```
import os
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression
```

```
# Set the working directory to the script's directory
script_dir = os.path.dirname(os.path.abspath("winequality-white.csv"))
os.chdir(script_dir)
```

```
#read the csv file as pandas dataframe
df = pd.read_csv('winequality-white.csv',sep=';')
```

```
#display the first 5 rows
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

Next steps: Generate code with df View recommended plots New interactive sheet

```
#the summary of the data
df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000

```
#check for null values
df.isna().sum()
```

	0
fixed acidity	0
volatile acidity	0
citric acid	0
residual sugar	0
chlorides	0
free sulfur dioxide	0
total sulfur dioxide	0
density	0
pH	0
sulphates	0
alcohol	0
quality	0

dtype: int64

```
#check the column names
df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
#check for data types of the attributes
df.dtypes
```

	0
fixed acidity	float64
volatile acidity	float64
citric acid	float64
residual sugar	float64
chlorides	float64
free sulfur dioxide	float64
total sulfur dioxide	float64
density	float64
pH	float64
sulphates	float64
alcohol	float64
quality	int64

dtype: object

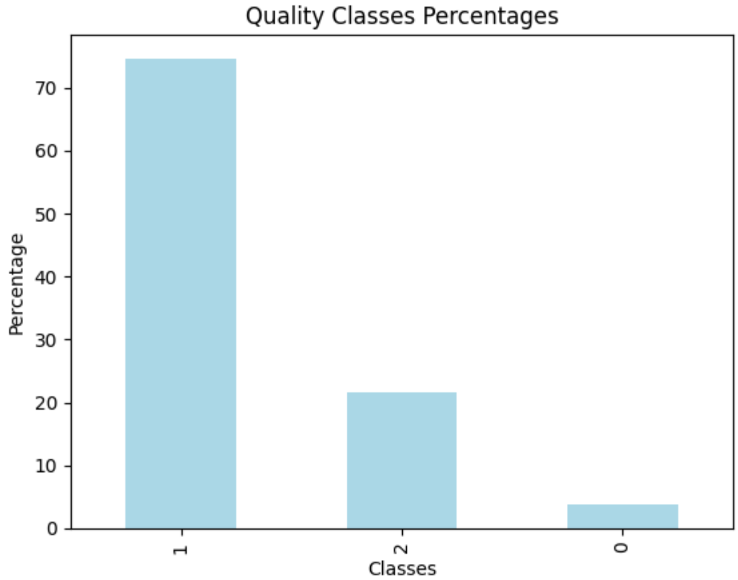
```
#we need to change the target attribute to 3 classes
#Function for classes
def classify_value(value):
    if value <=4:
        return 0
    elif 4 < value <= 6:
        return 1
    else:
        return 2

# Apply function to create classes in target variable
df['quality'] = df['quality'].apply(classify_value)

df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	1
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	1
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	1

Next steps: Generate code with df View recommended plots New interactive sheet

```
#display the percentage of the classes
percentages_target=df['quality'].value_counts(normalize=True)*100
percentages_target.plot(kind='bar', color='lightblue')
plt.title('Quality Classes Percentages')
plt.xlabel('Classes')
plt.ylabel('Percentage')
```

```
Text(0, 0.5, 'Percentage')
```




```
#separate target variable and features
y=df['quality'].values.reshape(-1,1)
x = df.drop(columns=['quality']).values
```

```
test_scores={'SVC':[],'Decision Tree':[],'Logistic Regression':[]}
#split the dataset ten times and train the models
for i in range(10):

#split training, validation and test set
  x_tr, x_ts, y_tr, y_ts = train_test_split(x,y, test_size=0.2) # get training and testing sets
  x_tr, x_vl, y_tr, y_vl = train_test_split(x_tr,y_tr, test_size=0.1) # get training and validation sets

#standarize the features
  scaler = StandardScaler()
  x_tr = scaler.fit_transform(x_tr)
  x_vl = scaler.transform(x_vl)
  x_ts = scaler.transform(x_ts)

#different hyperparameters for tuning the SVC model
  grid_svc = {
    'C': [0.01, 0.1, 1, 10],                    # Regularization parameter
    'kernel': ['linear', 'rbf'],                # Kernel type
    }

  scores=[] #initialize f1-scores list
#iterate for each hyperparameter
  for j in grid_svc['C']:
    for k in grid_svc['kernel']:

      model1 = SVC(C=j,kernel=k, decision_function_shape='ovo')
      model1.fit(x_tr, y_tr.ravel())
      y_pred1 = model1.predict(x_vl)
      f1 = f1_score(y_vl,y_pred1,average='micro')
      scores.append(f1)
```

```
      #find the best hyperparameters
      if f1==max(scores):
          best_param_svc=[j,k]
  #train the model with best hyperparameters
  model1 = SVC(C=best_param_svc[0],kernel=best_param_svc[1], decision_function_shape='ovo')
  model1.fit(x_tr, y_tr.ravel())
  #test the model
  y_pred1 = model1.predict(x_ts)
  f1 = f1_score(y_ts,y_pred1,average='micro')
  test_scores['SVC'].append(f1)

#different hyperparameters for tuning the SVC model
  grid_dtrees = {
    'max_depth': [None, 5, 10, 50],
    'min_samples_split': [2,10,30],
    'min_samples_leaf': [1,5,20]
    }

  scores=[] #initialize f1-scores list for decision trees
#iterate for each hyperparameter
  for j in grid_dtrees['max_depth']:
    for k in grid_dtrees['min_samples_split']:
      for l in grid_dtrees['min_samples_leaf']:
        model2 = DecisionTreeClassifier(max_depth=j,min_samples_split=k,min_samples_leaf=l)
        model2.fit(x_tr, y_tr.ravel())
        y_pred2 = model2.predict(x_vl)
        f1 = f1_score(y_vl,y_pred2,average='micro')
        scores.append(f1)
        #find the best hyperparameters
        if f1==max(scores):
          best_param_dtrees=[j,k,l]
  #train the model with best hyperparameters
  model2 = DecisionTreeClassifier(max_depth=best_param_dtrees[0],min_samples_split=best_param_dtrees[1], min_samples_leaf=best_param_dtr
  model2.fit(x_tr, y_tr.ravel())
  #test the model
  y_pred2 = model2.predict(x_ts)
  f1 = f1_score(y_ts,y_pred2,average='micro')
  test_scores['Decision Tree'].append(f1)

#different hyperparameters for tuning the logistic regression
  grid_lr = {
    'C': [0.01, 0.1, 1, 10],                    # Regularization parameter
    }

  scores=[] #initialize f1-scores list for logistic regression
#iterate for each hyperparameter
  for j in grid_lr['C']:

    model3 = LogisticRegression(C=j)
    model3.fit(x_tr, y_tr.ravel())
    y_pred3 = model3.predict(x_vl)
    f1 = f1_score(y_vl,y_pred3,average='micro')
    scores.append(f1)
    #find the best hyperparameters
    if f1==max(scores):
      best_param_dtrees=j
  #train the model with best hyperparameters
  model3 = LogisticRegression(C=best_param_dtrees)
  model3.fit(x_tr, y_tr.ravel())
  #test the model
  y_pred3 = model3.predict(x_ts)
  f1 = f1_score(y_ts,y_pred3,average='micro')
  test_scores['Logistic Regression'].append(f1)
```

```
import statistics
#calculate means and standard deviations for each model F1 score
mean_test_scores = {key: round(sum(values) / len(values),4) for key, values in test_scores.items()}
std_test_scores = {key: round(statistics.stdev(values),4) for key, values in test_scores.items()}
for i in mean_test_scores.keys():
  print(i," Mean :",mean_test_scores[i])
  print(i," Standard Deviation :",std_test_scores[i])
```

```
SVC  Mean : 0.7896
SVC  Standard Deviation : 0.015
Decision Tree  Mean : 0.7626
Decision Tree  Standard Deviation : 0.0085
Logistic Regression  Mean : 0.7587
```

```
Logistic Regression  Standard Deviation : 0.013
```

```
labels = [i for i in mean_test_scores.keys()]
```

```
# Define labels and metrics
labels = list(mean_test_scores.keys())  # Models Name
mean = {'F1 score': list(mean_test_scores.values())}
stdeviations = {'F1 score': list(std_test_scores.values())}
strmetrics = ['F1 score']

# Define the width of each bar and adjust spacing
bar_width = 0.6  # Slightly wider for closer bars
x_pos = np.arange(len(labels))

# Create a figure and axis object
fig, ax = plt.subplots(figsize=(12, 6))

# Adjust the subplot layout parameters
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9)

# Set colors and patterns
colors = ['#1f77b4', '#ff7f0e', '#2ca02c']  # Vibrant color palette

# Create a bar plot for the metric
for i, (label, value, std) in enumerate(zip(labels, mean['F1 score'], stdeviations['F1 score'])):
    # Position for each bar
    pos = x_pos[i]
    # Create the bar
    bar = ax.bar(pos, value, color=colors[i], edgecolor='black', yerr=std,
                 hatch=None, width=bar_width)
    # Add bar labels
    ax.bar_label(bar, padding=3, fontsize=12)

# Set the x-axis labels and tick positions
ax.set_xticks(x_pos)
ax.set_xticklabels(labels, fontsize=14)

# Add axis labels
ax.set_xlabel('Models', fontsize=16)
ax.set_ylabel(strmetrics[0], fontsize=16)

# Set the y-axis view limit
ax.set_ylim(0, 1.1)
ax.yaxis.set_tick_params(labelsize=12)

# Show the plot
plt.show()
```