# Feature Engineering

## 1 feature engineering

#label encoding using for different techniques

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[23]:
```python
df=pd.read_csv('train.csv')
```

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

[3]:

| | Sex |
|---|---|
| 0 | male |
| 1 | female |
| 2 | female |
| 3 | female |
| 4 | male |

[4]:
```python
#  1.pd.get_dummies
df1=pd.get_dummies(df['Sex'])
```

[5]:
```python
df1
```

[5]:

| | female | male |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 2 | 1 | 0 |
| 3 | 1 | 0 |
| 4 | 0 | 1 |
| .. | ... | ... |
| 886 | 0 | 1 |
| 887 | 1 | 0 |
| 888 | 1 | 0 |
| 889 | 0 | 1 |
| 890 | 0 | 1 |

[891 rows x 2 columns]

[6]:
```python
# 2.Label  Encoder
from sklearn.preprocessing import LabelEncoder
```

[9]:
```python
oe=LabelEncoder()
```

[17]:
```python
Label=oe.fit_transform(df['Sex'])
```

[19]:
```python
Label
```

[19]:
```
array([1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
```

```
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1])
```

[39]:
```
# 3. ordinal encoding

from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define ordinal encoding
encoder = OrdinalEncoder()
# transform data
result = encoder.fit_transform(data)
print(result)
```

```
[['red']
 ['green']
 ['blue']]
[[2.]
 [1.]
 [0.]]
```

[40]:
```
# 4. one hot encoding

from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(sparse=False)
# transform data
onehot = encoder.fit_transform(data)
print(onehot)
```

```
[['red']
 ['green']
 ['blue']]
[[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
```

[41]:
```
import datetime
```

[42]:
```
today_date=datetime.datetime.today()
```

[43]:
```python
today_date
```

[43]: datetime.datetime(2021, 10, 2, 22, 19, 45, 120558)

[44]:
```python
today_date-datetime.timedelta(3)
```

[44]: datetime.datetime(2021, 9, 29, 22, 19, 45, 120558)

[46]:
```python
days=[today_date-datetime.timedelta(x) for x in range(0,15)]
days
```

[46]:
```
[datetime.datetime(2021, 10, 2, 22, 19, 45, 120558),
 datetime.datetime(2021, 10, 1, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 30, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 29, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 28, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 27, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 26, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 25, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 24, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 23, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 22, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 21, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 20, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 19, 22, 19, 45, 120558),
 datetime.datetime(2021, 9, 18, 22, 19, 45, 120558)]
```

[47]:
```python
import pandas as pd
data=pd.DataFrame(days)
data.columns=["Day"]
```

[48]:
```python
data.head()
```

[48]:

| | Day |
|---|---|
| 0 | 2021-10-02 22:19:45.120558 |
| 1 | 2021-10-01 22:19:45.120558 |
| 2 | 2021-09-30 22:19:45.120558 |
| 3 | 2021-09-29 22:19:45.120558 |
| 4 | 2021-09-28 22:19:45.120558 |

[ ]:
```python
data["weekday"]=data["Day"].dt.weekday_name
data.head()
```

[52]:
```python
dictionary={"Monday":1,"Tuesday":2,"Wednesday":3,"Thursday":4,"Friday":
↪5,"Saturday":6,"Sunday":7



}
```

[53]:
```python
dictionary
```

[53]:
```
{"Monday": 1,
 "Tuesday": 2,
```

```
'Wednesday': 3,
'Thursday': 4,
'Friday': 5,
'Saturday': 6,
'Sunday': 7}
```

[ ]:
```
data['weekday_ordinal']=data['weekday'].map(dictionary)
```

[55]:
```
# 5.count fequency
train_set = pd.read_csv('http://archive.ics.uci.edu/ml/
↪machine-learning-databases/adult/adult.data' , header = None,index_col=None)
train_set.head()
```

[ ]:
```
columns=[1,3,5,6,7,8,9,13]
```

[ ]:
```
train_set=train_set[columns]
```

[ ]:
```
train_set.
↪columns=['Employment','Degree','Status','Designation','family_job','Race','Sex','Country']
train_set.head()
```

[ ]:
```
for feature in train_set.columns[:]:
    print(feature,":",len(train_set[feature].unique()),'labels')
```

[ ]:
```
country_map=train_set['Country'].value_counts().to_dict()
```

[ ]:
```
train_set['Country']=train_set['Country'].map(country_map)
```

[ ]:
```
train_set.head(20)
```

## 2 Missing Values- Feature Engineering

## 3 # Mean/ Median /Mode imputation

When should we apply? Mean/median imputation has the assumption that the data are missing completely at random(MCAR). We solve this by replacing the NAN with the most frequent occurance of the variables

[ ]:
```
#using the median
age=df.fillna(df.Age.median())
df1['Age']=df1['Age'].apply(np.int64)
```

[ ]:
```
#using the mean
df['Age']=df.fillna(df.Age.mean())
```

[ ]:
```
#When we deal the catogorical values we use mode
df['SibSp']=df.fillna(df.SibSp.Mode()[0])
```

[57]:
```
df.isnull().sum()
```

[57]:
```
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
```

[59]:
```python
df=pd.read_csv('train.csv',usecols=['Age','Fare','Survived'])
df.head()
```

[59]:

| | Survived | Age | Fare |
|---|---|---|---|
| 0 | 0 | 22.0 | 7.2500 |
| 1 | 1 | 38.0 | 71.2833 |
| 2 | 1 | 26.0 | 7.9250 |
| 3 | 1 | 35.0 | 53.1000 |
| 4 | 0 | 35.0 | 8.0500 |

[60]:
```python
## Lets go and see the percentage of missing values
df.isnull().mean()
```

[60]:
```
Survived    0.000000
Age         0.198653
Fare        0.000000
dtype: float64
```

[61]:
```python
def impute_nan(df,variable,median):
    df[variable+"_median"]=df[variable].fillna(median)
```

[62]:
```python
median=df.Age.median()
median
```

[62]: 28.0

[63]:
```python
impute_nan(df,'Age',median)
df.head()
```

[63]:

| | Survived | Age | Fare | Age_median |
|---|---|---|---|---|
| 0 | 0 | 22.0 | 7.2500 | 22.0 |
| 1 | 1 | 38.0 | 71.2833 | 38.0 |
| 2 | 1 | 26.0 | 7.9250 | 26.0 |
| 3 | 1 | 35.0 | 53.1000 | 35.0 |
| 4 | 0 | 35.0 | 8.0500 | 35.0 |

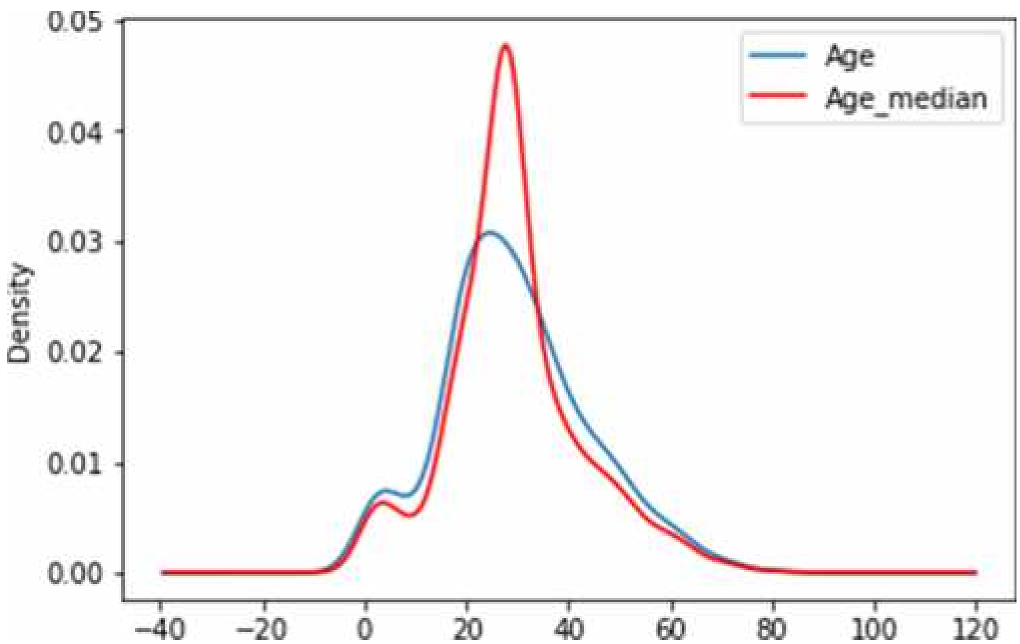
[64]:
```python
print(df['Age'].std())
print(df['Age_median'].std())
```

```
14.526497332334044
13.019696550973194
```

[65]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

[66]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
df['Age'].plot(kind='kde', ax=ax)
df.Age_median.plot(kind='kde', ax=ax, color='red')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

[66]: <matplotlib.legend.Legend at 0x13057ea5470>




# 4 Feature Scaling

#Normalization

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Here's the formula for normalization:

Normalization equation

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0 On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1 If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

#Standardization

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Here's the formula for standardization:

Standardization equation

Feature scaling: Muis the mean of the feature values andFeature scaling: Sigmais the standard deviation of the feature values. Note that in this case, the values are not restricted to a particular range.

Now, the big question in your mind must be when should we use normalization and when should we use standardization Let's find out

[67]:
```
from sklearn.preprocessing import StandardScaler,MinMaxScaler,RobustScaler
```

[ ]:
```
std=StandardScaler()
std.fit_transform(df['Variable'].reshape(-1,1).values())
```

[ ]:
```
min=MinMaxScaler()
min.fit_transform(['variable'].reshape(0,1).values())
```

[ ]:
```
Rob=RobustScaler()
Rob.fit_transform(df['variable'].reshape(0,1).values())
```

## 5 #Outliers Tretment

[69]:
```
df.head()
```

[69]:

| | Survived | Age | Fare | Age_median |
|---|---|---|---|---|
| 0 | 0 | 22.0 | 7.2500 | 22.0 |
| 1 | 1 | 38.0 | 71.2833 | 38.0 |
| 2 | 1 | 26.0 | 7.9250 | 26.0 |
| 3 | 1 | 35.0 | 53.1000 | 35.0 |
| 4 | 0 | 35.0 | 8.0500 | 35.0 |

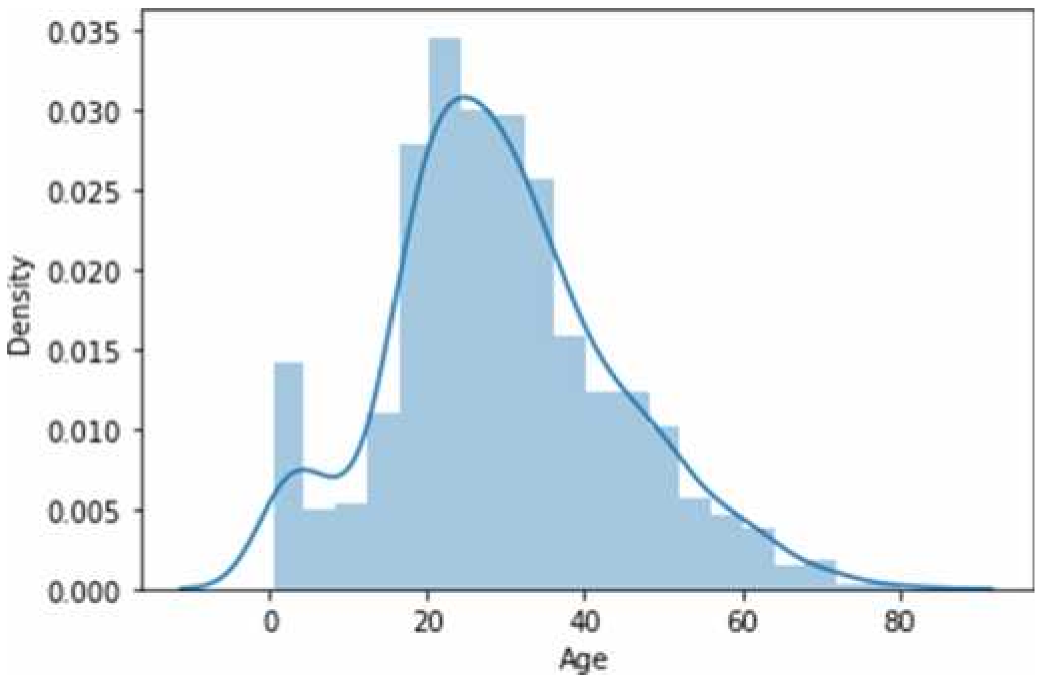
[70]:
```
df['Age'].isnull().sum()
```

[70]: 177

[71]:
```
import seaborn as sns
```

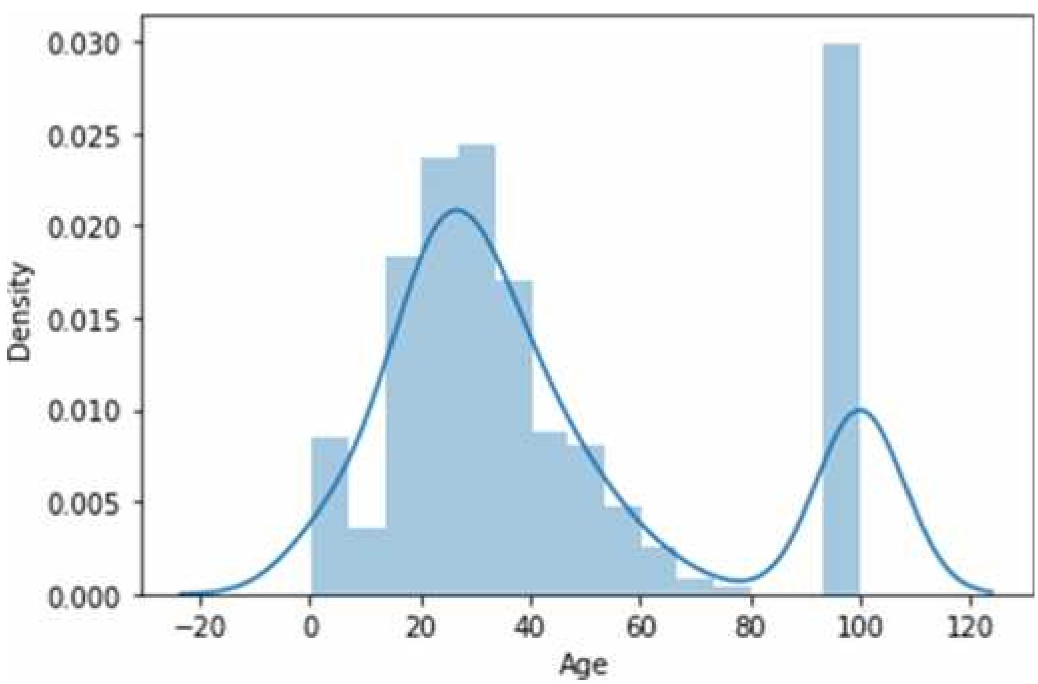
[72]:
```
sns.distplot(df['Age'].dropna())
```

[72]: <AxesSubplot:xlabel='Age', ylabel='Density'>



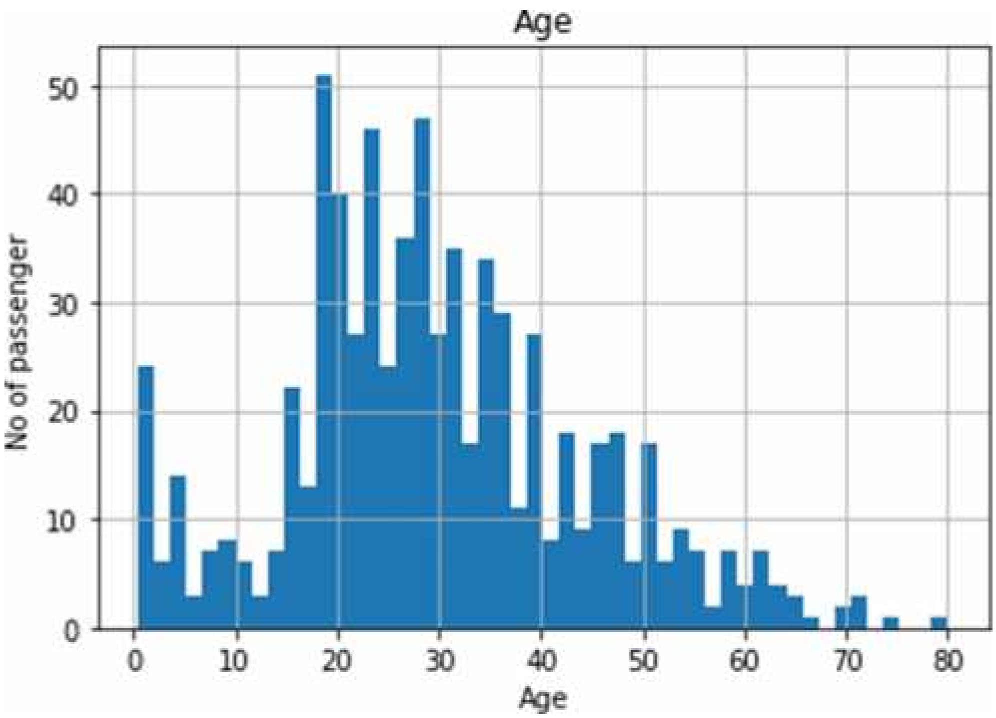

[73]:
```
sns.distplot(df['Age'].fillna(100))
```

[73]: <AxesSubplot:xlabel='Age', ylabel='Density'>

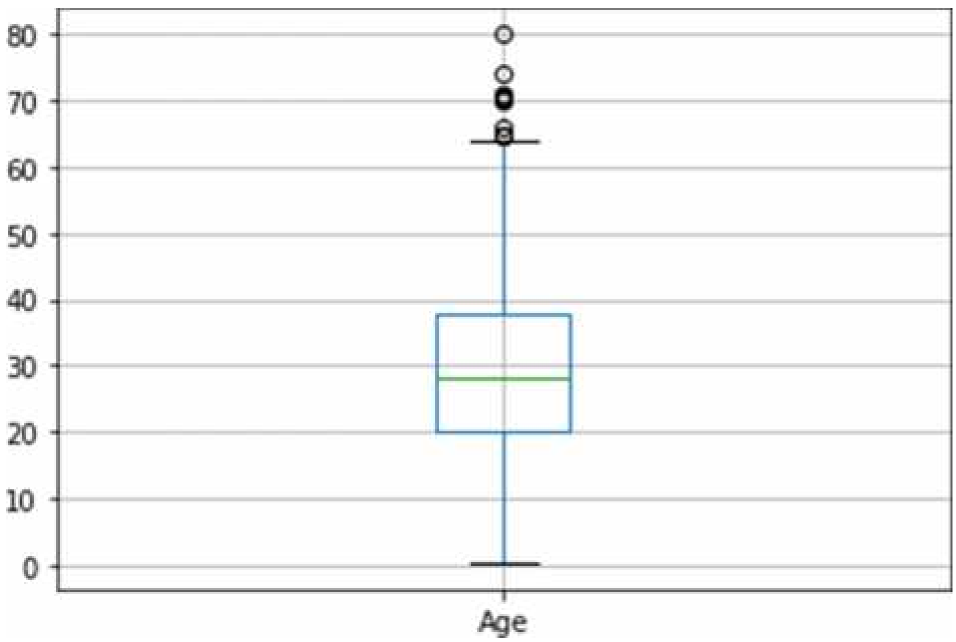


[74]:
```
figure=df.Age.hist(bins=50)
figure.set_title("Age")
figure.set_xlabel("Age")
figure.set_ylabel("No of passenger")
```

[74]: Text(0, 0.5, 'No of passenger')




[77]:
```
figure=df.boxplot(column="Age")
```




[76]:
```
df['Age'].describe()
```

[76]:
```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

## 6 If The Data Is Normally Distributed We use this

[78]:
```
### Assuming Age follows A Gaussian Distribution we will calculate the boundaries which differentiates the outliers

uppper_boundary=df['Age'].mean() + 3* df['Age'].std()
lower_boundary=df['Age'].mean() - 3* df['Age'].std()
print(lower_boundary), print(uppper_boundary),print(df['Age'].mean())
```

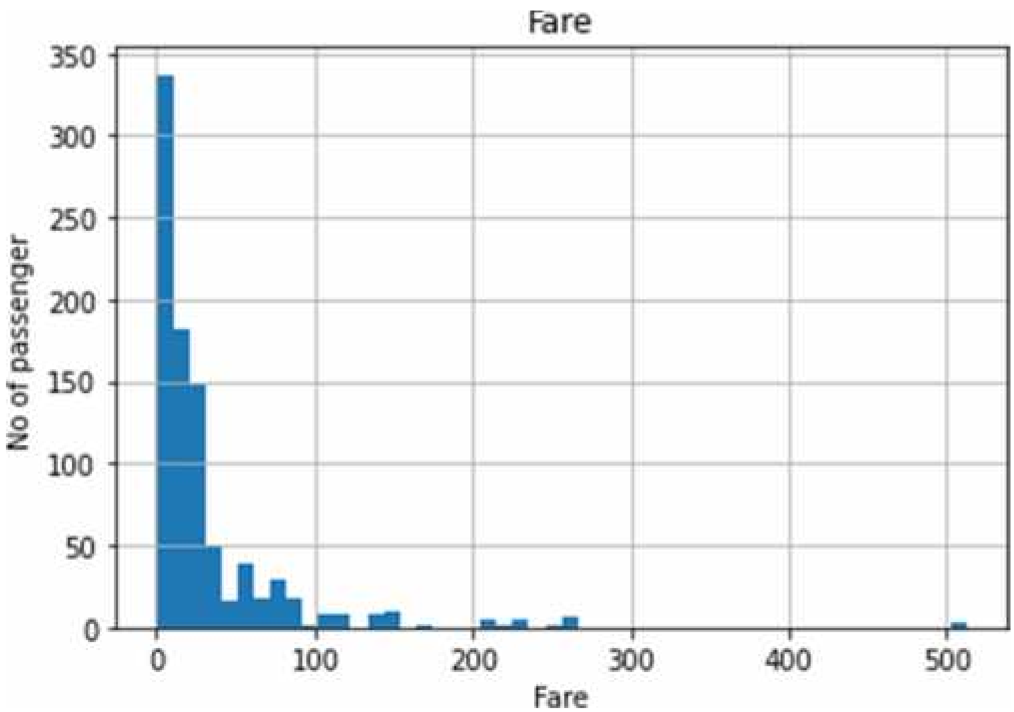
```
-13.88037434994331
73.27860964406095
29.69911764705882
```

[78] : (None, None, None)

## 7 If Features Are Skewed We Use the below Technique

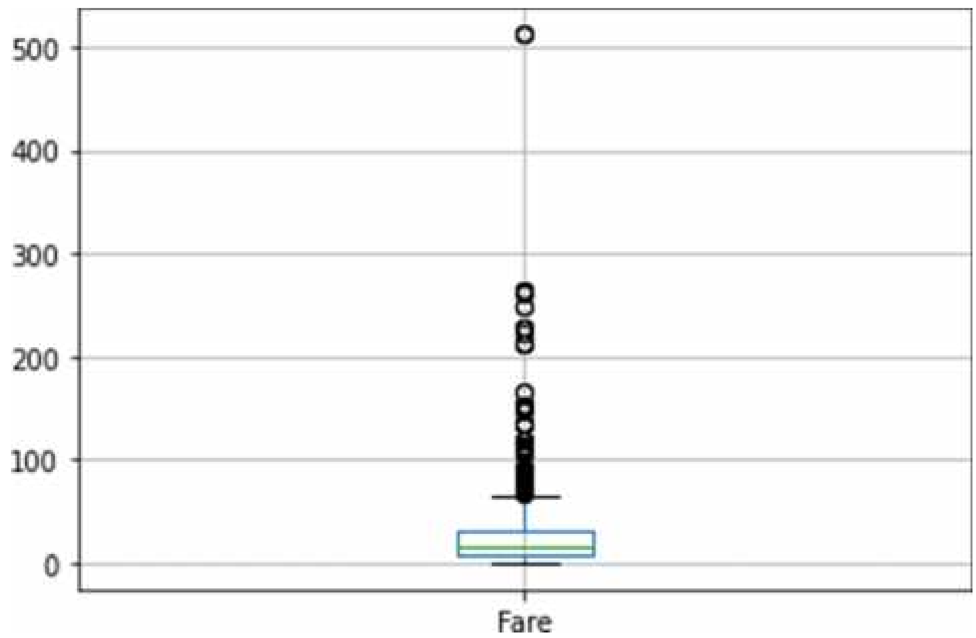
[79]:
```
figure=df.Fare.hist(bins=50)
figure.set_title("Fare")
figure.set_xlabel("Fare")
figure.set_ylabel("No of passenger")
```

[79] : Text(0, 0.5, 'No of passenger')




[80]:
```
df.boxplot(column="Fare")
```

[80] : <AxesSubplot:>




[81]:
```
df['Fare'].describe()
```

[81]:
```
count    891.000000
mean      32.204208
std       49.693429
min        0.000000
25%        7.910400
50%       14.454200
75%       31.000000
max      512.329200
Name: Fare, dtype: float64
```

[82]:
```
## Lets compute the Interquantile range to calculate the boundaries
IQR=df.Fare.quantile(0.75)-df.Fare.quantile(0.25)
```

[83]:
```
lower_bridge=df['Fare'].quantile(0.25)-(IQR*1.5)
upper_bridge=df['Fare'].quantile(0.75)+(IQR*1.5)
print(lower_bridge), print(upper_bridge)
```

```
-26.724
65.6344
```

[83]:
```
(None, None)
```

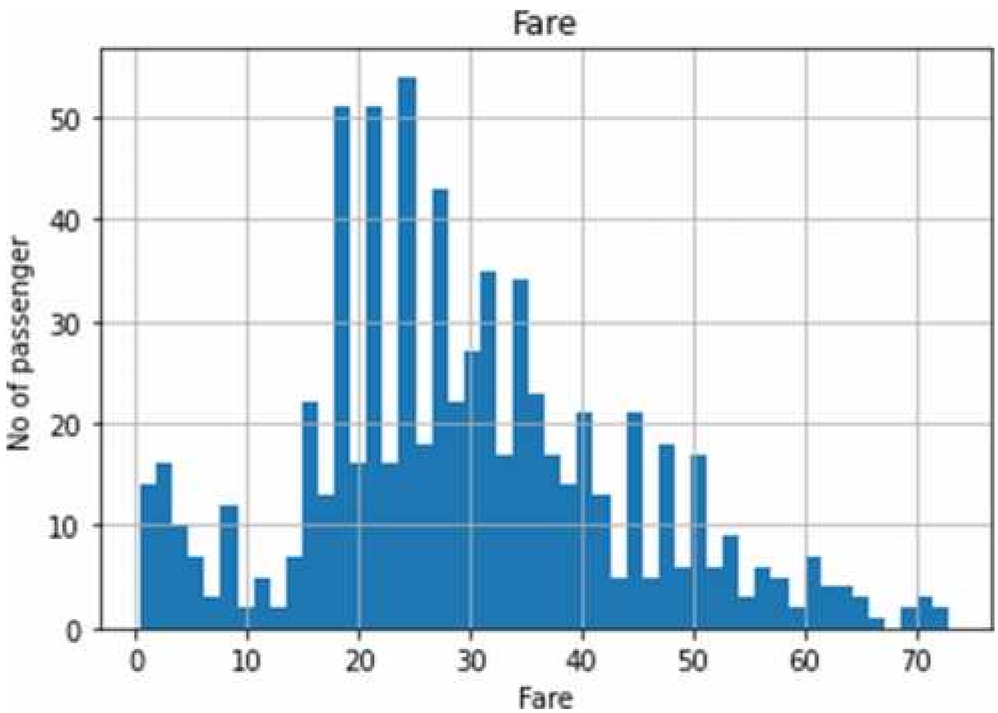
[84]:
```
data=df.copy()
```

[85]:
```
data.loc[data['Age']>=73,'Age']=73
```

[86]:
```
data.loc[data['Fare']>=100,'Fare']=100
```

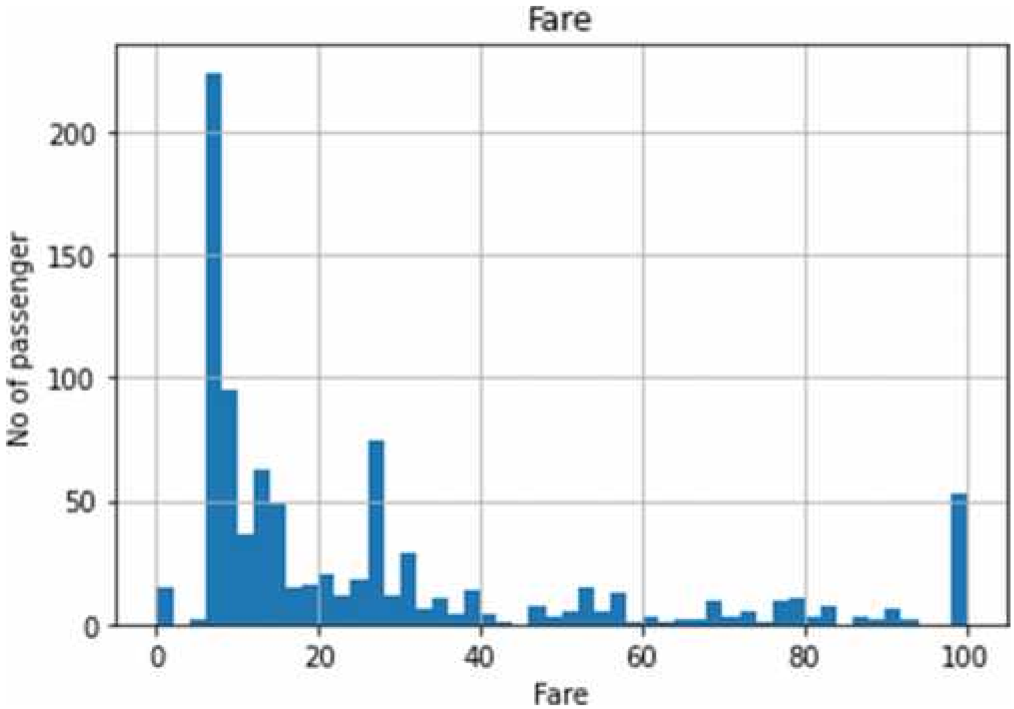
[87]:
```
figure=data.Age.hist(bins=50)
figure.set_title("Fare")
figure.set_xlabel("Fare")
figure.set_ylabel("No of passenger")
```

[87]: Text(0, 0.5, 'No of passenger')




[88]:
```
figure=data.Fare.hist(bins=50)
figure.set_title("Fare")
figure.set_xlabel("Fare")
figure.set_ylabel("No of passenger")
```

[88]: Text(0, 0.5, 'No of passenger')




# 8 Feature Selection

[ ]:
```
# Remove The correlated
threshold=0.8
```

[ ]:
```
# 1. find and remove correlated features


def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in␣
↪absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

[ ]:
```
correlation(df.iloc[:,:-1],threshold)
```

[ ]:
```
# 2.information gain get from indipendent variable and dependent variable

from sklearn.feature_selection import mutual_info_classif
```

[ ]:
```
mutual_info=mutual_info_classif(X,y)
```

[ ]:
```
mutual_data=pd.Series(mutual_info,index=X.columns)
mutual_data.sort_values(ascending=False)
```

[ ]:
```
# 3. Tree classifier extarct the importent features from dataset

from sklearn.ensemble import ExtraTreeClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_features=4, random_state=0)
clf = ExtraTreesClassifier(n_estimators=100, random_state=0)
clf.fit(X, y)
```

[ ]:
```
# 4. chi2 feature selection
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# Load iris data
iris_dataset = load_iris()

# Create features and target
X = iris_dataset.data
y = iris_dataset.target

# Convert to categorical data by converting data to integers
X = X.astype(int)

# Two features with highest chi-squared statistics are selected
chi2_features = SelectKBest(chi2, k = 2)
X_kbest_features = chi2_features.fit_transform(X, y)
```