1. BANK MARKETING: Predicting Whether The Customer Will Subscribe To Term Deposit (FIXED DEPOSIT) or not.

Business Use Case There has been a revenue decline for a Portuguese bank and they would like to know what actions to take. After investigation, they found out that the root cause is that their clients are not depositing as frequently as before. Knowing that term deposits allow banks to hold onto a deposit for a specific amount of time, so banks can invest in higher gain financial products to make a profit. In addition, banks also hold better chance to persuade term deposit clients into buying other products such as funds or insurance to further increase their revenues. As a result, the Portuguese bank would like to identify existing clients that have higher chance to subscribe for a term deposit and focus marketing efforts on such clients.

Project Description Your client is a retail banking institution. Term deposits are a major source of income for a bank. A term deposit is a cash investment held at a financial institution. Your money is invested for an agreed rate of interest over a fixed amount of time, or term. The bank has various outreach plans to sell term deposits to their customers such as email marketing, advertisements, telephonic marketing and digital marketing. Telephonic marketing campaigns still remain one of the most effective way to reach out to people. However, they require huge investment as large call centers are hired to actually execute these campaigns. Hence, it is crucial to identify the customers most likely to convert beforehand so that they can be specifically targeted via call. You are provided with the client data such as : age of the client, their job type, their marital status, etc. Along with the client data, you are also provided with the information of the call such as the duration of the call, day and month of the call, etc. Given this information, your task is to predict if the client will subscribe to term deposit. About The Dataset The dataset is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal of this dataset is to predict if the client or the customer of polish banking institution will subscribe a term deposit product of the bank or not.

You are provided with following 2 files:

1. train.csv : Use this dataset to train the model. This file contains all the client and call details as well as the target variable “subscribed”. You have to train your model using this file.
2. test.csv : Use the trained model to predict whether a new set of clients will subscribe the term deposit.

Dataset Attributes Here is the description of all the variables: • Variable: Definition • ID: Unique client ID • age: Age of the client • job: Type of job • marital: Marital status of the client • education: Education level • default: Credit in default. • housing: Housing loan • loan: Personal loan • contact: Type of communication • month: Contact month • day_of_week: Day of week of contact • duration: Contact duration • campaign: number of contacts performed during this campaign to the client • pdays: number of days that passed by after the client was last contacted • previous: number of contacts performed before this campaign • poutcome: outcome of the previous marketing campaign Output variable (desired target): • Subscribed (target): has the client subscribed a term deposit? (YES/NO)

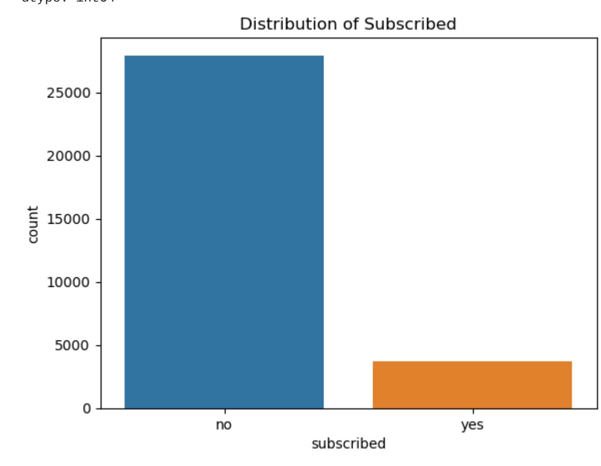
Dataset Link- • https://github.com/dsrscientist/dataset5 • https://raw.githubusercontent.com/dsrscientist/dataset5/main/termdeposit_train.csv • https://raw.githubusercontent.com/dsrscientist/dataset5/main/termdeposit_test.csv

In [59]:
```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

In [60]:
```
# Step 2: Load the Data
trainurl = "https://raw.githubusercontent.com/dsrscientist/dataset5/main/termdeposit_train.csv"
testurl = "https://raw.githubusercontent.com/dsrscientist/dataset5/main/termdeposit_test.csv"

train_df = pd.read_csv(trainurl)
test_df = pd.read_csv(testurl)
```

In [61]:
```
# Step 3: Exploratory Data Analysis (EDA)
# Display basic information about the dataset
print(train_df.info())

# Display summary statistics
print(train_df.describe())

# Check for missing values
print(train_df.isnull().sum())

# Explore the distribution of the target variable
sns.countplot(x='subscribed', data=train_data)
plt.title('Distribution of Subscribed')
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31647 entries, 0 to 31646
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          31647 non-null  int64
 1   age         31647 non-null  int64
 2   job         31647 non-null  object
 3   marital     31647 non-null  object
 4   education   31647 non-null  object
 5   default     31647 non-null  object
 6   balance     31647 non-null  int64
 7   housing     31647 non-null  object
 8   loan        31647 non-null  object
 9   contact     31647 non-null  object
 10  day         31647 non-null  int64
 11  month       31647 non-null  object
 12  duration    31647 non-null  int64
 13  campaign    31647 non-null  int64
 14  pdays       31647 non-null  int64
 15  previous    31647 non-null  int64
 16  poutcome    31647 non-null  object
 17  subscribed  31647 non-null  object
dtypes: int64(8), object(10)
memory usage: 4.3+ MB
None
                 ID           age        balance           day      duration  \
count  31647.000000  31647.000000   31647.000000  31647.000000  31647.000000
mean   22563.972162     40.957247    1363.890258     15.835466    258.113534
std    13075.936990     10.625134    3028.304293      8.337097    257.118973
min        2.000000     18.000000   -8019.000000      1.000000      0.000000
25%    11218.000000     33.000000      73.000000      8.000000    104.000000
50%    22519.000000     39.000000     450.000000     16.000000    180.000000
75%    33879.500000     48.000000    1431.000000     21.000000    318.500000
max    45211.000000     95.000000  102127.000000     31.000000   4918.000000

           campaign         pdays      previous
count  31647.000000  31647.000000  31647.000000
mean       2.765697     39.576042      0.574272
std        3.113830     99.317592      2.422529
min        1.000000     -1.000000      0.000000
25%        1.000000     -1.000000      0.000000
50%        2.000000     -1.000000      0.000000
75%        3.000000     -1.000000      0.000000
max       63.000000    871.000000    275.000000
ID            0
age           0
job           0
marital       0
education     0
default       0
balance       0
housing       0
loan          0
contact       0
day           0
month         0
duration      0
campaign      0
pdays         0
previous      0
poutcome      0
subscribed    0
dtype: int64
```



In [62]:
```
# Step 4: Data Preprocessing
# Handle missing values if any
train_df.fillna(method='ffill', inplace=True)

# Encode categorical variables
encoder = LabelEncoder()

# Adjust the list of categorical columns based on the actual columns in the dataset
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome']

for col in categorical_cols:
    try:
        train_df[col] = label_encoder.fit_transform(train_data[col])
    except KeyError:
        # Handle unseen labels by adding 'unknown' to the classes
        encoder.classes_ = np.append(encoder.classes_, 'unknown')
        train_df[col] = encoder.transform(train_df[col])

# Split the data into features and target variable
X = train_df.drop(['ID', 'subscribed'], axis=1)
y = train_df['subscribed']

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

In [63]:
```
# Step 5: Model Training
# Initialize and train the RandomForestClassifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

Out[63]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

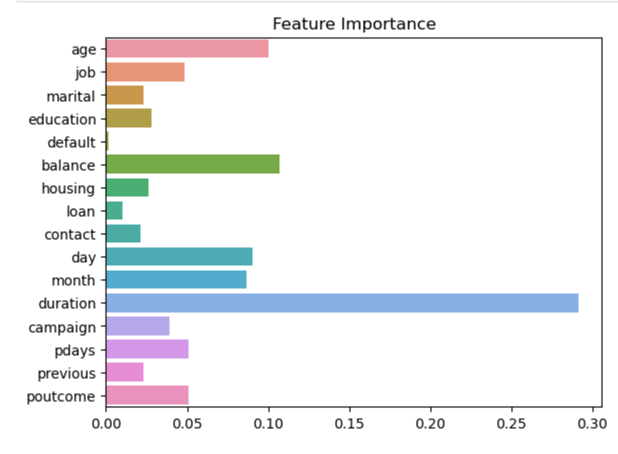
In [64]:
```
# Step 6: Model Evaluation
# Predict on the validation set
y_pred = model.predict(X_val)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred))
print("\nClassification Report:\n", classification_report(y_val, y_pred))
print("Accuracy:", accuracy_score(y_val, y_pred))
```

```
Confusion Matrix:
 [[5418  181]
 [ 415  316]]

Classification Report:
               precision    recall  f1-score   support

          no       0.93      0.97      0.95      5599
         yes       0.64      0.43      0.51       731

    accuracy                           0.91      6330
   macro avg       0.78      0.70      0.73      6330
weighted avg       0.90      0.91      0.90      6330

Accuracy: 0.9058451816745655
```

In [65]:
```
# Handle the case where there is a new category in the test data
if 'unknown' in label_encoder.classes_:
    # 'unknown' is already in classes, use its label
    test_df[col] = label_encoder.transform(['unknown'])[0]
else:
    # Add 'unknown' to the classes and use its label
    label_encoder.classes_ = np.append(label_encoder.classes_, 'unknown')
    test_df[col] = label_encoder.transform(['unknown'])[0]
```

```
C:\Users\nishi\AppData\Local\Temp\ipykernel_26400\1026747192.py:2: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  if 'unknown' in label_encoder.classes_:
```

In [49]:
```
feature_importances = model.feature_importances_
features = X.columns
sns.barplot(x=feature_importances, y=features)
plt.title('Feature Importance')
plt.show()
```



2. Student grades prediction

Project Description The dataset contains grades scored by students throughout their university tenure in various courses and their CGPA calculated based on their grades Columns Description- total 43 columns -Seat No : The enrolled number of candidate that took the exams

CGPA : The cumulative GPA based on the four year total grade progress of each candidate . CGPA is a Final Marks -- provided to student.

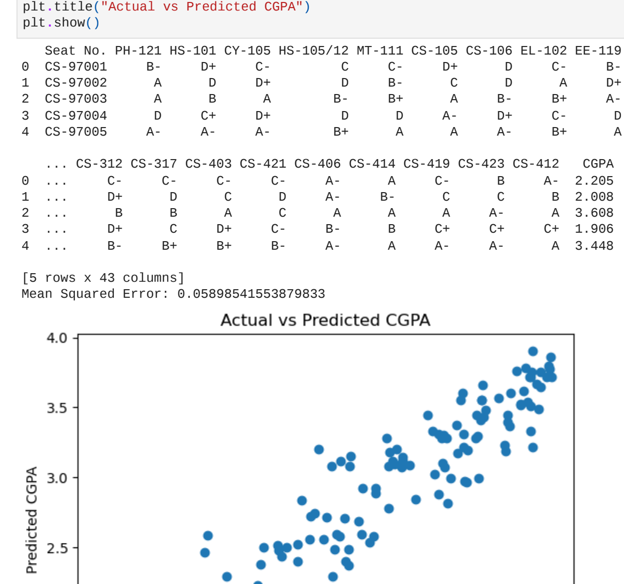
· All other columns are course codes in the format AB-XXX where AB are alphabets representing candidates' departments and XXX are numbers where first X represents the year the canditate took exam

Predict - CGPA of a student based on different grades in four years.

Dataset Link- • https://github.com/dsrscientist/dataset4 • https://github.com/dsrscientist/dataset4/blob/main/Grades.csv

In [70]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Load the dataset
link = "https://raw.githubusercontent.com/dsrscientist/dataset4/main/Grades.csv"
data = pd.read_csv(url)

# Display the first few rows of the dataset
print(data.head())

# Select relevant columns for prediction
# Assuming you want to use grades in four years as features
feature = ['PH-121', 'HS-101', 'CY-105', 'HS-105/12', 'MT-111', 'CS-105', 'CS-106', 'EL-102', 'EE-119']

# Convert grades to numerical values using Label Encoding
encoder = LabelEncoder()
data[feature] = data[feature].apply(encoder.fit_transform)

# X contains the features, and y contains the target variable (CGPA)
X = data[feature]
y = data['CGPA']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate Mean Squared Error to evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Visualize the predicted vs actual CGPA
plt.scatter(y_test, y_pred)
plt.xlabel("Actual CGPA")
plt.ylabel("Predicted CGPA")
plt.title("Actual vs Predicted CGPA")
plt.show()
```

```
  Seat No. PH-121 HS-101 CY-105 HS-105/12 MT-111 CS-105 CS-106 EL-102 EE-119  \
0  CS-97001     B-     D+     C-         C     C-     D+      D     C-     B-
1  CS-97002      A      D     D+         D     B-      C      D      A     D+
2  CS-97003      A      B      A        B-     B+      A     B-     B+     A-
3  CS-97004      D     C+     D+         D      D     A-     D+     C-      D
4  CS-97005     A-     A-     A-        B+      A      A     A-     B+      A

   ... CS-312 CS-317 CS-403 CS-421 CS-406 CS-414 CS-419 CS-423 CS-412   CGPA
0  ...     C-     C-     C-     C-     A-      A     C-      B     A-  2.205
1  ...     D+      D      C      D     A-     B-      C      C      B  2.008
2  ...      B      B      A      C      A      A      A     A-      A  3.608
3  ...     D+      C     D+     C-     B-      B     C+     C+     C+  1.906
4  ...     B-     B+     B+     B-     A-      A     A-     A-      A  3.448

[5 rows x 43 columns]
Mean Squared Error: 0.05898541553879833
```



Glass Identification

Project Description The dataset describes the chemical properties of glass and involves classifying samples of glass using their chemical properties as one of six classes. The dataset was credited to Vina Spiehler in 1987. The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!

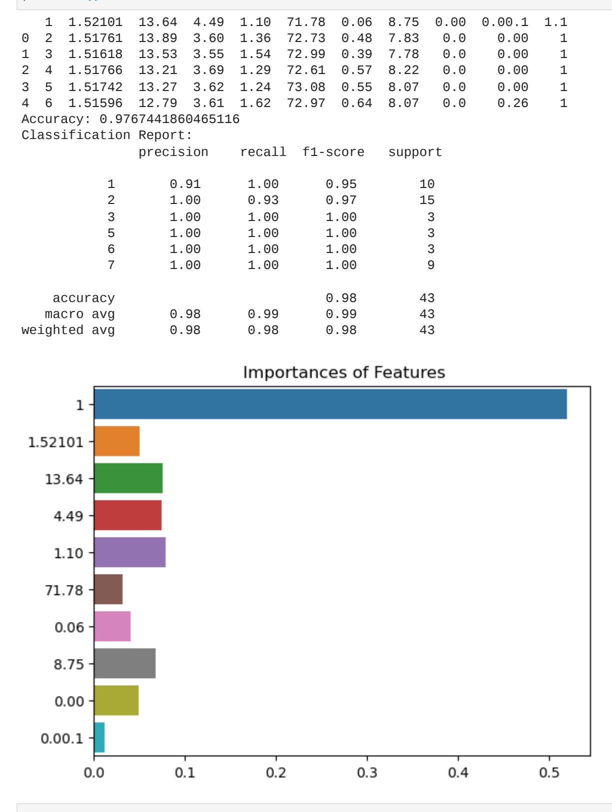
The chemical compositions are measured as the weight percent in corresponding oxide. Attribute Information-

1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass: (class attribute) • 1- building_windows_float_processed • 2- building_windows_non_float_processed • 3- vehicle_windows_float_processed • 4- vehicle_windows_non_float_processed (none in this database) • 5- containers • 6- tableware • 7- headlamps There are 214 observations in the dataset. The dataset can be divided into window glass (classes 1-4) and non-window glass (classes 5-7). Predict : Type of glass

Dataset Link- • https://raw.githubusercontent.com/dsrscientist/dataset3/main/glass.csv • https://github.com/dsrscientist/dataset3

In [87]:
```
# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
link = "https://raw.githubusercontent.com/dsrscientist/dataset3/main/glass.csv"
glassdf = pd.read_csv(url)

# Display the first few rows of the dataset
print(glassdf.head())

# Data Preprocessing
glassdf.columns = glass_data.columns.str.strip()  # Trim whitespace from column names

# Assuming the correct column name is '1.1', replace 'Type of glass' with the correct name
y = glassdf['1.1']  # Target variable

# Split the data into features (X) and target variable (y)
X = glassdf.drop(['1.1'], axis=1)  # Features

# 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_state=42)

# Initialize the Random Forest Classifier
clf = RandomForestClassifier(random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_report_result = classification_report(y_test, y_pred)

# Display the results
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", classification_report_result)

# Plot a bar graph of the feature importances
feature_significance = clf.feature_importances_
features = X.columns
sns.barplot(x=feature_significance, y=features)
plt.title(' Importances of Features')
plt.show()
```

```
   1  1.52101  13.64  4.49  1.10  71.78  0.06  8.75  0.00  0.00.1  1.1
0  2  1.51761  13.89  3.60  1.36  72.73  0.48  7.83   0.0    0.00    1
1  3  1.51618  13.53  3.55  1.54  72.99  0.39  7.78   0.0    0.00    1
2  4  1.51766  13.21  3.69  1.29  72.61  0.57  8.22   0.0    0.00    1
3  5  1.51742  13.27  3.62  1.24  73.08  0.55  8.07   0.0    0.00    1
4  6  1.51596  12.79  3.61  1.62  72.97  0.64  8.07   0.0    0.26    1
Accuracy: 0.9767441860465116
Classification Report:
               precision    recall  f1-score   support

           1       0.91      1.00      0.95        10
           2       1.00      0.93      0.97        15
           3       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      1.00      1.00         9

    accuracy                           0.98        43
   macro avg       0.98      0.99      0.99        43
weighted avg       0.98      0.98      0.98        43
```



In [ ]: