ed-variable-and-datetime-handeling

May 26, 2025

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

[ ]:
```
df=pd.read_csv('/content/Data Cleaning Toy Dataset day33.csv')
```

[ ]:
```
df.head(10)
```

[ ]:

	Cabin	Ticket number	Survived
0	NaN	A/5 21171	5	0
1	C85	PC 17599	3	1
2	NaN	STON/O2. 3101282	6	1
3	C123	113803	3	1
4	NaN	373450	A	0
5	NaN	330877	2	0
6	E46	17463	2	0
7	NaN	349909	5	0
8	NaN	347742	1	1
9	NaN	237736	A	1

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Cabin     204 non-null    object
 1   Ticket    891 non-null    object
 2   number    891 non-null    object
 3   Survived  891 non-null    int64
dtypes: int64(1), object(3)
memory usage: 28.0+ KB
```

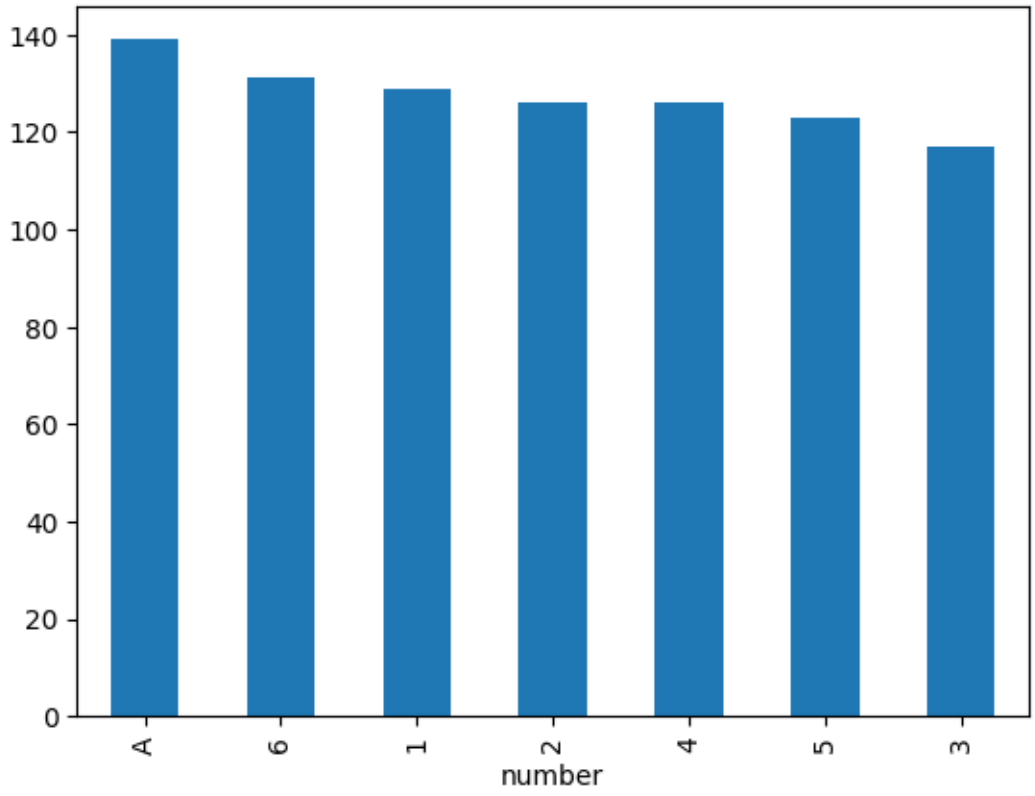
#Akti Column ar kico Row te Number and Kico row te String(Object)—->Like Number Column

[ ]:
```
df['number'].unique()
```

[ ]: array(['5', '3', '6', 'A', '2', '1', '4'], dtype=object)

[ ]:
```
df['number'].value_counts().plot(kind='bar')
```

[ ]: <Axes: xlabel='number'>



0.0.1 Step-by-step Example

```
import pandas as pd

# Sample DataFrame
df = pd.DataFrame({
    'Age': ['25', '30', 'unknown', '45', 'N/A', '50']
})

print(" Original DataFrame:")
print(df)

# Convert 'Age' column to numeric (invalid values become NaN)
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
```

```
print("\n  Cleaned DataFrame:")
print(df)
```

0.0.2 Output:

```
  Original DataFrame:
       Age
0       25
1       30
2  unknown
3       45
4      N/A
5       50

  Cleaned DataFrame:
    Age
0  25.0
1  30.0
2   NaN
3  45.0
4   NaN
5  50.0
```

0.0.3 Notice:

- `'unknown'` and `'N/A'` are converted to `NaN` because they're not valid numbers.
- Now you can apply imputation, drop missing, or continue modeling.

[ ]:
```
df['number_numirical']=pd.
  ↪to_numeric(df['number'],errors='coerce',downcast='integer')
df
```

[ ]:

	Cabin	Ticket	number	Survived	number_numirical
0	NaN	A/5 21171	5	0	5.0
1	C85	PC 17599	3	1	3.0
2	NaN	STON/O2. 3101282	6	1	6.0
3	C123	113803	3	1	3.0
4	NaN	373450	A	0	NaN
..	…	…	…	…	…
886	NaN	211536	3	0	3.0
887	B42	112053	3	1	3.0
888	NaN	W./C. 6607	1	0	1.0
889	C148	111369	2	1	2.0
890	NaN	370376	3	0	3.0

```
[891 rows x 5 columns]
```

[ ]:
```
#np.where(condition, value_if_true, value_if_false)

df['number_catagorical']=np.where(df['number_numirical'].
 ↪isnull(),df['number'],np.nan)
df
```

[ ]:
```
    Cabin            Ticket number  Survived  number_numirical  \
0     NaN         A/5 21171      5         0               5.0
1     C85          PC 17599      3         1               3.0
2     NaN  STON/O2. 3101282      6         1               6.0
3    C123            113803      3         1               3.0
4     NaN            373450      A         0               NaN
..    …                …    …       …                …
886   NaN            211536      3         0               3.0
887   B42            112053      3         1               3.0
888   NaN        W./C. 6607      1         0               1.0
889  C148            111369      2         1               2.0
890   NaN            370376      3         0               3.0

    number_catagorical
0                  NaN
1                  NaN
2                  NaN
3                  NaN
4                    A
..                 …
886                NaN
887                NaN
888                NaN
889                NaN
890                NaN

[891 rows x 6 columns]
```

[ ]:

[ ]:

#Data Example: abc234——>abc—->234

[ ]:
```
df['Cabin'].unique()
```

[ ]:
```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
```

```
'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
'C148'], dtype=object)
```

#NOTE:Data Cleaning For 'CABIN' column

0.0.4 `str.extract()` in Pandas

`str.extract()` is used to **extract specific patterns** (using regular expressions) from **string columns** in a DataFrame or Series.

0.0.5 Syntax

```
df['column'].str.extract('your-regex-pattern')
```

0.0.6 Use Case

When you want to extract parts of a string, like:

- Email domains
- Dates or numbers from strings
- Titles from names
- Codes from IDs

0.0.7 Example 1: Extract Number from Text

```
import pandas as pd

df = pd.DataFrame({'info': ['Age: 25', 'Age: 30', 'Age: 45']})

# Extract digits using regex
df['age'] = df['info'].str.extract(r'(\d+)')
print(df)
```

Output:

```
      info age
0  Age: 25  25
1  Age: 30  30
2  Age: 45  45
```

0.0.8 Example 2: Extract Multiple Groups

```
df = pd.DataFrame({'person': ['Mr. John Doe', 'Ms. Jane Smith', 'Dr. Alice Brown']})

# Extract title and name
df[['title', 'name']] = df['person'].str.extract(r'(Mr\.|Ms\.|Dr\.)\s+(.*)')
print(df)
```

Output:

```
          person title         name
0    Mr. John Doe   Mr.     John Doe
1  Ms. Jane Smith   Ms.   Jane Smith
2  Dr. Alice Brown  Dr.  Alice Brown
```

0.0.9 Example — Extract Email Domain

```
import pandas as pd

df = pd.DataFrame({
    'email': ['sifat01@gmail.com', 'user2@yahoo.com', 'admin@outlook.com']
})

# Extract domain name after '@'
df['domain'] = df['email'].str.extract(r'@(\w+\.\w+)')
print(df)
```

Output:

```
               email        domain
0   sifat01@gmail.com    gmail.com
1     user2@yahoo.com    yahoo.com
2  admin@outlook.com  outlook.com
```

If you have a specific string or pattern you want help with, feel free to paste it here, and I'll help you write the right `.str.extract()` expression for it.

0.0.10 Tips

- Parentheses `()` in regex are used to **capture groups** (these are returned in the result).
- Use `expand=False` if you want the result as a Series instead of a DataFrame.

[ ]:
```
#for Cabin column you can see each element first position is a catagorical(str)␣
 ↪and then Numerical data
df['cabin_num']=df['Cabin'].str.extract('(\d+)')
df
```

[ ]:
```
    Cabin            Ticket number  Survived  number_numirical  \
0     NaN         A/5 21171      5         0               5.0
1     C85          PC 17599      3         1               3.0
2     NaN  STON/O2. 3101282      6         1               6.0
3    C123            113803      3         1               3.0
4     NaN            373450      A         0               NaN
..    …                …    …       …               …
886   NaN            211536      3         0               3.0
887   B42            112053      3         1               3.0
888   NaN        W./C. 6607      1         0               1.0
889  C148            111369      2         1               2.0
890   NaN            370376      3         0               3.0

    number_catagorical cabin_num
0                  NaN       NaN
1                  NaN        85
2                  NaN       NaN
3                  NaN       123
4                    A       NaN
..                 …       …
886                NaN       NaN
887                NaN        42
888                NaN       NaN
889                NaN       148
890                NaN       NaN

[891 rows x 7 columns]
```

[ ]:
```
df['cabin_cat']=df['Cabin'].str[0]
df
```

[ ]:
```
    Cabin            Ticket number  Survived  number_numirical  \
0     NaN         A/5 21171      5         0               5.0
1     C85          PC 17599      3         1               3.0
2     NaN  STON/O2. 3101282      6         1               6.0
3    C123            113803      3         1               3.0
4     NaN            373450      A         0               NaN
..    …                …    …       …               …
886   NaN            211536      3         0               3.0
887   B42            112053      3         1               3.0
888   NaN        W./C. 6607      1         0               1.0
889  C148            111369      2         1               2.0
```

```
890   NaN            370376      3         0               3.0

    number_catagorical cabin_num cabin_cat
0                  NaN       NaN       NaN
1                  NaN        85         C
2                  NaN       NaN       NaN
3                  NaN       123         C
4                    A       NaN       NaN
..                 …       …        …
886                NaN       NaN       NaN
887                NaN        42         B
888                NaN       NaN       NaN
889                NaN       148         C
890                NaN       NaN       NaN

[891 rows x 8 columns]
```

[ ]:
```
#it converted a Catragorical data
df['cabin_cat'].unique()
```

[ ]:
```
array([nan, 'C', 'E', 'G', 'D', 'A', 'B', 'F', 'T'], dtype=object)
```

#NOTE: Data Cleaning Applying For 'TICKET' colnum

[ ]:
```
df['Ticket'].unique()
```

[ ]:
```
array(['A/5 21171', 'PC 17599', 'STON/O2. 3101282', '113803', '373450',
       '330877', '17463', '349909', '347742', '237736', 'PP 9549',
       '113783', 'A/5. 2151', '347082', '350406', '248706', '382652',
       '244373', '345763', '2649', '239865', '248698', '330923', '113788',
       '347077', '2631', '19950', '330959', '349216', 'PC 17601',
       'PC 17569', '335677', 'C.A. 24579', 'PC 17604', '113789', '2677',
       'A./5. 2152', '345764', '2651', '7546', '11668', '349253',
       'SC/Paris 2123', '330958', 'S.C./A.4. 23567', '370371', '14311',
       '2662', '349237', '3101295', 'A/4. 39886', 'PC 17572', '2926',
       '113509', '19947', 'C.A. 31026', '2697', 'C.A. 34651', 'CA 2144',
       '2669', '113572', '36973', '347088', 'PC 17605', '2661',
       'C.A. 29395', 'S.P. 3464', '3101281', '315151', 'C.A. 33111',
       'S.O.C. 14879', '2680', '1601', '348123', '349208', '374746',
       '248738', '364516', '345767', '345779', '330932', '113059',
       'SO/C 14885', '3101278', 'W./C. 6608', 'SOTON/OQ 392086', '343275',
       '343276', '347466', 'W.E.P. 5734', 'C.A. 2315', '364500', '374910',
       'PC 17754', 'PC 17759', '231919', '244367', '349245', '349215',
       '35281', '7540', '3101276', '349207', '343120', '312991', '349249',
       '371110', '110465', '2665', '324669', '4136', '2627',
       'STON/O 2. 3101294', '370369', 'PC 17558', 'A4. 54510', '27267',
       '370372', 'C 17369', '2668', '347061', '349241',
```

'SOTON/O.Q. 3101307', 'A/5. 3337', '228414', 'C.A. 29178',
'SC/PARIS 2133', '11752', '7534', 'PC 17593', '2678', '347081',
'STON/O2. 3101279', '365222', '231945', 'C.A. 33112', '350043',
'230080', '244310', 'S.O.P. 1166', '113776', 'A.5. 11206',
'A/5. 851', 'Fa 265302', 'PC 17597', '35851', 'SOTON/OQ 392090',
'315037', 'CA. 2343', '371362', 'C.A. 33595', '347068', '315093',
'363291', '113505', 'PC 17318', '111240', 'STON/O 2. 3101280',
'17764', '350404', '4133', 'PC 17595', '250653', 'LINE',
'SC/PARIS 2131', '230136', '315153', '113767', '370365', '111428',
'364849', '349247', '234604', '28424', '350046', 'PC 17610',
'368703', '4579', '370370', '248747', '345770', '3101264', '2628',
'A/5 3540', '347054', '2699', '367231', '112277',
'SOTON/O.Q. 3101311', 'F.C.C. 13528', 'A/5 21174', '250646',
'367229', '35273', 'STON/O2. 3101283', '243847', '11813',
'W/C 14208', 'SOTON/OQ 392089', '220367', '21440', '349234',
'19943', 'PP 4348', 'SW/PP 751', 'A/5 21173', '236171', '347067',
'237442', 'C.A. 29566', 'W./C. 6609', '26707', 'C.A. 31921',
'28665', 'SCO/W 1585', '367230', 'W./C. 14263',
'STON/O 2. 3101275', '2694', '19928', '347071', '250649', '11751',
'244252', '362316', '113514', 'A/5. 3336', '370129', '2650',
'PC 17585', '110152', 'PC 17755', '230433', '384461', '110413',
'112059', '382649', 'C.A. 17248', '347083', 'PC 17582', 'PC 17760',
'113798', '250644', 'PC 17596', '370375', '13502', '347073',
'239853', 'C.A. 2673', '336439', '347464', '345778', 'A/5. 10482',
'113056', '349239', '345774', '349206', '237798', '370373',
'19877', '11967', 'SC/Paris 2163', '349236', '349233', 'PC 17612',
'2693', '113781', '19988', '9234', '367226', '226593', 'A/5 2466',
'17421', 'PC 17758', 'P/PP 3381', 'PC 17485', '11767', 'PC 17608',
'250651', '349243', 'F.C.C. 13529', '347470', '29011', '36928',
'16966', 'A/5 21172', '349219', '234818', '345364', '28551',
'111361', '113043', 'PC 17611', '349225', '7598', '113784',
'248740', '244361', '229236', '248733', '31418', '386525',
'C.A. 37671', '315088', '7267', '113510', '2695', '2647', '345783',
'237671', '330931', '330980', 'SC/PARIS 2167', '2691',
'SOTON/O.Q. 3101310', 'C 7076', '110813', '2626', '14313',
'PC 17477', '11765', '3101267', '323951', 'C 7077', '113503',
'2648', '347069', 'PC 17757', '2653', 'STON/O 2. 3101293',
'349227', '27849', '367655', 'SC 1748', '113760', '350034',
'3101277', '350052', '350407', '28403', '244278', '240929',
'STON/O 2. 3101289', '341826', '4137', '315096', '28664', '347064',
'29106', '312992', '349222', '394140', 'STON/O 2. 3101269',
'343095', '28220', '250652', '28228', '345773', '349254',
'A/5. 13032', '315082', '347080', 'A/4. 34244', '2003', '250655',
'364851', 'SOTON/O.Q. 392078', '110564', '376564', 'SC/AH 3085',
'STON/O 2. 3101274', '13507', 'C.A. 18723', '345769', '347076',
'230434', '65306', '33638', '113794', '2666', '113786', '65303',
'113051', '17453', 'A/5 2817', '349240', '13509', '17464',

'F.C.C. 13531', '371060', '19952', '364506', '111320', '234360',
'A/S 2816', 'SOTON/O.Q. 3101306', '113792', '36209', '323592',
'315089', 'SC/AH Basle 541', '7553', '31027', '3460', '350060',
'3101298', '239854', 'A/5 3594', '4134', '11771', 'A.5. 18509',
'65304', 'SOTON/OQ 3101317', '113787', 'PC 17609', 'A/4 45380',
'36947', 'C.A. 6212', '350035', '315086', '364846', '330909',
'4135', '26360', '111427', 'C 4001', '382651', 'SOTON/OQ 3101316',
'PC 17473', 'PC 17603', '349209', '36967', 'C.A. 34260', '226875',
'349242', '12749', '349252', '2624', '2700', '367232',
'W./C. 14258', 'PC 17483', '3101296', '29104', '2641', '2690',
'315084', '113050', 'PC 17761', '364498', '13568', 'WE/P 5735',
'2908', '693', 'SC/PARIS 2146', '244358', '330979', '2620',
'347085', '113807', '11755', '345572', '372622', '349251',
'218629', 'SOTON/OQ 392082', 'SOTON/O.Q. 392087', 'A/4 48871',
'349205', '2686', '350417', 'S.W./PP 752', '11769', 'PC 17474',
'14312', 'A/4. 20589', '358585', '243880', '2689',
'STON/O 2. 3101286', '237789', '13049', '3411', '237565', '13567',
'14973', 'A./5. 3235', 'STON/O 2. 3101273', 'A/5 3902', '364848',
'SC/AH 29037', '248727', '2664', '349214', '113796', '364511',
'111426', '349910', '349246', '113804', 'SOTON/O.Q. 3101305',
'370377', '364512', '220845', '31028', '2659', '11753', '350029',
'54636', '36963', '219533', '349224', '334912', '27042', '347743',
'13214', '112052', '237668', 'STON/O 2. 3101292', '350050',
'349231', '13213', 'S.O./P.P. 751', 'CA. 2314', '349221', '8475',
'330919', '365226', '349223', '29751', '2623', '5727', '349210',
'STON/O 2. 3101285', '234686', '312993', 'A/5 3536', '19996',
'29750', 'F.C. 12750', 'C.A. 24580', '244270', '239856', '349912',
'342826', '4138', '330935', '6563', '349228', '350036', '24160',
'17474', '349256', '2672', '113800', '248731', '363592', '35852',
'348121', 'PC 17475', '36864', '350025', '223596', 'PC 17476',
'PC 17482', '113028', '7545', '250647', '348124', '34218', '36568',
'347062', '350048', '12233', '250643', '113806', '315094', '36866',
'236853', 'STON/O2. 3101271', '239855', '28425', '233639',
'349201', '349218', '16988', '376566', 'STON/O 2. 3101288',
'250648', '113773', '335097', '29103', '392096', '345780',
'349204', '350042', '29108', '363294', 'SOTON/O2 3101272', '2663',
'347074', '112379', '364850', '8471', '345781', '350047',
'S.O./P.P. 3', '2674', '29105', '347078', '383121', '36865',
'2687', '113501', 'W./C. 6607', 'SOTON/O.Q. 3101312', '374887',
'3101265', '12460', 'PC 17600', '349203', '28213', '17465',
'349244', '2685', '2625', '347089', '347063', '112050', '347087',
'248723', '3474', '28206', '364499', '112058', 'STON/O2. 3101290',
'S.C./PARIS 2079', 'C 7075', '315098', '19972', '368323', '367228',
'2671', '347468', '2223', 'PC 17756', '315097', '392092', '11774',
'SOTON/O2 3101287', '2683', '315090', 'C.A. 5547', '349213',
'347060', 'PC 17592', '392091', '113055', '2629', '350026',
'28134', '17466', '233866', '236852', 'SC/PARIS 2149', 'PC 17590',

```
       '345777', '349248', '695', '345765', '2667', '349212', '349217',
       '349257', '7552', 'C.A./SOTON 34068', 'SOTON/OQ 392076', '211536',
       '112053', '111369', '370376'], dtype=object)
```

##Using Split()

[ ]:
```
df['ticket_num']=df['Ticket'].str.split(' ').str.get(-1) #value akta thakele o␣
↪shes ar ta 2 ta thakle o ses ar ta [Number]
df

#df['Ticket'].str.split(' ')--->['A/5','21171'].str.get(1)--->'21171' but if␣
↪data is '21001'--->'21001' -->NaN see the problem
```

[ ]:
```
    Cabin            Ticket number  Survived  number_numirical  \
0     NaN         A/5 21171      5         0               5.0
1     C85          PC 17599      3         1               3.0
2     NaN  STON/O2. 3101282      6         1               6.0
3    C123            113803      3         1               3.0
4     NaN            373450      A         0               NaN
..    …                …    …       …               …
886   NaN            211536      3         0               3.0
887   B42            112053      3         1               3.0
888   NaN        W./C. 6607      1         0               1.0
889  C148            111369      2         1               2.0
890   NaN            370376      3         0               3.0

    number_catagorical cabin_num cabin_cat ticket_num
0                  NaN       NaN       NaN      21171
1                  NaN        85         C      17599
2                  NaN       NaN       NaN    3101282
3                  NaN       123         C     113803
4                    A       NaN       NaN     373450
..                 …       …       …       …
886                NaN       NaN       NaN     211536
887                NaN        42         B     112053
888                NaN       NaN       NaN       6607
889                NaN       148         C     111369
890                NaN       NaN       NaN     370376

[891 rows x 9 columns]
```

[ ]:
```
df['ticket_cat']=np.where(df['ticket_num']==df['Ticket'].str.split(' ').str.
↪get(0),np.nan,df['Ticket'].str.split(' ').str.get(0))
df
```

[ ]:
```
    Cabin            Ticket number  Survived  number_numirical  \
0     NaN         A/5 21171      5         0               5.0
```

```
1     C85          PC 17599      3         1               3.0
2     NaN  STON/O2. 3101282      6         1               6.0
3    C123            113803      3         1               3.0
4     NaN            373450      A         0               NaN
..    …                …    …       …              …
886   NaN            211536      3         0               3.0
887   B42            112053      3         1               3.0
888   NaN        W./C. 6607      1         0               1.0
889  C148            111369      2         1               2.0
890   NaN            370376      3         0               3.0

    number_catagorical cabin_num cabin_cat ticket_num ticket_cat
0                  NaN       NaN       NaN      21171        A/5
1                  NaN        85         C      17599         PC
2                  NaN       NaN       NaN    3101282   STON/O2.
3                  NaN       123         C     113803        NaN
4                    A       NaN       NaN     373450        NaN
..                 …       …       …       …        …
886                NaN       NaN       NaN     211536        NaN
887                NaN        42         B     112053        NaN
888                NaN       NaN       NaN       6607      W./C.
889                NaN       148         C     111369        NaN
890                NaN       NaN       NaN     370376        NaN

[891 rows x 10 columns]
```

##Using Extract()

0.0.11 Using `str.extract()` (Recommended for Pattern Extraction)

```
# Extract ticket number (digits) and category (letters after space)
df[['ticket_num', 'ticket_cat']] = df['Ticket'].str.extract(r'(\d+)\s*(.*)')
```

0.0.12 How This Works:

1. `(\d+)` - Captures one or more digits (ticket number)
2. `\s*` - Matches any whitespace (spaces/tabs) between parts
3. `(.*)` - Captures everything remaining (category)

[ ]:

#Day-34: Time And Date Data Handeling

- convert data(object type) to date(datetime type)

[ ]:
```
time=pd.read_csv('/content/messages.csv')
time.head(2)
```

[ ]:
```
                 date                                                msg
0  2013-12-15 00:50:00                                                37
1  2014-04-29 23:40:00                  !!      0955532826
```

[ ]:
```
time.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    1000 non-null   object
 1   msg     1000 non-null   object
dtypes: object(2)
memory usage: 15.8+ KB
```

[ ]:
```
time['date']=pd.to_datetime(time['date'])
```

[ ]:
```
time.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    1000 non-null   datetime64[ns]
 1   msg     1000 non-null   object
dtypes: datetime64[ns](1), object(1)
memory usage: 15.8+ KB
```

[ ]:
```
time.head(2)
```

[ ]:
```
                date                                                msg
0 2013-12-15 00:50:00                                                37
1 2014-04-29 23:40:00                  !!      0955532826
```

[ ]:
```
time['calander']=time['date'].dt.date
time.head(2)
```

[ ]:
```
                date                                                msg  \
0 2013-12-15 00:50:00                                                37
1 2014-04-29 23:40:00                  !!      0955532826

     calander
0  2013-12-15
1  2014-04-29
```

[ ]:
```
time['time']=time['date'].dt.time
time.head(2)
```

[ ]:
```
                 date                                                msg  \
0 2013-12-15 00:50:00                                              37
1 2014-04-29 23:40:00                    !!       0955532826

     calander      time
0  2013-12-15  00:50:00
1  2014-04-29  23:40:00
```

[ ]:
```
time['year']=time['date'].dt.year
time['month']=time['date'].dt.month
time['day']=time['date'].dt.day_name()
time.sample(3)
```

[ ]:
```
                   date                                                  msg  \
517 2012-02-25 02:00:00                                                +
83  2014-02-07 23:14:00                                    …
558 2014-10-06 00:00:00                                 …

       calander      time  year  month       day
517  2012-02-25  02:00:00  2012      2  Saturday
83   2014-02-07  23:14:00  2014      2    Friday
558  2014-10-06  00:00:00  2014     10    Monday
```

[ ]:
```
time['is_weekend']=np.where(time['date'].dt.day_name().
 ↪isin(['Friday','Saturday']),1,0)
time.head()
```

[ ]:
```
                 date                                                  msg  \
0 2013-12-15 00:50:00                                              37
1 2014-04-29 23:40:00                    !!       0955532826
2 2012-12-30 00:21:00                 .  43    .   /  *.   067.16.34.576
3 2014-11-28 00:31:00         /    45              093 629 9…
4 2013-10-26 23:11:00                               !)

     calander      time  year  month       day  is_weekend
0  2013-12-15  00:50:00  2013     12    Sunday           0
1  2014-04-29  23:40:00  2014      4   Tuesday           0
2  2012-12-30  00:21:00  2012     12    Sunday           0
3  2014-11-28  00:31:00  2014     11    Friday           1
4  2013-10-26  23:11:00  2013     10  Saturday           1
```

[ ]:
```
time['day_no']=time['date'].dt.day_of_week
time.head()
```

[ ]:

	date	msg
0	2013-12-15 00:50:00	37
1	2014-04-29 23:40:00	!! 0955532826
2	2012-12-30 00:21:00	. 43 . / *. 067.16.34.576
3	2014-11-28 00:31:00	/ 45 093 629 9…
4	2013-10-26 23:11:00	!)

	calander	time	year	month	day	is_weekend	day_no
0	2013-12-15	00:50:00	2013	12	Sunday	0	6
1	2014-04-29	23:40:00	2014	4	Tuesday	0	1
2	2012-12-30	00:21:00	2012	12	Sunday	0	6
3	2014-11-28	00:31:00	2014	11	Friday	1	4
4	2013-10-26	23:11:00	2013	10	Saturday	1	5