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

In [2]:
```
df = pd.read_csv('tested.csv')
```

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

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	Nal
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	Nal
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	Nal
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	Nal
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	Nal

In [4]:
```
df.shape
```

Out[4]:
```
(418, 12)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object 
 11  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [8]:
```
df.isnull().sum()/len(df)*100
```

Out[8]:
```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            20.574163
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.239234
Cabin          78.229665
Embarked        0.000000
dtype: float64
```

In [9]:
```
# Drop variable - Cabin because we have 78% missing data
df = df.drop(['PassengerId', 'Name','Ticket','Fare','Cabin'], axis=1)
```

In [10]:
```
df.isnull().sum() / len(df)*100
```

Out[10]:
```
Survived     0.000000
Pclass       0.000000
Sex          0.000000
Age         20.574163
SibSp        0.000000
Parch        0.000000
Embarked     0.000000
dtype: float64
```

In [11]:
```
# Age,fare and embarked also have missing values
```

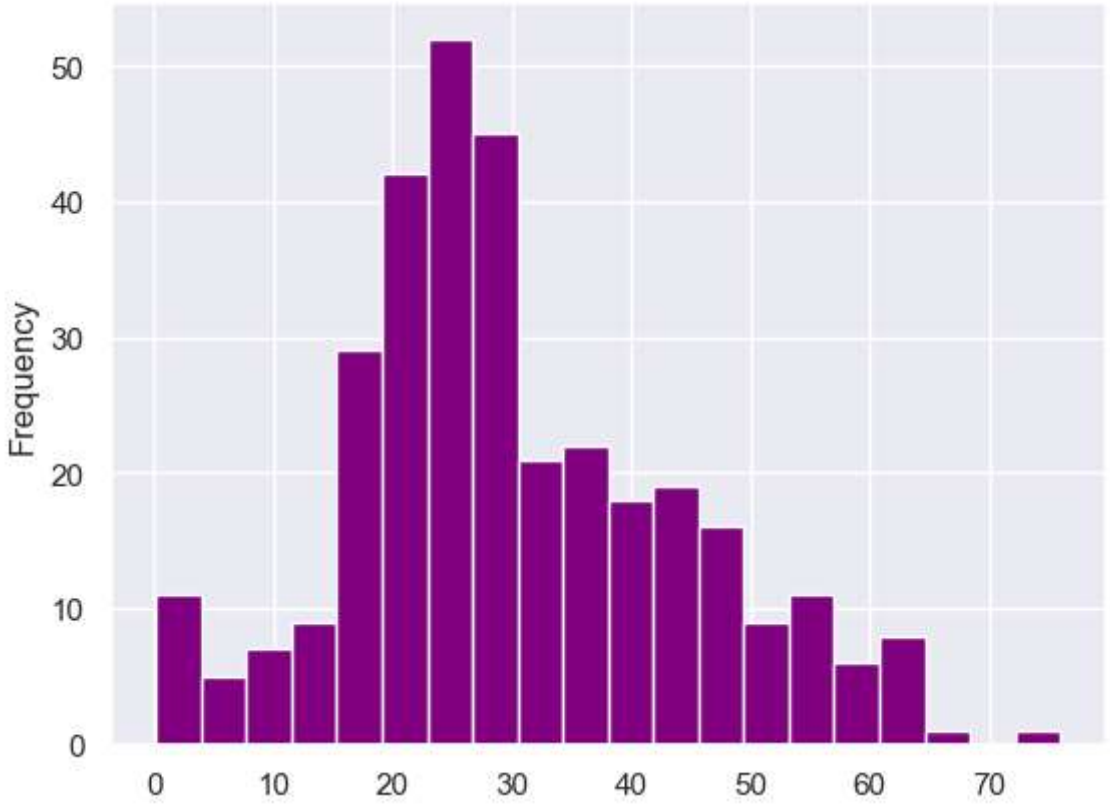
```
Age - conclusion
almost 20% of the values are missing
we have to check outlier and on that basis we have to decide imputation method
```

In [12]:
```
df['Age'].describe()
```

Out[12]:
```
count    332.000000
mean      30.272590
std       14.181209
min        0.170000
25%       21.000000
50%       27.000000
75%       39.000000
max       76.000000
Name: Age, dtype: float64
```

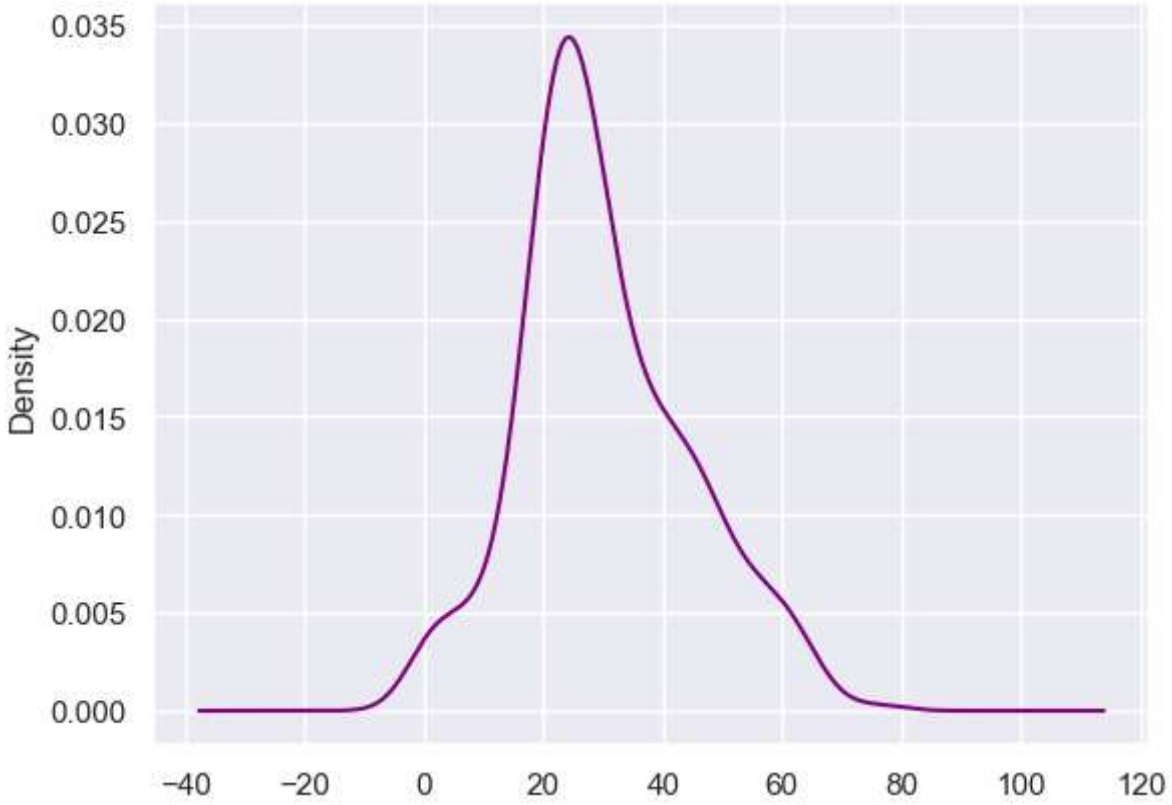
In [13]:
```
df['Age'].plot(kind='hist', bins=20,color='purple')
```

Out[13]:
```
<Axes: ylabel='Frequency'>
```



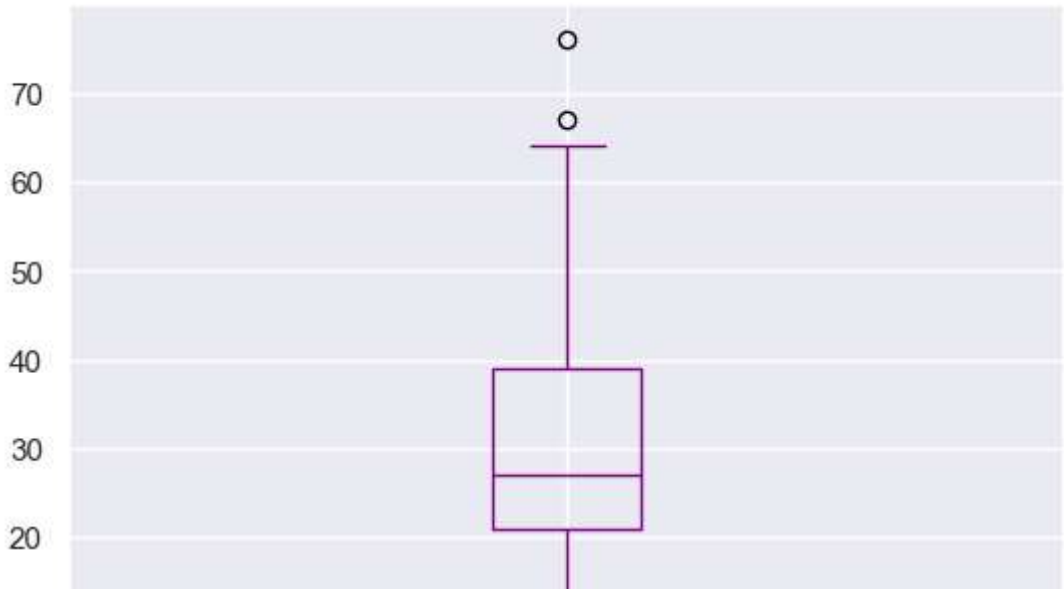
In [14]:
```
df['Age'].plot(kind='kde',color= 'purple')
```

Out[14]: <Axes: ylabel='Density'>



In [15]:
```
df['Age'].plot(kind='box',color='purple')
```

Out[15]: <Axes: >



In [16]:
```
df['Age'] = df['Age'].fillna(df['Age'].median())
```

In [17]:
```
df['Embarked'].value_counts()
```

Out[17]:
```
S    270
C    102
Q     46
Name: Embarked, dtype: int64
```

In [18]:
```
df['Embarked'] = df['Embarked'].fillna('S')
```

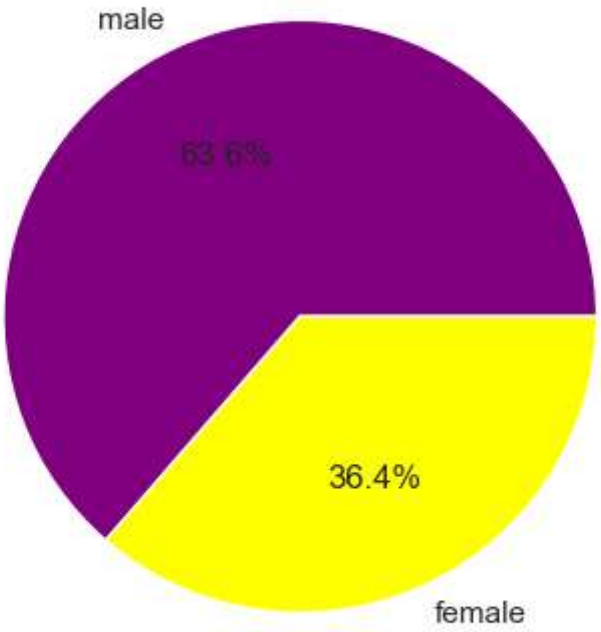
In [19]:
```
df.isnull().sum()
```

Out[19]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Embarked    0
dtype: int64
```

In [20]:
```
df['Sex'].value_counts()
```

Out[20]:
```
male      266
female    152
Name: Sex, dtype: int64
```

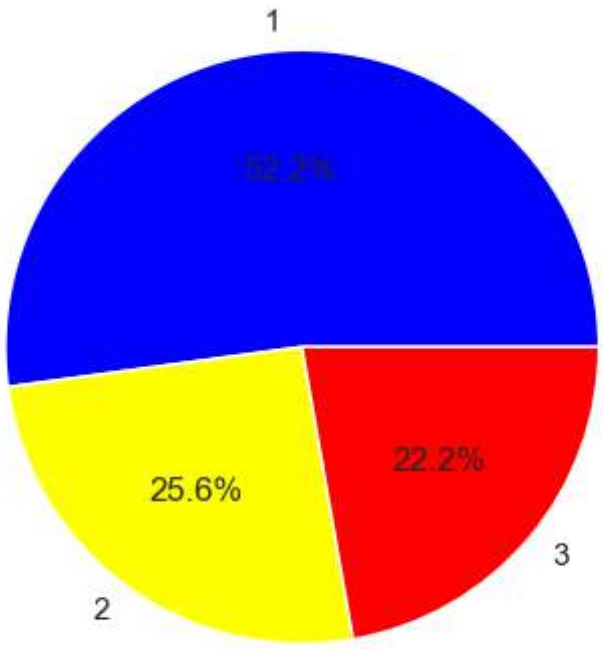
In [21]:
```
Sex = ['male','female']
quantity = [266,152]
plt.pie(quantity,labels=Sex,autopct='%0.1f%%',colors=['purple','yellow'])
plt.show()
```



In [22]:
```
df['Pclass'].value_counts()
```

Out[22]:
```
3    218
1    107
2     93
Name: Pclass, dtype: int64
```

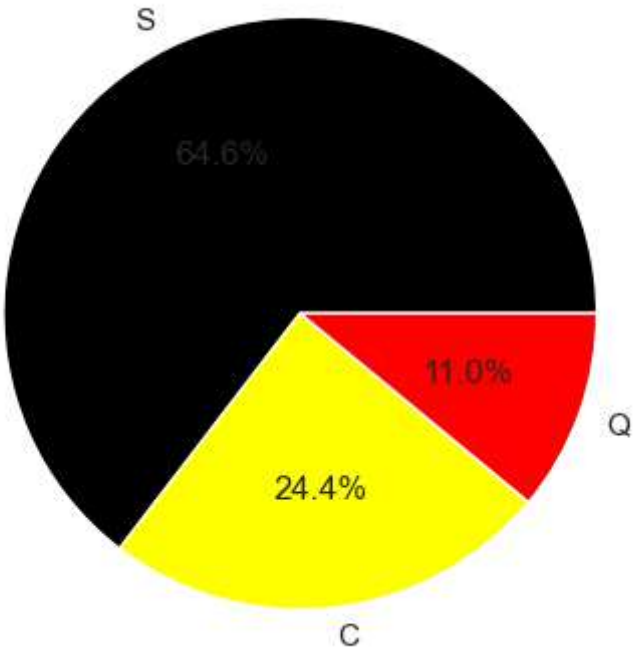
In [23]:
```
Pclass = [1,2,3]
quantity = [218,107,93]
plt.pie(quantity,labels=Pclass,autopct='%0.1f%%',colors=['blue','yellow','red'
plt.show()
```



In [24]:
```
df['Embarked'].value_counts()
```

Out[24]:
```
S    270
C    102
Q     46
Name: Embarked, dtype: int64
```

In [25]:
```
Embarked = ['S','C','Q']
quantity = [270,102,46]
plt.pie(quantity,labels=Embarked,autopct='%0.1f%%',colors=['black','yellow','r
plt.show()
```



label encoder

In [26]:
```
df['Sex'] = df['Sex'].astype('category')
df['Sex'] = df['Sex'].cat.codes
```

In [27]:
```
df = pd.get_dummies(df, columns=['Embarked'])
```

In [28]:
```
df.head()
```

Out[28]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked_C	Embarked_Q	Embarked_S
0	0	3	1	34.5	0	0	0	1	0
1	1	3	0	47.0	1	0	0	0	1
2	0	2	1	62.0	0	0	0	1	0
3	0	3	1	27.0	0	0	0	0	1
4	1	3	0	22.0	1	1	0	0	1

In [29]:
```
df['Pclass'].value_counts()
```

Out[29]:
```
3    218
1    107
2     93
Name: Pclass, dtype: int64
```

In [30]:
```
df = pd.get_dummies(df, columns=['Pclass'])
```

In [31]:
```
df.head()
```

Out[31]:

	Survived	Sex	Age	SibSp	Parch	Embarked_C	Embarked_Q	Embarked_S	Pclass_1	Pclass
0	0	1	34.5	0	0	0	1	0	0	
1	1	0	47.0	1	0	0	0	1	0	
2	0	1	62.0	0	0	0	1	0	0	
3	0	1	27.0	0	0	0	0	1	0	
4	1	0	22.0	1	1	0	0	1	0	

In [32]:
```
# dummy variables
df = df.drop(['Embarked_C', 'Pclass_1'], axis=1)
```

In [33]:
```
df.head()
```

Out[33]:

	Survived	Sex	Age	SibSp	Parch	Embarked_Q	Embarked_S	Pclass_2	Pclass_3
0	0	1	34.5	0	0	1	0	0	1
1	1	0	47.0	1	0	0	1	0	1
2	0	1	62.0	0	0	1	0	1	0
3	0	1	27.0	0	0	0	1	0	1
4	1	0	22.0	1	1	0	1	0	1

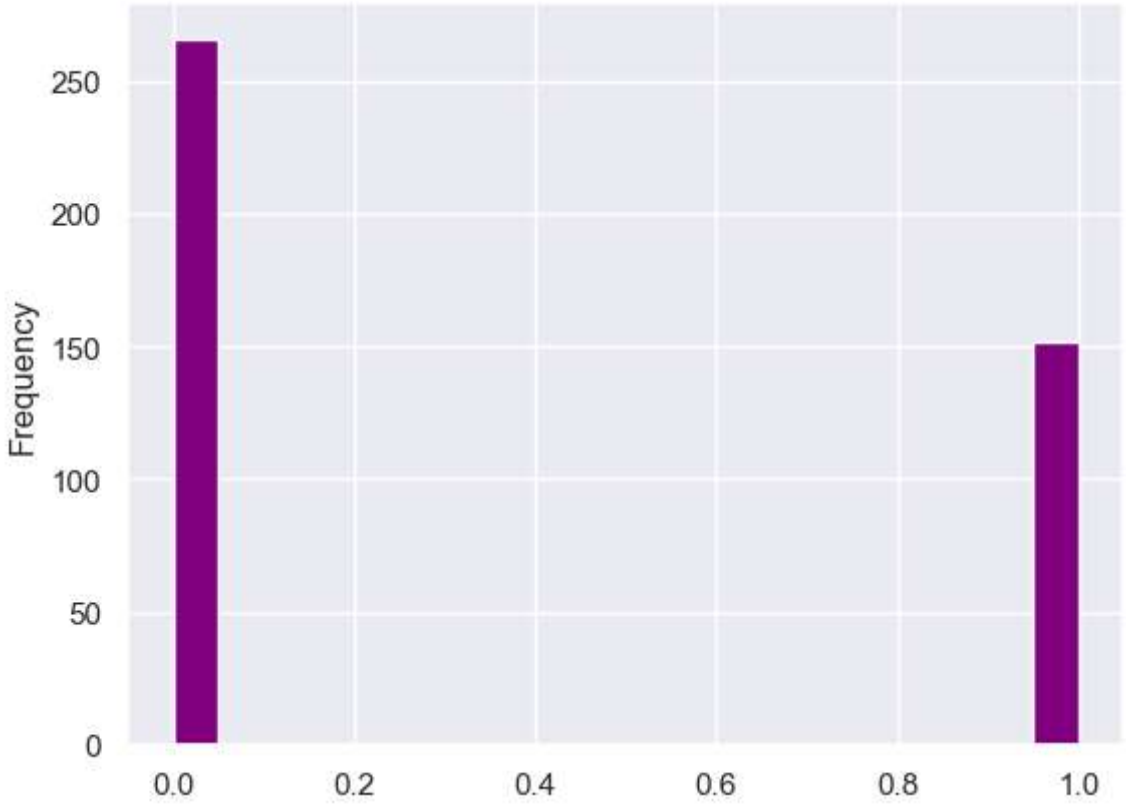
In [ ]:
```
```

In [34]:
```
df.columns
```

Out[34]:
```
Index(['Survived', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked_Q', 'Embarked_
S',
       'Pclass_2', 'Pclass_3'],
      dtype='object')
```

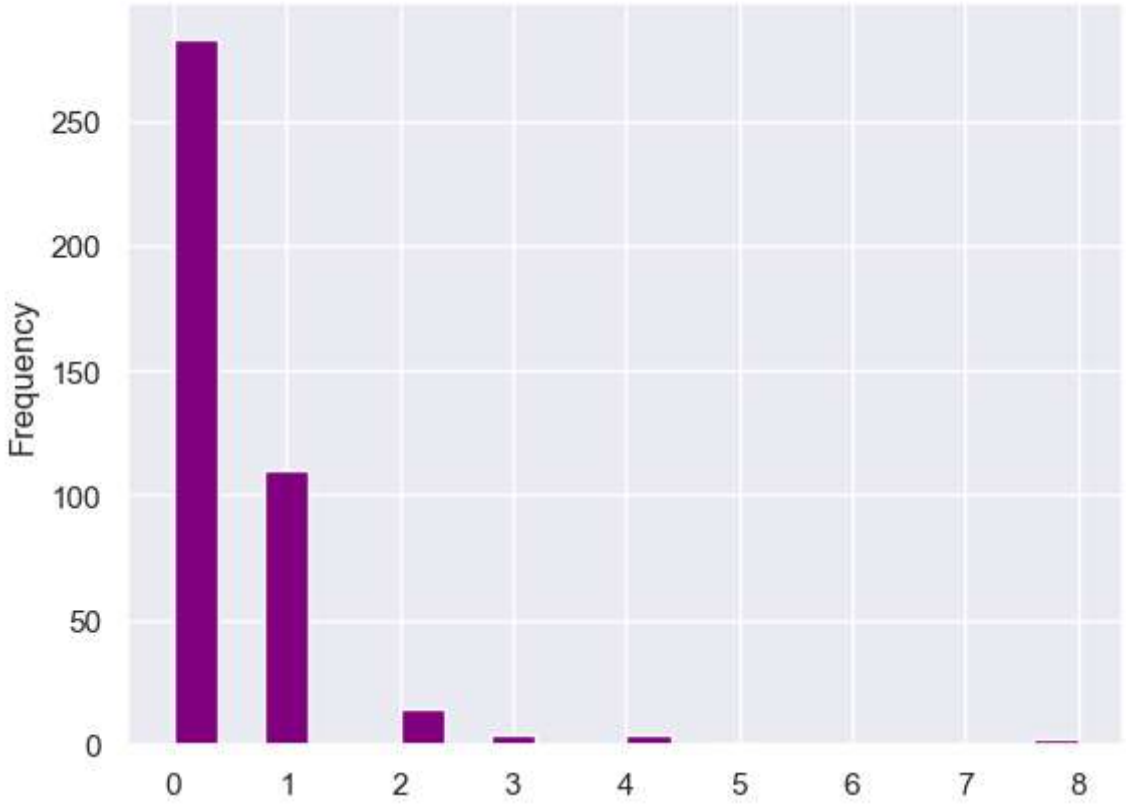
In [35]:
```
df['Survived'].plot(kind='hist', bins=20,color='purple')
```

Out[35]:
```
<Axes: ylabel='Frequency'>
```



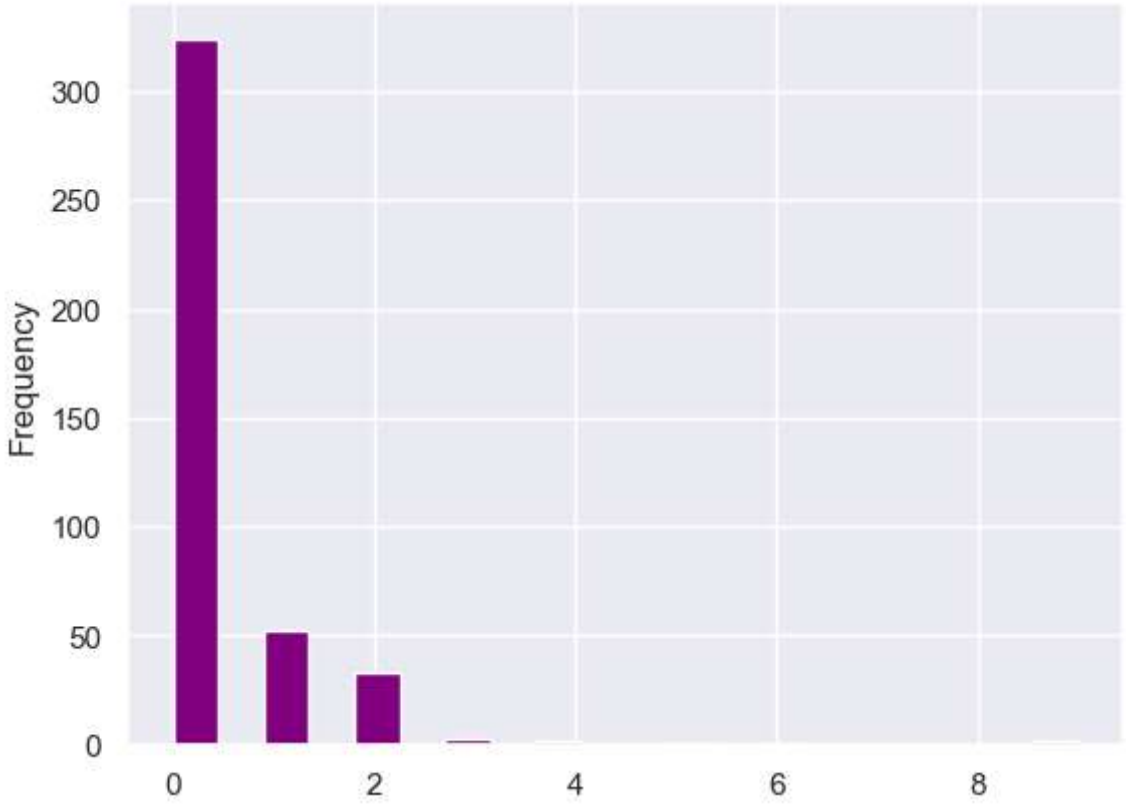
In [36]:
```
df['SibSp'].plot(kind='hist', bins=20,color='purple')
```

Out[36]:
```
<Axes: ylabel='Frequency'>
```



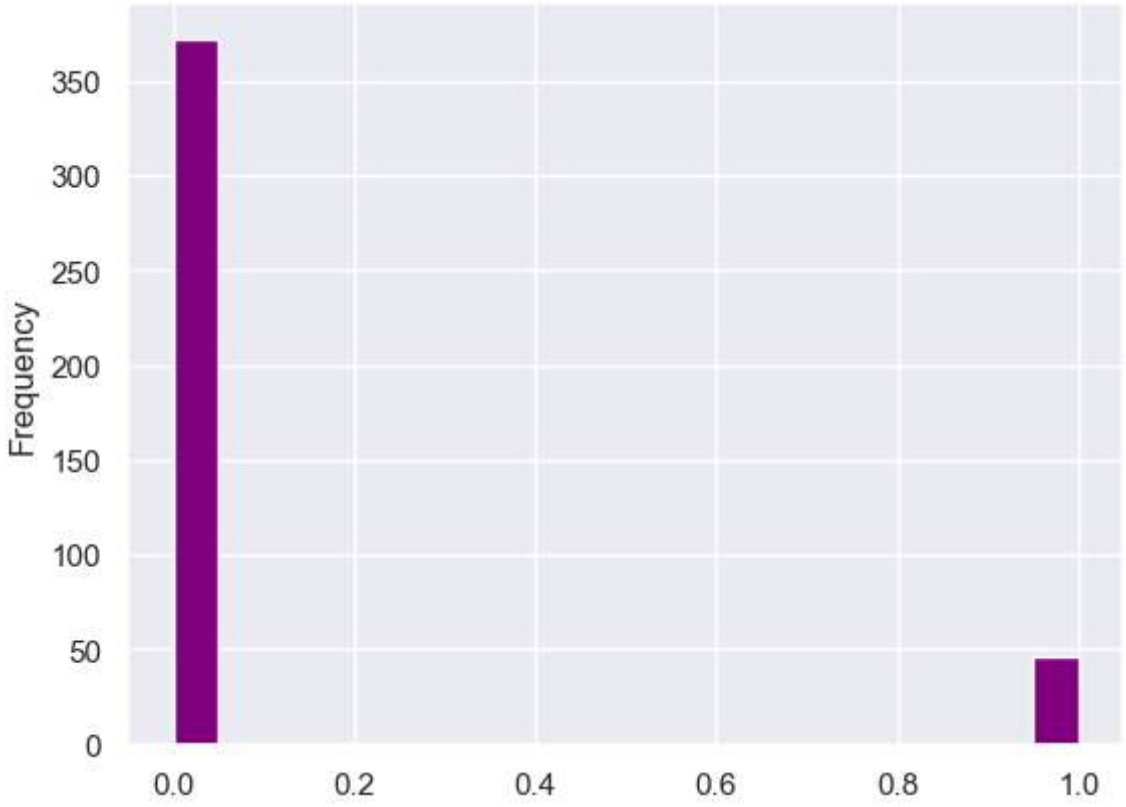
In [37]:
```
df['Parch'].plot(kind='hist', bins=20,color='purple')
```

Out[37]:
```
<Axes: ylabel='Frequency'>
```



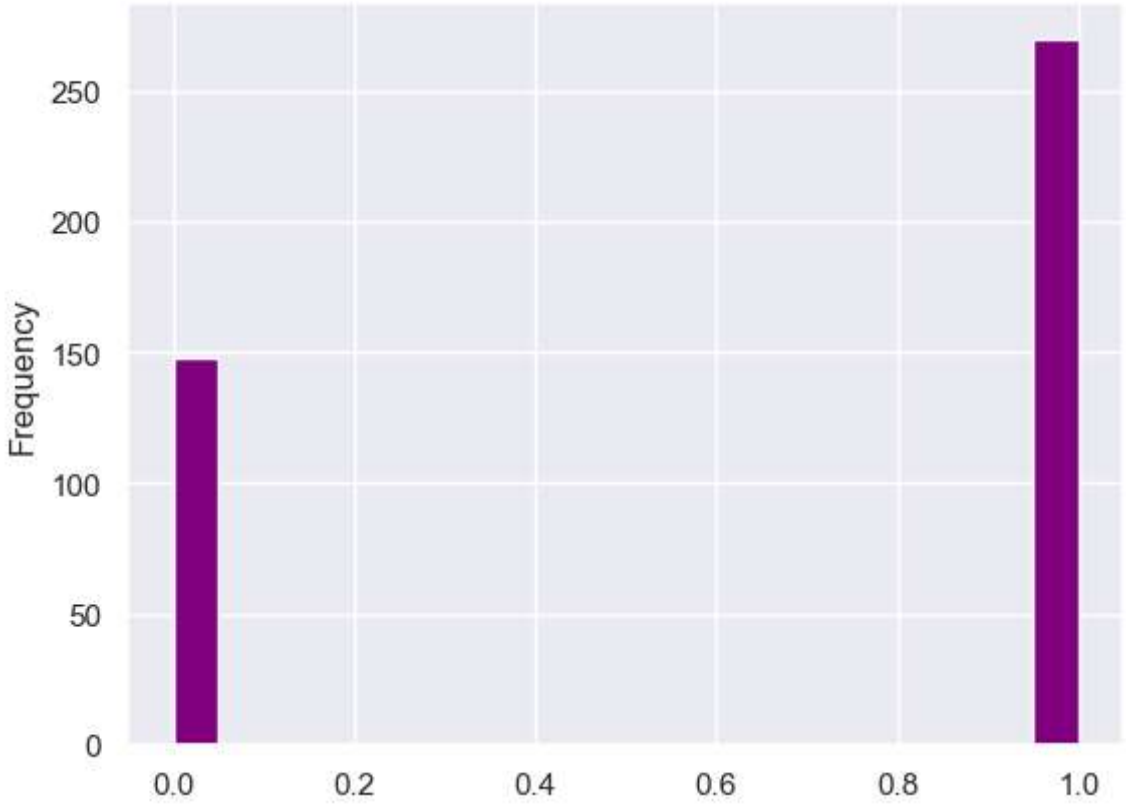
In [38]:
```
df['Embarked_Q'].plot(kind='hist', bins=20,color='purple')
```

Out[38]:
```
<Axes: ylabel='Frequency'>
```



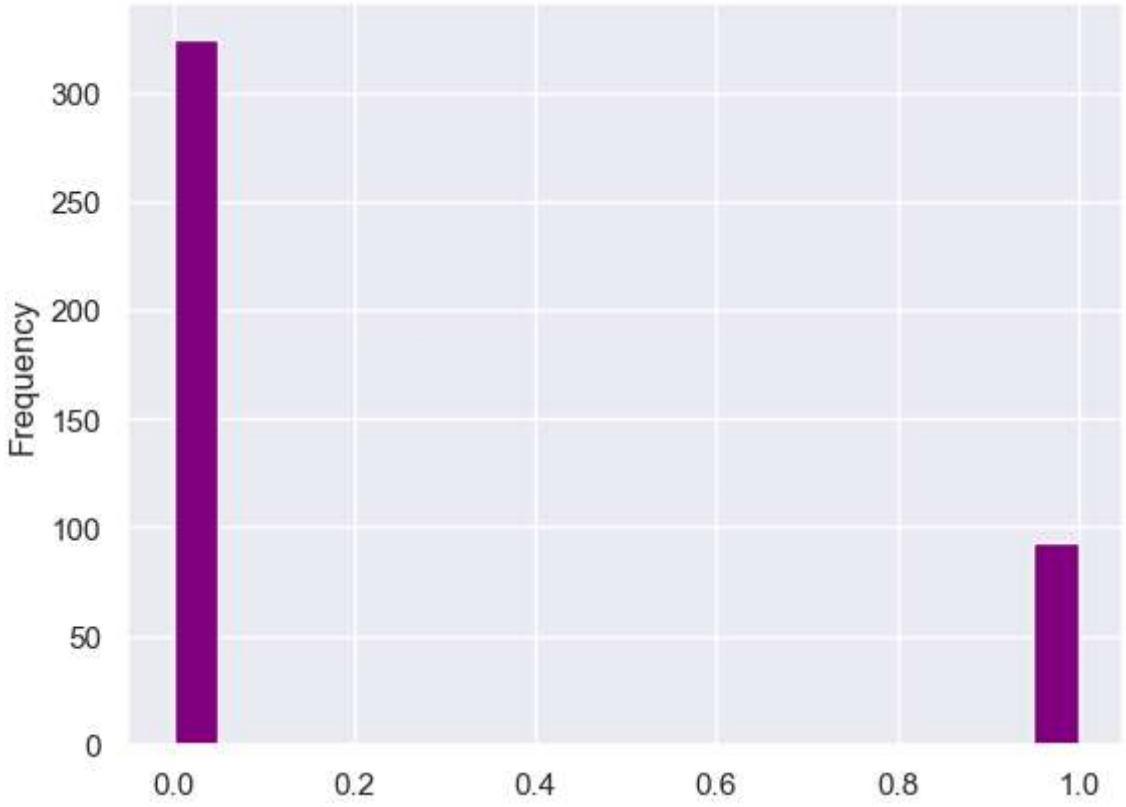
In [39]:
```
df['Embarked_S'].plot(kind='hist', bins=20,color='purple')
```

Out[39]:
```
<Axes: ylabel='Frequency'>
```



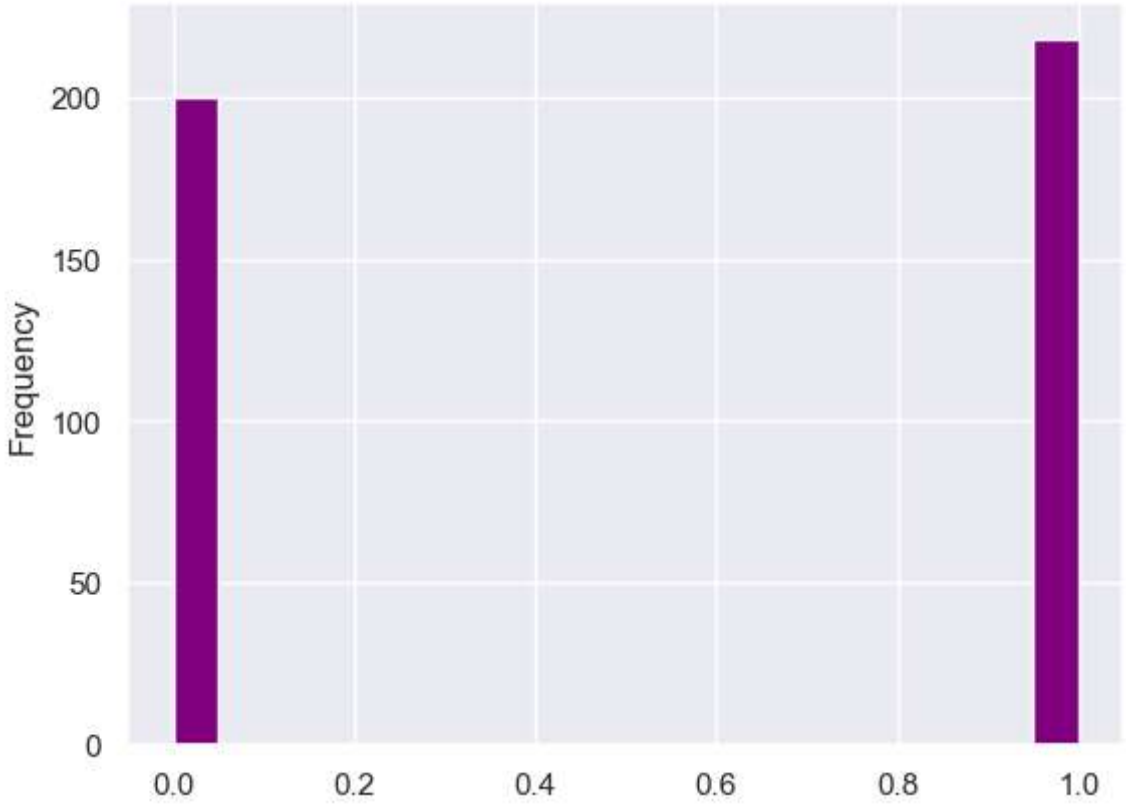
In [40]:
```
df['Pclass_2'].plot(kind='hist', bins=20,color='purple')
```

Out[40]:
```
<Axes: ylabel='Frequency'>
```



In [41]:
```
df['Pclass_3'].plot(kind='hist', bins=20,color='purple')
```

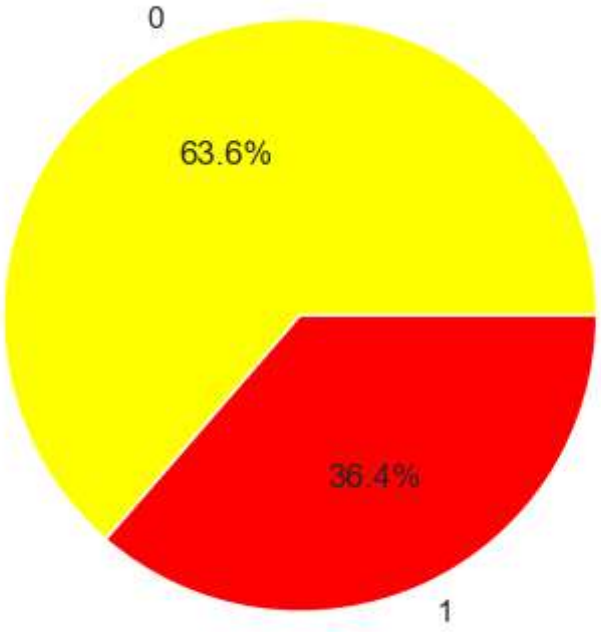
Out[41]:
```
<Axes: ylabel='Frequency'>
```



In [42]:
```
# Mandatory in classification problem - Imbalance check
df['Survived'].value_counts()
```

Out[42]:
```
0    266
1    152
Name: Survived, dtype: int64
```

In [43]:
```
Survived = [0,1]
quantity = [266,152]
plt.pie(quantity,labels=Survived,autopct='%0.1f%%',colors=['yellow','red'])
plt.show()
```



In [44]:
```
266/(266+152)
```

Out[44]:
```
0.6363636363636364
```

In [45]:
```
df.head()
```

Out[45]:

	Survived	Sex	Age	SibSp	Parch	Embarked_Q	Embarked_S	Pclass_2	Pclass_3
0	0	1	34.5	0	0	1	0	0	1
1	1	0	47.0	1	0	0	1	0	1
2	0	1	62.0	0	0	1	0	1	0
3	0	1	27.0	0	0	0	1	0	1
4	1	0	22.0	1	1	0	1	0	1

In [46]:
```
df.columns
```

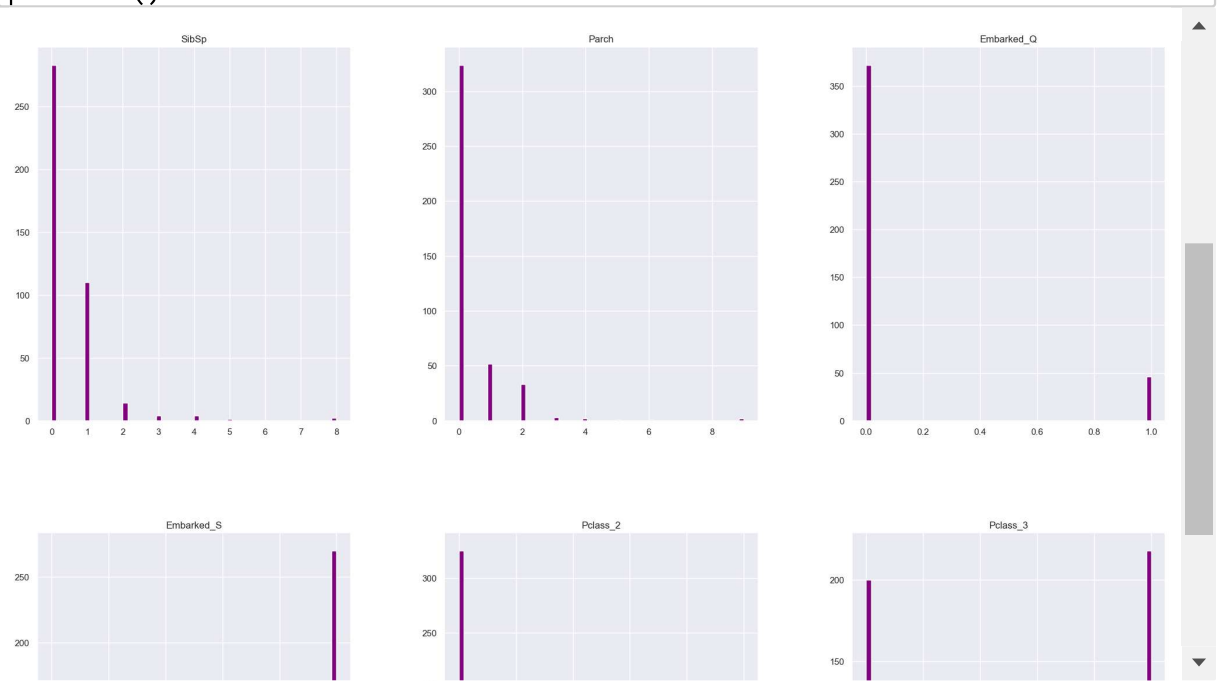
Out[46]:
```
Index(['Survived', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked_Q', 'Embarked_
S',
       'Pclass_2', 'Pclass_3'],
      dtype='object')
```

Find the unique values

In [47]:
```
for i in df.columns:
    print("**************************************************",i,
          "**************************************************")
    print()
    print(set(df[i].tolist()))
    print()
```

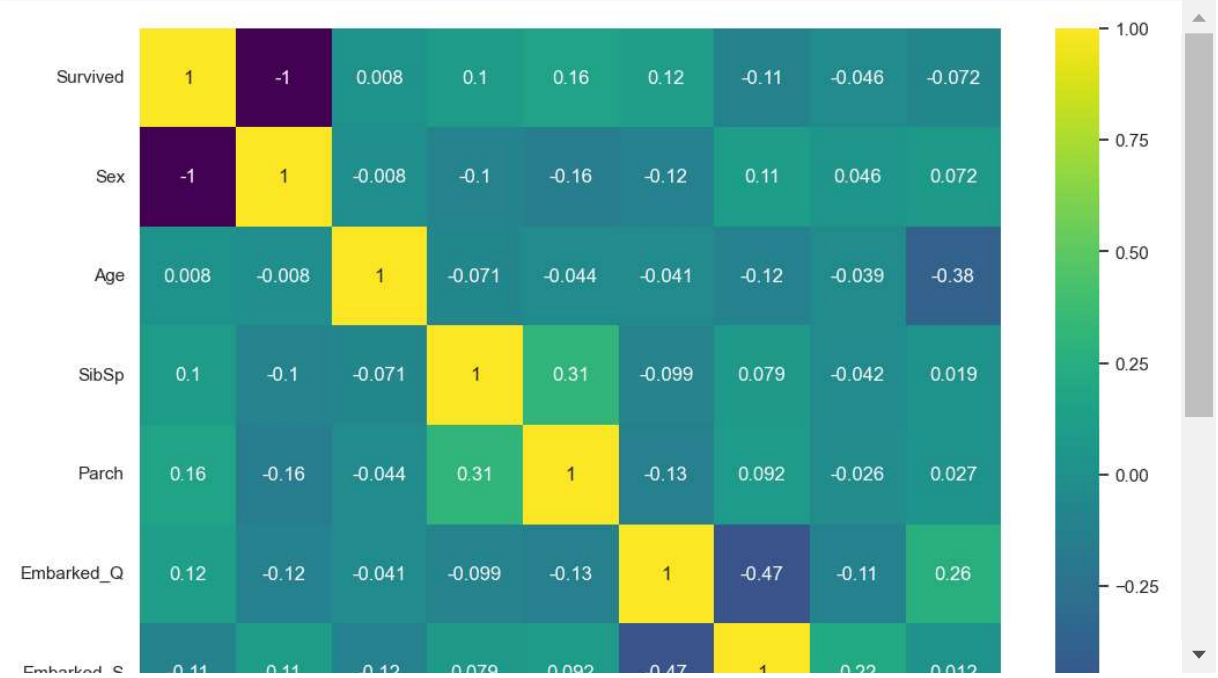
```
********************************************************** Survived **********
****************************************

{0, 1}

********************************************************** Sex ***************
************************************

{0, 1}

********************************************************** Age ***************
************************************

{0.75, 1.0, 2.0, 3.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.5, 12.0, 13.0, 14.0,
14.5, 16.0, 17.0, 18.0, 18.5, 20.0, 21.0, 22.0, 23.0, 24.0, 22.5, 26.5, 26.0,
28.5, 27.0, 25.0, 28.0, 30.0, 31.0, 32.0, 33.0, 34.5, 35.0, 36.0, 32.5, 39.0,
40.0, 41.0, 42.0, 43.0, 37.0, 45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 44.0, 53.0,
54.0, 55.0, 51.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0, 63.0, 60.5, 64.0, 67.0,
76.0, 15.0, 19.0, 29.0, 34.0, 36.5, 38.0, 0.17, 38.5, 40.5, 0.92, 0.83, 0.33}

********************************************************** SibSp *************
****************************************

{0, 1, 2, 3, 4, 5, 8}

********************************************************** Parch *************
****************************************

{0, 1, 2, 3, 4, 5, 6, 9}

********************************************************** Embarked_Q ********
**********************************************

{0, 1}

********************************************************** Embarked_S ********
**********************************************

{0, 1}

********************************************************** Pclass_2 **********
**********************************************

{0, 1}

********************************************************** Pclass_3 **********
**********************************************

{0, 1}
```

In [48]:
```
df.hist(bins=60,figsize=(25,30),color='purple')
plt.show()
```



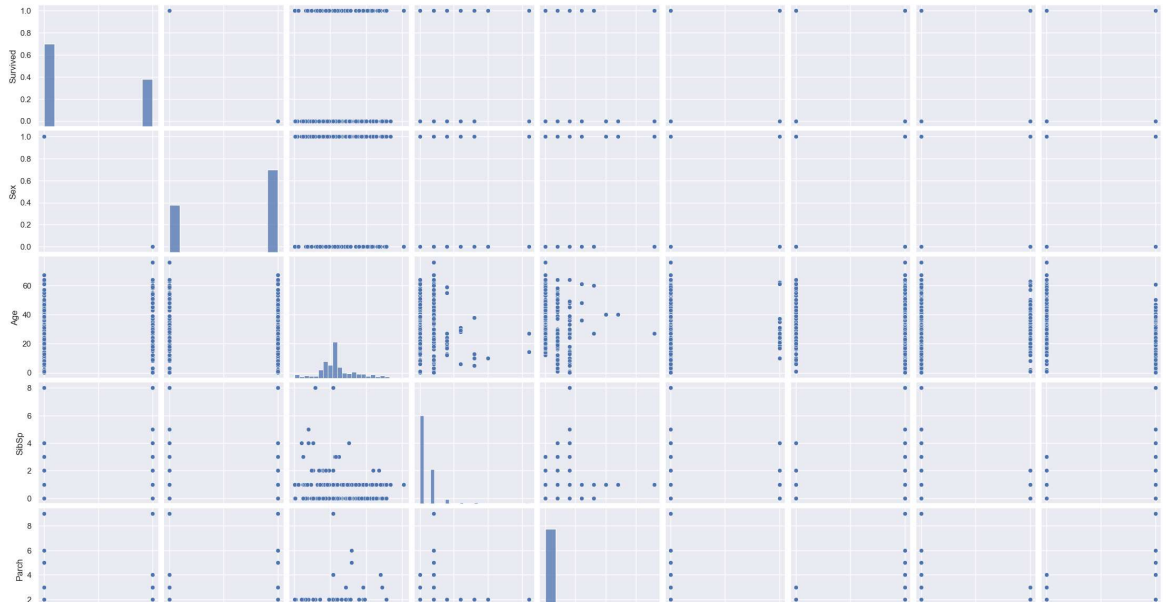
correlation

In [49]:
```
plt.figure(figsize=(12,10))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='viridis')
plt.show()
```



In [50]:
```
sns.pairplot(df)
```

Out[50]:
```
<seaborn.axisgrid.PairGrid at 0x25c67c8bb50>
```



split the data into dependent and independent variable

In [ ]:
```
```

In [51]:
```
x = df.iloc[:,1:]
y = df[['Survived']]
```

In [52]:
```
x.head()
```

Out[52]:

	Sex	Age	SibSp	Parch	Embarked_Q	Embarked_S	Pclass_2	Pclass_3
0	1	34.5	0	0	1	0	0	1
1	0	47.0	1	0	0	1	0	1
2	1	62.0	0	0	1	0	1	0
3	1	27.0	0	0	0	1	0	1
4	0	22.0	1	1	0	1	0	1

In [53]:
```
y.head()
```

Out[53]:

	Survived
0	0
1	1
2	0
3	0
4	1

In [54]:
```
# split the data into train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
                                                    random_state=101,stratify=
```

Applying all model together

In [55]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [56]:
```
# LogisticRegression
logistic = LogisticRegression()
lr = logistic.fit(x_train, y_train)
y_pred_lr = logistic.predict(x_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)

# DecisionTree
dtree = DecisionTreeClassifier()
dt = dtree.fit(x_train, y_train)
y_pred_dt = dtree.predict(x_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# RandomForest
rfmodel = RandomForestClassifier()
rf = rfmodel.fit(x_train, y_train)
y_pred_rf = rfmodel.predict(x_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# BaggingClassifier
bagg = BaggingClassifier()
bg = bagg.fit(x_train, y_train)
y_pred_bg = bagg.predict(x_test)
accuracy_bg = accuracy_score(y_test, y_pred_bg)

# AdaBoostClassifier
ada = AdaBoostClassifier()
ad = ada.fit(x_train, y_train)
y_pred_ad = ada.predict(x_test)
accuracy_ad = accuracy_score(y_test, y_pred_ad)

# GradientBoostingClassifier
gdb = GradientBoostingClassifier()
gd = gdb.fit(x_train, y_train)
y_pred_gd = gdb.predict(x_test)
accuracy_gd = accuracy_score(y_test, y_pred_gd)

# XGBClassifier = RF + GDBoosting - lambda - regularisation, gamma - autoprunn
xgb = XGBClassifier()
xg = xgb.fit(x_train, y_train)
y_pred_xg = xgb.predict(x_test)
accuracy_xg = accuracy_score(y_test, y_pred_xg)


# SVM
svc = SVC()
sv = svc.fit(x_train, y_train)
y_pred_sv = svc.predict(x_test)
accuracy_sv = accuracy_score(y_test, y_pred_sv)

# KNN
knn = KNeighborsClassifier()
kn = knn.fit(x_train, y_train)
y_pred_knn = knn.predict(x_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# GaussianNB
naive_gb = GaussianNB()
```

```
ngb = naive_gb.fit(x_train, y_train)
y_pred_ngb = naive_gb.predict(x_test)
accuracy_ngb = accuracy_score(y_test, y_pred_ngb)

# BernoulliNB
naive_bn = BernoulliNB()
nbr = naive_bn.fit(x_train, y_train)
y_pred_nbr = naive_bn.predict(x_test)
accuracy_nbr = accuracy_score(y_test, y_pred_nbr)
```

In [57]:
```
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import StackingClassifier
```

In [58]:
```
evc = VotingClassifier(estimators=[('lr',lr),('dt',dt),('rf', rf),('bg', bg),(
                                   ('gd', gd),('xg', xg),('sv', sv),('kn', kn),
                                   ('ngb', ngb),('nbr', nbr)], voting='hard')


model_evc = evc.fit(x_train, y_train)
pred_evc = evc.predict(x_test)
accuracy_evc = accuracy_score(y_test, pred_evc)
```

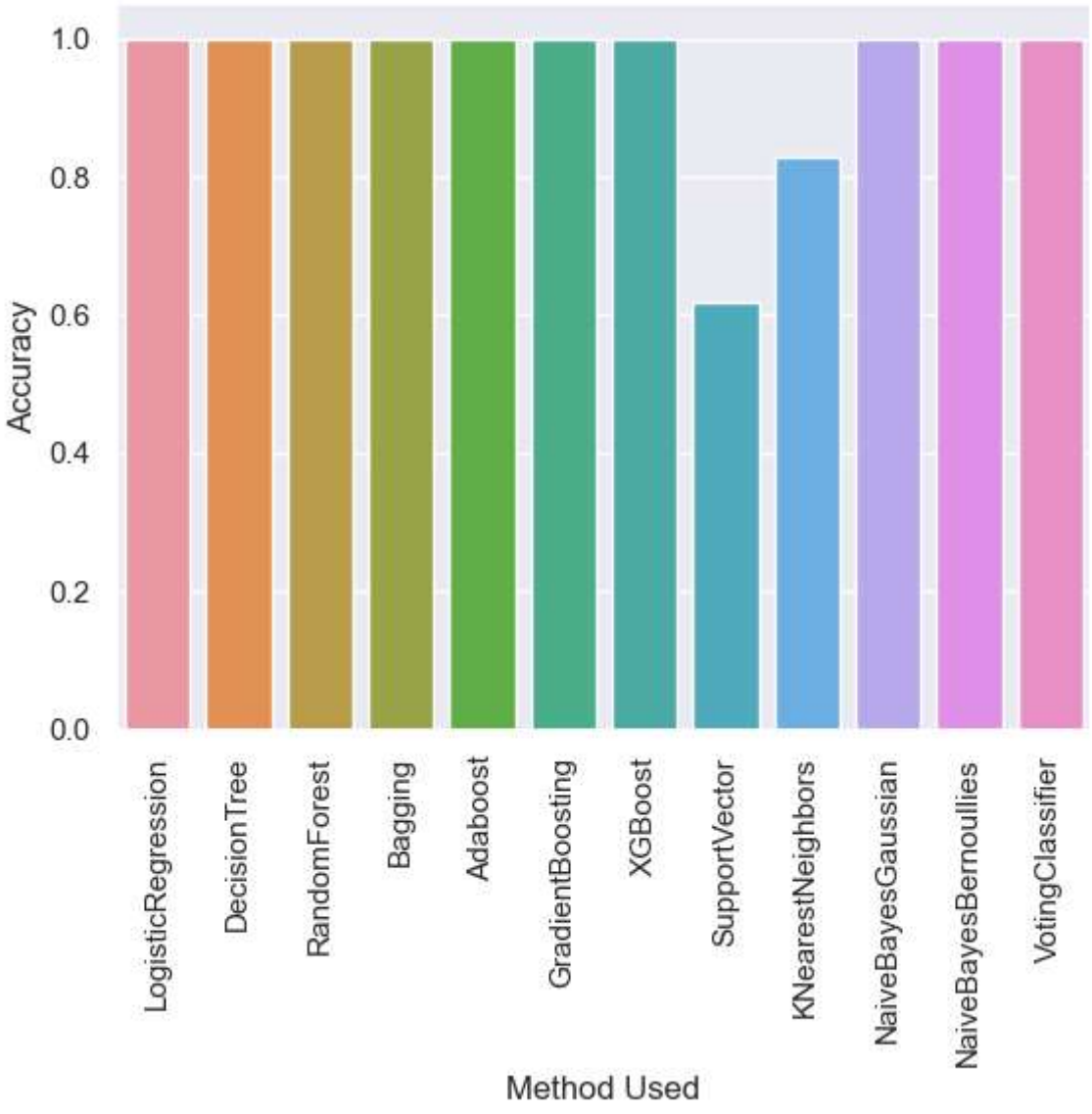
In [59]:
```
list1 = ['LogisticRegression','DecisionTree','RandomForest','Bagging','Adaboos
         'GradientBoosting', 'XGBoost','SupportVector','KNearestNeighbors',
         'NaiveBayesGaussian','NaiveBayesBernoullies','VotingClassifier']
```

In [60]:
```
list2 = [accuracy_lr, accuracy_dt, accuracy_rf, accuracy_bg,accuracy_ad, accur
         accuracy_xg, accuracy_sv, accuracy_knn, accuracy_ngb, accuracy_nbr, a
```

In [61]:
```
list3 = [logistic, dtree, rfmodel, bagg, ada, gdb, xgb, svc, knn, naive_gb,nai
```

In [62]:
```
final_accuracy = pd.DataFrame({'Method Used': list1, "Accuracy": list2})
print(final_accuracy)
charts = sns.barplot(x="Method Used", y = 'Accuracy', data=final_accuracy)
charts.set_xticklabels(charts.get_xticklabels(), rotation=90)
print(charts)
```

```
              Method Used  Accuracy
0      LogisticRegression  1.000000
1            DecisionTree  1.000000
2            RandomForest  1.000000
3                 Bagging  1.000000
4                Adaboost  1.000000
5        GradientBoosting  1.000000
6                 XGBoost  1.000000
7           SupportVector  0.619048
8       KNearestNeighbors  0.828571
9      NaiveBayesGaussian  1.000000
10  NaiveBayesBernoullies  1.000000
11       VotingClassifier  1.000000
Axes(0.125,0.11;0.775x0.77)
```



Conclution

The purpose of Titanic dataset is to use the existing features of passengers onboard Titanic predictors to predict their survival outcome, for 0 being dead and 1 being survived from the tragic ship crash.
Like we conclude from our analysis from the Titanic dataset, that more males would have survived if most of them belonged to upper-class or age is below 18.