Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer ,KNNImputer
import warnings
warnings.filterwarnings('ignore')
from sklearn.svm import SVC
from sklearn.ensemble import
RandomForestClassifier,RandomForestRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression,LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import
accuracy_score,precision_score,recall_score,f1_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import keras
from keras import layers
```

Load Data

```
data = pd.read_csv('E:\\dataset\\titanic\\train.csv')
data.head()
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                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
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0
```

```
   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Hndling missing Value

```
data.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
```

```
data.isna().sum()
```

```
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
```

Here I found

-in Cabin 204 non-null in training data then must drop 'Cabin' columns

-in Embarced 889 non-null just 2 Nulls ,I will drop

-in Age 714 non-null , I will impute with SimpleImputer

```
data.drop('Cabin' , axis=1 , inplace=True)
```

```
imputer=KNNImputer(n_neighbors=10)
data['Age']=imputer.fit_transform(data[['Age']])
data.dropna(inplace=True)
print(data.isna().sum())
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

here i tried to fill with simpler imputer but bad result

```
'''imputer=SimpleImputer()
data['Age']=imputer.fit_transform(data[['Age']])
data.dropna(inplace=True)
data.isna().sum()'''
```

```
"imputer=SimpleImputer()\
ndata['Age']=imputer.fit_transform(data[['Age']])\
ndata.dropna(inplace=True)\ndata.isna().sum()"
```

```
data.shape
```

```
(889, 11)
```

Extract Information

```
data['Name']
```

```
0                              Braund, Mr. Owen Harris
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
2                               Heikkinen, Miss. Laina
```

```
3           Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                                   Allen, Mr. William Henry
                                 ...
886                                    Montvila, Rev. Juozas
887                             Graham, Miss. Margaret Edith
888              Johnston, Miss. Catherine Helen "Carrie"
889                                    Behr, Mr. Karl Howell
890                                      Dooley, Mr. Patrick
Name: Name, Length: 889, dtype: object
```

Here i found the title of each one

```
data['Title']=data['Name'].str.extract(' ([A-Za-z]+)\.',expand=False)
data['Title'].unique()

array(['Mr', 'Mrs', 'Miss', 'Master', 'Don', 'Rev', 'Dr', 'Mme', 'Ms',
       'Major', 'Lady', 'Sir', 'Mlle', 'Col', 'Capt', 'Countess',
       'Jonkheer'], dtype=object)

data['TicketGroupSize']=data.groupby('Ticket')
['Ticket'].transform('count')
data['TicketGroupSize'].unique()

array([1, 2, 4, 3, 7, 5, 6], dtype=int64)

data.head()

   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                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
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Embarked Title  TicketGroupSize
0      0         A/5 21171   7.2500        S    Mr                1
1      0          PC 17599  71.2833        C   Mrs                1
```

```
2      0  STON/O2. 3101282   7.9250        S  Miss                   1
3      0            113803  53.1000        S   Mrs                   2
4      0            373450   8.0500        S    Mr                   1
```

Drop the columns

```
data=data.drop(['PassengerId','Ticket','Name'],axis=1)
data.head()
```

```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
Title  \
0         0       3    male  22.0      1      0   7.2500        S
Mr
1         1       1  female  38.0      1      0  71.2833        C
Mrs
2         1       3  female  26.0      0      0   7.9250        S
Miss
3         1       1  female  35.0      1      0  53.1000        S
Mrs
4         0       3    male  35.0      0      0   8.0500        S
Mr

   TicketGroupSize
0                1
1                1
2                1
3                2
4                1
```

Encoding

```
data.head()
```

```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
Title  \
0         0       3    male  22.0      1      0   7.2500        S
Mr
1         1       1  female  38.0      1      0  71.2833        C
Mrs
2         1       3  female  26.0      0      0   7.9250        S
Miss
3         1       1  female  35.0      1      0  53.1000        S
Mrs
4         0       3    male  35.0      0      0   8.0500        S
Mr

   TicketGroupSize
0                1
```

```
1                 1
2                 1
3                 2
4                 1
```

```python
data['Embarked'].unique()
```

```
array(['S', 'C', 'Q'], dtype=object)
```

```python
data['Sex'].unique()
```

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

```python
data['Sex'] = [1 if sex == 'male' else 0 for sex in data['Sex'] ]
data['Embarked'] = [0 if emb == 'S' else 1 if emb == 'C' else 2 for
emb in data['Embarked'] ]
data.head()
```

```
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked Title
\
0         0       3    1  22.0      1      0   7.2500         0    Mr

1         1       1    0  38.0      1      0  71.2833         1   Mrs

2         1       3    0  26.0      0      0   7.9250         0  Miss

3         1       1    0  35.0      1      0  53.1000         0   Mrs

4         0       3    1  35.0      0      0   8.0500         0    Mr


   TicketGroupSize
0                1
1                1
2                1
3                2
4                1
```

```python
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder ()
data ['Title']= encoder.fit_transform(data['Title'])
data.head()
```

```
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked  Title
\
0         0       3    1  22.0      1      0   7.2500         0     12

1         1       1    0  38.0      1      0  71.2833         1     13

2         1       3    0  26.0      0      0   7.9250         0      9

3         1       1    0  35.0      1      0  53.1000         0     13
```

```
4          0        3     1  35.0      0       0   8.0500         0      12


   TicketGroupSize
0                1
1                1
2                1
3                2
4                1
```

```
# data['Sex'] = [1 if sex == 'male' else 0 for sex in data['Sex'] ]
# test_data['Sex'] = [1 if sex == 'male' else 0 for sex in
test_data['Sex'] ]
# data['Embarked'] = [0 if emb == 'S' else 1 if emb == 'C' else 2 for
emb in data['Embarked'] ]
# test_data['Embarked'] = [0 if emb == 'S' else 1 if emb == 'C' else 2
for emb in test_data['Embarked'] ]
# data=pd.get_dummies(data,columns=['Title'],drop_first=True)
#
test_data=pd.get_dummies(test_data,columns=['Title'],drop_first=True)
# data.head()
```

Check Duplicates

```
duplicates = data[data.duplicated()]
data[data.duplicated()]
```

```
     Survived  Pclass  Sex        Age  SibSp  Parch     Fare  Embarked
Title  \
47          1       3    0  29.699118      0      0   7.7500         2
9
76          0       3    1  29.699118      0      0   7.8958         0
12
77          0       3    1  29.699118      0      0   8.0500         0
12
87          0       3    1  29.699118      0      0   8.0500         0
12
95          0       3    1  29.699118      0      0   8.0500         0
12
..        ...     ...  ...        ...    ...    ...      ...       ...
...
863         0       3    0  29.699118      8      2  69.5500         0
9
870         0       3    1  26.000000      0      0   7.8958         0
12
877         0       3    1  19.000000      0      0   7.8958         0
12
```

```
878          0        3    1  29.699118       0      0   7.8958         0
12
884          0        3    1  25.000000       0      0   7.0500         0
12

     TicketGroupSize
47                 1
76                 1
77                 1
87                 1
95                 1
..               ...
863                7
870                1
877                1
878                1
884                1

[98 rows x 10 columns]
```

```
data.drop_duplicates(inplace=True)
```

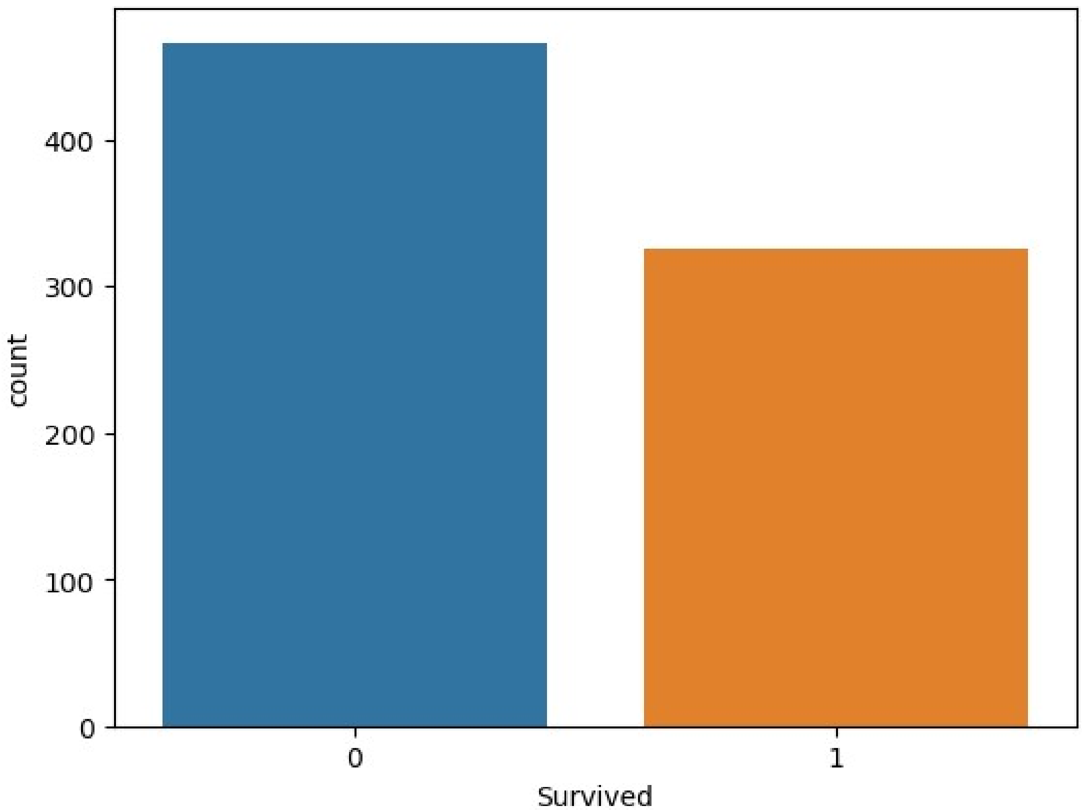
Check Balance

```
data['Survived'].value_counts()
```

```
Survived
0    466
1    325
Name: count, dtype: int64
```

```
sns.countplot(x='Survived',data=data)
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



Correlation

```
data.columns
```

```
Index(['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
       'Embarked', 'Title', 'TicketGroupSize'],
      dtype='object')
```

```
data.corr()
```

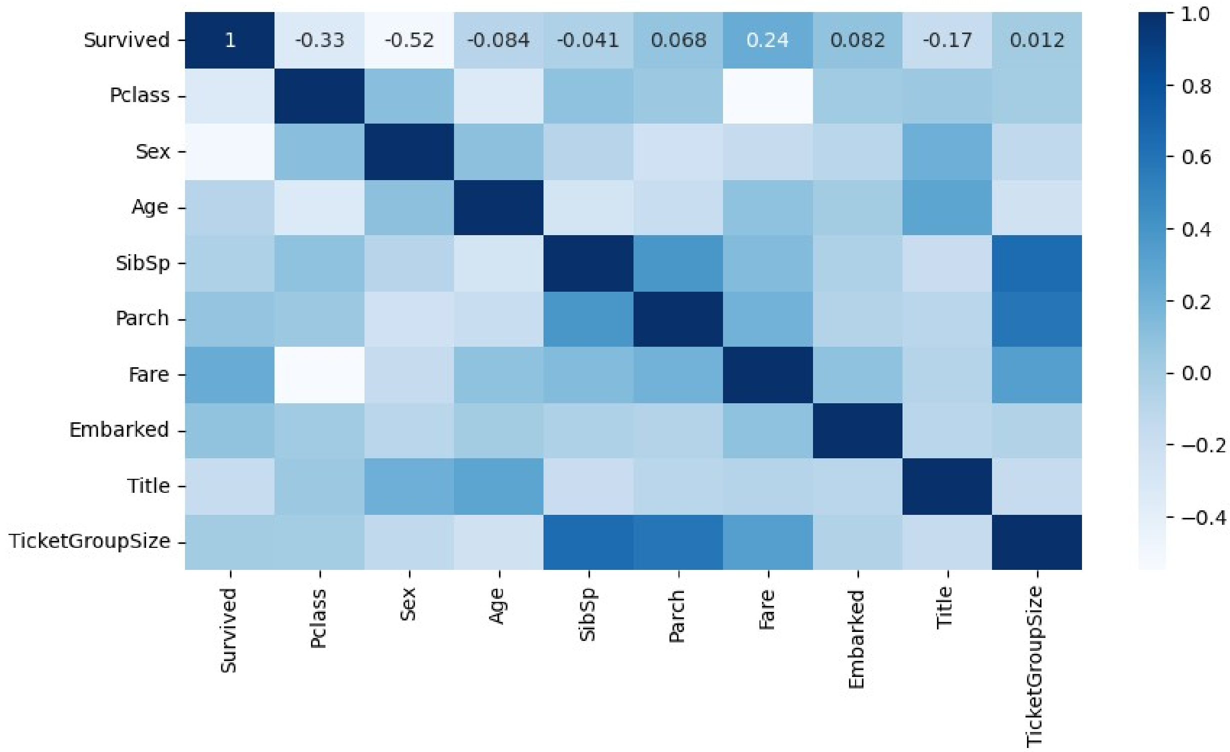
```
                Survived    Pclass       Sex       Age     SibSp
Parch  \
Survived        1.000000 -0.331985 -0.517096 -0.084048 -0.041388
0.068392
Pclass         -0.331985  1.000000  0.112139 -0.332997  0.091895
0.038266
Sex            -0.517096  0.112139  1.000000  0.098667 -0.087514 -
0.232233
Age            -0.084048 -0.332997  0.098667  1.000000 -0.268670 -
0.185226
SibSp          -0.041388  0.091895 -0.087514 -0.268670  1.000000
```

```
0.383767
Parch            0.068392  0.038266 -0.232233 -0.185226  0.383767
1.000000
Fare             0.243415 -0.549401 -0.164645  0.086809  0.137428
0.195339
Embarked         0.081819  0.018286 -0.096421  0.010531 -0.042889 -
0.072401
Title           -0.168193  0.042396  0.217822  0.286130 -0.188157 -
0.104197
TicketGroupSize  0.012190  0.006370 -0.137022 -0.243106  0.638109
0.588351

                     Fare  Embarked     Title  TicketGroupSize
Survived         0.243415  0.081819 -0.168193         0.012190
Pclass          -0.549401  0.018286  0.042396         0.006370
Sex             -0.164645 -0.096421  0.217822        -0.137022
Age              0.086809  0.010531  0.286130        -0.243106
SibSp            0.137428 -0.042889 -0.188157         0.638109
Parch            0.195339 -0.072401 -0.104197         0.588351
Fare             1.000000  0.086560 -0.077789         0.332740
Embarked         0.086560  1.000000 -0.100287        -0.060594
Title           -0.077789 -0.100287  1.000000        -0.164793
TicketGroupSize  0.332740 -0.060594 -0.164793         1.000000
```

```python
plt.figure(figsize=(10,5))
sns.heatmap(data.corr(),annot=True , cmap='Blues')
```

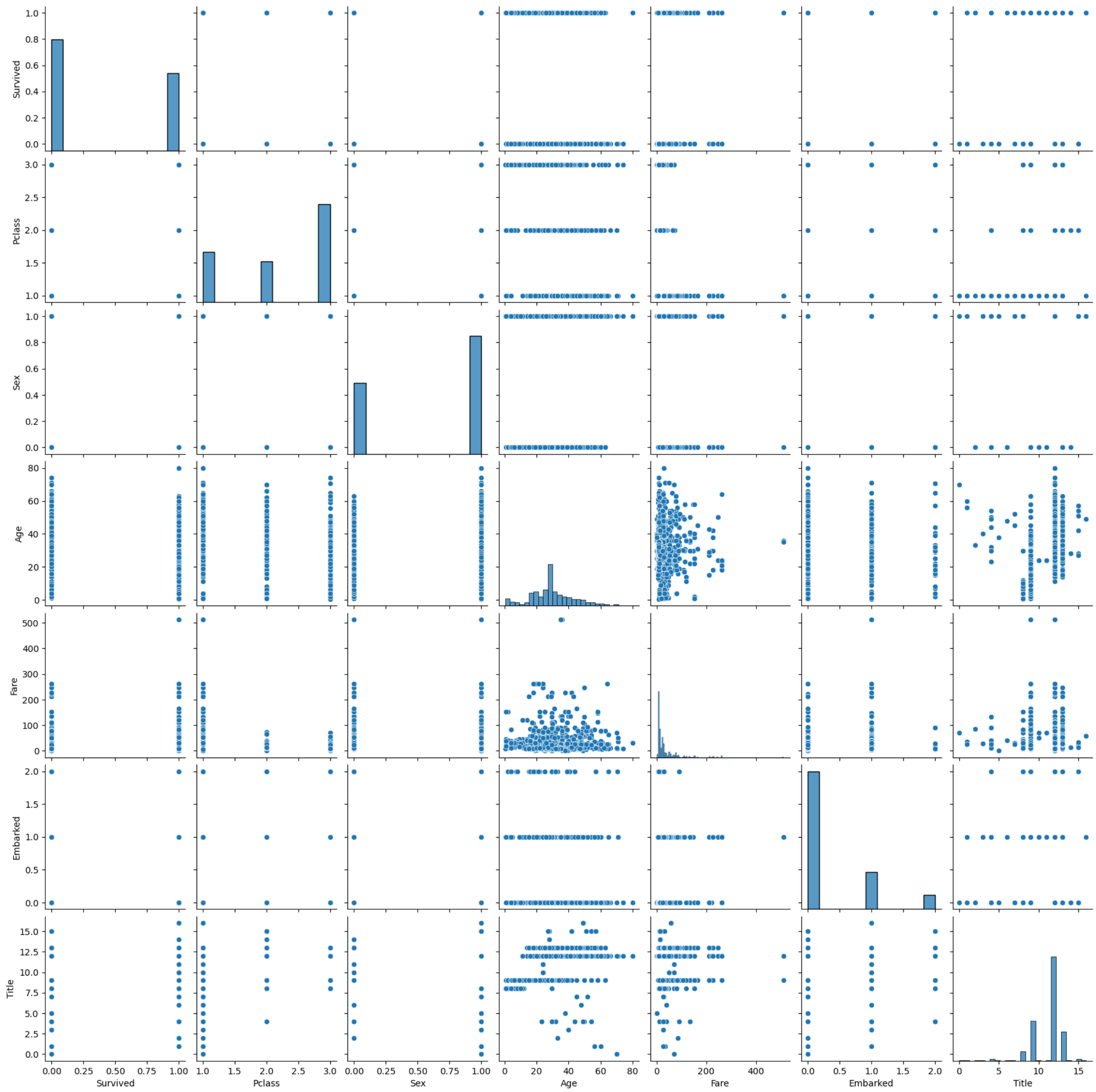
```
<Axes: >
```



I found that ['TicketGroupSize', 'Parch','SibSp'] not affect

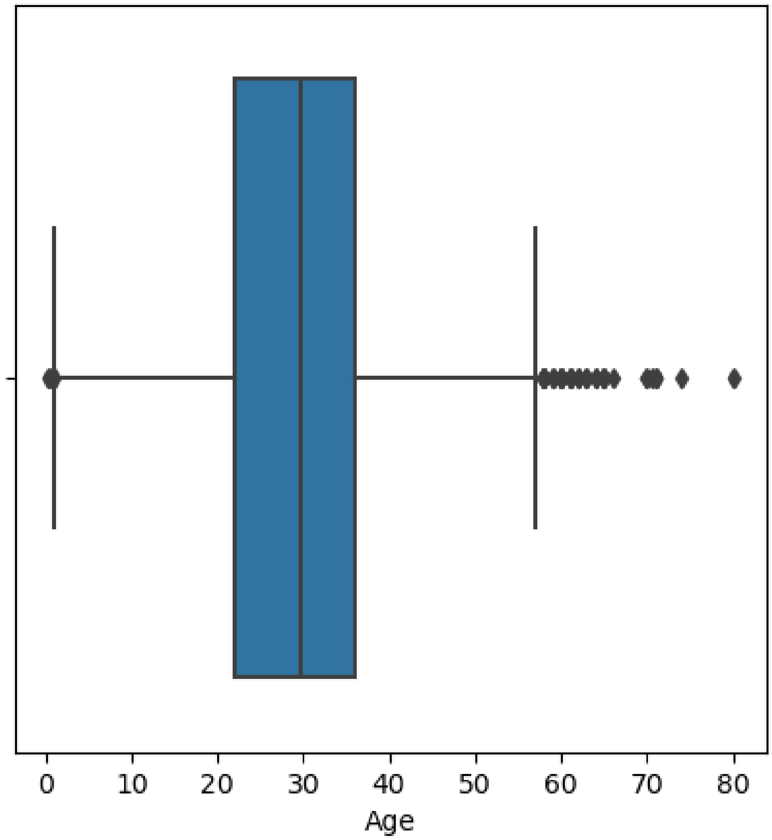
```
data.drop(['TicketGroupSize', 'Parch','SibSp'],inplace=True,axis=1)
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x292bce58a90>
```



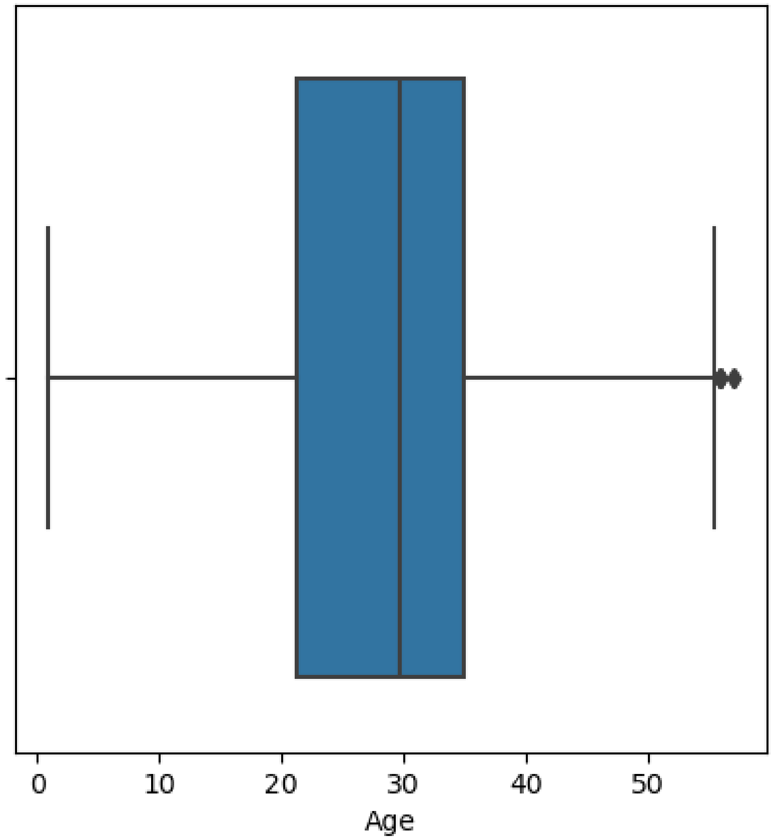
Check Outlires

```
plt.figure(figsize=(5,5))
sns.boxplot(x=data['Age'])
plt.show()
```




```
def remove_outlires(data,cols):
    for col in cols:
        Q1=data[col].quantile(0.25)
        Q3=data[col].quantile(0.75)
        IQR= Q3-Q1
        lower_bound=Q1-(1.5*IQR)
        upper_bound=Q3+(1.5*IQR)
        data=data[(data[col]>=lower_bound) & (data[col]<=upper_bound)]
    return data
cols=['Age']
data=remove_outlires(data,cols)

plt.figure(figsize=(5,5))
sns.boxplot(x=data['Age'])
plt.show()
```




```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 754 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  754 non-null    int64  
 1   Pclass    754 non-null    int64  
 2   Sex       754 non-null    int64  
 3   Age       754 non-null    float64
 4   Fare      754 non-null    float64
 5   Embarked  754 non-null    int64  
 6   Title     754 non-null    int32  
dtypes: float64(2), int32(1), int64(4)
memory usage: 44.2 KB
```

Split the data

```
X=data.drop('Survived',axis=1)
y=data['Survived']
```

```
from imblearn.over_sampling import SMOTE
from collections import Counter
smote = SMOTE(sampling_strategy='minority', k_neighbors=5)
# apply SMOTE to the data
X_resampled, y_resampled = smote.fit_resample(X, y)
# print the new class distribution
print('Resampled class distribution:', Counter(y_resampled))
```

```
Resampled class distribution: Counter({0: 444, 1: 444})
```

```
x_train,x_test,y_train,y_test=train_test_split(X_resampled,y_resampled
,test_size=0.2,random_state=33,shuffle=True)
```

```
from sklearn.preprocessing import StandardScaler
scler=StandardScaler()
x_train=scler.fit_transform(x_train)
x_test=scler.transform(x_test)
```

Build Model

```
modelName = ['Logistic Regression','Decision Tree' , 'Random Forest' ,
'KNN','SVC']
model =
[LogisticRegression(solver='saga',C=1,max_iter=100,multi_class='multin
omial',tol=0.001),DecisionTreeClassifier(),
RandomForestClassifier(n_estimators=10,criterion='entropy',random_stat
e=44,max_features= 'auto') , KNeighborsClassifier(n_neighbors=4) ,
SVC(C=0.3,kernel='linear',gamma=0.1)]

def train_Model(modelName,model,x_train,x_test,y_train,y_test):
    result={}
    for name , model in zip(modelName,model):
        model.fit(x_train,y_train)
        train_score=model.score(x_train,y_train)
        y_pred = model.predict(x_test)
        ac=accuracy_score(y_pred,y_test)
        precision=precision_score(y_pred,y_test,average="micro")
        recall=recall_score(y_pred,y_test,average='micro')
        f1score=f1_score(y_pred,y_test,average='micro')
        result[name]= [train_score,ac,precision,recall,f1score]
    return result

result = train_Model(modelName,model,x_train,x_test,y_train,y_test)
result =
pd.DataFrame(result,index=['train_score','Accuracy','precision','recal
l','f1score'])
result
```

```
                 Logistic Regression  Decision Tree  Random Forest
KNN  \
train_score                 0.783099       0.984507       0.976056
0.854930
Accuracy                    0.803371       0.780899       0.837079
0.803371
precision                   0.803371       0.780899       0.837079
0.803371
recall                      0.803371       0.780899       0.837079
0.803371
f1score                     0.803371       0.780899       0.837079
0.803371

                  SVC
train_score  0.785915
Accuracy     0.792135
precision    0.792135
recall       0.792135
f1score      0.792135
```

with cross validation

```
from sklearn.model_selection import cross_val_score
modelrandom =
RandomForestClassifier(n_estimators=100,criterion='entropy',random_sta
te=44,max_features= 'auto')
socre =cross_val_score(modelrandom,X,y ,cv=5)
socre

array([0.74834437, 0.74834437, 0.79470199, 0.7615894 , 0.8       ])

socre.mean()

0.7705960264900662
```

Build Bagging

```
from sklearn.ensemble import BaggingClassifier
svmodel= SVC(random_state=42,C=0.3,kernel='linear',gamma=0.1)
dtmodel= DecisionTreeClassifier(random_state=42)
lrmodel= LogisticRegression()
knmodel= KNeighborsClassifier(n_neighbors=5)

bagg_SV =
BaggingClassifier( base_estimator=svmodel ,bootstrap=True,n_estimators
=100)
bagg_DS =
BaggingClassifier( base_estimator=dtmodel ,bootstrap=True,n_estimators
```

```
=5)
bagg_KN =
BaggingClassifier( base_estimator=knmodel ,bootstrap=True,n_estimators
=20,random_state=42)
bagg_LR =
BaggingClassifier( base_estimator=lrmodel ,bootstrap=True,n_estimators
=100)

BaggModelName = ['Bagg Logistic Regression','Bagg Decision Tree' ,
'Bagg KNN','Bagg SVC']
BaggModel = [bagg_LR,bagg_DS, bagg_KN, bagg_SV]

result =
train_Model(BaggModelName,BaggModel,x_train,x_test,y_train,y_test)
result =
pd.DataFrame(result,index=['train_score','Accuracy','precision','recal
l','f1score'])
result
```

	Bagg Logistic Regression	Bagg Decision Tree	Bagg KNN	Bagg SVC
train_score	0.778873	0.957746	0.860563	0.785915
Accuracy	0.797753	0.808989	0.837079	0.792135
precision	0.797753	0.808989	0.837079	0.792135
recall	0.797753	0.808989	0.837079	0.792135
f1score	0.797753	0.808989	0.837079	0.792135

NN

```
x_train.shape
```

```
(710, 6)
```

```
model = keras.Sequential(
    [
        layers.Dense(32, activation="relu",input_shape=(6,)),
        layers.Dropout(0.4),
        layers.Dense(128, activation="relu"),
        # layers.Dropout(0.5),
        layers.BatchNormalization(),
        layers.Dense(64, activation="relu"),
        # layers.Dropout(0.4),
        layers.Dense(32, activation="relu"),
        layers.BatchNormalization(),
```

```
        layers.Dense(16, activation="relu"),
        layers.Dropout(0.2),
        layers.Dense(8, activation="relu"),
        layers.BatchNormalization(),
        layers.Dense(1, activation="sigmoid"),
    ]
)
model.summary()
```

```
Model: "sequential_2"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ dense_12 (Dense)                     │ (None, 32)                   │
224 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dropout_4 (Dropout)                  │ (None, 32)                   │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_13 (Dense)                     │ (None, 128)                  │
4,224 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ batch_normalization_6                │ (None, 128)                  │
512 │
│ (BatchNormalization)                 │                              │
│
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_14 (Dense)                     │ (None, 64)                   │
8,256 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_15 (Dense)                     │ (None, 32)                   │
2,080 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ batch_normalization_7                │ (None, 32)                   │
128 │
│ (BatchNormalization)                 │                              │
│
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_16 (Dense)                     │ (None, 16)                   │
```

```
528 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dropout_5 (Dropout)                  │ (None, 16)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dense_17 (Dense)                     │ (None, 8)                   │
136 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ batch_normalization_8                │ (None, 8)                   │
32 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dense_18 (Dense)                     │ (None, 1)                   │
9 │
└──────────────────────────────────────┴─────────────────────────────┘
─────────────────┘

 Total params: 16,129 (63.00 KB)

 Trainable params: 15,793 (61.69 KB)

 Non-trainable params: 336 (1.31 KB)

batch_size = 64
epochs = 100

model.compile(loss="binary_crossentropy", optimizer="adam",
metrics=["accuracy"])

history=model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=0.1)

Epoch 1/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 19s 139ms/step - accuracy: 0.4642 - loss:
0.7999 - val_accuracy: 0.5070 - val_loss: 0.6948
Epoch 2/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6019 - loss:
0.6648 - val_accuracy: 0.5775 - val_loss: 0.6860
Epoch 3/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6611 - loss:
0.6236 - val_accuracy: 0.6338 - val_loss: 0.6807
Epoch 4/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6923 - loss:
0.5794 - val_accuracy: 0.6479 - val_loss: 0.6760
Epoch 5/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7250 - loss:
```

```
0.5571 - val_accuracy: 0.6620 - val_loss: 0.6687
Epoch 6/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7424 - loss:
0.5416 - val_accuracy: 0.6761 - val_loss: 0.6624
Epoch 7/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7246 - loss:
0.5563 - val_accuracy: 0.6761 - val_loss: 0.6541
Epoch 8/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7567 - loss:
0.5319 - val_accuracy: 0.7042 - val_loss: 0.6458
Epoch 9/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7131 - loss:
0.5363 - val_accuracy: 0.7042 - val_loss: 0.6406
Epoch 10/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7097 - loss:
0.5456 - val_accuracy: 0.6620 - val_loss: 0.6356
Epoch 11/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7586 - loss:
0.5073 - val_accuracy: 0.6761 - val_loss: 0.6288
Epoch 12/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7874 - loss:
0.4743 - val_accuracy: 0.7042 - val_loss: 0.6184
Epoch 13/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7639 - loss:
0.5072 - val_accuracy: 0.7042 - val_loss: 0.6082
Epoch 14/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7631 - loss:
0.5074 - val_accuracy: 0.6901 - val_loss: 0.6002
Epoch 15/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7444 - loss:
0.4992 - val_accuracy: 0.6901 - val_loss: 0.5970
Epoch 16/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7567 - loss:
0.5014 - val_accuracy: 0.7042 - val_loss: 0.5896
Epoch 17/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7539 - loss:
0.4972 - val_accuracy: 0.7042 - val_loss: 0.5821
Epoch 18/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7659 - loss:
0.4991 - val_accuracy: 0.7324 - val_loss: 0.5773
Epoch 19/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7687 - loss:
0.4813 - val_accuracy: 0.7183 - val_loss: 0.5774
Epoch 20/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7439 - loss:
0.5286 - val_accuracy: 0.7183 - val_loss: 0.5689
Epoch 21/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7528 - loss:
0.4932 - val_accuracy: 0.7183 - val_loss: 0.5575
```

```
Epoch 22/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7972 - loss: 0.4597 - val_accuracy: 0.7183 - val_loss: 0.5511
Epoch 23/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7791 - loss: 0.4888 - val_accuracy: 0.7324 - val_loss: 0.5488
Epoch 24/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7728 - loss: 0.4690 - val_accuracy: 0.7324 - val_loss: 0.5438
Epoch 25/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7992 - loss: 0.4525 - val_accuracy: 0.7324 - val_loss: 0.5462
Epoch 26/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7704 - loss: 0.4701 - val_accuracy: 0.7465 - val_loss: 0.5553
Epoch 27/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7997 - loss: 0.4527 - val_accuracy: 0.7324 - val_loss: 0.5616
Epoch 28/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7995 - loss: 0.4330 - val_accuracy: 0.7324 - val_loss: 0.5520
Epoch 29/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7701 - loss: 0.4899 - val_accuracy: 0.7465 - val_loss: 0.5432
Epoch 30/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.7674 - loss: 0.4782 - val_accuracy: 0.7465 - val_loss: 0.5311
Epoch 31/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.7730 - loss: 0.4878 - val_accuracy: 0.7465 - val_loss: 0.5296
Epoch 32/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7734 - loss: 0.4995 - val_accuracy: 0.7465 - val_loss: 0.5306
Epoch 33/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8048 - loss: 0.4514 - val_accuracy: 0.7465 - val_loss: 0.5274
Epoch 34/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7814 - loss: 0.4529 - val_accuracy: 0.7606 - val_loss: 0.5248
Epoch 35/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8046 - loss: 0.4638 - val_accuracy: 0.7465 - val_loss: 0.5263
Epoch 36/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7809 - loss: 0.4804 - val_accuracy: 0.7465 - val_loss: 0.5254
Epoch 37/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7944 - loss: 0.4555 - val_accuracy: 0.7606 - val_loss: 0.5186
Epoch 38/100
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8070 - loss:
0.4544 - val_accuracy: 0.7324 - val_loss: 0.5226
Epoch 39/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7817 - loss:
0.4822 - val_accuracy: 0.7324 - val_loss: 0.5245
Epoch 40/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8103 - loss:
0.4504 - val_accuracy: 0.7465 - val_loss: 0.5264
Epoch 41/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8035 - loss:
0.4513 - val_accuracy: 0.7606 - val_loss: 0.5226
Epoch 42/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7915 - loss:
0.4537 - val_accuracy: 0.7183 - val_loss: 0.5316
Epoch 43/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7711 - loss:
0.4536 - val_accuracy: 0.7183 - val_loss: 0.5331
Epoch 44/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7913 - loss:
0.4493 - val_accuracy: 0.7324 - val_loss: 0.5320
Epoch 45/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8119 - loss:
0.4322 - val_accuracy: 0.6901 - val_loss: 0.5426
Epoch 46/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8006 - loss:
0.4568 - val_accuracy: 0.7042 - val_loss: 0.5482
Epoch 47/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7904 - loss:
0.4401 - val_accuracy: 0.7465 - val_loss: 0.5472
Epoch 48/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7819 - loss:
0.4519 - val_accuracy: 0.7465 - val_loss: 0.5536
Epoch 49/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7938 - loss:
0.4571 - val_accuracy: 0.7465 - val_loss: 0.5542
Epoch 50/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.7723 - loss:
0.4623 - val_accuracy: 0.7465 - val_loss: 0.5533
Epoch 51/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.7800 - loss:
0.4700 - val_accuracy: 0.7465 - val_loss: 0.5486
Epoch 52/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7635 - loss:
0.4501 - val_accuracy: 0.7465 - val_loss: 0.5527
Epoch 53/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8003 - loss:
0.4562 - val_accuracy: 0.7465 - val_loss: 0.5578
Epoch 54/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7923 - loss:
```

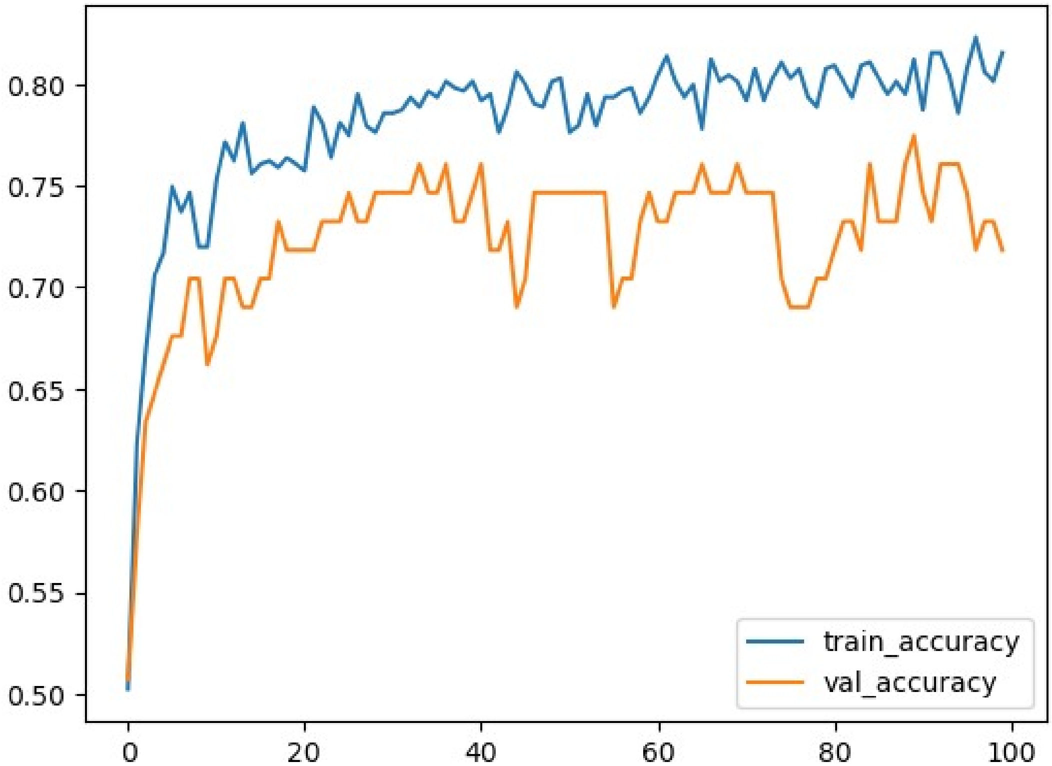
```
0.4338 - val_accuracy: 0.7465 - val_loss: 0.5510
Epoch 55/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7705 - loss:
0.4956 - val_accuracy: 0.7465 - val_loss: 0.5431
Epoch 56/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7708 - loss:
0.4976 - val_accuracy: 0.6901 - val_loss: 0.5491
Epoch 57/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8100 - loss:
0.4403 - val_accuracy: 0.7042 - val_loss: 0.5547
Epoch 58/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7909 - loss:
0.4295 - val_accuracy: 0.7042 - val_loss: 0.5544
Epoch 59/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8088 - loss:
0.4228 - val_accuracy: 0.7324 - val_loss: 0.5514
Epoch 60/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8173 - loss:
0.4204 - val_accuracy: 0.7465 - val_loss: 0.5509
Epoch 61/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7961 - loss:
0.4354 - val_accuracy: 0.7324 - val_loss: 0.5613
Epoch 62/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8074 - loss:
0.4233 - val_accuracy: 0.7324 - val_loss: 0.5569
Epoch 63/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7822 - loss:
0.4679 - val_accuracy: 0.7465 - val_loss: 0.5540
Epoch 64/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7751 - loss:
0.4527 - val_accuracy: 0.7465 - val_loss: 0.5516
Epoch 65/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7814 - loss:
0.4833 - val_accuracy: 0.7465 - val_loss: 0.5526
Epoch 66/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7714 - loss:
0.4512 - val_accuracy: 0.7606 - val_loss: 0.5544
Epoch 67/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8198 - loss:
0.4172 - val_accuracy: 0.7465 - val_loss: 0.5523
Epoch 68/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7966 - loss:
0.4193 - val_accuracy: 0.7465 - val_loss: 0.5505
Epoch 69/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.8059 - loss:
0.4182 - val_accuracy: 0.7465 - val_loss: 0.5494
Epoch 70/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.8056 - loss:
0.4509 - val_accuracy: 0.7606 - val_loss: 0.5478
```

```
Epoch 71/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7857 - loss:
0.4417 - val_accuracy: 0.7465 - val_loss: 0.5406
Epoch 72/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8205 - loss:
0.4050 - val_accuracy: 0.7465 - val_loss: 0.5427
Epoch 73/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7874 - loss:
0.4428 - val_accuracy: 0.7465 - val_loss: 0.5415
Epoch 74/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7982 - loss:
0.4440 - val_accuracy: 0.7465 - val_loss: 0.5403
Epoch 75/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8132 - loss:
0.4266 - val_accuracy: 0.7042 - val_loss: 0.5459
Epoch 76/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8020 - loss:
0.4275 - val_accuracy: 0.6901 - val_loss: 0.5543
Epoch 77/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8174 - loss:
0.4201 - val_accuracy: 0.6901 - val_loss: 0.5558
Epoch 78/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8151 - loss:
0.4066 - val_accuracy: 0.6901 - val_loss: 0.5584
Epoch 79/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7882 - loss:
0.4506 - val_accuracy: 0.7042 - val_loss: 0.5695
Epoch 80/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8002 - loss:
0.4434 - val_accuracy: 0.7042 - val_loss: 0.5659
Epoch 81/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7946 - loss:
0.4144 - val_accuracy: 0.7183 - val_loss: 0.5460
Epoch 82/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8158 - loss:
0.4247 - val_accuracy: 0.7324 - val_loss: 0.5386
Epoch 83/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7993 - loss:
0.4227 - val_accuracy: 0.7324 - val_loss: 0.5360
Epoch 84/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8181 - loss:
0.3827 - val_accuracy: 0.7183 - val_loss: 0.5482
Epoch 85/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7815 - loss:
0.4425 - val_accuracy: 0.7606 - val_loss: 0.5544
Epoch 86/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8038 - loss:
0.4383 - val_accuracy: 0.7324 - val_loss: 0.5743
Epoch 87/100
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7917 - loss:
0.4251 - val_accuracy: 0.7324 - val_loss: 0.5818
Epoch 88/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7992 - loss:
0.4591 - val_accuracy: 0.7324 - val_loss: 0.5769
Epoch 89/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7937 - loss:
0.4369 - val_accuracy: 0.7606 - val_loss: 0.5715
Epoch 90/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8191 - loss:
0.4255 - val_accuracy: 0.7746 - val_loss: 0.5619
Epoch 91/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7782 - loss:
0.4528 - val_accuracy: 0.7465 - val_loss: 0.5698
Epoch 92/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8048 - loss:
0.4440 - val_accuracy: 0.7324 - val_loss: 0.5793
Epoch 93/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8216 - loss:
0.4034 - val_accuracy: 0.7606 - val_loss: 0.5895
Epoch 94/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8116 - loss:
0.4018 - val_accuracy: 0.7606 - val_loss: 0.5863
Epoch 95/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7744 - loss:
0.4472 - val_accuracy: 0.7606 - val_loss: 0.5819
Epoch 96/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.7923 - loss:
0.4354 - val_accuracy: 0.7465 - val_loss: 0.5854
Epoch 97/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.8321 - loss:
0.4097 - val_accuracy: 0.7183 - val_loss: 0.6059
Epoch 98/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8071 - loss:
0.4208 - val_accuracy: 0.7324 - val_loss: 0.6169
Epoch 99/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8133 - loss:
0.4147 - val_accuracy: 0.7324 - val_loss: 0.6140
Epoch 100/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8257 - loss:
0.4104 - val_accuracy: 0.7183 - val_loss: 0.6034
```

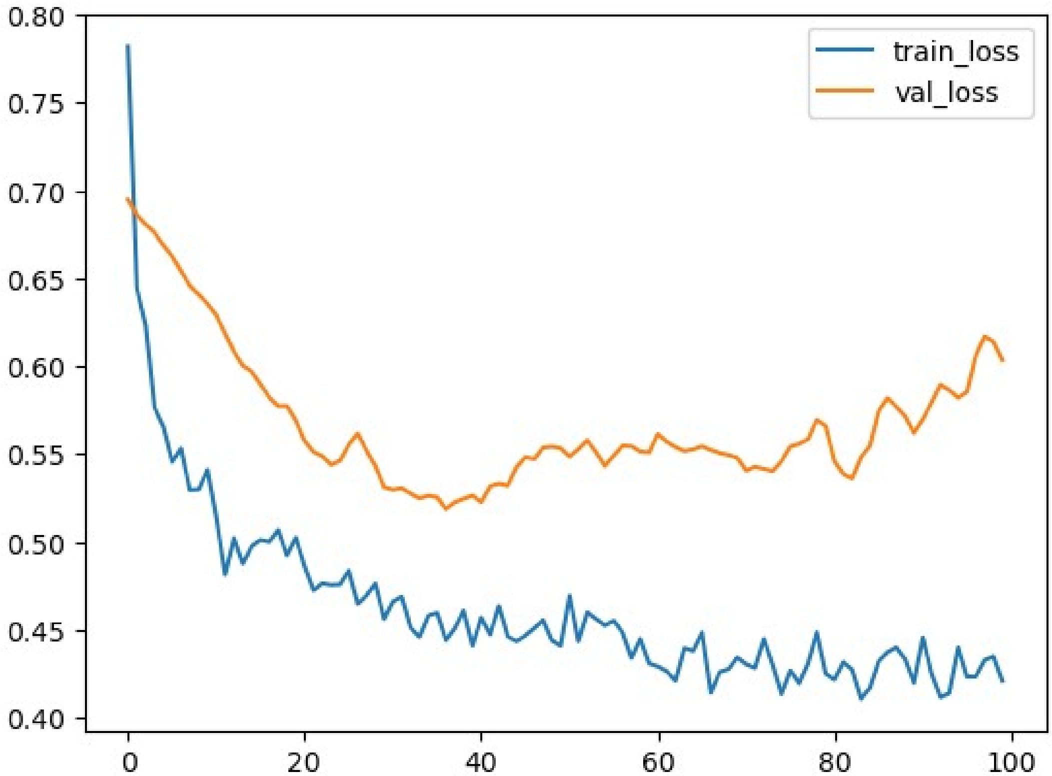
```
plt.plot(history.history['accuracy'],label='train_accuracy')
plt.plot(history.history['val_accuracy'],label='val_accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x292d3e3a1d0>
```




```
plt.plot(history.history['loss'],label='train_loss')
plt.plot(history.history['val_loss'],label='val_loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x292d3ef1f50>
```


0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
20
40
60
80
100
train_loss
val_loss