In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
social_data = pd.read_csv("Social_Network_Ads.csv")
```

In [3]:

```
social_data.head()
```

Out[3]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

In [4]:

```
social_data.tail()
```

Out[4]:

	Age	EstimatedSalary	Purchased
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

In [5]:

```
social_data.shape
```

Out[5]:

```
(400, 3)
```

In [6]:

```
social_data.columns
```

Out[6]:

```
Index(['Age', 'EstimatedSalary', 'Purchased'], dtype='object')
```

In [7]:

```
social_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Age              400 non-null    int64
 1   EstimatedSalary  400 non-null    int64
 2   Purchased        400 non-null    int64
dtypes: int64(3)
memory usage: 9.5 KB
```

In [8]:

```
social_data.describe()
```

Out[8]:

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

In [9]:

```
social_data.isnull().sum()
```

Out[9]:

```
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [10]:

```
social_data.nunique()
```

Out[10]:

```
Age                43
EstimatedSalary    117
Purchased            2
dtype: int64
```

In [11]:

```
social_data.Purchased.unique()
```

Out[11]:

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

In [12]:

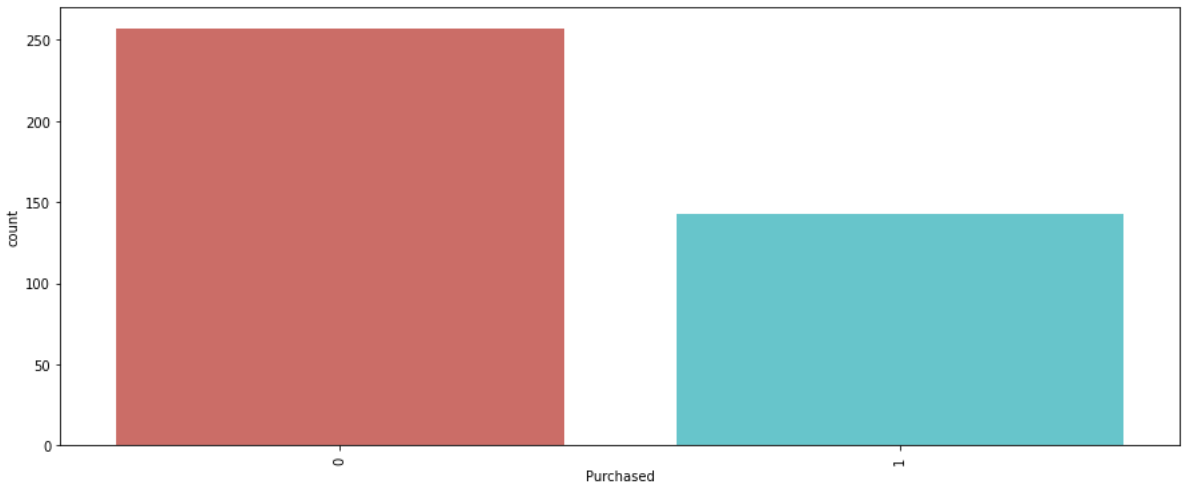
```
social_data.Purchased.value_counts()
```

Out[12]:

```
0    257
1    143
Name: Purchased, dtype: int64
```

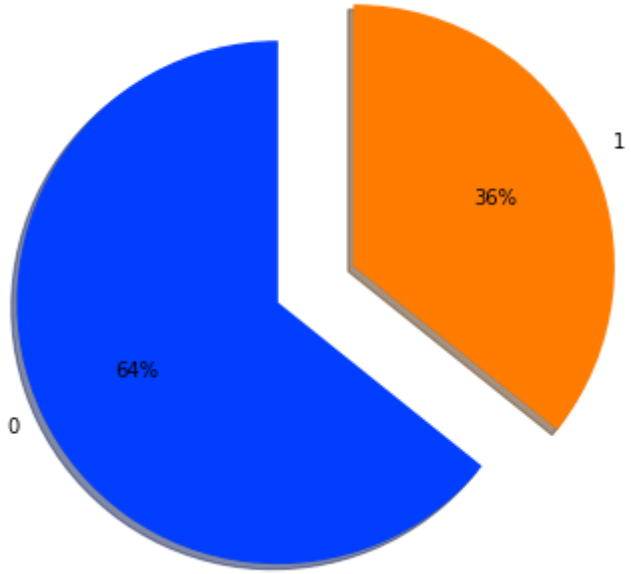
In [13]:

```
plt.figure(figsize=(15,6))
sns.countplot('Purchased', data = social_data, palette='hls')
plt.xticks(rotation = 90)
plt.show()
```



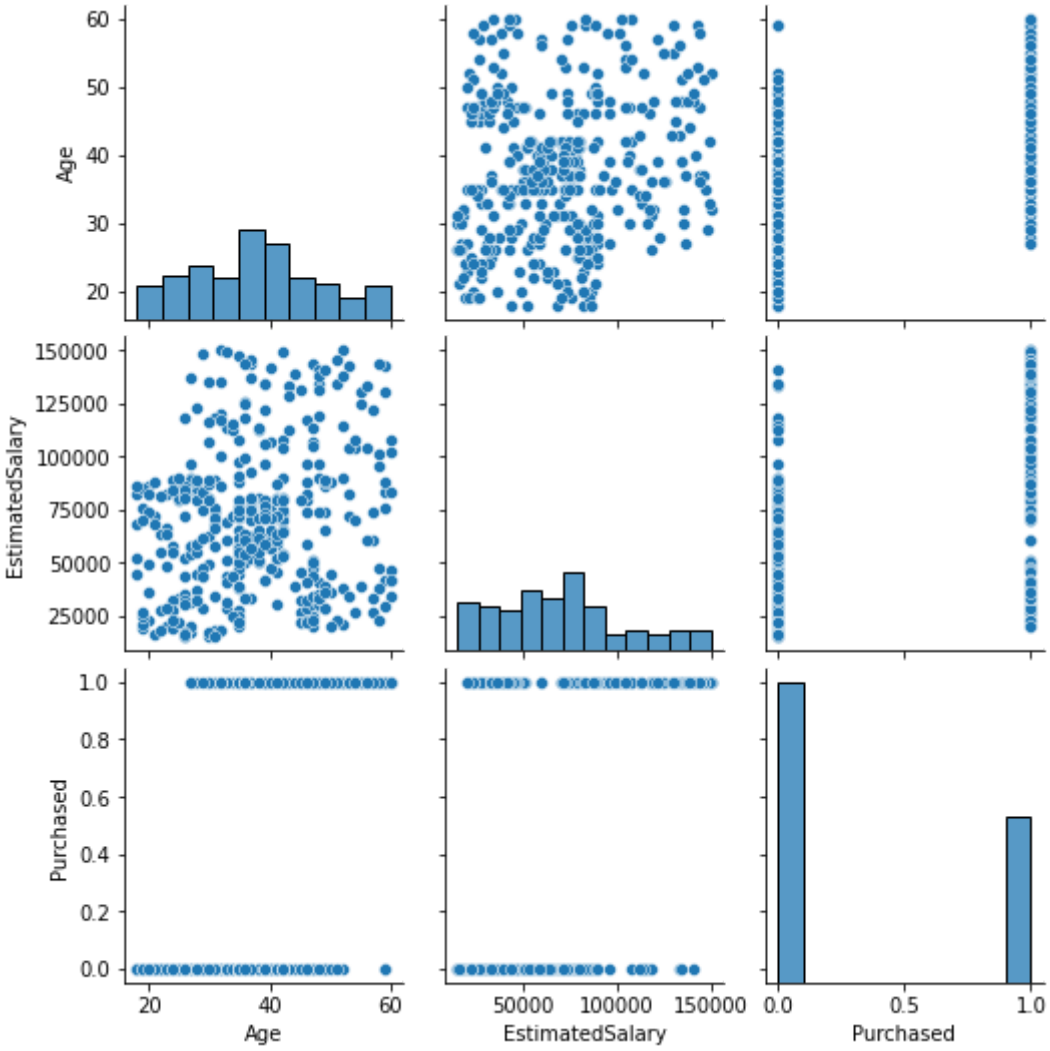
In [14]:

```
plt.figure(figsize=(15,6))
colors = sns.color_palette('bright')
explode = [0.3, 0.02]
plt.pie(social_data.Purchased.value_counts(), colors = colors, labels = [0, 1],
        explode = explode, autopct = '%0.0f%%', shadow = 'True', startangle = 90)
plt.show()
```



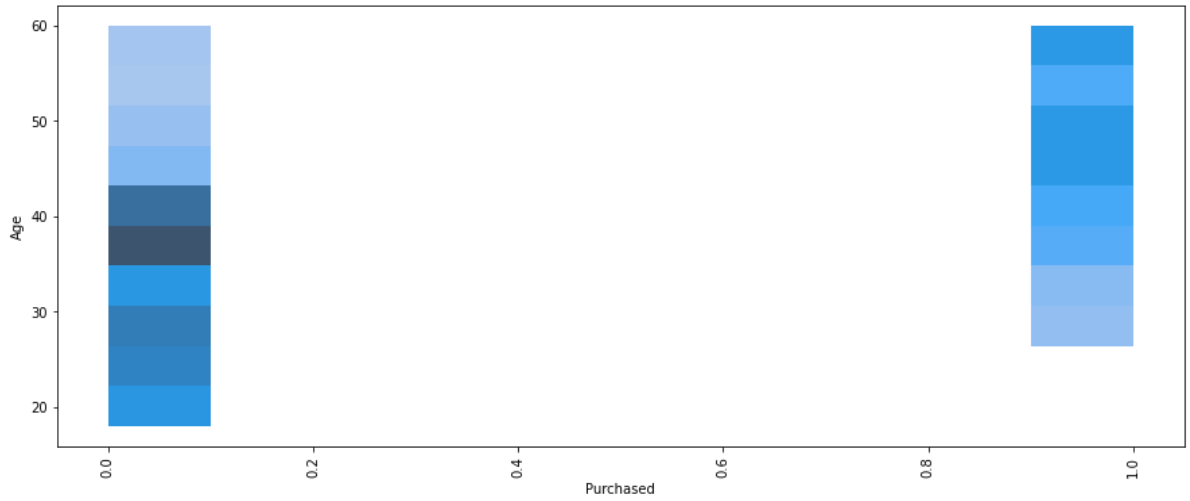
In [15]:

```
sns.pairplot(social_data)
plt.show()
```



In [19]:

```
plt.figure(figsize=(15,6))
sns.histplot(y = 'Age', x = 'Purchased', data = social_data)
plt.xticks(rotation = 90)
plt.show()
```

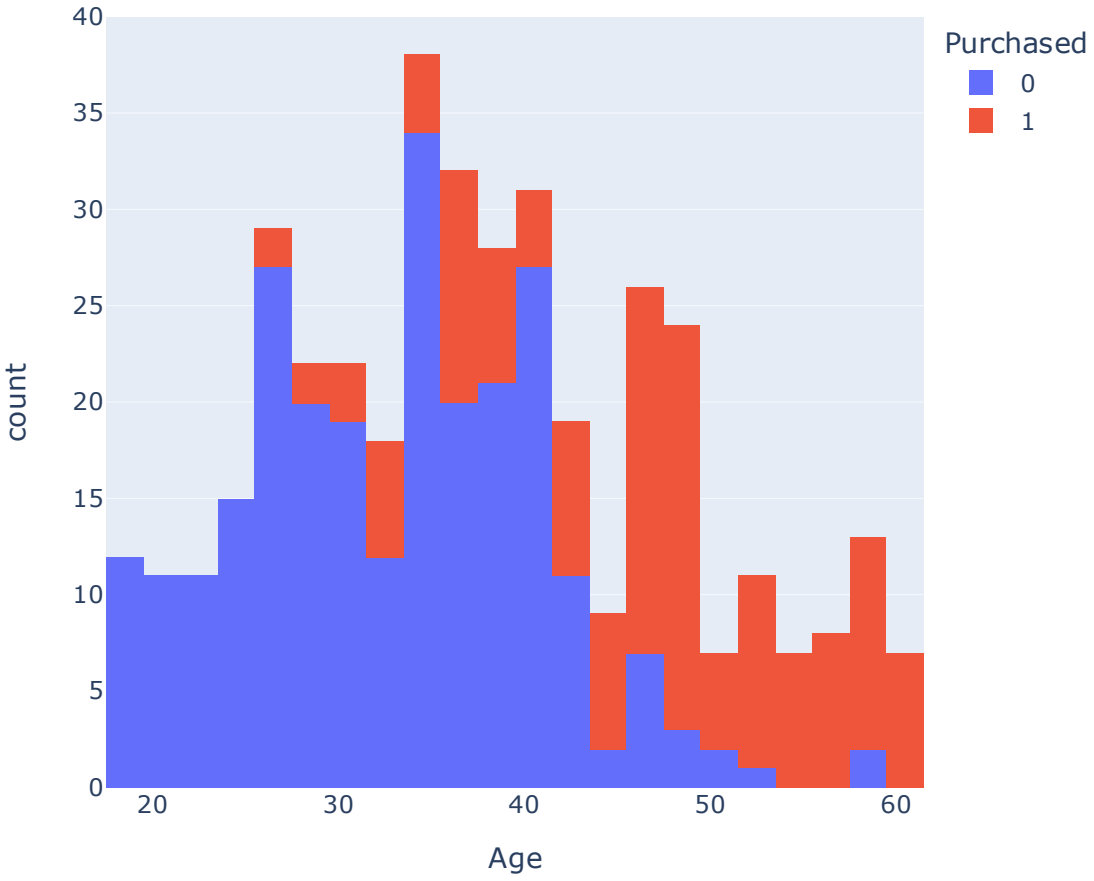


In [20]:

```
import plotly.express as px
```

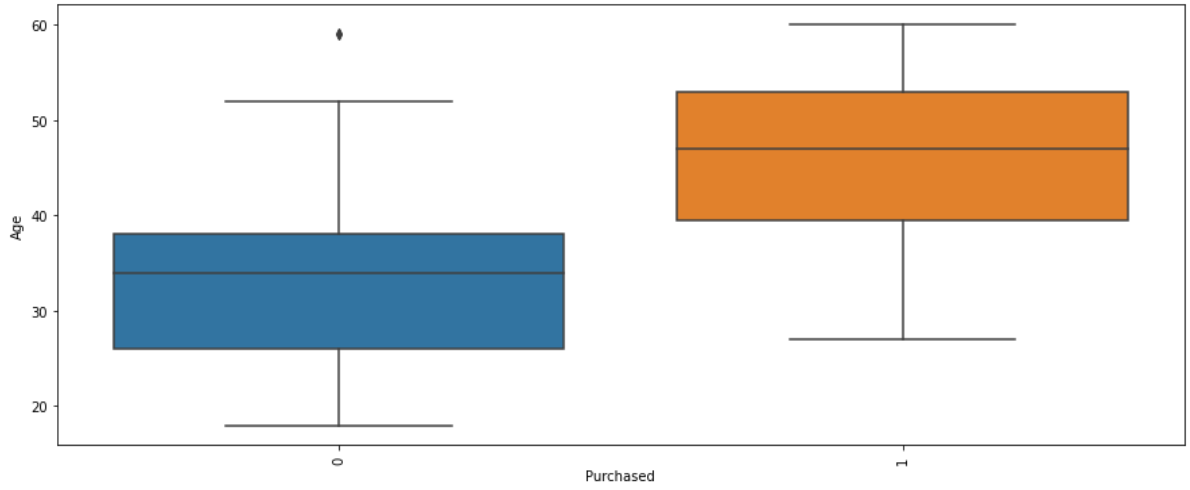
In [22]:

```
fig1 = px.histogram(social_data, x = 'Age', color = 'Purchased')
fig1.show()
```



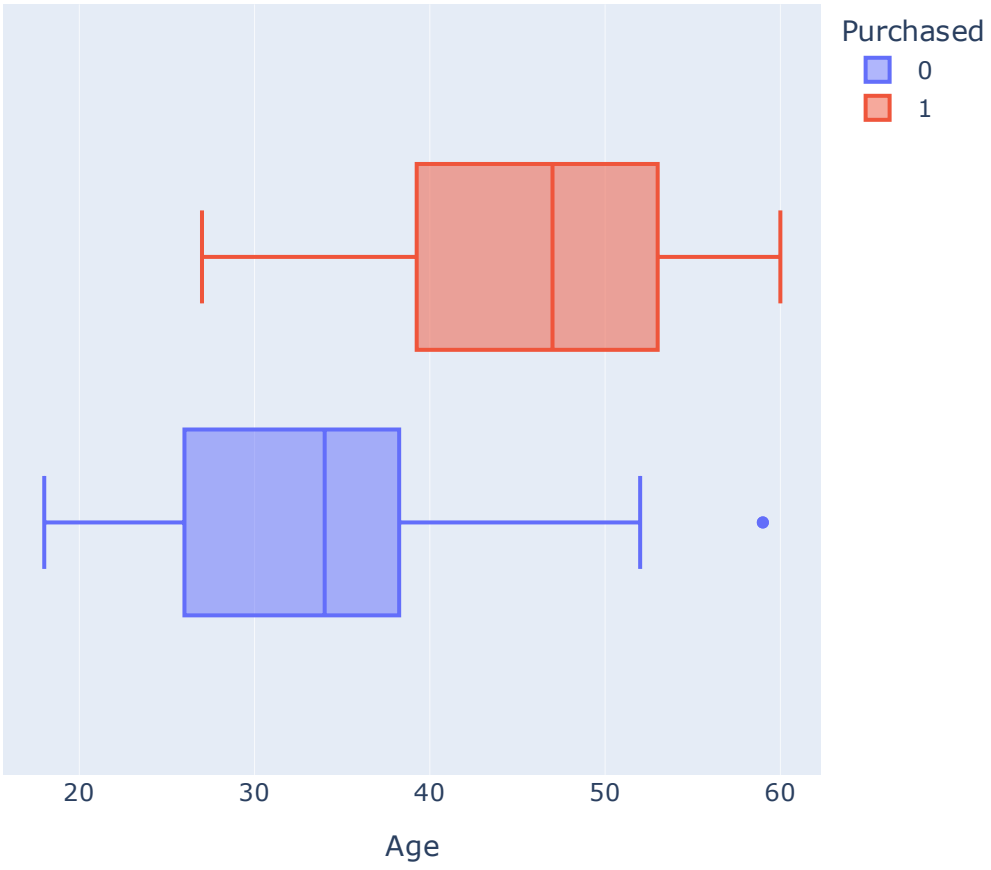
In [23]:

```
plt.figure(figsize=(15,6))
sns.boxplot(y = 'Age', x = 'Purchased', data = social_data)
plt.xticks(rotation = 90)
plt.show()
```



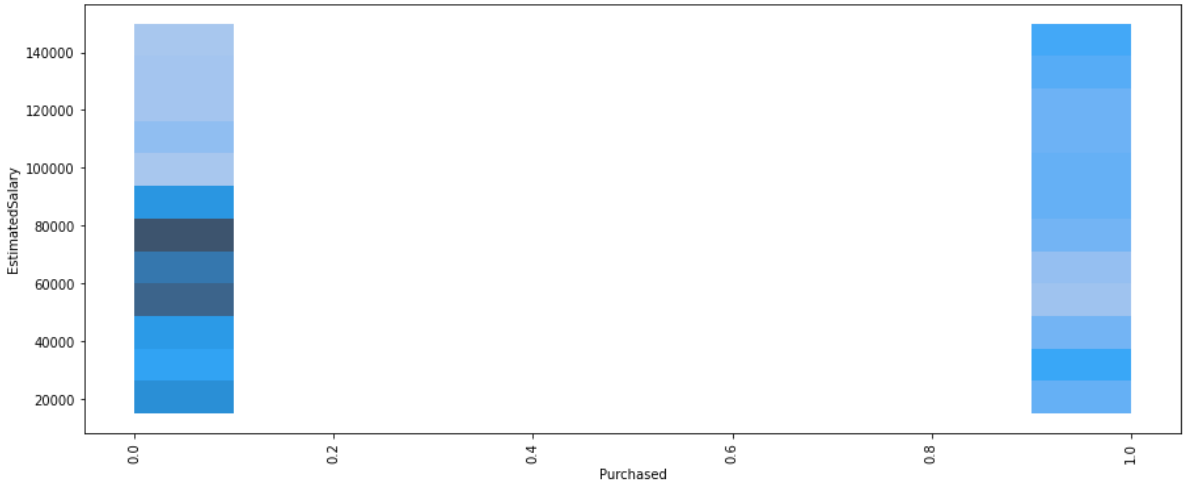
In [24]:

```
fig2 = px.box(social_data, x = 'Age', color = 'Purchased')
fig2.show()
```



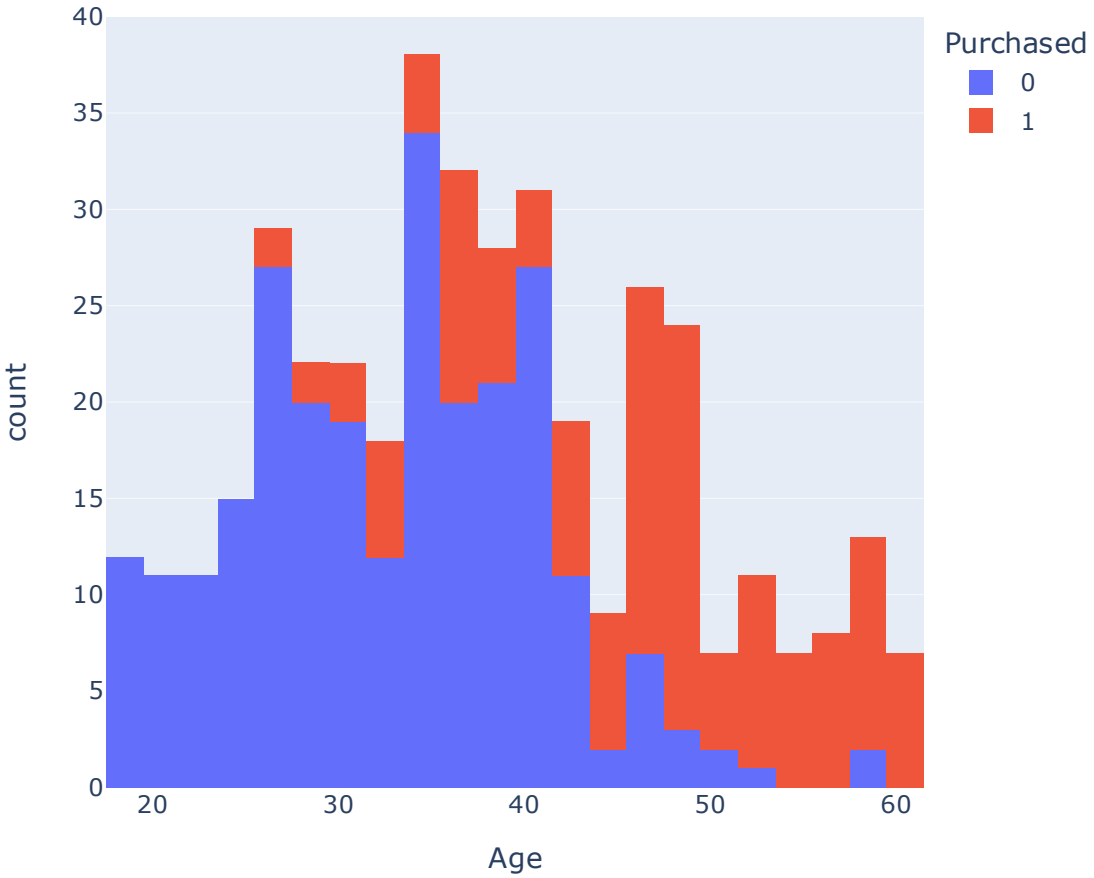
In [25]:

```
plt.figure(figsize=(15,6))
sns.histplot(y = 'EstimatedSalary', x = 'Purchased', data = social_data)
plt.xticks(rotation = 90)
plt.show()
```



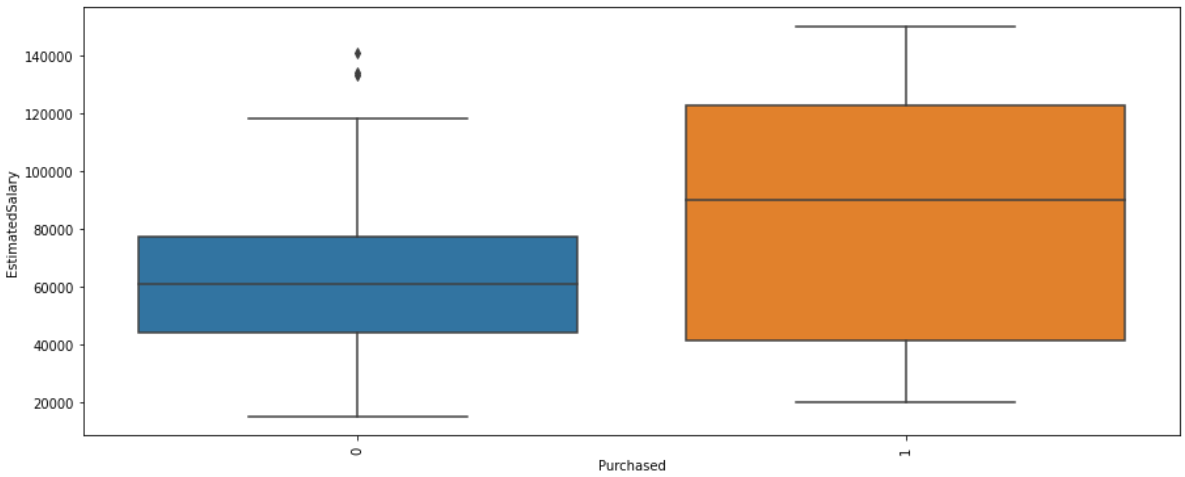
In [26]:

```
fig3 = px.histogram(social_data, x = 'EstimatedSalary', color = 'Purchased')
fig1.show()
```



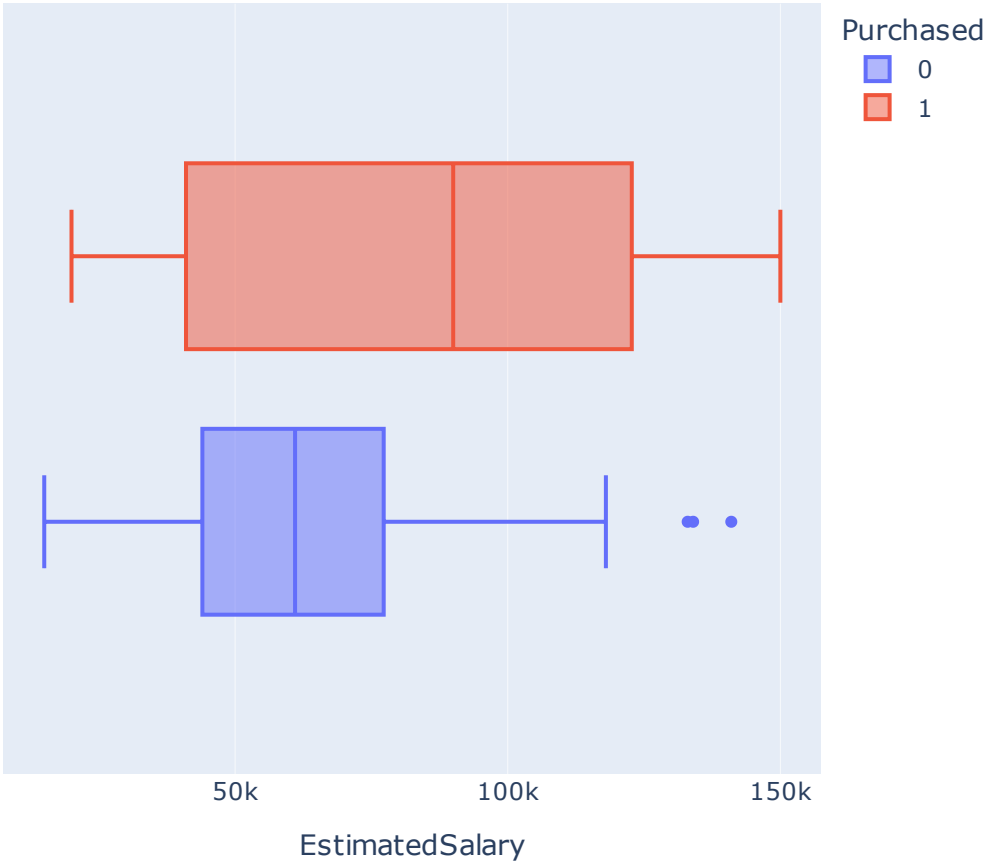
In [27]:

```
plt.figure(figsize=(15,6))
sns.boxplot(y = 'EstimatedSalary', x = 'Purchased', data = social_data)
plt.xticks(rotation = 90)
plt.show()
```



In [28]:

```
fig4 = px.box(social_data, x = 'EstimatedSalary', color = 'Purchased')
fig4.show()
```



In [29]:

```
x = social_data.drop(['Purchased'], axis = 1)
y = social_data.Purchased
```

In [30]:

```
x.shape
```

Out[30]:

```
(400, 2)
```

In [31]:

```
y.shape
```

Out[31]:

```
(400,)
```

In [32]:

```
x_train = x.iloc[:301]
```

In [33]:

```
y_train = y.iloc[:301]
```

In [34]:

```
x_test = x.iloc[301:401]
```

In [35]:

```
y_test = y.iloc[301:401]
```

In [36]:

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

In [37]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [39]:

```
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski',
                                  p = 2)
classifier.fit(x_train,y_train)
```

Out[39]:

```
KNeighborsClassifier()
```

In [40]:

```
y_pred = classifier.predict(x_test)
```

In [41]:

```
print("Training Accuracy :", classifier.score(x_train, y_train))
print("Testing Accuracy :", classifier.score(x_test, y_test))
```

```
Training Accuracy : 0.9235880398671097
Testing Accuracy : 0.696969696969697
```

In [42]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test,y_pred)
print(cm)
accuracy_score(y_test,y_pred)
```

```
[[37  1]
 [29 32]]
```

Out[42]:

```
0.696969696969697
```

In [43]:

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression, SGDClassifier, Perceptron, RidgeCla
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoos
from sklearn.svm import SVC,NuSVC
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.metrics import precision_score
```

In [45]:

```
models =[("LR", LogisticRegression()),("SVC", SVC()),
         ('KNN',KNeighborsClassifier()),
         ("DTC", DecisionTreeClassifier()),
         ("GNB", GaussianNB()),("SGDC", SGDClassifier()),
         ("Perc", Perceptron()),
         ("Ridge", RidgeClassifier()),("NuSVC", NuSVC()),
         ("BNB", BernoulliNB()),('RF',RandomForestClassifier()),
         ('ADA',AdaBoostClassifier()),('XGB',GradientBoostingClassifier()),('PAC',Passiv
pred = []
names = []
modelsprecision = []
```

In [46]:

```
for name,model in models:
    model.fit(x_train, y_train)
    prediction = model.predict(x_test)
    score = precision_score(y_test, prediction,average = 'macro')
    pred.append(score)
    names.append(name)
    modelsprecision.append((name,score))

modelsprecision.sort(key=lambda k:k[1],reverse=True)
```

In [47]:

```
modelsprecision
```

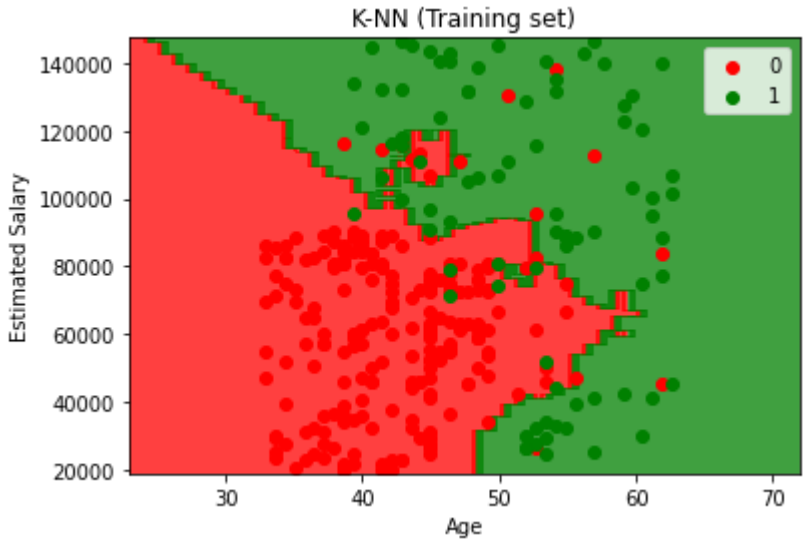
Out[47]:

```
[('SVC', 0.7848735832606801),
 ('KNN', 0.7651515151515151),
 ('DTC', 0.7567567567567568),
 ('NuSVC', 0.7470443349753695),
 ('GNB', 0.7427062374245472),
 ('XGB', 0.7384259259259258),
 ('LR', 0.7341938883034773),
 ('PAC', 0.7341938883034773),
 ('SGDC', 0.73),
 ('RF', 0.73),
 ('Ridge', 0.7216819221967964),
 ('ADA', 0.7091772151898734),
 ('BNB', 0.6723684210526316),
 ('Perc', 0.6296583850931676)]
```

In [50]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = scale.inverse_transform(x_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max()
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max(
plt.contourf(X1, X2, classifier.predict(scale.transform(np.array([X1.ravel(), X2.ravel()
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
plt.title('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which shoul
d be avoided as value-mapping will have precedence in case its length matc
hes with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA v
alue for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which shoul
d be avoided as value-mapping will have precedence in case its length matc
hes with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA v
alue for all points.
```



In [52]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = scale.inverse_transform(x_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max()
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max(
plt.contourf(X1, X2, classifier.predict(scale.transform(np.array([X1.ravel(), X2.ravel()
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which shoul
d be avoided as value-mapping will have precedence in case its length matc
hes with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA v
alue for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which shoul
d be avoided as value-mapping will have precedence in case its length matc
hes with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA v
alue for all points.
```