Date Fruit Classification

In [23]:
```
! pip install openpyxl
```

```
Requirement already satisfied: openpyxl in /opt/conda/lib/python3.7/site-packages (3.1.2)
Requirement already satisfied: et-xmlfile in /opt/conda/lib/python3.7/site-packages (from openpyxl) (1.1.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system pack
age manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

Importing Libraries

In [24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.transforms as transforms
import seaborn as sns
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from collections import OrderedDict
import warnings
warnings.filterwarnings("ignore")

plt.rcParams["figure.figsize"] = (10,6)
plt.rcParams['figure.dpi'] = 300
plt.rcParams['axes.facecolor']='#CDE2E5'

colors = ["#49C3CB", "#7D49CB", "#FFC156", "#365C89",
              "#E9937E", "#9EE97E", "#F384BC"]
```

In [25]:
```
df = pd.read_excel('/input/Date_Fruit_Datasets/Date_Fruit_Datasets.xlsx')
```

Data Analysis and Visualizations

In [26]:
```
df.head(2).style.set_properties(**{'background-color': '#EED4CF',
                           'color': 'black'})
```

Out[26]:

	AREA	PERIMETER	MAJOR_AXIS	MINOR_AXIS	ECCENTRICITY	EQDIASQ	SOLIDITY	CONVEX_AREA	EXTENT	ASPECT_RATIO	ROUNDN
0	422163	2378.908000	837.848400	645.669300	0.637300	733.153900	0.994700	424428	0.783100	1.297600	0.93
1	338136	2085.144000	723.819800	595.207300	0.569000	656.146400	0.997400	339014	0.779500	1.216100	0.97

In [27]:
```
df.shape
```

Out[27]:
```
(898, 35)
```

In [28]:
```
df.isnull().sum().to_frame()
```

Out[28]:

	0
AREA	0
PERIMETER	0
MAJOR_AXIS	0
MINOR_AXIS	0
ECCENTRICITY	0
EQDIASQ	0
SOLIDITY	0
CONVEX_AREA	0
EXTENT	0
ASPECT_RATIO	0
ROUNDNESS	0
COMPACTNESS	0
SHAPEFACTOR_1	0
SHAPEFACTOR_2	0
SHAPEFACTOR_3	0
SHAPEFACTOR_4	0
MeanRR	0
MeanRG	0
MeanRB	0
StdDevRR	0
StdDevRG	0
StdDevRB	0
SkewRR	0
SkewRG	0
SkewRB	0
KurtosisRR	0
KurtosisRG	0
KurtosisRB	0
EntropyRR	0
EntropyRG	0
EntropyRB	0
ALLdaub4RR	0
ALLdaub4RG	0
ALLdaub4RB	0
Class	0

In [29]:
```
df.describe().style.set_properties(**{'background-color': '#EED4CF',
                                      'color': 'black'})
```

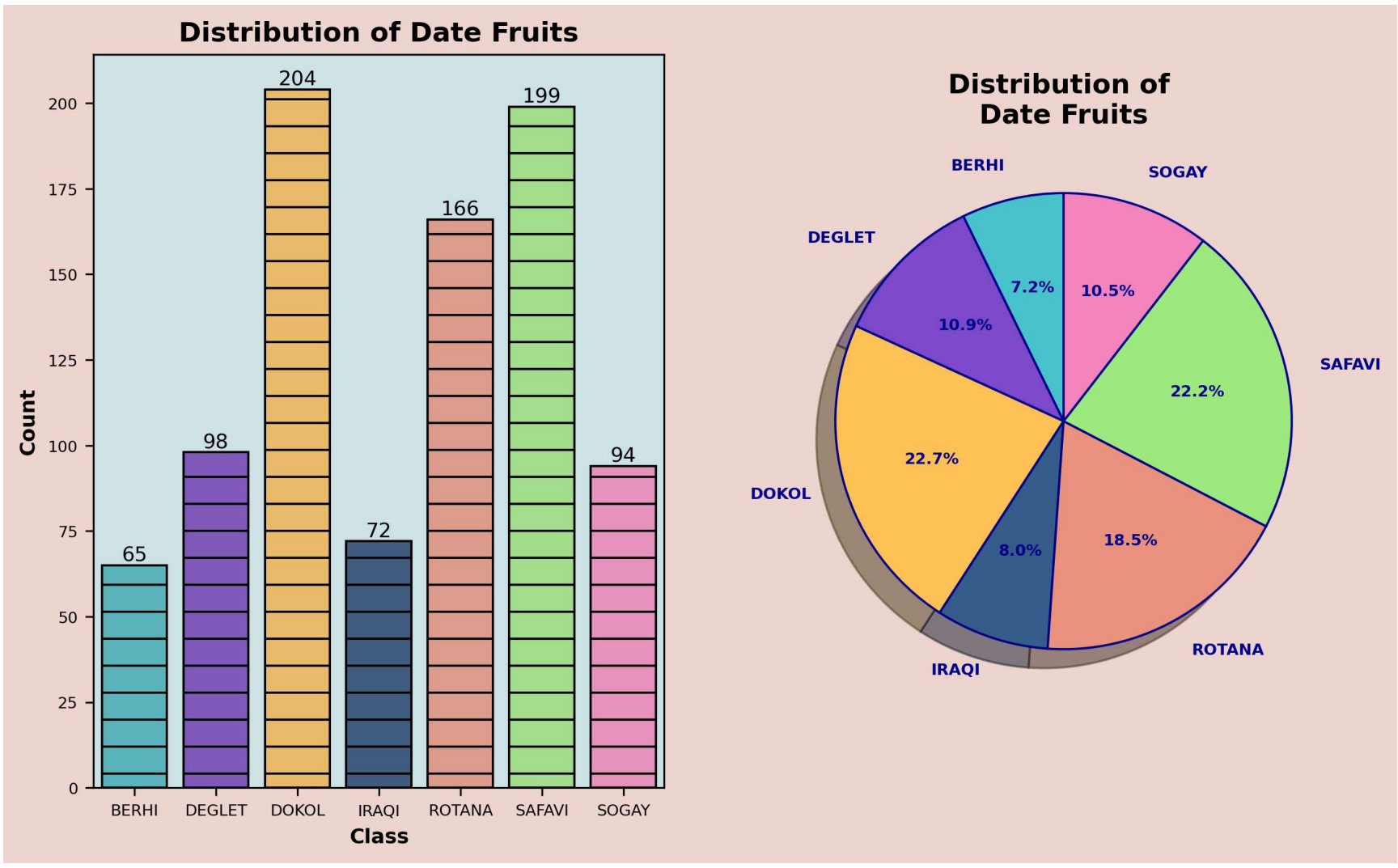
Out[29]:

	AREA	PERIMETER	MAJOR_AXIS	MINOR_AXIS	ECCENTRICITY	EQDIASQ	SOLIDITY	CONVEX_AREA	EXTENT	ASPECT_R
count	898.000000	898.000000	898.000000	898.000000	898.000000	898.000000	898.000000	898.000000	898.000000	898.00
mean	298295.207127	2057.660953	750.811994	495.872785	0.737468	604.577938	0.981840	303845.592428	0.736267	2.13
std	107245.205337	410.012459	144.059326	114.268917	0.088727	119.593888	0.018157	108815.656947	0.053745	17.82
min	1987.000000	911.828000	336.722700	2.283200	0.344800	50.298400	0.836600	2257.000000	0.512300	1.06
25%	206948.000000	1726.091500	641.068650	404.684375	0.685625	513.317075	0.978825	210022.750000	0.705875	1.37
50%	319833.000000	2196.345450	791.363400	495.054850	0.754700	638.140950	0.987300	327207.000000	0.746950	1.52
75%	382573.000000	2389.716575	858.633750	589.031700	0.802150	697.930525	0.991800	388804.000000	0.775850	1.67
max	546063.000000	2811.997100	1222.723000	766.453600	1.000000	833.827900	0.997400	552598.000000	0.856200	535.52

In [30]:
```
type_dict = df["Class"].value_counts().to_dict()
type_dict = OrderedDict(sorted(type_dict.items()))
type_label = type_dict.keys()
type_value = type_dict.values()
explode = (0,)*df["Class"].nunique()

fig, ax = plt.subplots(1,2, facecolor='#EED4CF')
sns.countplot(data=df, x="Class", palette=colors, edgecolor="black", hatch="-", ax=ax[0])
ax[0].set_title("Distribution of Date Fruits", size=12, fontweight="bold")
ax[0].set_xlabel("Class", size=9, fontweight="bold")
ax[0].set_ylabel("Count", size=9, fontweight="bold")
```

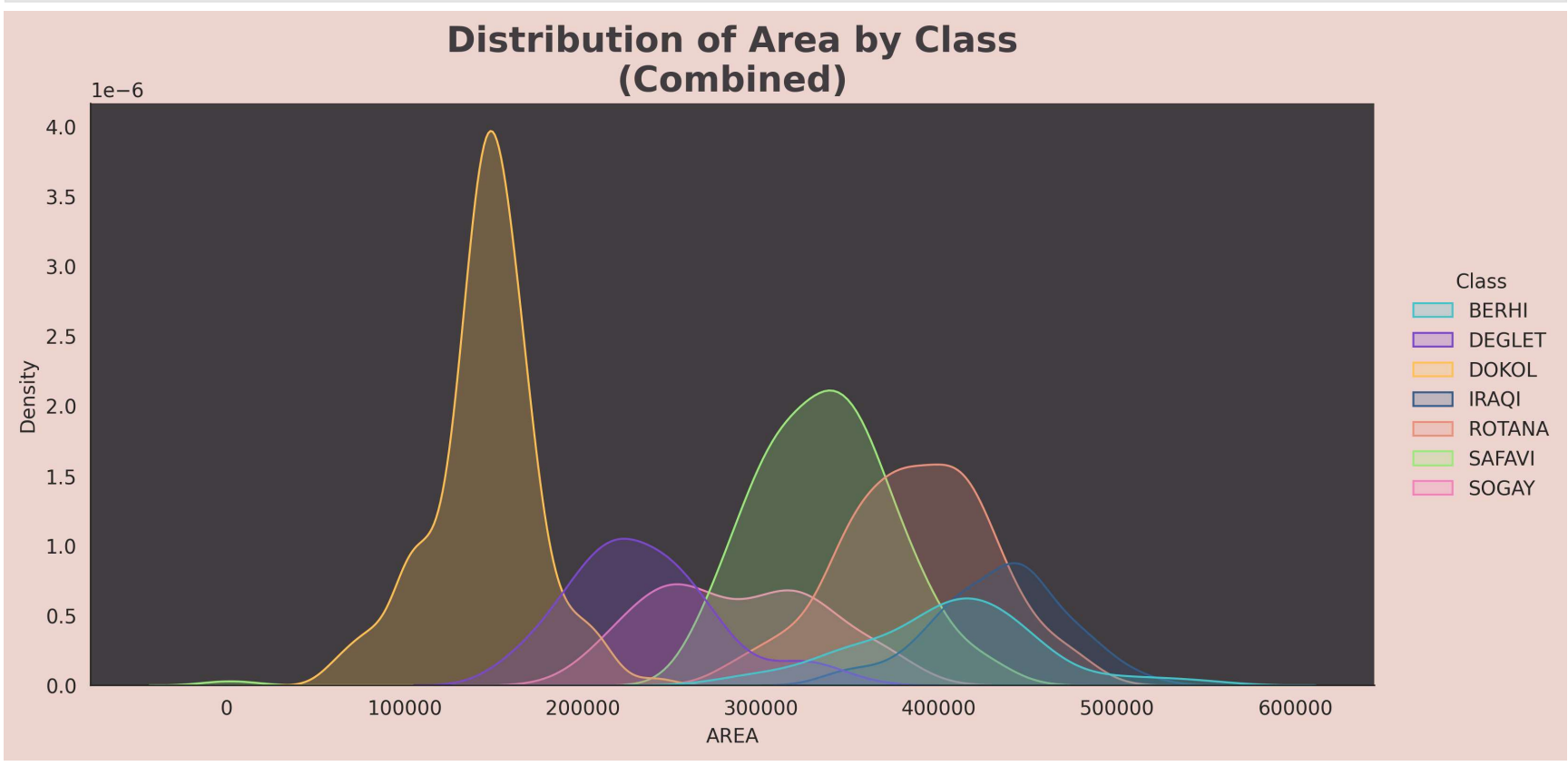
```
ax[0].tick_params(axis="x", labelsize=7)
ax[0].tick_params(axis="y", labelsize=7)
ax[0].bar_label(ax[0].containers[0], fmt='%.0f', color="black", fontsize=9)

plt.pie(type_value, explode=explode, labels=type_label, autopct='%1.1f%%',
        shadow=True, startangle=90, textprops={'fontsize': 7, "fontweight" : "bold", "color":"darkblue"},  wedgeprops=
       {'edgecolor':'darkblue'} ,colors=colors, labeldistance=1.15)
plt.title("Distribution of \nDate Fruits", size=12, fontweight="bold")
```

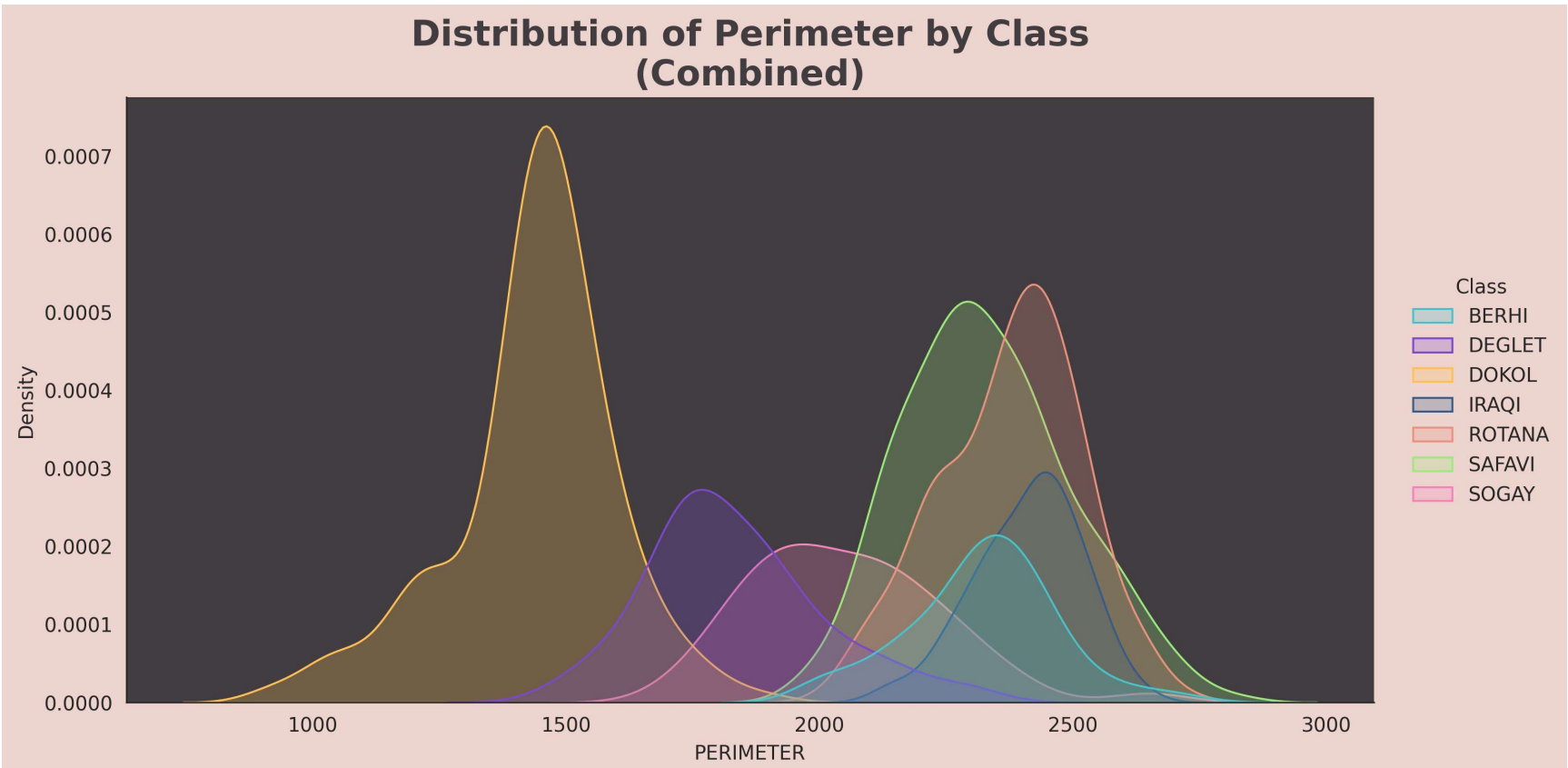
Out[30]: Text(0.5, 1.0, 'Distribution of \nDate Fruits')

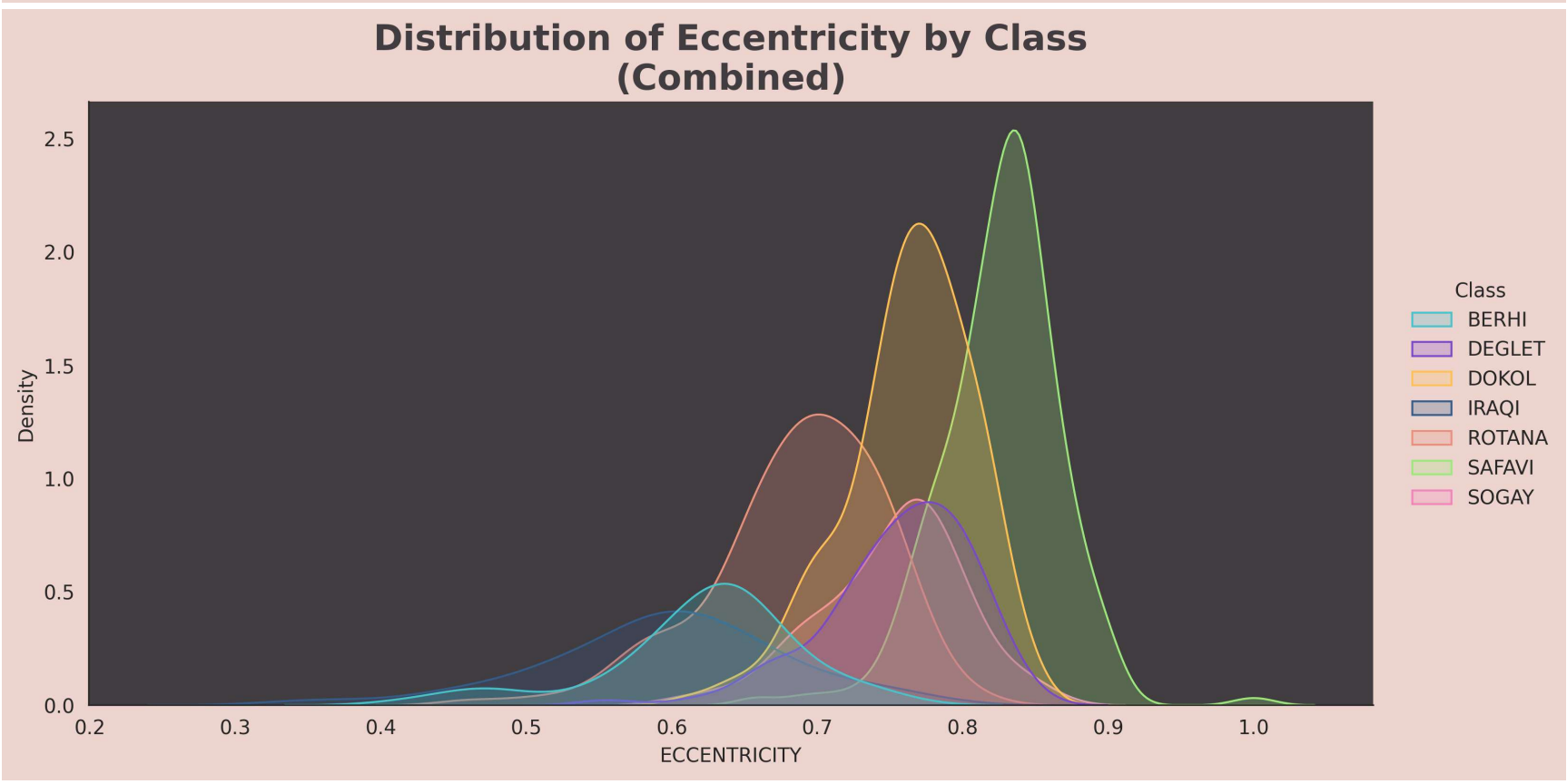


Let's select some features to compare the distributions between the classes. The number of features might increase for other analyses. To avoid overwhelming readers with too many figures, I've chosen only three features (Area, Perimeter, and Eccentricity).

In [31]:
```
dis_feat = ["AREA", "PERIMETER", "ECCENTRICITY"]

for i in dis_feat:
    with sns.axes_style("white", rc={"axes.facecolor":"#413D41",'figure.facecolor':'#EED4CF'}):
        sns.displot(data=df, x=i, hue="Class", kind="kde", palette=colors, fill=True, legend=True
                                ,aspect=2)
        plt.title("Distribution of " + i.capitalize() + " by Class\n(Combined)", size=18, fontweight="bold", color="#41
```

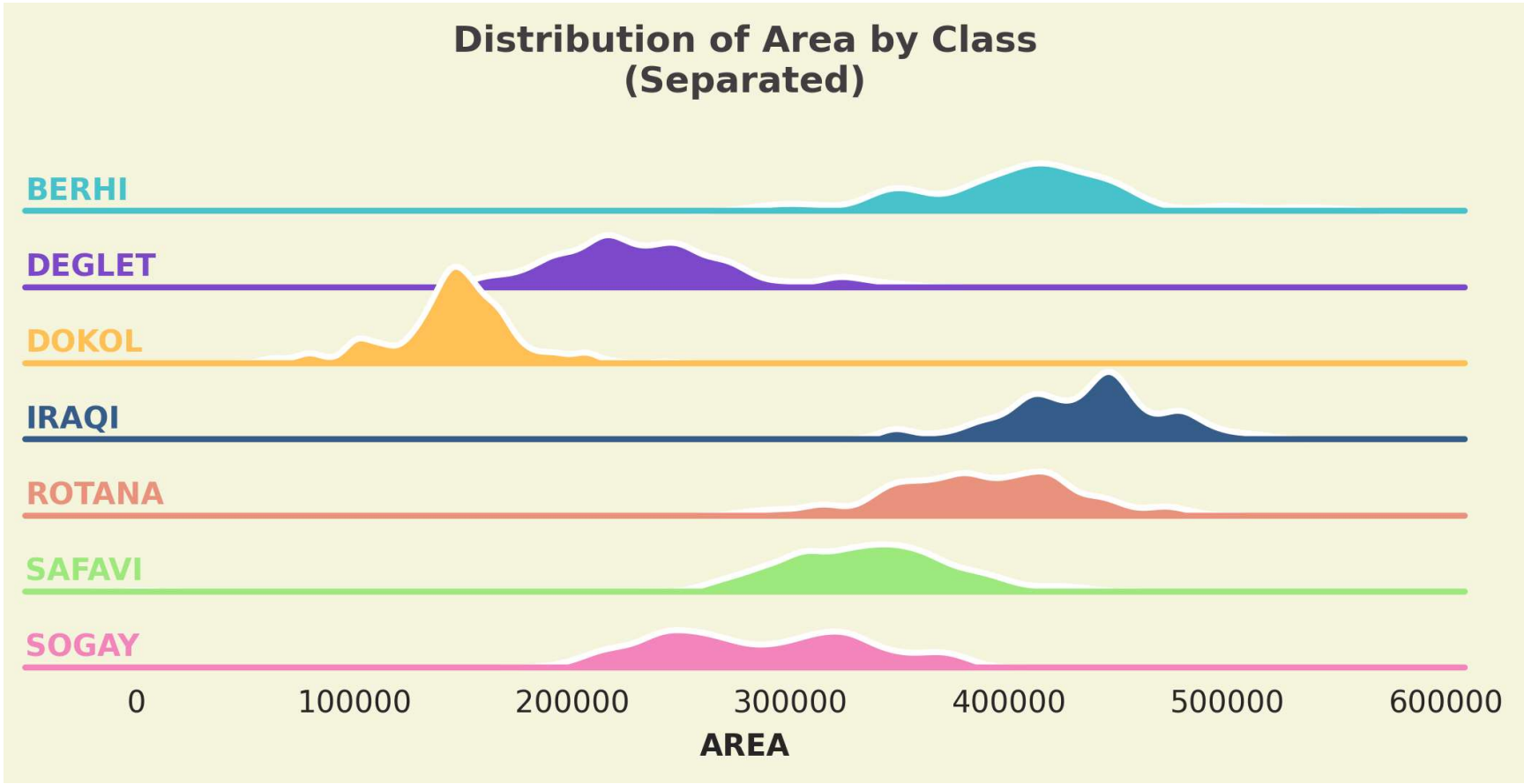


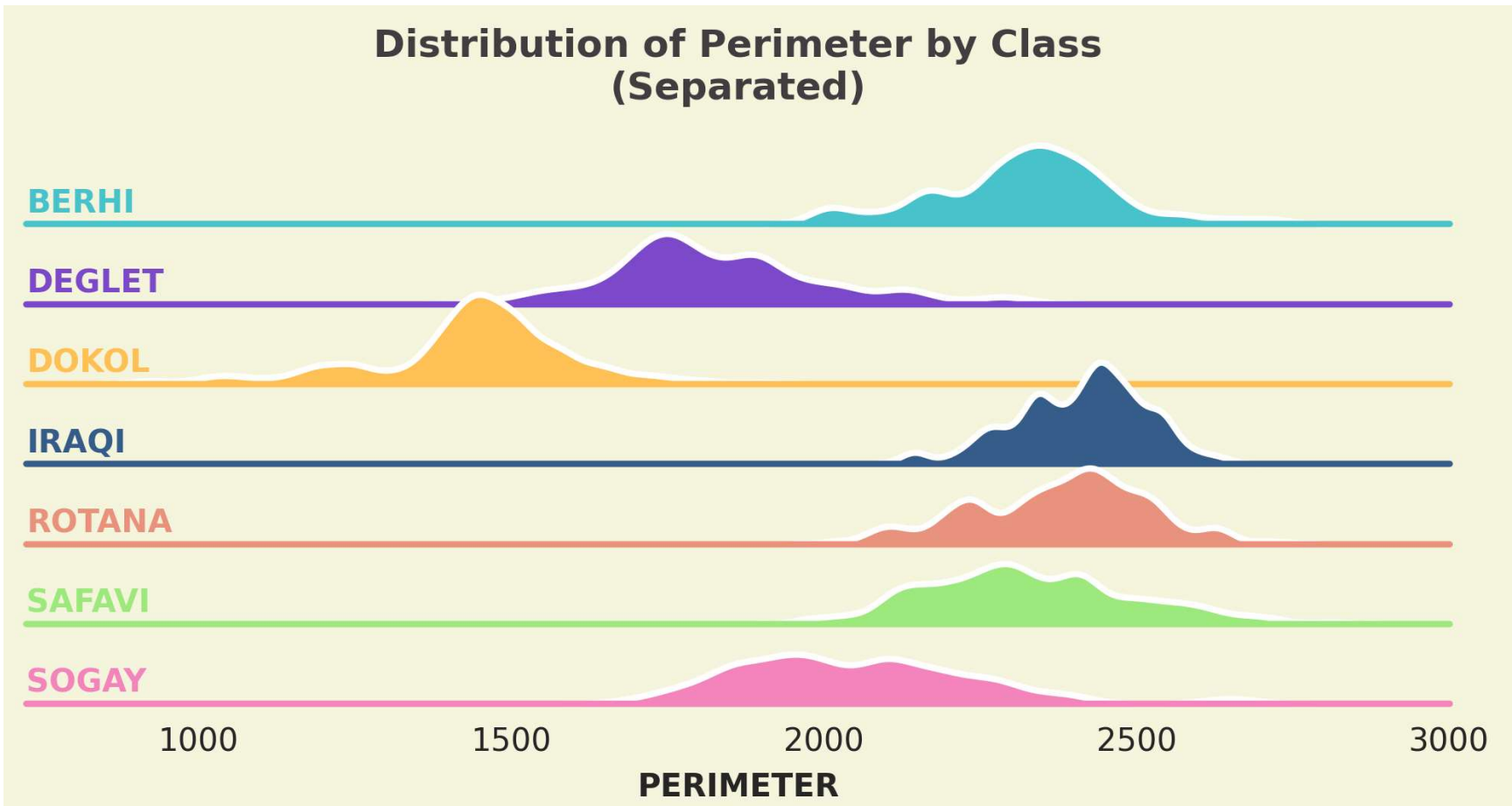


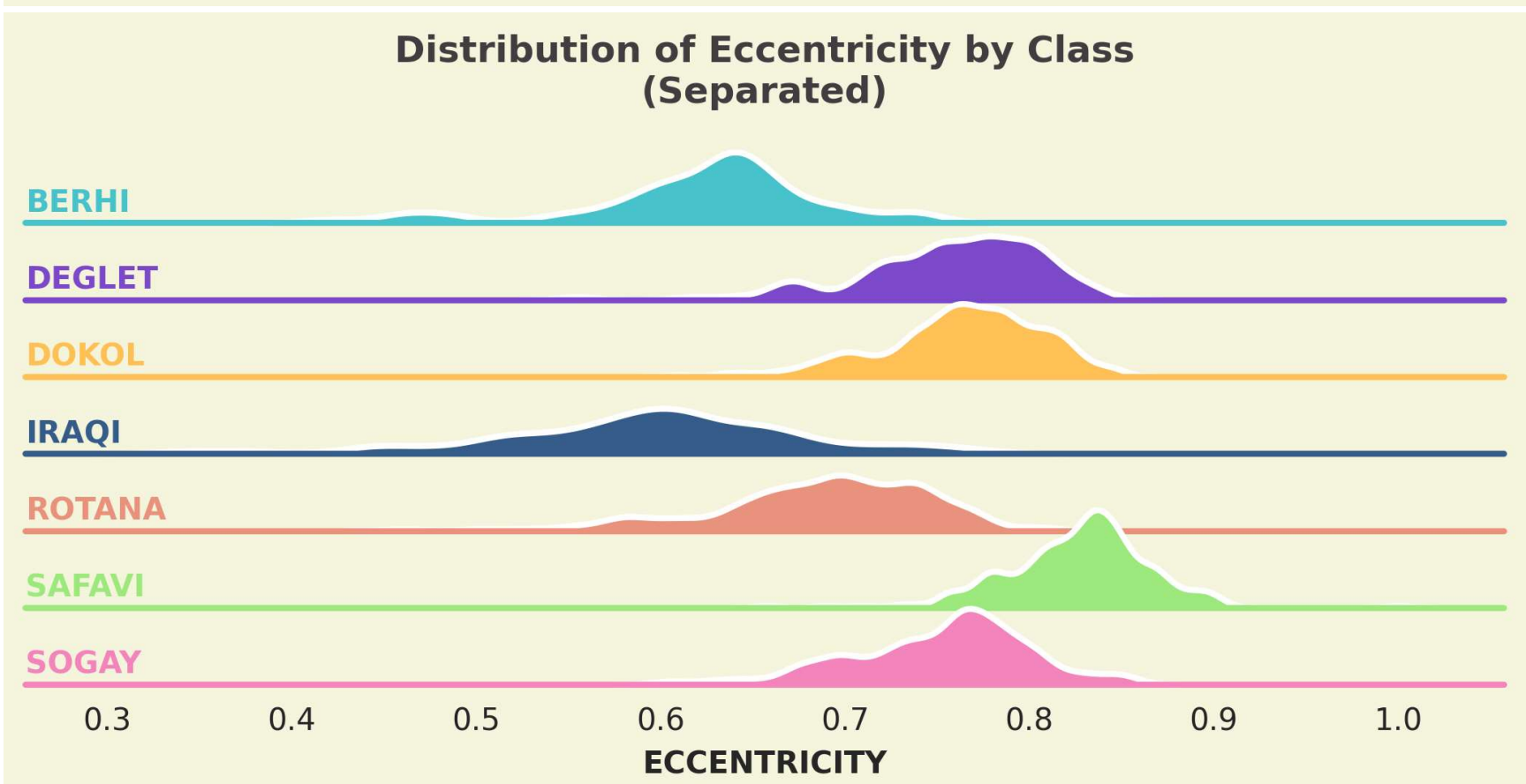


In [32]:

```
for i in dis_feat :
    with sns.axes_style("white", rc={"axes.facecolor": (0, 0, 0, 0), 'figure.facecolor':'beige'}):

        g = sns.FacetGrid(df, row="Class", hue="Class", aspect=15, height=.5, palette=colors)

        g.map(sns.kdeplot, i,
              bw_adjust=.5, clip_on=False,
              fill=True, alpha=1, linewidth=1.5)
        g.map(sns.kdeplot, i, clip_on=False, color="w", lw=2, bw_adjust=.5)

        g.map(plt.axhline, y=0, linewidth=2, linestyle="-", color=None, clip_on=False)
        def label(x, color, label):
            ax = plt.gca()
            ax.text(0, .2, label, fontweight="bold", color=color,
                    ha="left", va="center", transform=ax.transAxes)

        g.map(label, "Class")
        g.fig.subplots_adjust(hspace=-.25)
        g.set_titles("")
        g.set(yticks=[], xlabel="", ylabel="")
        plt.xlabel(i, fontweight="bold")
        g.despine(bottom=True, left=True)
        plt.title("Distribution of " + i.capitalize() + " by Class\n(Separated)", y=5.5, size=12, fontweight="bold", co
```



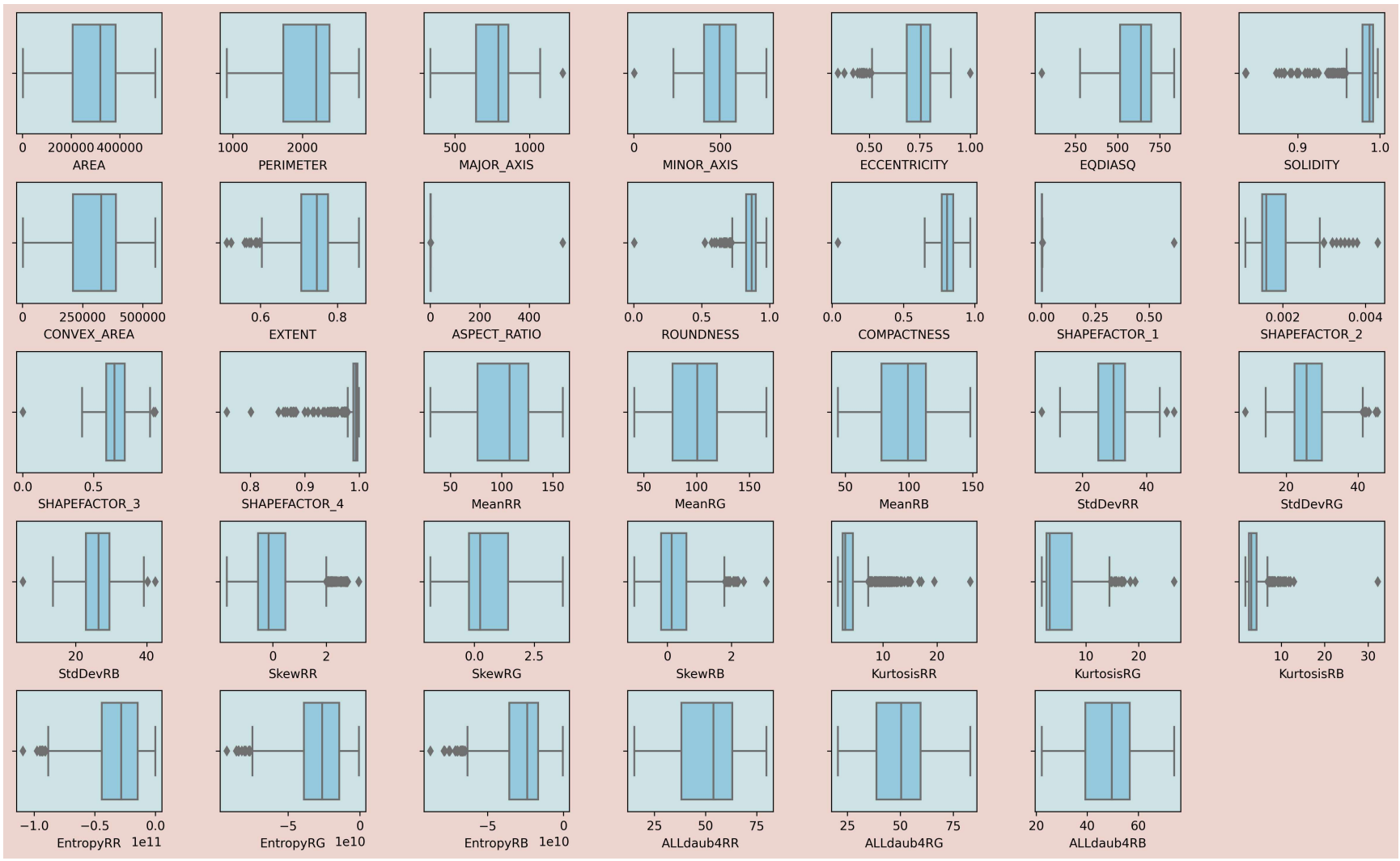




Outliers

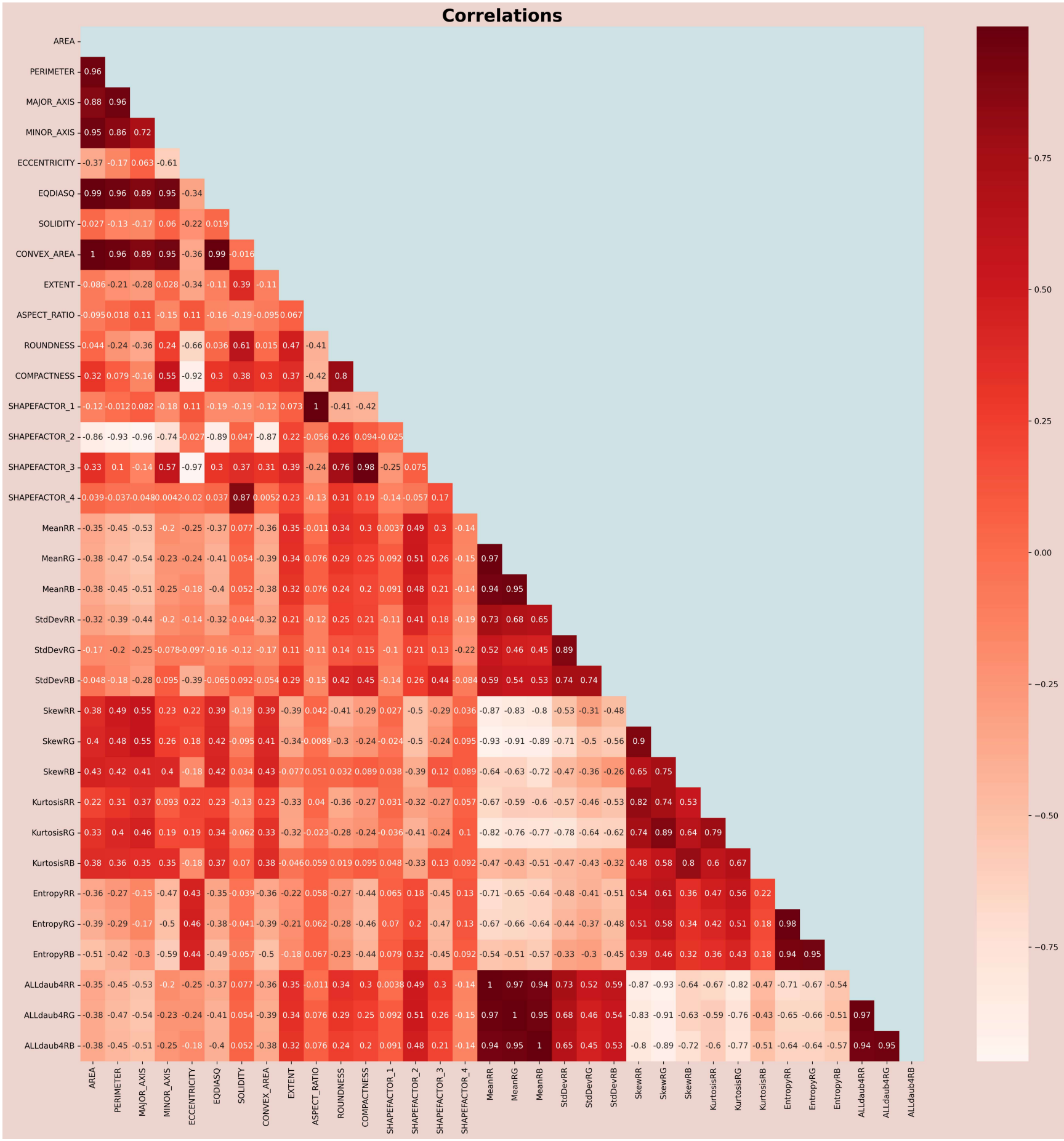
In [33]:
```
list_col = list(df.columns)
list_col.remove("Class")

fig = plt.figure(figsize=(20,12), facecolor='#EED4CF')
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for num, column_name in enumerate(list_col):
```

```
    ax = fig.add_subplot(5, 7, num +1)
    ax = sns.boxplot(x=df[column_name], color='skyblue')
```



Correlation Matrix

In [34]:
```
plt.figure(figsize=(20,20), facecolor='#EED4CF')
df_corr = df.corr()
mask = np.triu(np.ones_like(df_corr, dtype=bool))
sns.heatmap(df_corr, mask=mask, annot=True, cmap="Reds")
plt.title("Correlations", size=22, fontweight="bold")
plt.tight_layout()
```



Data Preprocessing

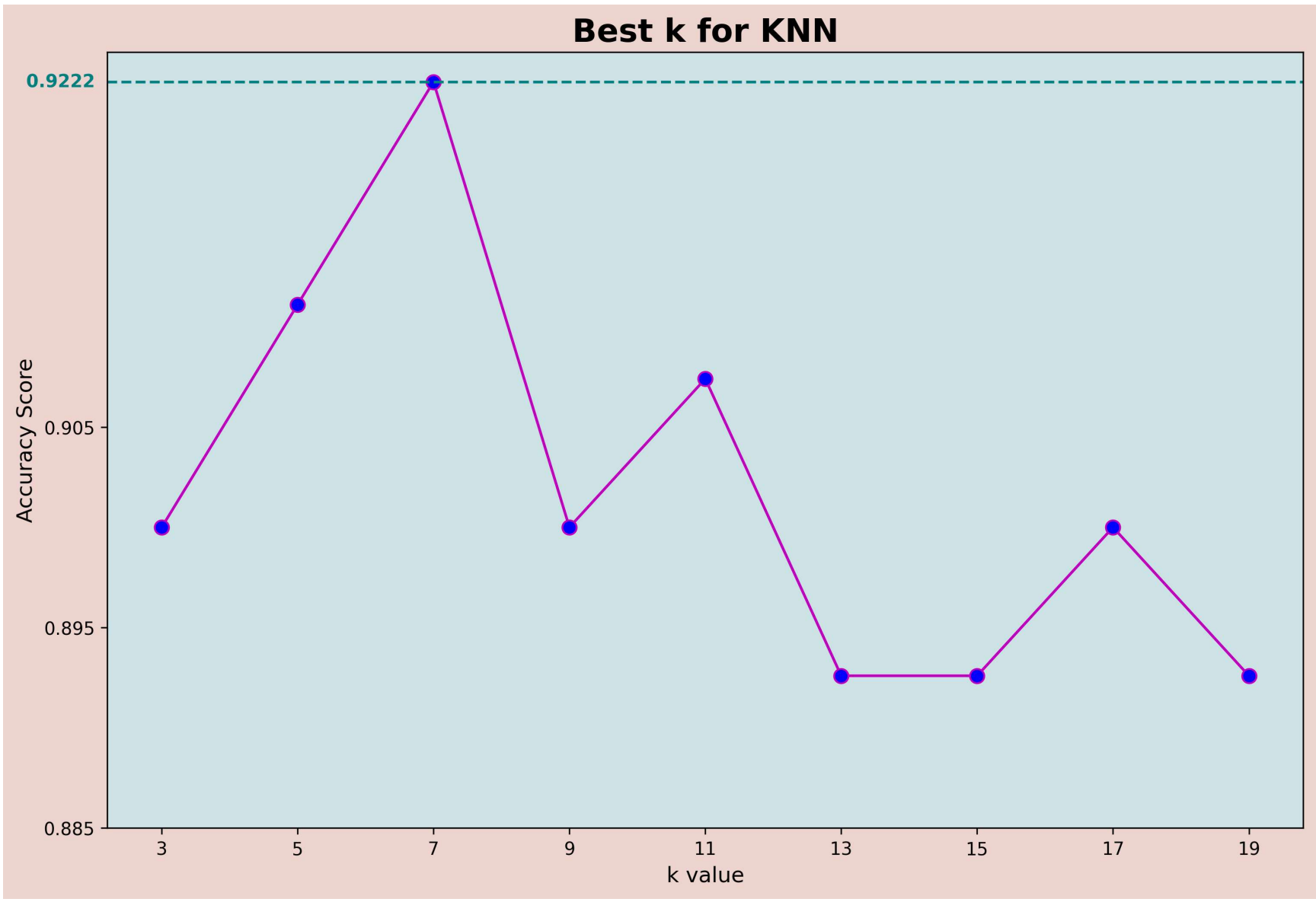
In [35]:
```
X = df.drop("Class", axis = 1)
y = df["Class"]

le = LabelEncoder() # encoding
y = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
                                                    stratify=y, random_state=42) # splitting

scaler = StandardScaler() # scaling
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train = pd.DataFrame(X_train_scaled)
X_test = pd.DataFrame(X_test_scaled)
```

Model Building

In [36]:
```
neighbors_list = list(range(3,20,2))
knn_score_list = []

for number in neighbors_list:
    knn = KNeighborsClassifier(n_neighbors=number)
    knn.fit(X_train, y_train)
    y_predict_knn = knn.predict(X_test)
    knn_score_list.append(accuracy_score(y_test, y_predict_knn))

fig, ax = plt.subplots(1,1, figsize=(12,8), facecolor="#EED4CF")
plt.plot(neighbors_list, knn_score_list, marker="o", markerfacecolor="blue", markersize=8, color="m")
plt.xticks(np.arange(3, 20, 2))
plt.yticks(np.arange(0.885, 0.910, 0.010))
plt.xlabel("k value", size=12)
plt.ylabel("Accuracy Score", size=12)
ax.axhline(y = max(knn_score_list) , linewidth = 1.5, color = "teal", linestyle="dashed")
```

```
trans = transforms.blended_transform_factory(
    ax.get_yticklabels()[0].get_transform(), ax.transData)
ax.text(0, max(knn_score_list), "{:.4f}".format(max(knn_score_list)), color="teal", transform=trans,
        ha="right", va="center", fontweight="bold")
plt.title("Best k for KNN", size=18, fontweight="bold")
```

Out[36]: Text(0.5, 1.0, 'Best k for KNN')



In [37]:
```
class models:

    def __init__(self, X_train, X_test, y_train, y_test):

        self.score_dict = {}

        self.X_train = X_train
        self.X_test = X_test
        self.y_train = y_train
        self.y_test = y_test

        #Logistic Regression
        self.lr = LogisticRegression(C = 0.1)
        self.lr.fit(self.X_train, self.y_train)
        self.y_predict_lr = self.lr.predict(self.X_test)
        #DTC
        self.dtc = DecisionTreeClassifier(criterion = 'gini', min_samples_split = 10, random_state=42)
        self.dtc.fit(self.X_train, self.y_train)
        self.y_predict_dtc = self.dtc.predict(self.X_test)
        #RFC
        self.rfc = RandomForestClassifier(n_estimators=100, criterion='gini', random_state=10)
        self.rfc.fit(self.X_train, self.y_train)
        self.y_predict_rfc = self.rfc.predict(self.X_test)
        #SVM
        self.svm = SVC(probability=True, C = 1, kernel = 'linear')
        self.svm.fit(self.X_train, self.y_train)
        self.y_predict_svm = self.svm.predict(self.X_test)
        #KNN
        self.knn = KNeighborsClassifier(n_neighbors=7)
        self.knn.fit(self.X_train, self.y_train)
        self.y_predict_knn = knn.predict(self.X_test)
        #Naive Bayes
        self.nb = GaussianNB()
        self.nb.fit(self.X_train, self.y_train)
        self.y_predict_nb = self.nb.predict(self.X_test)

    def accuracy_scores(self):

        self.score_dict["LR"] = round(self.lr.score(self.X_test, self.y_test), 3)
        self.score_dict["DTC"] = round(self.dtc.score(self.X_test, self.y_test), 3)
        self.score_dict["RFC"] = round(self.rfc.score(self.X_test, self.y_test), 3)
        self.score_dict["SVM"] = round(self.svm.score(self.X_test, self.y_test), 3)
        self.score_dict["KNN"] = round(self.knn.score(self.X_test, self.y_test), 3)
        self.score_dict["NB"] = round(self.nb.score(self.X_test, self.y_test), 3)

        fig, ax = plt.subplots(2,1, figsize=(12,10), dpi=300, facecolor="#EED4CF")
        acc_scores = [list(self.score_dict.values())]
        collabel = tuple(self.score_dict.keys())
        index_max = list(self.score_dict.values()).index(max(list(self.score_dict.values())))
        ax[0].axis('tight')
        ax[0].axis('off')
```

```
        the_table = ax[0].table(cellText=acc_scores, colLabels=collabel, loc='center', cellLoc="center")
        ax[0].set_title("Acc. Comparison Table", y=0.6, color="purple", fontweight="bold")

        for i in range(0, len(list(self.score_dict.keys()))):
            the_table[(0, i)].set_facecolor("#EBCE5A")

        the_table[(1, index_max)].set_facecolor("#98DE6F")
        ax[1].plot(list(self.score_dict.keys()), list(self.score_dict.values()), color = "purple", marker="o")
        plt.title("Comparison of Models According to Acc. Scores", color="purple", fontweight="bold")
        plt.xlabel("Model", color="purple")
        plt.ylabel("Acc. Score", color="purple")
        plt.show()


    def con_matrix(self):

        cm_lr = confusion_matrix(self.y_test, self.y_predict_lr)
        cm_lr_df = pd.DataFrame(cm_lr, index=le.classes_, columns=le.classes_)
        plt.figure(figsize=(10,6), dpi=300, facecolor="#EED4CF")
        sns.heatmap(cm_lr_df, annot=True, cmap="Oranges", fmt=".1f")
        plt.title("LR Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("True")

        cm_dtc = confusion_matrix(self.y_test, self.y_predict_dtc)
        cm_dtc_df = pd.DataFrame(cm_dtc, index=le.classes_, columns=le.classes_)
        plt.figure(figsize=(10,6), dpi=300, facecolor="#EED4CF")
        sns.heatmap(cm_dtc_df, annot=True, cmap="Reds", fmt=".1f")
        plt.title("DTC Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("True")

        cm_rfc = confusion_matrix(self.y_test, self.y_predict_rfc)
        cm_rfc_df = pd.DataFrame(cm_rfc, index=le.classes_, columns=le.classes_)
        plt.figure(figsize=(10,6), dpi=300, facecolor="#EED4CF")
        sns.heatmap(cm_rfc_df, annot=True, cmap="Reds",fmt=".1f")
        plt.title("RFC Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("True")

        cm_svm = confusion_matrix(self.y_test, self.y_predict_svm)
        cm_svm_df = pd.DataFrame(cm_svm, index=le.classes_, columns=le.classes_)
        plt.figure(figsize=(10,6), dpi=300, facecolor="#EED4CF")
        sns.heatmap(cm_svm_df, annot=True, cmap="Reds",fmt=".1f")
        plt.title("SVM Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("True")

        cm_knn = confusion_matrix(self.y_test, self.y_predict_knn)
        cm_knn_df = pd.DataFrame(cm_knn, index=le.classes_, columns=le.classes_)
        plt.figure(figsize=(10,6), dpi=300, facecolor="#EED4CF")
        sns.heatmap(cm_knn_df, annot=True, cmap="Reds", fmt=".1f")
        plt.title("KNN Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("True")

        cm_nb = confusion_matrix(self.y_test, self.y_predict_nb)
        cm_nb_df = pd.DataFrame(cm_nb, index=le.classes_, columns=le.classes_)
        plt.figure(figsize=(10,6), dpi=300, facecolor="#EED4CF")
        sns.heatmap(cm_nb_df, annot=True, cmap="Reds", fmt=".1f")
        plt.title("NB Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("True")

    def classification_reports(self):

        red_start = '\033[91m'
        red_end = '\033[90m'
        blue_start = '\033[94m'
        blue_end = '\033[90m'
        bold_start = "\033[1m"
        bold_end = "\033[0;0m"

        print(red_start + bold_start + "LR\n" + bold_end + red_end)
        print(blue_start + classification_report(self.y_test, self.y_predict_lr, target_names=le.classes_) + bold_end)
        print(red_start + bold_start +  "*"* 53 +  "\n"  + bold_end + red_end)
        print(red_start + bold_start + "DTC\n" + bold_end + red_end)
        print(blue_start + classification_report(self.y_test, self.y_predict_dtc, target_names=le.classes_) + blue_end)
        print(red_start + bold_start + "*"* 53 + bold_end + red_end + "\n" )
        print(red_start + bold_start + "RFC\n" + bold_end + red_end)
        print(blue_start + classification_report(self.y_test, self.y_predict_rfc, target_names=le.classes_) + bold_end)
        print(red_start + bold_start + "*"* 53 + bold_end + red_end + "\n")
        print(red_start + bold_start + "SVM\n" + bold_end + red_end)
        print(blue_start + classification_report(self.y_test, self.y_predict_svm, target_names=le.classes_) + bold_end)
        print(red_start + bold_start + "*"* 53 + bold_end + red_end + "\n")
        print(red_start + bold_start + "KNN\n" + bold_end)
        print(blue_start + classification_report(self.y_test, self.y_predict_knn, target_names=le.classes_) + bold_end)
        print(red_start + bold_start + "*"* 53 + bold_end + red_end + "\n")
        print(red_start + bold_start + "NB\n" + bold_end)
        print(blue_start + classification_report(self.y_test, self.y_predict_nb, target_names=le.classes_) + bold_end)
        print(red_start + bold_start + "*"* 53 + bold_end + red_end + "\n")

    def cross_validations(self):

        cv_dict = {}
```

```
        cv = 10
        cv_dict["LR"] = round(cross_val_score(estimator=self.lr, X = self.X_train, y = self.y_train, cv=cv).mean(), 3)
        cv_dict["DTC"] = round(cross_val_score(estimator=self.dtc, X = self.X_train, y = self.y_train, cv=cv).mean(), 3
        cv_dict["RFC"] = round(cross_val_score(estimator=self.rfc, X = self.X_train, y = self.y_train, cv=cv).mean(), 3
        cv_dict["SVM"] = round(cross_val_score(estimator=self.svm, X = self.X_train, y = self.y_train, cv=cv).mean(), 3
        cv_dict["KNN"] = round(cross_val_score(estimator=self.knn, X = self.X_train, y = self.y_train, cv=cv).mean(), 3
        cv_dict["NB"] = round(cross_val_score(estimator=self.nb, X = self.X_train, y = self.y_train, cv=cv).mean(), 3)

        fig, ax = plt.subplots(2,1, figsize=(12,10), dpi=300, facecolor="#EED4CF")
        acc_scores = [list(cv_dict.values())]
        collabel = tuple(cv_dict.keys())
        index_max = list(cv_dict.values()).index(max(list(cv_dict.values())))
        ax[0].axis('tight')
        ax[0].axis('off')
        the_table = ax[0].table(cellText=acc_scores, colLabels=collabel, loc='center', cellLoc="center")
        ax[0].set_title("CV Comparison Table", y=0.6, color="purple", fontweight="bold")

        for i in range(0, len(list(cv_dict.keys()))):
            the_table[(0, i)].set_facecolor("#EDB2F6")

        color_list = ['lightgreen' if (x == max(list(cv_dict.values()))) else '#974576' for x in list(cv_dict.values())
        hatchs = ['O' if (x == max(list(cv_dict.values()))) else None for x in list(cv_dict.values())]


        the_table[(1, index_max)].set_facecolor("#98DE6F")
        ax[1].bar(list(cv_dict.keys()), list(cv_dict.values()), color = color_list, hatch=hatchs, edgecolor="black", lw
        ax[1].set_ylim([0.75, 0.95])
        ax[1].bar_label(ax[1].containers[0], fmt='%.3f', color="black", fontsize=12, fontweight="bold")
        plt.title("Comparison of Models According to CV Scores", color="purple", fontweight="bold")
        plt.xlabel("Model", color="purple")
        plt.ylabel("CV Score", color="purple")
        plt.show()
```

In [38]:
```
models = models(X_train, X_test, y_train, y_test)
```

Model Evaluation

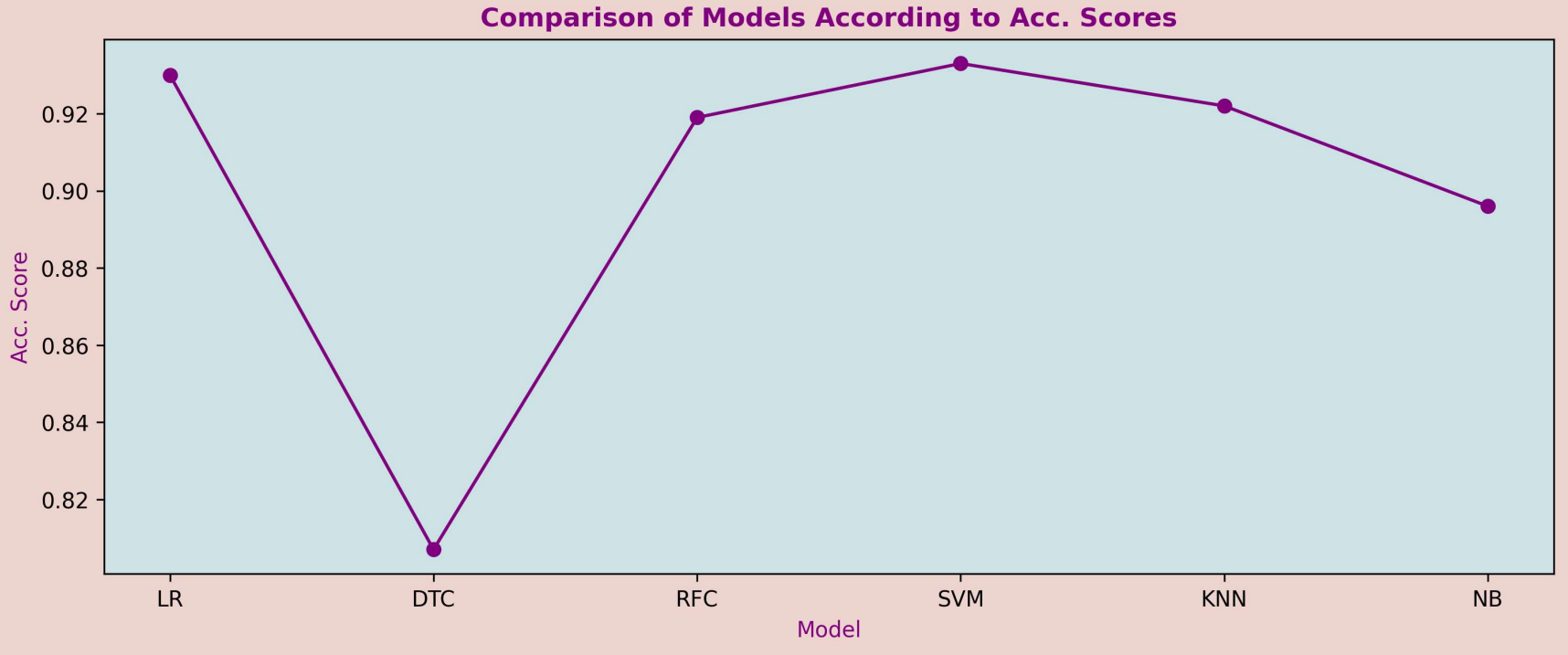
Accuracy Scores

In [39]:
```
models.accuracy_scores()
```

Acc. Comparison Table

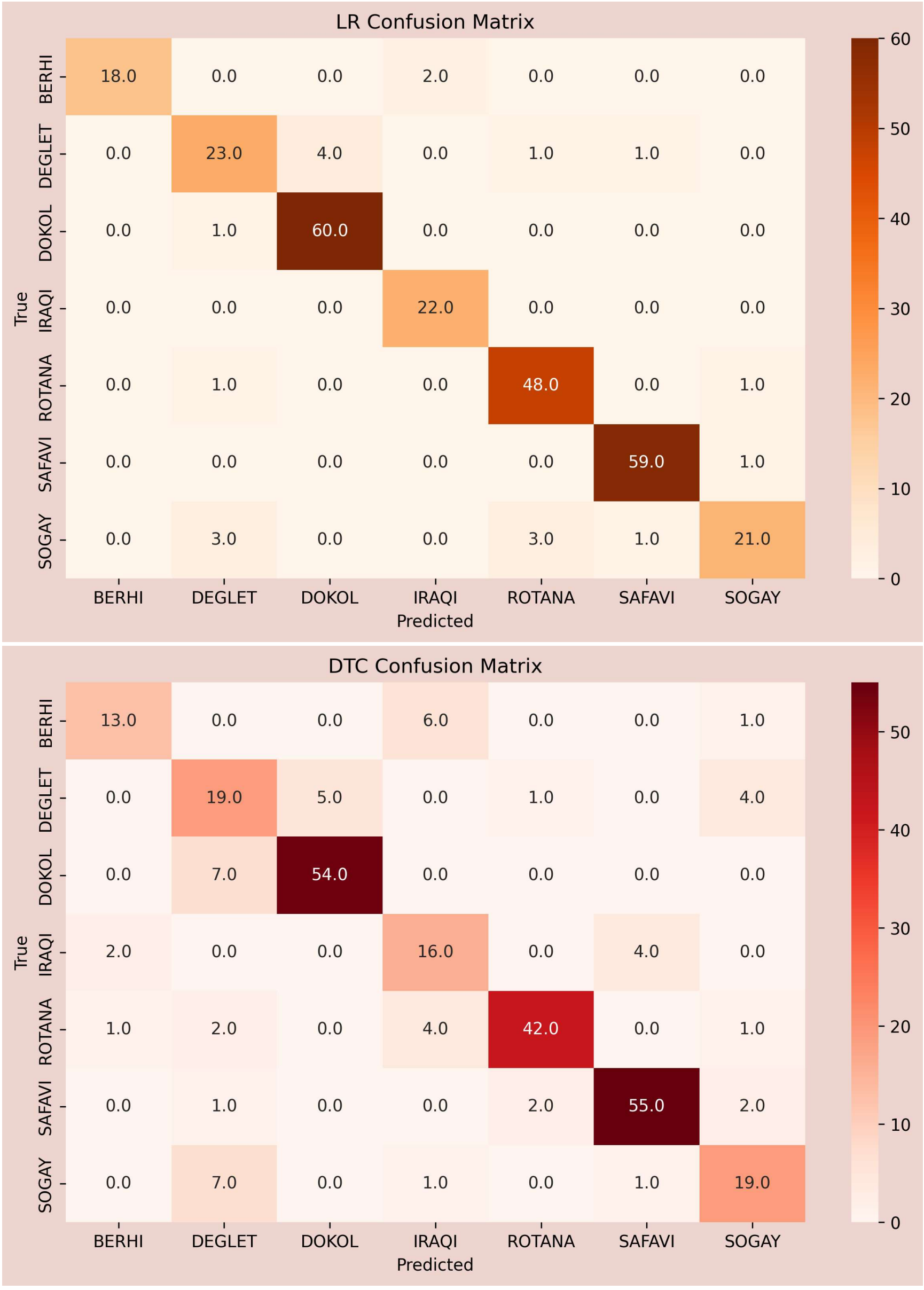
LR	DTC	RFC	SVM	KNN	NB
0.93	0.807	0.919	0.933	0.922	0.896

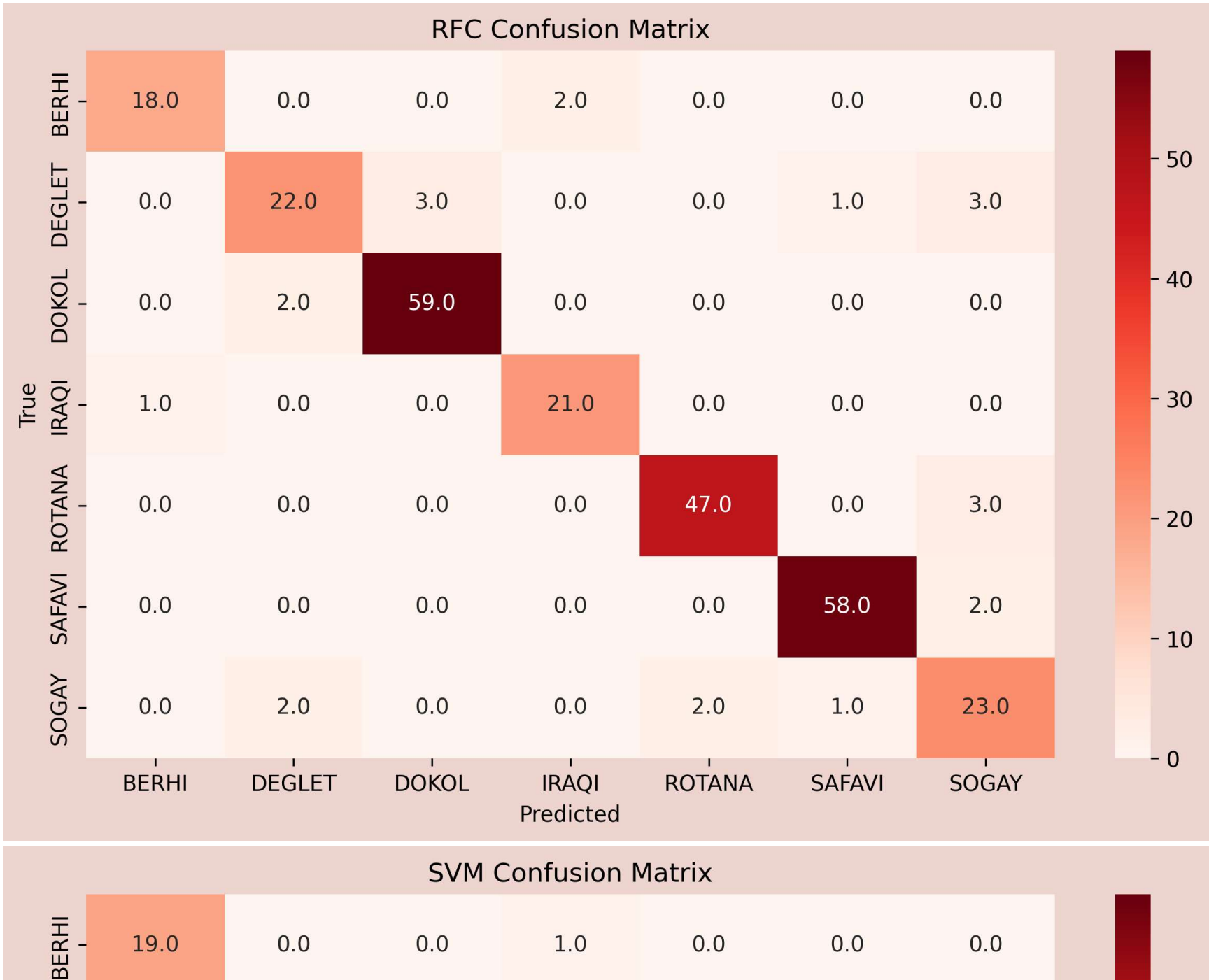
Comparison of Models According to Acc. Scores

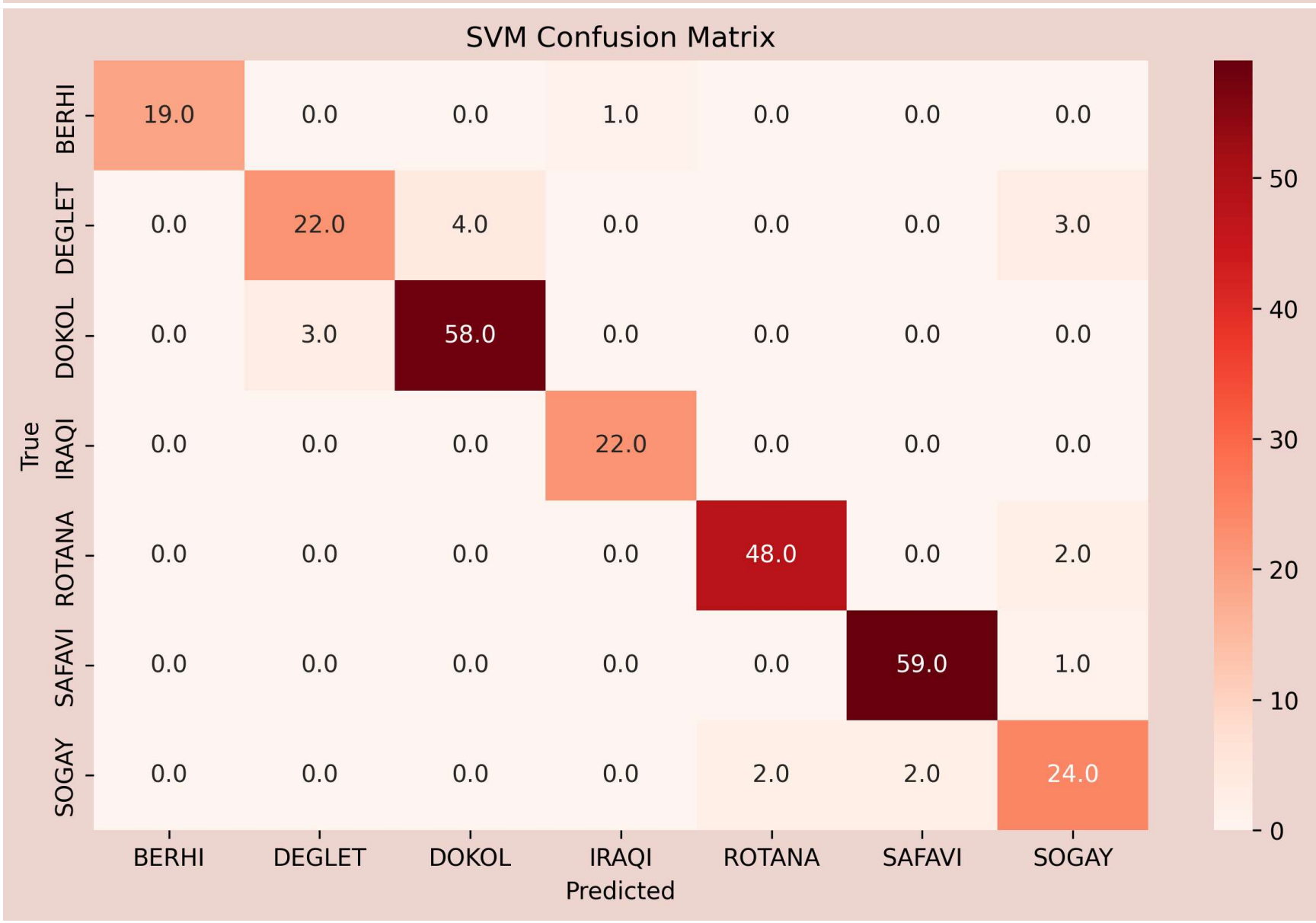


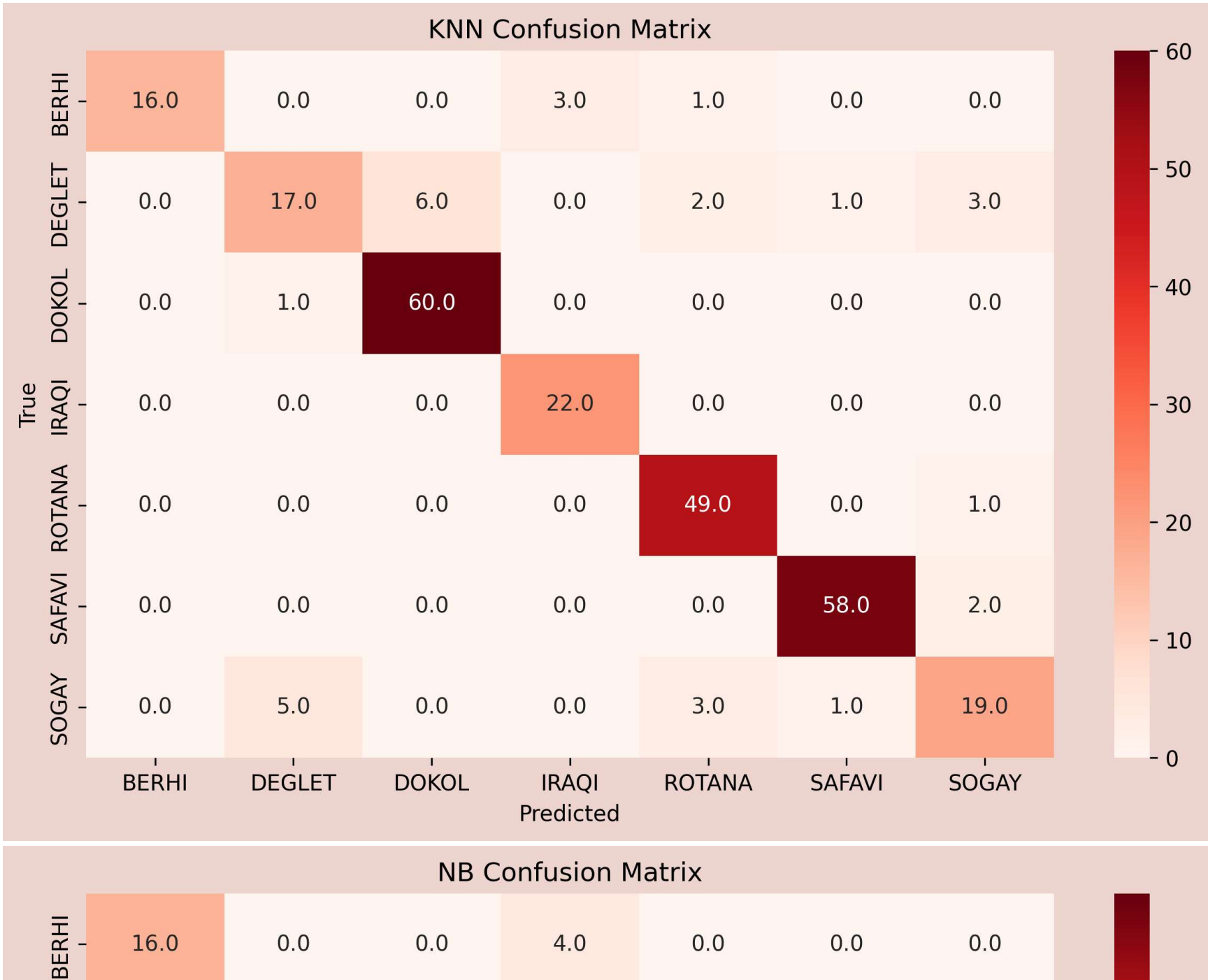
Confusion Matrix

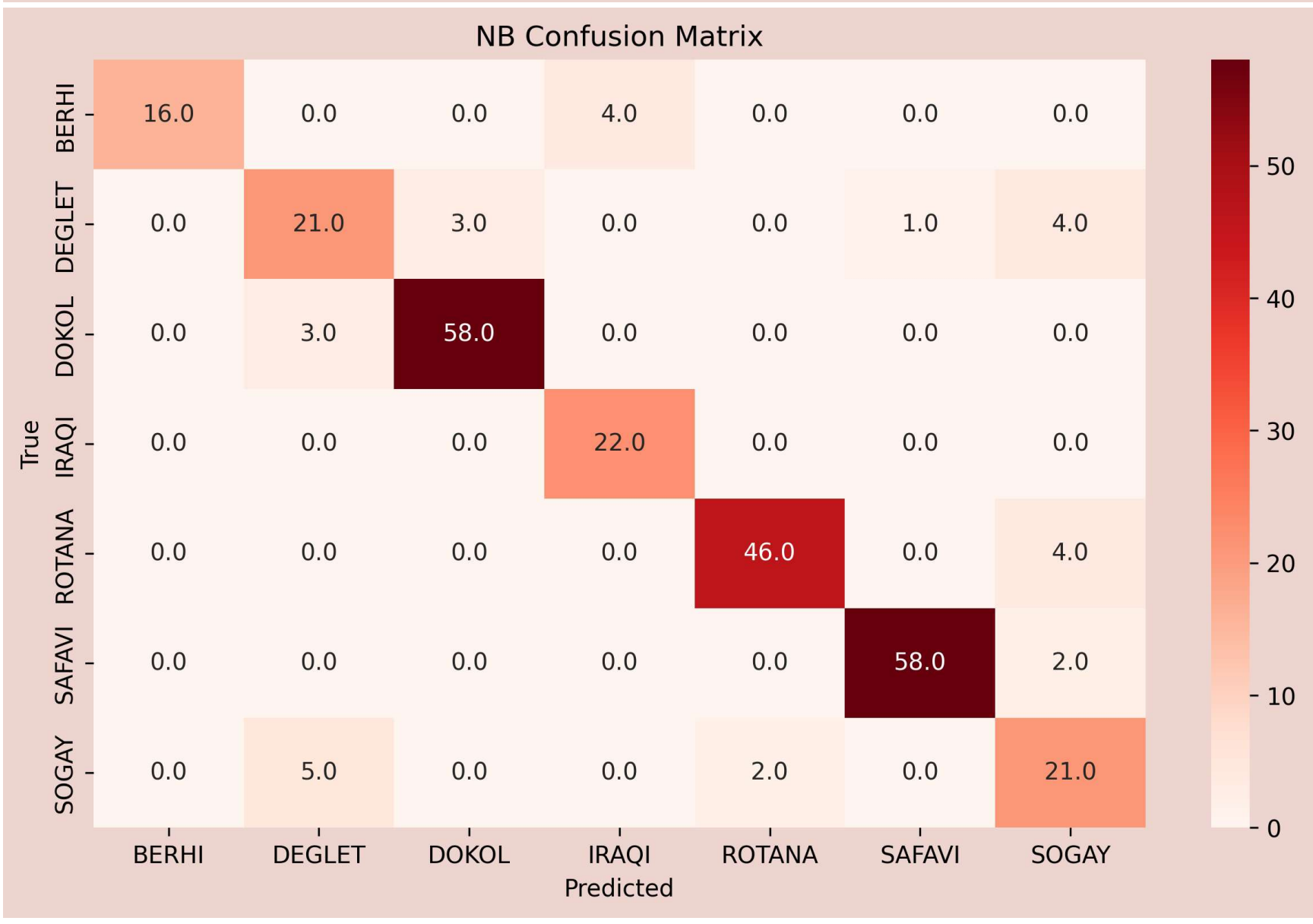
In [40]:
```
models.con_matrix()
```


LR Confusion Matrix
True
BERHI 18.0 0.0 0.0 2.0 0.0 0.0 0.0
DEGLET 0.0 23.0 4.0 0.0 1.0 1.0 0.0
DOKOL 0.0 1.0 60.0 0.0 0.0 0.0 0.0
IRAQI 0.0 0.0 0.0 22.0 0.0 0.0 0.0
ROTANA 0.0 1.0 0.0 0.0 48.0 0.0 1.0
SAFAVI 0.0 0.0 0.0 0.0 0.0 59.0 1.0
SOGAY 0.0 3.0 0.0 0.0 3.0 1.0 21.0
BERHI DEGLET DOKOL IRAQI ROTANA SAFAVI SOGAY
Predicted
60 50 40 30 20 10 0
DTC Confusion Matrix
True
BERHI 13.0 0.0 0.0 6.0 0.0 0.0 1.0
DEGLET 0.0 19.0 5.0 0.0 1.0 0.0 4.0
DOKOL 0.0 7.0 54.0 0.0 0.0 0.0 0.0
IRAQI 2.0 0.0 0.0 16.0 0.0 4.0 0.0
ROTANA 1.0 2.0 0.0 4.0 42.0 0.0 1.0
SAFAVI 0.0 1.0 0.0 0.0 2.0 55.0 2.0
SOGAY 0.0 7.0 0.0 1.0 0.0 1.0 19.0
BERHI DEGLET DOKOL IRAQI ROTANA SAFAVI SOGAY
Predicted
50 40 30 20 10 0


RFC Confusion Matrix
True
BERHI
DEGLET
DOKOL
IRAQI
ROTANA
SAFAVI
SOGAY
18.0 0.0 0.0 2.0 0.0 0.0 0.0
0.0 22.0 3.0 0.0 0.0 1.0 3.0
0.0 2.0 59.0 0.0 0.0 0.0 0.0
1.0 0.0 0.0 21.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 47.0 0.0 3.0
0.0 0.0 0.0 0.0 0.0 58.0 2.0
0.0 2.0 0.0 0.0 2.0 1.0 23.0
BERHI
DEGLET
DOKOL
IRAQI
ROTANA
SAFAVI
SOGAY
Predicted
50
40
30
20
10
0


SVM Confusion Matrix
True
BERHI
DEGLET
DOKOL
IRAQI
ROTANA
SAFAVI
SOGAY
19.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 22.0 4.0 0.0 0.0 0.0 3.0
0.0 3.0 58.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 22.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 48.0 0.0 2.0
0.0 0.0 0.0 0.0 0.0 59.0 1.0
0.0 0.0 0.0 0.0 2.0 2.0 24.0
BERHI
DEGLET
DOKOL
IRAQI
ROTANA
SAFAVI
SOGAY
Predicted
50
40
30
20
10
0





Classification Reports

In [41]:
```
models.classification_reports()
```

LR

	precision	recall	f1-score	support
BERHI	1.00	0.90	0.95	20
DEGLET	0.82	0.79	0.81	29
DOKOL	0.94	0.98	0.96	61
IRAQI	0.92	1.00	0.96	22
ROTANA	0.92	0.96	0.94	50
SAFAVI	0.97	0.98	0.98	60
SOGAY	0.91	0.75	0.82	28
accuracy			0.93	270
macro avg	0.93	0.91	0.92	270
weighted avg	0.93	0.93	0.93	270

****

DTC

	precision	recall	f1-score	support
BERHI	0.81	0.65	0.72	20
DEGLET	0.53	0.66	0.58	29
DOKOL	0.92	0.89	0.90	61
IRAQI	0.59	0.73	0.65	22
ROTANA	0.93	0.84	0.88	50
SAFAVI	0.92	0.92	0.92	60
SOGAY	0.70	0.68	0.69	28
accuracy			0.81	270
macro avg	0.77	0.76	0.76	270
weighted avg	0.82	0.81	0.81	270

****

RFC

	precision	recall	f1-score	support
BERHI	0.95	0.90	0.92	20
DEGLET	0.85	0.76	0.80	29
DOKOL	0.95	0.97	0.96	61
IRAQI	0.91	0.95	0.93	22
ROTANA	0.96	0.94	0.95	50
SAFAVI	0.97	0.97	0.97	60
SOGAY	0.74	0.82	0.78	28
accuracy			0.92	270
macro avg	0.90	0.90	0.90	270
weighted avg	0.92	0.92	0.92	270

****

SVM

	precision	recall	f1-score	support
BERHI	1.00	0.95	0.97	20
DEGLET	0.88	0.76	0.81	29
DOKOL	0.94	0.95	0.94	61
IRAQI	0.96	1.00	0.98	22
ROTANA	0.96	0.96	0.96	50
SAFAVI	0.97	0.98	0.98	60
SOGAY	0.80	0.86	0.83	28
accuracy			0.93	270
macro avg	0.93	0.92	0.92	270
weighted avg	0.93	0.93	0.93	270

****

KNN

	precision	recall	f1-score	support
BERHI	1.00	0.80	0.89	20
DEGLET	0.74	0.59	0.65	29
DOKOL	0.91	0.98	0.94	61
IRAQI	0.88	1.00	0.94	22
ROTANA	0.89	0.98	0.93	50
SAFAVI	0.97	0.97	0.97	60
SOGAY	0.76	0.68	0.72	28
accuracy			0.89	270
macro avg	0.88	0.86	0.86	270
weighted avg	0.89	0.89	0.89	270

****

NB

	precision	recall	f1-score	support
BERHI	1.00	0.80	0.89	20

```
      DEGLET       0.72      0.72      0.72        29
       DOKOL       0.95      0.95      0.95        61
       IRAQI       0.85      1.00      0.92        22
      ROTANA       0.96      0.92      0.94        50
      SAFAVI       0.98      0.97      0.97        60
       SOGAY       0.68      0.75      0.71        28

    accuracy                           0.90       270
   macro avg       0.88      0.87      0.87       270
weighted avg       0.90      0.90      0.90       270

**************************************************
```

CV Scores

In [42]:
```
models.cross_validations()
```

CV Comparison Table

LR	DTC	RFC	SVM	KNN	NB
0.898	0.815	0.885	0.922	0.874	0.881