STEP-1 Business Problem Understanding

Predict if a passenger survived the sinking of the Titanic or not.

STEP-2 Data Understanding

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.pipeline import Pipeline
import warnings
warnings.simplefilter("ignore")
```

In [2]:
```
df1=pd.read_csv("titanic_train.csv")
df1
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [3]:
```
df1.head(10)
df1
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [4]:
```
df2=pd.read_csv("titanic_test.csv")
df2
```

Out[4]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN	S
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN	C

418 rows × 11 columns

In [5]:
```
df2.head(5)
df2
```

Out[5]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN	S
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN	C

418 rows × 11 columns

In [6]:
```
df1["PassengerId"].value_counts()
```

Out[6]:
```
PassengerId
1      1
599    1
588    1
589    1
590    1
      ..
301    1
302    1
303    1
304    1
891    1
Name: count, Length: 891, dtype: int64
```

In [7]:
```
df1["Name"].value_counts()
```

Out[7]:
```
Name
Braund, Mr. Owen Harris                      1
Boulos, Mr. Hanna                            1
Frolicher-Stehli, Mr. Maxmillian             1
Gilinski, Mr. Eliezer                        1
Murdlin, Mr. Joseph                          1
                                            ..
Kelly, Miss. Anna Katherine "Annie Kate"     1
McCoy, Mr. Bernard                           1
Johnson, Mr. William Cahoone Jr              1
Keane, Miss. Nora A                          1
Dooley, Mr. Patrick                          1
Name: count, Length: 891, dtype: int64
```

In [8]:
```
df1["Pclass"].unique()
```

Out[8]:
```
array([3, 1, 2], dtype=int64)
```

In [9]:
```
df1["Pclass"].value_counts()
```

Out[9]:
```
Pclass
3    491
1    216
2    184
Name: count, dtype: int64
```

In [10]:
```
df1["Sex"].unique()
```

Out[10]:
```
array(['male', 'female'], dtype=object)
```

In [11]:
```
df1["Sex"].value_counts()
```

Out[11]:
```
Sex
male      577
female    314
Name: count, dtype: int64
```

In [12]:
```
df1["Survived"].unique()
```

Out[12]:
```
array([0, 1], dtype=int64)
```

In [13]:
```
df1["Survived"].value_counts()
```

Out[13]:
```
Survived
0    549
1    342
Name: count, dtype: int64
```

In [14]:
```
df1["Age"].unique()
```

Out[14]:
```
array([22.  , 38.  , 26.  , 35.  ,   nan, 54.  ,  2.  , 27.  , 14.  ,
        4.  , 58.  , 20.  , 39.  , 55.  , 31.  , 34.  , 15.  , 28.  ,
        8.  , 19.  , 40.  , 66.  , 42.  , 21.  , 18.  ,  3.  ,  7.  ,
       49.  , 29.  , 65.  , 28.5 ,  5.  , 11.  , 45.  , 17.  , 32.  ,
       16.  , 25.  ,  0.83, 30.  , 33.  , 23.  , 24.  , 46.  , 59.  ,
       71.  , 37.  , 47.  , 14.5 , 70.5 , 32.5 , 12.  ,  9.  , 36.5 ,
       51.  , 55.5 , 40.5 , 44.  ,  1.  , 61.  , 56.  , 50.  , 36.  ,
       45.5 , 20.5 , 62.  , 41.  , 52.  , 63.  , 23.5 ,  0.92, 43.  ,
       60.  , 10.  , 64.  , 13.  , 48.  ,  0.75, 53.  , 57.  , 80.  ,
       70.  , 24.5 ,  6.  ,  0.67, 30.5 ,  0.42, 34.5 , 74.  ])
```

In [15]:
```
df1["Age"].value_counts()
```

Out[15]:
```
Age
24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
         ..
36.50     1
55.50     1
0.92      1
23.50     1
74.00     1
Name: count, Length: 88, dtype: int64
```

In [16]:
```
df1["SibSp"].unique()
```

Out[16]:
```
array([1, 0, 3, 4, 2, 5, 8], dtype=int64)
```

In [17]:
```
df1["SibSp"].value_counts()
```

Out[17]:
```
SibSp
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: count, dtype: int64
```

In [18]:
```
df1["Parch"].unique()
```

Out[18]:
```
array([0, 1, 2, 5, 3, 4, 6], dtype=int64)
```

In [19]:
```
df1["Parch"].value_counts()
```

Out[19]:
```
Parch
0    678
1    118
2     80
5      5
3      5
4      4
6      1
Name: count, dtype: int64
```

In [20]:
```
df1["Ticket"].unique()
```

Out[20]:
```
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)
```

In [21]:
```
df1["Ticket"].value_counts()
```

Out[21]:
```
Ticket
347082      7
CA. 2343    7
1601        7
3101295     6
CA 2144     6
           ..
9234        1
19988       1
2693        1
PC 17612    1
370376      1
Name: count, Length: 681, dtype: int64
```

In [22]:
```
df1["Fare"].unique()
```

Out[22]:
```
array([  7.25  ,  71.2833,   7.925 ,  53.1   ,   8.05  ,   8.4583,
        51.8625,  21.075 ,  11.1333,  30.0708,  16.7   ,  26.55  ,
        31.275 ,   7.8542,  16.    ,  29.125 ,  13.    ,  18.    ,
         7.225 ,  26.    ,   8.0292,  35.5   ,  31.3875, 263.    ,
         7.8792,   7.8958,  27.7208, 146.5208,   7.75  ,  10.5   ,
        82.1708,  52.    ,   7.2292,  11.2417,   9.475 ,  21.    ,
        41.5792,  15.5   ,  21.6792,  17.8   ,  39.6875,   7.8   ,
        76.7292,  61.9792,  27.75  ,  46.9   ,  80.    ,  83.475 ,
        27.9   ,  15.2458,   8.1583,   8.6625,  73.5   ,  14.4542,
        56.4958,   7.65  ,  29.    ,  12.475 ,   9.    ,   9.5   ,
         7.7875,  47.1   ,  15.85  ,  34.375 ,  61.175 ,  20.575 ,
        34.6542,  63.3583,  23.    ,  77.2875,   8.6542,   7.775 ,
        24.15  ,   9.825 ,  14.4583, 247.5208,   7.1417,  22.3583,
         6.975 ,   7.05  ,  14.5   ,  15.0458,  26.2833,   9.2167,
        79.2   ,   6.75  ,  11.5   ,  36.75  ,   7.7958,  12.525 ,
        66.6   ,   7.3125,  61.3792,   7.7333,  69.55  ,  16.1   ,
        15.75  ,  20.525 ,  55.    ,  25.925 ,  33.5   ,  30.6958,
        25.4667,  28.7125,   0.    ,  15.05  ,  39.    ,  22.025 ,
        50.    ,   8.4042,   6.4958,  10.4625,  18.7875,  31.    ,
       113.275 ,  27.    ,  76.2917,  90.    ,   9.35  ,  13.5   ,
         7.55  ,  26.25  ,  12.275 ,   7.125 ,  52.5542,  20.2125,
        86.5   , 512.3292,  79.65  , 153.4625, 135.6333,  19.5   ,
        29.7   ,  77.9583,  20.25  ,  78.85  ,  91.0792,  12.875 ,
         8.85  , 151.55  ,  30.5   ,  23.25  ,  12.35  , 110.8833,
       108.9   ,  24.    ,  56.9292,  83.1583, 262.375 ,  14.    ,
       164.8667, 134.5   ,   6.2375,  57.9792,  28.5   , 133.65  ,
        15.9   ,   9.225 ,  35.    ,  75.25  ,  69.3   ,  55.4417,
       211.5   ,   4.0125, 227.525 ,  15.7417,   7.7292,  12.    ,
       120.    ,  12.65  ,  18.75  ,   6.8583,  32.5   ,   7.875 ,
        14.4   ,  55.9   ,   8.1125,  81.8583,  19.2583,  19.9667,
        89.1042,  38.5   ,   7.725 ,  13.7917,   9.8375,   7.0458,
         7.5208,  12.2875,   9.5875,  49.5042,  78.2667,  15.1   ,
         7.6292,  22.525 ,  26.2875,  59.4   ,   7.4958,  34.0208,
        93.5   , 221.7792, 106.425 ,  49.5   ,  71.    ,  13.8625,
         7.8292,  39.6   ,  17.4   ,  51.4792,  26.3875,  30.    ,
        40.125 ,   8.7125,  15.    ,  33.    ,  42.4   ,  15.55  ,
        65.    ,  32.3208,   7.0542,   8.4333,  25.5875,   9.8417,
         8.1375,  10.1708, 211.3375,  57.    ,  13.4167,   7.7417,
         9.4833,   7.7375,   8.3625,  23.45  ,  25.9292,   8.6833,
         8.5167,   7.8875,  37.0042,   6.45  ,   6.95  ,   8.3   ,
         6.4375,  39.4   ,  14.1083,  13.8583,  50.4958,   5.    ,
         9.8458,  10.5167])
```

In [23]:
```
df1["Fare"].value_counts()
```

Out[23]:
```
Fare
8.0500     43
13.0000    42
7.8958     38
7.7500     34
26.0000    31
           ..
35.0000     1
28.5000     1
6.2375      1
14.0000     1
10.5167     1
Name: count, Length: 248, dtype: int64
```

In [24]:
```
df1["Cabin"].unique()
```

Out[24]:
```
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)
```

In [25]:
```
df1["Cabin"].value_counts()
```

Out[25]:
```
Cabin
B96 B98        4
G6             4
C23 C25 C27    4
C22 C26        3
F33            3
              ..
E34            1
C7             1
C54            1
E36            1
C148           1
Name: count, Length: 147, dtype: int64
```

In [26]:
```
df1["Embarked"].unique()
```

Out[26]:
```
array(['S', 'C', 'Q', nan], dtype=object)
```

In [27]:
```
df1["Embarked"].value_counts()
```

Out[27]:
```
Embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```

Exploratory Data Analysis

For continuous and discrete variables

In [28]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [29]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

In [30]:
```
df1.continuous = ["Age","Fare","PassengerId","Pclass","SibSp","Parch","Survived"]
df1.discrete_categorical = ["Sex","Ticket","Embarked","Cabin"]
```

In [31]:
```
df2.continuous = ["Age","Fare","PassengerId","Pclass","SibSp","Parch"]
df2.discrete_categorical = ["Sex","Ticket","Embarked","Cabin"]
```

In [32]:
```
df1[df1.continuous].describe()
```

Out[32]:

	Age	Fare	PassengerId	Pclass	SibSp	Parch	Survived
count	714.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	29.699118	32.204208	446.000000	2.308642	0.523008	0.381594	0.383838
std	14.526497	49.693429	257.353842	0.836071	1.102743	0.806057	0.486592
min	0.420000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000
25%	20.125000	7.910400	223.500000	2.000000	0.000000	0.000000	0.000000
50%	28.000000	14.454200	446.000000	3.000000	0.000000	0.000000	0.000000
75%	38.000000	31.000000	668.500000	3.000000	1.000000	0.000000	1.000000
max	80.000000	512.329200	891.000000	3.000000	8.000000	6.000000	1.000000

In [33]:

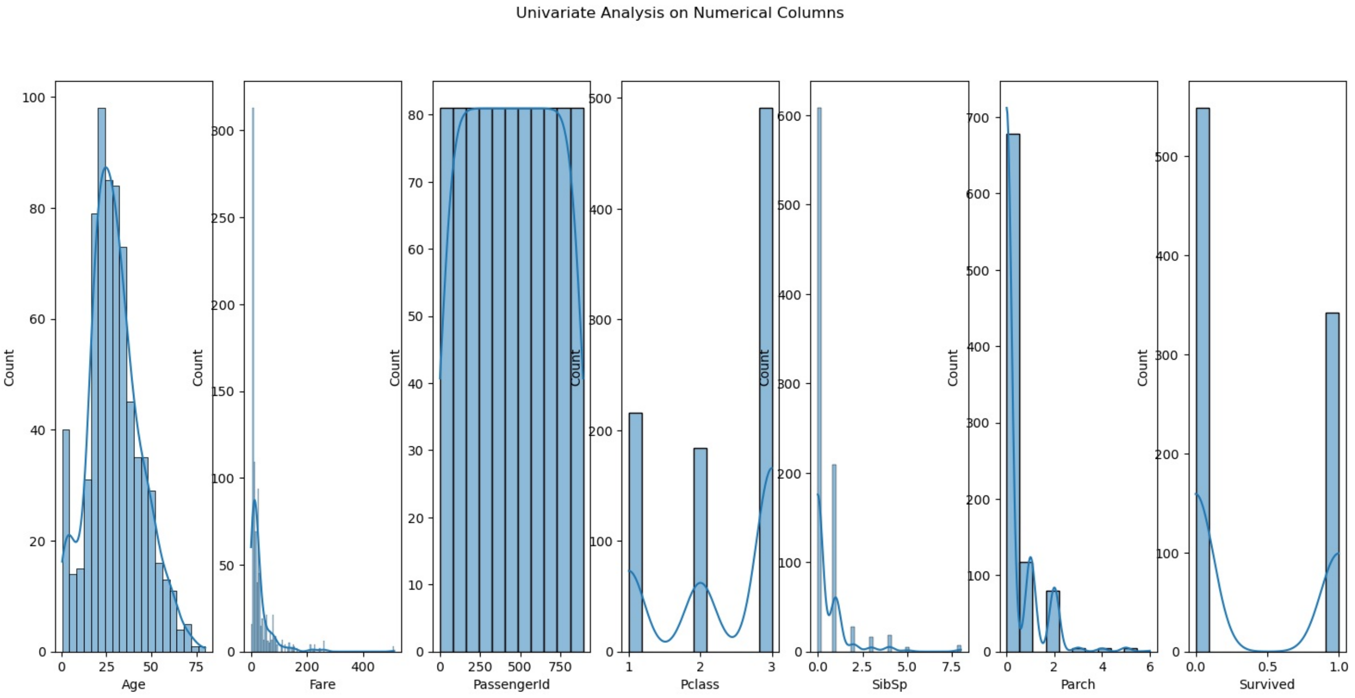
```
df2[df2.continuous].describe()
```

Out[33]:

	Age	Fare	PassengerId	Pclass	SibSp	Parch
count	332.000000	417.000000	418.000000	418.000000	418.000000	418.000000
mean	30.272590	35.627188	1100.500000	2.265550	0.447368	0.392344
std	14.181209	55.907576	120.810458	0.841838	0.896760	0.981429
min	0.170000	0.000000	892.000000	1.000000	0.000000	0.000000
25%	21.000000	7.895800	996.250000	1.000000	0.000000	0.000000
50%	27.000000	14.454200	1100.500000	3.000000	0.000000	0.000000
75%	39.000000	31.500000	1204.750000	3.000000	1.000000	0.000000
max	76.000000	512.329200	1309.000000	3.000000	8.000000	9.000000

In [34]:

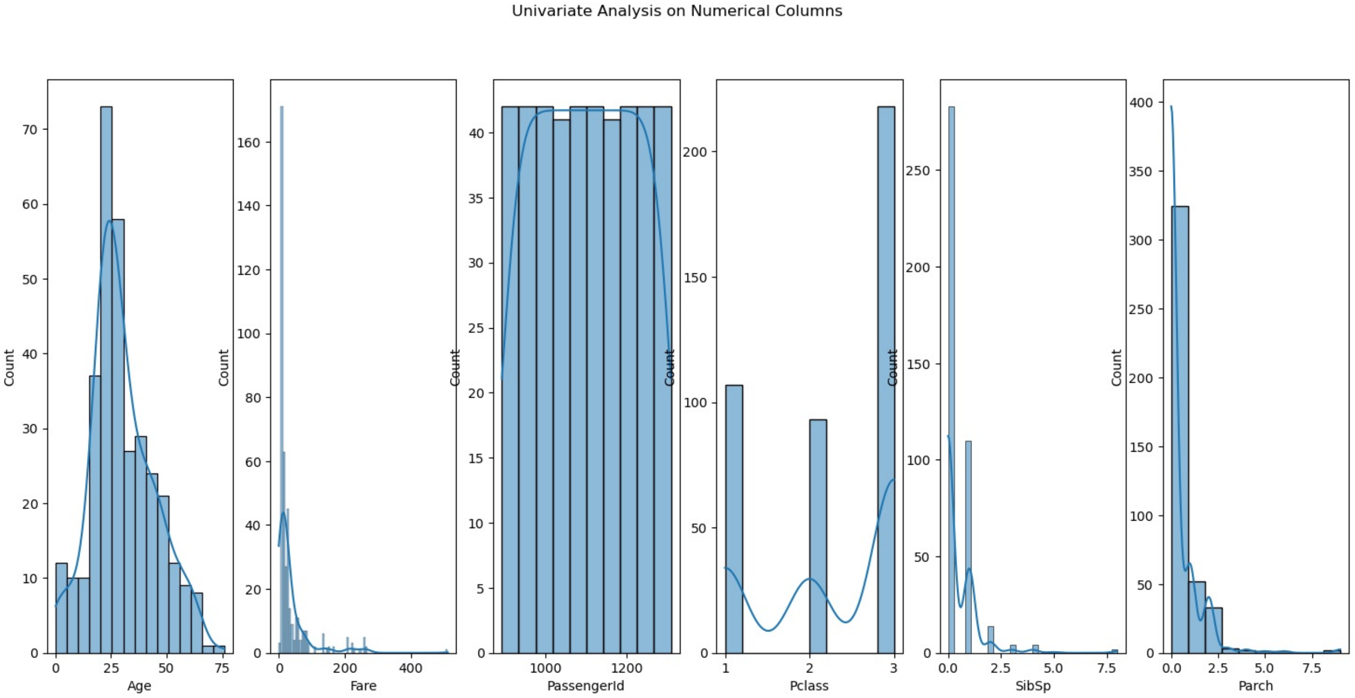
```
plt.rcParams["figure.figsize"] = (18,8)
plt.subplot(1,7, 1)
sns.histplot(df1["Age"],kde=True)
plt.subplot(1,7, 2)
sns.histplot(df1["Fare"],kde=True)
plt.subplot(1,7, 3)
sns.histplot(df1["PassengerId"],kde=True)
plt.subplot(1,7, 4)
sns.histplot(df1["Pclass"],kde=True)
plt.subplot(1,7, 5)
sns.histplot(df1["SibSp"],kde=True)
plt.subplot(1,7, 6)
sns.histplot(df1["Parch"],kde=True)
plt.subplot(1,7, 7)
sns.histplot(df1["Survived"],kde=True)
plt.suptitle("Univariate Analysis on Numerical Columns")
plt.show()
```



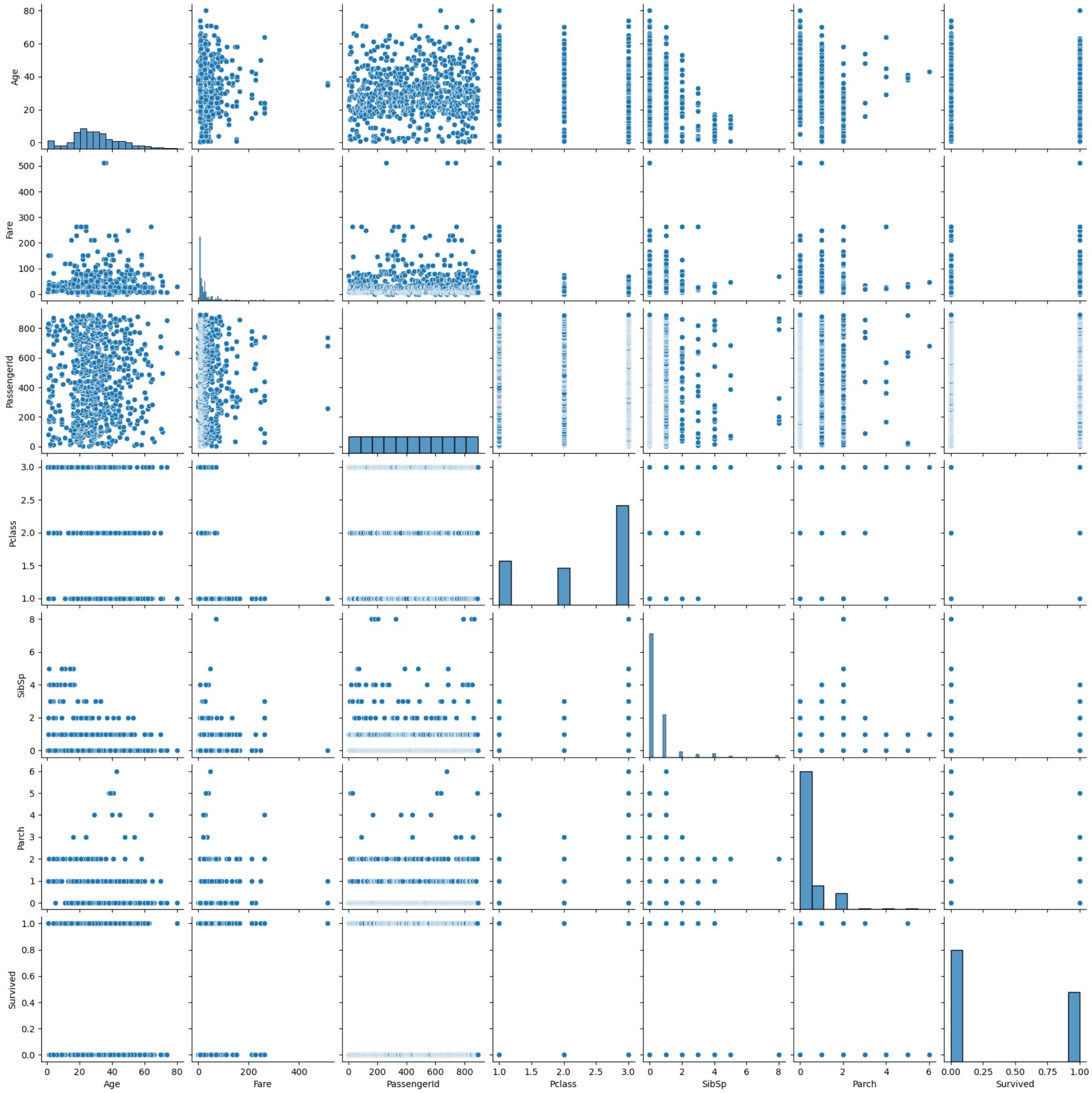
In [35]:

```
plt.rcParams["figure.figsize"] = (18,8)
plt.subplot(1,6, 1)
sns.histplot(df2["Age"],kde=True)
```

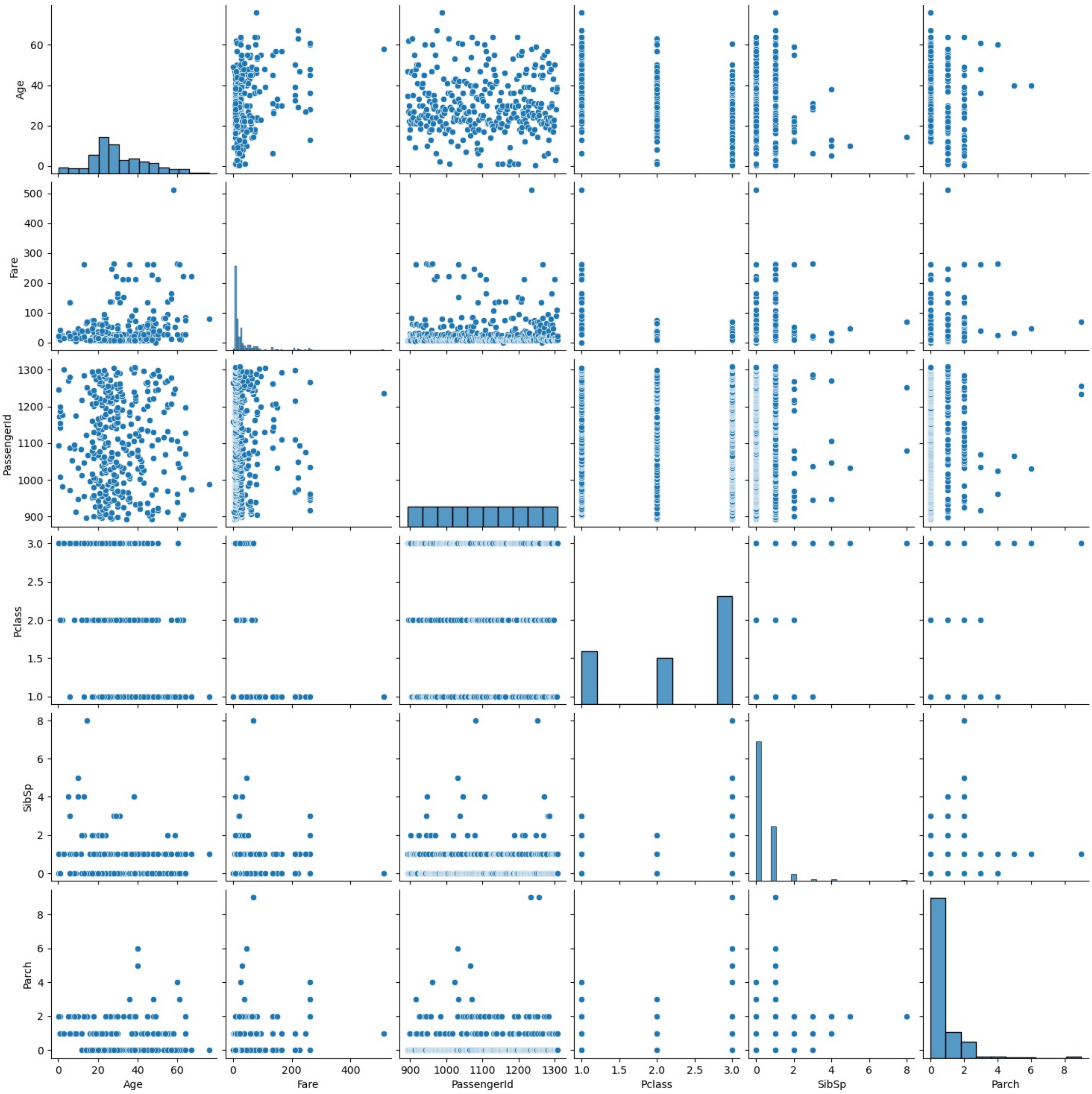
```
plt.subplot(1,6, 2)
sns.histplot(df2["Fare"],kde=True)
plt.subplot(1,6, 3)
sns.histplot(df2["PassengerId"],kde=True)
plt.subplot(1,6, 4)
sns.histplot(df2["Pclass"],kde=True)
plt.subplot(1,6, 5)
sns.histplot(df2["SibSp"],kde=True)
plt.subplot(1,6, 6)
sns.histplot(df2["Parch"],kde=True)
plt.suptitle("Univariate Analysis on Numerical Columns")
plt.show()
```



In [36]:
```
sns.pairplot(df1[df1.continuous])
plt.show()
```



In [37]:
```
sns.pairplot(df2[df2.continuous])
plt.show()
```



In [38]:
```
df1[df1.continuous].corr()
```

Out[38]:

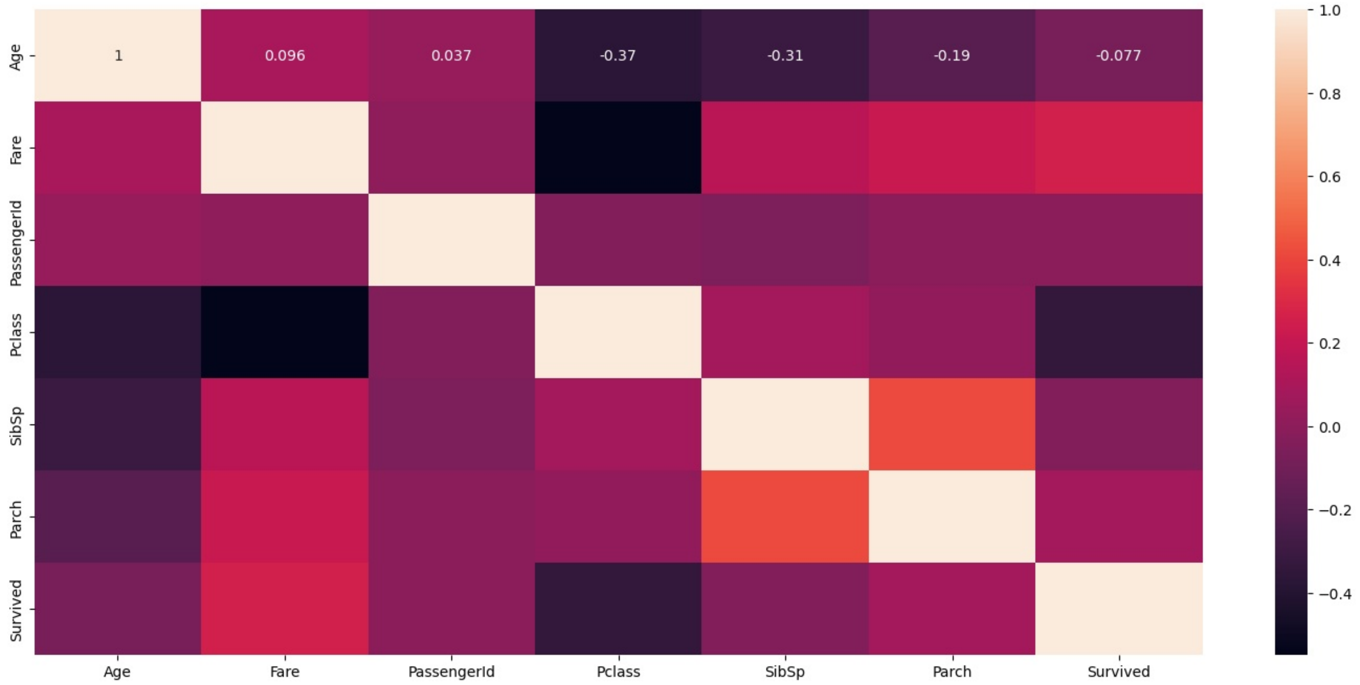
	Age	Fare	PassengerId	Pclass	SibSp	Parch	Survived
Age	1.000000	0.096067	0.036847	-0.369226	-0.308247	-0.189119	-0.077221
Fare	0.096067	1.000000	0.012658	-0.549500	0.159651	0.216225	0.257307
PassengerId	0.036847	0.012658	1.000000	-0.035144	-0.057527	-0.001652	-0.005007
Pclass	-0.369226	-0.549500	-0.035144	1.000000	0.083081	0.018443	-0.338481
SibSp	-0.308247	0.159651	-0.057527	0.083081	1.000000	0.414838	-0.035322
Parch	-0.189119	0.216225	-0.001652	0.018443	0.414838	1.000000	0.081629
Survived	-0.077221	0.257307	-0.005007	-0.338481	-0.035322	0.081629	1.000000

In [39]:
```
df2[df2.continuous].corr()
```

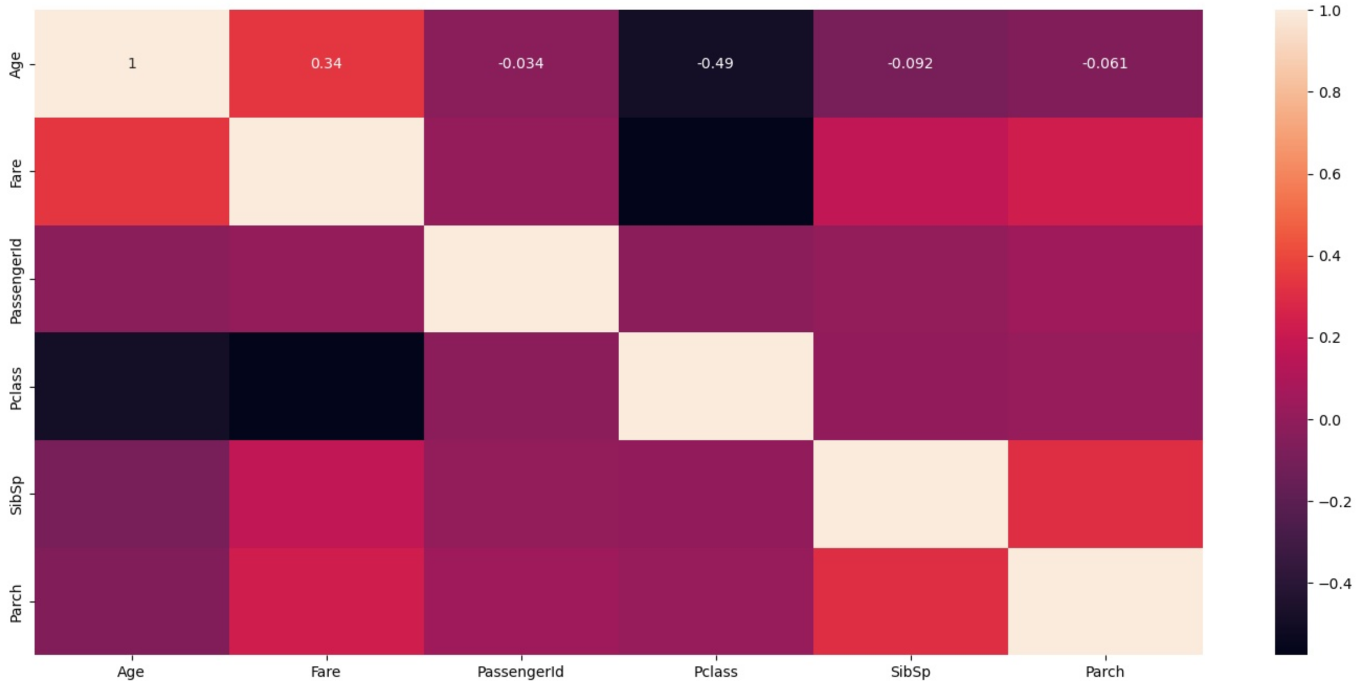
Out[39]:

	Age	Fare	PassengerId	Pclass	SibSp	Parch
Age	1.000000	0.337932	-0.034102	-0.492143	-0.091587	-0.061249
Fare	0.337932	1.000000	0.008211	-0.577147	0.171539	0.230046
PassengerId	-0.034102	0.008211	1.000000	-0.026751	0.003818	0.043080
Pclass	-0.492143	-0.577147	-0.026751	1.000000	0.001087	0.018721
SibSp	-0.091587	0.171539	0.003818	0.001087	1.000000	0.306895
Parch	-0.061249	0.230046	0.043080	0.018721	0.306895	1.000000

In [40]:
```
sns.heatmap(df1[df1.continuous].corr(),annot=True)
plt.show()
```



In [41]:
```
sns.heatmap(df2[df2.continuous].corr(),annot=True)
plt.show()
```



In [42]:
```
df1[df1.discrete_categorical].describe()
```

Out[42]:

	Sex	Ticket	Embarked	Cabin
count	891	891	889	204
unique	2	681	3	147
top	male	347082	S	B96 B98
freq	577	7	644	4

In [43]:
```
df2[df2.discrete_categorical].describe()
```

Out[43]:

	Sex	Ticket	Embarked	Cabin
count	418	418	418	91
unique	2	363	3	76
top	male	PC 17608	S	B57 B59 B63 B66
freq	266	5	270	3

Steps to be followed for data cleaning

Check for wrong data

Check for wrong Datatype

In [44]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [45]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

Check for Duplicates

In [46]:
```
df1.duplicated().sum()
```

Out[46]: 0

In [47]:
```
df2.duplicated().sum()
```

Out[47]: 0

Check for Missing Values

In [48]:
```
df1.isnull().sum()
```

Out[48]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [49]:
```
df1.isnull().sum()/len(df1)*100
```

Out[49]:
```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```

In [50]:
```
df2.isnull().sum()
```

Out[50]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [51]:
```
df2.isnull().sum()/len(df2)*100
```

Out[51]:
```
PassengerId     0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            20.574163
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.239234
Cabin          78.229665
Embarked        0.000000
dtype: float64
```

Check for Skewness

In [52]:
```
df1[["Age","Fare","PassengerId","Pclass","SibSp","Parch","Survived"]].skew()
```

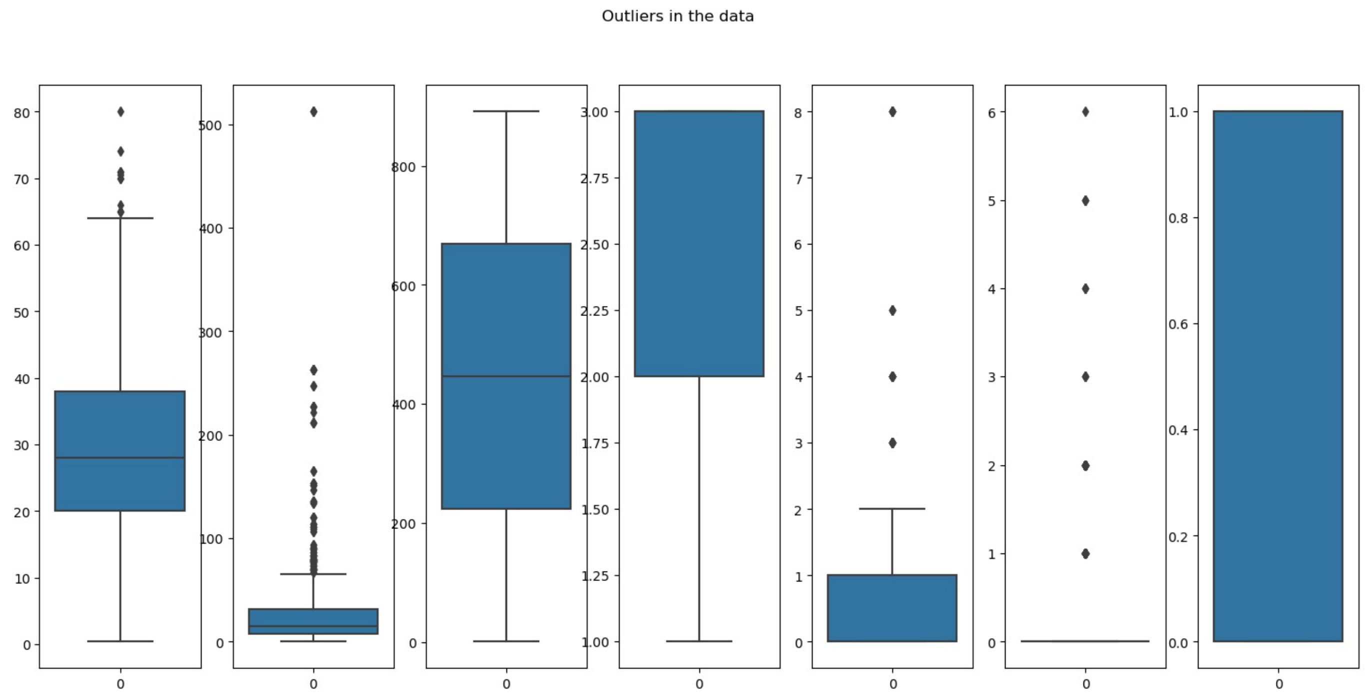
Out[52]:
```
Age            0.389108
Fare           4.787317
PassengerId    0.000000
Pclass        -0.630548
SibSp          3.695352
Parch          2.749117
Survived       0.478523
dtype: float64
```

In [53]:
```
df2[["Age","Fare","PassengerId","Pclass","SibSp","Parch"]].skew()
```

Out[53]:
```
Age            0.457361
Fare           3.687213
PassengerId    0.000000
Pclass        -0.534170
SibSp          4.168337
Parch          4.654462
dtype: float64
```

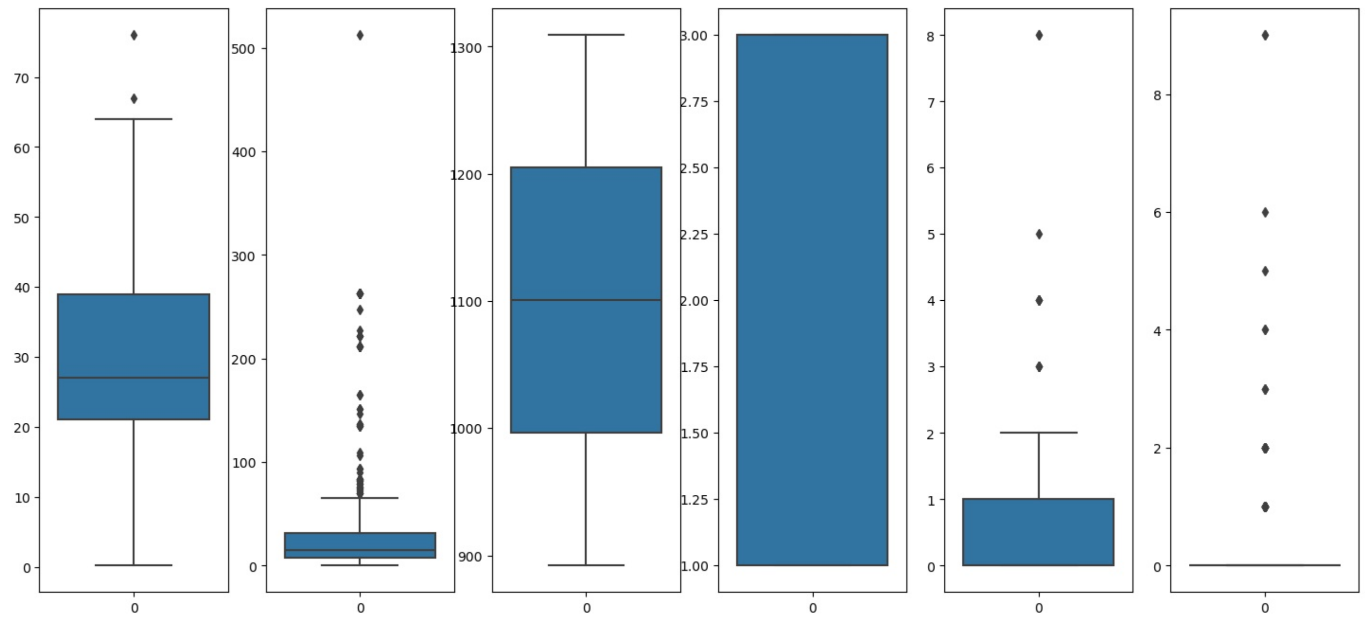
Check for Outliers

In [54]:
```python
# Lets visualize the outliers using Boxplot
plt.subplot(1,7, 1)
sns.boxplot(df1["Age"])
plt.subplot(1,7, 2)
sns.boxplot(df1["Fare"])
plt.subplot(1,7, 3)
sns.boxplot(df1["PassengerId"])
plt.subplot(1,7, 4)
sns.boxplot(df1["Pclass"])
plt.subplot(1,7, 5)
sns.boxplot(df1["SibSp"])
plt.subplot(1,7, 6)
sns.boxplot(df1["Parch"])
plt.subplot(1,7, 7)
sns.boxplot(df1["Survived"])
plt.suptitle("Outliers in the data")
plt.show()
```



In [55]:
```python
plt.subplot(1,6, 1)
sns.boxplot(df2["Age"])
plt.subplot(1,6, 2)
sns.boxplot(df2["Fare"])
plt.subplot(1,6, 3)
sns.boxplot(df2["PassengerId"])
plt.subplot(1,6, 4)
sns.boxplot(df2["Pclass"])
plt.subplot(1,6, 5)
sns.boxplot(df2["SibSp"])
plt.subplot(1,6, 6)
sns.boxplot(df2["Parch"])

plt.suptitle("Outliers in the data")
plt.show()
```

Outliers in the data



STEP-3 DATA PREPROCESSING

I.Data Cleaning

No treatment for wrong data

No treatment for wrong datatype

No treatment for duplicates

In [56]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [57]:
```
df1.drop(columns=["PassengerId","Name","Ticket","Cabin"],inplace=True)
df1
```

Out[57]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S
887	1	1	female	19.0	0	0	30.0000	S
888	0	3	female	NaN	1	2	23.4500	S
889	1	1	male	26.0	0	0	30.0000	C
890	0	3	male	32.0	0	0	7.7500	Q

891 rows × 8 columns

In [58]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

In [59]:

```
df2.drop(columns=["Name","Ticket","Cabin"],inplace=True)
df2
```

Out[59]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	male	34.5	0	0	7.8292	Q
1	893	3	female	47.0	1	0	7.0000	S
2	894	2	male	62.0	0	0	9.6875	Q
3	895	3	male	27.0	0	0	8.6625	S
4	896	3	female	22.0	1	1	12.2875	S
...	...	...	...	...	...	...	...	...
413	1305	3	male	NaN	0	0	8.0500	S
414	1306	1	female	39.0	0	0	108.9000	C
415	1307	3	male	38.5	0	0	7.2500	S
416	1308	3	male	NaN	0	0	8.0500	S
417	1309	3	male	NaN	1	1	22.3583	C

418 rows × 8 columns

Treating Missing Values

In [60]:

```
df1.isnull().sum()
```

Out[60]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

In [61]:
```
df1["Age"].mean()
```

Out[61]:
```
29.69911764705882
```

In [62]:
```
df1['Age'].fillna(df1['Age'].median(), inplace=True)
df1
```

Out[62]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S
887	1	1	female	19.0	0	0	30.0000	S
888	0	3	female	28.0	1	2	23.4500	S
889	1	1	male	26.0	0	0	30.0000	C
890	0	3	male	32.0	0	0	7.7500	Q

891 rows × 8 columns

In [63]:
```
df1["Embarked"].fillna(df1["Embarked"].mode()[0],inplace=True)
df1
```

Out[63]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S
887	1	1	female	19.0	0	0	30.0000	S
888	0	3	female	28.0	1	2	23.4500	S
889	1	1	male	26.0	0	0	30.0000	C
890	0	3	male	32.0	0	0	7.7500	Q

891 rows × 8 columns

In [64]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df1["Sex"] = le.fit_transform(df1["Sex"])
df1
```

Out[64]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	1	0	7.2500	S
1	1	1	0	38.0	1	0	71.2833	C
2	1	3	0	26.0	0	0	7.9250	S
3	1	1	0	35.0	1	0	53.1000	S
4	0	3	1	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...
886	0	2	1	27.0	0	0	13.0000	S
887	1	1	0	19.0	0	0	30.0000	S
888	0	3	0	28.0	1	2	23.4500	S
889	1	1	1	26.0	0	0	30.0000	C
890	0	3	1	32.0	0	0	7.7500	Q

891 rows × 8 columns

In [65]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df1["Embarked"] = le.fit_transform(df1["Embarked"])
df1
```

Out[65]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	1	0	7.2500	2
1	1	1	0	38.0	1	0	71.2833	0
2	1	3	0	26.0	0	0	7.9250	2
3	1	1	0	35.0	1	0	53.1000	2
4	0	3	1	35.0	0	0	8.0500	2
...	...	...	...	...	...	...	...	...
886	0	2	1	27.0	0	0	13.0000	2
887	1	1	0	19.0	0	0	30.0000	2
888	0	3	0	28.0	1	2	23.4500	2
889	1	1	1	26.0	0	0	30.0000	0
890	0	3	1	32.0	0	0	7.7500	1

891 rows × 8 columns

In [66]:

```
df1
```

Out[66]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	1	0	7.2500	2
1	1	1	0	38.0	1	0	71.2833	0
2	1	3	0	26.0	0	0	7.9250	2
3	1	1	0	35.0	1	0	53.1000	2
4	0	3	1	35.0	0	0	8.0500	2
...	...	...	...	...	...	...	...	...
886	0	2	1	27.0	0	0	13.0000	2
887	1	1	0	19.0	0	0	30.0000	2
888	0	3	0	28.0	1	2	23.4500	2
889	1	1	1	26.0	0	0	30.0000	0
890	0	3	1	32.0	0	0	7.7500	1

891 rows × 8 columns

In [67]:

```
df2.isnull().sum()
```

Out[67]:
```
PassengerId     0
Pclass          0
Sex             0
Age            86
SibSp           0
Parch           0
Fare            1
Embarked        0
dtype: int64
```

In [68]:
```
df2["Age"].mean()
```

Out[68]:
```
30.272590361445783
```

In [69]:
```
df2['Age'].fillna(df2['Age'].median(), inplace=True)
df2
```

Out[69]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	male	34.5	0	0	7.8292	Q
1	893	3	female	47.0	1	0	7.0000	S
2	894	2	male	62.0	0	0	9.6875	Q
3	895	3	male	27.0	0	0	8.6625	S
4	896	3	female	22.0	1	1	12.2875	S
...	...	...	...	...	...	...	...	...
413	1305	3	male	27.0	0	0	8.0500	S
414	1306	1	female	39.0	0	0	108.9000	C
415	1307	3	male	38.5	0	0	7.2500	S
416	1308	3	male	27.0	0	0	8.0500	S
417	1309	3	male	27.0	1	1	22.3583	C

418 rows × 8 columns

In [70]:
```
df2['Fare'].fillna(df2['Fare'].median(), inplace=True)
df2
```

Out[70]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	male	34.5	0	0	7.8292	Q
1	893	3	female	47.0	1	0	7.0000	S
2	894	2	male	62.0	0	0	9.6875	Q
3	895	3	male	27.0	0	0	8.6625	S
4	896	3	female	22.0	1	1	12.2875	S
...	...	...	...	...	...	...	...	...
413	1305	3	male	27.0	0	0	8.0500	S
414	1306	1	female	39.0	0	0	108.9000	C
415	1307	3	male	38.5	0	0	7.2500	S
416	1308	3	male	27.0	0	0	8.0500	S
417	1309	3	male	27.0	1	1	22.3583	C

418 rows × 8 columns

In [71]:
```
df2
```

Out[71]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	male	34.5	0	0	7.8292	Q
1	893	3	female	47.0	1	0	7.0000	S
2	894	2	male	62.0	0	0	9.6875	Q
3	895	3	male	27.0	0	0	8.6625	S
4	896	3	female	22.0	1	1	12.2875	S
...	...	...	...	...	...	...	...	...
413	1305	3	male	27.0	0	0	8.0500	S
414	1306	1	female	39.0	0	0	108.9000	C
415	1307	3	male	38.5	0	0	7.2500	S
416	1308	3	male	27.0	0	0	8.0500	S
417	1309	3	male	27.0	1	1	22.3583	C

418 rows × 8 columns

In [72]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df2["Sex"] = le.fit_transform(df2["Sex"])
df2
```

Out[72]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	1	34.5	0	0	7.8292	Q
1	893	3	0	47.0	1	0	7.0000	S
2	894	2	1	62.0	0	0	9.6875	Q
3	895	3	1	27.0	0	0	8.6625	S
4	896	3	0	22.0	1	1	12.2875	S
...	...	...	...	...	...	...	...	...
413	1305	3	1	27.0	0	0	8.0500	S
414	1306	1	0	39.0	0	0	108.9000	C
415	1307	3	1	38.5	0	0	7.2500	S
416	1308	3	1	27.0	0	0	8.0500	S
417	1309	3	1	27.0	1	1	22.3583	C

418 rows × 8 columns

In [73]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df2["Embarked"] = le.fit_transform(df2["Embarked"])
df2
```

Out[73]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	1	34.5	0	0	7.8292	1
1	893	3	0	47.0	1	0	7.0000	2
2	894	2	1	62.0	0	0	9.6875	1
3	895	3	1	27.0	0	0	8.6625	2
4	896	3	0	22.0	1	1	12.2875	2
...	...	...	...	...	...	...	...	...
413	1305	3	1	27.0	0	0	8.0500	2
414	1306	1	0	39.0	0	0	108.9000	0
415	1307	3	1	38.5	0	0	7.2500	2
416	1308	3	1	27.0	0	0	8.0500	2
417	1309	3	1	27.0	1	1	22.3583	0

418 rows × 8 columns

In [74]:

```
df1.isnull().sum()
```

Out[74]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

In [75]:
```
df2.isnull().sum()
```

Out[75]:
```
PassengerId    0
Pclass         0
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
Embarked       0
dtype: int64
```

No treatment for Outliers

In this case, outliers should be retrained(if we change the values then we can't get the accurate results)

II.Data Wrangling

Treating Skewness

In [76]:
```
df1[["Age","Fare","Pclass","SibSp","Parch","Survived"]].skew()
```

Out[76]:
```
Age         0.510245
Fare        4.787317
Pclass     -0.630548
SibSp       3.695352
Parch       2.749117
Survived    0.478523
dtype: float64
```

In [77]:
```
df2[["Age","Fare","Pclass","SibSp","Parch"]].skew()
```

Out[77]:
```
Age       0.660747
Fare      3.692299
Pclass   -0.534170
SibSp     4.168337
Parch     4.654462
dtype: float64
```

In [78]:
```
df1["SibSp"]=np.log1p(df1["SibSp"])
df1["SibSp"].skew()
```

Out[78]:
```
1.6612454204052132
```

In [79]:
```
df1["SibSp"]=np.sqrt(df1["SibSp"])
df1["SibSp"].skew()
```

Out[79]:
```
0.9672484745036443
```

In [80]:
```
from scipy.stats import boxcox
df1["SibSp"], param = boxcox(df1["SibSp"] + 1)
df1["SibSp"].skew()
```

Out[80]:
```
0.7879894636695959
```

In [81]:
```
df1["Fare"]=np.log1p(df1["Fare"])
df1["Fare"].skew()
```

Out[81]:
```
0.3949280095189306
```

In [82]:
```
df1["Parch"]=np.log1p(df1["Parch"])
df1["Parch"].skew()
```

Out[82]:
```
1.6754394553891907
```

In [83]:
```
df1["Parch"]=np.sqrt(df1["Parch"])
df1["Parch"].skew()
```

Out[83]: 1.3277619890454673

In [84]:
```
from scipy.stats import boxcox
df1["Parch"], param = boxcox(df1["Parch"] + 1)
df1["Parch"].skew()
```

Out[84]: 1.2259981794888024

In [85]:
```
df1[["Fare","SibSp","Parch"]].skew()
```

Out[85]:
```
Fare     0.394928
SibSp    0.787989
Parch    1.225998
dtype: float64
```

In [86]:
```
df1
```

Out[86]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	0.243915	0.00000	2.110213	2
1	1	1	0	38.0	0.243915	0.00000	4.280593	0
2	1	3	0	26.0	0.000000	0.00000	2.188856	2
3	1	1	0	35.0	0.243915	0.00000	3.990834	2
4	0	3	1	35.0	0.000000	0.00000	2.202765	2
...	...	...	...	...	...	...	...	...
886	0	2	1	27.0	0.000000	0.00000	2.639057	2
887	1	1	0	19.0	0.000000	0.00000	3.433987	2
888	0	3	0	28.0	0.243915	0.17108	3.196630	2
889	1	1	1	26.0	0.000000	0.00000	3.433987	0
890	0	3	1	32.0	0.000000	0.00000	2.169054	1

891 rows × 8 columns

In [87]:
```
df2["SibSp"]=np.log1p(df2["SibSp"])
df2["SibSp"].skew()
```

Out[87]: 1.5279477471579679

In [88]:
```
df2["SibSp"]=np.sqrt(df2["SibSp"])
df2["SibSp"].skew()
```

Out[88]: 0.8922146262951693

In [89]:
```
from scipy.stats import boxcox
df2["SibSp"], param = boxcox(df2["SibSp"] + 1)
df2["SibSp"].skew()
```

Out[89]: 0.7621261596334947

In [90]:
```
df2["Parch"]=np.log1p(df2["Parch"])
df2["Parch"].skew()
```

Out[90]: 2.0216027756280854

In [91]:
```
df2["Parch"]=np.sqrt(df2["Parch"])
df2["Parch"].skew()
```

Out[91]: 1.463892888346309

In [92]:
```
from scipy.stats import boxcox
df2["Parch"], param = boxcox(df2["Parch"] + 1)
df2["Parch"].skew()
```

Out[92]: 1.3228931130506922

In [93]:
```
df2["Fare"]=np.log1p(df2["Fare"])
df2["Fare"].skew()
```

Out[93]: 0.86495458308337

In [94]:
```
df2["Fare"]=np.sqrt(df2["Fare"])
df2["Fare"].skew()
```

Out[94]: -0.38477377469192947

In [95]:
```python
from scipy.stats import boxcox
df2["Fare"], param = boxcox(df2["Fare"] + 1)
df2["Fare"].skew()
```

Out[95]: 0.21150943837461503

In [96]:
```python
df2[["Fare","SibSp","Parch"]].skew()
```

Out[96]:
```
Fare     0.211509
SibSp    0.762126
Parch    1.322893
dtype: float64
```

No Feature Scaling (Because all the values are in normalized format)

X&y (Train Based)

In [97]:
```python
X= df1.drop("Survived",axis=1)
y = df1["Survived"]
```

Identify the best random number

In [98]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

Train = []
Test = []
CV = []

for i in range(0, 101):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=i)


    log_default = LogisticRegression()
    log_default.fit(X_train, y_train)

    ypred_train = log_default.predict(X_train)
    ypred_test = log_default.predict(X_test)

    Train.append(accuracy_score(y_train, ypred_train))
    Test.append(accuracy_score(y_test, ypred_test))

    CV.append(cross_val_score(log_default, X_train, y_train, cv=5, scoring="accuracy").mean())

em = pd.DataFrame({"Train": Train, "Test": Test, "CV": CV})
gm = em[(abs(em["Train"] - em["Test"]) <= 0.05) & (abs(em["Test"] - em["CV"]) <= 0.05)]
rs = gm[gm["CV"] == gm["CV"].max()].index.to_list()[0]

print("best random_state number:", rs)
```

```
best random_state number: 62
```

III.train_test_split

In [99]:
```python
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=62)
```

STEP-4 ML MODELLING

In [100…
```python
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
```

1.Logistic Regression Algorithm (train based)

In [101…

```
#Modelling
log_model = LogisticRegression()
log_model.fit(X_train,y_train)

#Evaluation
ypred_train = log_model.predict(X_train)
ypred_test = log_model.predict(X_test)


print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
print("cross validation score:",cross_val_score(log_model,X_train,y_train,cv=5,scoring="accuracy").mean())
```

```
Train Accuracy : 0.7991573033707865
Test Accuracy : 0.7932960893854749
cross validation score: 0.8019797104304146
```

2.KNN Classifier Algorithm (train based)

In [102…

```
#Hyper Parameter Tuning
estimator = KNeighborsClassifier()
param_grid = {"n_neighbors": list(range(1,50))}
knn_grid = GridSearchCV(estimator, param_grid,scoring="accuracy",cv=5)
knn_grid.fit(X_train,y_train)
knn_model = knn_grid.best_estimator_
knn_model
```

Out[102…

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

In [103…

```
#Modelling
knn_model = KNeighborsClassifier(n_neighbors=1)
knn_model.fit(X_train,y_train)

#Evaluation
ypred_train = knn_model.predict(X_train)
ypred_test = knn_model.predict(X_test)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
print("cross validation score :",cross_val_score(knn_model,X_train,y_train,cv=5,scoring="accuracy").mean())
```

```
Train Accuracy : 0.9831460674157303
Test Accuracy : 0.7374301675977654
cross validation score : 0.7486358711710824
```

3.Support Vector Machine Algorithm (train based)

In [104…

```
#Hyper Parameter Tuning
estimator = SVC()
param_grid = {"C":[0.01,0.1,1],"kernel":["linear","rbf","sigmoid","poly"]}
svm_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
svm_grid.fit(X_train,y_train)
svm_model = svm_grid.best_estimator_
svm_model
```

Out[104…

```
▾ SVC
SVC(C=0.1, kernel='linear')
```

In [105…

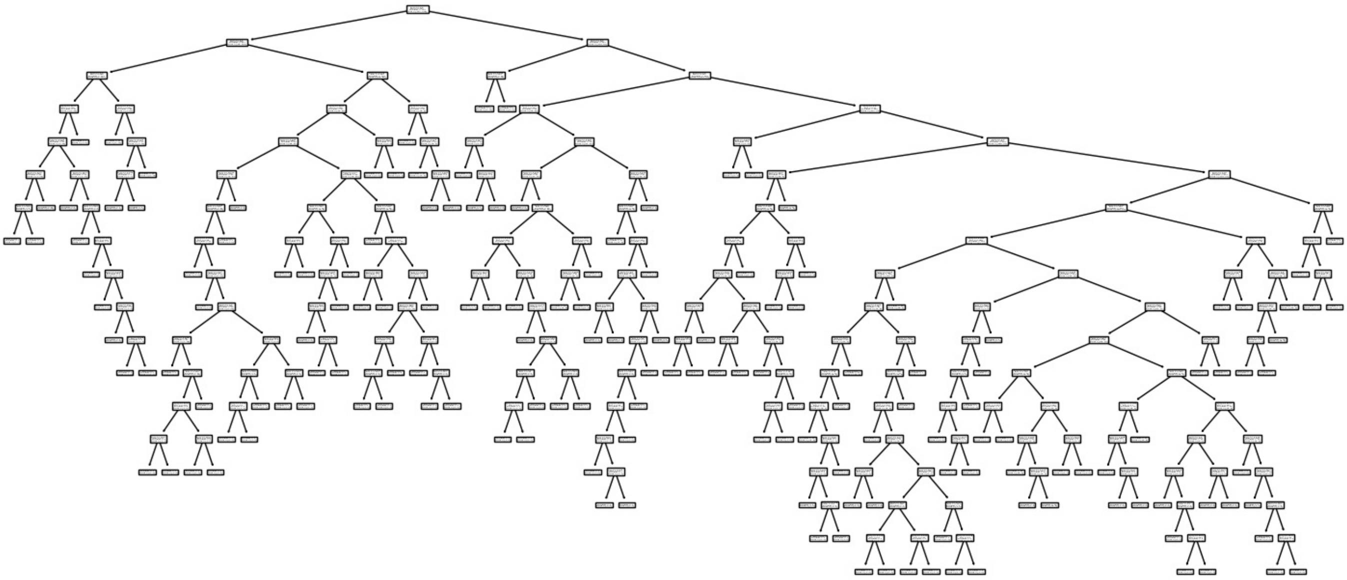
```
#Modelling
svm_model = SVC(C=0.1, kernel="linear")
svm_model.fit(X_train,y_train)

#Evaluation
ypred_train = svm_model.predict(X_train)
ypred_test = svm_model.predict(X_test)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
print("cross validation score :",cross_val_score(svm_model,X_train,y_train,cv=5,scoring="accuracy").mean())
```

```
Train Accuracy : 0.7893258426966292
Test Accuracy : 0.776536312849162
cross validation score : 0.7893135033980105
```

4.Decision Tree Classifier Algorithm (train based)

In [106…

```
model = DecisionTreeClassifier(random_state=True)
model.fit(X_train,y_train)
from sklearn.tree import plot_tree
plot_tree(model)
plt.show()
```



In [107…

```
#Hyper Parameter Tuning
estimator = DecisionTreeClassifier(random_state=True)
param_grid = {"criterion":["gini", "entropy"],
              "max_depth":list(range(1,16))}
dt_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
dt_grid.fit(X_train,y_train)
dt = dt_grid.best_estimator_
dt
```

Out[107…

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=True)
```

In [108…

```
#Important features
feats_dt = pd.DataFrame(data=dt.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_dt = feats_dt[feats_dt["Importance"]>0].index.tolist()
important_features_dt
```

Out[108…

```
['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
```

In [109…

```
#Selecting train & test data
X_train_dt = X_train[important_features_dt]


#Modelling
dt.fit(X_train_dt,y_train)

#Evaluation
ypred_train = dt.predict(X_train_dt)


print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
print("cross validation score :",cross_val_score(dt,X_train_dt,y_train,cv=5,scoring="accuracy").mean())
```

```
Train Accuracy : 0.8665730337078652
Test Accuracy : 0.776536312849162
cross validation score : 0.824386880724909
```

5.Random Forest Classifier Algorithm (train based)

In [110…

```
#Hyper Parameter Tuning
estimator = RandomForestClassifier(random_state=True)
```

```
param_grid = {"n_estimators":list(range(1,51))}
rf_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
rf_grid.fit(X_train,y_train)
rf = rf_grid.best_estimator_
rf
```

Out[110…

```
RandomForestClassifier
RandomForestClassifier(n_estimators=9, random_state=True)
```

In [111…

```
#Important features
feats_rf = pd.DataFrame(data=rf.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_rf = feats_rf[feats_rf["Importance"]>0].index.tolist()
important_features_rf
```

Out[111…

```
['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
```

In [112…

```
#Selecting train & test data
X_train_rf = X_train[important_features_rf]


#Modelling
rf.fit(X_train_rf,y_train)

#Evaluation
ypred_train = rf.predict(X_train_rf)


print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(rf,X_train_rf,y_train,cv=5,scoring="accuracy").mean())
```

```
Train Accuracy : 0.9719101123595506
cross validation score : 0.8118093174431202
```

6.Ada Boost Classifier Algorithm (train based)

In [113…

```
#Hyper Parameter Tuning
estimator = AdaBoostClassifier(random_state=True)
param_grid = {"n_estimators":list(range(1,51))}
ab_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
ab_grid.fit(X_train,y_train)
ab = ab_grid.best_estimator_
ab
```

Out[113…

```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=48, random_state=True)
```

In [114…

```
#Important features
feats_ab = pd.DataFrame(data=ab.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_ab = feats_ab[feats_ab["Importance"]>0].index.tolist()
important_features_ab
```

Out[114…

```
['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
```

In [115…

```
#Selecting train & test data
X_train_ab = X_train[important_features_ab]


#Modelling
ab.fit(X_train_ab,y_train)

#Evaluation
ypred_train = ab.predict(X_train_ab)


print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(ab,X_train_ab,y_train,cv=5,scoring="accuracy").mean())
```

```
Train Accuracy : 0.8384831460674157
cross validation score : 0.8202107751403528
```

7.Gradient Boosting Classifier Algorithm (train based)

In [116…

```
#Hyper Parameter Tuning
estimator = GradientBoostingClassifier(random_state=True)
```

```
param_grid = {"n_estimators":list(range(1,10)),
              "learning_rate":[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]}

gb_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
gb_grid.fit(X_train,y_train)

gb = gb_grid.best_estimator_
gb
```

Out[116…

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.7, n_estimators=7, random_state=True)
```

In [117…

```
#Important features
feats_gb = pd.DataFrame(data=gb.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_gb = feats_ab[feats_gb["Importance"]>0].index.tolist()
important_features_gb
```

Out[117…

```
['Pclass', 'Sex', 'Age', 'SibSp', 'Fare', 'Embarked']
```

In [118…

```
#Selecting train & test data
X_train_gb = X_train[important_features_gb]


#Modelling
gb.fit(X_train_gb,y_train)

#Evaluation
ypred_train = gb.predict(X_train_gb)


print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(gb,X_train_gb,y_train,cv=5,scoring="accuracy").mean())
```

```
Train Accuracy : 0.8693820224719101
cross validation score : 0.8229784300206836
```

8.XGBoost Classifier Algorithm (train based)

In [119…

```
#Hyper Parameter Tuning
estimator = XGBClassifier()
param_grid = {"n_estimators":[10,20,40,100],
              "max_depth":[3,4,5],
              "gamma":[0,0.15,0.3,0.5,1]}

xgb_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
xgb_grid.fit(X_train,y_train)

xgb = xgb_grid.best_estimator_
xgb
```

Out[119…

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=Non
e,
              enable_categorical=False, eval_metric=None, feature_types=Non
e,
              gamma=0.15, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=Non
e,
```

In [120…

```
#Important features
feats_xgb = pd.DataFrame(data=xgb.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_xgb = feats_gb[feats_xgb["Importance"]>0].index.tolist()
important_features_xgb
```

Out[120…

```
['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
```

In [121…

```
#Selecting train & test data
X_train_xgb = X_train[important_features_xgb]
```

```python
#Modelling
xgb.fit(X_train_xgb,y_train)

#Evaluation
ypred_train = xgb.predict(X_train_xgb)


print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(xgb,X_train_xgb,y_train,cv=5,scoring="accuracy").mean())
```

```
Train Accuracy : 0.8904494382022472
cross validation score : 0.8384319905446667
```

STEP-5 SAVE THE BEST MODEL (Train Based)

In [122…

```python
from joblib import dump

dump(log_model,"titanic_train.joblib")
```

Out[122…

```
['titanic_train.joblib']
```

STEP-6 PREDICT ON NEW DATA (Train Based)

In [123…

```python
df1_features = df1.drop(columns=["Survived"])  # Drop the target variable
train_output = model.predict(df1_features)     # Use the remaining features for prediction
```

In [124…

```python
train_output
```

Out[124…

```
array([0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

In [ ]:

X&y (Test Based)

In [125…

```python
X_= df2.drop("PassengerId",axis=1)
y = df2["PassengerId"]
```

1.Logistic Regression Algorithm (Test Based)

In [126…
```
#Modelling
log_model = LogisticRegression()
log_model.fit(X_test,y_test)

#Evaluation
ypred_test = log_model.predict(X_test)


print("Test Accuracy :",accuracy_score(y_test,ypred_test))
print("cross validation score:",cross_val_score(log_model,X_test,y_test,cv=5,scoring="accuracy").mean())
```

```
Test Accuracy : 0.7877094972067039
cross validation score: 0.7815873015873016
```

KNN Cllassifier Algorithm (Test Based)

In [127…
```
#Hyper Parameter Tuning
estimator = KNeighborsClassifier()
param_grid = {"n_neighbors": list(range(1,50))}
knn_grid = GridSearchCV(estimator, param_grid,scoring="accuracy",cv=5)
knn_grid.fit(X_test,y_test)
knn_model = knn_grid.best_estimator_
knn_model
```

Out[127…
```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [128…
```
#Modelling
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_test,y_test)

#Evaluation

ypred_test  = knn_model.predict(X_test)


print("cross validation score :",cross_val_score(knn_model,X_test,y_test,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
cross validation score : 0.7095238095238094
Test Accuracy : 0.8659217877094972
```

3.Support Vector Machine Algorithm (Test Based)

In [129…
```
#Hyper Parameter Tuning
estimator = SVC()
param_grid = {"C":[0.01,0.1,1],"kernel":["linear","rbf","sigmoid","poly"]}
svm_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
svm_grid.fit(X_test,y_test)
svm_model = svm_grid.best_estimator_
svm_model
```

Out[129…
```
▾        SVC
SVC(C=1, kernel='linear')
```

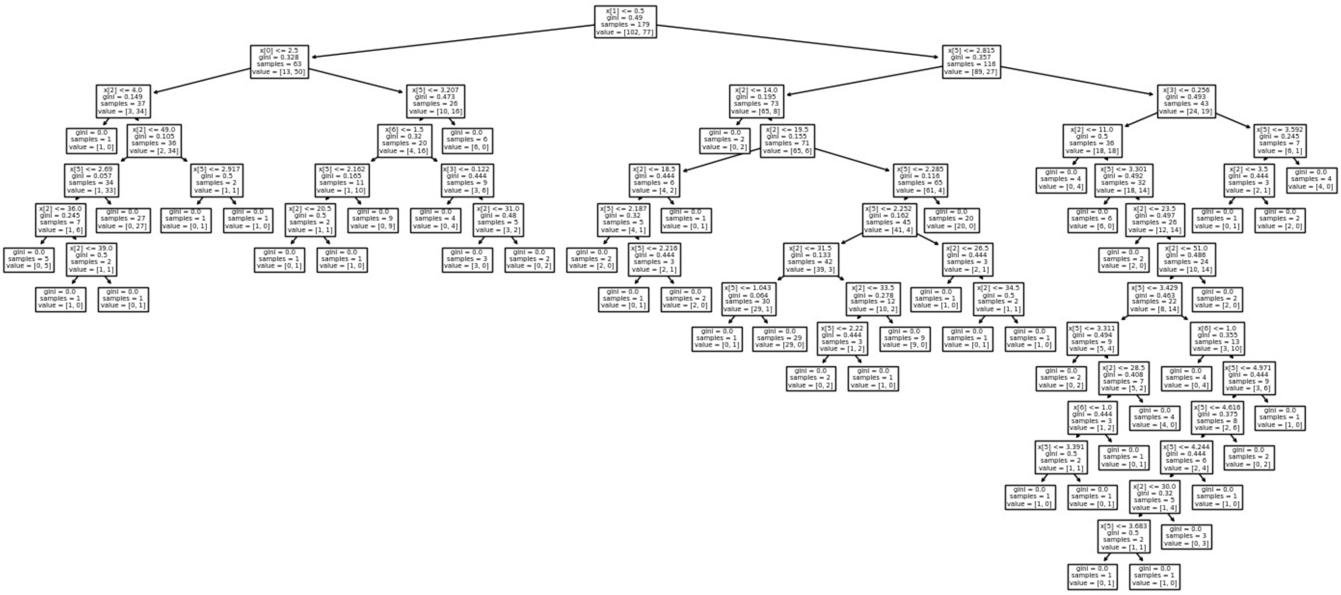
In [130…
```
#Modelling
svm_model = SVC(C=1, kernel="linear")
svm_model.fit(X_test,y_test)

#Evaluation

ypred_test  = svm_model.predict(X_test)


print("cross validation score :",cross_val_score(svm_model,X_test,y_test,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
cross validation score : 0.7760317460317461
Test Accuracy : 0.776536312849162
```

4.Decision Tree classifier Algorithm

In [131…
```
model = DecisionTreeClassifier(random_state=True)
model.fit(X_test,y_test)
from sklearn.tree import plot_tree
```

```
plot_tree(model)
plt.show()
```



In [132…

```
#Hyper Parameter Tuning
estimator = DecisionTreeClassifier(random_state=True)
param_grid = {"criterion":["gini", "entropy"],
              "max_depth":list(range(1,16))}
dt_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
dt_grid.fit(X_test,y_test)
dt = dt_grid.best_estimator_
dt
```

Out[132…

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=True)
```

In [133…

```
#Important features
feats_dt = pd.DataFrame(data=dt.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_dt = feats_dt[feats_dt["Importance"]>0].index.tolist()
important_features_dt
```

Out[133…

```
['Pclass', 'Sex', 'Age', 'SibSp', 'Fare', 'Embarked']
```

In [134…

```
#Selecting train & test data

X_test_dt = X_test[important_features_dt]

#Modelling
dt.fit(X_test_dt,y_test)

#Evaluation

ypred_test  = dt.predict(X_test_dt)


print("cross validation score :",cross_val_score(dt,X_test_dt,y_test,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
cross validation score : 0.7484126984126984
Test Accuracy : 0.9106145251396648
```

5.Random Forest Classifier Algorithm (Test Based)

In [135…

```
#Hyper Parameter Tuning
estimator = RandomForestClassifier(random_state=True)
param_grid = {"n_estimators":list(range(1,51))}
rf_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
rf_grid.fit(X_test,y_test)
rf = rf_grid.best_estimator_
rf
```

Out[135…

```
RandomForestClassifier
RandomForestClassifier(n_estimators=47, random_state=True)
```

In [136…
```
#Important features
feats_rf = pd.DataFrame(data=rf.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_rf = feats_rf[feats_rf["Importance"]>0].index.tolist()
important_features_rf
```

Out[136…
```
['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
```

In [137…
```
#Selecting train & test data

X_test_rf = X_test[important_features_rf]

#Modelling
rf.fit(X_test_rf,y_test)

#Evaluation

ypred_test  = rf.predict(X_test_rf)


print("cross validation score :",cross_val_score(rf,X_test_rf,y_test,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
cross validation score : 0.7873015873015874
Test Accuracy : 0.9888268156424581
```

6.Ada Boost Classifier Algorithm (Test Based)

In [138…
```
#Hyper Parameter Tuning
estimator = AdaBoostClassifier(random_state=True)
param_grid = {"n_estimators":list(range(1,51))}
ab_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
ab_grid.fit(X_test,y_test)
ab = ab_grid.best_estimator_
ab
```

Out[138…
```
▾ AdaBoostClassifier
AdaBoostClassifier(n_estimators=35, random_state=True)
```

In [139…
```
#Important features
feats_ab = pd.DataFrame(data=ab.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_ab = feats_ab[feats_ab["Importance"]>0].index.tolist()
important_features_ab
```

Out[139…
```
['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']
```

In [140…
```
#Selecting train & test data

X_test_ab = X_test[important_features_ab]

#Modelling
ab.fit(X_test_ab,y_test)

#Evaluation

ypred_test  = ab.predict(X_test_ab)


print("cross validation score :",cross_val_score(ab,X_test_ab,y_test,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
cross validation score : 0.8099999999999999
Test Accuracy : 0.8994413407821229
```

7.Gradient Boost Classifier Algorithm (Test Based)

In [141…
```
#Hyper Parameter Tuning
estimator = GradientBoostingClassifier(random_state=True)
param_grid = {"n_estimators":list(range(1,10)),
              "learning_rate":[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]}

gb_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
gb_grid.fit(X_test,y_test)

gb = gb_grid.best_estimator_
gb
```

Out[141…

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=1.0, n_estimators=4, random_state=True)
```

In [142…

```
#Important features
feats_gb = pd.DataFrame(data=gb.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_gb = feats_gb[feats_gb["Importance"]>0].index.tolist()
important_features_gb
```

Out[142…

```
['Pclass', 'Sex', 'Age', 'SibSp', 'Fare']
```

In [143…

```
#Selecting train & test data

X_test_gb = X_test[important_features_gb]

#Modelling
gb.fit(X_test_gb,y_test)

#Evaluation

ypred_test  = gb.predict(X_test_gb)


print("cross validation score :",cross_val_score(gb,X_test_gb,y_test,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
cross validation score : 0.7987301587301587
Test Accuracy : 0.9106145251396648
```

8.XGBoost Classifier Algorithm (Test Based)

In [144…

```
#Hyper Parameter Tuning
estimator = XGBClassifier()
param_grid = {"n_estimators":[10,20,40,100],
              "max_depth":[3,4,5],
              "gamma":[0,0.15,0.3,0.5,1]}

xgb_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
xgb_grid.fit(X_test,y_test)

xgb = xgb_grid.best_estimator_
xgb
```

Out[144…

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=Non
e,
              enable_categorical=False, eval_metric=None, feature_types=Non
e,
              gamma=0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=Non
e,
```

In [145…

```
#Important features
feats_xgb = pd.DataFrame(data=xgb.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_xgb = feats_gb[feats_xgb["Importance"]>0].index.tolist()
important_features_xgb
```

Out[145…

```
['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
```

In [146…

```
#Selecting train & test data

X_test_xgb = X_test[important_features_xgb]

#Modelling
xgb.fit(X_test_xgb,y_test)

#Evaluation

ypred_test  = xgb.predict(X_test_xgb)
```

```python
print("cross validation score :",cross_val_score(xgb,X_test_xgb,y_test,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
cross validation score : 0.8096825396825398
Test Accuracy : 0.88268156424581
```

5.SAVE THE BEST MODEL (Test Based)

In [147…

```python
from joblib import dump

dump(xgb,"titanic_test.joblib")
```

Out[147…

```
['titanic_test.joblib']
```

6.PREDICT ON NEW DATA (Test Based)

In [148…

```python
df2_features = df2.drop(columns=["PassengerId"])  # Drop the target variable
titanic_test_output = model.predict(df2_features)
titanic_test_output
```

Out[148…

```
array([0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0],
      dtype=int64)
```

In [149…

```python
import pandas as pd

# Assuming train_output is your prediction array and df1 has the corresponding PassengerId
titanic_test_output_df = pd.DataFrame({
    'PassengerId': df2['PassengerId'],  # Replace 'PassengerId' with the appropriate identifier column if diffe
    'Survived': titanic_test_output
})

# Save the DataFrame to a CSV file
titanic_test_output_df.to_csv('test_output.csv', index=False)
titanic_test_output_df
```

Out[149…

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	0
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

In [ ]: