In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
from scipy import stats
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split, RandomizedSearch
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classifi
from sklearn.utils.class_weight import compute_class_weight
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
```

In [3]:

```
# read in data - this is a CSV file of general features for around 730
data_path = "ALL_full_star_table_all_types.txt"
data = pd.read_csv(data_path, sep='\t', skiprows=6, header=0)
```

```
/opt/apps/intel19/python3/3.9.7/lib/python3.9/site-packages/IPython/c
ore/interactiveshell.py:3444: DtypeWarning: Columns (6,36) have mixed
types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [4]:
```
data
```

Out[4]:

	# V_I	ID	Type	Subtype	RA	DECL	ID_OGLE_II	ID_OGLE_III
0	3.464	OGLE-BLG-CEP-001	Cep	F	17.570842	-27.398250	-99.99	BLG333.8.30568
1	1.855	OGLE-BLG-CEP-002	Cep	F	17.632956	-22.503361	-99.99	BLG336.2.114493
2	1.700	OGLE-BLG-CEP-003	Cep	F1	17.745497	-23.723639	-99.99	BLG344.2.150516
3	1.579	OGLE-BLG-CEP-004	Cep	12	17.763842	-33.768778	-99.99	BLG138.1.170393
4	2.301	OGLE-BLG-CEP-005	Cep	F	17.818625	-23.121861	-99.99	BLG343.2.121449
...	...	...	...	...	...	...	...	...
735937	0.884	OGLE-SMC-T2CEP-50	T2Cep	WVir	1.028956	-75.017250	-99.99	-99.99
735938	0.635	OGLE-SMC-T2CEP-51	T2Cep	BLHer	1.102853	-71.079444	-99.99	-99.99
735939	0.637	OGLE-SMC-T2CEP-52	T2Cep	BLHer	1.161111	-70.477722	-99.99	-99.99
735940	0.970	OGLE-SMC-T2CEP-53	T2Cep	WVir	1.214036	-74.588444	-99.99	-99.99
735941	0.649	OGLE-SMC-T2CEP-54	T2Cep	BLHer	1.319739	-75.067194	-99.99	-99.99

735942 rows × 38 columns

In [5]:
```
data.rename(columns={'# V_I': 'V_I'}, inplace=True)
```

In [6]:
```
labels = data["Type"]
```

In [7]:
```
np.unique(labels)
```

Out[7]:
```
array(['Cep', 'DSCT', 'ECL', 'HB', 'LPV', 'RRLyr', 'T2Cep', 'aCep'],
      dtype=object)
```

In [9]:
```
data.head()
```

Out[9]:

	V_I	ID	Type	Subtype	RA	DECL	ID_OGLE_II	ID_OGLE_III	ID_
0	3.464	OGLE-BLG-CEP-001	Cep	F	17.570842	-27.398250	-99.99	BLG333.8.30568	BLG61
1	1.855	OGLE-BLG-CEP-002	Cep	F	17.632956	-22.503361	-99.99	BLG336.2.114493	BLG62
2	1.700	OGLE-BLG-CEP-003	Cep	F1	17.745497	-23.723639	-99.99	BLG344.2.150516	BLG632.
3	1.579	OGLE-BLG-CEP-004	Cep	12	17.763842	-33.768778	-99.99	BLG138.1.170393	BLG60
4	2.301	OGLE-BLG-CEP-005	Cep	F	17.818625	-23.121861	-99.99	BLG343.2.121449	

5 rows × 38 columns

In [10]:
```
data.shape
```

Out[10]:
```
(735942, 38)
```

In [11]:
```
labels
```

Out[11]:
```
0           Cep
1           Cep
2           Cep
3           Cep
4           Cep
          ...
735937    T2Cep
735938    T2Cep
735939    T2Cep
735940    T2Cep
735941    T2Cep
Name: Type, Length: 735942, dtype: object
```

In [12]:
```
# counts of each variable star type
type_counts = data['Type'].value_counts()
type_counts, type_counts / type_counts.sum()
```

Out[12]:
```
(ECL      499203
 RRLyr    128273
 LPV       65981
 DSCT      27392
 Cep       11703
 T2Cep      2010
 HB          991
 aCep        389
 Name: Type, dtype: int64,
 ECL      0.678318
 RRLyr    0.174298
 LPV      0.089655
 DSCT     0.037220
 Cep      0.015902
 T2Cep    0.002731
 HB       0.001347
 aCep     0.000529
 Name: Type, dtype: float64)
```

In [13]:
```
# replacing -99.99s with NaNs
data = data.replace(-99.99, np.nan)
data = data.replace("-99.99", np.nan)
```

In [14]:
```
high_nan = data.columns[data.isna().mean() > 0.2]

# create a DataFrame with non-NaN columns
df_filtered = data.drop(columns=high_nan)
df_filtered.head()
```

Out[14]:

	ID	Type	Subtype	RA	DECL	ID_OGLE_IV	I	P_1	T0
0	OGLE-BLG-CEP-001	Cep	F	17.570842	-27.398250	BLG611.14.36983	17.395	2.597573	7002.541
1	OGLE-BLG-CEP-002	Cep	F	17.632956	-22.503361	BLG625.32.78667	15.734	2.025573	7000.984
2	OGLE-BLG-CEP-003	Cep	F1	17.745497	-23.723639	BLG632.13.133301	16.424	1.235729	7000.555
3	OGLE-BLG-CEP-004	Cep	12	17.763842	-33.768778	BLG603.29.45415	16.178	0.240046	7000.165
4	OGLE-BLG-CEP-005	Cep	F	17.818625	-23.121861	NaN	15.374	3.795593	7002.172

In [15]:
```
non_numeric_columns = df_filtered.select_dtypes(exclude=['number'])
df_numeric = df_filtered.drop(columns=non_numeric_columns)
df_numeric
```

Out[15]:

	RA	DECL	I	P_1	T0_1	A_1
0	17.570842	-27.398250	17.395	2.597573	7002.54120	0.523
1	17.632956	-22.503361	15.734	2.025573	7000.98498	0.730
2	17.745497	-23.723639	16.424	1.235729	7000.55567	0.046
3	17.763842	-33.768778	16.178	0.240046	7000.16541	0.110
4	17.818625	-23.121861	15.374	3.795593	7002.17287	0.409
...	...	...	...	...	...	...
735937	1.028956	-75.017250	17.464	4.227618	7001.52773	0.299
735938	1.102853	-71.079444	18.708	1.065770	7000.11577	0.288
735939	1.161111	-70.477722	18.147	1.746251	7000.80591	0.444
735940	1.214036	-74.588444	16.307	14.912622	7013.31086	0.631
735941	1.319739	-75.067194	17.891	1.842325	7000.66340	0.699

735942 rows × 6 columns

In [16]:
```
features = ['T0_1', 'A_1', 'I', 'P_1']
data = df_numeric[features]
```

In [17]:
```
data
```

Out[17]:

	T0_1	A_1	I	P_1
0	7002.54120	0.523	17.395	2.597573
1	7000.98498	0.730	15.734	2.025573
2	7000.55567	0.046	16.424	1.235729
3	7000.16541	0.110	16.178	0.240046
4	7002.17287	0.409	15.374	3.795593
...	...	...	...	...
735937	7001.52773	0.299	17.464	4.227618
735938	7000.11577	0.288	18.708	1.065770
735939	7000.80591	0.444	18.147	1.746251
735940	7013.31086	0.631	16.307	14.912622
735941	7000.66340	0.699	17.891	1.842325

735942 rows × 4 columns

In [18]:
```
# dropping NaNs from dataframe
nan_indices = data[data.isna().any(axis=1)].index
data = data.dropna()
```

In [19]:
```
data
```

Out[19]:

	T0_1	A_1	I	P_1
0	7002.54120	0.523	17.395	2.597573
1	7000.98498	0.730	15.734	2.025573
2	7000.55567	0.046	16.424	1.235729
3	7000.16541	0.110	16.178	0.240046
4	7002.17287	0.409	15.374	3.795593
...	...	...	...	...
735937	7001.52773	0.299	17.464	4.227618
735938	7000.11577	0.288	18.708	1.065770
735939	7000.80591	0.444	18.147	1.746251
735940	7013.31086	0.631	16.307	14.912622
735941	7000.66340	0.699	17.891	1.842325

669897 rows × 4 columns

In [20]:
```
# removing corresponding indices from labels
labels = [label for i, label in enumerate(labels) if i not in nan_indi
classes = np.unique(labels)
data = data.reset_index(drop=True)
```

In [21]:
```
data
```

Out[21]:

	T0_1	A_1	I	P_1
0	7002.54120	0.523	17.395	2.597573
1	7000.98498	0.730	15.734	2.025573
2	7000.55567	0.046	16.424	1.235729
3	7000.16541	0.110	16.178	0.240046
4	7002.17287	0.409	15.374	3.795593
...	...	...	...	...
669892	7001.52773	0.299	17.464	4.227618
669893	7000.11577	0.288	18.708	1.065770
669894	7000.80591	0.444	18.147	1.746251
669895	7013.31086	0.631	16.307	14.912622
669896	7000.66340	0.699	17.891	1.842325

669897 rows × 4 columns

In [22]:
```
classes
```

Out[22]:
```
array(['Cep', 'DSCT', 'ECL', 'HB', 'RRLyr', 'T2Cep', 'aCep'], dtype
='<U5')
```

In [23]:
```
data.shape, len(labels)
```

Out[23]:
```
((669897, 4), 669897)
```

In [24]:
```
encoder = LabelEncoder()
labels_encoded = encoder.fit_transform(labels)
labels_encoded
```

Out[24]:
```
array([0, 0, 0, ..., 5, 5, 5])
```

In [25]:
```
# smote instance
smote = SMOTE(sampling_strategy='auto', random_state=21)
X_resampled, y_resampled = smote.fit_resample(data, labels_encoded)
```

In [26]:
```
# Count the class distribution before and after SMOTE
print("Class distribution before SMOTE:\n", pd.Series(labels).value_co
print("Class distribution after SMOTE:\n", pd.Series(y_resampled).valu
```

```
Class distribution before SMOTE:
 ECL      499181
RRLyr    128272
DSCT      27392
Cep       11662
T2Cep      2010
HB          991
aCep        389
dtype: int64
Class distribution after SMOTE:
 0    499181
1    499181
2    499181
3    499181
4    499181
5    499181
6    499181
dtype: int64
```

In [27]:
```
X_train, X_temp, y_train, y_temp = train_test_split(X_resampled,
                                                    y_resampled,
                                                    test_size=0.3,
                                                    random_state=21)

X_val, X_test, y_val, y_test = train_test_split(X_temp,
                                                y_temp,
                                                test_size=0.5,
                                                random_state=21)
```

In [28]:
```
[len(dataset) for dataset in [X_train, y_train, X_val, X_test, y_val,
```

Out[28]:
```
[2445986, 2445986, 524140, 524141, 524140, 524141]
```

In [29]:
```
# hyperparameter tuning for random forest
param_dist = {
    "n_estimators": [100, 200, 300, 400],
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": [2, 5, 10, 15],
    "min_samples_leaf": [1, 2, 4, 8]
}

rf_tuning = RandomForestClassifier(random_state=21)

rand_search = RandomizedSearchCV(estimator=rf_tuning,
                                 param_distributions=param_dist,
                                 n_iter=20,
                                 cv=5,
                                 scoring="accuracy",
                                 random_state=21,
                                 n_jobs=-1)

rand_search.fit(X_val, y_val)

best_params = rand_search.best_params_
best_params
```

Out[29]:
```
{'n_estimators': 400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_depth': 30}
```

In [30]:
```
rf = RandomForestClassifier(n_estimators=400,
                            min_samples_split=2,
                            min_samples_leaf=1,
                            max_depth=30,
                            random_state=21)
rf.fit(X_train, y_train)
```

Out[30]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=30, n_estimators=400, random_state=
21)
```

In [31]:
```
predictions = rf.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
accuracy * 100
```

Out[31]:
```
97.71301996981728
```

In [33]:
```
conf_mat = confusion_matrix(y_test, predictions)
conf_mat_pct = conf_mat.astype("float") / conf_mat.sum(axis=1)[:, np.n

plt.figure(figsize=(16, 14))
ax = sns.heatmap(conf_mat_pct, fmt=".2f", cmap="Blues", cbar=False,
            xticklabels=classes,  # Predicted
            yticklabels=classes)

for i in range(len(classes)):
    for j in range(len(classes)):
        count = conf_mat[i, j]
        percent = conf_mat_pct[i, j]
        text = f"{count} ({percent:.2f}%)"
        color = 'white' if i == j else 'black'  # White for diagonal,
        ax.text(j + 0.5, i + 0.5, text, ha='center', va='center', colo

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Random Forest Confusion Matrix")
plt.show()
```

In [35]:
```
import pickle
```

In [36]:
```
with open('conf_mat_2_RF.pkl','wb') as f:
     pickle.dump(conf_mat, f)
```

In [37]:
```
with open('conf_mat_pct_RF.pkl','wb') as f:
     pickle.dump(conf_mat_pct, f)
```

In [ ]: