Data preparation

Sampling

In [1]:
```
# Import packages and specific functions
import pandas as pd
from sklearn.model_selection import train_test_split
```

In [2]:
```
# Read file
df = pd.read_csv('transactions.csv', index_col='transaction_id')
df.head()
```

Out[2]:

transaction_id	amount	company	label
2301	29257.40	Walmart	valid
2302	22489.10	Kadoozee	fraudulent
2303	70422.30	Apple	valid
2304	51889.70	Kadoozee	fraudulent
2305	66261.95	Verizon	valid

In [3]:
```
# Return a simple random sample of size 4 without replacement
sample = df.sample(n=4, replace=False, random_state=12345)
sample
```

Out[3]:

transaction_id	amount	company	label
2301	29257.40	Walmart	valid
2312	37073.50	Swisscom	valid
2308	76382.05	McDonalds	valid
2309	80163.05	Nestle	valid

In [4]:
```
# Return a stratified sample of size 4 without replacement
split = train_test_split(df, test_size=4, random_state=12345, stratify=df.label)
split[1]
```

Out[4]:

transaction_id	amount	company	label
2307	27973.55	Tesla	valid
2313	67169.05	Kadoozee	fraudulent
2312	37073.50	Swisscom	valid
2303	70422.30	Apple	valid

Missing values

In [5]:
```
# Read file
df = pd.read_csv('credit.csv', index_col='application_id')
df
```

Out[5]:

application_id	age	income	marital_status	score	class
1	34	1800.0	NaN	620.0	Default
2	60	2200.0	Single	700.0	Default
3	44	NaN	NaN	NaN	No-default
4	26	1500.0	Married	351.0	No-default
5	34	NaN	Single	NaN	Default
6	50	2100.0	Divorced	NaN	No-default

In [6]:
```
# Check if DataFrame contains missing values
df.isnull().any()
```

Out[6]:
```
age               False
income             True
marital_status     True
score              True
class             False
dtype: bool
```

In [7]:
```
# Delete rows which contain missing values
df_new = df.dropna()
df_new
```

Out[7]:

application_id	age	income	marital_status	score	class
2	60	2200.0	Single	700.0	Default
4	26	1500.0	Married	351.0	No-default

In [8]:
```
# Keep missing values
df['income_missing'] = df.income.isna()
df
```

Out[8]:

application_id	age	income	marital_status	score	class	income_missing
1	34	1800.0	NaN	620.0	Default	False
2	60	2200.0	Single	700.0	Default	False
3	44	NaN	NaN	NaN	No-default	True
4	26	1500.0	Married	351.0	No-default	False
5	34	NaN	Single	NaN	Default	True
6	50	2100.0	Divorced	NaN	No-default	False

In [9]:
```
# Replace missing values in column income by median
df.income.fillna(df.income.median(), inplace=True)
df
```

Out[9]:

application_id	age	income	marital_status	score	class	income_missing
1	34	1800.0	NaN	620.0	Default	False
2	60	2200.0	Single	700.0	Default	False
3	44	1950.0	NaN	NaN	No-default	True
4	26	1500.0	Married	351.0	No-default	False
5	34	1950.0	Single	NaN	Default	True
6	50	2100.0	Divorced	NaN	No-default	False

In [10]:
```
# Replace missing values in column marital_status by mode
df.marital_status.fillna(df.marital_status.mode()[0], inplace=True)
df
```

Out[10]:

application_id	age	income	marital_status	score	class	income_missing
1	34	1800.0	Single	620.0	Default	False
2	60	2200.0	Single	700.0	Default	False
3	44	1950.0	Single	NaN	No-default	True
4	26	1500.0	Married	351.0	No-default	False
5	34	1950.0	Single	NaN	Default	True
6	50	2100.0	Divorced	NaN	No-default	False

In [11]:
```
# Replace missing values in column score by mean
df.score.fillna(df.score.mean(), inplace=True)
df
```

Out[11]:

application_id	age	income	marital_status	score	class	income_missing
1	34	1800.0	Single	620.0	Default	False
2	60	2200.0	Single	700.0	Default	False
3	44	1950.0	Single	557.0	No-default	True
4	26	1500.0	Married	351.0	No-default	False
5	34	1950.0	Single	557.0	Default	True
6	50	2100.0	Divorced	557.0	No-default	False

Outlier detection

In [12]:
```
# Import NumPy package
import numpy as np
```

In [13]:
```
# Read file
df = pd.read_csv('visitors.csv')
df.head()
```

Out[13]:

	day	visitors
0	1	64
1	2	85
2	3	59
3	4	120
4	5	88

In [14]:
```
# Detect outliers using boxplots
q1 = df.visitors.quantile(0.25)
q3 = df.visitors.quantile(0.75)
iqr = q3-q1
upper_threshold = q3 + 1.5 * iqr
lower_threshold = q1 - 1.5 * iqr
idx = (df.visitors > upper_threshold) | (df.visitors < lower_threshold)
df.visitors[idx]
```

Out[14]:
```
231      8
445      5
494    175
655      7
842    156
974      5
1164     0
1294   171
1441    -1
1447    10
1532     6
1736   159
1840   153
1851   151
Name: visitors, dtype: int64
```

In [15]:
```
# Detect outliers using boxplots: shortcut
ax = df.visitors.plot.box(return_type='dict')
ax['fliers'][0].get_data()[1]
```

Out[15]:
```
array([  8,   5,   7,   5,   0,  -1,  10,   6, 175, 156, 171, 159, 153,
       151], dtype=int64)
```

In [16]:
```
# Detect outliers using z-scores
z = (df.visitors - df.visitors.mean()) / df.visitors.std()
df.visitors[np.abs(z) > 3]
```

Out[16]:
```
494    175
1164     0
1294   171
1441    -1
1736   159
Name: visitors, dtype: int64
```

In [17]:
```
# Treatment of invalid outliers
df.visitors[df.visitors < 0] = np.nan
```

In [18]:
```
# Treatment of valid outliers
mu = df.visitors.mean()
sd = df.visitors.std()
lower_limit = mu - 3 * sd
upper_limit = mu + 3 * sd
df.visitors[df.visitors < lower_limit] = lower_limit
df.visitors[df.visitors > upper_limit] = upper_limit
```

Categorization

In [19]:
```
# Create NumPy array
x = np.array([1, 1, 1, 1, 3, 3, 4, 10, 12, 14, 16, 44])
x
```

Out[19]:
```
array([ 1,  1,  1,  1,  3,  3,  4, 10, 12, 14, 16, 44])
```

In [20]:
```
# Equal interval binning
pd.cut(x, 3)
```

Out[20]:
```
[(0.957, 15.333], (0.957, 15.333], (0.957, 15.333], (0.957, 15.333], (0.957, 15.333],
..., (0.957, 15.333], (0.957, 15.333], (0.957, 15.333], (15.333, 29.667], (29.667, 44.
0]]
Length: 12
Categories (3, interval[float64]): [(0.957, 15.333] < (15.333, 29.667] < (29.667, 44.
0]]
```

In [21]:
```
# Equal frequency binning
pd.qcut(x, 3)
```

Out[21]:
```
[(0.999, 2.333], (0.999, 2.333], (0.999, 2.333], (0.999, 2.333], (2.333, 10.667], ...,
(2.333, 10.667], (10.667, 44.0], (10.667, 44.0], (10.667, 44.0], (10.667, 44.0]]
Length: 12
Categories (3, interval[float64]): [(0.999, 2.333] < (2.333, 10.667] < (10.667, 44.0]]
```

In [22]:
```
# Equal frequency binning with labels
pd.qcut(x, 3, labels=['small', 'medium', 'big'])
```

Out[22]:
```
[small, small, small, small, medium, ..., medium, big, big, big, big]
Length: 12
Categories (3, object): [small < medium < big]
```

Managerial categorization

In [23]:
```
# Import function to compute Pearson's test statistic
from scipy.stats import chi2_contingency
```

In [24]:
```
# Import data
df = pd.read_csv('credit_risk.csv')
```

In [25]:
```
# Manually add categories in a column called new_categories
df['new_categories'] = 'Owner'
df.new_categories[df.residential_status == 'Ru'] = 'Renters'
df.new_categories[df.residential_status == 'Rf'] = 'Renters'
df.new_categories[df.residential_status == 'Wp'] = 'Others'
df.new_categories[df.residential_status == 'Ot'] = 'Others'
df.new_categories[df.residential_status == 'Na'] = 'Others'
```

In [26]:
```
# Compute Person's test statistic and p-value
result = chi2_contingency(pd.crosstab(df.risk, df.new_categories))
```

In [27]:
```
# Get test statistic
result[0]
```

Out[27]:
```
583.9019201560275
```

In [28]:
```
# Get p-value
result[1]
```

Out[28]:
```
1.61179220748454e-127
```

In [29]:
```
# Get degree of freedom
result[2]
```

Out[29]:
```
2
```

In [30]:
```
# Get E_ij
result[3]
```

Out[30]:
```
array([[ 121.,  630.,  249.],
       [1089., 5670., 2241.]])
```

Odds-based categorization

In [31]:
```
# Compute contingency table based on initial categories
counts = pd.crosstab(df.risk, df.residential_status)
counts
```

Out[31]:

residential_status	Na	Ot	Ow	Rf	Ru	Wp
risk						
Bad	10	50	300	140	400	100
Good	10	90	6000	350	1600	950

In [32]:
```
# Calculate odds for each initial category
odds = counts.iloc[1, :] / counts.iloc[0, :]
odds
```

Out[32]:
```
residential_status
Na     1.0
Ot     1.8
Ow    20.0
Rf     2.5
Ru     4.0
Wp     9.5
dtype: float64
```

In [33]:
```
# Apply equal interval binning strategy
pd.cut(odds, 3, precision=0, labels=['cat_1', 'cat_2', 'cat_3'])
```

Out[33]:
```
residential_status
Na    cat_1
Ot    cat_1
Ow    cat_3
Rf    cat_1
Ru    cat_1
Wp    cat_2
dtype: category
Categories (3, object): [cat_1 < cat_2 < cat_3]
```