sk-1-titanic-survival-prediction

July 8, 2024

[1]:
```
TASK 1 - TITANIC SURVIVAL PREDICTION

Use the Titanic dataset to build a model that predicts whether a passenger on␣
 ↪the Titanic survived or not.
This is a classic beginner project with readily available data.

The dataset typically used for this project contains information about␣
 ↪individual passengers,
such as their age, gender, ticket class, fare, cabin, and whether or not they␣
 ↪survived.
```

[ ]:
```
```

[78]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
train = pd.read_csv(r"C:\Users\divya\OneDrive\Documents\CodSoft␣
 ↪Internship\titanic_dataset.csv")
train.head()
```

[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0

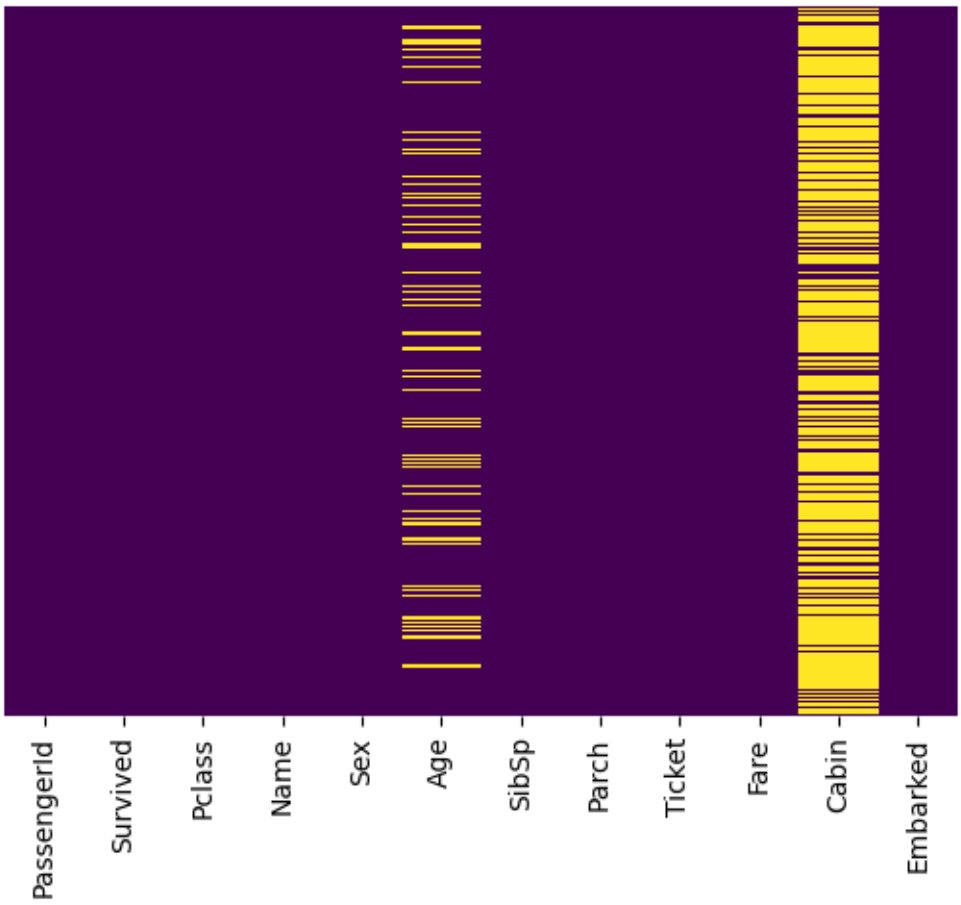
```
   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN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

[3]:
```
train.isnull()
```

[3]:
```
     PassengerId  Survived  Pclass   Name    Sex    Age  SibSp  Parch  Ticket  \
0          False     False   False  False  False  False  False  False   False
1          False     False   False  False  False  False  False  False   False
2          False     False   False  False  False  False  False  False   False
3          False     False   False  False  False  False  False  False   False
4          False     False   False  False  False  False  False  False   False
..           …         …     …      …      …      …      …      …
886        False     False   False  False  False  False  False  False   False
887        False     False   False  False  False  False  False  False   False
888        False     False   False  False  False   True  False  False   False
889        False     False   False  False  False  False  False  False   False
890        False     False   False  False  False  False  False  False   False

      Fare  Cabin  Embarked
0    False   True     False
1    False  False     False
2    False   True     False
3    False  False     False
4    False   True     False
..     …    …       …
886  False   True     False
887  False  False     False
888  False   True     False
889  False  False     False
890  False   True     False

[891 rows x 12 columns]
```

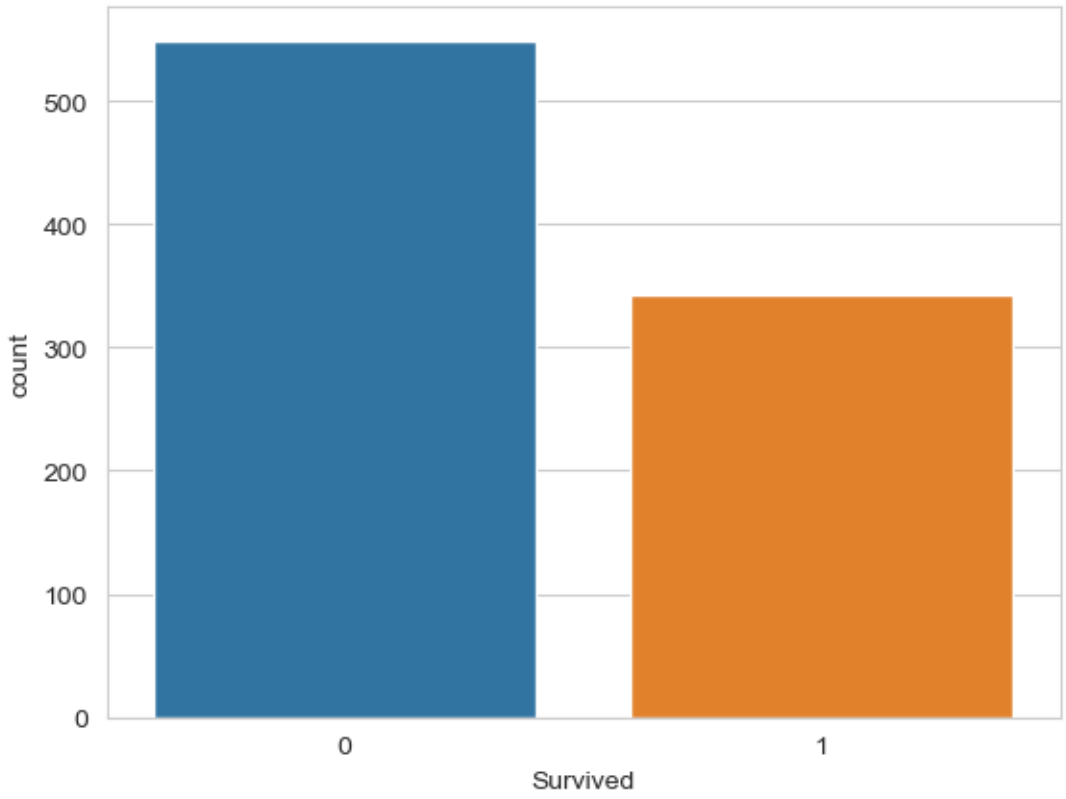
[4]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

[4]:
```
<Axes: >
```



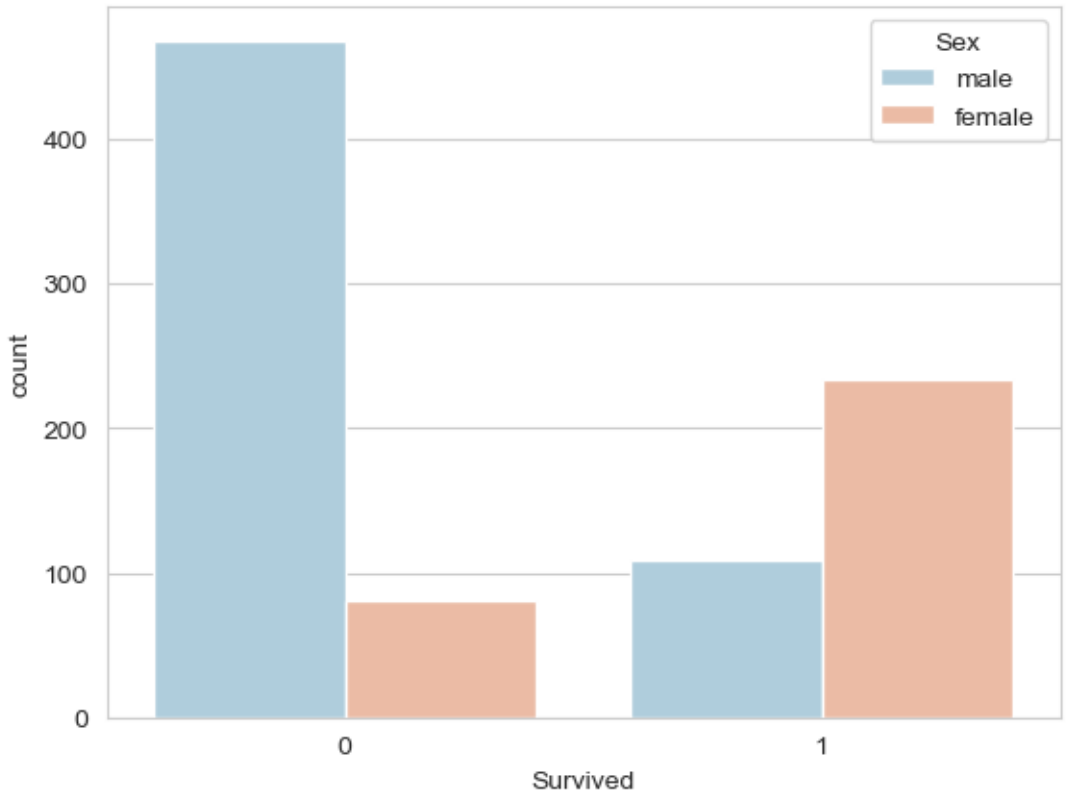
[5]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train)
```

[5]: <Axes: xlabel='Survived', ylabel='count'>



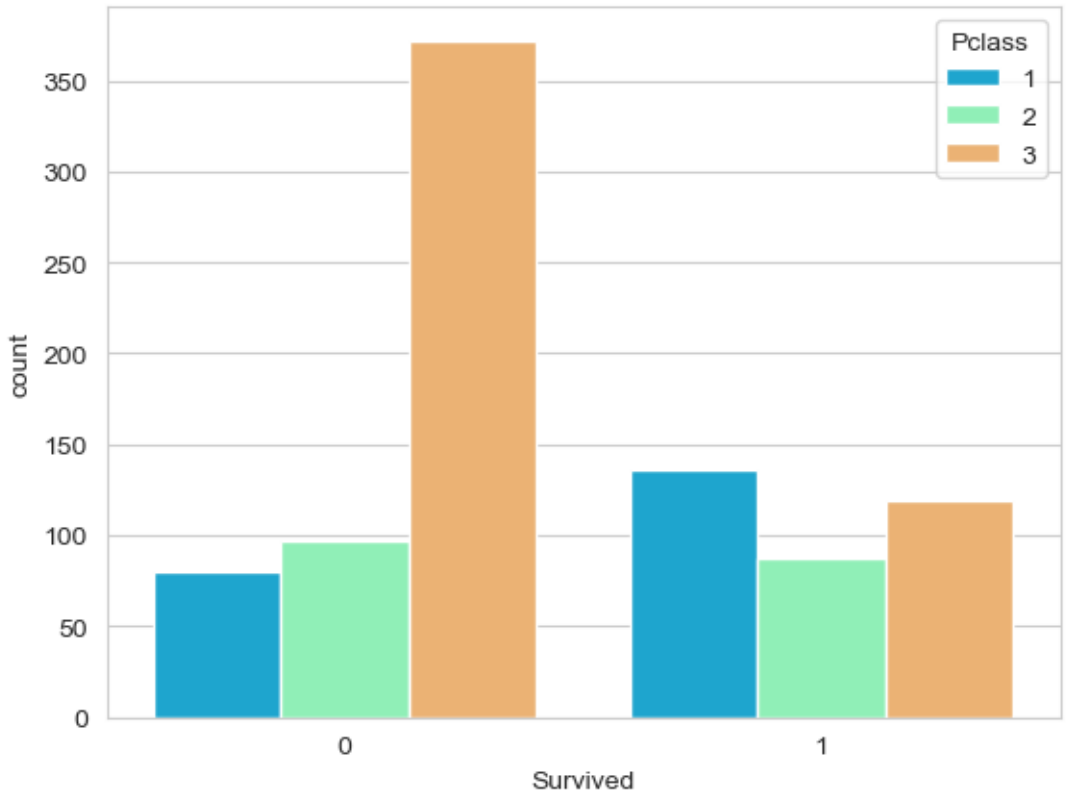
[6]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

[6]: <Axes: xlabel='Survived', ylabel='count'>



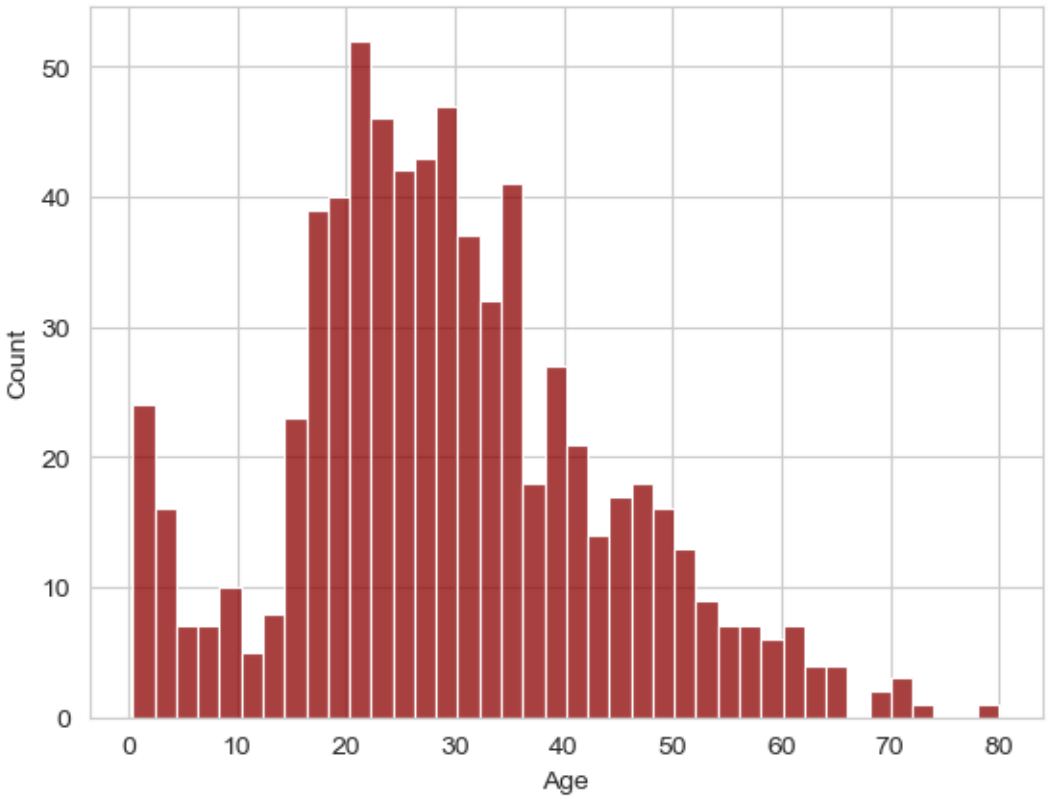
[7]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

[7]: <Axes: xlabel='Survived', ylabel='count'>



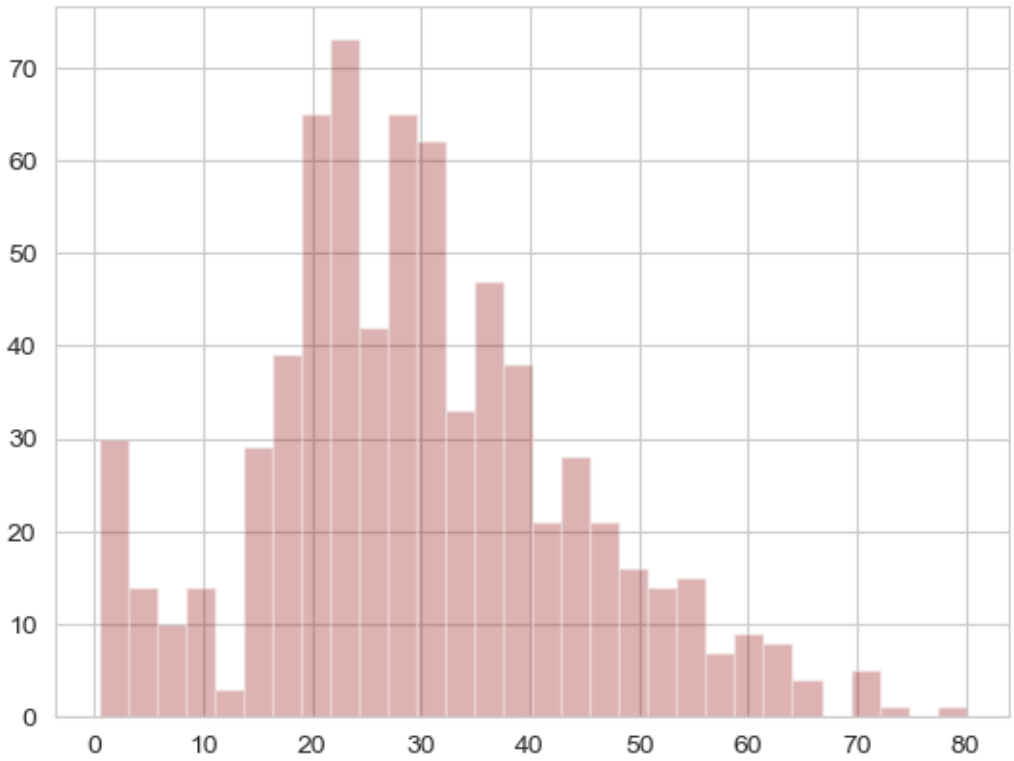
[8]:
```
sns.histplot(train['Age'].dropna(),kde=False,color='darkred',bins=40)
```

[8]: <Axes: xlabel='Age', ylabel='Count'>



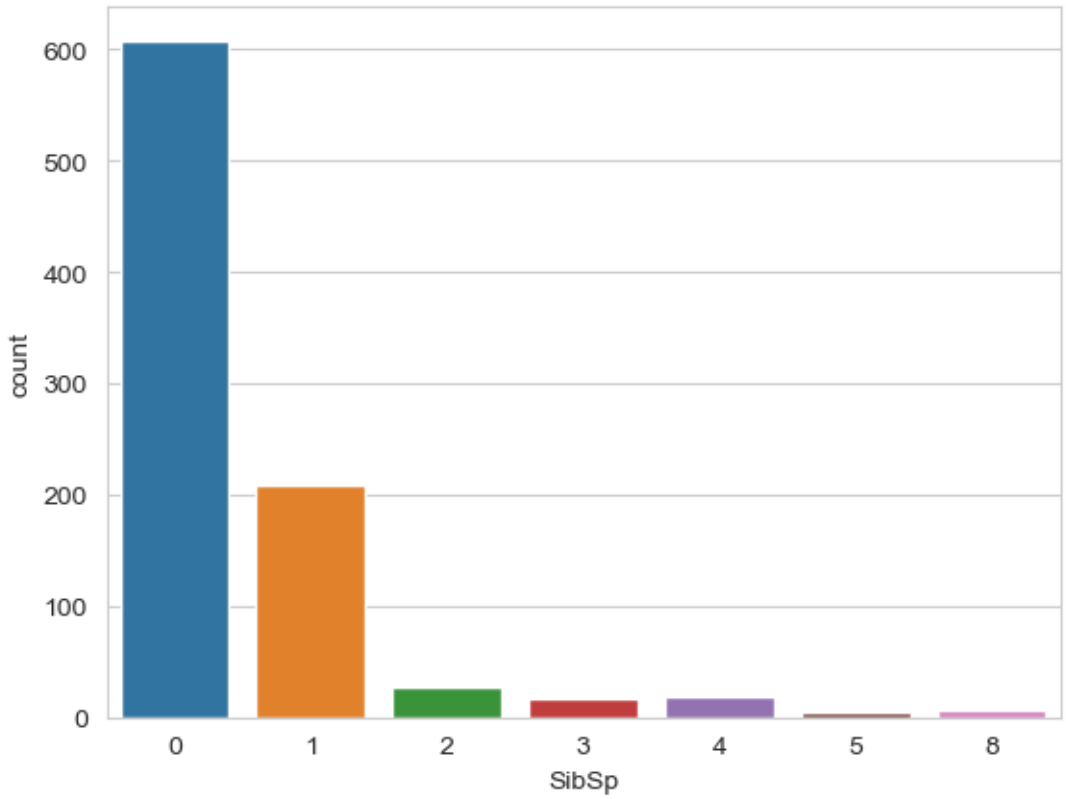
[9]:
```
train['Age'].hist(bins=30,color='darkred',alpha=0.3)
```

[9]: <Axes: >



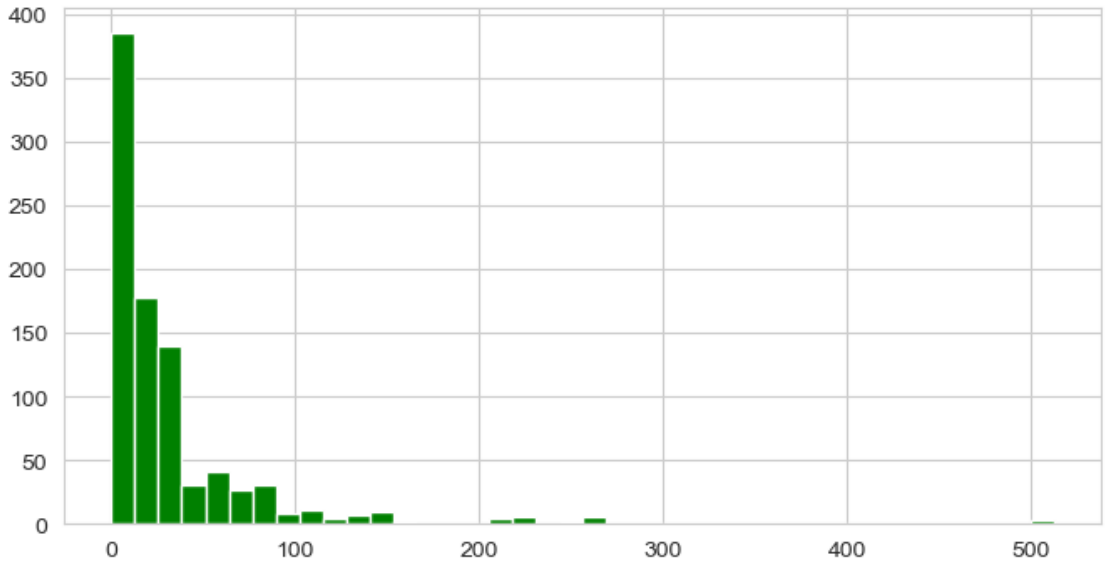
[10]:
```
sns.countplot(x='SibSp',data=train)
```

[10]: <Axes: xlabel='SibSp', ylabel='count'>



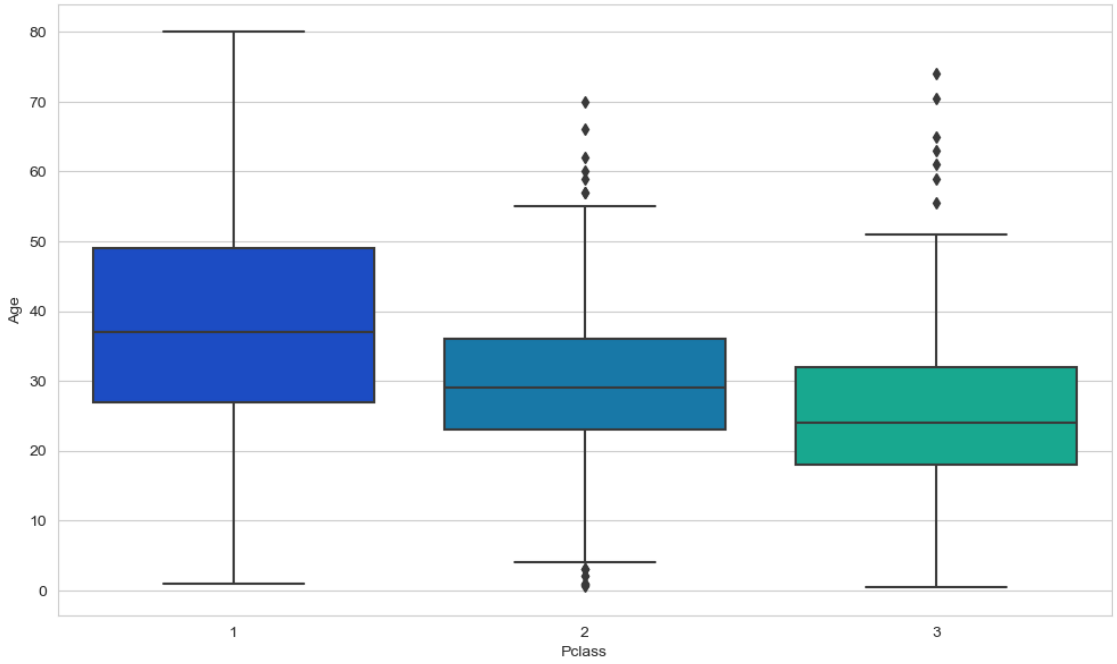
[11]:
```
train['Fare'].hist(color='green',bins=40,figsize=(8,4))
```

[11]: <Axes: >



[12]:
```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

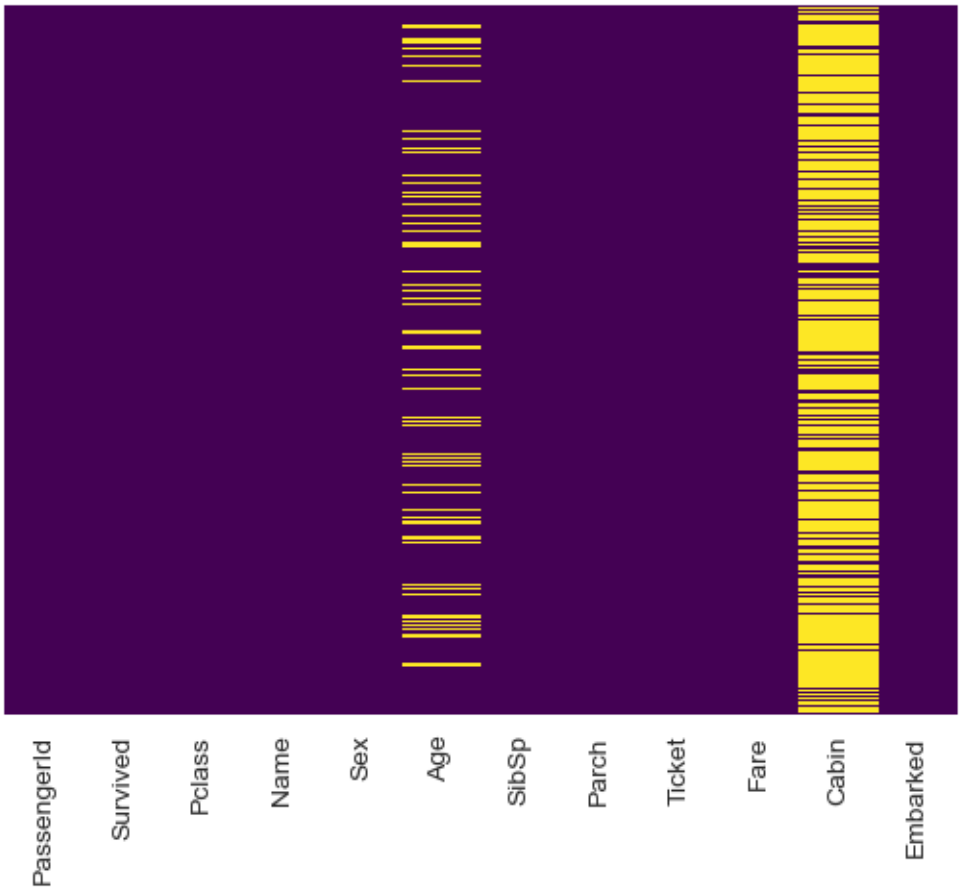
[12]:
```
<Axes: xlabel='Pclass', ylabel='Age'>
```



[13]:
```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

[14]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

[14]: <Axes: >



[15]:
```
train.drop('Cabin',axis=1,inplace=True)
```

[16]:
```
train.head()
```

[16]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0

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

[17]:
```
train.dropna(inplace=True)
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 712 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  712 non-null    int64
 1   Survived     712 non-null    int64
 2   Pclass       712 non-null    int64
 3   Name         712 non-null    object
 4   Sex          712 non-null    object
 5   Age          712 non-null    float64
 6   SibSp        712 non-null    int64
 7   Parch        712 non-null    int64
 8   Ticket       712 non-null    object
 9   Fare         712 non-null    float64
 10  Embarked     712 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 66.8+ KB
```

[18]:
```
pd.get_dummies(train['Embarked'],drop_first=True).head()
```

[18]:
```
   Q  S
0  0  1
1  0  0
2  0  1
3  0  1
4  0  1
```

[20]:
```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
train.head()
```

[20]:
```
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare
0            1         0       3  22.0      1      0   7.2500
1            2         1       1  38.0      1      0  71.2833
```

```
2            3         1       3  26.0      0      0   7.9250
3            4         1       1  35.0      1      0  53.1000
4            5         0       3  35.0      0      0   8.0500
```

[21]:
```
train = pd.concat([train,sex,embark],axis=1)
train.head()
```

[21]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	1	0	1
1	2	1	1	38.0	1	0	71.2833	0	0	0
2	3	1	3	26.0	0	0	7.9250	0	0	1
3	4	1	1	35.0	1	0	53.1000	0	0	1
4	5	0	3	35.0	0	0	8.0500	1	0	1

[22]:
```
train.drop('Survived',axis=1).head()
```

[22]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	3	22.0	1	0	7.2500	1	0	1
1	2	1	38.0	1	0	71.2833	0	0	0
2	3	3	26.0	0	0	7.9250	0	0	1
3	4	1	35.0	1	0	53.1000	0	0	1
4	5	3	35.0	0	0	8.0500	1	0	1

[23]:
```
train['Survived'].head()
```

[23]:
```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

[34]:
```
from sklearn.linear_model import LogisticRegression
```

[36]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
C:\ProgramData\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
```

```
  n_iter_i = _check_optimize_result(
```

[36]:
```
LogisticRegression()
```

[37]:
```
predictions = logmodel.predict(X_test)
```

[38]:
```
from sklearn.metrics import confusion_matrix
```

[39]:
```
accuracy=confusion_matrix(y_test,predictions)
```

[40]:
```
accuracy
```

[40]:
```
array([[104,  24],
       [ 26,  60]], dtype=int64)
```

[41]:
```
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(y_test,predictions)
accuracy
```

[41]:
```
0.7663551401869159
```

[42]:
```
predictions
```

[42]:
```
array([0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1], dtype=int64)
```

[43]:
```
from sklearn.metrics import classification_report
```

[44]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.80      0.81      0.81       128
           1       0.71      0.70      0.71        86

    accuracy                           0.77       214
   macro avg       0.76      0.76      0.76       214
weighted avg       0.77      0.77      0.77       214
```

[48]:
```
train.isna().sum()
```

[48]:
```
PassengerId    0
Survived       0
Pclass         0
Age            0
SibSp          0
Parch          0
Fare           0
male           0
Q              0
S              0
dtype: int64
```

[49]:
```
for val in train:
   print(train[val].value_counts())
   print()
```

```
1      1
622    1
595    1
596    1
598    1
      ..
298    1
300    1
303    1
306    1
891    1
Name: PassengerId, Length: 712, dtype: int64

0    424
1    288
Name: Survived, dtype: int64

3    355
1    184
2    173
Name: Pclass, dtype: int64

24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
         ..
36.50     1
```

```
55.50     1
0.92      1
23.50     1
74.00     1
Name: Age, Length: 88, dtype: int64

0    469
1    183
2     25
4     18
3     12
5      5
Name: SibSp, dtype: int64

0    519
1    110
2     68
5      5
3      5
4      4
6      1
Name: Parch, dtype: int64

13.0000    41
26.0000    30
8.0500     29
10.5000    24
7.8958     23
           ..
6.2375      1
14.0000     1
9.4750      1
8.8500      1
10.5167     1
Name: Fare, Length: 219, dtype: int64

1    453
0    259
Name: male, dtype: int64

0    684
1     28
Name: Q, dtype: int64

1    554
0    158
Name: S, dtype: int64
```

[56]:
```
train.shape
```

[56]:
```
(712, 10)
```

[63]:
```
train.dtypes
```

[63]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Age            float64
SibSp            int64
Parch            int64
Fare           float64
male             int64
Q                uint8
S                uint8
dtype: object
```

[66]:
```
#Split the data into independent 'X' and dependent 'Y' variables
X = train.iloc[:, 1:8].values
Y = train.iloc[:, 0].values
```

[67]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state = 0)
```

[68]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[69]:
```
def models(X_train,Y_train):

  #Using Logistic Regression Algorithm to the Training Set
  from sklearn.linear_model import LogisticRegression
  log = LogisticRegression(random_state = 0)
  log.fit(X_train, Y_train)

  #Using KNeighborsClassifier Method of neighbors class to use Nearest Neighbor␣
 ↪algorithm
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)

  #Using SVC method of svm class to use Support Vector Machine Algorithm
  from sklearn.svm import SVC
  svc_lin = SVC(kernel = 'linear', random_state = 0)
```

```
    svc_lin.fit(X_train, Y_train)

    #Using SVC method of svm class to use Kernel SVM Algorithm
    from sklearn.svm import SVC
    svc_rbf = SVC(kernel = 'rbf', random_state = 0)
    svc_rbf.fit(X_train, Y_train)

    #Using GaussianNB method of naïve_bayes class to use Naïve Bayes Algorithm
    from sklearn.naive_bayes import GaussianNB
    gauss = GaussianNB()
    gauss.fit(X_train, Y_train)

    #Using DecisionTreeClassifier of tree class to use Decision Tree Algorithm
    from sklearn.tree import DecisionTreeClassifier
    tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    tree.fit(X_train, Y_train)

    #Using RandomForestClassifier method of ensemble class to use Random Forest␣
    ↪Classification algorithm
    from sklearn.ensemble import RandomForestClassifier
    forest = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',␣
    ↪random_state = 0)
    forest.fit(X_train, Y_train)

    #print model accuracy on the training data.
    print('[0]Logistic Regression Training Accuracy:', log.score(X_train,␣
    ↪Y_train))
    print('[1]K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
    print('[2]Support Vector Machine (Linear Classifier) Training Accuracy:',␣
    ↪svc_lin.score(X_train, Y_train))
    print('[3]Support Vector Machine (RBF Classifier) Training Accuracy:',␣
    ↪svc_rbf.score(X_train, Y_train))
    print('[4]Gaussian Naive Bayes Training Accuracy:', gauss.score(X_train,␣
    ↪Y_train))
    print('[5]Decision Tree Classifier Training Accuracy:', tree.score(X_train,␣
    ↪Y_train))
    print('[6]Random Forest Classifier Training Accuracy:', forest.score(X_train,␣
    ↪Y_train))

    return log, knn, svc_lin, svc_rbf, gauss, tree, forest
```

[70]:
```
#Get and train all of the models
model = models(X_train,Y_train)
```

```
[0]Logistic Regression Training Accuracy: 0.3022847100175747
[1]K Nearest Neighbor Training Accuracy: 0.21616871704745166
[2]Support Vector Machine (Linear Classifier) Training Accuracy:
```

```
0.9472759226713533
[3]Support Vector Machine (RBF Classifier) Training Accuracy: 0.9507908611599297
[4]Gaussian Naive Bayes Training Accuracy: 0.9507908611599297
[5]Decision Tree Classifier Training Accuracy: 0.9507908611599297
[6]Random Forest Classifier Training Accuracy: 0.9314586994727593
```

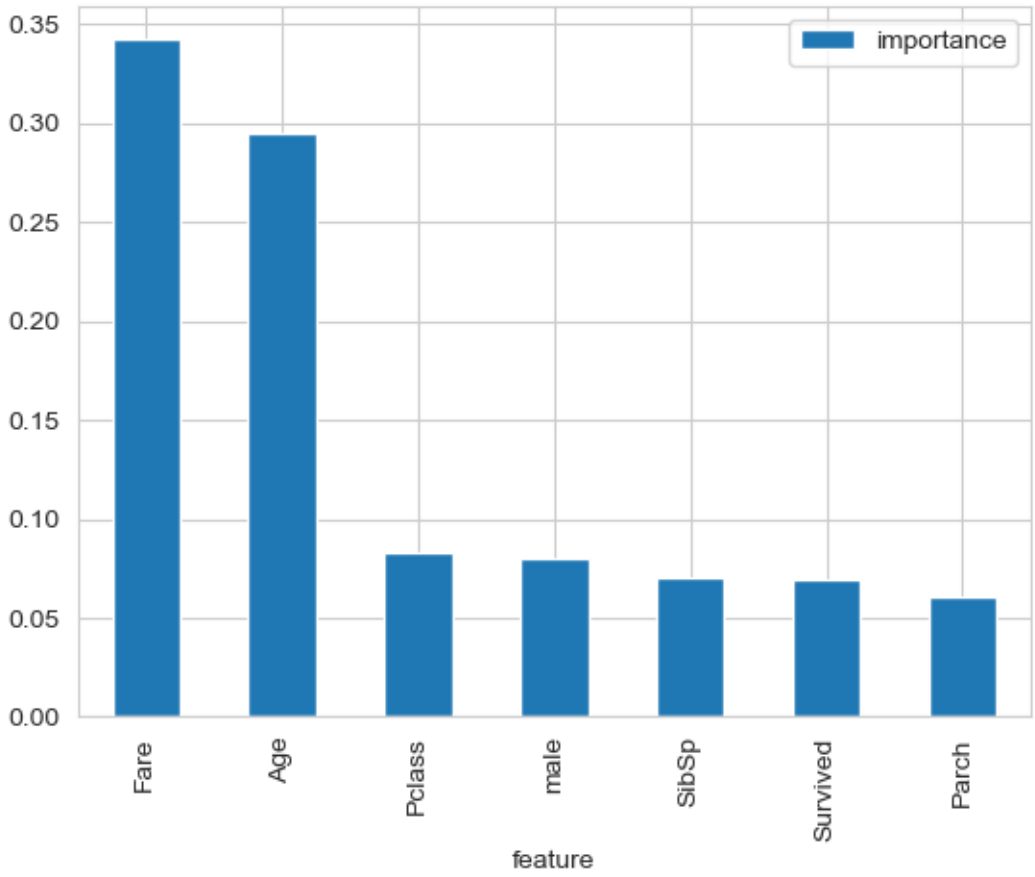
[74]:
```
forest = model[6]
importances = pd.DataFrame({'feature':train.iloc[:, 1:8].columns,'importance':
 ↪np.round(forest.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).
 ↪set_index('feature')
importances
```

[74]:

feature	importance
Fare	0.342
Age	0.295
Pclass	0.083
male	0.080
SibSp	0.070
Survived	0.069
Parch	0.061

[75]:
```
importances.plot.bar()
```

[75]:
```
<Axes: xlabel='feature'>
```



[76]:
```
pred = model[6].predict(X_test)
print(pred)

#Print a space
print()

#Print the actual values
print(Y_test)
```

```
[252 343 803 345  95 451  26 156 556 114 711   3 258 647 145 162 440 470
 631 809 363 866 782 184 873 175 345 621 691 504  28 267 442 763 551 294
 578 702 188 438 674  82 401 286 282  94 428 121 871 538 324 475   9 440
  34 101   5 163 514  93 252 130 132 756  21 685 196 600 499 260 343 231
 447  54 154 873 194  45 282 734 873 524 841 238 797 489 189 620 876 228
 223  94 890  85  40 106 231 497 871 674 132 480 345 556 499 373 691 734
 556 151 396 475 400 280 744 343 674 105 406 143 380  94 440  25 699 282
 788 315 420 291 580 459 400  23  54 701   3  95  64  76 156 350 609]

[424 179 306 293 593 596 473  55 457 112 310 316 505 567 862  16 243 645
```

```
 450 796 133 347  10 550 823 493 723  74 725  41 439 687 683 377 446 200
   2 513 461 775 227 287 529 104 757 747 888 656 629 731 601 883 824 220
 773 730 615 443 557  36 658 632 379 349 552 746 610 713 298  99 214 487
 418 727 198 264 341  69 423 659 807 880 665 507  12  91 617 884 781 492
 153 549 608 390 831 314 413 592 811 289 144 692 801 546 686  68 371 865
 253 627  52 678  67 329 354 419  22 861 100  86 716 425  71 853 263 772
 279 638 703 628 430 527 323 778 317 308 217 327 851 704 125 472 616]
```

[77]:
```
my_survival = [[3,1,21,0, 0, 0, 1]]
#Print Prediction of Random Forest Classifier model
pred = model[6].predict(my_survival)
print(pred)

if pred == 0:
    print("Oh no! You didn't make it")
else:
    print('Nice! You survived')
```

```
[600]
Nice! You survived
```

[ ]:
```

```

[ ]:
```
Conclusion:
The task to predict if a passenger would survive the Titanic or not is␣
↪completed.
```