0-31-function-power-transformation

May 26, 2025

#Note * Ato din amra jei transformation(SimpleImputer,OHE,etc) porci oi golo bahire kico Algorithm ase je Normally Distributed Data te valo perform kore like(LR,LL,PCA,Naive Bayes and more) * to convert irragular data to Normal Distribution Need “Function or Power” Transformation(log,box-cox,etc)

#Day-30:Fucntion Transformation

#   numeric column a Used hoi

How to Understand data is normal or not?

- sns.distplot
- pd.skew()==0
- QQ Plot

###QQ plot:

A Q-Q plot (Quantile-Quantile plot) is a statistical tool used to check whether a dataset follows a particular distribution, most commonly a normal distribution.

What Is a Q-Q Plot?

Plots the quantiles of your data against the quantiles of a theoretical distribution (like normal).

If your data is normally distributed, the points will fall approximately along a straight 45-degree line.

How to Interpret a Q-Q Plot:

Straight diagonal line → Data is normally distributed.

S-shaped curve → Data has heavy tails or light tails.

Curved away from the line → Data is skewed.

[ ]:

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

```
[ ]: df=pd.read_csv('/content/Titanic-Dataset.csv')
     df.head(2)
```

```
[ ]:    PassengerId  Survived  Pclass  \
     0            1         0       3
     1            2         1       1

                                                  Name     Sex   Age  SibSp  \
     0                            Braund, Mr. Owen Harris    male  22.0      1
     1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1

        Parch     Ticket     Fare Cabin Embarked
     0      0  A/5 21171   7.2500   NaN        S
     1      0   PC 17599  71.2833   C85        C
```

```
[ ]: df=df.iloc[:,[1,2,4,5,6,7,9,11]]
```

```
[ ]: df.head(3)
```

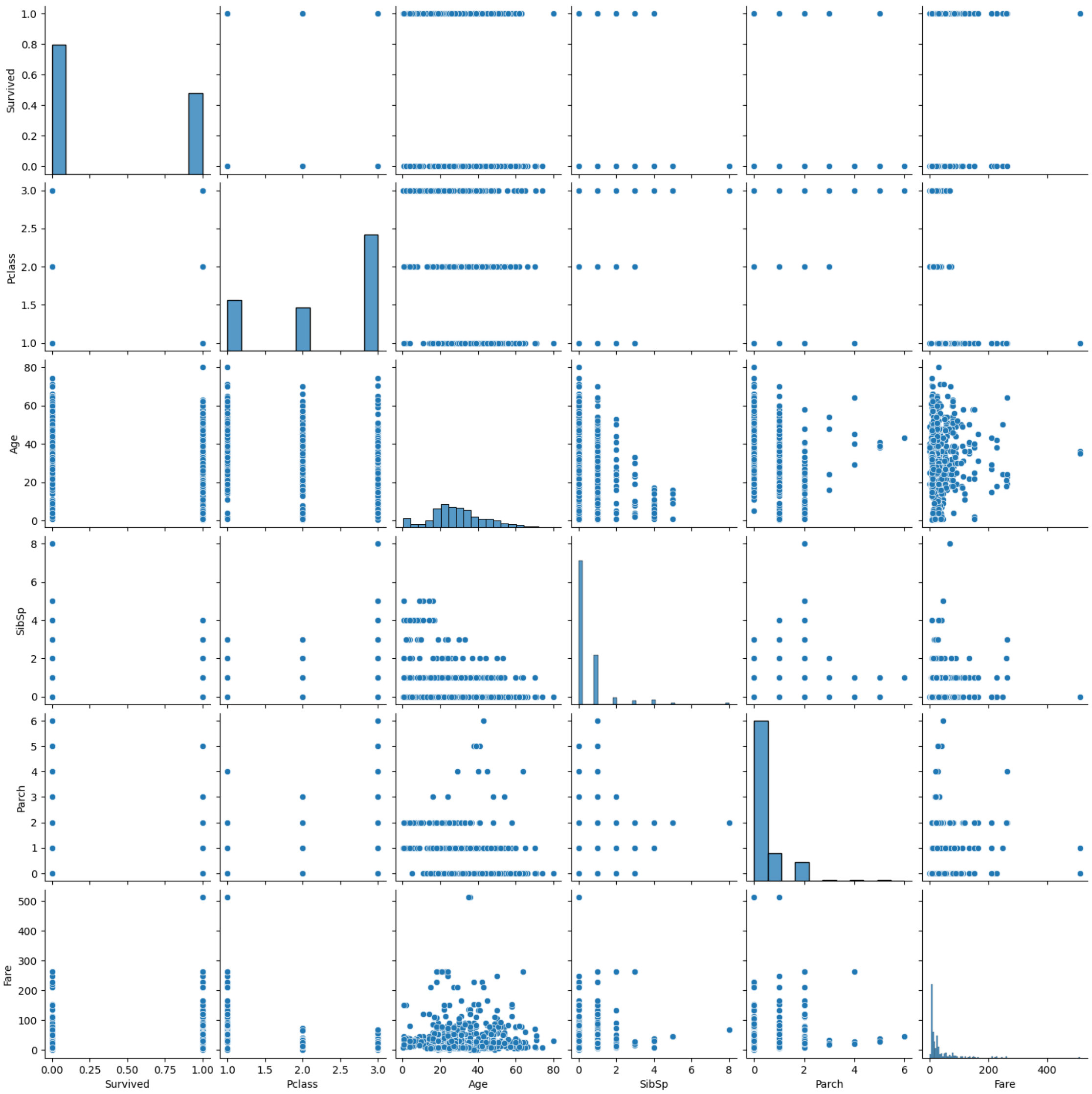
```
[ ]:    Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
     0         0       3    male  22.0      1      0   7.2500        S
     1         1       1  female  38.0      1      0  71.2833        C
     2         1       3  female  26.0      0      0   7.9250        S
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    object
 3   Age       714 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  889 non-null    object
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

```
[ ]: sns.pairplot(df)
```

```
[ ]: <seaborn.axisgrid.PairGrid at 0x7852f0b94150>
```

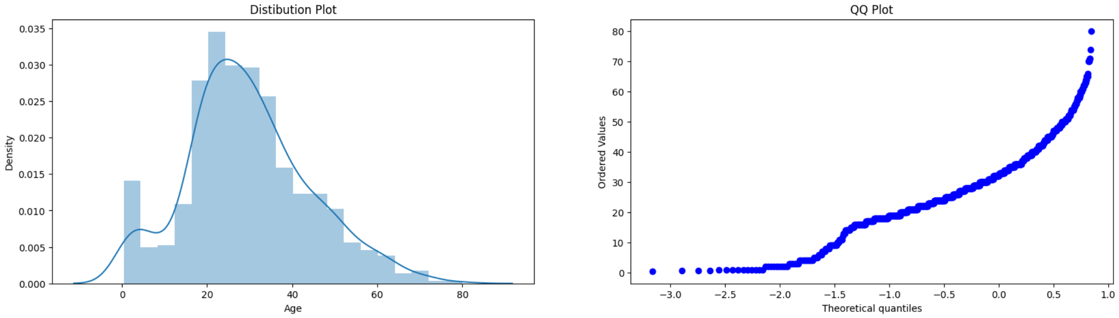


```
[ ]: # df['Age'].fillna(df['Age'].mean(),inplace=True)
     # df.isnull().sum()
```

```
[ ]: import scipy.stats as stats

     plt.figure(figsize=(20,5))
     plt.subplot(121)
     sns.distplot(df['Age'])
     plt.title('Distibution Plot')

     plt.subplot(122)
     stats.probplot(df['Age'],dist='norm',plot=plt)
     plt.title('QQ Plot')
```

```
<ipython-input-237-ca1b29434fca>:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Age'])
```

```
[ ]: Text(0.5, 1.0, 'QQ Plot')
```



```
[ ]: from sklearn.compose import ColumnTransformer
     from sklearn.preprocessing import FunctionTransformer
```

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

```
[ ]:    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
```

```
[ ]: from sklearn.preprocessing import MinMaxScaler
     Ct=ColumnTransformer([
         ('ft',MinMaxScaler(),[3])
     ],remainder='passthrough')
```

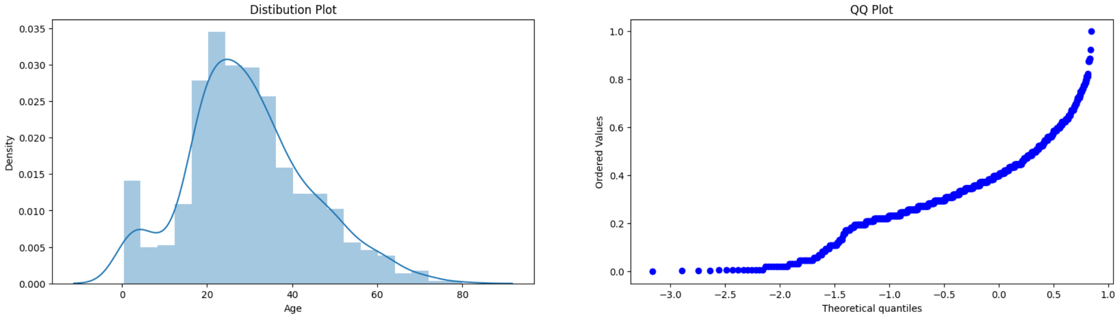
```
[ ]: # Ct=ColumnTransformer([
     #     ('ft',FunctionTransformer(np.log1p),['Age'])
     # ],remainder='passthrough')
```

```
[ ]: fited_df=Ct.fit_transform(df)
```

```
[ ]: np.array(fited_df[:,0],dtype=np.float64)#convert object array to numpy array
[ ]: array([0.27117366, 0.4722292 , 0.32143755, 0.43453129, 0.43453129,
                   nan, 0.67328474, 0.01985423, 0.33400352, 0.17064589,
            0.04498618, 0.72354863, 0.24604172, 0.48479517, 0.17064589,
            0.68585072, 0.01985423,        nan, 0.3842674 ,        nan,
            0.43453129, 0.42196532, 0.18321186, 0.34656949, 0.09525006,
            0.4722292 ,        nan, 0.23347575,        nan,        nan,
            0.49736115,        nan,        nan, 0.8240764 , 0.34656949,
            0.52249309,        nan, 0.25860769, 0.22090978, 0.17064589,
            0.49736115, 0.33400352,        nan, 0.03242021, 0.23347575,
                   nan,        nan,        nan,        nan, 0.22090978,
            0.08268409, 0.25860769, 0.61045489, 0.35913546, 0.81151043,
                   nan, 0.25860769, 0.35285248, 0.05755215, 0.13294798,
            0.27117366, 0.4722292 , 0.560191  , 0.04498618,        nan,
                   nan, 0.35913546, 0.23347575, 0.2083438 , 0.32143755,
            0.39683338, 0.19577783, 0.25860769, 0.32143755, 0.39683338,
            0.30887158,        nan,        nan, 0.00515205, 0.37170143,
            0.27117366, 0.35913546,        nan, 0.34656949, 0.2083438 ,
            0.40939935, 0.19577783,        nan, 0.28373963, 0.2963056 ,
            0.35913546, 0.24604172, 0.57275697, 0.32143755, 0.7361146 ,
                   nan, 0.88690626, 0.28373963, 0.42196532, 0.42196532,
            0.34656949,        nan, 0.25860769, 0.40939935, 0.45966323,
            0.34656949, 0.25860769,        nan, 0.4722292 ,        nan,
            0.58532295, 0.17692888, 0.27117366, 0.24604172, 0.2083438 ,
            0.25860769, 0.88062327, 0.35913546, 0.2963056 , 0.01985423,
            0.25860769,        nan, 0.40311636, 0.40311636, 0.67328474,
            0.14551395,        nan, 0.2963056 ,        nan, 0.560191  ,
            0.40939935, 0.24604172, 0.58532295, 0.35913546, 0.30887158,
            0.28373963, 0.23347575, 0.45966323, 0.19577783, 0.2963056 ,
                   nan, 0.27117366, 0.2963056 , 0.23347575, 0.22090978,
            0.23347575, 0.33400352, 0.10781603, 0.45338025, 0.52249309,
            0.63558683, 0.27117366, 0.6921337 , 0.50364413,        nan,
            0.63558683, 0.19577783, 0.37170143,        nan,        nan,
            0.54762503, 0.49736115, 0.32143755, 0.2083438 , 0.00728826,
            0.10781603,        nan, 0.560191  ,        nan, 0.34656949,
            0.76124654, 0.04498618, 0.00728826, 0.25860769, 0.69841669,
            0.22090978,        nan, 0.62302086, 0.37170143, 0.44709726,
                   nan,        nan, 0.10781603, 0.00728826, 0.04498618,
                   nan,        nan, 0.560191  , 0.49736115, 0.44709726,
            0.39683338, 0.23347575, 0.23347575, 0.03242021, 0.54762503,
            0.72354863,        nan, 0.52249309,        nan, 0.2963056 ,
            0.34656949,        nan, 0.42196532, 0.56647399, 0.22090978,
            0.01985423, 0.39683338, 0.32143755, 0.19577783, 0.49736115,
            0.2963056 , 0.43453129, 0.27117366, 0.37170143,        nan,
            0.3842674 , 0.33400352, 0.52249309, 0.39683338, 0.37170143,
            0.19577783, 0.33400352, 0.63558683,        nan, 0.4722292 ,
            0.27117366, 0.23347575, 0.2523247 , 0.22090978,        nan,
            0.43453129, 0.35913546, 0.7361146 , 0.05755215, 0.2963056 ,
       nan, 0.54762503, 0.09525006, 0.23347575, 0.40939935,
       nan,        nan, 0.35913546, 0.27117366, 0.37170143,
0.54762503, 0.30887158, 0.2963056 , 0.45966323, 0.67328474,
       nan, 0.35913546, 0.77381252, 0.37170143, 0.50992712,
0.35913546,        nan, 0.37170143, 0.43453129, 0.62302086,
       nan, 0.03242021, 0.6481528 , 0.49736115,        nan,
0.44709726, 0.19577783, 0.30887158, 0.72354863, 0.43453129,
       nan, 0.30887158, 0.50992712, 0.45966323,        nan,
0.78637849, 0.560191  ,        nan, 0.08268409, 0.43453129,
0.81151043, 0.34656949, 0.19577783, 0.23347575,        nan,
0.40939935, 0.37170143, 0.27117366, 0.52249309, 0.27117366,
0.32143755, 0.23347575, 0.44709726, 0.2963056 , 0.2963056 ,
       nan, 0.29002262, 0.01985423,        nan, 0.62302086,
       nan,        nan, 0.23347575,        nan,        nan,
0.00628299,        nan, 0.2083438 , 0.37170143, 0.37170143,
0.2963056 , 0.22090978, 0.32143755, 0.34656949, 0.53505906,
0.32143755, 0.2963056 , 0.67328474, 0.3842674 , 0.49736115,
0.27117366, 0.33400352, 0.37170143, 0.27117366,        nan,
0.44709726, 0.76124654, 0.44709726, 0.3842674 , 0.19577783,
       nan, 0.56647399, 0.4722292 , 0.19577783,        nan,
       nan, 0.35913546, 0.50992712, 0.560191  , 0.560191  ,
0.01985423, 0.2963056 , 0.34656949, 0.30887158, 0.44709726,
0.2963056 , 0.49736115,        nan, 0.03242021, 0.52249309,
0.28373963,        nan, 0.18321186, 0.30887158,        nan,
0.34656949, 0.27117366, 0.4722292 ,        nan,        nan,
0.49736115, 0.35913546, 0.560191  , 0.43453129,        nan,
0.37170143, 0.74868057,        nan,        nan, 0.2963056 ,
0.30887158, 0.22090978, 0.23347575, 0.27117366, 0.03242021,
       nan, 0.27117366, 0.33400352, 0.24604172, 0.23347575,
0.52249309, 0.00728826, 0.39683338, 0.43453129,        nan,
0.22090978, 0.00728826, 0.44709726,        nan, 0.2083438 ,
0.44709726, 0.25860769, 0.34656949, 0.28373963, 0.2963056 ,
0.27117366, 0.3842674 , 0.57275697, 0.28373963, 0.34656949,
0.48479517, 0.32143755, 0.25860769, 0.34656949, 0.24604172,
0.42196532, 0.63558683, 0.03242021, 0.25860769,        nan,
       nan,        nan, 0.40939935,        nan, 0.54762503,
       nan, 0.42196532, 0.22090978, 0.37170143, 0.12038201,
       nan, 0.25860769, 0.35913546, 0.34656949, 0.22090978,
       nan, 0.34656949, 0.23347575,        nan, 0.39683338,
0.34656949,        nan, 0.52249309, 0.2083438 , 0.62302086,
0.17064589, 0.25860769, 0.2963056 , 0.79894446, 0.3842674 ,
0.560191  , 0.24604172, 0.30887158, 0.34656949,        nan,
0.04498618, 0.15807992, 0.42196532, 0.05755215, 0.6481528 ,
0.44709726,        nan, 0.37170143, 0.61045489,        nan,
0.35913546, 0.81151043,        nan, 0.62302086,        nan,
0.59788892, 0.42196532, 0.58532295, 0.59788892,        nan,
0.4722292 ,        nan, 0.69841669,        nan, 0.00414677,
       nan, 0.4722292 , 0.40939935, 0.28373963, 0.27117366,
       nan, 0.42196532, 0.35913546, 0.27117366, 0.01985423,
0.10781603,        nan, 0.62302086, 0.78637849, 0.30887158,
       nan, 0.43453129, 0.72354863, 0.37170143, 0.10781603,
       nan, 0.25860769, 0.68585072, 0.88690626, 0.25860769,
       nan, 0.67328474,        nan, 0.30887158, 0.2963056 ,
0.2083438 , 0.25860769,        nan, 0.45966323, 0.19577783,
0.22090978, 0.40939935,        nan, 0.34656949, 0.32143755,
0.35913546,        nan, 0.44709726, 0.67328474, 0.2963056 ,
0.58532295, 0.42196532,        nan, 0.44709726, 0.39683338,
0.37170143, 0.27117366,        nan, 0.54762503,        nan,
0.50364413, 0.62302086,        nan, 0.48479517, 0.28373963,
0.01985423,        nan, 0.2083438 ,        nan, 0.37170143,
0.08268409, 0.560191  , 0.37170143,        nan, 0.27117366,
0.44709726, 0.10781603, 0.13294798, 0.39683338, 0.62302086,
0.79894446, 0.23347575,        nan, 0.40939935, 0.09525006,
0.2083438 , 0.33400352,        nan, 0.27117366, 0.27117366,
0.77381252, 0.59788892,        nan, 0.48479517, 0.44709726,
       nan, 0.49736115, 0.34656949,        nan,        nan,
0.2963056 , 0.23347575, 0.35913546,        nan, 0.39683338,
0.77381252, 0.66071877, 0.44709726,        nan, 0.19577783,
0.23347575, 0.42196532, 0.48479517,        nan, 0.39683338,
0.30887158, 0.48479517, 0.67328474, 0.44709726,        nan,
0.22090978, 0.58532295, 0.74868057, 0.27117366,        nan,
0.43453129, 0.6481528 , 0.58532295,        nan, 0.45966323,
0.44709726,        nan, 0.61045489,        nan, 0.61045489,
0.2963056 ,        nan,        nan, 0.54762503, 0.43453129,
0.44709726, 0.37170143, 0.33400352, 0.27117366, 0.49736115,
0.48479517,        nan,        nan,        nan, 0.43453129,
0.2963056 , 0.42196532, 0.32143755, 0.04498618, 0.32143755,
0.33400352, 0.52249309, 0.24604172, 0.25860769, 0.25860769,
0.76124654, 0.71098266, 0.25860769, 0.32143755,        nan,
1.        , 0.63558683, 0.39683338,        nan, 0.10781603,
0.34656949, 0.39683338, 0.3842674 , 0.50992712,        nan,
0.24604172, 0.2963056 , 0.01985423,        nan, 0.00414677,
0.59788892, 0.23347575, 0.69841669,        nan, 0.28373963,
       nan, 0.22090978, 0.25860769,        nan, 0.22090978,
0.2963056 ,        nan, 0.39683338, 0.28373963, 0.72354863,
0.62302086, 0.49736115, 0.58532295, 0.44709726, 0.24604172,
0.39683338, 0.30887158,        nan, 0.53505906,        nan,
0.49736115, 0.3842674 , 0.87434029, 0.3842674 ,        nan,
0.22090978, 0.30258859, 0.22090978, 0.53505906, 0.44709726,
       nan, 0.33400352, 0.24604172, 0.17064589, 0.74868057,
0.30887158, 0.17064589, 0.23347575, 0.22090978, 0.18321186,
0.3842674 , 0.04498618,        nan, 0.30887158, 0.74868057,
0.6481528 , 0.54762503,        nan, 0.61045489, 0.52249309,
0.22090978, 0.43453129, 0.22090978, 0.30887158, 0.32143755,
0.48479517, 0.560191  , 0.52249309, 0.27117366,        nan,
0.2963056 ,        nan, 0.59788892, 0.35913546, 0.6481528 ,
0.23347575, 0.4722292 , 0.33400352,        nan, 0.40939935,
0.07011812, 0.2083438 , 0.42196532, 0.62302086, 0.33400352,
0.24604172, 0.37170143,        nan, 0.30887158, 0.30887158,
0.35913546, 0.13294798,        nan, 0.28373963, 0.28373963,
0.35285248, 0.59788892, 0.43453129,        nan,        nan,
       nan, 0.44709726, 0.25860769, 0.2963056 , 0.3842674 ,
0.87434029, 0.19577783, 0.37170143, 0.23347575, 0.3842674 ,
0.04498618, 0.07011812, 0.40939935, 0.28373963, 0.59788892,
0.00314149, 0.34656949, 0.22090978, 0.42196532, 0.40939935,
       nan, 0.50992712, 0.24604172, 0.44709726, 0.19577783,
0.63558683,        nan, 0.37798442,        nan, 0.39683338,
0.2963056 , 0.59788892, 0.71098266,        nan, 0.67328474,
0.22090978,        nan, 0.05755215,        nan, 0.53505906,
0.15807992, 0.2083438 , 0.35913546,        nan, 0.30887158,
0.30887158, 0.22090978, 0.09525006, 0.00728826, 0.57275697,
       nan, 0.19577783,        nan,        nan, 0.30887158,
0.48479517, 0.61045489, 0.3842674 , 0.37170143, 0.37170143,
0.42196532, 0.3842674 , 0.13294798, 0.        , 0.33400352,
0.3842674 , 0.48479517, 0.22090978, 0.48479517, 0.40939935,
0.32143755, 0.48479517, 0.43453129, 0.07011812, 0.37798442,
       nan, 0.28373963, 0.3842674 , 0.53505906, 0.12038201,
0.6481528 , 0.33400352, 0.4722292 , 0.33400352, 0.01985423,
       nan,        nan, 0.00728826,        nan, 0.77381252,
0.18321186, 0.00515205,        nan, 0.28373963, 0.22090978,
0.48479517, 0.25860769,        nan, 0.39683338,        nan,
0.24604172, 0.19577783, 0.37170143, 0.4282483 , 0.2083438 ,
0.52249309,        nan, 0.43453129, 0.34656949,        nan,
0.04498618, 0.92460417, 0.10781603, 0.19577783, 0.54762503,
0.22090978, 0.560191  , 0.63558683, 0.2963056 ,        nan,
0.50992712, 0.25860769, 0.59788892,        nan, 0.2963056 ,
0.52249309, 0.33400352, 0.3842674 ,        nan, 0.04498618,
0.32143755, 0.58532295, 0.40939935, 0.58532295, 0.34656949,
0.18321186, 0.24604172, 0.23347575,        nan, 0.69841669,
0.30887158, 0.40939935, 0.27117366, 0.34656949, 0.30887158,
0.48479517, 0.33400352, 0.23347575,        nan, 0.32143755,
0.39683338])
```

```
[ ]: plt.figure(figsize=(20,5))
     plt.subplot(121)
     sns.distplot(df['Age'])
     plt.title('Distibution Plot')

     plt.subplot(122)
     stats.probplot(np.array(fited_df[:,0],dtype=np.float64),dist='norm',plot=plt)
     plt.title('QQ Plot')
plt.show()
```

```
<ipython-input-244-3dfa41174bf4>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Age'])
```



```
[ ]:
```

#Target: * Accuracy of LR,DT * using data alalysis, which data should which taransformation to perform batter.

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

```
[ ]:    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
```

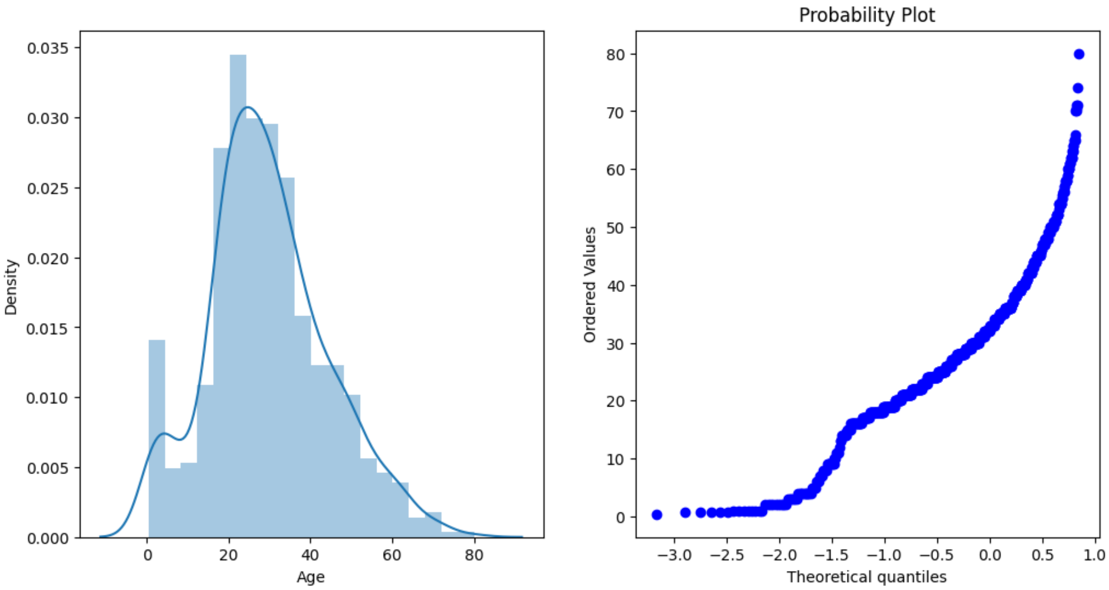
```
[ ]: from sklearn.preprocessing import FunctionTransformer
     from sklearn.compose import ColumnTransformer,make_column_transformer
     from sklearn.linear_model import LogisticRegression
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.model_selection import train_test_split
     from sklearn.impute import SimpleImputer
     from sklearn.preprocessing import StandardScaler,MinMaxScaler
     from sklearn.pipeline import Pipeline,make_pipeline
     from sklearn.preprocessing import OneHotEncoder
```

#Analysis: * Numarical Data should analyze for detection of Which Transformation should use

Using Graph(distplot,QQ plot,etc)

```
[ ]: #we have tow Numarical column (Age and Fare)
     plt.figure(figsize=(12,6))
     plt.subplot(121)
     sns.distplot(df['Age'])
     plt.subplot(122)
     stats.probplot(df['Age'],dist='norm',plot=plt)
     plt.show()
     #line does not show case of having missing value
```

```
<ipython-input-247-c197af16950e>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Age'])
```

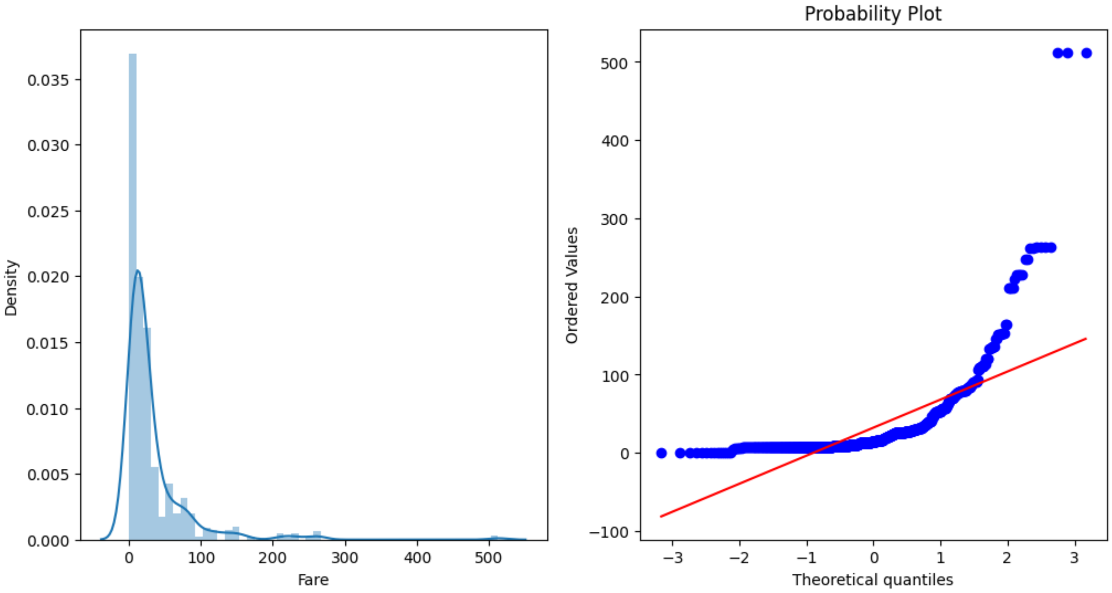


```
[ ]: df['Age'].skew()
     #we see data little bit skewed,if i want then i can be Function Transformation
```

```
[ ]: np.float64(0.38910778230082704)
```

[ ]:
```
#we have tow Numarical column (Age and Fare)
plt.figure(figsize=(12,6))
plt.subplot(121)
sns.distplot(df['Fare'])
plt.subplot(122)
stats.probplot(df['Fare'],dist='norm',plot=plt)
plt.show()
```

```
<ipython-input-249-a3f41a2ad77d>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Fare'])
```



[ ]:
```
df['Fare'].skew()
#hare we see data is left skewed , That is way if we use LR,LL,SVM,KNN ,we need␣
 ↪Functional Transform
```

```
[ ]: np.float64(4.787316519674893)
```

[ ]:
```

```

#Compleate Packege * Train Test Split * Missing value Handeling * Analyze Data about Skewness of is applicable for Function Transformation * Encoding Catagorical Data * Scaling data if need * Feature Selection * Model Train * Go though Pipeline * Find accuracy before and After

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

```
[ ]:    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
```

```
[ ]: df.isnull().sum()
     #two column contain missing value (Age,Embarked)
```

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

```
[ ]: X_train,X_test,y_train,y_test=train_test_split(df.drop('Survived',axis=1),df['Survived'],test_size=0.2,random_state=42)
     X_train.head(2)
```

```
[ ]:      Pclass   Sex   Age  SibSp  Parch  Fare Embarked
     331       1  male  45.5      0      0  28.5        S
     733       2  male  23.0      0      0  13.0        S
```

```
[ ]:
```

```
[ ]: #Age--->missing value handeling
     #Age+Fare--->log Transform


     #for Numerical(int+float)
     age_fare_pipe=Pipeline([
         ('age_imp',SimpleImputer()),
         ('age_log_tf',FunctionTransformer(np.log1p))
     ])


     #aikhane Age ar jonne alada pipeline and Fare ar jonne alada pipeline kora jeto kinto oi ta valo poddoti hoto na
```

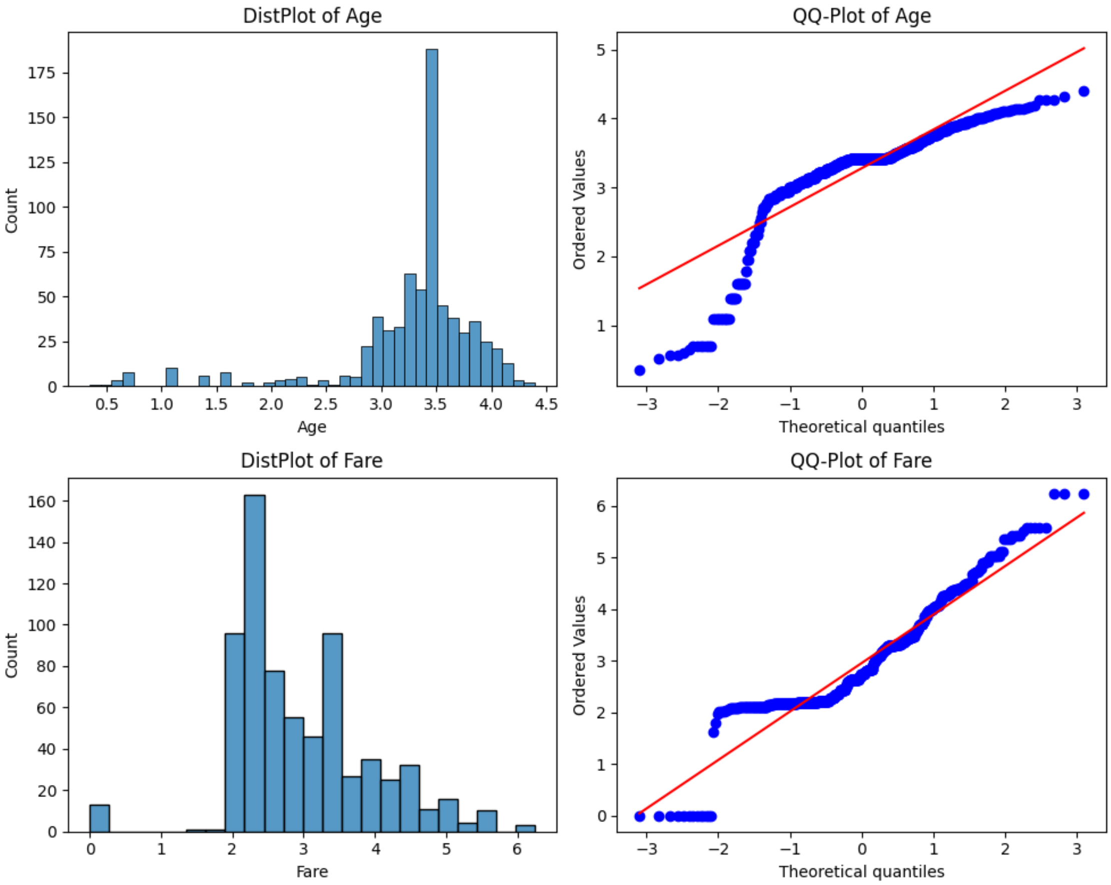
```
#ai khane age ar opr missing value handeling kaj korbe just and Age and Fare ar␣
 ↪opor log Transformation kaj korbe.
```

```
[ ]: temp_ct=ColumnTransformer([
         ('abc',age_fare_pipe,[2,5])
     ])
     age_fare_tf=temp_ct.fit_transform(X_train)
     age_fare_tf
```

```
[ ]: array([[3.83945231, 3.38439026],
            [3.17805383, 2.63905733],
            [3.49650756, 2.18885633],
            …,
            [3.73766962, 2.71524426],
            [2.7080502 , 4.79579055],
            [3.09104245, 4.36038795]])
```

```
[ ]: #analyze ,is function transformation need?
     plt.figure(figsize=(10,8))

     plt.subplot(221)
     sns.histplot(age_fare_tf[:,0])
     plt.xlabel('Age')
     plt.title("DistPlot of Age")

     plt.subplot(222)
     stats.probplot(age_fare_tf[:,0].astype(np.float64),plot=plt)
     # plt.xlabel('Age')
     plt.title("QQ-Plot of Age")

     plt.subplot(223)
     sns.histplot(age_fare_tf[:,1])
     plt.xlabel('Fare')
     plt.title("DistPlot of Fare")

     plt.subplot(224)
     stats.probplot(age_fare_tf[:,1].astype(np.float64),plot=plt)
     # plt.xlabel('Age')
     plt.title("QQ-Plot of Fare")


     plt.tight_layout()

     #Age ar Distribution ager thake onk khrap hoise tai opor a age ar modde log␣
      ↪transformation bad dite hobe . tai Age and Fare k alada alada pipeline a dia␣
      ↪korle valo hobe ai khatre
```

```
#Ages are likly Normal Distribution,But Fares are Left Skewed(need to be Normal␣
↪Distribution for better performance)
```



[ ]:
```
#for pipe2 (Age and Fare Individual Pipeline)

#opor ar 'age_fare_pipe use korle accuracy kharap ashte pare karon age normal␣
↪distribution(skewness) thik tahke na

#Age pipeline
age_pipe=make_pipeline(SimpleImputer())

# fare Pipeline
fare_pipe=make_pipeline(FunctionTransformer(np.log1p))
```

[ ]:
```
#Embarked---->Missing value handeling
#Embarked+Sex --->Encoding

#for Catagorical(String)
sex_emb_pipe=Pipeline([
    ('emb_imp',SimpleImputer(strategy='most_frequent')),
    ('emb_ohe',OneHotEncoder(dtype=np.
 ↪int32,drop='first',sparse_output=False,handle_unknown='ignore'))
])
```

[ ]:
```
CT1=ColumnTransformer([
    ('Age_Fare_Pipe',age_fare_pipe,['Age','Fare']),
    ('Sex_Emb_Pipe',sex_emb_pipe,['Sex','Embarked'])
],remainder='passthrough')
```

[ ]:
```
#for pipe2 (Age and Fare Individual Pipeline) [used in pipe2]

CT2=ColumnTransformer([
    ('age_pipe',age_pipe,[2]),
    ('fare_pipe',fare_pipe,[5]),
    ('sex_emb_pipe',sex_emb_pipe,[1,6])
],remainder='passthrough')
```

[ ]:
```
#for pipe3 (Without Log transformation)[used in pipe3]

CT3=make_column_transformer(␣
 ↪(age_pipe,['Age']),(sex_emb_pipe,['Sex','Embarked']))
```

[ ]:
```
tf_data=CT1.fit_transform(X_train)
tf_data
```

[ ]:
```
array([[3.83945231, 3.38439026, 1.        , …, 1.        , 0.        ,
        0.        ],
       [3.17805383, 2.63905733, 1.        , …, 2.        , 0.        ,
        0.        ],
       [3.49650756, 2.18885633, 1.        , …, 3.        , 0.        ,
        0.        ],
       …,
       [3.73766962, 2.71524426, 1.        , …, 3.        , 2.        ,
        0.        ],
       [2.7080502 , 4.79579055, 0.        , …, 1.        , 1.        ,
        2.        ],
       [3.09104245, 4.36038795, 1.        , …, 1.        , 0.        ,
        1.        ]])
```

[ ]:
```
# ct2=ColumnTransformer([
#     ('all_scal',MinMaxScaler(),slice(0,8))
# ])# ai khane remainder='passthrough' use korle error ashbe
```

[ ]:
```
ct3=LogisticRegression()
```

#Pipe-1 (Age and Fare both applied Log Transformation)

[ ]:
```
pipe1=Pipeline([
    ('CT1',CT1),
    # ('ct2',ct2),
    ('scal_all',MinMaxScaler()),
    ('ct3',ct3)
])

pipe1.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[ ]:
```
Pipeline(steps=[('CT1',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('Age_Fare_Pipe',
                                                  Pipeline(steps=[('age_imp',
SimpleImputer()),
                                                                  ('age_log_tf',
FunctionTransformer(func=<ufunc 'log1p'>))]),
                                                  ['Age', 'Fare']),
                                                 ('Sex_Emb_Pipe',
                                                  Pipeline(steps=[('emb_imp',
SimpleImputer(strategy='most_frequent')),
                                                                  ('emb_ohe',
OneHotEncoder(drop='first',
 dtype=<class 'numpy.int32'>,
 handle_unknown='ignore',
 sparse_output=False))]),
                                                  ['Sex', 'Embarked'])])),
                ('scal_all', MinMaxScaler()), ('ct3', LogisticRegression())])
```

[ ]:
```
y_pred=pipe1.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy_score(y_pred,y_test)
```

[ ]:
```
0.7988826815642458
```

[ ]:

```python
from sklearn.model_selection import cross_val_score
cross_val_score(pipe1,X_train,y_train,cv=5,scoring='accuracy').mean()
```

[ ]:

```
np.float64(0.8033290653008963)
```

#Pipe-2 (Age and Fare Individual Pipeline)

[ ]:

```python
#for pipe2 (Age and Fare Individual Pipeline)

pipe2=make_pipeline(CT2,MinMaxScaler(),ct3)

pipe2.fit(X_train,y_train)
```

[ ]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('age_pipe',
Pipeline(steps=[('simpleimputer',
SimpleImputer())]),
                                                   [2]),
                                                  ('fare_pipe',
Pipeline(steps=[('functiontransformer',
FunctionTransformer(func=<ufunc 'log1p'>))]),
                                                   [5]),
                                                  ('sex_emb_pipe',
                                                   Pipeline(steps=[('emb_imp',
SimpleImputer(strategy='most_frequent')),
                                                                   ('emb_ohe',
OneHotEncoder(drop='first',
 dtype=<class 'numpy.int32'>,
 handle_unknown='ignore',
 sparse_output=False))]),
                                                   [1, 6])])),
                ('minmaxscaler', MinMaxScaler()),
                ('logisticregression', LogisticRegression())])
```

[ ]:

```python
#for pipe2 (Age and Fare Individual Pipeline)
y_pred=pipe2.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy_score(y_pred,y_test)
```

[ ]:

```
0.7932960893854749
```

[ ]:

```python
#for pipe2 (Age and Fare Individual Pipeline)

from sklearn.model_selection import cross_val_score
cross_val_score(pipe2,X_train,y_train,cv=5,scoring='accuracy').mean()
```

[ ]:
```
np.float64(0.7921008568895893)
```

#Pipe-3 (Without Log Transformation)

[ ]:
```
#for pipe3 (Without Log transformation)

pipe3=Pipeline([
    ('CT3',CT3),
    ('Scaling',StandardScaler()),
    ('model',LogisticRegression())
])

pipe3.fit(X_train,y_train)
```

[ ]:
```
Pipeline(steps=[('CT3',
                 ColumnTransformer(transformers=[('pipeline-1',
Pipeline(steps=[('simpleimputer',
SimpleImputer())]),
                                                  ['Age']),
                                                 ('pipeline-2',
                                                  Pipeline(steps=[('emb_imp',
SimpleImputer(strategy='most_frequent')),
                                                                  ('emb_ohe',
OneHotEncoder(drop='first',
 dtype=<class 'numpy.int32'>,
 handle_unknown='ignore',
 sparse_output=False))]),
                                                  ['Sex', 'Embarked'])])),
                ('Scaling', StandardScaler()),
                ('model', LogisticRegression())])
```

[ ]:
```
#for pipe3 (without Log Transformation)

y_pred=pipe3.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

[ ]:
```
0.7821229050279329
```

[ ]:
```
#for pipe3 (without Log Transformation)

from sklearn.model_selection import cross_val_score
cross_val_score(pipe2,X_train,y_train,cv=5,scoring='accuracy').mean()
```

[ ]:
```
np.float64(0.7921008568895893)
```

[ ]:

```

```

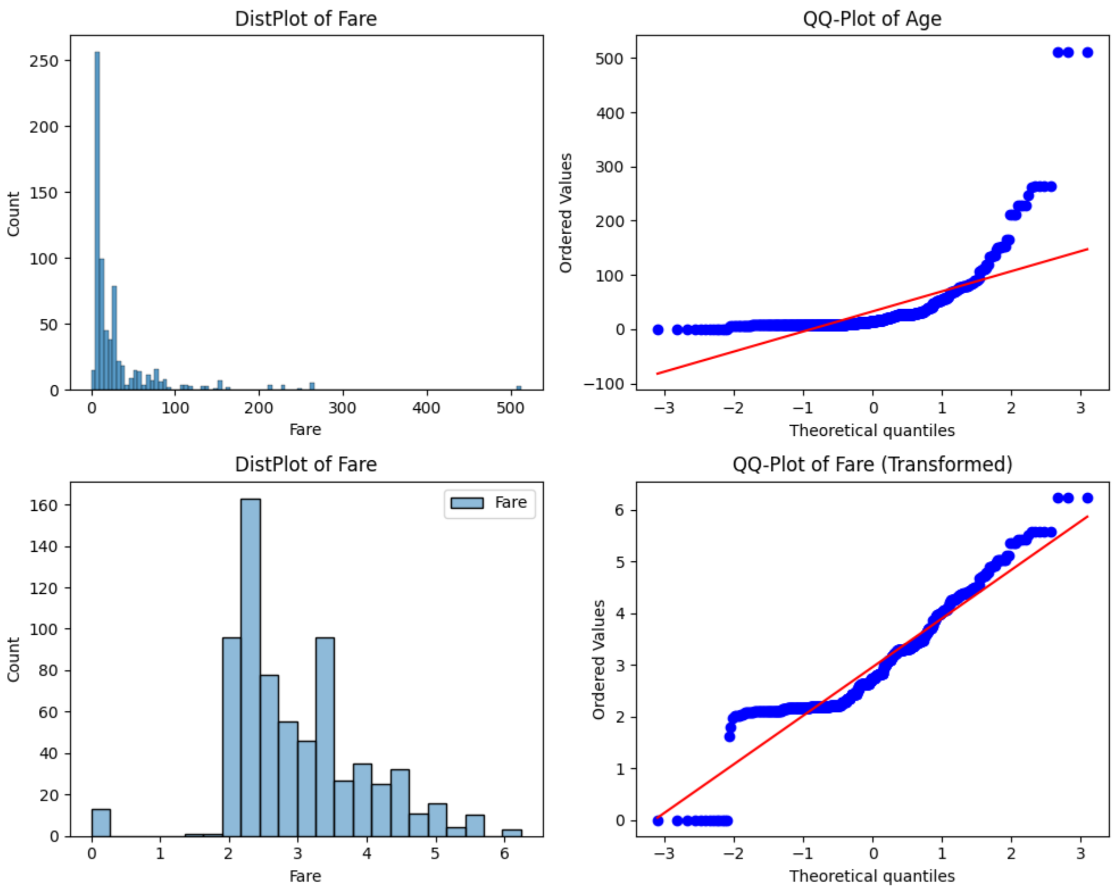
[ ]:

```
def CheckDistribution(fun):
  ft=FunctionTransformer(fun)
  fare_tf=ft.fit_transform(X_train[['Fare']])
  print(fare_tf)
  #analyze ,is function transformation need?
  plt.figure(figsize=(10,8))

  plt.subplot(221)
  sns.histplot(X_train['Fare'])
  plt.xlabel('Fare')
  plt.title("DistPlot of Fare")

  plt.subplot(222)
  stats.probplot(X_train['Fare'],plot=plt)
  # plt.xlabel('Age')
  plt.title("QQ-Plot of Age")

  plt.subplot(223)
  sns.histplot(fare_tf)
  plt.xlabel('Fare')
  plt.title("DistPlot of Fare")

  plt.subplot(224)
  stats.probplot(fare_tf.values.flatten(), plot=plt)
  plt.title("QQ-Plot of Fare (Transformed)")


  plt.tight_layout()
```

[ ]:

```
CheckDistribution(np.log1p)
```

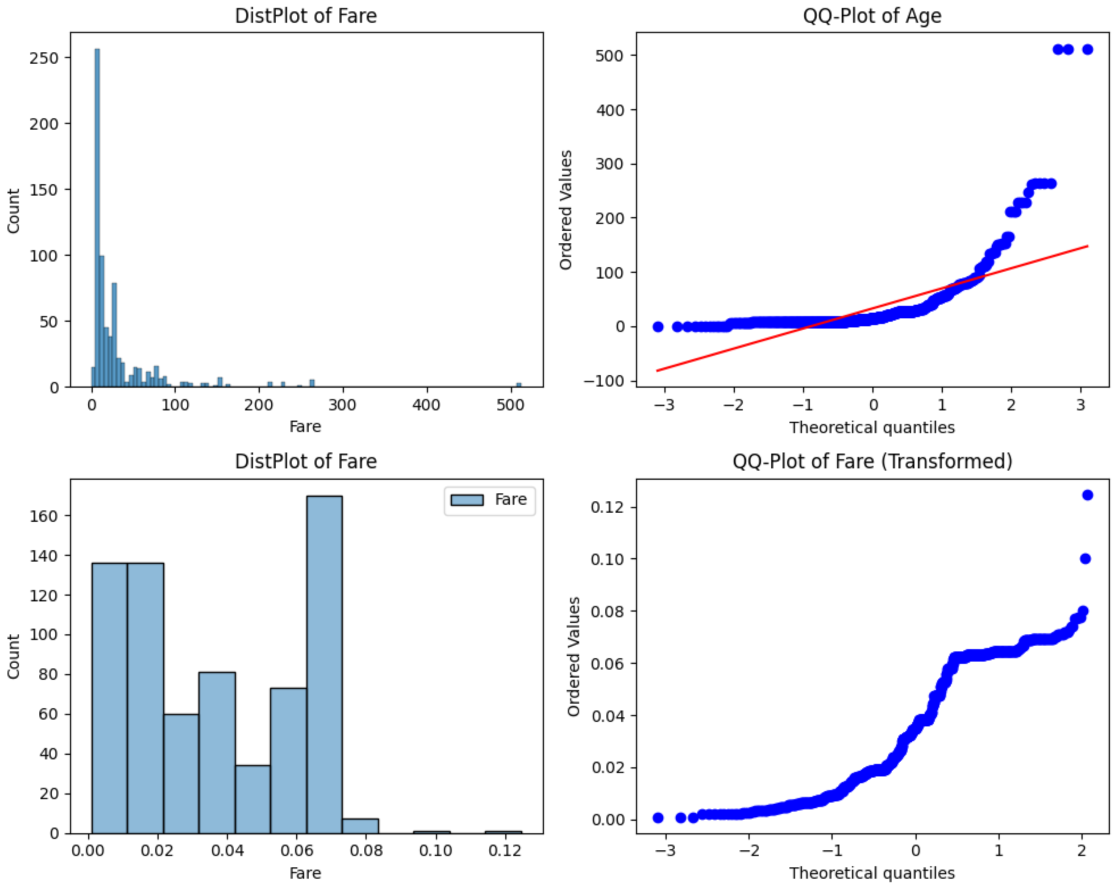
```
         Fare
331  3.384390
733  2.639057
382  2.188856
704  2.180892
813  3.474293
..        …
106  2.157559
270  3.465736
860  2.715244
435  4.795791
102  4.360388

[712 rows x 1 columns]
```



```
[ ]: #check which Function give you More Normarl Distribution and use this function␣
      ↪in FunctionTransformation
     CheckDistribution(lambda a: 1/(a+a))
```

```
         Fare
331  0.017544
733  0.038462
382  0.063091
704  0.063660
813  0.015987
..        …
106  0.065359
270  0.016129
860  0.035440
435  0.004167
102  0.006469

[712 rows x 1 columns]

/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:2767:
```

```
RuntimeWarning: invalid value encountered in subtract
  X -= avg[:, None]
```



[ ]:
```
#End of Day-1
```

#Day-31:Power Transformation

- Box-Cox
- Yeo Jhanson

0.1 1. Formulas

0.1.1 Box-Cox Transformation

Used **only for positive values**.

$$y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ \ln(y), & \text{if } \lambda = 0 \end{cases}$$

0.1.2 Yeo-Johnson Transformation

Used for **both positive and negative values**.

$$y(\lambda) = \begin{cases} \frac{[(y+1)^{\lambda}-1]}{\lambda}, & y \geq 0, \lambda \neq 0 \\ \ln(y+1), & y \geq 0, \lambda = 0 \\ \frac{-[(-y+1)^{2-\lambda}-1]}{2-\lambda}, & y < 0, \lambda \neq 2 \\ -\ln(-y+1), & y < 0, \lambda = 2 \end{cases}$$

0.2 2. When to Use

Transformation	When to Use
Box-Cox	When all your data is **positive** and **right/left skewed**
Yeo-Johnson	When data contains **zero or negative values**, or mix of signs

---

0.3 3. Where to Use (Use Cases)

Both are often used:

- Before **Linear Regression**, **Logistic Regression**, or **SVM** (models that assume normality)
- For **feature engineering** (to make features more Gaussian-like)
- When you want to **stabilize variance**
- When you're preparing data for **parametric models** (those that assume normality)

0.4 4. Key Differences

Feature	Box-Cox	Yeo-Johnson
Supports Negative?	No	Yes
Supports Zero?	No	Yes
Suitable For?	Positive data only	Any real numbers
In scikit-learn?	`power_transform()` with method='box-cox'	`power_transform()` with method='yeo-johnson'

0.5 5. How to Use in Python

```
from sklearn.preprocessing import PowerTransformer

# Box-Cox (only positive values)
```

```python
pt_boxcox = PowerTransformer(method='box-cox')
X_trans_boxcox = pt_boxcox.fit_transform(X)

# Yeo-Johnson (handles negatives too)
pt_yj = PowerTransformer(method='yeo-johnson')
X_trans_yj = pt_yj.fit_transform(X)
```

0.6 Pro Tips

- If you're not sure whether your data has negatives or not → use **Yeo-Johnson** (safer choice).
- Always **check skewness** before and after transformation.
- Don't use these transformations with **categorical features**.
- After transformation, it's common to apply **scaling** like `StandardScaler`.

Function Transformation Power Transformation

	Function Transformation	Power Transformation
(Definition) (Goal)	transformation (: log, sqrt, exp) Distribution scale	mathematical family (Box-Cox Yeo-Johnson) normalize Distribution- **Gaussian/Normal-**
	`np.log`, `np.sqrt`, `np.exp`, `1/x`,	`PowerTransformer` → method: `box-cox` or `yeo-johnson`
	, optimal	
Parameter optimization	,	, lambda automatically optimize
Negative value support	(`log`)	`Box-Cox`: `Yeo-Johnson`:
Where used		best transformation

[ ]:

[ ]:
```python
X_train.head(2)
```

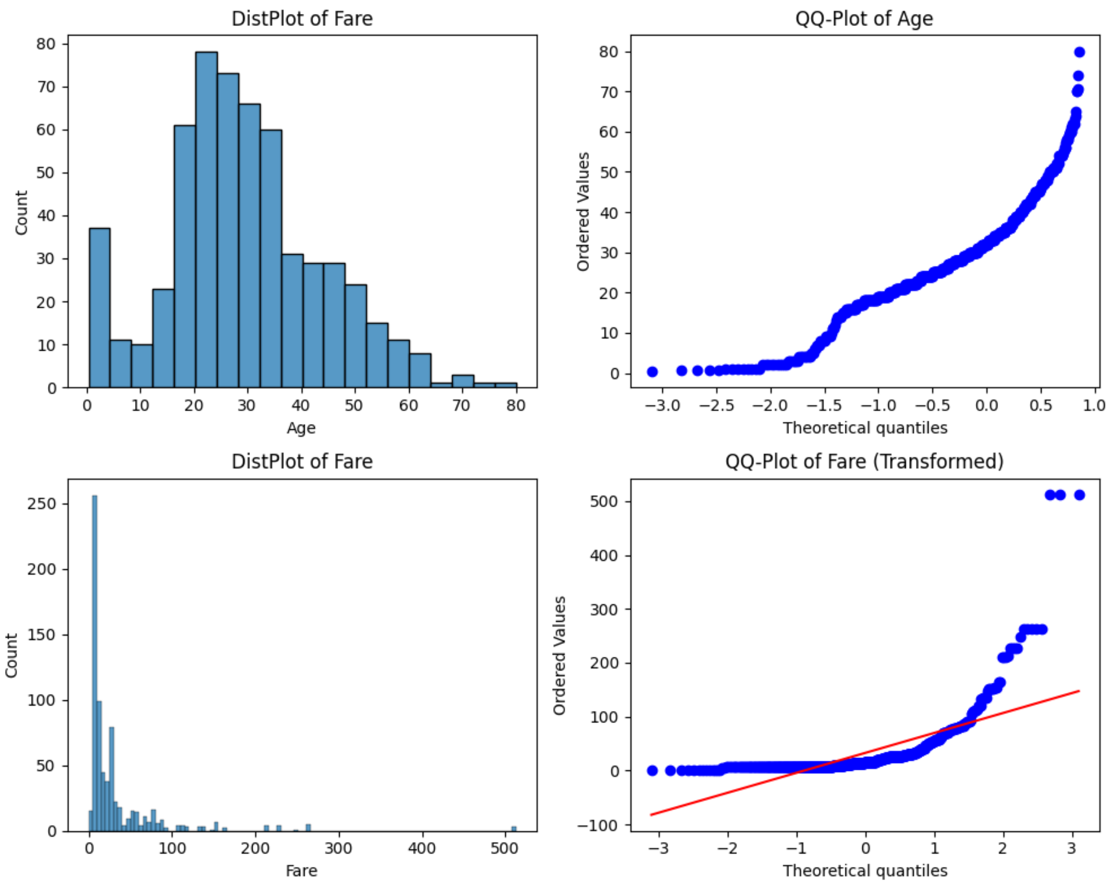
[ ]:
```
     Pclass   Sex   Age  SibSp  Parch  Fare Embarked
331       1  male  45.5      0      0  28.5        S
733       2  male  23.0      0      0  13.0        S
```

[ ]:
```python
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 712 entries, 331 to 102
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Pclass    712 non-null    int64  
 1   Sex       712 non-null    object 
 2   Age       572 non-null    float64
 3   SibSp     712 non-null    int64  
 4   Parch     712 non-null    int64  
 5   Fare      712 non-null    float64
 6   Embarked  710 non-null    object 
dtypes: float64(2), int64(3), object(2)
memory usage: 44.5+ KB
```

[ ]:

```
#Before Power Transformation

plt.figure(figsize=(10,8))

plt.subplot(221)
sns.histplot(X_train['Age'])
plt.xlabel('Age')
plt.title("DistPlot of Fare")

plt.subplot(222)
stats.probplot(X_train['Age'],plot=plt)
# plt.xlabel('Age')
plt.title("QQ-Plot of Age")

plt.subplot(223)
sns.histplot(X_train['Fare'])
plt.xlabel('Fare')
plt.title("DistPlot of Fare")

plt.subplot(224)
stats.probplot(X_train['Fare'], plot=plt)
plt.title("QQ-Plot of Fare (Transformed)")


plt.tight_layout()
```



```
[ ]: from sklearn.preprocessing import PowerTransformer
```

```
[ ]: age_fare_pipe=Pipeline([
         ('age_imp',SimpleImputer()),
         ('age_fare_pow_tf',PowerTransformer(method='yeo-johnson'))
     ])
     #Fare or Age contain 0 or (-)ve value thats why we can't use 'box-cox'␣
      ↪powerTransfomation
```

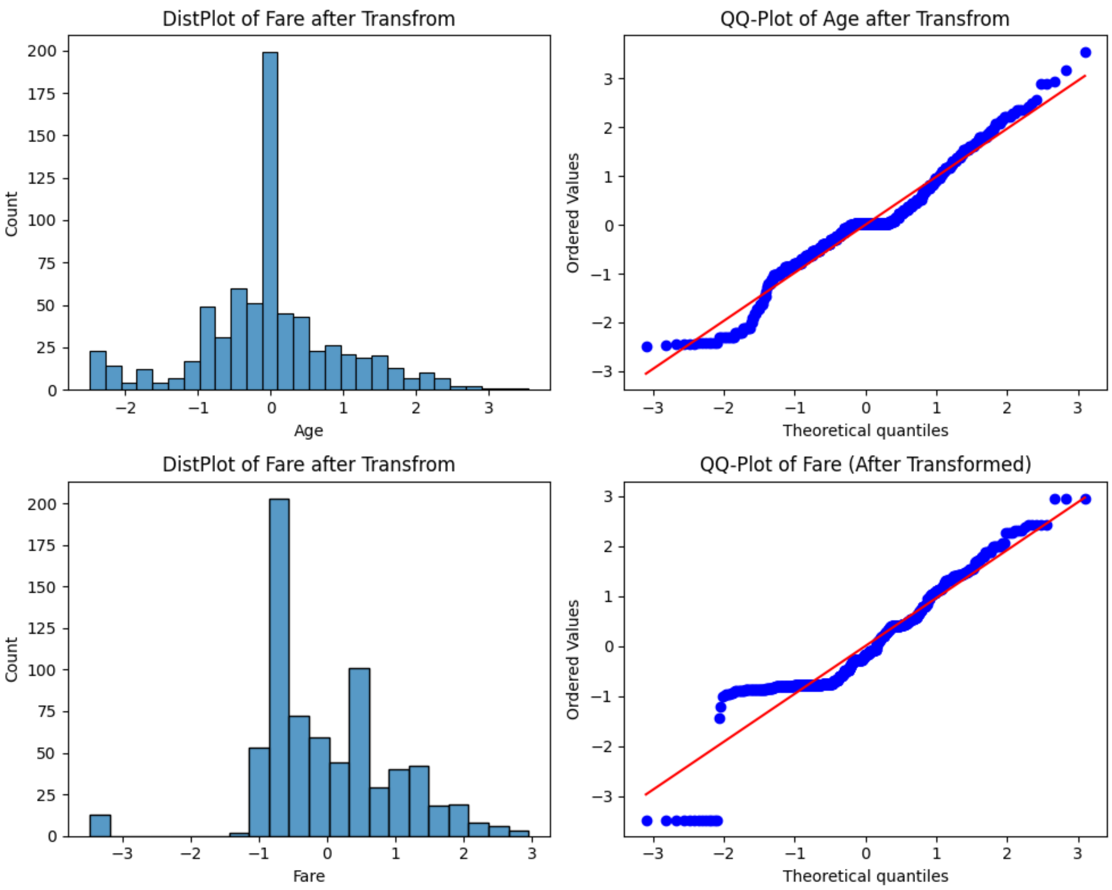
```
[ ]: pow_tf=ColumnTransformer([
         ('age_fare_pipe',age_fare_pipe,['Age','Fare'])
     ])
     pow_tf_data=pow_tf.fit_transform(X_train)
     pow_tf_data
```

```
[ ]: array([[ 1.20928949,  0.47999826],
            [-0.4654696 , -0.28375264],
            [ 0.22501445, -0.77244668],
            …,
            [ 0.88680715, -0.20313477],
            [-1.19799775,  1.78386853],
            [-0.62393578,  1.40043606]])
```

[ ]:
```
pow_tf.named_transformers_['age_fare_pipe'].named_steps['age_fare_pow_tf'].
  ↪lambdas_
```

[ ]:
```
array([ 0.84650131, -0.09664147])
```

[ ]:
```
#After Power Transformation

plt.figure(figsize=(10,8))

plt.subplot(221)
sns.histplot(pow_tf_data[:,0])
plt.xlabel('Age')
plt.title("DistPlot of Fare after Transfrom")

plt.subplot(222)
stats.probplot(pow_tf_data[:,0],plot=plt)
# plt.xlabel('Age')
plt.title("QQ-Plot of Age after Transfrom")

plt.subplot(223)
sns.histplot(pow_tf_data[:,1])
plt.xlabel('Fare')
plt.title("DistPlot of Fare after Transfrom")

plt.subplot(224)
stats.probplot(pow_tf_data[:,1], plot=plt)
plt.title("QQ-Plot of Fare (After Transformed)")


plt.tight_layout()
```



```
[ ]: sex_emb_pipe=make_pipeline(
         SimpleImputer(strategy='most_frequent'),
         OneHotEncoder(dtype=np.
      ↪int32,drop='first',sparse_output=False,handle_unknown='ignore')

      )
```

```
[ ]: CT1=ColumnTransformer([
         ('age_fare_pipe',age_fare_pipe,['Age','Fare']),
         ('sex_emb_pipe',sex_emb_pipe,['Sex','Embarked'])
     ],remainder='passthrough')
```

```
[ ]: pipe_for_power_tf=make_pipeline(
         CT1,
         StandardScaler(),
         LogisticRegression()
     )
```

```
[ ]: pipe_for_power_tf.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

```
[ ]: Pipeline(steps=[('columntransformer',
                      ColumnTransformer(remainder='passthrough',
                                        transformers=[('age_fare_pipe',
                                                       Pipeline(steps=[('age_imp',
     SimpleImputer()),
     ('age_fare_pow_tf',
     PowerTransformer())]),
                                                       ['Age', 'Fare']),
                                                      ('sex_emb_pipe',
     Pipeline(steps=[('simpleimputer',
     SimpleImputer(strategy='most_frequent')),
     ('onehotencoder',
     OneHotEncoder(drop='first',
      dtype=<class 'numpy.int32'>,
      handle_unknown='ignore',
      sparse_output=False))]),
                                                       ['Sex', 'Embarked'])])),
                     ('standardscaler', StandardScaler()),
                     ('logisticregression', LogisticRegression())])
```

```
[ ]: y_pred=pipe_for_power_tf.predict(X_test)
```

```
[ ]: from sklearn.metrics import accuracy_score
     accuracy_score(y_pred,y_test)
```

```
[ ]: 0.8044692737430168
```

```
[ ]: from sklearn.model_selection import cross_val_score
     cross_val_score(pipe_for_power_tf,X_train,y_train,cv=5,scoring='accuracy').
      ↪mean()
```

```
[ ]: np.float64(0.7892642568698907)
```

```
[ ]:
```

#Extra for Known

```
[ ]: pt=PowerTransformer()
     pt.fit_transform(X_train[['Age','Fare']])
     pt.lambdas_

     # aikhane age a missing value ase bole Lambdas ar mand kom hocce  na hoi opor␣
      ↪ar shate thik ei ase
```

```
[ ]: array([ 0.78461262, -0.09664147])
```

```
[ ]:
```