In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.cm as cm

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.ensemble import RandomForestClassifier

#load the dataset
df=pd.read_csv(r"C:\Users\sandi\OneDrive\Desktop\final project\archive (5).zip")
df.head()
```

Out[3]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etes
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	5
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	8
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	7
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	6
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	9

In [4]:
```
#set the sl_no as a row no
df.set_index('sl_no',inplace=True)
df.head()
```

Out[4]:

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p
sl_no										
1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0
2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5
3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0
4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0
5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8

In [6]:
```
#check datatype or null
#prints information about dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 215 entries, 1 to 215
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   gender          215 non-null    object 
 1   ssc_p           215 non-null    float64
 2   ssc_b           215 non-null    object 
 3   hsc_p           215 non-null    float64
 4   hsc_b           215 non-null    object 
 5   hsc_s           215 non-null    object 
 6   degree_p        215 non-null    float64
 7   degree_t        215 non-null    object 
 8   workex          215 non-null    object 
 9   etest_p         215 non-null    float64
 10  specialisation  215 non-null    object 
 11  mba_p           215 non-null    float64
 12  status          215 non-null    object 
 13  salary          148 non-null    float64
dtypes: float64(6), object(8)
memory usage: 25.2+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
count	215.000000	215.000000	215.000000	215.000000	215.000000	148.000000
mean	67.303395	66.333163	66.370186	72.100558	62.278186	288655.405405
std	10.827205	10.897509	7.358743	13.275956	5.833385	93457.452420
min	40.890000	37.000000	50.000000	50.000000	51.210000	200000.000000
25%	60.600000	60.900000	61.000000	60.000000	57.945000	240000.000000
50%	67.000000	65.000000	66.000000	71.000000	62.000000	265000.000000
75%	75.700000	73.000000	72.000000	83.500000	66.255000	300000.000000
max	89.400000	97.700000	91.000000	98.000000	77.890000	940000.000000

In [6]:
```
print(df.shape[0]) # nummber of rows
print(df.shape[1]) # total number of columns
```

```
215
14
```

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

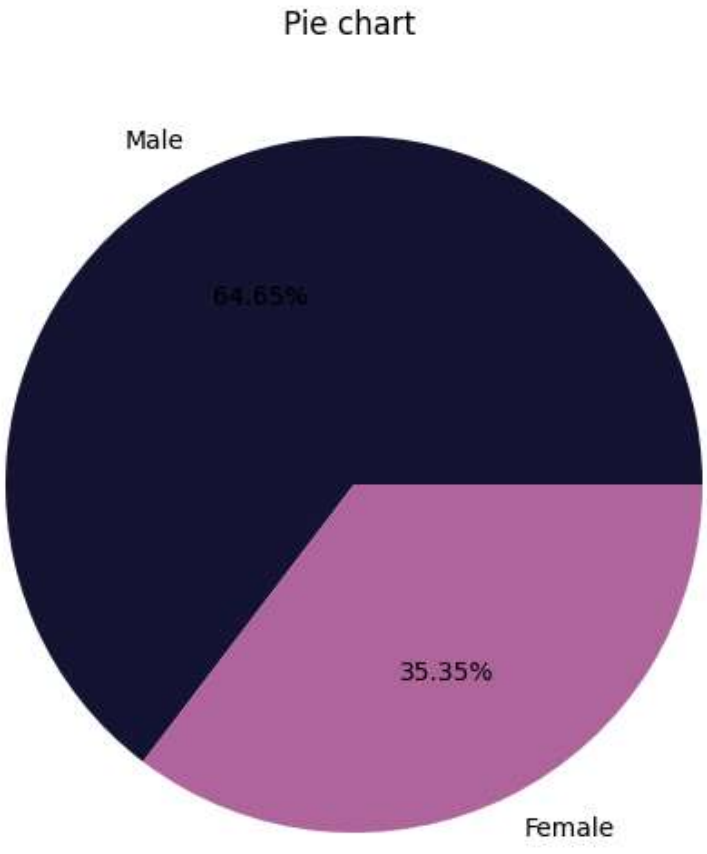
Out[7]:
```
gender            0
ssc_p             0
ssc_b             0
hsc_p             0
hsc_b             0
hsc_s             0
degree_p          0
degree_t          0
workex            0
etest_p           0
specialisation    0
mba_p             0
status            0
salary           67
dtype: int64
```

In [8]:
```
# Replace the value with 0 and 1

df["gender"].replace(["F","M"],[0,1],inplace=True)
df["status"].replace(['Placed',"Not Placed"],[1,0],inplace=True)

df['workex'].replace(to_replace ="Yes", value =1,inplace=True)
df['workex'].replace(to_replace ="No", value =0,inplace=True)
df.head()
```

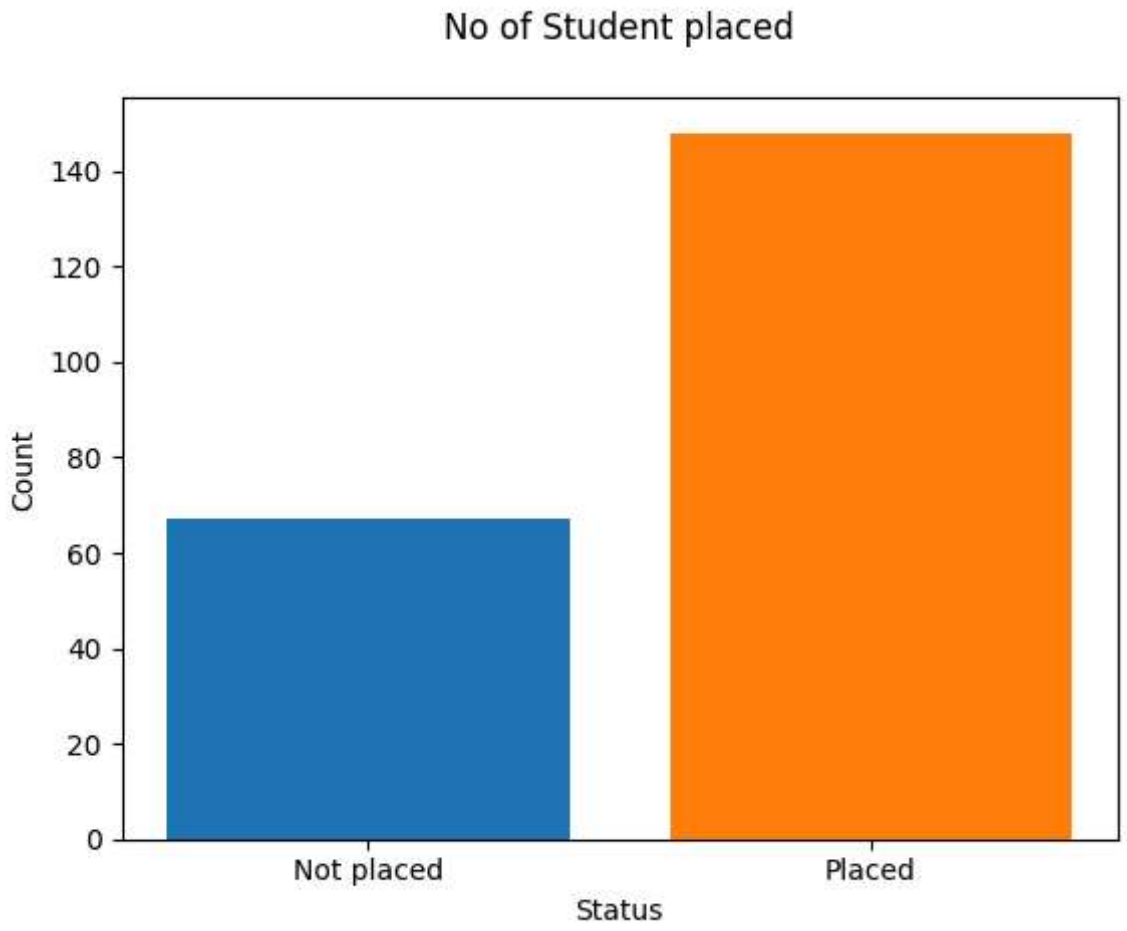
Out[8]:

sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p
1	1	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	0	55.0
2	1	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	1	86.5
3	1	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	0	75.0
4	1	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	0	66.0
5	1	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	0	96.8

In [9]:
```
# percentage of male or female
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
gender = ['Male','Female']
students = [139,76]
ax.pie(students, labels = gender, autopct='%1.2f%%', colors = ["#151436", "#b0689e"
plt.title('Pie chart ')
plt.show()
```



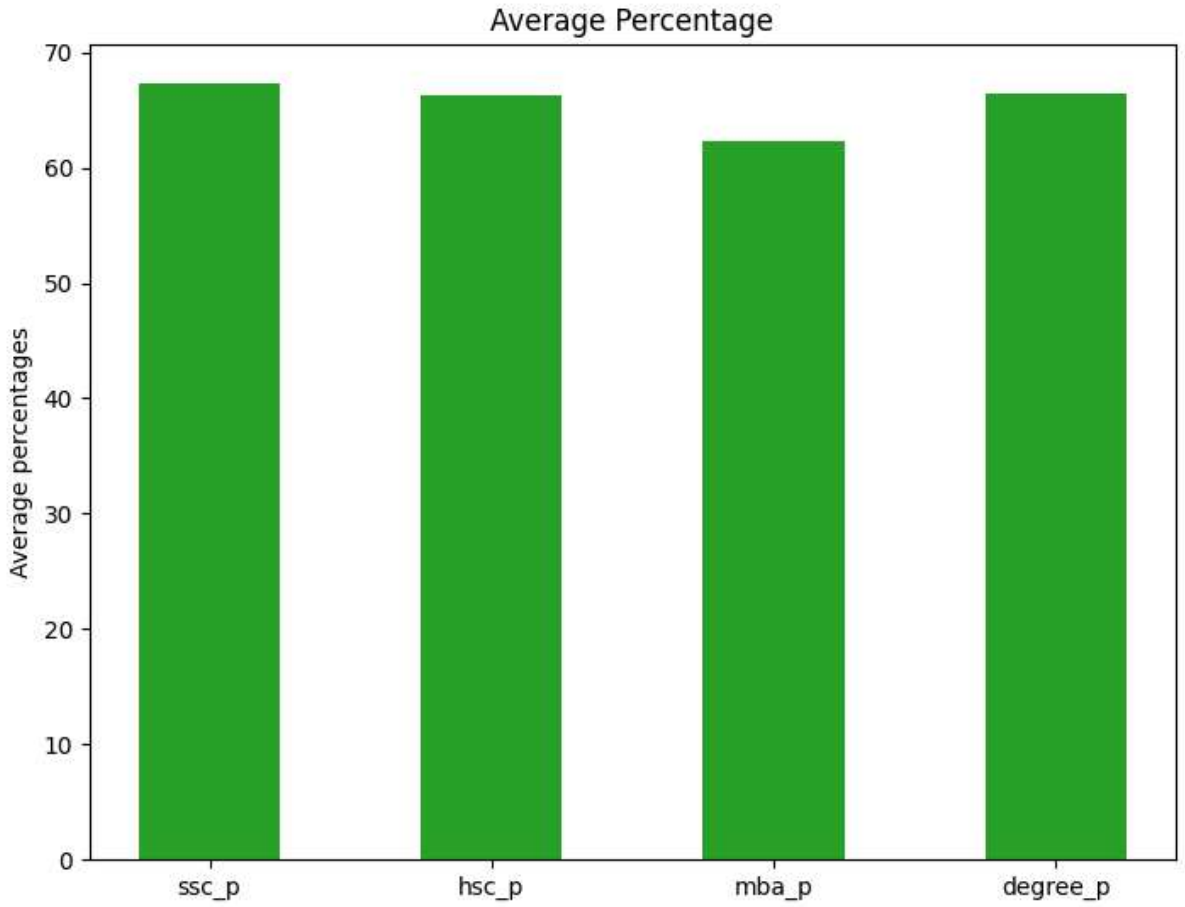
In [10]:
```
#show the no. of student how placed or not placed.
plt.bar([0],height=len(df[df["status"]==0]))
plt.bar([1],height=len(df[df["status"]==1]))

plt.xlabel("Status")
plt.ylabel("Count")

plt.xticks(np.arange(2), ('Not placed', 'Placed'))
plt.title("No of Student placed\n")
plt.show()
```



In [11]:
```
#create a new dataframe of only numerical values
numerical_df=df.select_dtypes(["float64","int64"])
```

In [12]:
```
#average percentage of all education qualification
values = [(numerical_df['ssc_p'].mean()),(numerical_df['hsc_p'].mean()),(numerical_
print('scc_p mean = ' +str(numerical_df['ssc_p'].mean()))
print('hsc_p mean = ' +str(numerical_df['hsc_p'].mean()))
print('mba_p mean = ' +str(numerical_df['mba_p'].mean()))
print('degree_p mean = ' +str(numerical_df['degree_p'].mean()))
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
names = ['ssc_p','hsc_p','mba_p','degree_p']
ax.set_ylabel('Average percentages')
ax.set_title('Average Percentage')
ax.bar(names,values,width = 0.5,color=["#2ca02c"])
plt.show()
```

```
scc_p mean = 67.30339534883721
hsc_p mean = 66.33316279069768
mba_p mean = 62.278186046511635
degree_p mean = 66.37018604651163
```



In [13]:
```
x_train , x_test , y_train , y_test = train_test_split(df[['ssc_p','hsc_p','degree_
```

In [14]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)             # determine parameters and transfo
x_test = sc.transform(x_test)                   # transform data for further analy
```

In [15]:
```
print("Scaled x_train:")
print(x_train)

print("Scaled x_test:")
print(x_test)
```

```
Scaled x_train:
[[ 0.10828215 -0.3089132  -0.20707511  1.40196906 -1.29484885 -0.65336684]
 [-0.45878969 -0.01233929 -0.12287104  1.40196906 -0.91192477  0.06365275]
 [ 1.43144976  0.71055964 -0.06673499 -0.7132825  -0.22878823 -0.61510361]
 ...
 [ 0.9399875  -0.21623385 -0.31934721 -0.7132825   0.27513985  1.66072655]
 [ 1.33693778  2.19342922  2.31904706 -0.7132825   0.61977152  1.8903059 ]
 [ 0.69425637  0.63641616  0.2700813  -0.7132825  -0.5290007   0.13685197]]
Scaled x_test:
[[-0.36427771  1.26663573 -0.20707511  1.40196906 -0.22266144 -0.74985671]
 [-0.36427771  0.06180419  0.43848944  1.40196906 -1.24353702  1.55259134]
 [-0.17525377  0.89591833  0.77530573  1.40196906  0.92611077 -0.5335863 ]
 [ 1.52596173  1.17395638  0.21394525  1.40196906  0.92611077  0.75405445]
 [ 0.86437792 -0.4942719   0.21394525  1.40196906 -1.10338681 -0.14097233]
 [-0.36427771  0.35837811 -0.76843559 -0.7132825  -1.67777292  0.05699828]
 [-2.25451716 -0.77230995 -1.61047632  1.40196906 -0.58643931 -0.60179466]
 [ 0.10828215 -0.40159255  0.35428537  1.40196906 -0.37583107  0.032044  ]
 [ 0.58084201 -0.40159255  0.21394525 -0.7132825   0.16026263 -0.69329368]
 [-0.55330166 -1.42106539 -2.03149668 -0.7132825  -0.26554894  0.10191598]
 [-0.0807418  -0.4942719   0.77530573 -0.7132825   0.007093   -0.19088088]
 [ 1.20462102  1.45199443  1.56121041 -0.7132825   1.46220448  2.0516769 ]
 [ 0.2500501  -0.35525288 -0.76843559 -0.7132825   1.68506629  1.14334118]
 [-0.26976574  0.15448354 -0.34741523 -0.7132825   0.23684744 -0.72490243]
 [-0.55330166 -0.77230995 -1.89115656 -0.7132825   1.30903485 -0.32230675]
 [-1.49842138 -1.32838604 -1.61047632 -0.7132825  -0.37583107 -0.47203241]
 [ 1.05340187  0.89591833  2.5997273  -0.7132825   1.76854373  1.14833204]
 [-2.34902913 -0.58695125 -0.20707511 -0.7132825   1.58933527  0.12687026]
 [-0.0807418   0.33984224 -0.20707511 -0.7132825   1.23245003  1.63078141]
 [-1.29049504 -0.3089132  -1.18945596 -0.7132825   0.5431867  -0.61843085]
 [ 0.48633004  1.17395638  0.77530573 -0.7132825   1.48824331 -0.14429956]
 [-2.54844939 -1.90021762 -1.89115656 -0.7132825  -0.05417485  0.55442022]
 [ 0.86437792  0.61788029  2.03836682  1.40196906  1.30903485  1.25480362]
 [-1.49842138 -0.40159255 -1.75081644 -0.7132825   0.007093   -1.12250726]
 [ 0.03267257  0.59934442  0.01746908  1.40196906  1.84512855  1.44611976]
 [ 0.39181806 -0.58695125  0.35428537 -0.7132825  -1.25655644 -0.62508532]
 [ 0.9588899   0.98859768  1.8980267  -0.7132825  -0.91192477  0.7590453 ]
 [ 0.58084201  0.33984224  0.77530573  1.40196906 -0.91192477 -0.81806507]
 [-0.64781363 -0.40159255 -0.20707511 -0.7132825  -0.75875514 -0.88960067]
 [-2.16000518 -0.8649893  -1.18945596  1.40196906 -1.18839595  0.46458482]
 [ 0.48633004 -0.21623385  1.47700634  1.40196906 -0.5290007  -0.19587174]
 [ 0.48633004  2.84218466  1.75768658  1.40196906  1.30903485  1.43946528]
 [-1.97098124 -1.05034799 -0.20707511 -0.7132825  -0.75875514  0.47955739]
 [-1.30939744 -1.79178278 -1.32979608 -0.7132825   1.36187837 -0.84135573]
 [-0.26976574  0.80323899  0.35428537  1.40196906  0.007093    0.41633988]
 [ 1.80949764  1.91539117  0.21394525 -0.7132825   1.76854373  0.12354302]
 [-1.59293335 -1.14302734 -0.76843559 -0.7132825  -0.91192477 -0.25243477]
 [ 0.66590278 -0.8649893  -0.62809547  1.40196906  0.61977152 -0.22914411]
 [ 0.76986595 -0.21623385  0.77530573 -0.7132825  -1.0650944   0.72743655]
 [-0.93134955 -0.4942719  -0.76843559 -0.7132825  -1.0650944  -1.36705919]
 [-0.0807418   2.28610857 -1.18945596 -0.7132825  -1.29484885 -0.55854058]
 [-0.17525377 -0.40159255  0.91564585 -0.7132825  -1.0650944   0.36643133]
 [-0.93134955 -0.4942719  -0.90877572  1.40196906 -0.75875514 -0.21749878]]
```

In [16]:

```python
#RANDOM FOREST CLASSIFICATION
accuracy_scores = np.zeros(4)
def rfc_model(x_train,y_train,x_test,y_test):

    '''
    Input
      x_train :training featue
      x_test  :test featue
      y_train :training output data
      y_test  :testing  output data
```

```
    Output :
     accuracy_score       :
     classification_report:
    '''
    print("using Random Forest Classification method")
    model=RandomForestClassifier(n_estimators =14, criterion = 'entropy', random_st
    model.fit(x_train,y_train)
    prediction=model.predict(x_test)
    print('Accuracy {0:.2f}%'.format((accuracy_score(y_test,prediction)*100)))
    print(classification_report(y_test,prediction))
    accuracy_scores[0] = accuracy_score(y_test, prediction)*100
rfc_model(x_train,y_train,x_test,y_test)
print('Randon Forest Classifier accuracy: {}%'.format(accuracy_scores[0]))
```

```
using Random Forest Classification method
Accuracy 83.72%
              precision    recall  f1-score   support

           0       0.85      0.69      0.76        16
           1       0.83      0.93      0.88        27

    accuracy                           0.84        43
   macro avg       0.84      0.81      0.82        43
weighted avg       0.84      0.84      0.83        43

Randon Forest Classifier accuracy: 83.72093023255815%
```

In [17]:

```
#K NEAREST NEIGHBOUR
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
result_1 = pd.DataFrame(columns=['Model','Accuracy Score'])
#K Nearest Neighbours Classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
weights = ['uniform','distance']
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
metric = ['minkowski','euclidean','manhattan']
algorithm=['auto']
p=[1,2]
knn_hyperparameters = dict(n_neighbors=n_neighbors, weights=weights, algorithm=algo
KNN_gridSearch = RandomizedSearchCV(knn,param_distributions=knn_hyperparameters,cv=
KNN_gridSearch.fit(x_train,y_train)
knn_y_pred = KNN_gridSearch.predict(x_test)
print('KNN Classifier')
print('\n Confusion Matrix')
print(confusion_matrix(y_test,knn_y_pred))
print('\n Classification Report')
print(classification_report(y_test,knn_y_pred))
knn_AS = accuracy_score(y_test,knn_y_pred)

accuracy_scores[1] = accuracy_score(y_test,knn_y_pred)*100
print('K Nearest Neighbors Classifier accuracy: {}%'.format(accuracy_scores[1]))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
KNN Classifier

 Confusion Matrix
[[11  5]
 [ 1 26]]

 Classification Report
              precision    recall  f1-score   support

           0       0.92      0.69      0.79        16
           1       0.84      0.96      0.90        27

    accuracy                           0.86        43
   macro avg       0.88      0.83      0.84        43
weighted avg       0.87      0.86      0.86        43

K Nearest Neighbors Classifier accuracy: 86.04651162790698%
```

In [20]:

```
#SVM
from sklearn.svm import SVC
svc = SVC()
C = [0.1, 1, 10, 100, 1000]
kernel = ['rbf','poly','sigmoid','linear']
degree = list(range(1,5,1))
gamma = [1, 0.1, 0.01, 0.001, 0.0001]
svc_hyperparameters = dict(C=C, kernel=kernel, degree=degree, gamma=gamma)
svc_gridSearch = RandomizedSearchCV(svc,param_distributions=svc_hyperparameters,cv=
svc_gridSearch.fit(x_train,y_train)  # grid search simplest algo for hyperparameter
svc_y_pred = svc_gridSearch.predict(x_test)
print('Support Vector Machine (SVC)')
print('\n Confusion Matrix')
print(confusion_matrix(y_test,svc_y_pred))
print('\n Classification Report')
print(classification_report(y_test,svc_y_pred))
svc_AS = accuracy_score(y_test,svc_y_pred)
accuracy_scores[2] = accuracy_score(y_test,svc_y_pred)*100
print('SVM Classifier accuracy: {}%'.format(accuracy_scores[2]))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Support Vector Machine (SVC)

 Confusion Matrix
[[10  6]
 [ 2 25]]

 Classification Report
              precision    recall  f1-score   support

           0       0.83      0.62      0.71        16
           1       0.81      0.93      0.86        27

    accuracy                           0.81        43
   macro avg       0.82      0.78      0.79        43
weighted avg       0.82      0.81      0.81        43

SVM Classifier accuracy: 81.3953488372093%
```

In [18]:

```
#DECISION TREE
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
criterion=['gini','entropy']
splitter=['best', 'random']
max_depth = list(range(10,100,10))
```

```
min_samples_split = [2, 5, 10]
max_features = ['auto', 'sqrt','log2',None]
max_depth = list(range(10,100,10))
min_samples_leaf = [1, 2, 4]
dtc_hyperparameters = dict(criterion=criterion, splitter=splitter, max_depth=max_de
dtc_gridSearch = RandomizedSearchCV(dtc,param_distributions=dtc_hyperparameters,cv=
dtc_gridSearch.fit(x_train,y_train)
dtc_y_pred = dtc_gridSearch.predict(x_test)
print('Decision Tree Classifier')
print('\n Confusion Matrix')
print(confusion_matrix(y_test,dtc_y_pred))
print('\n Classification Report')
print(classification_report(y_test,dtc_y_pred))
dtc_AS = accuracy_score(y_test,dtc_y_pred)
accuracy_scores[3] = accuracy_score(y_test,dtc_y_pred)*100
print('decision tree Classifier accuracy: {}%'.format(accuracy_scores[3]))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Decision Tree Classifier

 Confusion Matrix
[[13  3]
 [ 3 24]]

 Classification Report
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        16
           1       0.89      0.89      0.89        27

    accuracy                           0.86        43
   macro avg       0.85      0.85      0.85        43
weighted avg       0.86      0.86      0.86        43

decision tree Classifier accuracy: 86.04651162790698%
```

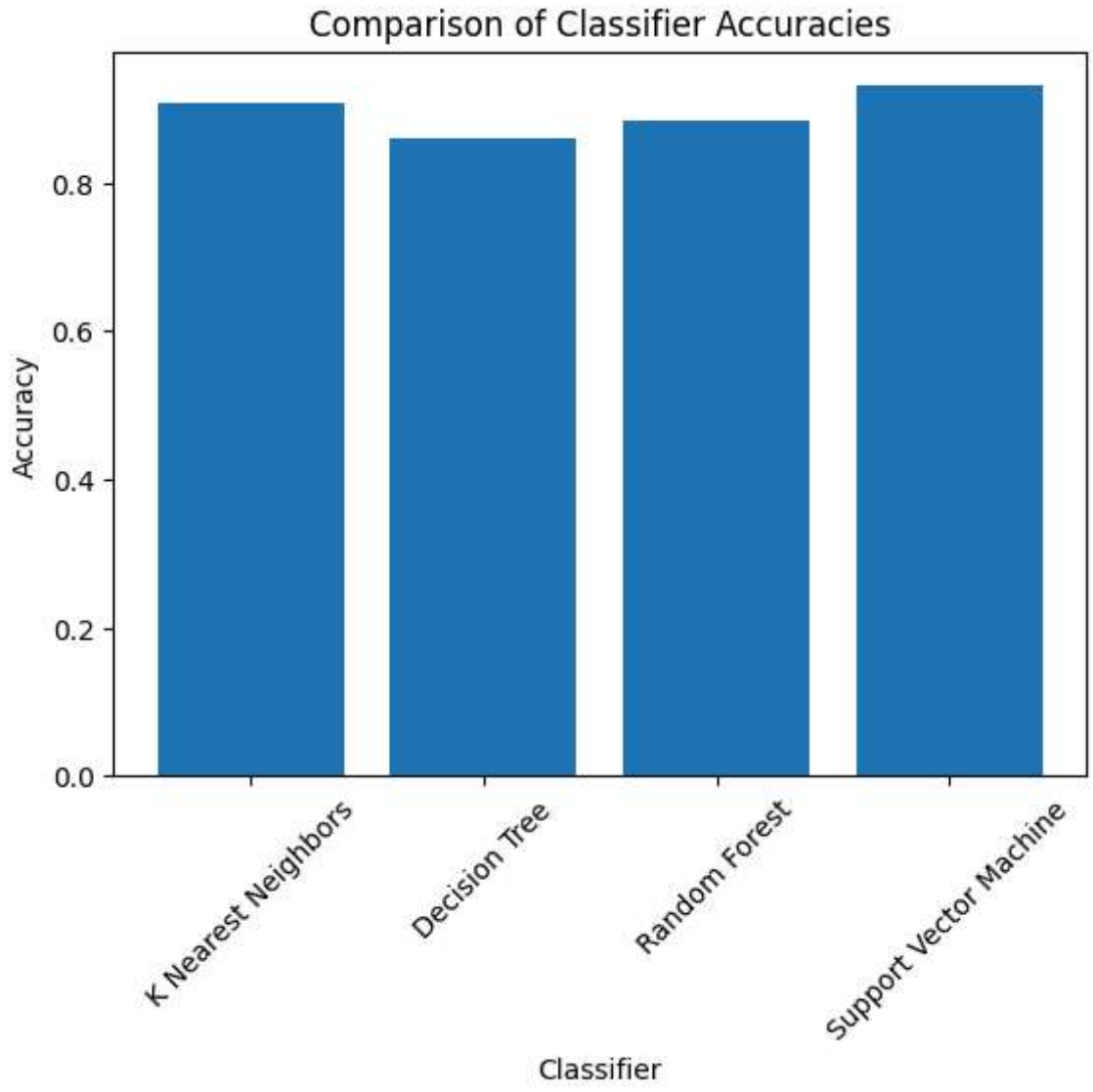
```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py:
425: FitFailedWarning:
12 fits failed out of a total of 30.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sco
re='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
12 fits failed with the following error:
Traceback (most recent call last):
  File "C:\ProgramData\anaconda3\Lib\site-packages\sklearn\model_selection\_valida
tion.py", line 732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py", line 1144, in
wrapper
    estimator._validate_params()
  File "C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py", line 637, in
_validate_params
    validate_parameter_constraints(
  File "C:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils\_param_validatio
n.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramete
r of DecisionTreeClassifier must be an int in the range [1, inf), a float in the r
ange (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:976:
UserWarning: One or more of the test scores are non-finite: [       nan 0.74944545
nan 0.75579754        nan 0.76154467
 0.76729179        nan 0.8256705  0.76719097]
  warnings.warn(
```

In [19]:

```python
#comparision of types of classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Sample Data
# Assuming X_train, X_test, y_train, y_test are defined

# Define classifiers
classifiers = {
    "K Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
}

# Train and evaluate each classifier
results = {}
for name, clf in classifiers.items():
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy

# Print accuracy results
for name, accuracy in results.items():
    print(f"{name}: {accuracy}")
```

```
# Compare accuracy scores visually (optional)
import matplotlib.pyplot as plt

plt.bar(results.keys(), results.values())
plt.xlabel('Classifier')
plt.ylabel('Accuracy')
plt.title('Comparison of Classifier Accuracies')
plt.xticks(rotation=45)
plt.show()
```

```
K Nearest Neighbors: 0.9069767441860465
Decision Tree: 0.8604651162790697
Random Forest: 0.8837209302325582
Support Vector Machine: 0.9302325581395349
```



In [21]:
```
import streamlit as st
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import matplotlib.pyplot as plt

# Title of the Streamlit app
st.title("Machine Learning Classifier Comparison")
```

```
# Upload the dataset
uploaded_file = st.file_uploader(r"C:\Users\sandi\OneDrive\Desktop\final project\ar

if uploaded_file is not None:
    df = pd.read_csv(uploaded_file)

    # Display the dataset
    st.write("Dataset")
    st.write(df.head())

    # Handle categorical variables using one-hot encoding
    df_encoded = pd.get_dummies(df, drop_first=True)

    # Split the dataset into features and labels
    x = df_encoded.drop('status', axis=1)  # Features
    y = df_encoded['status']  # Target variable

    # Split the data into training and testing sets
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random

    # Define classifiers
    classifiers = {
        "K Nearest Neighbors": KNeighborsClassifier(),
        "Decision Tree": DecisionTreeClassifier(),
        "Random Forest": RandomForestClassifier(),
        "Support Vector Machine": SVC(),

    }

    # Train and evaluate each classifier
    results = {}
    for name, clf in classifiers.items():
        clf.fit(x_train, y_train)
        y_pred = clf.predict(x_test)
        accuracy = accuracy_score(y_test, y_pred)
        results[name] = {
            'model': clf,
            'accuracy': accuracy
        }

    # Print accuracy results
    for name, result in results.items():
        st.write(f"{name}: {result['accuracy']}")

    # Find the classifier with the highest accuracy
    best_model_name = max(results, key=lambda name: results[name]['accuracy'])
    best_model = results[best_model_name]['model']
    best_accuracy = results[best_model_name]['accuracy']

    st.write(f"\nBest Model: {best_model_name} with accuracy: {best_accuracy}")

    # Make predictions with the best model
    best_model_predictions = best_model.predict(x_test)

    # Evaluate the best model
    st.write('\nConfusion Matrix')
    st.write(confusion_matrix(y_test, best_model_predictions))
    st.write('\nClassification Report')
    st.write(classification_report(y_test, best_model_predictions))

    # Visualize accuracy comparison
    fig, ax = plt.subplots()
    ax.bar(results.keys(), [result['accuracy'] for result in results.values()])
```

```
    ax.set_xlabel('Classifier')
    ax.set_ylabel('Accuracy')
    ax.set_title('Comparison of Classifier Accuracies')
    ax.set_xticklabels(results.keys(), rotation=45)
    st.pyplot(fig)
```

In [ ]: