In [1]:
```
import pandas as pd
pd.set_option('max_colwidth', 200)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
from statsmodels.stats.weightstats import *
import scipy.stats

#this is the entire dataset, but we'll only be able to use to extract samples from it.
city_hall_dataset = pd.read_csv('/Users/akheruddinahmed/Desktop/Python/ML Lab/Statistical Tests/train.csv')
```

In [2]:
```
def results(p):
    if(p['p_value']<0.05):p['hypothesis_accepted'] = 'alternative'
    if(p['p_value']>=0.05):p['hypothesis_accepted'] = 'null'

    df = pd.DataFrame(p, index=[''])
    cols = ['value1', 'value2', 'score', 'p_value', 'hypothesis_accepted']
    return df[cols]
```

In [3]:
```
import numpy as np
city_hall_dataset['SalePrice'] = np.log1p(city_hall_dataset['SalePrice'])
logged_budget = np.log1p(120000) #logged $120 000 is 11.695
logged_budget
```

Out[3]:
```
11.695255355062795
```

In [4]:
```
sample = city_hall_dataset.sample(n=25)
p = {} #dictionnary we'll use to stock information and results
```

In [5]:
```
p['value1'], p['value2'] = sample['SalePrice'].mean(), logged_budget
p['score'], p['p_value'] = stats.ttest_1samp(sample['SalePrice'], popmean=logged_budget)
results(p)
```

Out[5]:

	value1	value2	score	p_value	hypothesis_accepted
	12.022	11.695	4.993	0.000	alternative

In [6]:
```
p['value1'], p['value2'] = sample['SalePrice'].mean(), logged_budget
p['score'], p['p_value'] = stats.ttest_1samp(sample['SalePrice'], popmean=logged_budget)
p['p_value'] = p['p_value']/2 #one-tailed test (with scipy function), we need to divide p-value by 2 ourselve
results(p)
```

Out[6]:

value1	value2	score	p_value	hypothesis_accepted
12.022	11.695	4.993	0.000	alternative

In [7]:
```
smaller_houses = city_hall_dataset.sort_values('GrLivArea')[:730].sample(n=25)
larger_houses = city_hall_dataset.sort_values('GrLivArea')[730:].sample(n=25)
```

In [8]:
```
p['value1'], p['value2'] = smaller_houses['SalePrice'].mean(), larger_houses['SalePrice'].mean()
p['score'], p['p_value'], p['df'] = ttest_ind(smaller_houses['SalePrice'], larger_houses['SalePrice'])
results(p)
```

Out[8]:

value1	value2	score	p_value	hypothesis_accepted
11.748	12.328	-7.098	0.000	alternative

In [9]:
```
p['value1'], p['value2'] = smaller_houses['SalePrice'].mean(), larger_houses['SalePrice'].mean()
p['score'], p['p_value'], p['df'] = ttest_ind(smaller_houses['SalePrice'], larger_houses['SalePrice'], alterr
results(p)
```

Out[9]:

value1	value2	score	p_value	hypothesis_accepted
11.748	12.328	-7.098	0.000	alternative

In [10]:
```
smaller_houses = city_hall_dataset.sort_values('GrLivArea')[:730].sample(n=100, random_state=1)
larger_houses = city_hall_dataset.sort_values('GrLivArea')[730:].sample(n=100, random_state=1)
```

In [11]:
```
p['value1'], p['value2'] = smaller_houses['SalePrice'].mean(), larger_houses['SalePrice'].mean()
p['score'], p['p_value'] = ztest(smaller_houses['SalePrice'], larger_houses['SalePrice'], alternative='smalle
results(p)
```

Out[11]:

value1	value2	score	p_value	hypothesis_accepted
11.786	12.249	-10.772	0.000	alternative

In [12]:
```
from statsmodels.stats.proportion import *
A1 = len(smaller_houses[smaller_houses.SalePrice>logged_budget])
B1 = len(smaller_houses)
A2 = len(larger_houses[larger_houses.SalePrice>logged_budget])
B2 = len(larger_houses)
p['value1'], p['value2'] = A1/B1, A2/B2
p['score'], p['p_value'] = proportions_ztest([A1, A2], [B1, B2], alternative='smaller')
results(p)
```

Out[12]:

value1	value2	score	p_value	hypothesis_accepted
0.670	0.950	-5.047	0.000	alternative

In [13]:
```
p['value1'], p['value2'] = smaller_houses['SalePrice'].mean(), logged_budget
p['score'], p['p_value'] = ztest(smaller_houses['SalePrice'], value=logged_budget, alternative='larger')
results(p)
```

Out[13]:

value1	value2	score	p_value	hypothesis_accepted
11.786	11.695	3.593	0.000	alternative

In [14]:
```
from statsmodels.stats.proportion import *
A = len(smaller_houses[smaller_houses.SalePrice<logged_budget])
B = len(smaller_houses)
p['value1'], p['value2'] = A/B, 0.25
p['score'], p['p_value'] = proportions_ztest(A, B, alternative='larger', value=0.25)
results(p)
```

Out[14]:

value1	value2	score	p_value	hypothesis_accepted
0.320	0.250	1.501	0.067	null

In [15]:
```
replacement = {'FV': "Floating Village Residential", 'C (all)': "Commercial", 'RH': "Residential High Density
               'RL': "Residential Low Density", 'RM': "Residential Medium Density"}
smaller_houses['MSZoning_FullName'] = smaller_houses['MSZoning'].replace(replacement)
mean_price_by_zone = smaller_houses.groupby('MSZoning_FullName')['SalePrice'].mean().to_frame()
mean_price_by_zone
```

Out[15]:

MSZoning_FullName	SalePrice
Commercial	11.590
Floating Village Residential	12.030
Residential High Density	11.705
Residential Low Density	11.828
Residential Medium Density	11.617

In [16]:
```
sh = smaller_houses.copy()
p['score'], p['p_value'] = stats.f_oneway(sh.loc[sh.MSZoning=='FV', 'SalePrice'],
                sh.loc[sh.MSZoning=='C (all)', 'SalePrice'],
                sh.loc[sh.MSZoning=='RH', 'SalePrice'],
                sh.loc[sh.MSZoning=='RL', 'SalePrice'],
                sh.loc[sh.MSZoning=='RM', 'SalePrice'],)
results(p)[['score', 'p_value', 'hypothesis_accepted']]
```

Out[16]:

score	p_value	hypothesis_accepted
4.146	0.004	alternative

In [17]:
```
smaller_houses['GarageType'].fillna('No Garage', inplace=True)
smaller_houses['GarageType'].value_counts().to_frame()
```

Out[17]:

GarageType	count
Attchd	46
Detchd	41
No Garage	10
CarPort	2
Basment	1

In [18]:
```
city_hall_dataset['GarageType'].fillna('No Garage', inplace=True)
sample1 = city_hall_dataset.sort_values('GrLivArea')[:183].sample(n=100)
sample2 = city_hall_dataset.sort_values('GrLivArea')[183:366].sample(n=100)
sample3 = city_hall_dataset.sort_values('GrLivArea')[366:549].sample(n=100)
sample4 = city_hall_dataset.sort_values('GrLivArea')[549:730].sample(n=100)
dff = pd.concat([
    sample1['GarageType'].value_counts().to_frame(),
    sample2['GarageType'].value_counts().to_frame(),
    sample3['GarageType'].value_counts().to_frame(),
    sample4['GarageType'].value_counts().to_frame()],
    axis=1, sort=False)
dff.columns = ['Sample1 (smallest houses)', 'Sample2', 'Sample3', 'Sample4 (largest houses)']
dff = dff[:3] #chi-square tests do not work when table contains some 0, we take only the most frequent attrib
dff
```

Out[18]:

GarageType	Sample1 (smallest houses)	Sample2	Sample3	Sample4 (largest houses)
Detchd	52.000	40.000	39.000	26.000
Attchd	34.000	50.000	55.000	62.000
No Garage	11.000	7.000	5.000	5.000

In [19]:
```
p['score'], p['p_value'], p['ddf'], p['contigency'] = stats.chi2_contingency(dff)
p.pop('contigency')
results(p)[['score', 'p_value', 'hypothesis_accepted']]
```

Out[19]:

score	p_value	hypothesis_accepted
20.519	0.002	alternative

In [ ]:
```

```