Mastering Data Visualization Techniques (Part 2)

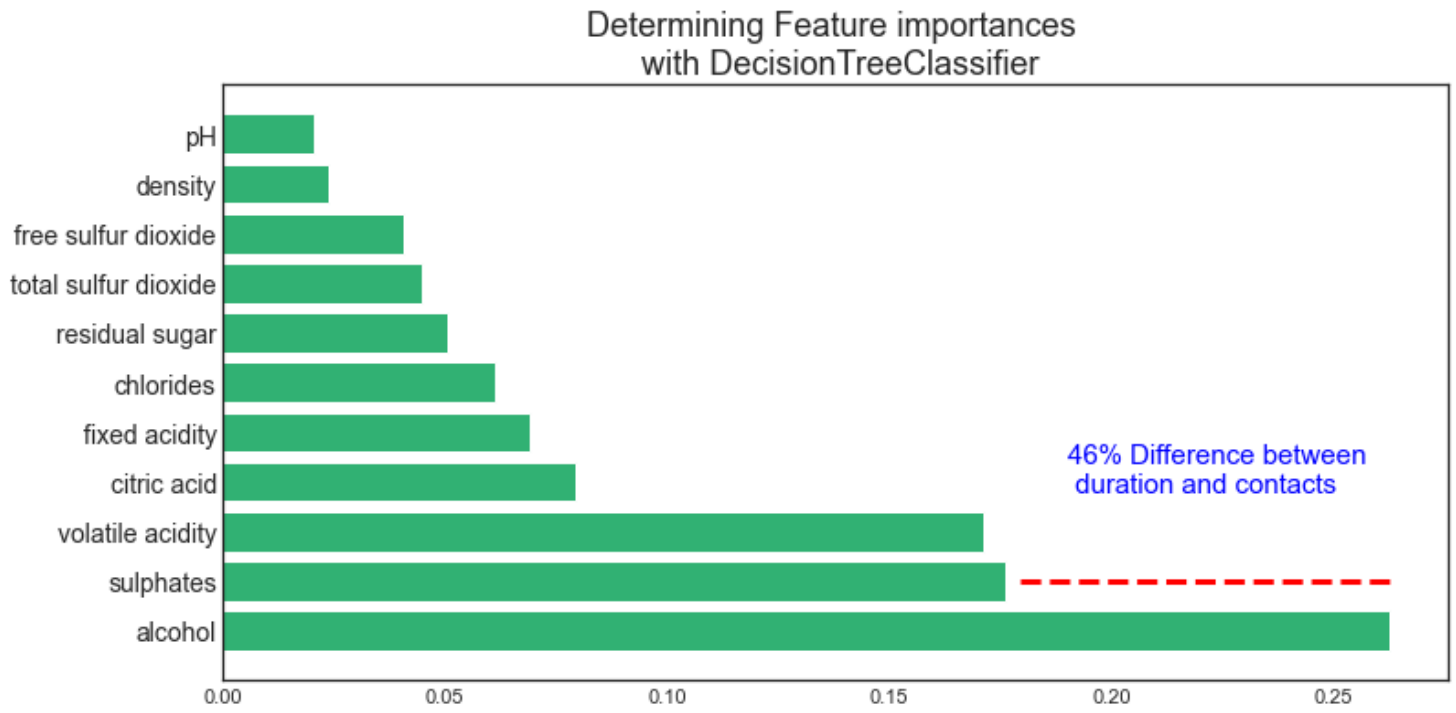
Prepared by: Syed Afroz Ali

Feature Importance Visualization

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
plt.style.use('seaborn-white')

# Create train and test splits
target_name = 'quality'
X = wine.drop('quality', axis=1)
label=wine[target_name]
```

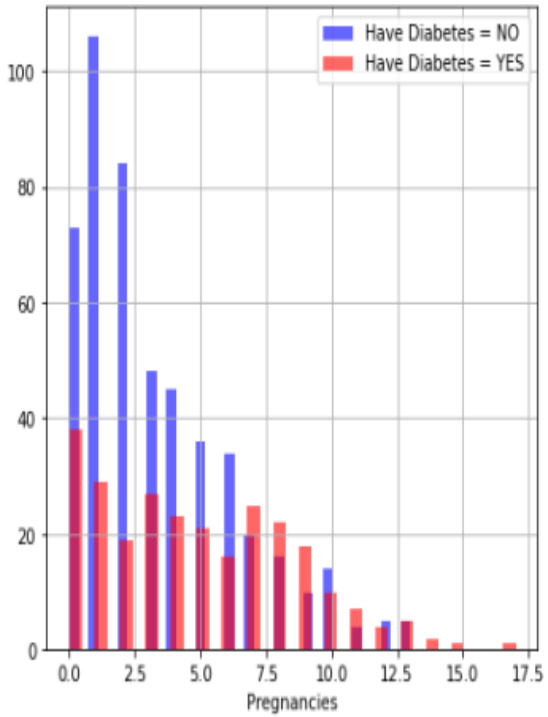
```
X_train, X_test, y_train, y_test = train_test_split(X,label,test_size=0.2,
random_state=42, stratify=label)
# Build a classification task using 3 informative features
tree = tree.DecisionTreeClassifier(
    class_weight='balanced',
    min_weight_fraction_leaf = 0.01
)
tree = tree.fit(X_train, y_train)
importances = tree.feature_importances_
feature_names = wine.drop('quality', axis=1).columns
indices = np.argsort(importances)[::-1]

# Print the feature ranking

for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f],
importances[indices[f]]))
```

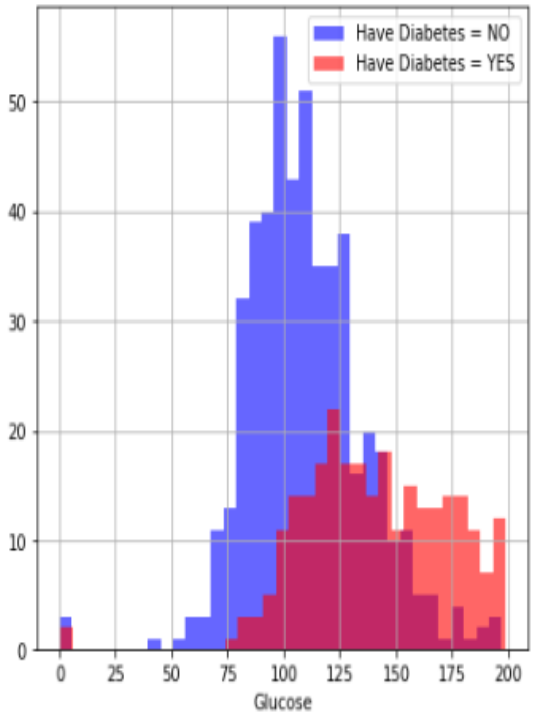
```
# Plot the feature importances of the forest
def feature_importance_graph(indices, importances, feature_names):
    plt.figure(figsize=(12,6))
    plt.title("Determining Feature importances \n with
DecisionTreeClassifier", fontsize=18)
    plt.barh(range(len(indices)), importances[indices],
color='#31B173',  align="center")
    plt.yticks(range(len(indices)), feature_names[indices],
rotation='horizontal',fontsize=14)
    plt.ylim([-1, len(indices)])
    plt.axhline(y=1.0, xmin=0.65, xmax=0.952, color='red', linewidth=3,
linestyle='--')
    plt.text(0.19, 2.8, '46% Difference between \n duration and
contacts', color='Blue', fontsize=15)

feature_importance_graph(indices, importances, feature_names)
plt.show()
```

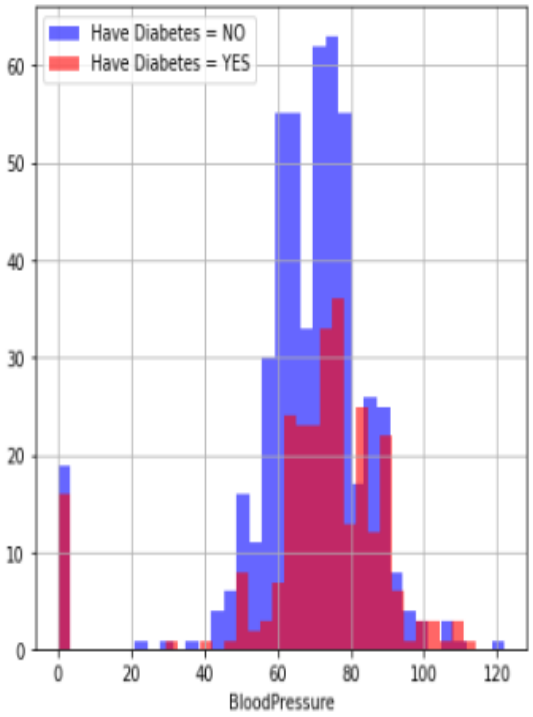



```
# Visulazing the distibution of the data for every feature
plt.figure(figsize=(20, 20))

for i, column in enumerate(df.columns, 1):
    plt.subplot(3, 3, i)
```

```
    df[df["Outcome"] == 0] [column].hist(bins=35, color='blue',
label='Have Diabetes = NO', alpha=0.6)
    df[df["Outcome"] == 1] [column].hist(bins=35, color='red',
label='Have Diabetes = YES', alpha=0.6)
    plt.legend()
    plt.xlabel(column)
```







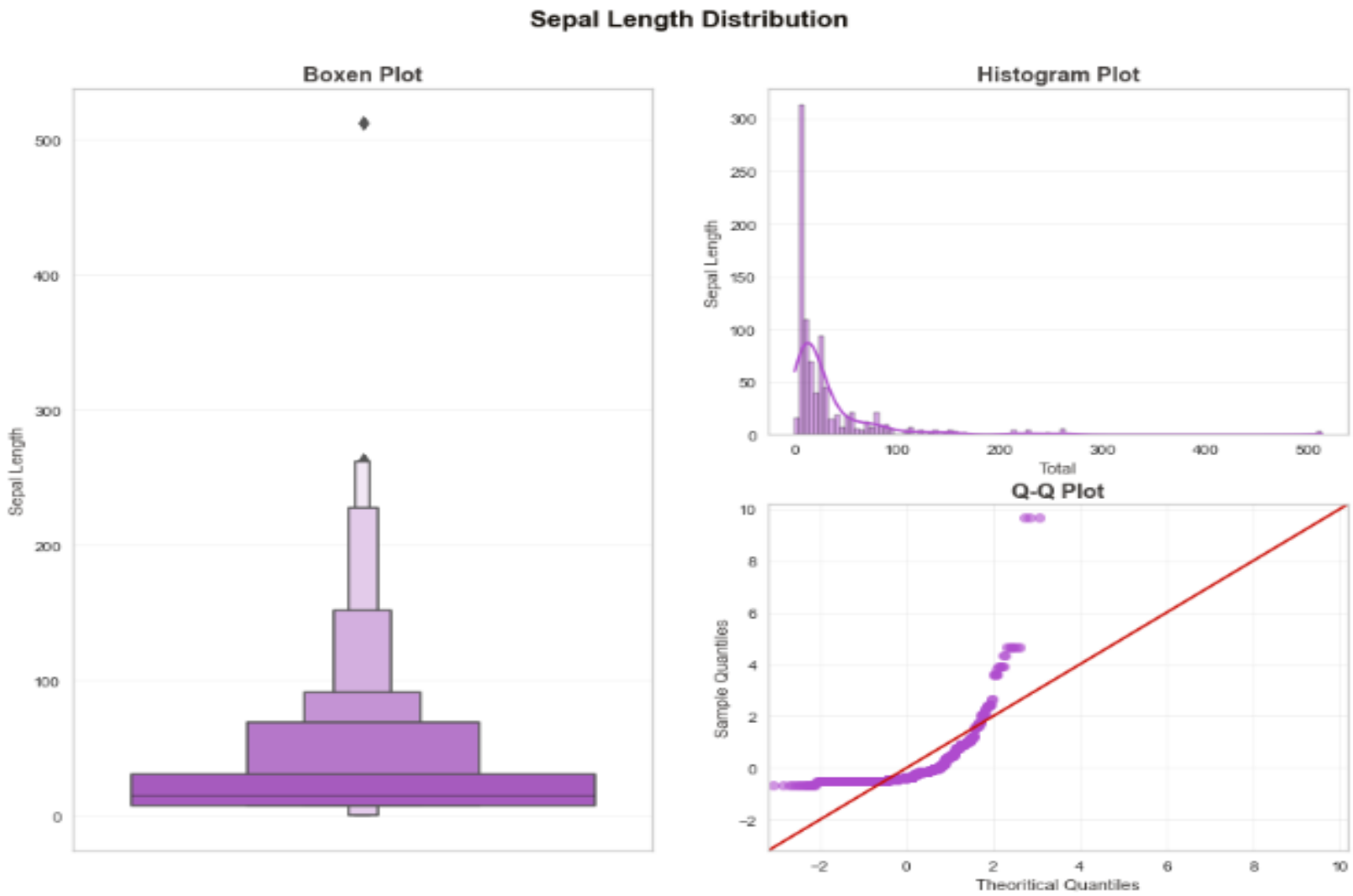
```
from yellowbrick.classifier import ConfusionMatrix
from yellowbrick.classifier import ClassPredictionError
from yellowbrick.classifier import ROCAUC
from yellowbrick.style import set_palette

from statsmodels.graphics.gofplots import qqplot

# --- Variable, Color & Plot Size ---
var = titanic['Fare']
color = color_mix[2]
fig = plt.figure(figsize = (14, 10))

# --- Skewness & Kurtosis ---
print('\033[35m\033[1m'+'.: Sepal Length Skewness & Kurtosis
:.'+'\033[0m')
print('*' * 40)
```

```
print('Skewness:'+'\033[35m\033[1m {:.3f}'.format(var.skew(axis = 0,
skipna = True)))
print('\033[0m'+'Kurtosis:'+'\033[35m\033[1m {:.3f}'.format(var.kurt(axis
= 0, skipna = True)))
print('\n')

# --- General Title ---
fig.suptitle('Sepal Length Distribution', fontweight = 'bold', fontsize =
16, fontfamily = 'sans-serif',
             color = black_grad[0])
fig.subplots_adjust(top = 0.9)

# --- Histogram ---
ax_1=fig.add_subplot(2, 2, 2)
plt.title('Histogram Plot', fontweight = 'bold', fontsize = 14, fontfamily =
'sans-serif', color = black_grad[1])
sns.histplot(data = titanic, x = var, kde = True, color = color)
```

```
plt.xlabel('Total', fontweight = 'regular', fontsize = 11, fontfamily =
'sans-serif', color = black_grad[1])
plt.ylabel('Sepal Length', fontweight = 'regular', fontsize = 11,
fontfamily = 'sans-serif', color = black_grad[1])
plt.grid(axis = 'x', alpha = 0)
plt.grid(axis = 'y', alpha = 0.2)

# --- Q-Q Plot ---
ax_2 = fig.add_subplot(2, 2, 4)
plt.title('Q-Q Plot', fontweight = 'bold', fontsize = 14, fontfamily = 'sans-
serif', color = black_grad[1])
qqplot(var, fit = True, line = '45', ax = ax_2, markerfacecolor = color,
markeredgecolor = color, alpha = 0.6)
plt.xlabel('Theoritical Quantiles', fontweight = 'regular', fontsize = 11,
fontfamily = 'sans-serif',
          color = black_grad[1])
```

```
plt.ylabel('Sample Quantiles', fontweight = 'regular', fontsize = 11,
fontfamily = 'sans-serif', color = black_grad[1])
plt.grid(axis = 'both', alpha = 0.2)

# --- Boxen Plot ---
ax_3 = fig.add_subplot(1, 2, 1)
plt.title('Boxen Plot', fontweight = 'bold', fontsize = 14, fontfamily =
'sans-serif', color = black_grad[1])
sns.boxenplot(y = var, data = titanic, color = color, linewidth = 1.5)
plt.ylabel('Sepal Length', fontweight = 'regular', fontsize = 11,
fontfamily = 'sans-serif', color = black_grad[1])
plt.grid(axis = 'y', alpha = 0.2)
plt.show();
```

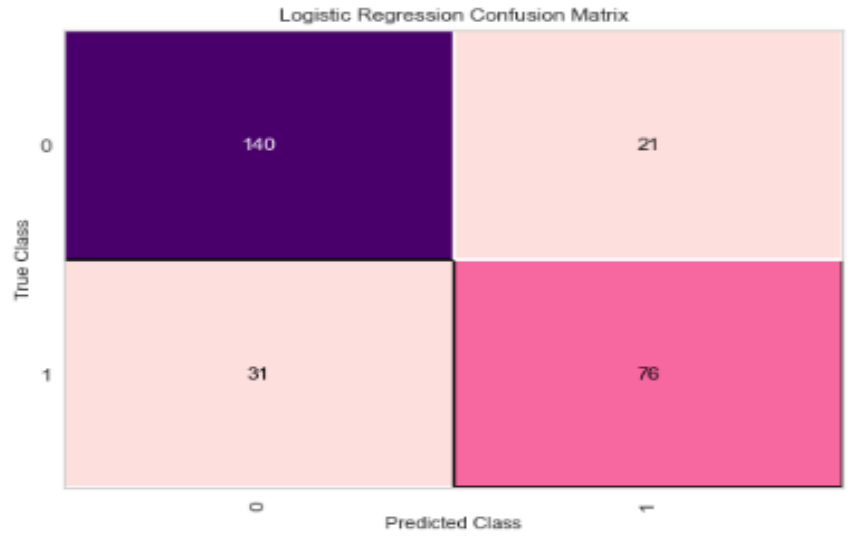
```
.: Sepal Length Skewness & Kurtosis :.
****************************************
Skewness: 4.787
Kurtosis: 33.398
```

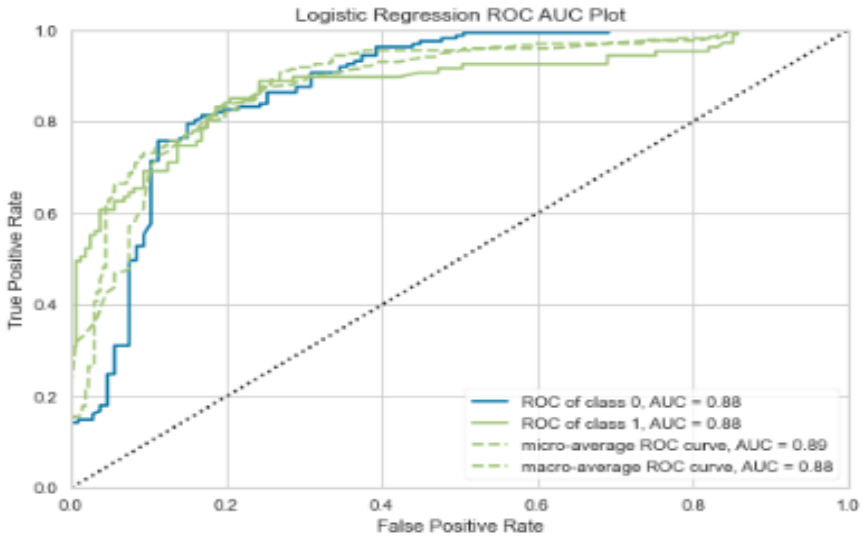


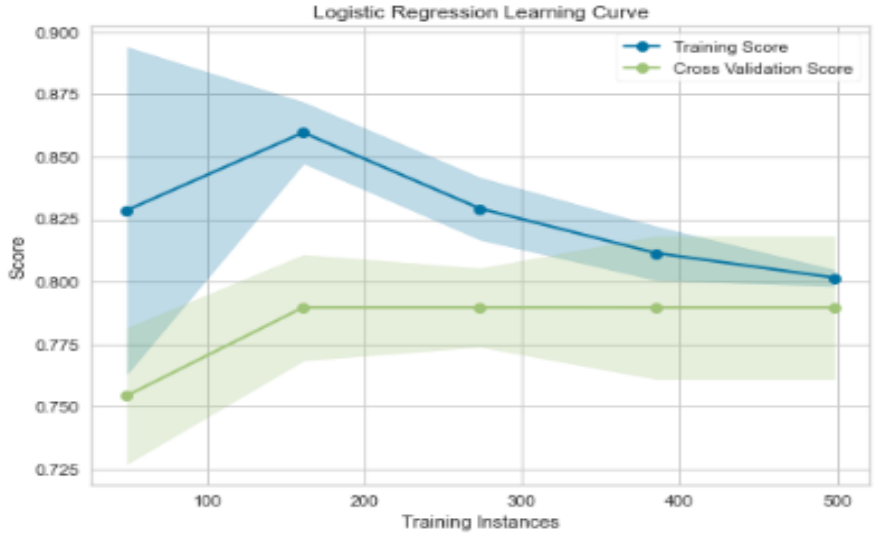
```
from yellowbrick.model_selection import LearningCurve,
FeatureImportances
from sklearn.metrics import
accuracy_score,precision_recall_curve

# --- Applying Logistic Regression ---
LRclassifier = LogisticRegression(solver='liblinear')
LRclassifier.fit(X_train, y_train)

y_pred_LR = LRclassifier.predict(X_test)

# --- LR Accuracy ---
LRAcc = accuracy_score(y_pred_LR, y_test)
print('.:. Logistic Regression Accuracy:'+'\033[35m\033[1m
{:.2f}%'.format(LRAcc*100)+' \033[0m.:.')
```

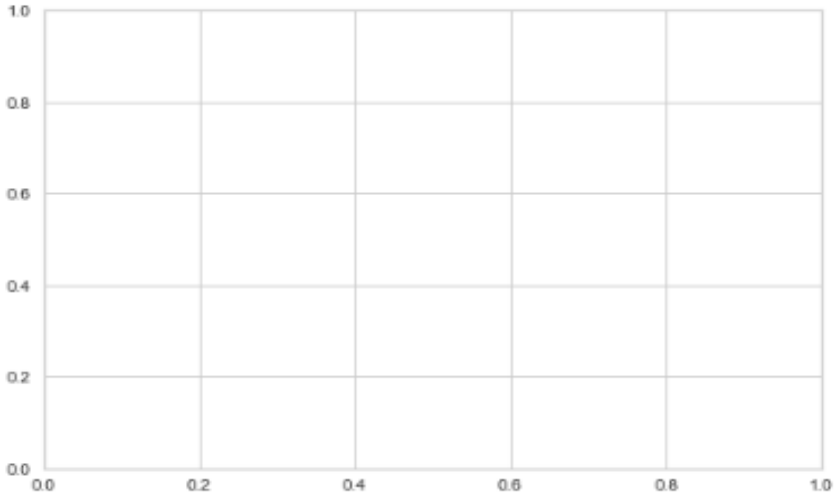
```
# --- LR Classification Report ---
print('\033[35m\033[1m\n.: Classification Report'+'\033[0m')
print('*' * 25)
print(classification_report(y_test, y_pred_LR))

# --- Performance Evaluation ---
print('\033[35m\n\033[1m'+'.: Performance
Evaluation'+'\033[0m')
print('*' * 26)
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize = (14, 10))

# --- LR Confusion Matrix ---
logmatrix = ConfusionMatrix(LRclassifier, ax=ax1,
cmap='RdPu', title='Logistic Regression Confusion Matrix')
logmatrix.fit(X_train, y_train)
logmatrix.score(X_test, y_test)
```

```
logmatrix.finalize()

# --- LR ROC AUC ---
logrocauc = ROCAUC(LRclassifier, ax = ax2, title = 'Logistic
Regression ROC AUC Plot')
logrocauc.fit(X_train, y_train)
logrocauc.score(X_test, y_test)
logrocauc.finalize()

# --- LR Learning Curve ---
loglc = LearningCurve(LRclassifier, ax = ax3, title = 'Logistic
Regression Learning Curve')
loglc.fit(X_train, y_train)
loglc.finalize()
plt.tight_layout();
```

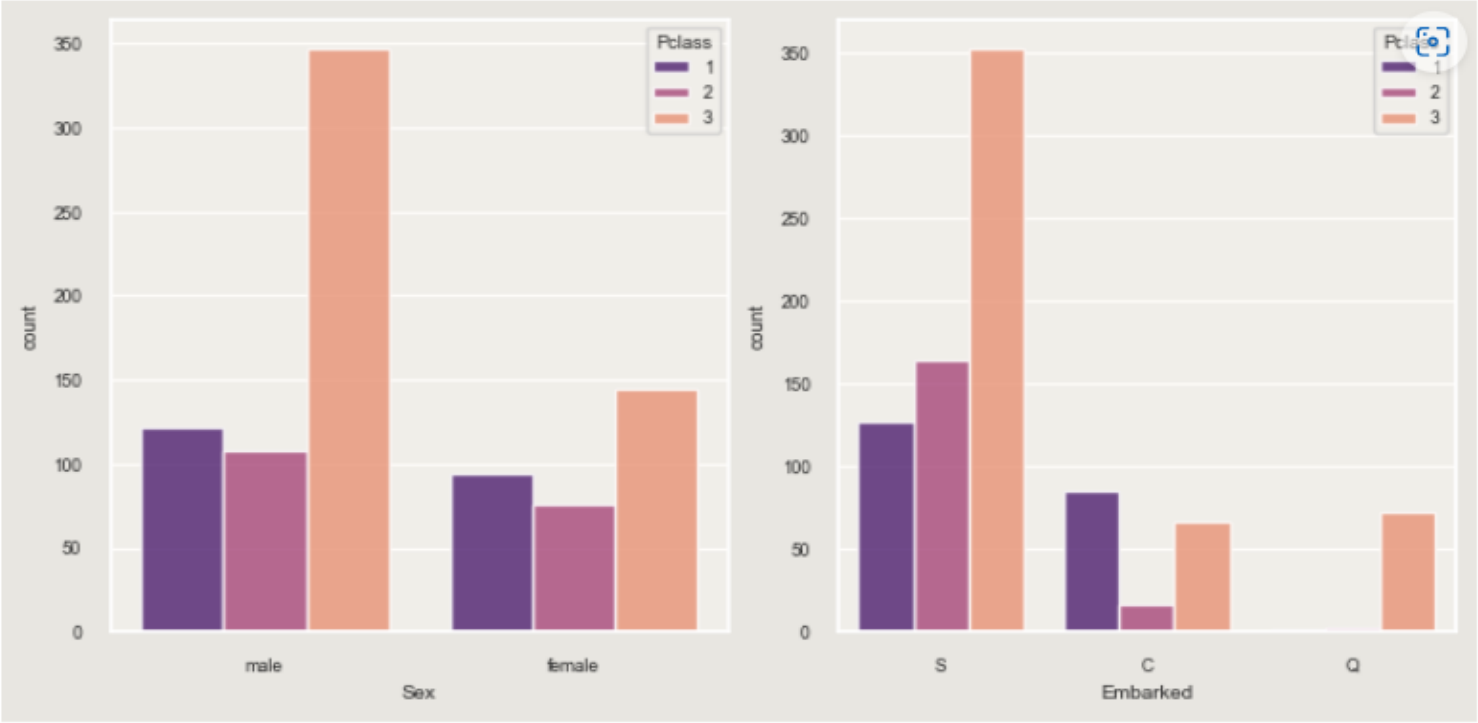

Logistic Regression Confusion Matrix
140
21
31
76
0
1
True Class
0
1
Predicted Class


Logistic Regression ROC AUC Plot
True Positive Rate
False Positive Rate
ROC of class 0, AUC = 0.88
ROC of class 1, AUC = 0.88
micro-average ROC curve, AUC = 0.89
macro-average ROC curve, AUC = 0.88


Logistic Regression Learning Curve
Training Score
Cross Validation Score
Score
Training Instances

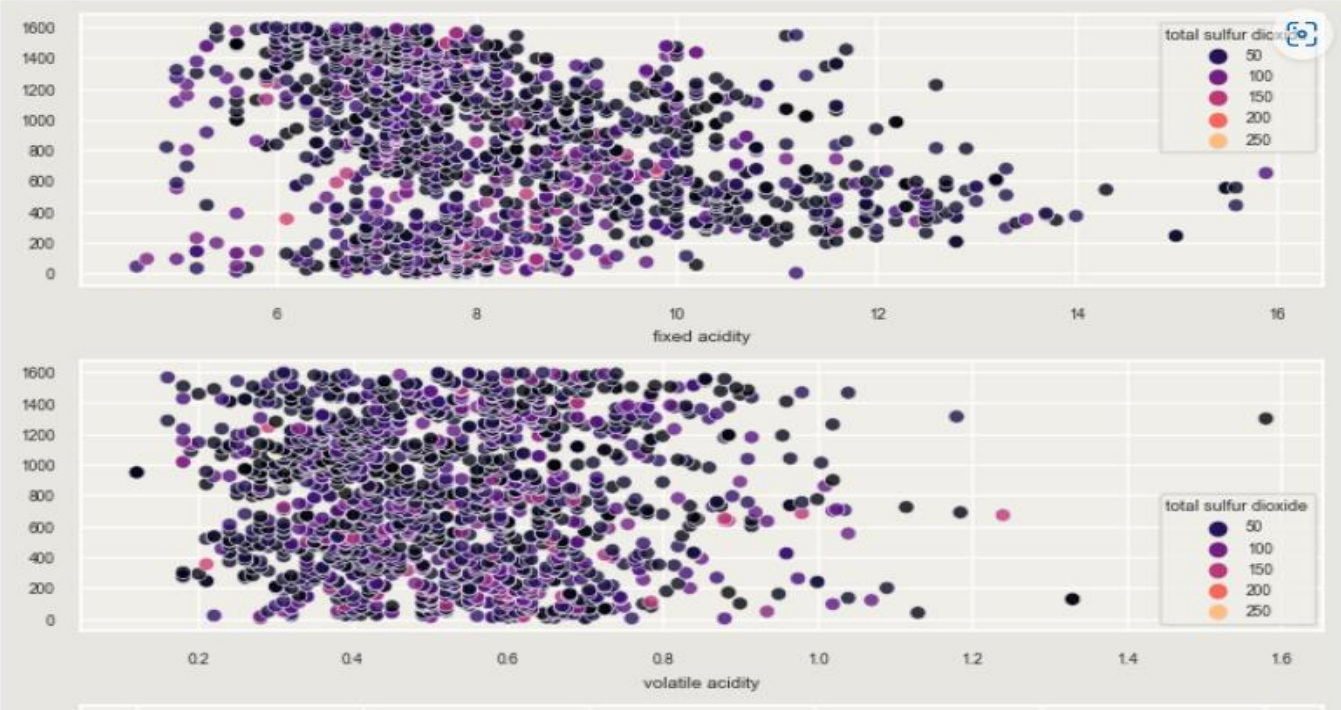


```
cat = ['Sex','Embarked']
sns.set_theme(rc = {'figure.dpi': 100, 'axes.labelsize': 7,
                    'axes.facecolor': '#f0eee9', 'grid.color': '#fffdfa',
                   'figure.facecolor': '#e8e6e1'}, font_scale = 0.55)
fig, ax = plt.subplots(5, 2, figsize = (7, 18))
for indx, (column, axes) in list(enumerate(list(zip(cat,
                                                    ax.flatten())))):

    sns.countplot(ax = axes, x = titanic[column], hue = titanic['Pclass'],
                  palette = 'magma', alpha = 0.8)

else:
    [axes.set_visible(False) for axes in ax.flatten()[indx + 1:]]
plt.tight_layout()
plt.show()
```

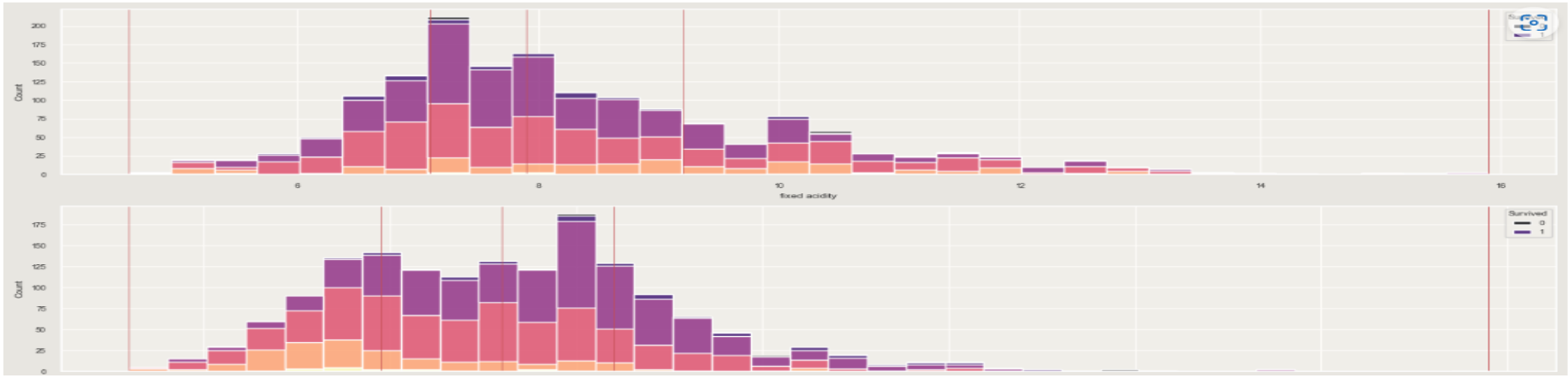



```
num = wine.select_dtypes(include="number")
fig, ax = plt.subplots(14, 1, figsize = (7, 30))
for indx, (column, axes) in list(enumerate(list(zip(num, ax.flatten())))):

    sns.scatterplot(ax = axes, y = wine[column].index, x =
wine[column],hue = wine['total sulfur dioxide'],
```

```
                palette = 'magma', alpha = 0.8)

else:
    [axes.set_visible(False) for axes in ax.flatten()[indx + 1:]]
plt.tight_layout()
plt.show()
```

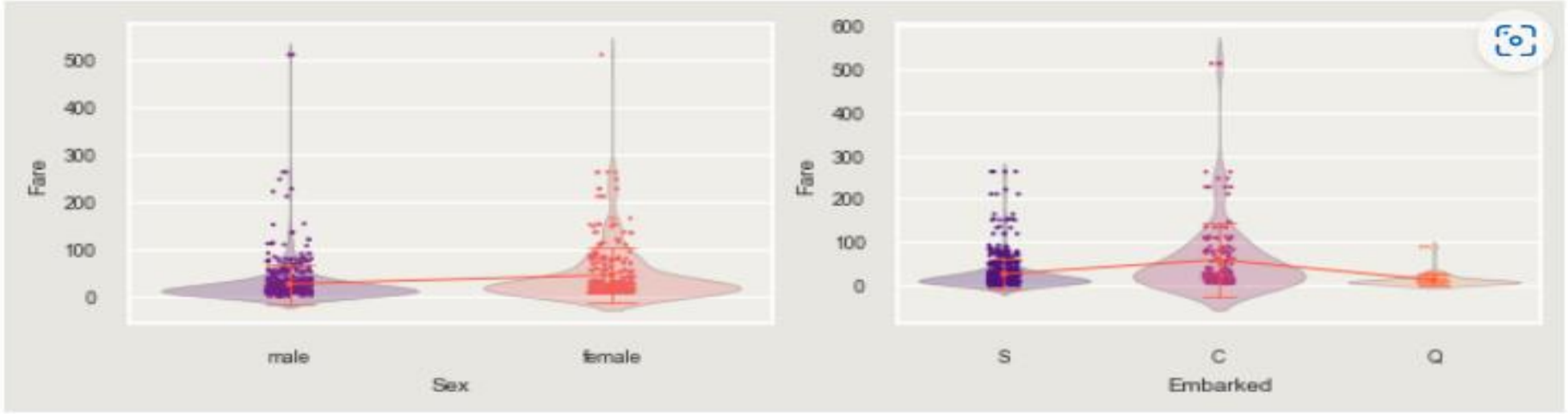


```
num = wine.select_dtypes(include="number")
fig, ax = plt.subplots(12, 1, figsize = (14, 35))
for indx, (column, axes) in list(enumerate(list(zip(num, ax.flatten())))):

    sns.histplot(ax = axes, x = wine[column],hue = wine['quality'],
                 palette = 'magma', alpha = 0.8, multiple = 'stack')
    legend = axes.get_legend() # sns.hisplot has some issues with
legend
    handles = legend.legendHandles
    legend.remove()
    axes.legend(handles, ['0', '1'], title = 'Survived', loc = 'upper right')
    Quantiles = np.quantile(wine[column], [0, 0.25, 0.50, 0.75, 1])

    for q in Quantiles: axes.axvline(x = q, linewidth = 0.5, color =
'r')
plt.tight_layout()
plt.show()
```

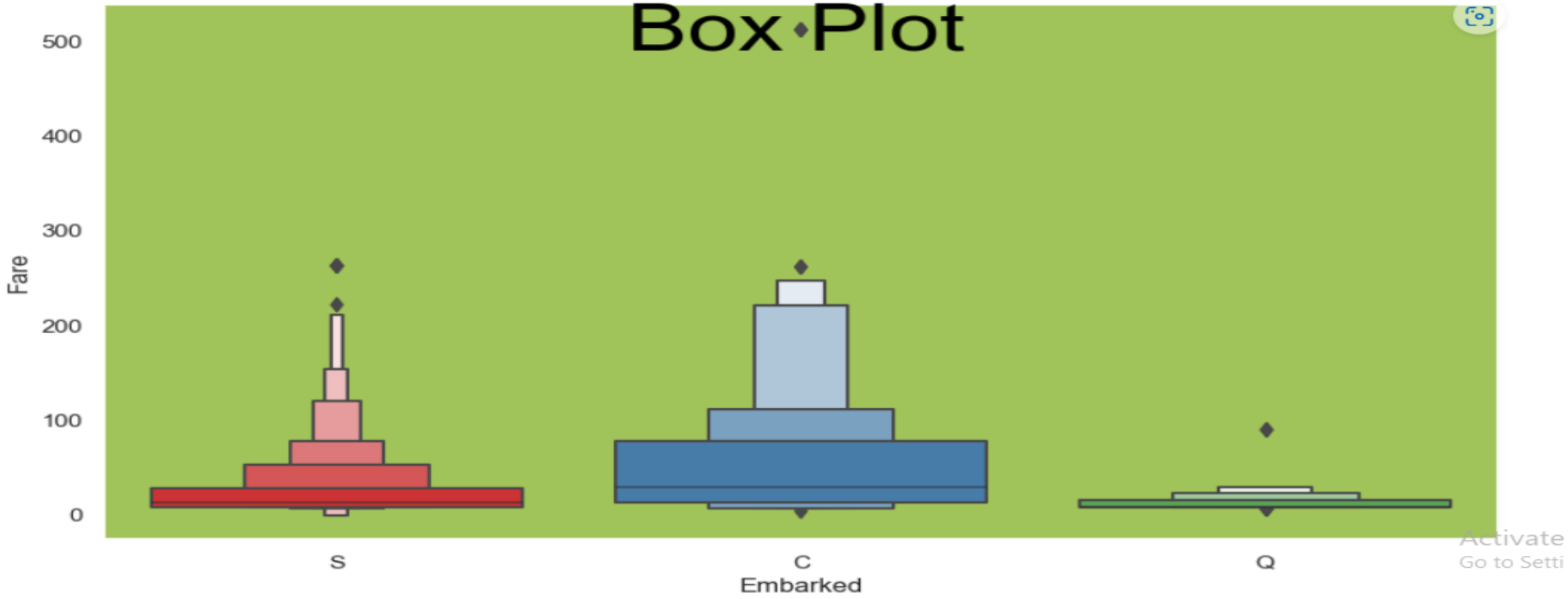



```
cat = ['Sex','Embarked']
fig, ax = plt.subplots(5, 2, figsize = (6.5, 10))
for indx, (column, axes) in list(enumerate(list(zip(cat, ax.flatten())))):
    sns.violinplot(ax = axes, x = titanic[column],
                   y = titanic['Fare'],
                   scale = 'width', linewidth = 0.5,
                   palette = 'magma', inner = None)
    plt.setp(axes.collections, alpha = 0.3)
    sns.stripplot(ax = axes, x = titanic[column],
                  y = titanic['Fare'],
                  palette = 'magma', alpha = 0.9,
                  s = 1.5, jitter = 0.07)
```

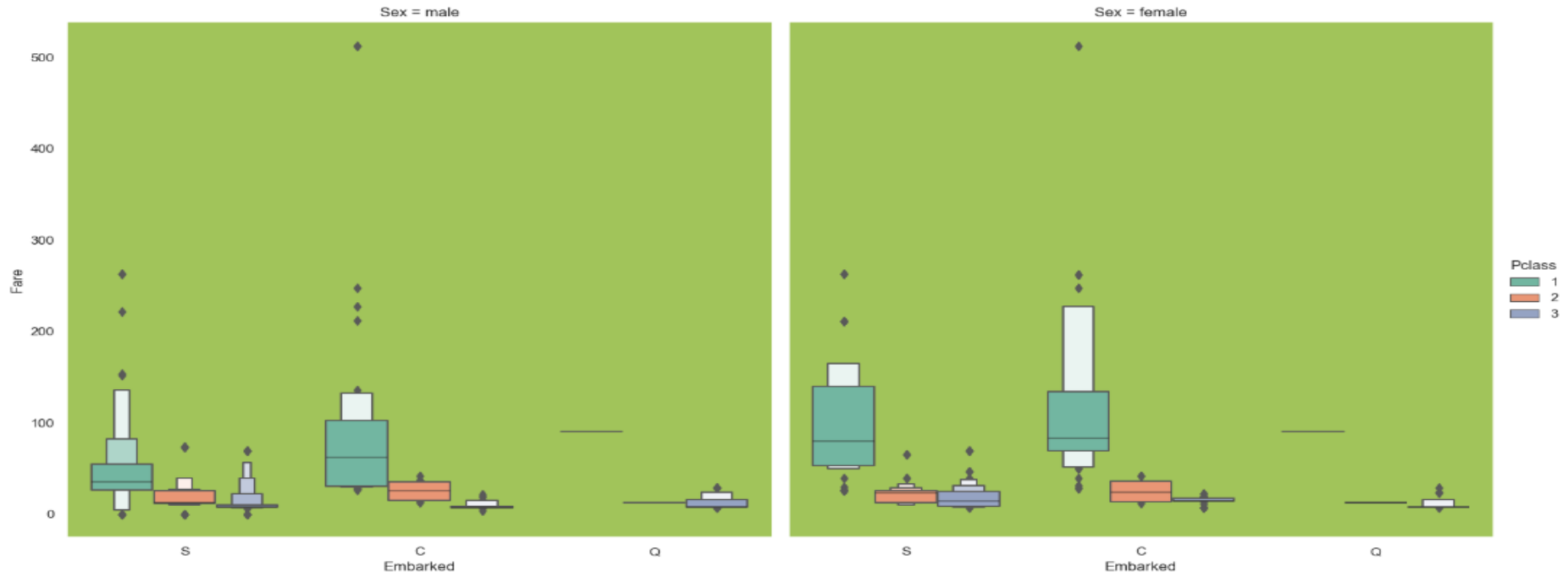
```
    sns.pointplot(ax = axes, x = titanic[column],
                  y = titanic['Fare'],
                  color = '#ff5736', scale = 0.25,
                  estimator = np.mean, ci = 'sd',
                  errwidth = 0.5, capsize = 0.15, join = True)
    plt.setp(axes.lines, zorder = 100)
    plt.setp(axes.collections, zorder = 100)
else:
    [axes.set_visible(False) for axes in ax.flatten()[indx + 1:]]
plt.tight_layout()
plt.show()
```



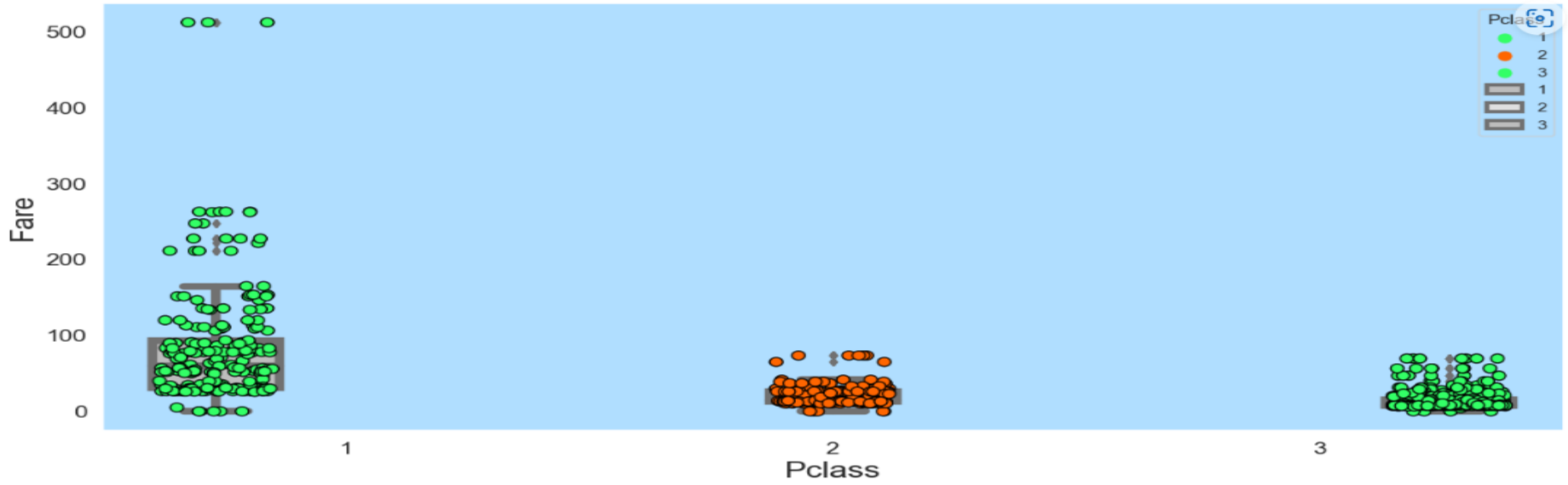
```
sns.set(rc={"axes.facecolor":"#a1c45a" , "axes.grid" : False})
plt.figure(figsize=(11,6))
plt.gcf().text(.51, .84, "Box Plot", fontsize = 40, color='Black' ,ha='center',
va='center')
sns.boxenplot(x=titanic['Embarked'] , y = titanic['Fare'],palette="Set1")
plt.show()
```



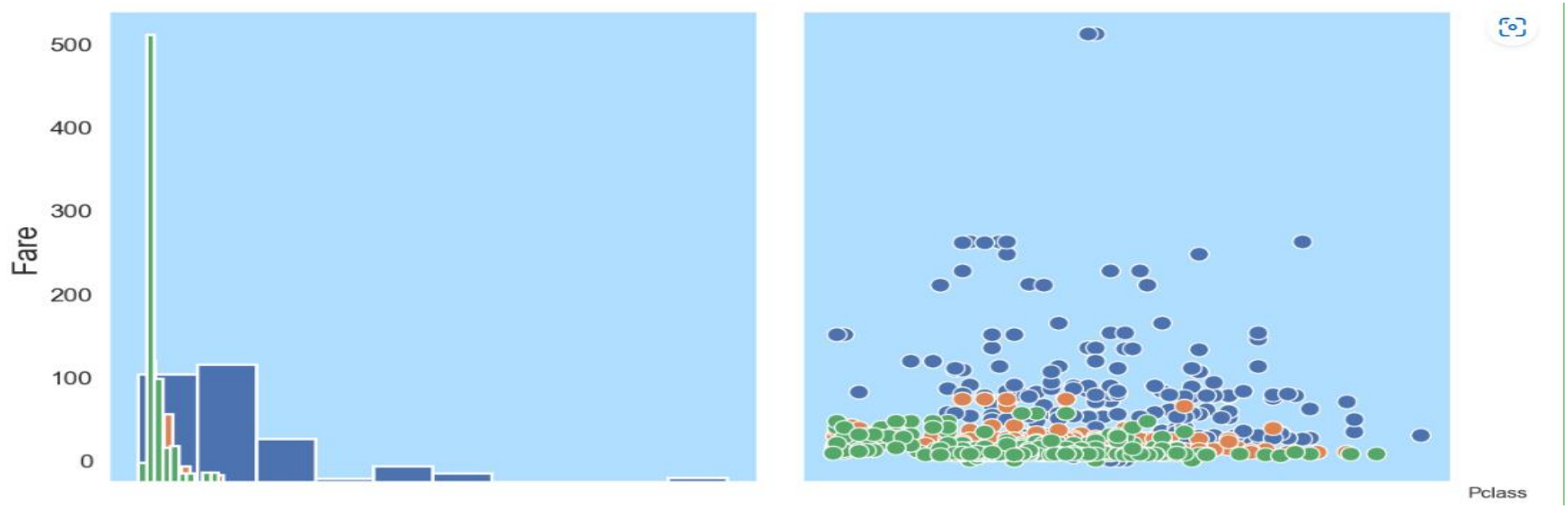
```
# Facet along the columns to show a categorical variable using "col"
plt.figure(figsize=(11,7))
sns.catplot(x="Embarked" , y = "Fare", hue= "Pclass",
            col="Sex", kind="boxen",palette="Set2" , height=8, aspect=1
,data=titanic)
plt.show();
```



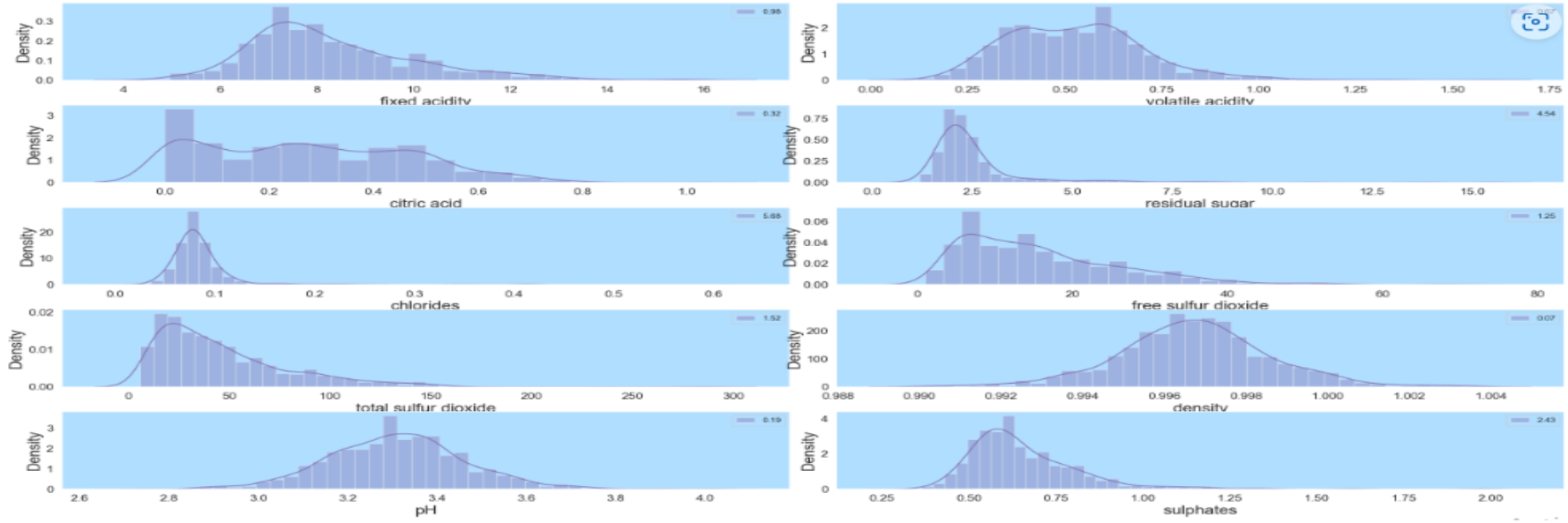
```
plt.figure(figsize=(16,7))
sns.set(rc={"axes.facecolor":"#b0deff","axes.grid":False,
            'xtick.labelsize':15,'ytick.labelsize':15,
            'axes.labelsize':20,'figure.figsize':(20.0, 9.0)})
params = dict(data=titanic ,x = titanic.Pclass ,y = titanic.Fare
,hue=titanic.Pclass,dodge=True)
sns.stripplot(**params ,
size=8,jitter=0.35,palette=['#33FF66','#FF6600'],edgecolor='black',linewidth=1)
sns.boxplot(**params ,palette=['#BDBDBD','#E0E0E0'],linewidth=6)
plt.show()
```



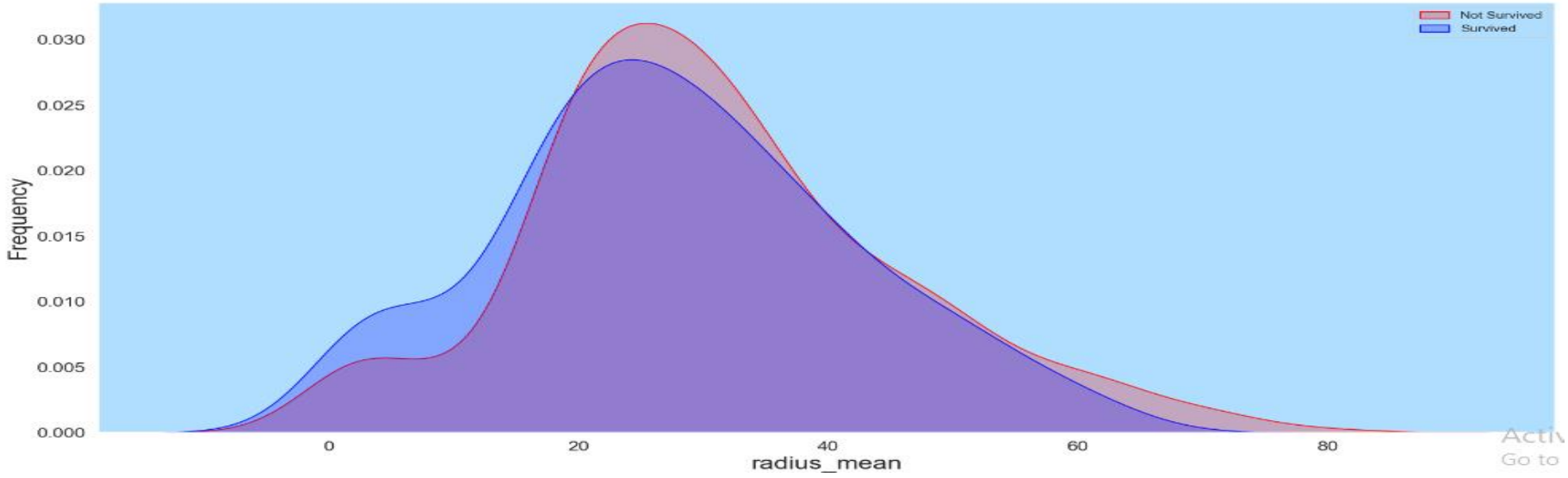
```
# Plot a subset of variables
g = sns.PairGrid(titanic, hue='Pclass' ,x_vars=["Fare" , "Age"],y_vars=["Fare" ,
"Age"],
                 height=6, aspect=1)
g = g.map_offdiag(plt.scatter , edgecolor="w", s=130)
g = g.map_diag(plt.hist , edgecolor ='w', linewidth=2)
g = g.add_legend()
plt.show()
```



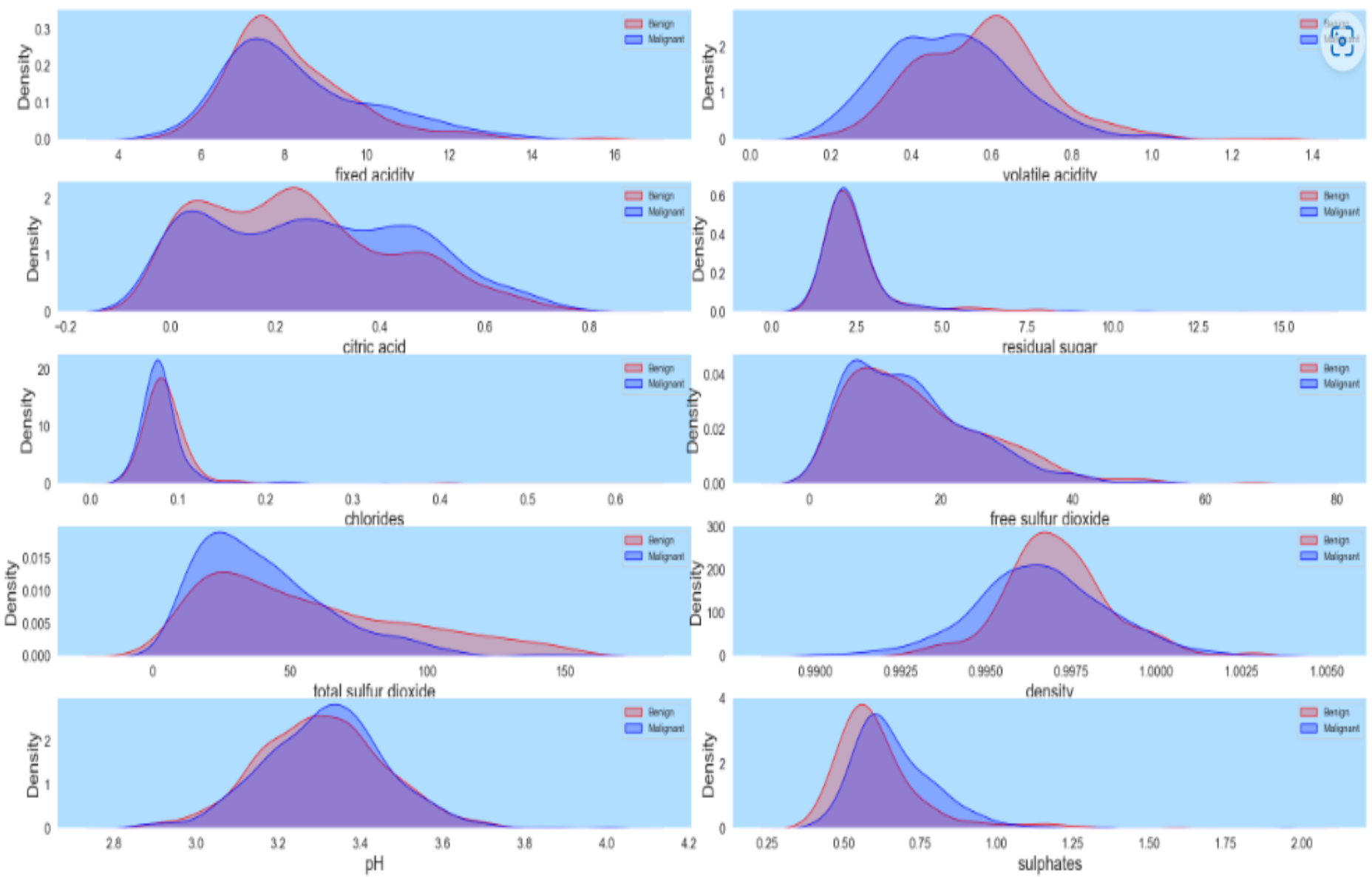
```
df = pd.read_csv("winequality-red.csv")
features_mean= list(df.columns[:10])
num_rows, num_cols = 5,2
fig, axes = plt.subplots(num_rows, num_cols, figsize=(25, 12))
fig.tight_layout()
for index, column in enumerate(df[features_mean].columns):
    i,j = (index // num_cols, index % num_cols)
    g = sns.distplot(df[column], color="m", label="%.2f"%(df[column].skew()),
ax=axes[i,j])
    g = g.legend(loc="best")
```



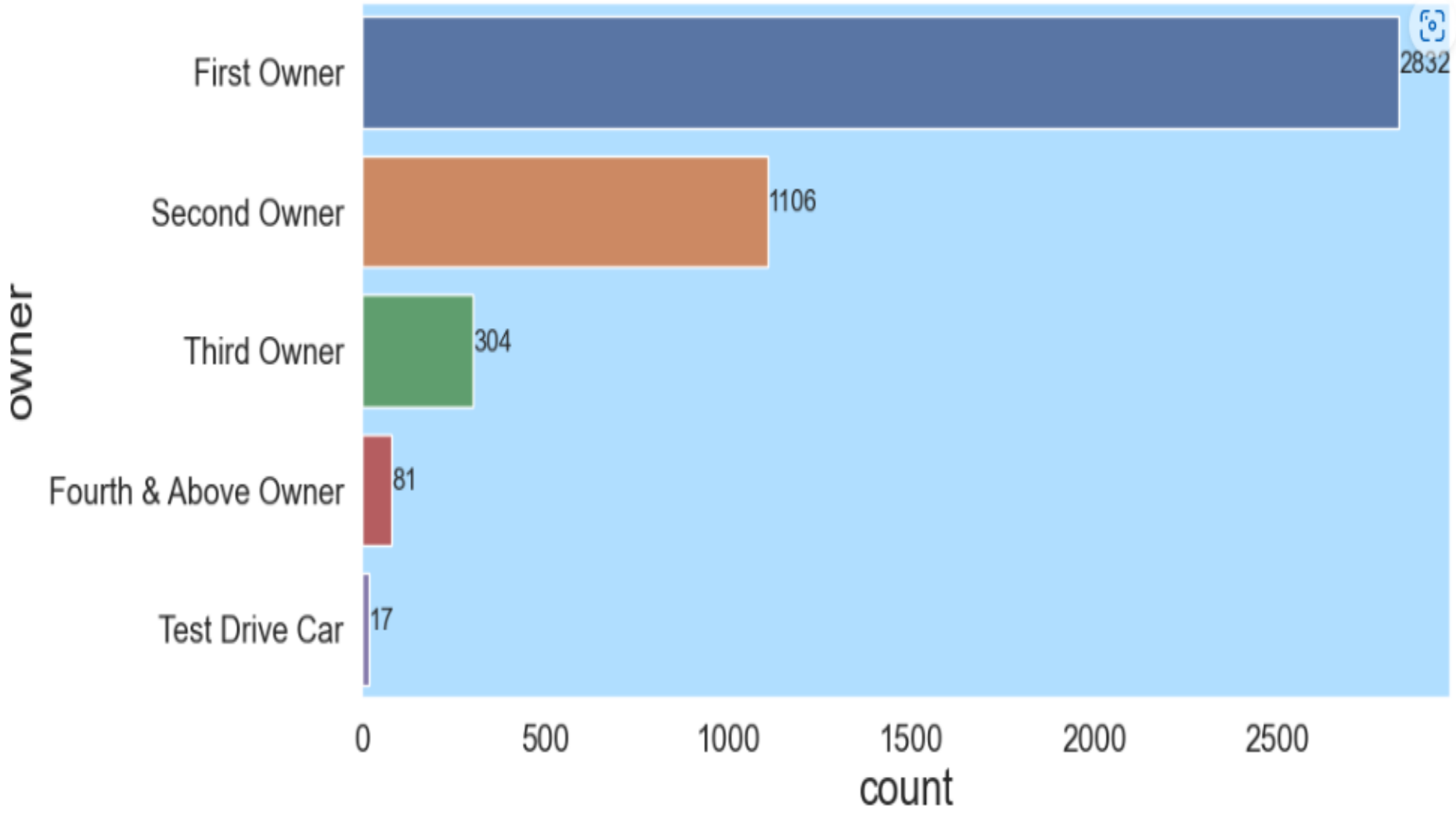
```
y = df['Sex']

# Explore Age distibution
g = sns.kdeplot(df["Age"][(y == 'male') & (df["Age"].notnull())], color="Red",
shade=True)
g = sns.kdeplot(df["Age"][(y == 'female') & (df["Age"].notnull())], ax=g,
color="Blue", shade=True)
g.set_xlabel("radius_mean")
g.set_ylabel("Frequency")
g = g.legend(["Not Survived","Survived"])
```



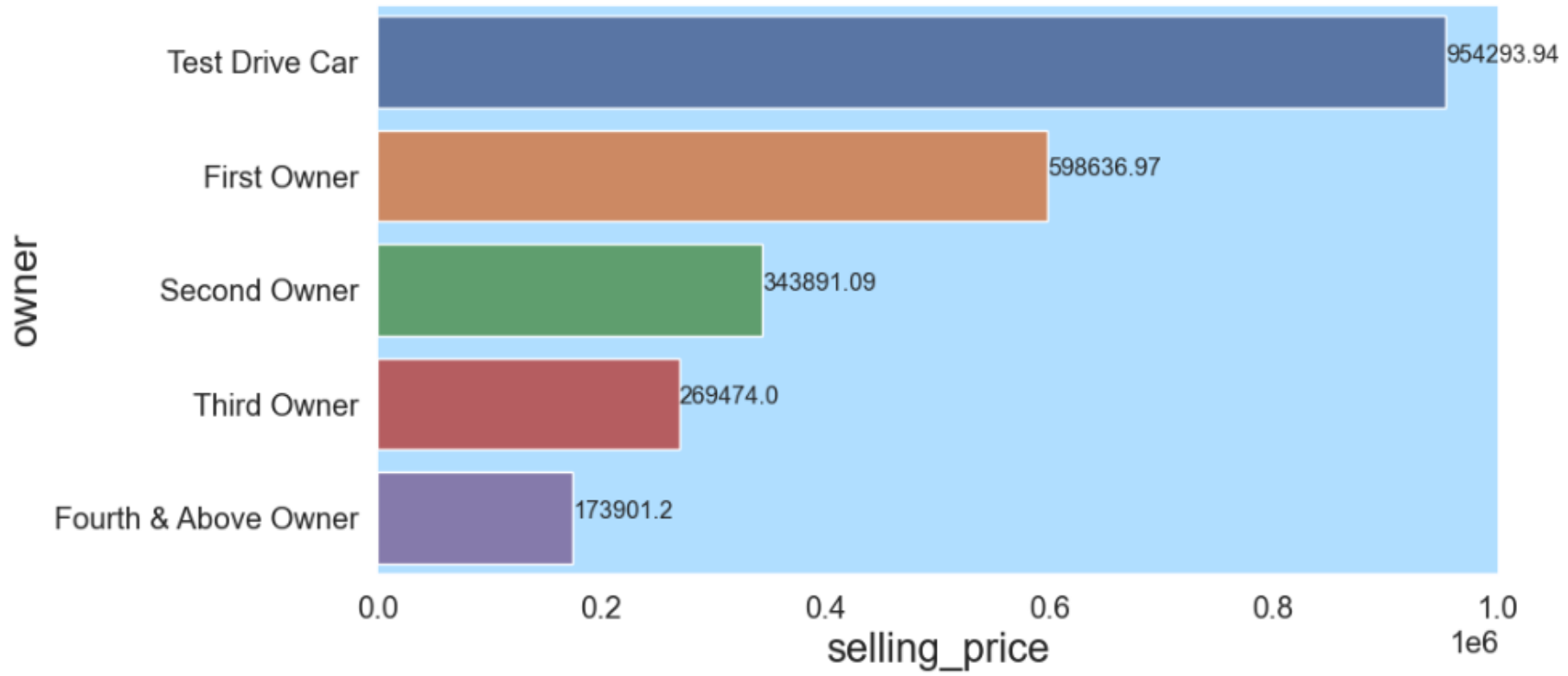
```
df = pd.read_csv("winequality-red.csv")
features_mean= list(df.columns[:10])
df_b = df[df['quality'] == 5]
df_m = df[df['quality'] == 6]
num_rows, num_cols = 5,2
fig, axes = plt.subplots(num_rows, num_cols, figsize=(25, 12))
fig.tight_layout()
for index, column in enumerate(df[features_mean].columns):
    i,j = (index // num_cols, index % num_cols)
    g = sns.kdeplot(df_b[column], color="Red", shade=True,
ax=axes[i,j])
    g = sns.kdeplot(df_m[column], ax=g, color="Blue", shade=True)
    g.set_xlabel(column)
    g = g.legend(["Benign","Malignant"])
```


Density
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
Benign
Malignant

```
raw_df = raw_df [['name', 'year', 'selling_price', 'km_driven',
'fuel', 'seller_type',
       'transmission', 'owner']]
# Function to print width of barcharts on the bars
def barw(ax):
    for p in ax.patches:
        val = p.get_width() #height of the bar
        x = p.get_x()+ p.get_width() # x- position
        y = p.get_y() + p.get_height()/2 #y-position
        ax.annotate(round(val,2),(x,y))
plt.figure(figsize=(10,5))
ax0 = sns.countplot(data = raw_df, y ='owner', order =
raw_df['owner'].value_counts().index)
barw(ax0)
plt.show()
```

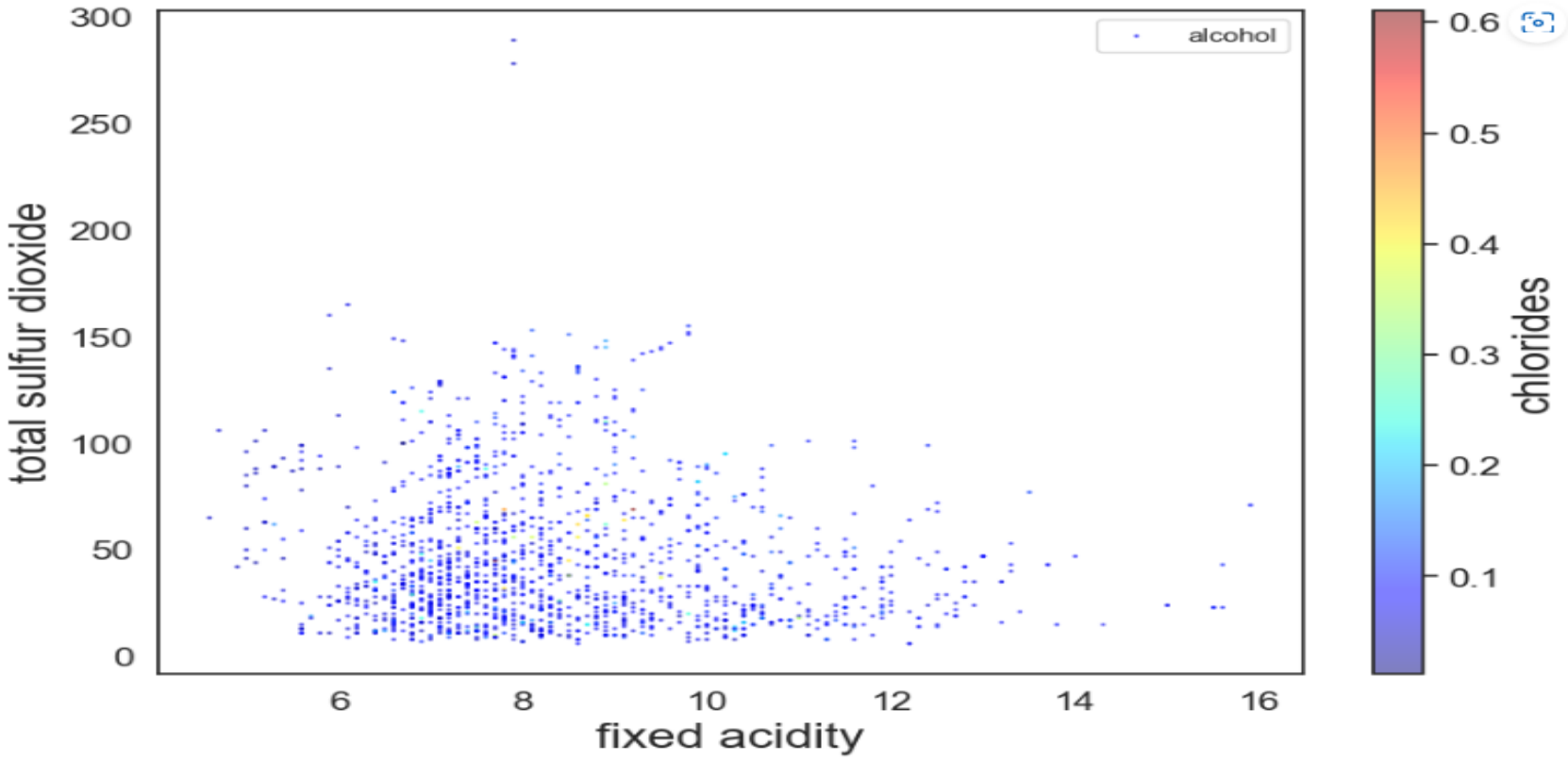

First Owner
Second Owner
Third Owner
Fourth & Above Owner
Test Drive Car
owner
2832
1106
304
81
17
0
500
1000
1500
2000
2500
count

```
raw_df = pd.read_csv('datasets_33080_1320127_CAR DETAILS FROM
CAR DEKHO.csv')
raw_df = raw_df [['name', 'year', 'selling_price', 'km_driven', 'fuel',
'seller_type',
       'transmission', 'owner']]
df_gc = raw_df.groupby('owner').mean()
df_gc.reset_index(inplace= True)
df_gc[['owner','selling_price']].sort_values('selling_price', ascending
=False)

plt.figure(figsize=(10,5))
ax1 = sns.barplot(data = raw_df, x='selling_price', y ='owner', order =
df_gc.sort_values('selling_price',ascending =False)['owner'], ci =None)
barw(ax1)
plt.show()
```

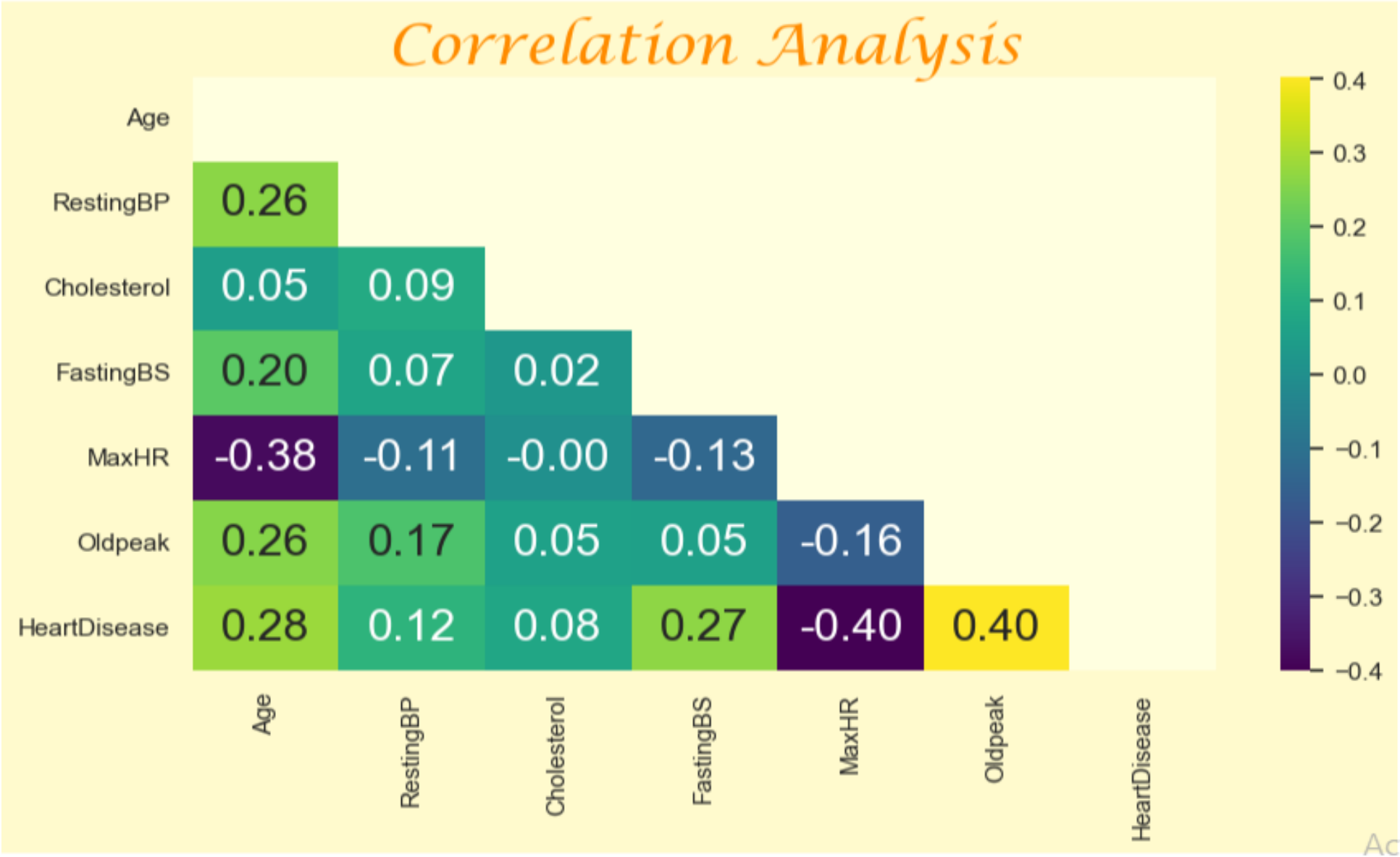


```
sns.set_style('white')
df.plot(kind="scatter", x="fixed acidity", y="total sulfur dioxide",
alpha=.5,
        s=df["alcohol"]/10, label="alcohol", figsize=(10,7),
        c="chlorides", cmap=plt.get_cmap("jet"), colorbar=True,
```

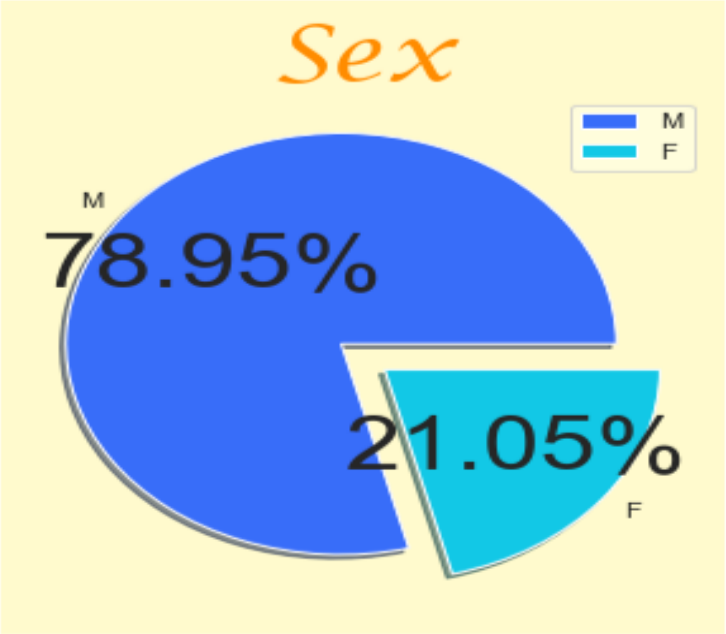
```
            sharex=False)
plt.legend()
plt.show()
```



```
matplotlib.rcParams.update({'font.size': 20})
corr = heart.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
plt.figure(dpi=100)
plt.title('Correlation Analysis',
          fontsize=25,
          color='DarkOrange',
          font='Lucida Calligraphy')
sns.heatmap(corr,
            mask=mask,
            annot=True,
            lw=0,
            linecolor='white',
            cmap='viridis',
            fmt="0.2f")
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.show()
```


Correlation Analysis
Age
RestingBP
Cholesterol
FastingBS
MaxHR
Oldpeak
HeartDisease
0.26
0.05 0.09
0.20 0.07 0.02
-0.38 -0.11 -0.00 -0.13
0.26 0.17 0.05 0.05 -0.16
0.28 0.12 0.08 0.27 -0.40 0.40
Age
RestingBP
Cholesterol
FastingBS
MaxHR
Oldpeak
HeartDisease
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4

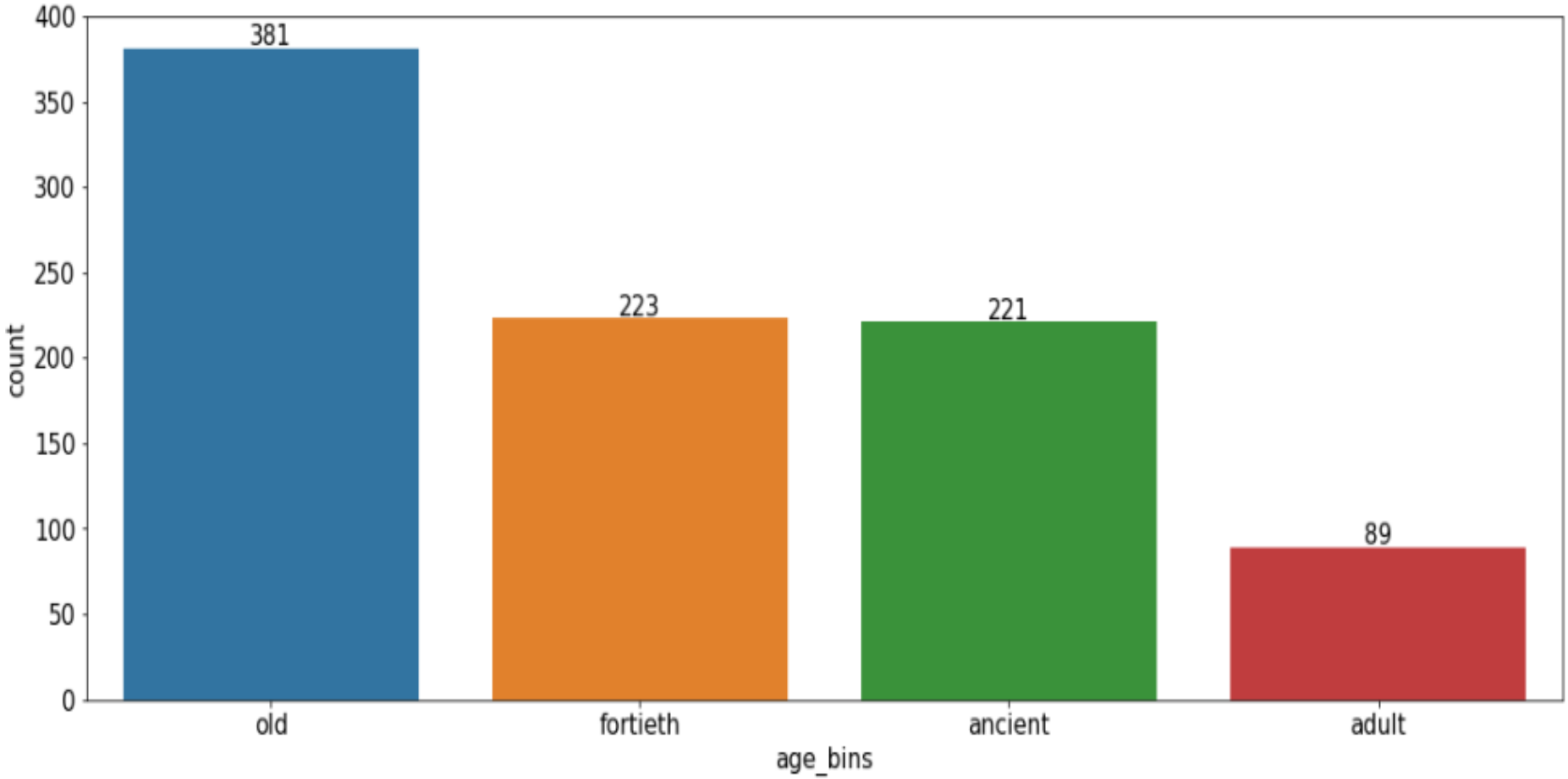
```
matplotlib.rcParams.update({'font.size': 40})
ax=heart['Sex'].value_counts().plot.pie(explode=[0.1,
0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "Sex", fontsize =
40,color='DarkOrange',font='Lucida Calligraphy');
plt.legend(labels=['M','F'])
plt.axis('off');
```



```
heart["age_bins"]= pd.cut(heart["Age"] , bins=[29 , 40 , 50 , 60
,  80] , labels=["adult" , "fortieth" , "old" , "ancient"] )
def count_plot(data  , x=None , y=None  , figsize =None , title
=None , color =None , prop=False , rotation_x =0 ):
    if x is None and y is None :
        raise("Expected y or x")
    if x is not None and y is not None:
        raise("Expected y or x not both")
    count_type = data[y if x is None else
x].value_counts(ascending =False)
    Sum = count_type.sum()
    type_order = count_type.index
    plt.figure(figsize=figsize if figsize is None else (12 , 7))
    if x is None:
        sns.countplot(data = data , y=y , color = color
,order=type_order)
```

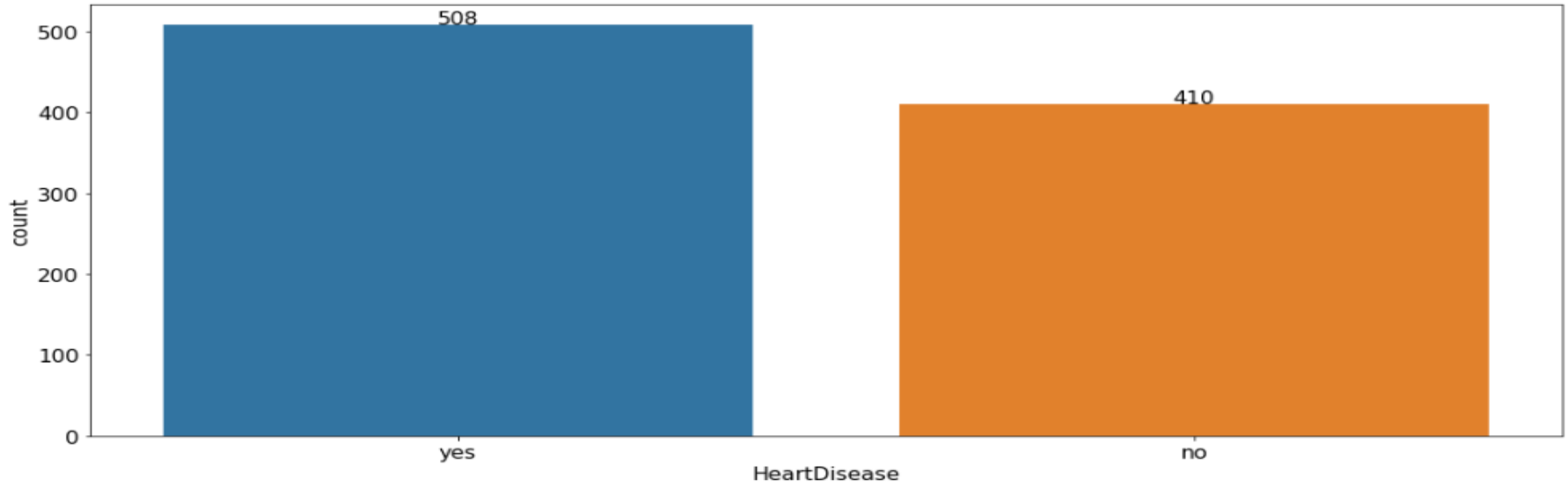
```
        if prop==True:
            for i in range(len(count_type)):
                count = count_type[i]
                pct_string ="{:0.1f}%".format(100*count/Sum)
                plt.text(count+1 , i , pct_string , va="center")
        if prop==False:
            for i in range(len(count_type)):
                count = count_type[i]
                pct_string ="{}".format(count)
                plt.text(count+1 , i , pct_string , va="center")
        plt.title(title)
        plt.show()
    if y is None :
        sns.countplot(data = data , x = x , color = color , order =
type_order)
        locs , labels =plt.xticks(rotation = rotation_x)
```

```
        if prop == True :
            for loc , label in zip(locs , labels):
                count = count_type[label.get_text()]
                pct_string ="{:0.1f}%".format(100*count/Sum)
                plt.text(loc , count+2 ,pct_string,ha ="center")
        if prop==False :
            for loc , label in zip(locs , labels):
                count = count_type[label.get_text()]
                pct_string ="{}".format(count)
                plt.text(loc , count+2 ,pct_string,ha ="center")
        plt.title(title)
        plt.show()


count_plot(data = heart , x ="age_bins")
```

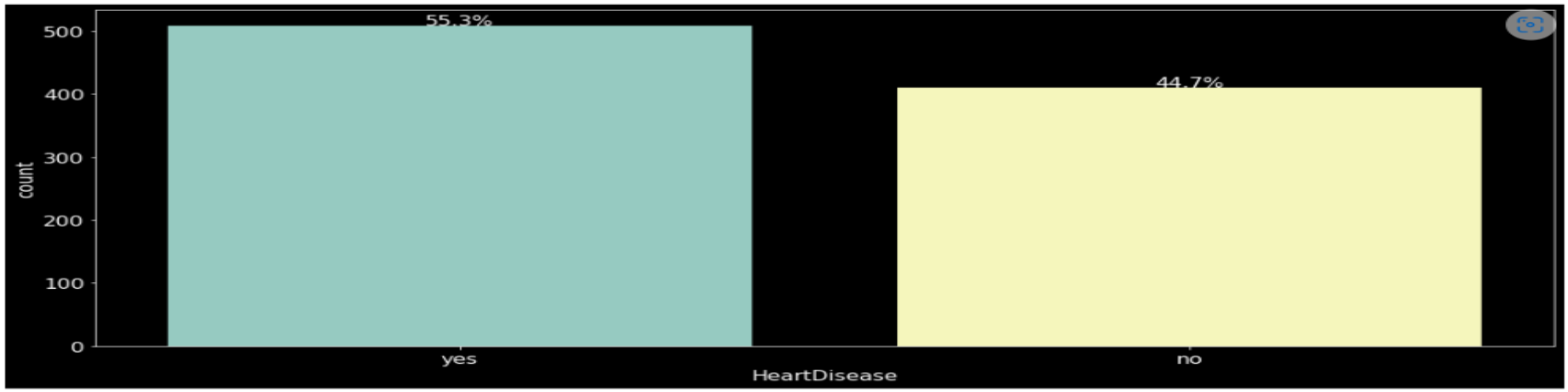



```
heart.rename(columns={"target":"have disease"} , inplace=True)
heart.replace({1:"yes" , 0:"no"} , inplace =True)
```

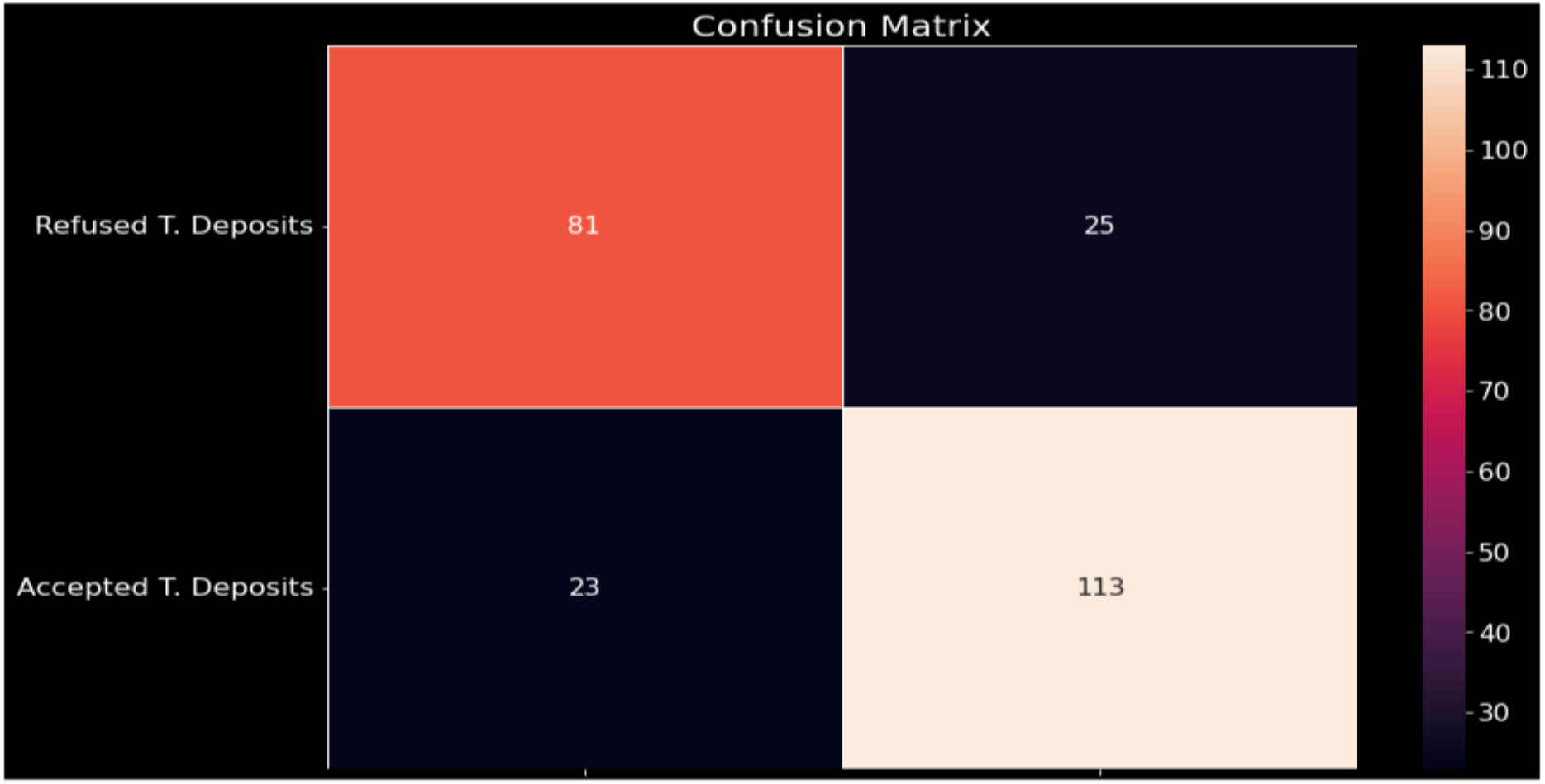
count_plot(data = heart , x ="HeartDisease")



count_plot(data = heart , x ="HeartDisease" , prop=True)



```
from sklearn.metrics import confusion_matrix
# 4697: no's, 4232: yes
conf_matrix = confusion_matrix(y_train, y_train_pred)
f, ax = plt.subplots(figsize=(12, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", linewidths=.5,
ax=ax)
plt.title("Confusion Matrix", fontsize=20)
plt.subplots_adjust(left=0.15, right=0.99, bottom=0.15,
top=0.99)
ax.set_yticks(np.arange(conf_matrix.shape[0]) + 0.5,
minor=False)
ax.set_xticklabels("")
ax.set_yticklabels(['Refused T. Deposits', 'Accepted T.
Deposits'], fontsize=16, rotation=360)
plt.show()
```




```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
lr = LogisticRegression()
```

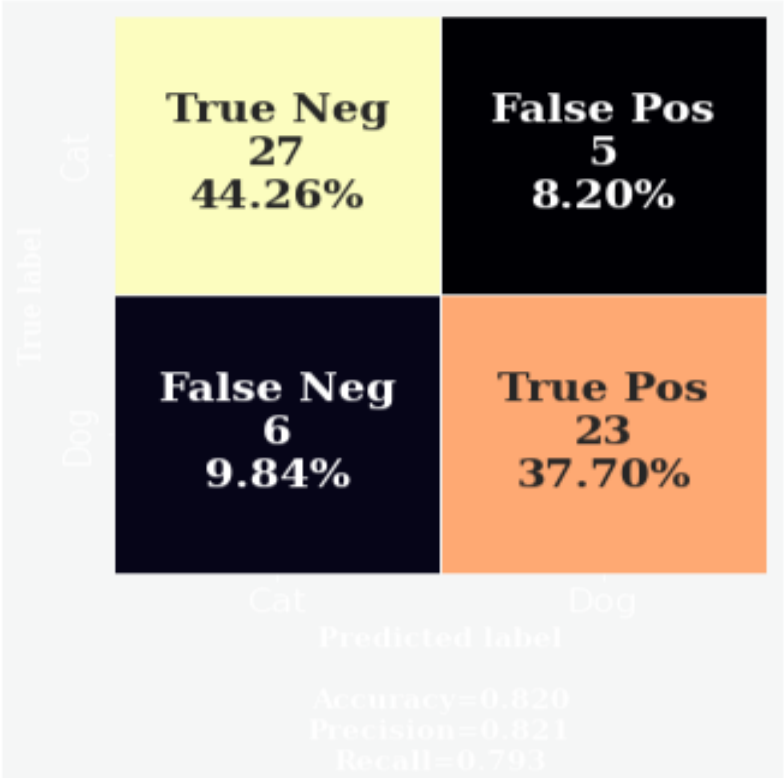
```
lr.fit(X_train,y_train)
y_pred_lr = lr.predict(X_test)
confusion_matrix(y_test,y_pred_lr)

def make_confusion_matrix(cf,
                          group_names=None,
                          categories='auto',
                          count=True,
                          percent=True,
                          cbar=True,
                          xyticks=True,
                          xyplotlabels=True,
                          sum_stats=True,
                          figsize=None,
                          cmap='Blues',
                          title=None):


    # CODE TO GENERATE TEXT INSIDE EACH SQUARE
    blanks = ['' for i in range(cf.size)]

    if group_names and len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
```

```
    else:
        group_labels = blanks

    if count:
        group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
    else:
        group_counts = blanks

    if percent:
        group_percentages = ["{0:.2%}".format(value) for value in cf.flatten()/np.sum(cf)]
    else:
        group_percentages = blanks

    box_labels = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in
zip(group_labels,group_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])


    # CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
    if sum_stats:
        #Accuracy is sum of diagonal divided by total observations
        accuracy  = np.trace(cf) / float(np.sum(cf))
```

```
        #if it is a binary confusion matrix, show some more stats
        if len(cf)==2:
            #Metrics for Binary Confusion Matrices
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.3f}\nF1
Score={:0.3f}".format(
                accuracy,precision,recall,f1_score)
        else:
            stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    else:
        stats_text = ""


    # SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
    if figsize==None:
        #Get default figure size if not set
        figsize = plt.rcParams.get('figure.figsize')

    if xyticks==False:
        #Do not show categories if xyticks is False
```

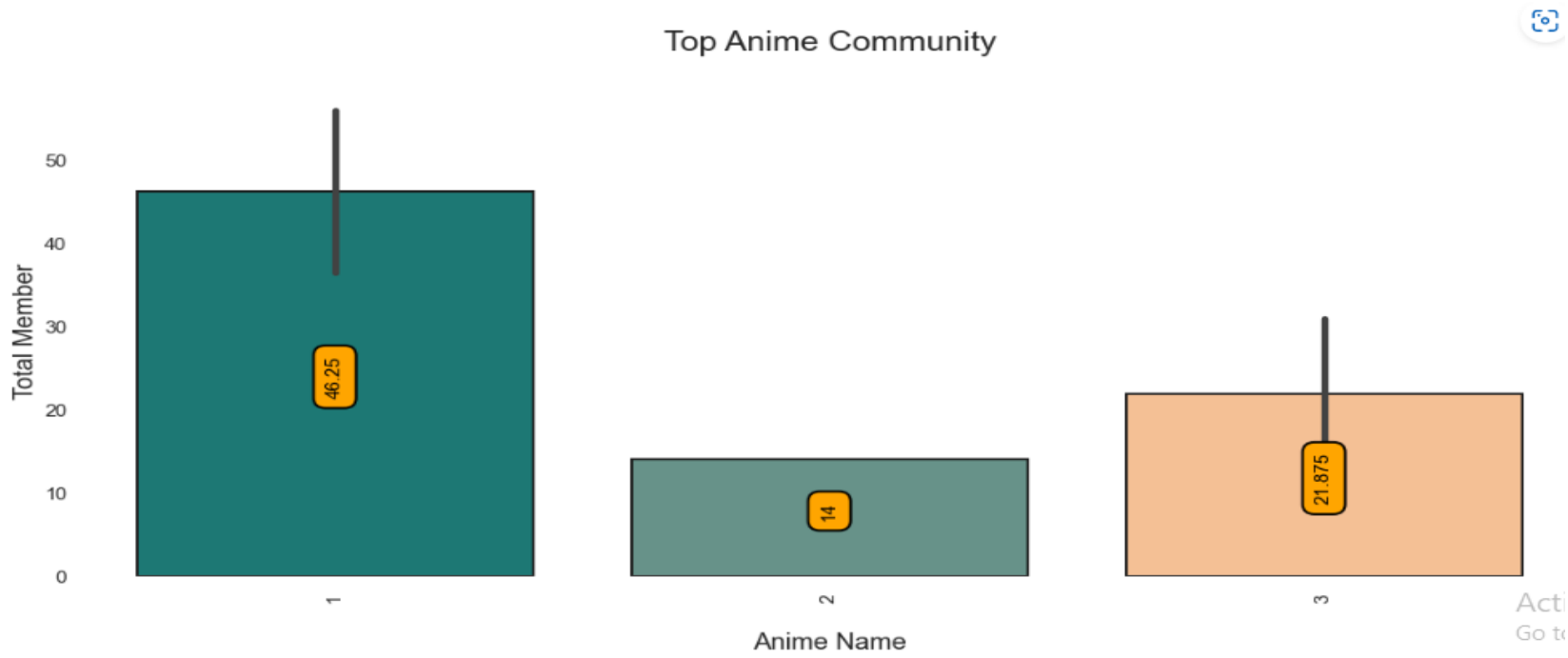
```
        categories=False

          # MAKE THE HEATMAP VISUALIZATION
    fig = plt.figure(figsize=figsize)
    fig.patch.set_facecolor('#f5f6f6')
    sns.heatmap(cf,annot=box_labels,fmt="",linewidths = 1,square = True,linecolor=
'#f5f6f6',
                cmap=cmap,cbar=cbar,annot_kws={'fontfamily':'serif','size':18,'weight':'bold'},
                xticklabels=categories,
                yticklabels=categories,)

    if xyplotlabels:
        plt.ylabel('True label', **{'fontfamily':'serif','size':12,'weight':'bold'})
        plt.xlabel('Predicted label' + stats_text,**{'fontfamily':'serif','size':12,'weight':'bold'})
    else:
        plt.xlabel(stats_text,**{'fontfamily':'serif','size':12,'weight':'bold'})



vani_cf_matrix = confusion_matrix(y_test,y_pred_lr)
my_cols = [colors[3],colors[2]]

labels = [ 'True Neg','False Pos','False Neg','True Pos']
```

```
categories = ['Cat', 'Dog']
make_confusion_matrix(vani_cf_matrix,figsize = (10,5),group_names=labels,cbar =
False,cmap = 'magma',categories=categories,
                      title = 'Vanila CNN comfusion matrix')


plt.show()
```



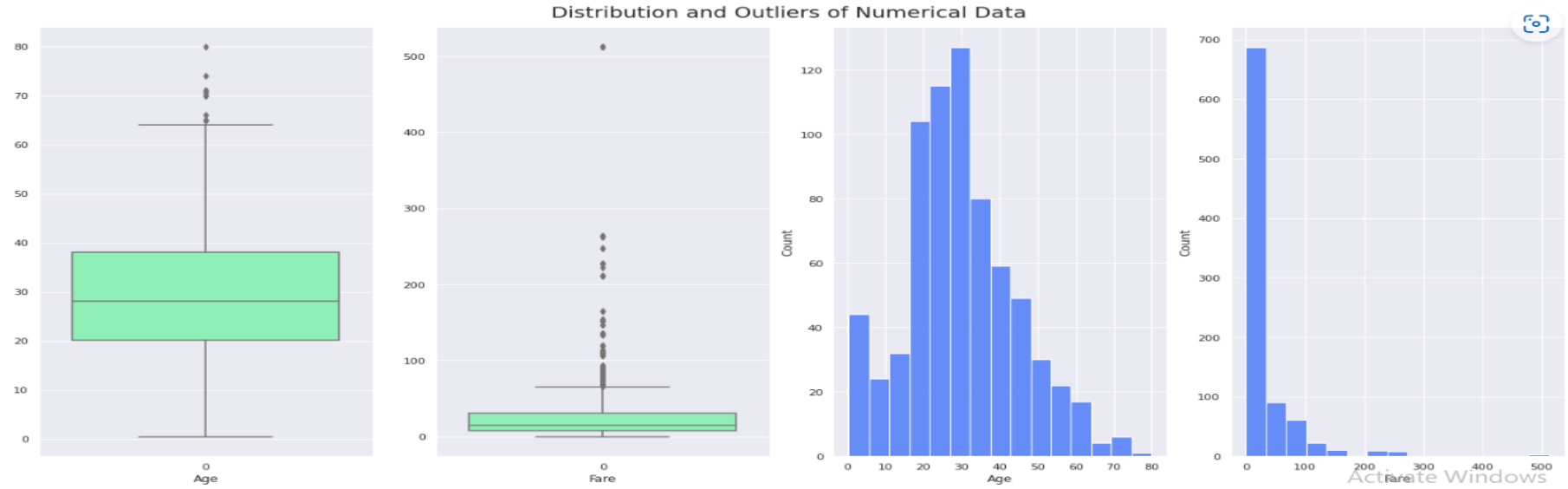
```
sns.set_style("white")
sns.set_context("poster",font_scale = .7)
palette =
["#1d7874","#679289","#f4c095","#ee2e31","#ffb563","#918450","#f85e00","#a
41623","#9a031e","#d6d6d6","#ffee32","#ffd100","#333533","#202020"]
# sns.palplot(sns.color_palette(palette))
```

```
# plt.show()

plt.subplots(figsize=(20,8))
p = sns.barplot(x=dataset["Pclass"][:14],y=dataset["Age"],palette=palette,
saturation=1, edgecolor = "#1c1c1c", linewidth = 2)
p.axes.set_title("\nTop Anime Community\n", fontsize=25)
plt.ylabel("Total Member" , fontsize = 20)
plt.xlabel("\nAnime Name" , fontsize = 20)
# plt.yscale("log")
plt.xticks(rotation = 90)
for container in p.containers:
    p.bar_label(container,label_type = "center",padding = 6,size = 15,color =
"black",rotation = 90,
    bbox={"boxstyle": "round", "pad": 0.6, "facecolor": "orange", "edgecolor":
"black", "alpha": 1})

sns.despine(left=True, bottom=True)
plt.show()
```

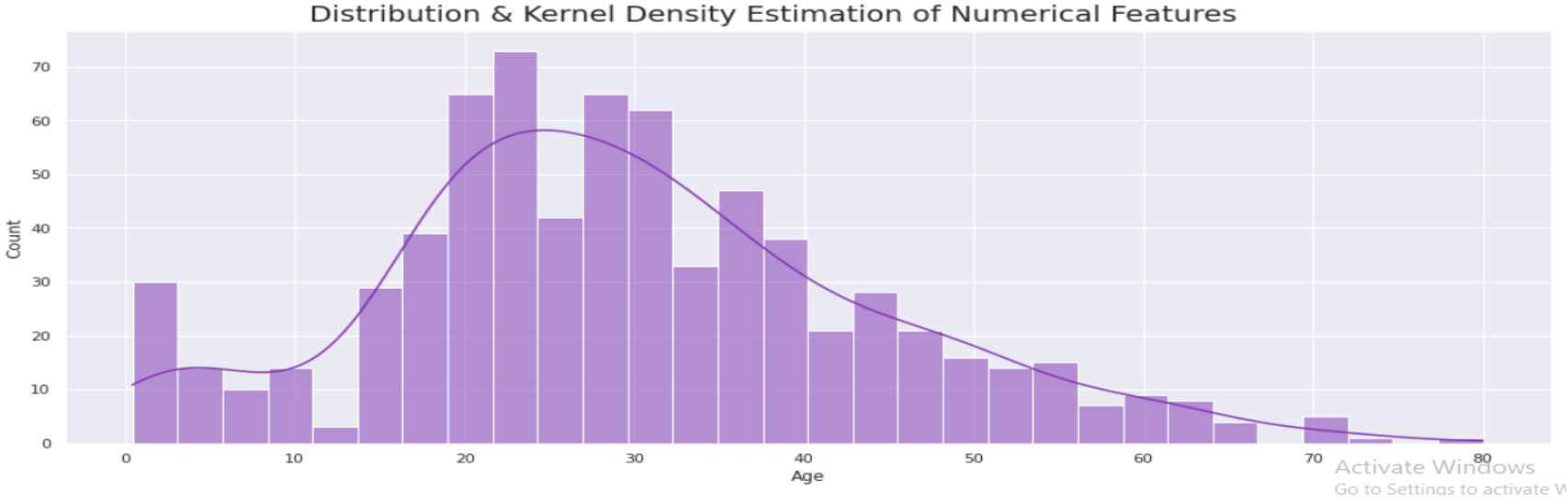



```
numfeature = ["Age", "Fare"]
enumfeat = list(enumerate(numfeature))

plt.figure(figsize=(20,9))
plt.suptitle("Distribution and Outliers of Numerical Data", fontsize=20)
for i in enumfeat:
```

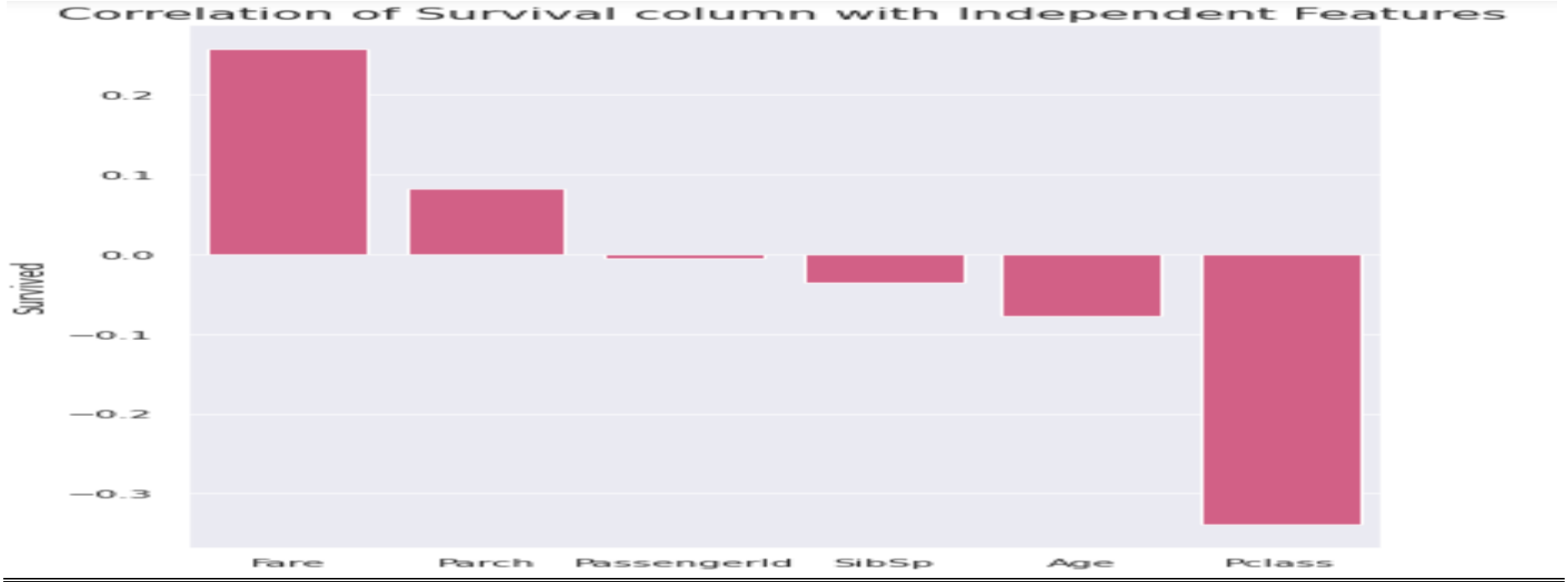
```
    plt.subplot(1,4,i[0]+1)
    sns.boxplot(data = train[i[1]], palette="rainbow")
    plt.xlabel(str(i[1]))
for i in enumfeat:
    plt.subplot(1,4,i[0]+3)
    sns.histplot(data = train[i[1]], palette="rainbow", bins=15)
    plt.xlabel(str(i[1]))
plt.tight_layout()
plt.show()
```



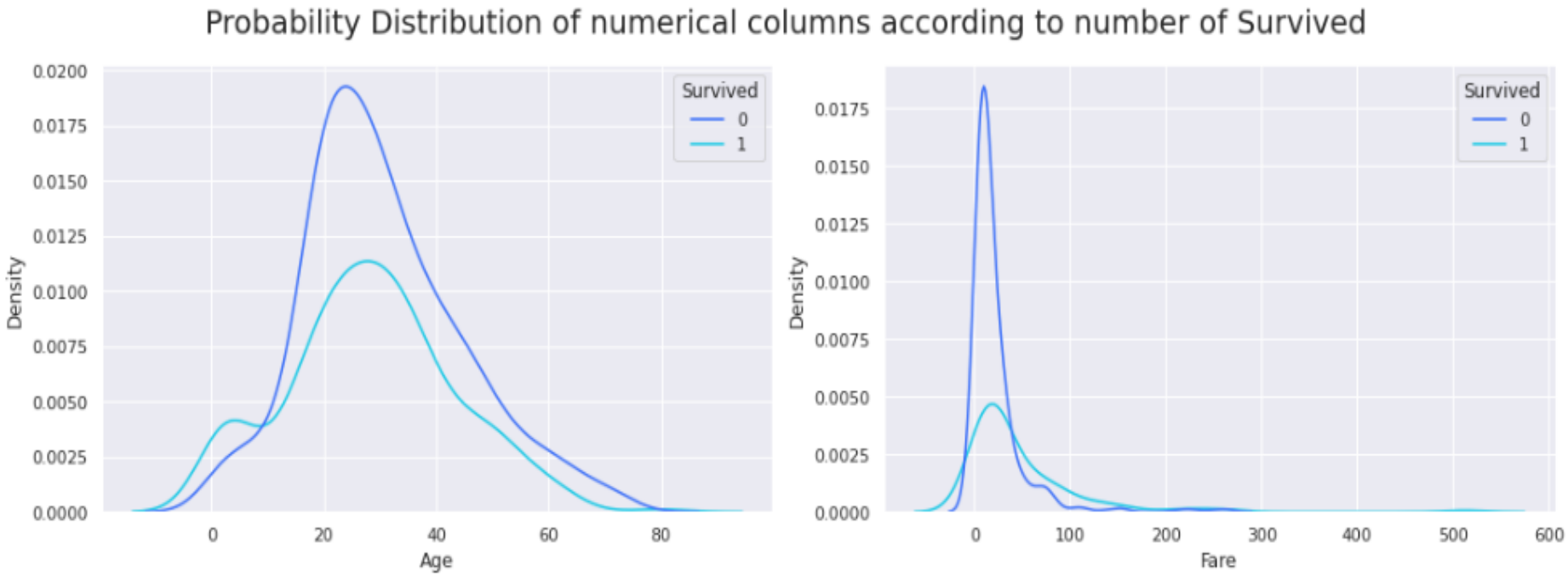
```
plt.figure(figsize=(15,12))
plt.suptitle("Distribution & Kernel Density Estimation of Numerical
Features", fontsize=20)
for i in enumfeat:
    plt.subplot(2,1,i[0]+1)
    sns.histplot(x = train[i[1]], kde=True, bins=30,
color=(0.50,0.20,0.70))
plt.tight_layout()
plt.show()
```



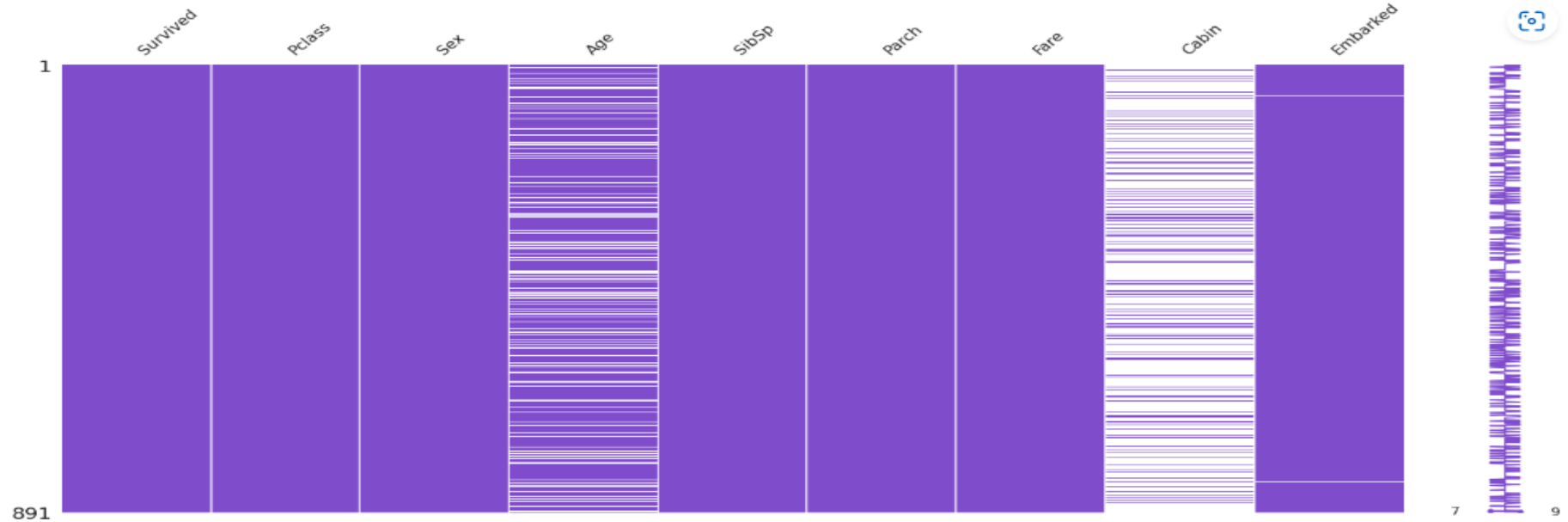
```
plt.figure(figsize=(6,8))
plt.title("Correlation of Survival column with Independent Features",
fontsize=15)
corr = train.corr()["Survived"].sort_values(ascending=False)[1:]
sns.barplot(x=corr.index, y=corr, color=(0.90,0.30,0.50))
plt.tight_layout()
plt.show()
```



```
plt.figure(figsize=(15,5))
plt.suptitle("Probability Distribution of numerical columns according to number of Survived", fontsize = 20)
for i in enumfeat:
    plt.subplot(1,2,i[0]+1)
    sns.kdeplot(data=train, x=i[1], hue="Survived")
plt.tight_layout()
plt.show()
```



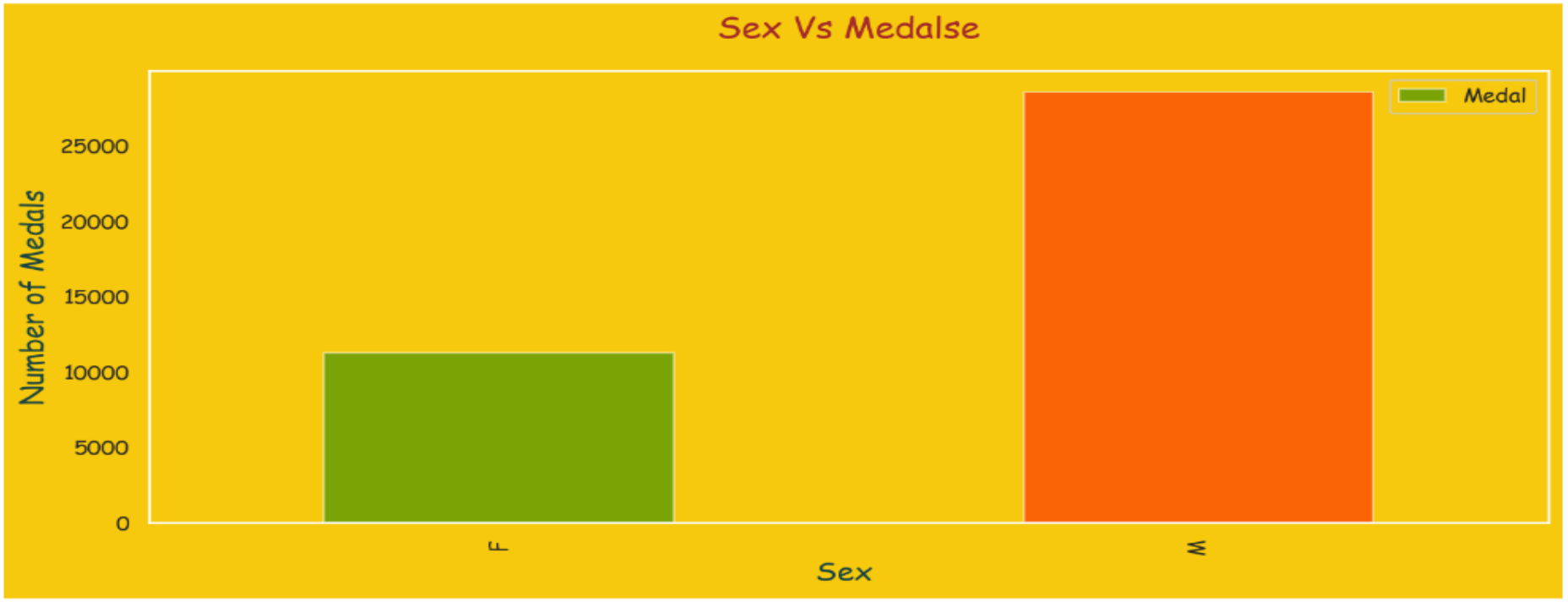
```
import missingno as msno
msno.matrix(train, color=(0.50,0.30,0.80))
plt.show()
x = train.isnull().sum()
for a, b in x.items():
    if b > 0:
        print(f"There are {b} missing values in column: {a}")
```




```
There are 177 missing values in column: Age
There are 687 missing values in column: Cabin
There are 2 missing values in column: Embarked
```

```
rc = {'figure.dpi': 150, 'axes.labelsize': 4,
                'axes.facecolor': '#F6C90E', 'grid.color': 'Red','figure.figsize':(12,5),
                'figure.facecolor': '#F6C90E'}
sns.set_theme(context='notebook',
              style='dark',
              palette='deep',
              font='Comic Sans Ms',
              font_scale=1,
              color_codes='red',
              rc=rc)

color = ['Green',"Red"]
df.groupby('Sex')['Medal'].count().sort_values(ascending=True).plot(kind="bar",
color=color,alpha=.5);
plt.title("Sex Vs Medalse",fontsize=17,color='Brown',font='Comic Sans
Ms',pad=20);
plt.xlabel("Sex ",fontsize=15,color='#1a4441',font='Comic Sans Ms')
plt.ylabel("Number of Medals",fontsize=15,color='#1a4441',font='Comic Sans
Ms');
```

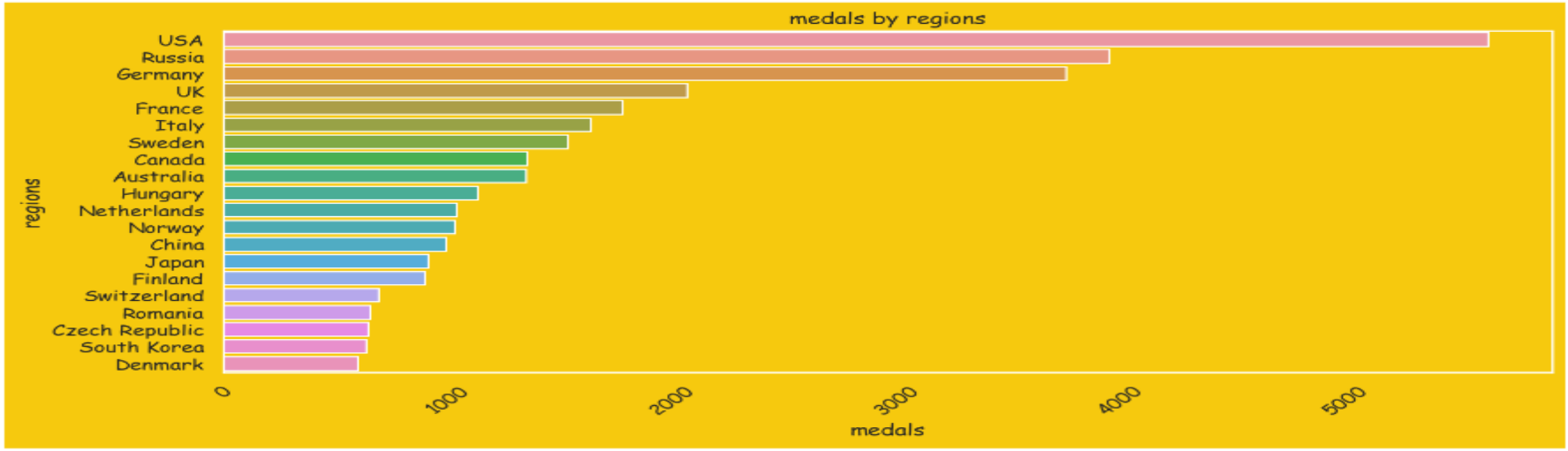
```
plt.legend(loc='best');
plt.savefig('world regions.png');
```



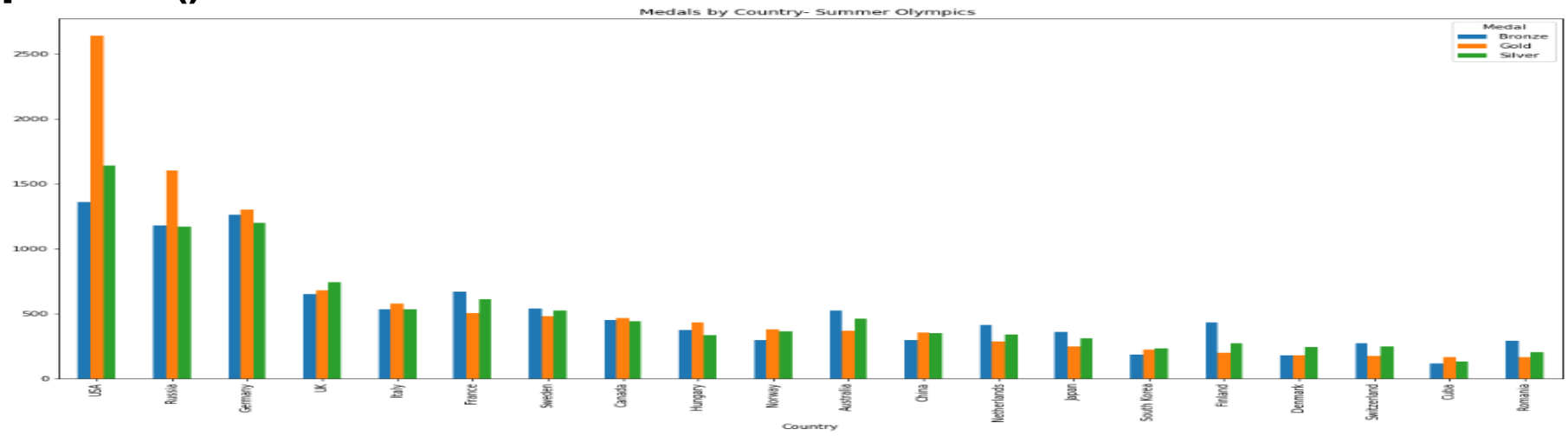

```
region_medal=df.groupby('region')['Medal'].count().nlargest(20).reset_index()
region_medal.head()
```

	region	Medal
0	USA	5637
1	Russia	3947
2	Germany	3756
3	UK	2068
4	France	1777

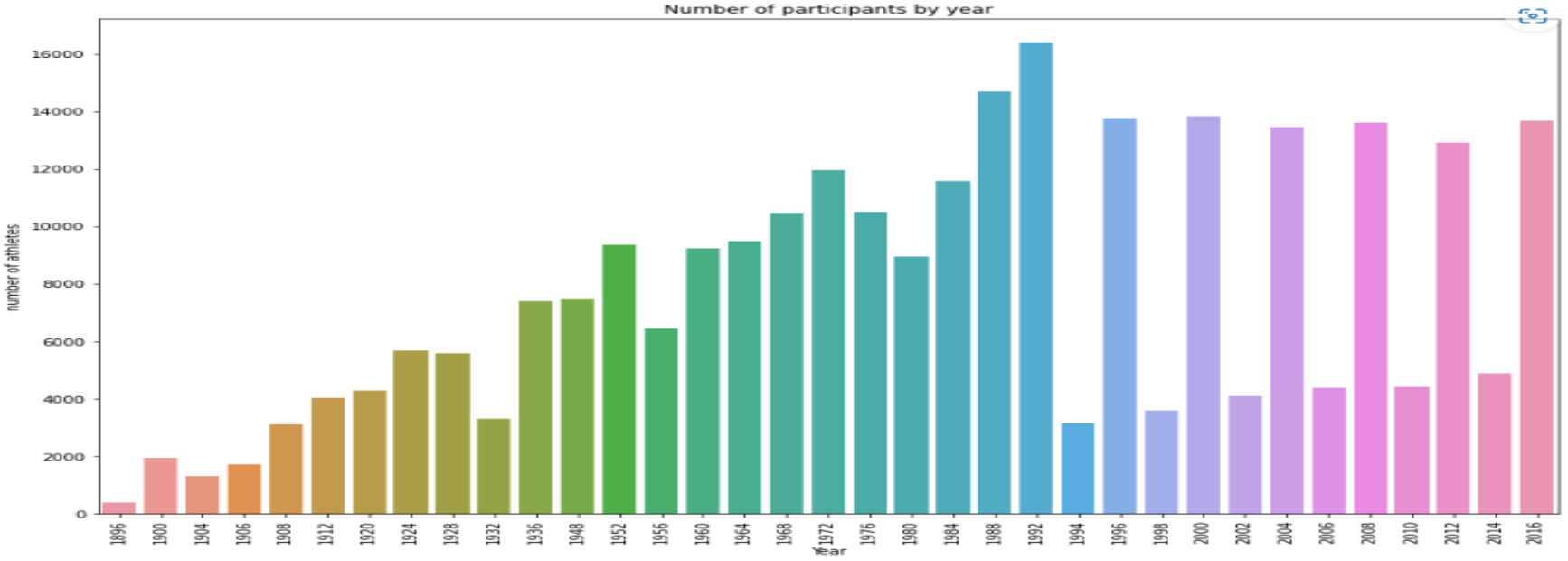
```
sns.barplot(y='region',x='Medal',data=region_medal)
plt.title('medals by regions')
plt.xlabel('medals')
plt.ylabel('regions')
plt.xticks(rotation=45)
plt.show()
```



```
summer_medals=df.groupby(['region', 'Medal']).size().reset_index()
summer_medals.columns=['region', 'Medal', 'count']
summer_medals.pivot('region', 'Medal', 'count').fillna(0)
summer_medals_20=summer_medals.pivot('region', 'Medal',
'count').fillna(0).sort_values(['Gold'], ascending=False).head(20)
summer_medals_20.plot(kind='bar')
plt.xlabel('Country')
plt.title('Medals by Country- Summer Olympics ')
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
plt.show()
```



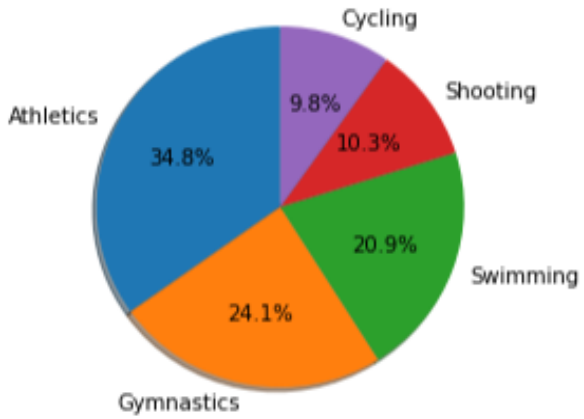
```
year=df['Year'].value_counts()

plt.figure(figsize=(15,10))
sns.barplot(x=year.index, y=year.values)
plt.xticks(rotation=90)
plt.xlabel("Year")
plt.ylabel("number of athletes")
plt.title("Number of participants by year")
plt.show()
```



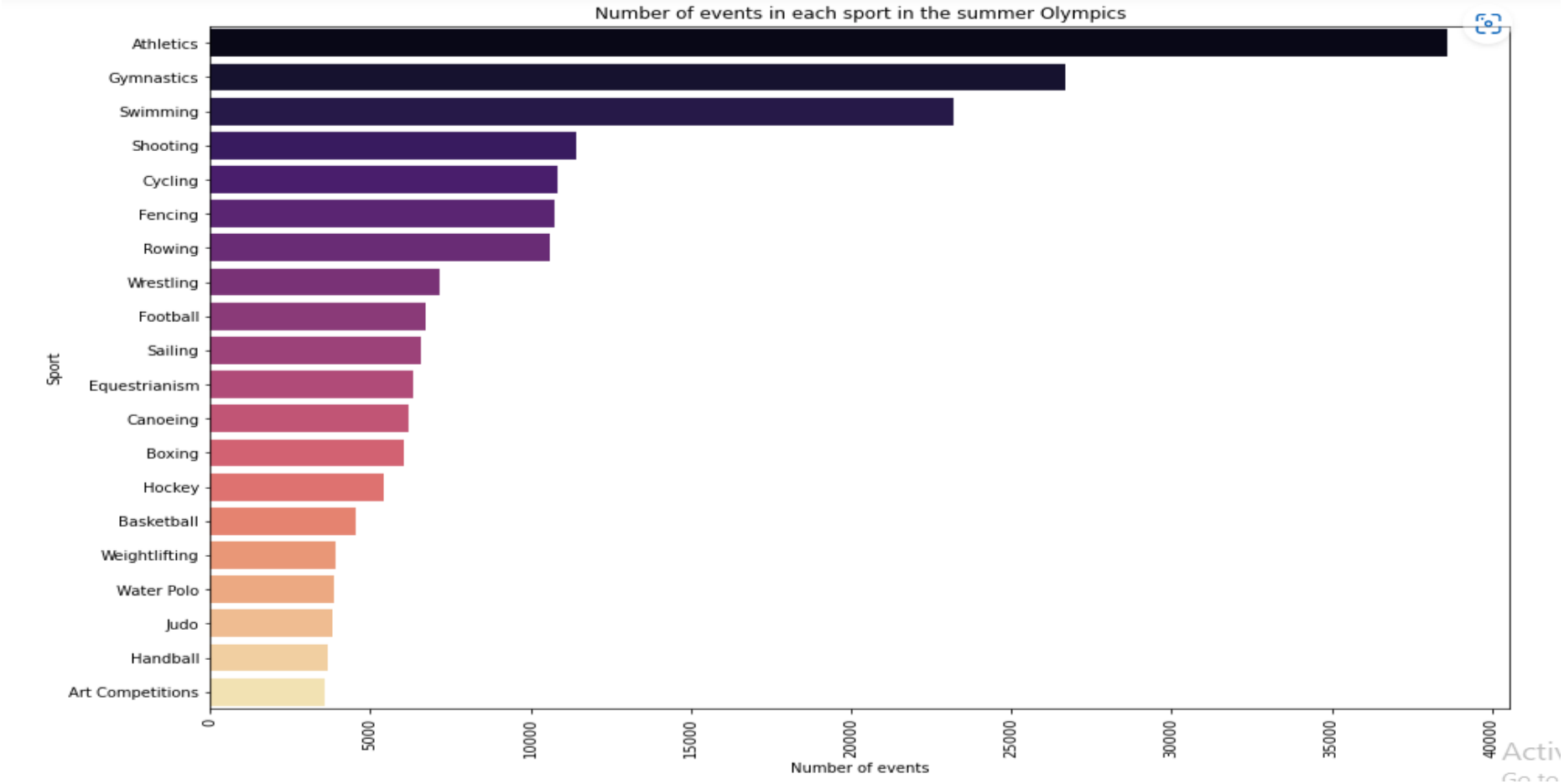
```
sport=df['Sport'].value_counts()[:5]
print(sport)
```

```
Athletics     38624
Gymnastics    26707
Swimming      23195
Shooting      11448
Cycling       10859
Name: Sport, dtype: int64
```

```
labels=sport.index
sizes=sport.values
plt.pie(sizes,labels=labels,autopct='%1.1f%%',
        shadow=True,startangle=90)
plt.show()
```



```
sport_summer=df[df['Season']=='Summer']['Sport'].value_counts
().sort_values(ascending=False).head(20)
sport_summer

plt.figure(figsize=(15,10))
sns.barplot(y=sport_summer.index, x=sport_summer.values,
palette='magma')
plt.xlabel('Number of events')
plt.ylabel('Sport')
plt.xticks(rotation=90)
plt.title("Number of events in each sport in the summer
Olympics")
plt.show()
```




```
sport_winter=df[df['Season']=='Winter']['Sport'].value_counts().sort_values(ascending=False)
```

```
Cross Country Skiing    9133
Alpine Skiing           8829
Speed Skating           5613
Ice Hockey              5456
Biathlon                4893
```

```
plt.figure(figsize=(15,10))
sns.barplot(y=sport_winter.head(20).index, x=sport_winter.head(20).values,
palette='magma')
plt.xlabel('Number of events')
plt.ylabel('Sport')
plt.xticks(rotation=90)
plt.title("Number of events in each sport in the winter Olympics")
plt.show()
```