In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.style
import seaborn as sns
import nltk #Natural Language Toolkit
#To ignore warnings
import warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
```

In [10]:
```
df = pd.read_csv('spam.csv',encoding='latin-1')
df.head()
```

Out[10]:

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [11]:
```
df=df.rename(columns={"Category":"Category","Message":"Text"})

df.head()
```

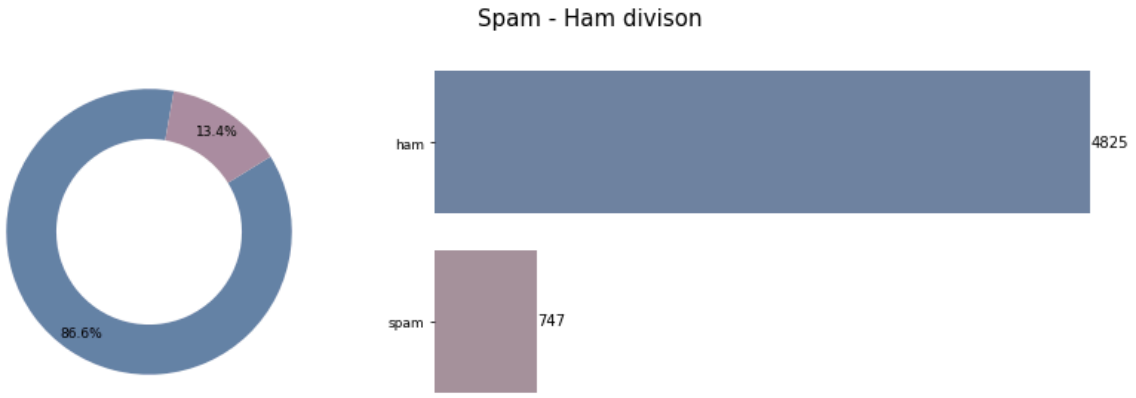
Out[11]:

	Category	Text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [12]:
```
print(" Total number of rows in the dataset are", len(df))
```

```
 Total number of rows in the dataset are 5572
```

In [28]:
```
plt.rcParams['figure.facecolor'] = 'white'
plt.rcParams['axes.facecolor'] = 'white'
fig, ax = plt.subplots(1, 2, figsize=(15, 4))
ax = ax.flatten()
value_counts = df['Category'].value_counts()
labels = value_counts.index.tolist()
colors =["#6782a8", "#ab90a0" ]
# Donut Chart
wedges, texts, autotexts = ax[0].pie(
    value_counts, autopct='%1.1f%%',textprops={'size': 9, 'color': 'black','f
    wedgeprops=dict(width=0.35),  startangle=80,   pctdistance=0.85  )
centre_circle = plt.Circle((0, 0), 0.6, fc='white')
ax[0].add_artist(centre_circle)
sns.countplot(data=df, y=df['Category'], ax=ax[1], palette=colors, order=labe
for i, v in enumerate(value_counts):
    ax[1].text(v + 1, i, str(v), color='black',fontsize=10, va='center')
sns.despine(left=True, bottom=True)
plt.yticks(fontsize=9,color='black')
ax[1].set_ylabel(None)
plt.xlabel("")
plt.xticks([])
fig.suptitle('Spam - Ham divison', fontsize=15)
plt.tight_layout(rect=[0, 0, 0.85, 1])
plt.show()
```



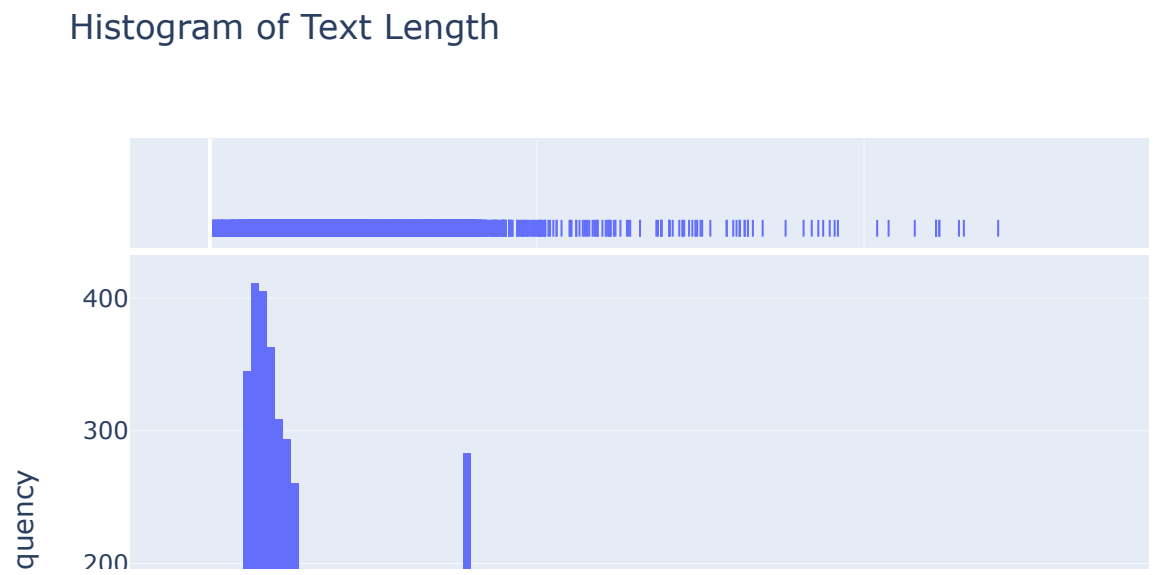
In [29]:
```
df.describe()
```

Out[29]:

	Category	Text
count	5572	5572
unique	2	5157
top	ham	Sorry, I'll call later
freq	4825	30

In [30]:
```
df['Length']=df['Text'].apply(len)
display(df.head())

#distribution of the data
import plotly.express as px
fig = px.histogram(df, x='Length', marginal='rug',
                   title='Histogram of Text Length')
fig.update_layout(
    xaxis_title='Length',
    yaxis_title='Frequency',
    showlegend=True)
```

	Category	Text	Length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61



In [31]:
```
import plotly.express as px
fig = px.histogram(df, x='Length', color='Category', marginal='rug',
                   title='Histogram of Text Length by Category')
fig.update_layout(
    xaxis_title='Length',
    yaxis_title='Frequency',
    showlegend=True)
```

In [32]:
```
df.loc[:,'Category']=df.Category.map({'ham':0, 'spam':1})
df['Category'] = df['Category'].astype(int)
df.head()
```

Out[32]:

	Category	Text	Length
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61

In [33]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

count = CountVectorizer()
text = count.fit_transform(df['Text'])
#Train & test split
x_train, x_test, y_train, y_test = train_test_split(text, df['Category'], tes
text
```

Out[33]:
```
<5572x8745 sparse matrix of type '<class 'numpy.int64'>'
        with 74225 stored elements in Compressed Sparse Row format>
```

In [34]:
```
display('X-Train :', x_train.shape)
display('X-Test :',x_test.shape)
display('Y-Train :',y_train.shape)
display('X-Test :',y_test.shape)
```

```
'X-Train :'

(3900, 8745)

'X-Test :'

(1672, 8745)

'Y-Train :'

(3900,)

'X-Test :'

(1672,)
```

In [46]:
```
from sklearn.naive_bayes import MultinomialNB

multinomial_nb_model = MultinomialNB()
multinomial_nb_model.fit(x_train, y_train)  # Train the model

prediction = multinomial_nb_model.predict(x_test)

print("Multinomial NB")
print("Accuracy score: {}". format(accuracy_score(y_test, prediction)) )
print("Precision score: {}". format(precision_score(y_test, prediction)) )
print("Recall score: {}". format(recall_score(y_test, prediction)))
print("F1 score: {}". format(f1_score(y_test, prediction)))
```

```
Multinomial NB
Accuracy score: 0.9778708133971292
Precision score: 0.9155555555555556
Recall score: 0.9196428571428571
F1 score: 0.9175946547884187
```

In [41]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
svm_model = SVC()
svm_model.fit(x_train, y_train)
prediction = svm_model.predict(x_test)
print("Support Vector Machine (SVM)")
print("Accuracy score: {}".format(accuracy_score(y_test, prediction)))
print("Precision score: {}".format(precision_score(y_test, prediction)))
print("Recall score: {}".format(recall_score(y_test, prediction)))
print("F1 score: {}".format(f1_score(y_test, prediction)))
```

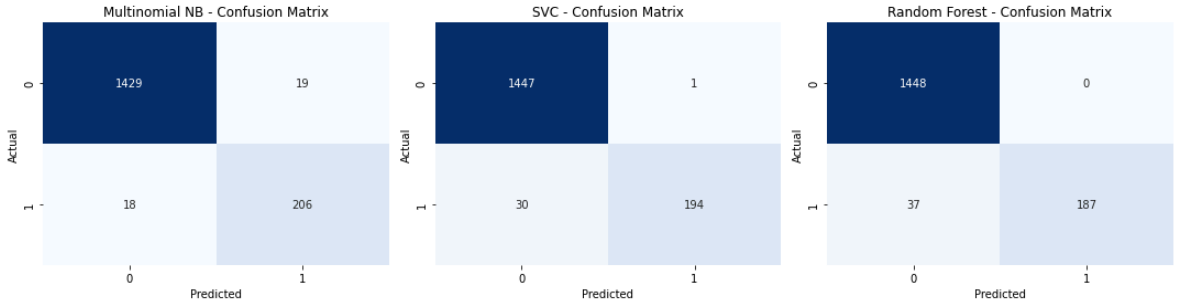
```
Support Vector Machine (SVM)
Accuracy score: 0.9814593301435407
Precision score: 0.9948717948717949
Recall score: 0.8660714285714286
F1 score: 0.9260143198090692
```

In [53]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

# Initialize the Random Forest classifier
random_forest_model = RandomForestClassifier()

# Train the model
random_forest_model.fit(x_train, y_train)

# Make predictions
prediction = random_forest_model.predict(x_test)

# Evaluate the model
print("Random Forest Classifier")
print("Accuracy score: {}".format(accuracy_score(y_test, prediction)))
print("Precision score: {}".format(precision_score(y_test, prediction)))
print("Recall score: {}".format(recall_score(y_test, prediction)))
print("F1 score: {}".format(f1_score(y_test, prediction)))
```

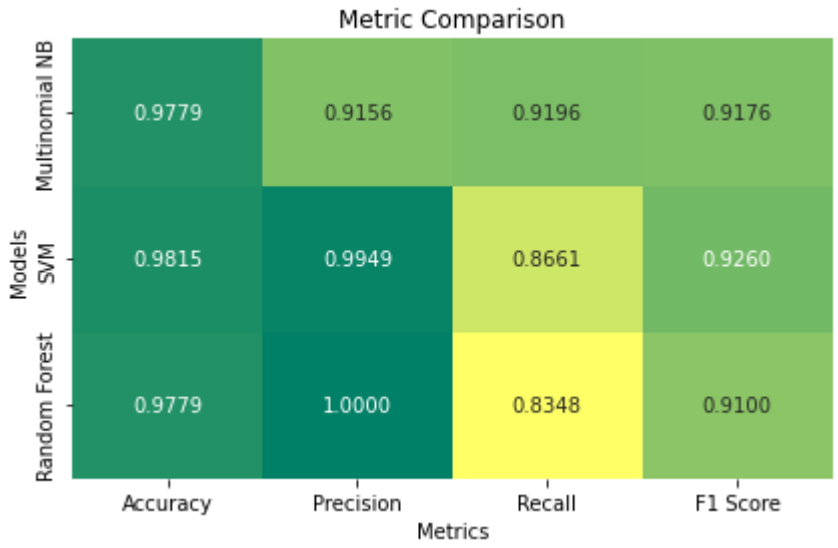
```
Random Forest Classifier
Accuracy score: 0.9778708133971292
Precision score: 1.0
Recall score: 0.8348214285714286
F1 score: 0.9099756690997567
```

In [55]:

```
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Instantiate SVC and Random Forest classifiers
svc_model = SVC()
rf_model = RandomForestClassifier()

# Define the models list
models = [("Multinomial NB", multinomial_nb_model), ("SVC", svc_model), ("Ran

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Iterate through models
for i, (model_name, model) in enumerate(models):
    # Train the model
    model.fit(x_train, y_train)

    # Make predictions
    prediction = model.predict(x_test)

    # Compute confusion matrix
    cm = confusion_matrix(y_test, prediction)

    # Plot confusion matrix
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, ax=axes[i]
    axes[i].set_title(f"{model_name} - Confusion Matrix")
    axes[i].set_xlabel("Predicted")
    axes[i].set_ylabel("Actual")

plt.tight_layout()
plt.show()
```



In [58]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define the metric data
metric_data = [
    [0.9778708133971292, 0.9155555555555556, 0.9196428571428571, 0.9175946547
    [0.9814593301435407, 0.9948717948717949, 0.8660714285714286, 0.9260143198
    [0.9778708133971292, 1.0, 0.8348214285714286, 0.9099756690997567]  # Rand
]

# Define the metric labels
metric_labels = ["Accuracy", "Precision", "Recall", "F1 Score"]

# Define the model names
model_names = ["Multinomial NB", "SVM", "Random Forest"]

# Create the heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(metric_data, annot=True, fmt=".4f", cbar=False, cmap="summer_r", x
plt.title("Metric Comparison")
plt.xlabel("Metrics")
plt.ylabel("Models")
plt.tight_layout()
plt.show()
```



In [ ]: