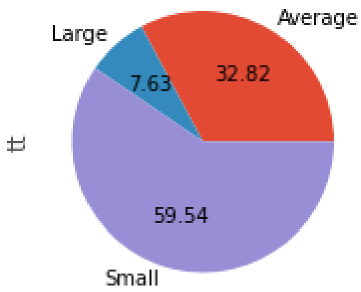
In [24]:

```
# Home to College Travel Time (tt) Feature
# Prediction: Students who need a short time to travel to school are more likely to


print('Total of Students with tt(Large)    :', train['tt'].tolist().count('Large'),
print('Total of Students with tt(Average)  :', train['tt'].tolist().count('Average')
print('Total of Students with tt(Small)    :', train['tt'].tolist().count('Small'),

train.groupby('tt')['tt'].count().plot.pie(autopct='%.2f',figsize=(3,3))
```

```
Total of Students with tt(Large)    : 10

Total of Students with tt(Average)  : 43

Total of Students with tt(Small)    : 78
```

Out[24]:

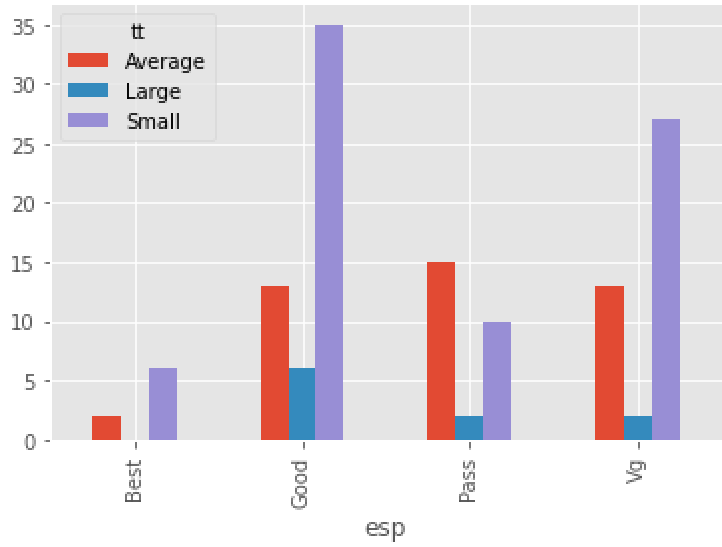
```
<AxesSubplot:ylabel='tt'>
```



So, almost 60% of the students have a Small travel time (less than 1 hour).

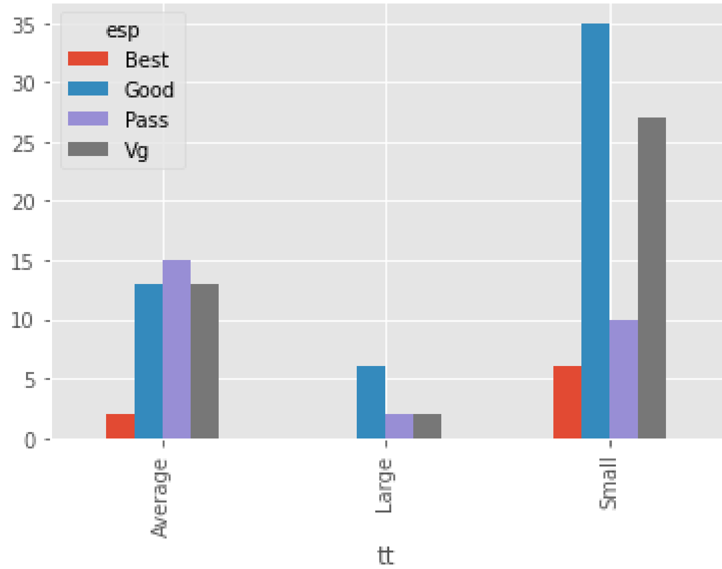
In [25]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['esp', 'tt'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [26]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['tt', 'esp'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [27]:

```
#print percentages of students with Best result
print('% of Students with tt(Average) with Best result  :',
      train['esp'][train['tt'] == 'Average'].value_counts(normalize = True)['Best']*
print('% of Students with tt(Small) with Best result    :',
      train['esp'][train['tt'] == 'Small'].value_counts(normalize = True)['Best']*10

#print percentages of students with Very Good result
print('% of Students with tt(Large) with Very Good result    :',
      train['esp'][train['tt'] == 'Large'].value_counts(normalize = True)['Vg']*100)
print('% of Students with tt(Average) with Very Good result  :',
      train['esp'][train['tt'] == 'Average'].value_counts(normalize = True)['Vg']*10
print('% of Students with tt(Small) with Very Good result    :',
      train['esp'][train['tt'] == 'Small'].value_counts(normalize = True)['Vg']*100,


#print percentages of students with Good result
print('% of Students with tt(Large) with Good result    :',
      train['esp'][train['tt'] == 'Large'].value_counts(normalize = True)['Good']*10
print('% of Students with tt(Average) with Good result  :',
      train['esp'][train['tt'] == 'Average'].value_counts(normalize = True)['Good']*
print('% of Students with tt(Small) with Good result    :',
      train['esp'][train['tt'] == 'Small'].value_counts(normalize = True)['Good']*10

#print percentages of students with Passable result
print('% of Students with tt(Large) with Passable result    :',
      train['esp'][train['tt'] == 'Large'].value_counts(normalize = True)['Pass']*10
print('% of Students with tt(Average) with Passable result  :',
      train['esp'][train['tt'] == 'Average'].value_counts(normalize = True)['Pass']*
print('% of Students with tt(Small) with Passable result    :',
      train['esp'][train['tt'] == 'Small'].value_counts(normalize = True)['Pass']*10
```

```
% of Students with tt(Average) with Best result  : 4.651162790697675
% of Students with tt(Small) with Best result    : 7.6923076923076925

% of Students with tt(Large) with Very Good result    : 20.0
% of Students with tt(Average) with Very Good result  : 30.23255813953488
% of Students with tt(Small) with Very Good result    : 34.61538461538461

% of Students with tt(Large) with Good result    : 60.0
% of Students with tt(Average) with Good result  : 30.23255813953488
% of Students with tt(Small) with Good result    : 44.871794871794876

% of Students with tt(Large) with Passable result    : 20.0
% of Students with tt(Average) with Passable result  : 34.883720930232556
```

```
% of Students with tt(Small) with Passable result    : 12.82051282051282
```

LEGEND

tt = Home to College Travel Time Large = Large Average = Average Small = Small

esp = End Semester Percentage Best = Best Vg = Very Good Good = Good Pass = Pass

PREDICTION RESULT

The number of those from tt(Large) is too low to compare with others.

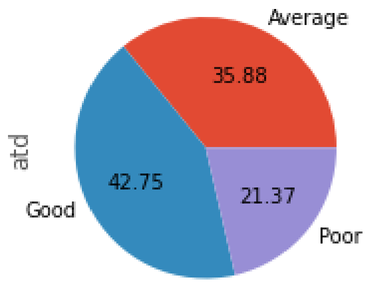
But, if we compare between tt(Small) and tt(Average), as predicted, those who travel to college for a Small period of time have better results compared to others. They have the lowest percentage of grades lower than Good compared to grades that are Good or higher.

In [28]:

```
# Class Attendance Percentage (atd) Feature
# Prediction: Students with good attendance are more likely to have better grades


print('Total of Students with atd(Good)    :', train['atd'].tolist().count('Good'),
print('Total of Students with atd(Average) :', train['atd'].tolist().count('Average'
print('Total of Students with atd(Poor)    :', train['atd'].tolist().count('Poor'),

train.groupby('atd')['atd'].count().plot.pie(autopct='%.2f',figsize=(3,3))
```

```
Total of Students with atd(Good)    : 56

Total of Students with atd(Average) : 47

Total of Students with atd(Poor)    : 28
```

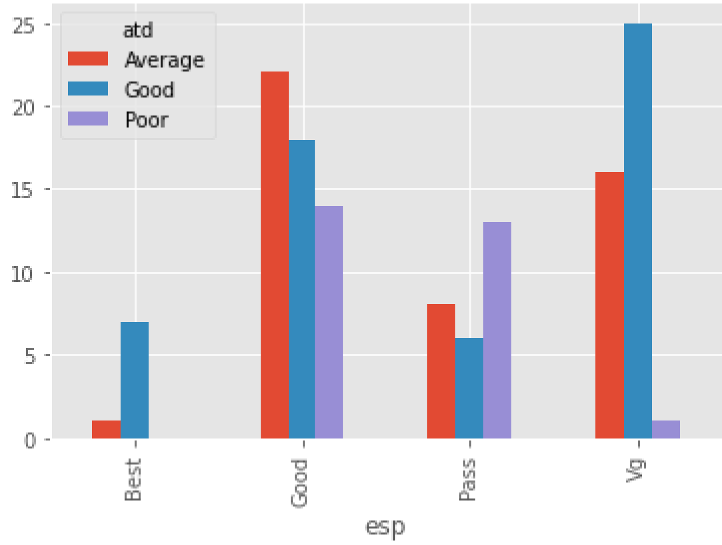
Out[28]: `<AxesSubplot:ylabel='atd'>`



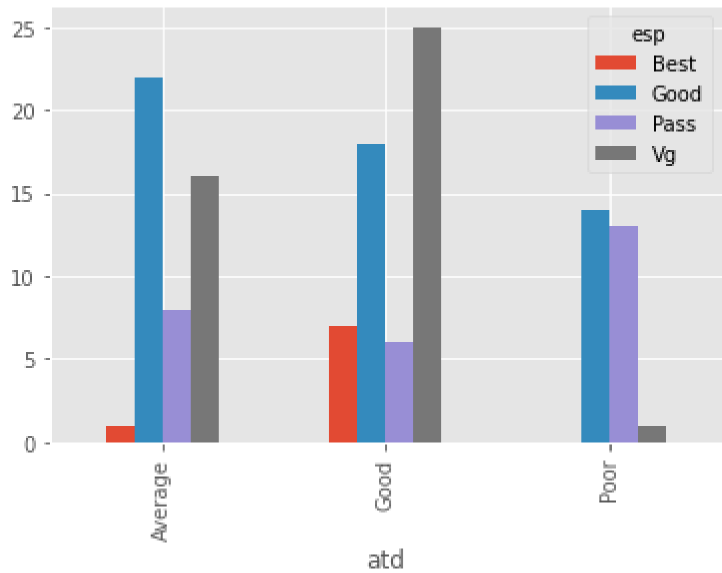
So, it's pretty even but most of the students have Good attendance.

In [29]:

```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['esp', 'atd'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [30]:
```
#Plot Bar Chart

plt.style.use('ggplot')

train.groupby(['atd', 'esp'])\
      .ls.count().unstack().plot.bar(legend=True)

plt.show()
```



In [31]:
```
#print percentages of students with Best result
print('% of Students with atd(Good) with Best result         :',
      train['esp'][train['atd'] == 'Good'].value_counts(normalize = True)['Best']*10
print('% of Students with atd(Average) with Best result      :',
      train['esp'][train['atd'] == 'Average'].value_counts(normalize = True)['Best']


#print percentages of students with Very Good result
print('% of Students with atd(Good) with Very Good result    :',
      train['esp'][train['atd'] == 'Good'].value_counts(normalize = True)['Vg']*100)
print('% of Students with atd(Average) with Very Good result :',
      train['esp'][train['atd'] == 'Average'].value_counts(normalize = True)['Vg']*1
print('% of Students with atd(Poor) with Very Good result    :',
      train['esp'][train['atd'] == 'Poor'].value_counts(normalize = True)['Vg']*100,
```

```
#print percentages of students with Good result
print('% of Students with atd(Good) with Good result         :',
      train['esp'][train['atd'] == 'Good'].value_counts(normalize = True)['Good']*10
print('% of Students with atd(Average) with Good result      :',
      train['esp'][train['atd'] == 'Average'].value_counts(normalize = True)['Good']
print('% of Students with atd(Poor) with Good result         :',
      train['esp'][train['atd'] == 'Poor'].value_counts(normalize = True)['Good']*10


#print percentages of students with Passable result
print('% of Students with atd(Good) with Passable result     :',
      train['esp'][train['atd'] == 'Good'].value_counts(normalize = True)['Pass']*10
print('% of Students with atd(Average) with Passable result  :',
      train['esp'][train['atd'] == 'Average'].value_counts(normalize = True)['Pass']
print('% of Students with atd(Poor) with Passable result     :',
      train['esp'][train['atd'] == 'Poor'].value_counts(normalize = True)['Pass']*10
```

```
% of Students with atd(Good) with Best result         : 12.5
% of Students with atd(Average) with Best result      : 2.127659574468085

% of Students with atd(Good) with Very Good result    : 44.642857142857146
% of Students with atd(Average) with Very Good result : 34.04255319148936
% of Students with atd(Poor) with Very Good result    : 3.571428571428571

% of Students with atd(Good) with Good result         : 32.142857142857146
% of Students with atd(Average) with Good result      : 46.808510638297875
% of Students with atd(Poor) with Good result         : 50.0

% of Students with atd(Good) with Passable result     : 10.714285714285714
% of Students with atd(Average) with Passable result  : 17.02127659574468
% of Students with atd(Poor) with Passable result     : 46.42857142857143
```

LEGEND

atd = Attendance Percentage Good = Good Average = Average Poor = Poor

esp = End Semester Percentage Best = Best Vg = Very Good Good = Good Pass = Pass

PREDICTION RESULT

As expected, those with Good attendance have better results compared to others. Even though they made up the majority, this group has the lowest percentage of those with grades lower than Good.

In [32]:

```
# 5) Cleaning Data - Also known as Step 3b(i) in WMU102 project instruction
#
# Time to clean our data to account for missing values and unnecessary information!
#

# Well, at this point, that's what we are supposed to do but luckily,
# the dataset has no missing data so we can go straight to
# dropping unnecessary information. By dropping features we are dealing with
# fewer data points. Speeds up our notebook and eases the analysis.


# Check features within the dataset before the drop
print('BEFORE DROP (train): ', train.columns, '\n')
print('BEFORE DROP (test): ', train.columns, '\n')

#Drop features

train = train.drop(['cst', 'tnp', 'twp', 'iap', 'arr', 'ms', 'fq', 'mq', 'fo', 'mo',
test = test.drop(['cst', 'tnp', 'twp', 'iap', 'arr', 'ms', 'fq', 'mq', 'fo', 'mo', '
```

```
#Check features within the dataset after the drop
print('AFTER DROP (train): ', train.columns, '\n')
print('AFTER DROP (test): ', train.columns, '\n')
```

```
BEFORE DROP (train):  Index(['ge', 'cst', 'tnp', 'twp', 'iap', 'esp', 'arr', 'ms',
'ls', 'as', 'fmi',
       'fs', 'fq', 'mq', 'fo', 'mo', 'nf', 'sh', 'ss', 'me', 'tt', 'atd'],
      dtype='object')

BEFORE DROP (test):  Index(['ge', 'cst', 'tnp', 'twp', 'iap', 'esp', 'arr', 'ms', 'l
s', 'as', 'fmi',
       'fs', 'fq', 'mq', 'fo', 'mo', 'nf', 'sh', 'ss', 'me', 'tt', 'atd'],
      dtype='object')

AFTER DROP (train):  Index(['ge', 'esp', 'ls', 'as', 'fmi', 'fs', 'nf', 'sh', 'tt',
'atd'], dtype='object')

AFTER DROP (test):  Index(['ge', 'esp', 'ls', 'as', 'fmi', 'fs', 'nf', 'sh', 'tt',
'atd'], dtype='object')
```

Features dropped:

- Caste (cs) because the system is irrelevant to Malaysia
- Class X Percentage (tnp), Class XII Percentage (twp), Internal Assessment Percentage (iap) because they are irrelevant to Malaysia and End Semester Percentage (esp) is enough as the representative
- Whether the student has back or arrear papers (ARR) because we don't even know what this means
- Marital Status (ms) because all of the students in this dataset have Unmarried status
- Father Qualification (fq), Mother Qualification (mq), Father Occupation (fo), Mother Occupation (mo) because Family Monthly Income (fmi) is enough as the representative
- Student School attended at Class X (ss) because the way private and government schools operate in India is probably different to Malaysia
- Medium (me) because Assamese, Hindi, and Bengali are not the language medium for teaching in Malaysia.

In [33]:

```
# 5) Cleaning Data - Continued
# Convert non-numerical features to numerical features
# because this is required by most model algorithms.

# result sample
print('BEFORE DATA TYPE CONVERT: \n', train.head(), '\n')

# map each string value to a numerical value
# for GE (Gender) : (Male, Female)
ge_mapping = {'M': 0, 'F': 1}
train['ge'] = train['ge'].map(ge_mapping)
test['ge'] = test['ge'].map(ge_mapping)

# map each string value to a numerical value
# for ESP (End Semester Percentage) : (Best, Very Good, Good, Pass, Fail)
esp_mapping = {'Best': 0, 'Vg': 1, 'Good': 2, 'Pass': 3, 'Fail': 4}
train['esp'] = train['esp'].map(esp_mapping)
```

```
test['esp'] = test['esp'].map(esp_mapping)

# map each string value to a numerical value
# for LS (Lived in Town or Village) : (Town, Village)
ls_mapping = {'T': 0, 'V': 1}
train['ls'] = train['ls'].map(ls_mapping)
test['ls'] = test['ls'].map(ls_mapping)

# map each string value to a numerical value
# for AS (Admission Category) : (Free, Paid)
as_mapping = {'Free': 0, 'Paid': 1}
train['as'] = train['as'].map(as_mapping)
test['as'] = test['as'].map(as_mapping)

# map each string value to a numerical value
# for FMI (Family Monthly Income (in INR)) : (Very High, High, Above Medium, Medium,
fmi_mapping = {'Vh': 0, 'High': 1, 'Am': 2, 'Medium': 3, 'Low': 4}
train['fmi'] = train['fmi'].map(fmi_mapping)
test['fmi'] = test['fmi'].map(fmi_mapping)

# map each string value to a numerical value
# for FS (Family Size) : (Large, Average, Small)
fs_mapping = {'Large': 0, 'Average': 1, 'Small': 2}
train['fs'] = train['fs'].map(fs_mapping)
test['fs'] = test['fs'].map(fs_mapping)

# map each string value to a numerical value
# for NF (Number of Friends) : (Large, Average, Small)
nf_mapping = {'Large': 0, 'Average': 1, 'Small': 2}
train['nf'] = train['nf'].map(nf_mapping)
test['nf'] = test['nf'].map(nf_mapping)

# map each string value to a numerical value
# for SH (Study Hours) : (Good, Average, Poor)
sh_mapping = {'Good': 0, 'Average': 1, 'Poor': 2}
train['sh'] = train['sh'].map(sh_mapping)
test['sh'] = test['sh'].map(sh_mapping)

# map each string value to a numerical value
# for TT (Home to College Travel Time) : (Large, Average, Small)
tt_mapping = {'Large': 0, 'Average': 1, 'Small': 2}
train['tt'] = train['tt'].map(tt_mapping)
test['tt'] = test['tt'].map(tt_mapping)

# map each string value to a numerical value
# for ATD (Class Attendance Percentage) : (Good, Average, Poor)
atd_mapping = {'Good': 0, 'Average': 1, 'Poor': 2}
train['atd'] = train['atd'].map(atd_mapping)
test['atd'] = test['atd'].map(atd_mapping)

# result sample
print('AFTER DATA TYPE CONVERT: \n', train.head(), '\n')
```

```
BEFORE DATA TYPE CONVERT:
   ge   esp ls    as     fmi       fs       nf       sh       tt      atd
0  F  Good  V  Paid  Medium  Average    Large     Poor    Small     Good
1  M    Vg  V  Paid     Low  Average    Small     Poor  Average  Average
2  F  Good  V  Paid      Am  Average  Average  Average    Large     Good
3  M  Good  V  Paid  Medium    Small    Large     Poor  Average  Average
4  M    Vg  V  Paid      Am  Average    Large     Poor    Small     Good

AFTER DATA TYPE CONVERT:
    ge  esp  ls  as  fmi  fs  nf  sh  tt  atd
0   1    2   1   1    3   1   0   2   2    0
1   0    1   1   1    4   1   2   2   1    1
2   1    2   1   1    2   1   1   1   0    0
```

```
3   0    2   1   1    3   2   0   2   1    1
4   0    1   1   1    2   1   0   2   2    0
```

In [34]:

```
# 6) Choosing the Best Model - also know as Step 3c in WMU102 project instruction
#
# Splitting the Training Data
#

# 70% of the samples would be used for dataset training
# 30% of the samples would be used for dataset testing

from sklearn.model_selection import train_test_split

predictors = train.drop(['esp'], axis=1)
target = train["esp"]
x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.

# We will use part of our training data (0.3 or 30% in this case) to test the accura

# Since we have a small dataset, we will choose 3 random machine learning methods
# said to be suitable for a small dataset or
# a dataset where the number of observations is higher as compared to the number of

# Based on this article: https://www.kdnuggets.com/2020/05/guide-choose-right-machin
# The 3 machine learning methods are: Naïve Bayes, KNN, and Decision Tree

# For each model, we set the model, fit it with 80% of our training data,
# predict for 20% of the training data and check the accuracy

# Now, onto Step 3d in WMU102 project instruction
```

In [35]:

```
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

gaussian = GaussianNB()
gaussian.fit(x_train, y_train)
y_pred = gaussian.predict(x_val)
acc_gaussian = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Gaussian Naive Bayes : ', acc_gaussian)


# KNN or k-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
y_pred = knn.predict(x_val)
acc_knn = round(accuracy_score(y_pred, y_val) * 100, 2)
print('k-Nearest Neighbors  : ', acc_knn)


#Decision Tree
from sklearn.tree import DecisionTreeClassifier

decisiontree = DecisionTreeClassifier()
decisiontree.fit(x_train, y_train)
y_pred = decisiontree.predict(x_val)
acc_decisiontree = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Decision Tree        : ', acc_decisiontree)
```

```
Gaussian Naive Bayes :  45.0
```

```
k-Nearest Neighbors  :  50.0
Decision Tree        :  52.5
```

In [36]:

```
# How come the accuracy rate is so bad?
# All of them are not above 50% while the ones in the referenced project
# www.kaggle.com/nadintamer/titanic-survival-predictions-beginner/ are around 70-80%
# Is it because we use unsuitable methods?
# In that case, let's try the other methods used in that project to confirm


# Logistic Regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_val)
acc_logreg = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Logistic Regression          : ', acc_logreg)


# Support Vector Machines
from sklearn.svm import SVC

svc = SVC()
svc.fit(x_train, y_train)
y_pred = svc.predict(x_val)
acc_svc = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Support Vector Machines      : ', acc_svc)


# Linear SVC
from sklearn.svm import LinearSVC

linear_svc = LinearSVC()
linear_svc.fit(x_train, y_train)
y_pred = linear_svc.predict(x_val)
acc_linear_svc = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Linear SVC                   : ', acc_linear_svc)


# Perceptron
from sklearn.linear_model import Perceptron

perceptron = Perceptron()
perceptron.fit(x_train, y_train)
y_pred = perceptron.predict(x_val)
acc_perceptron = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Perceptron                   : ', acc_perceptron)


# Random Forest
from sklearn.ensemble import RandomForestClassifier

randomforest = RandomForestClassifier()
randomforest.fit(x_train, y_train)
y_pred = randomforest.predict(x_val)
acc_randomforest = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Random Forest                : ', acc_randomforest)


# Stochastic Gradient Descent
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier()
sgd.fit(x_train, y_train)
```

```
y_pred = sgd.predict(x_val)
acc_sgd = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Stochastic Gradient Descent  : ', acc_sgd)


# Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier

gbk = GradientBoostingClassifier()
gbk.fit(x_train, y_train)
y_pred = gbk.predict(x_val)
acc_gbk = round(accuracy_score(y_pred, y_val) * 100, 2)
print('Gradient Boosting Classifier : ', acc_gbk)
```

```
Logistic Regression          :  50.0
Support Vector Machines      :  57.5
Linear SVC                   :  47.5
Perceptron                   :  32.5
Random Forest                :  57.5
Stochastic Gradient Descent  :  50.0
Gradient Boosting Classifier :  60.0
```

In [37]:

```
#Hmm... still none 70% or above.
# At least, some of them are around 60% in accuracy rate
# which are much better rates than the previous three.

# We don't know what exactly the problem is due to our limited knowledge
# so for now, let's move on to the last step (Step 3e in  WMU102 project instruction

# Evaluate the models using performance metrics.

models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron', 'Linear SVC',
              'Decision Tree', 'Stochastic Gradient Descent', 'Gradient Boosting Cla
    'Score': [acc_svc, acc_knn, acc_logreg,
              acc_randomforest, acc_gaussian, acc_perceptron,acc_linear_svc, acc_dec
              acc_sgd, acc_gbk]})
models.sort_values(by='Score', ascending=False)
```

Out[37]:

	Model	Score
9	Gradient Boosting Classifier	60.0
0	Support Vector Machines	57.5
3	Random Forest	57.5
7	Decision Tree	52.5
1	KNN	50.0
2	Logistic Regression	50.0
8	Stochastic Gradient Descent	50.0
6	Linear SVC	47.5
4	Naive Bayes	45.0
5	Perceptron	32.5

Based on the above result, we decided to use the Gradient Boosting Classifier model for the testing data because it has the best accuracy rate which is 60%.

That's all for now.

Thank you.

BONUS MARK

We have uploaded a copy of this notebook on Github: https://github.com/YasminRfd/makersXskymindProject

We also put an article about on our blog https://g2-wmu102-2021.blogspot.com/