Telco Customer Churn Predictions

CS 513-A
Knowledge Discovery & Data Mining
Master of Science in Computer Science
Stevens Institute of Technology

Project Group No. 19

Thanapoom Phatthanaphan
20011296
tphattha@stevens.edu

Shiva Rama Krishna Mandadapu
20016347
smandad2@stevens.edu

Hantao Guo
10433245
hguo15@stevens.edu

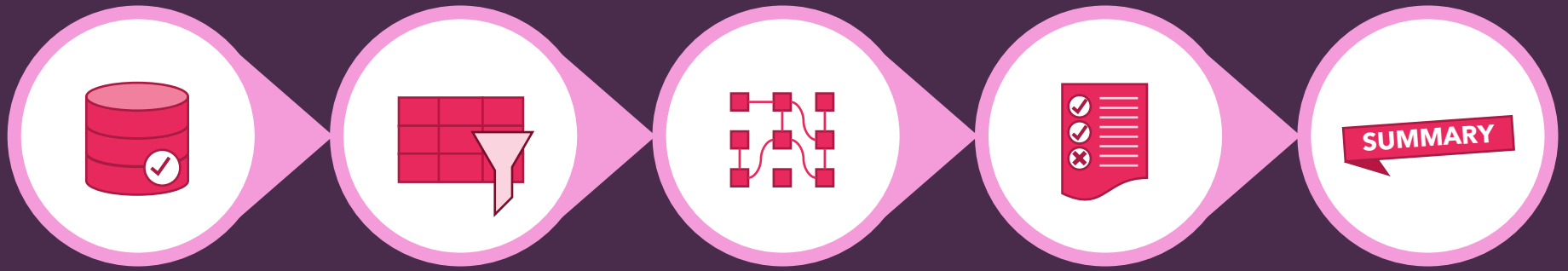
OVERVIEW

❖ Problem Statement

Develop a churn prediction model for a telecommunication company using historical customer data **to identify customers who might leave their services**.

Source of Dataset: https://www.kaggle.com/datasets/blastchar/telco-customer-churn
The dataset contains 7043 customer service data of a telecommunication company

❖ Presentation Flow



Data Understanding | **Data Preparation** | **Modeling** | **Evaluation** | **Summary**

❖ Input features (19 variables)

No.	Features	
1	Gender	Male / Female
2	Senior Citizen	1 (Yes) / 0 (No)
3	Partner	Yes / No
4	Dependents	Yes / No
5	Tenure	#months in service
6	Phone Service	Yes / No
7	Multiple Lines Service	Yes / No / No phone service
8	Internet Service	DSL / Fiber optic / No
9	Online Security	Yes / No / No internet service
10	Online Backup	Yes / No / No internet service

No.	Features	
11	Device Protection	Yes / No / No internet service
12	Tech SUpport	Yes / No / No internet service
13	Streaming TV	Yes / No / No internet service
14	Streaming Movies	Yes / No / No internet service
15	Contract	Month-to-month / One-year / Two-year
16	Paperless Billing	Yes / No
17	Payment Method	Electronic check / Mailed check / Bank transfer (automatic) / Credit card (automatic)
18	Monthly Charges	Amount charges by monthly
19	Total Charges	Total amount charges

❖ Target variable

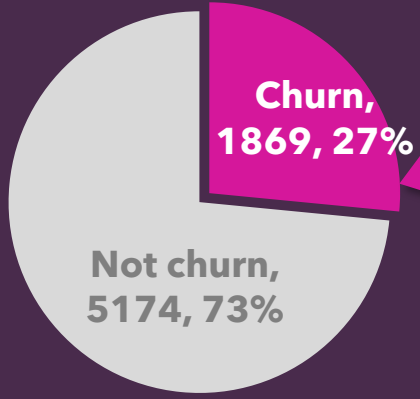
Churn

- **Yes** – A customer **churns** a telecommunications company's service
- **No** – A customer **do not churn** a telecommunications company's service

❖ Concerned features review

7043
Customers

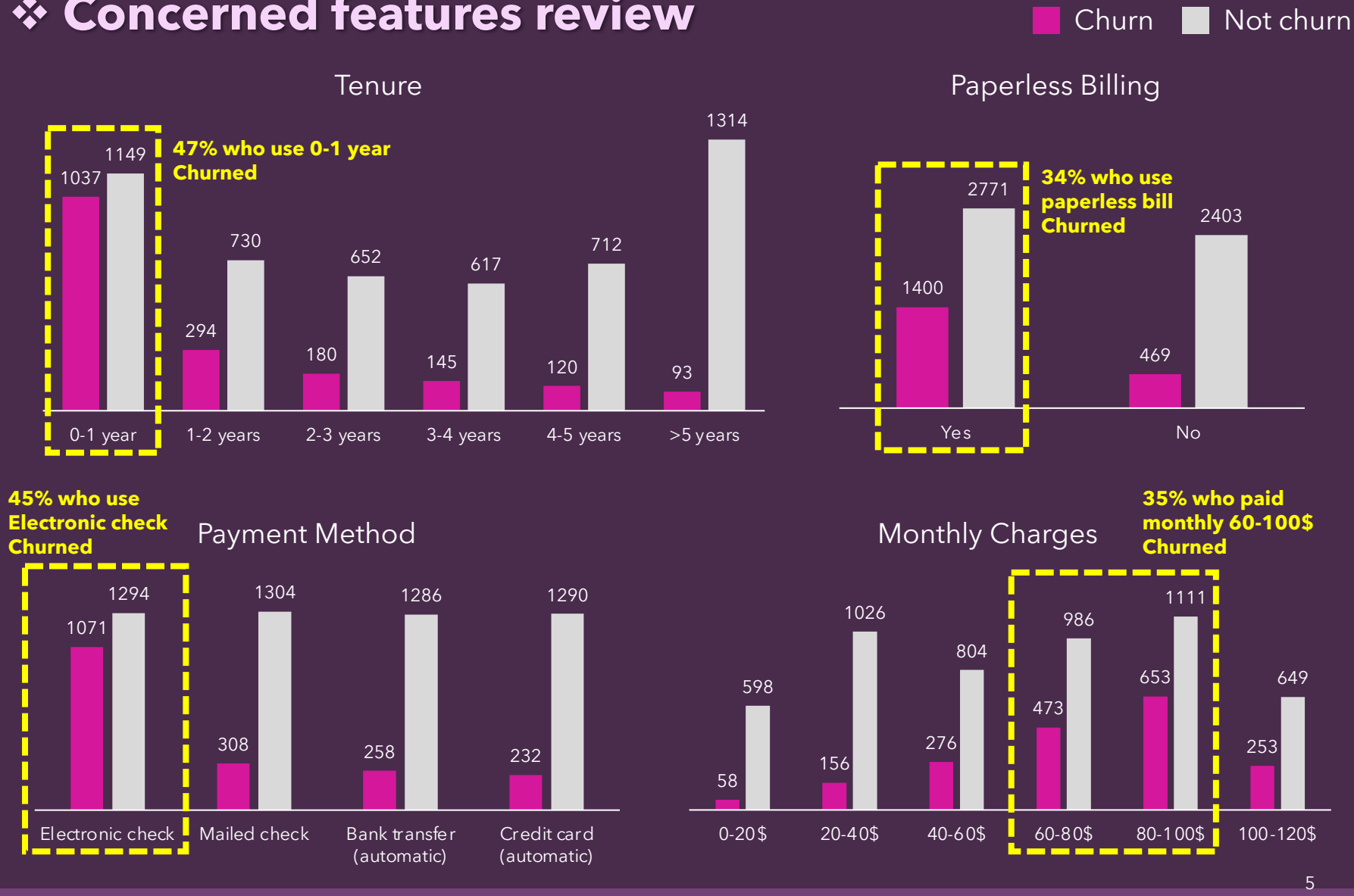
Churn status



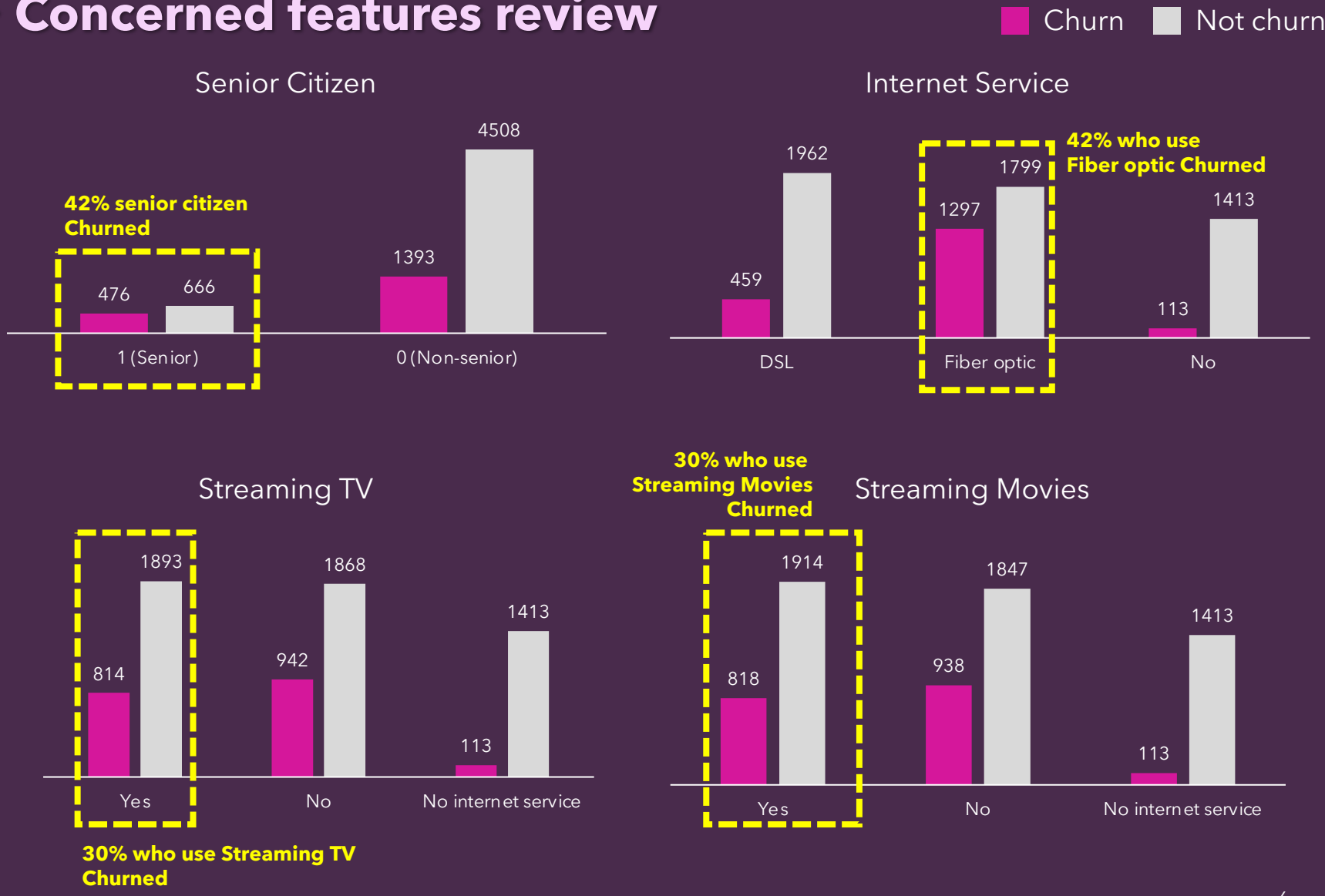
Analyze concerned features

Phone service
Monthly charges
Partner
Tenure
Multiple lines service
Internet service
Gender
Senior Citizen
Tech support
Streaming Movies
Dependents
Contract
Device protection
Streaming TV
Paperless billing
Online security
Online backup
Total charges
Payment method

❖ Concerned features review



❖ Concerned features review



❖ Target customers who might churn

Customers who ...	% Churn
are **new customer** (received service **< 1 year)**	**47%**
use **paperless billing**	**34%**
pay by **electronic check**	**45%**
have **monthly charges** around **60-100$**	**35%**
are **senior citizens**	**42%**
use **Fiber optic** (Internet Service)	**42%**
subscript **Streaming TV**	**30%**
subscribt **Streaming Movies**	**30%**

❖ Categorize concerned group *that might be useful for implementing predictive models*

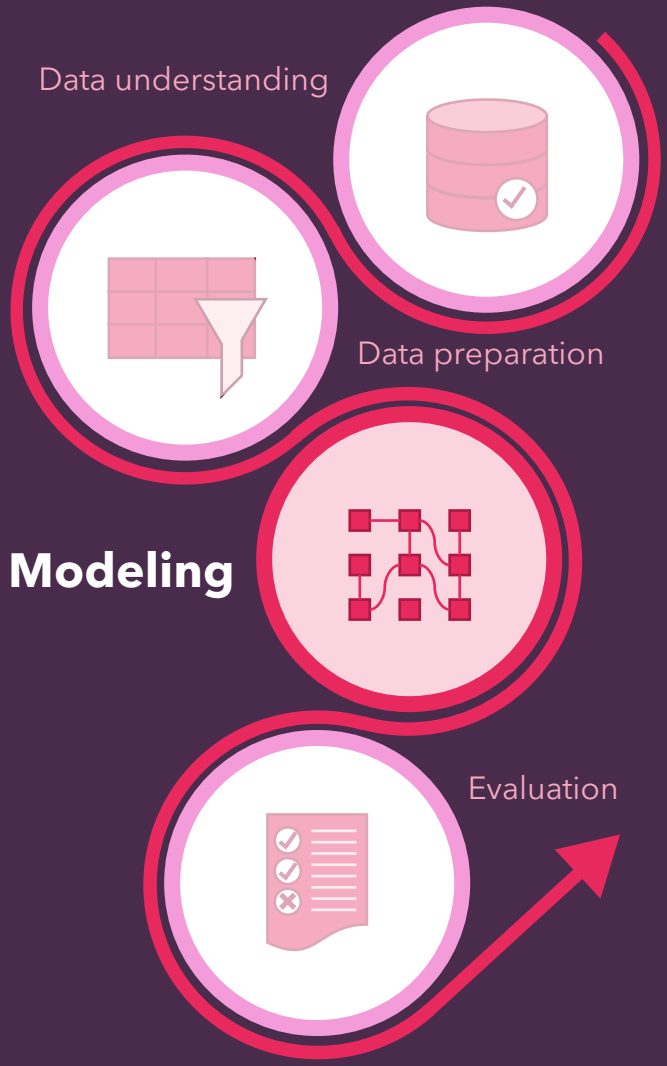
No.	Category	Detail
1	Tenure group (Every 1 year)	**Categorize by tenure range**: 0-1 year, 1-2 years, 2-3 years, 3-4 years, 4-5 years, and >5 years
2	Total services	**The number of services** that customer **receives**
3	Security concerns	Customers who **receive Online security & Device Protection** services
4	Entertainment concerns	Customers who **receive both Streaming TV and Movies services**
5	Technical support	**Senior citizens** who select to **receive Technical support service**
6	Billing and payment comfortable concerns	Customers who **select Paperless billing and Electronic check** as their services
7	Price range group	**Categorize by monthly paid range:** 0-20$, 20-40$, 40-60$, 60-80$, 80-100$, and 100-120$

*After categorized, **converted all concerned data into numerical term** to prepare for implementing predictive models*

❖ Implementing predictive models

Implemented in 9 selected models including

- Decision Trees
- Naive Bayes Classifier
- AdaBoost Classifier
- Multilayer Perceptron
- Bagging along with Random Forest
- K-nearest neighbor with Grid Search CV
- Logistic Regression with Grid Search CV
- Random Forest with Randomized Search CV
- Support Vector Machine with Grid Search CV



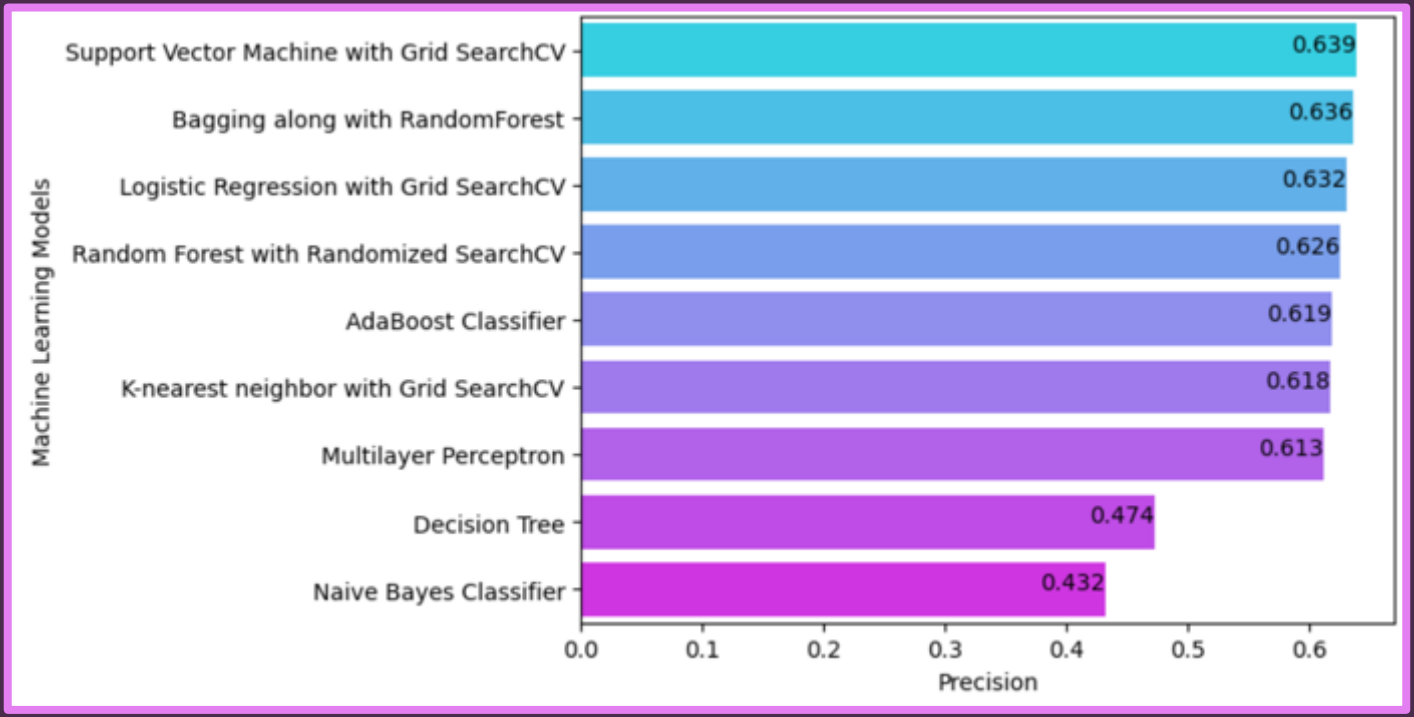
❖ Performance result

We want to focus on the performance to correctly identify most of churn customers (positive cases), then we **focus on "Precision" result**

No	Models	Accuracy	AUC-ROC	Precision	Recall	F1-Score
1	**Support Vector Machine with Grid Search CV**	79.56%	70.00%	**63.89%**	50.00%	56.10%
2	Bagging along with Random Forest	79.56%	70.27%	63.60%	50.82%	56.50%
3	Logistic Regression with Grid Search CV	79.77%	71.47%	63.17%	54.08%	58.27%
4	Random Forest with Randomized Search CV	79.28%	70.25%	62.58%	51.36%	56.42%
5	AdaBoost Classifier	79.28%	71.04%	61.88%	53.80%	57.56%
6	K-Nearest Neighbor with Grid Search CV	79.56%	72.20%	61.83%	56.80%	59.21%
7	Multilayer Perceptron	79.20%	71.52%	61.26%	55.43%	58.20%
8	Decision Tree	72.46%	64.76%	47.35%	48.64%	47.99%
9	Naïve Bayes Classifier	66.86%	73.09%	43.25%	86.14%	57.58%

❖ Conclusion

We have identified the optimal model based on precision evaluations is **Support Vector Machine with Grid Search CV produced the highest precision of 63.89%** for this specific problem. The Decision Tree and Naive Bayes Classifier models exhibited the poorest performance are 47.35% and 43.25%



THANK YOU

APPENDIX

APPENDIX

❖ Importing file & necessary libraries

Import the necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Import the dataset

In [2]:
```
data = pd.read_csv("Telco_Customer_Churn_Dataset.csv")
data.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupp
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	

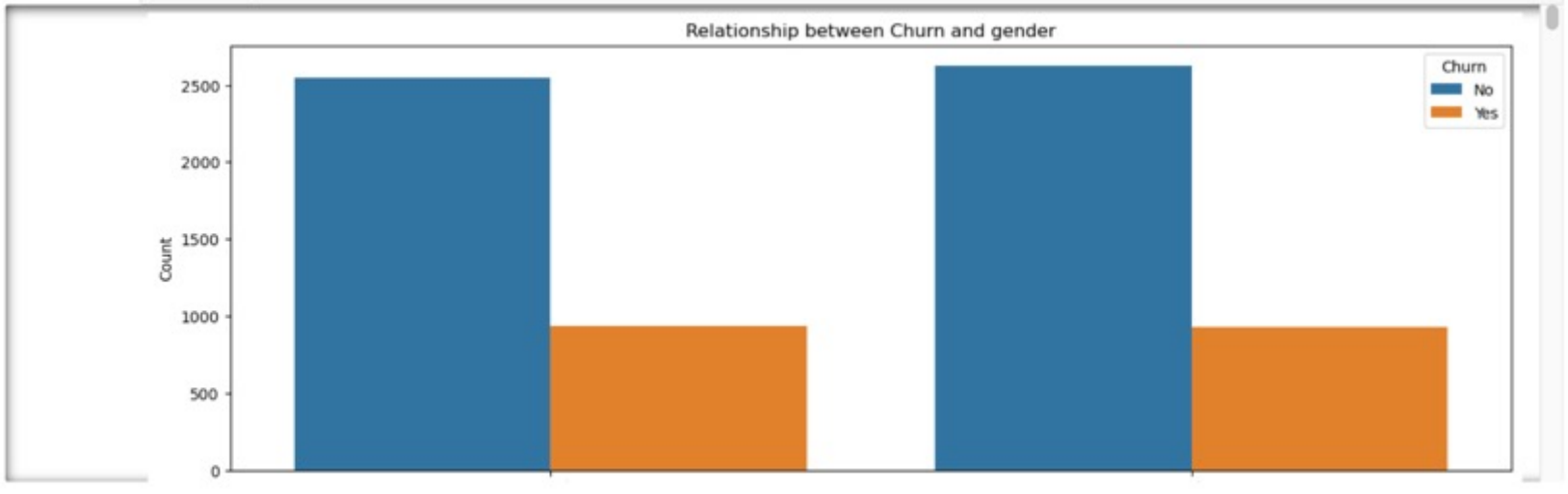
5 rows × 21 columns

APPENDIX

❖ Plotting bar charts *Relationship of Churn & each input*

Display the relationship between each factor and churn status

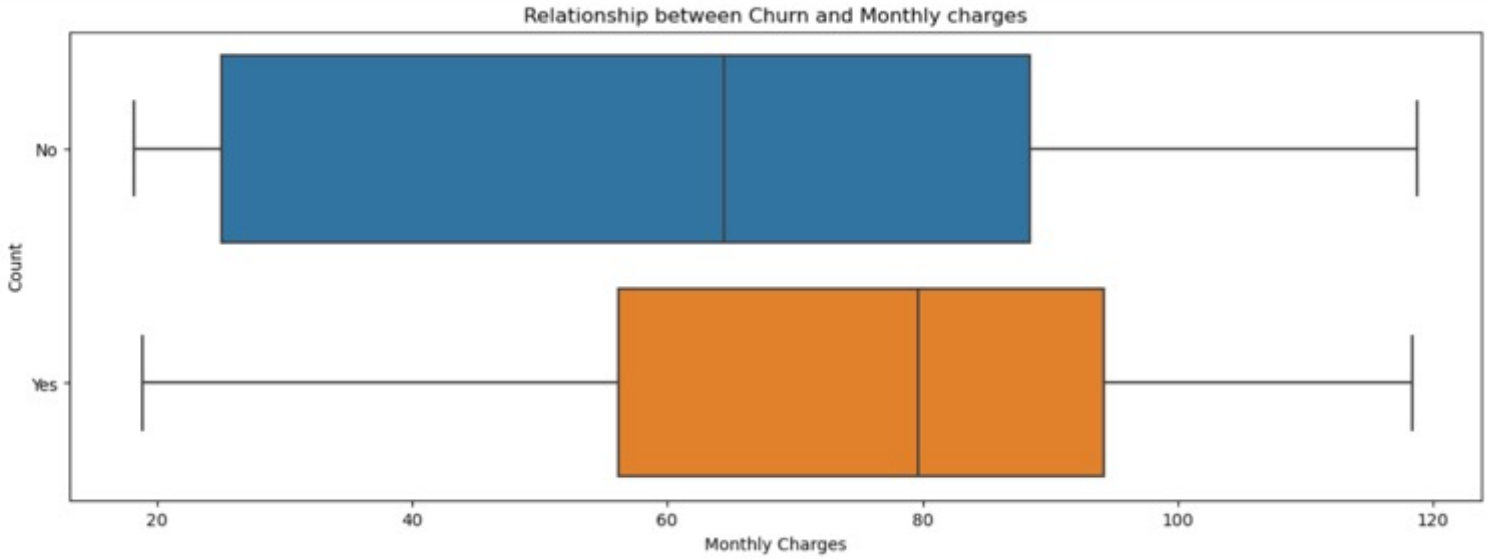
In [9]:
```
# Create bar chart to see the relationship between each column and "Churn" column
# Loop through each column in the dataframe
for col in categorical_columns[:-1]:
    # Create a bar chart for the column
    plt.figure(figsize=(15, 5))
    sns.countplot(x=col, hue="Churn", data=analyzed_data)
    plt.title(f"Relationship between Churn and {col}")
    plt.xlabel(col)
    plt.ylabel("Count")

# Create bar chart for the column "tenure"
plt.figure(figsize=(15,5))
sns.countplot(x="tenure", hue="Churn", data=analyzed_data)
plt.title("Relationship between Churn and tenure")
plt.xlabel("Tenure")
plt.ylabel("Count")
plt.show()
```



APPENDIX

❖ Plotting box chart *Relationship of Churn & Monthly charges*

```
In [10]: # Create the chart to see the relationship between Monthly charges and Churn
         plt.figure(figsize=(15,5))
         sns.boxplot(x="MonthlyCharges", y="Churn", data=analyzed_data)
         plt.title("Relationship between Churn and Monthly charges")
         plt.xlabel("Monthly Charges")
         plt.ylabel("Count")
         plt.show()
```



APPENDIX

❖ Checking missing values

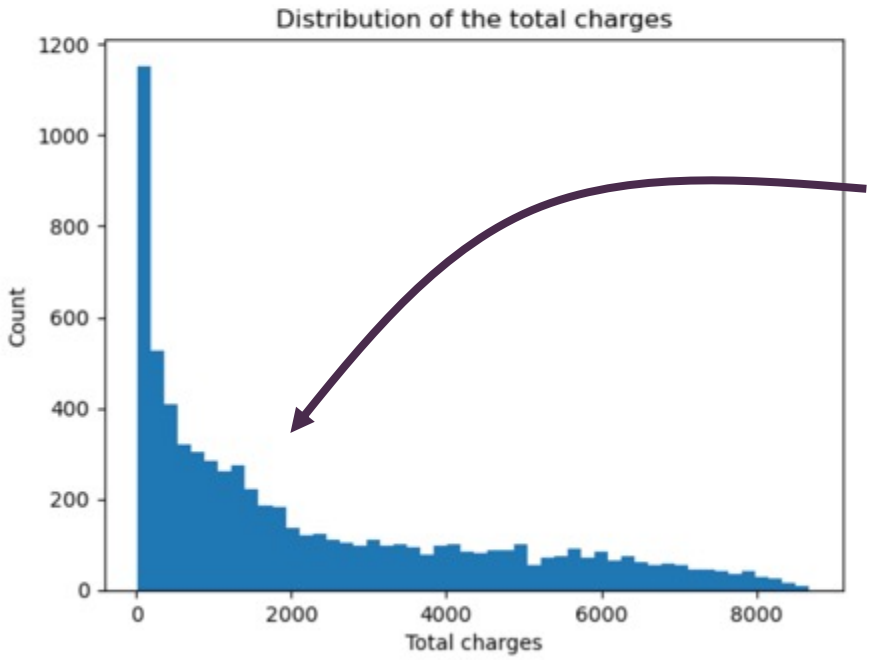
```
In [12]: # check missing values in each columns
         analyzed_data.isnull().sum()

Out[12]: customerID          0
         gender              0
         SeniorCitizen       0
         Partner             0
         Dependents          0
         tenure              0
         PhoneService        0
         MultipleLines       0
         InternetService     0
         OnlineSecurity      0
         OnlineBackup        0
         DeviceProtection    0
         TechSupport         0
         StreamingTV         0
         StreamingMovies     0
         Contract            0
         PaperlessBilling    0
         PaymentMethod       0
         MonthlyCharges      0
         TotalCharges       11
         Churn               0
         dtype: int64
```

← TotalCharges column contains missing values

❖ Skewed checking of the data distribution

Plot histogram to check the data distribution of "TotalCharges" that is skew or not

```
In [13]: # Plot histogram for the data of the column "TotalCharges" to see that the data distribution is skewed or not skewed
         # in order to decide that we should fill the data with mean, or median of the column.
         plt.hist(x=analyzed_data["TotalCharges"], bins=50)
         plt.xlabel("Total charges")
         plt.ylabel("Count")
         plt.title("Distribution of the total charges")

Out[13]: Text(0.5, 1.0, 'Distribution of the total charges')
```



Skewed = Fill those missing values by median of the column "TotalCharges"

APPENDIX

❖ Filling up & Rechecking missing values

Fill up the missing values with median

In [14]:
```
# Fill up the missing values in the column "TotalCharges" with the median of the column
analyzed_data["TotalCharges"] = analyzed_data["TotalCharges"].fillna(analyzed_data["TotalCharges"].median())

# Recheck the missing values of every columns
analyzed_data.isnull().sum()
```

Out[14]:
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

APPENDIX

❖ Categorize concerned features

In [15]:
```
# Create an another dataset for new features and implementing predictive models
implementing_data = analyzed_data.copy()
```

In [16]:
```
# Create new features that divides tenure by each 1 year tenure range
implementing_data['TenureRange'] = pd.cut(implementing_data['tenure'], bins=[-1,12,24,36,48,60,72], labels=['0-1 year',

# Create new feature that indicates the total number of services that the customer use
implementing_data["TotalServices"] = implementing_data[['PhoneService', 'InternetService',
                                                         'OnlineSecurity', 'OnlineBackup',
                                                         'DeviceProtection', 'TechSupport',
                                                         'StreamingTV', 'StreamingMovies']].replace({'Yes': 1, 'No': 0,

# Create new feature that concerns about security
implementing_data["Security"] = (implementing_data["OnlineSecurity"] == 'Yes') & (implementing_data["DeviceProtection"]

# Create new feature that concerns about entertainment
implementing_data["Entertainment"] = (implementing_data["StreamingTV"] == 'Yes') & (implementing_data["StreamingMovies"

# Create new feature that concerns about senior citizen that need technical support
implementing_data["SeniorTechSupport"] = (implementing_data["SeniorCitizen"] == 1) & (implementing_data["TechSupport"]

# Create new feature that concerns about billing and payment method
implementing_data["BillingAndPayment"] = (implementing_data["PaperlessBilling"] == 'Yes') & (implementing_data["Payment

# Create new feature that divides monthly charges by every 20 dollars range
implementing_data['MonthlyChargesRange'] = pd.cut(implementing_data['MonthlyCharges'], bins=[-1,20,40,60,80,100,120], l
```

APPENDIX

❖ Converting all values into numerical term

In [21]:
```
# Convert all values into numerical term

# Convert columns that contain 2 categories
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
bi_group_columns = ['gender', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling',
                    'Churn', 'Security', 'Entertainment', 'SeniorTechSupport', 'BillingAndPayment']
for col in bi_group_columns:
    train_data[col] = encoder.fit_transform(train_data[col])

# Convert columns that contain multiple categories (more than 2 groups)
# Provide one-hot encoding method
multi_group_columns = ['MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
                       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
                       'Contract', 'PaymentMethod', 'TenureRange', 'MonthlyChargesRange']
train_data = pd.get_dummies(train_data, columns=multi_group_columns, drop_first=True)
```

In [22]:
```
train_data.info()
```

APPENDIX

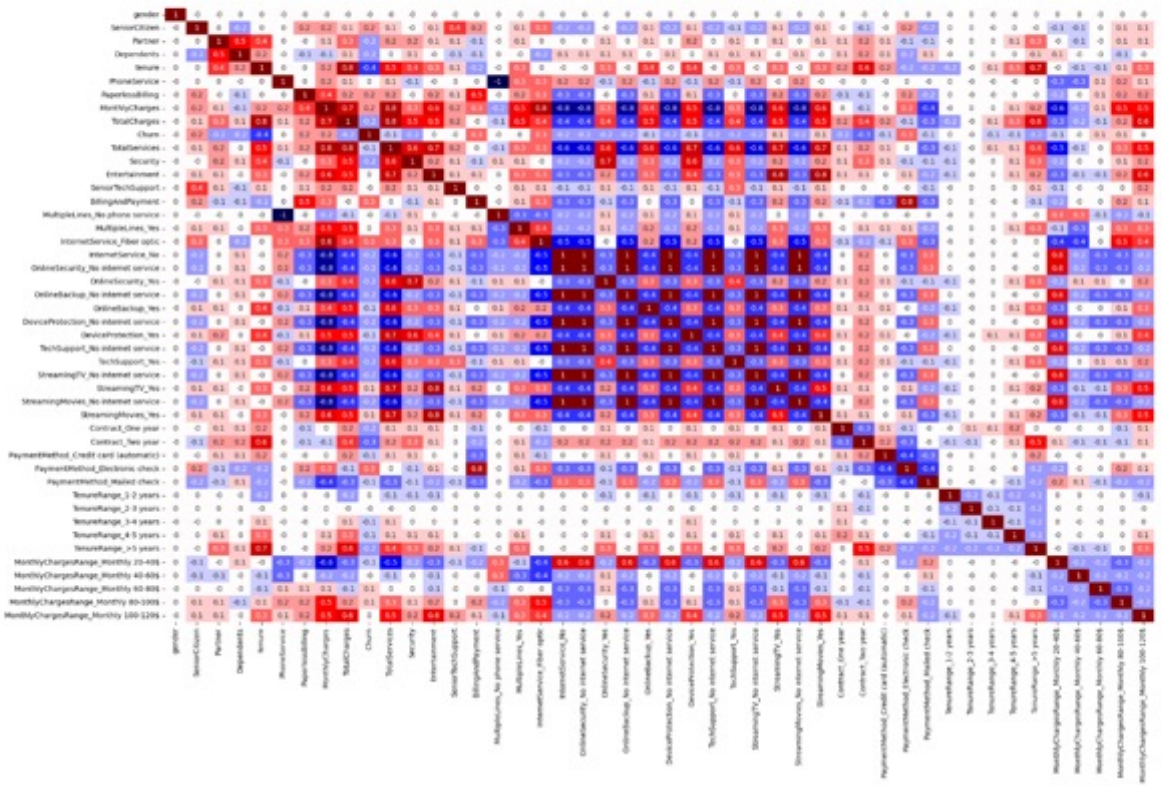
❖ Displaying correlation (Heatmap) *Between Churn and each input*

To select significant input features for implementing in the predictive models

Heatmap, displaying correlation between each variables and Churn

In [23]:
```
corr = train_data.corr()
plt.figure(figsize=(30, 15))
sns.heatmap(round(corr, 1), annot=True, cmap='seismic')
```

Out[23]: <AxesSubplot:>



APPENDIX

❖ Random Forest Importance

To select significant input features for implementing in the predictive models

Random Forest Importance

In [25]:
```
# Import the necessary libraries
from sklearn.ensemble import RandomForestClassifier

# Split your dataset into features and target
X_RF_GB = train_data.drop('Churn', axis=1)
y_RF_GB = train_data['Churn']

# Create a Random Forest classifier
rf_class = RandomForestClassifier()

# Train the model
rf_class.fit(X_RF_GB, y_RF_GB)

# Get the feature importances
rf_imp = rf_class.feature_importances_

# Create a DataFram for rf_imp
imp_table = pd.DataFrame({"Features": X_RF_GB.columns,
                          "Importance": rf_imp}).sort_values(by="Importance", ascending=False)
imp_table.reset_index(drop=True)
```

APPENDIX

❖ Standardizing the variables

Standardize the variables

In [27]:
```
# Import the library for normalizing the dataset
from sklearn.preprocessing import StandardScaler
```

In [28]:
```
scaler = StandardScaler()
scaler.fit(selected_train_data.drop('Churn', axis=1))
scaled_selected_train_data = scaler.transform(selected_train_data.drop('Churn',axis=1))

# See the dataframe after standardize the variables
table_scaled_selected_train_data = pd.DataFrame(scaled_selected_train_data, columns=selected_train_data.columns[:-1])
table_scaled_selected_train_data.head()
```

Out[28]:

	InternetService_Fiber optic	PaymentMethod_Electronic check	BillingAndPayment	MonthlyCharges	PaperlessBilling	SeniorCitizen	Partner	Security	Dependents	Tech
0	-0.885660	1.406418	1.744435	-1.160323	0.829798	-0.439916	1.034530	-0.432769	-0.654012	
1	-0.885660	-0.711026	-0.573251	-0.259629	-1.205113	-0.439916	-0.966622	2.310700	-0.654012	
2	-0.885660	-0.711026	-0.573251	-0.362660	0.829798	-0.439916	-0.966622	-0.432769	-0.654012	
3	-0.885660	-0.711026	-0.573251	-0.746535	-1.205113	-0.439916	-0.966622	2.310700	-0.654012	
4	1.129102	1.406418	1.744435	0.197365	0.829798	-0.439916	-0.966622	-0.432769	-0.654012	

5 rows × 23 columns

APPENDIX

❖ Splitting the dataset into train and test data

Split the dataset into train and test data (we select to divide into 80% as train data, and 20% as test data)

In [29]:
```
# Import the library for spliting the dataset into train and test data
from sklearn.model_selection import train_test_split
```

In [30]:
```
X = scaled_selected_train_data
y = selected_train_data['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50) # train 80% and test 20%
```

APPENDIX

❖ Decision Trees

Implement the model

1. Decision Trees

In [31]:
```
# Import the necessary library
from sklearn.tree import DecisionTreeClassifier

# Fit the model to training data
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)

# Make predictions
dtree_predictions = dtree.predict(X_test)

# Evaluate the model
# Import the necesasry library for evalution
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_auc_score, accuracy_score

print("The result of the decision trees model")
print("Classification report: \n", classification_report(y_test, dtree_predictions))
dtree_conf_matrix = confusion_matrix(y_test, dtree_predictions)
dtree_conf_matrix_df = pd.DataFrame(dtree_conf_matrix, index=["Actual 0", "Actual 1"], columns=["Predicted 0", "Predict
print("Confusion Matrix: \n", dtree_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, dtree_predictions))
print("Accuracy: ", accuracy_score(y_test, dtree_predictions))
```

APPENDIX

❖ Decision Trees

Confusion matrix of the model

```
The result of the decision trees model
Classification report:
              precision    recall  f1-score   support

           0       0.82      0.81      0.81      1041
           1       0.47      0.49      0.48       368

    accuracy                           0.72      1409
   macro avg       0.65      0.65      0.65      1409
weighted avg       0.73      0.72      0.73      1409

Confusion Matrix:
           Predicted 0  Predicted 1
Actual 0           842          199
Actual 1           189          179

AUC-ROC:  0.6476253497890824
Accuracy:  0.7246273953158269
```

APPENDIX

❖ Naïve Bayes Classifier

Implement the model

2. Naive Bayes Classifier

In [32]:
```
# Import the necessary library
from sklearn.naive_bayes import GaussianNB

# Fit the model to training data
nb = GaussianNB()
nb.fit(X_train, y_train)

# Make predictions
nb_predictions = nb.predict(X_test)

# Evaluate the model
print("The result of the naive bayes classifier")
print("Classification report: \n", classification_report(y_test, nb_predictions))
nb_conf_matrix = confusion_matrix(y_test, nb_predictions)
nb_conf_matrix_df = pd.DataFrame(nb_conf_matrix, index=["Actual 0", "Actual 1"], columns=["Predicted 0", "Predicted 1"]
print("Confusion Matrix: \n", nb_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, nb_predictions))
print("Accuracy: ", accuracy_score(y_test, nb_predictions))
```

APPENDIX

❖ Naïve Bayes Classifier

Confusion matrix of the model

```
The result of the naive bayes classifier
Classification report:
              precision    recall  f1-score   support

           0       0.92      0.60      0.73      1041
           1       0.43      0.86      0.58       368

    accuracy                           0.67      1409
   macro avg       0.68      0.73      0.65      1409
weighted avg       0.80      0.67      0.69      1409

Confusion Matrix:
           Predicted 0  Predicted 1
Actual 0           625          416
Actual 1            51          317

AUC-ROC:  0.7308986446978241
Accuracy:  0.6685592618878637
```

APPENDIX

❖ AdaBoost Classifier

Implement the model

3. AdaBoost Classifier

In [33]:
```
# Import the necessary library
from sklearn.ensemble import AdaBoostClassifier

# Fit the model to training data
ab = AdaBoostClassifier()
ab.fit(X_train, y_train)

# Make predictions
ab_predictions = ab.predict(X_test)

# Evaluate the model
print("The result of the adaboost classifier")
print("Classification report: \n", classification_report(y_test, ab_predictions))
ab_conf_matrix = confusion_matrix(y_test, ab_predictions)
ab_conf_matrix_df = pd.DataFrame(ab_conf_matrix, index=["Actual 0", "Actual 1"], columns=["Predicted 0", "Predicted 1"]
print("Confusion Matrix: \n", ab_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, ab_predictions))
print("Accuracy: ", accuracy_score(y_test, ab_predictions))
```

APPENDIX

❖ AdaBoost Classifier

Confusion matrix of the model

```
The result of the adaboost classifier
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1041
           1       0.62      0.54      0.58       368

    accuracy                           0.79      1409
   macro avg       0.73      0.71      0.72      1409
weighted avg       0.79      0.79      0.79      1409

Confusion Matrix:
           Predicted 0  Predicted 1
Actual 0           919          122
Actual 1           170          198

AUC-ROC:  0.7104242367288978
Accuracy:  0.7927608232789212
```

APPENDIX

❖ Multilayer Perceptron

Implement the model

4. Multilayer Perceptron

In [34]:
```
# Import the necessary library
from sklearn.neural_network import MLPClassifier

# Fit the MLP classifier to training data (set max iteration = 500)
mlp = MLPClassifier(max_iter=500)
mlp.fit(X_train, y_train)

# Make predictions
mlp_predictions = mlp.predict(X_test)

# Evaluate the model
print("The result of the multilayer perceptron")
print("Classification report: \n", classification_report(y_test, mlp_predictions))
mlp_conf_matrix = confusion_matrix(y_test, mlp_predictions)
mlp_conf_matrix_df = pd.DataFrame(mlp_conf_matrix, index=["Actual 0", "Actual 1"], columns=["Predicted 0", "Predicted 1
print("Confusion Matrix: \n", mlp_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, mlp_predictions))
print("Accuracy: ", accuracy_score(y_test, mlp_predictions))
```

APPENDIX

❖ Multilayer Perceptron

Confusion matrix of the model

```
The result of the multilayer perceptron
Classification report:
              precision    recall  f1-score   support

           0       0.85      0.88      0.86      1041
           1       0.61      0.55      0.58       368

    accuracy                           0.79      1409
   macro avg       0.73      0.72      0.72      1409
weighted avg       0.79      0.79      0.79      1409

Confusion Matrix:
           Predicted 0  Predicted 1
Actual 0           912          129
Actual 1           164          204

AUC-ROC:  0.7152142588648039
Accuracy:  0.7920511000709723
```

APPENDIX

❖ Bagging along with Random Forest

Implement the model

5. Bagging along with RandomForest

In [35]:
```
# Import the necessary library
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier

# Define the base classifier to use in the bagging ensemble
base_clf = RandomForestClassifier(n_estimators=100)

# Define the bagging classifier
bagging_clf = BaggingClassifier(base_estimator=base_clf, n_estimators=10)

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

# Make predictions
bg_rf_predictions = bagging_clf.predict(X_test)

# Evaluate the model
print("The result of the bagging along with random forest")
print("Classification report: \n", classification_report(y_test, bg_rf_predictions))
bg_rf_conf_matrix = confusion_matrix(y_test, bg_rf_predictions)
bg_rf_conf_matrix_df = pd.DataFrame(bg_rf_conf_matrix, index=["Actual 0", "Actual 1"], columns=["Predicted 0", "Predict
print("Confusion Matrix: \n", bg_rf_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, bg_rf_predictions))
print("Accuracy: ", accuracy_score(y_test, bg_rf_predictions))
```

APPENDIX

❖ Bagging along with Random Forest

Confusion matrix of the model

```
The result of the bagging along with random forest
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1041
           1       0.64      0.51      0.56       368

    accuracy                           0.80      1409
   macro avg       0.74      0.70      0.72      1409
weighted avg       0.79      0.80      0.79      1409

Confusion Matrix:
            Predicted 0  Predicted 1
Actual 0            934          107
Actual 1            181          187

AUC-ROC:  0.7026831955059935
Accuracy:  0.7955997161107168
```

APPENDIX

❖ K-Nearest Neighbor with Grid Search CV

Selecting the best k value for this specific problem

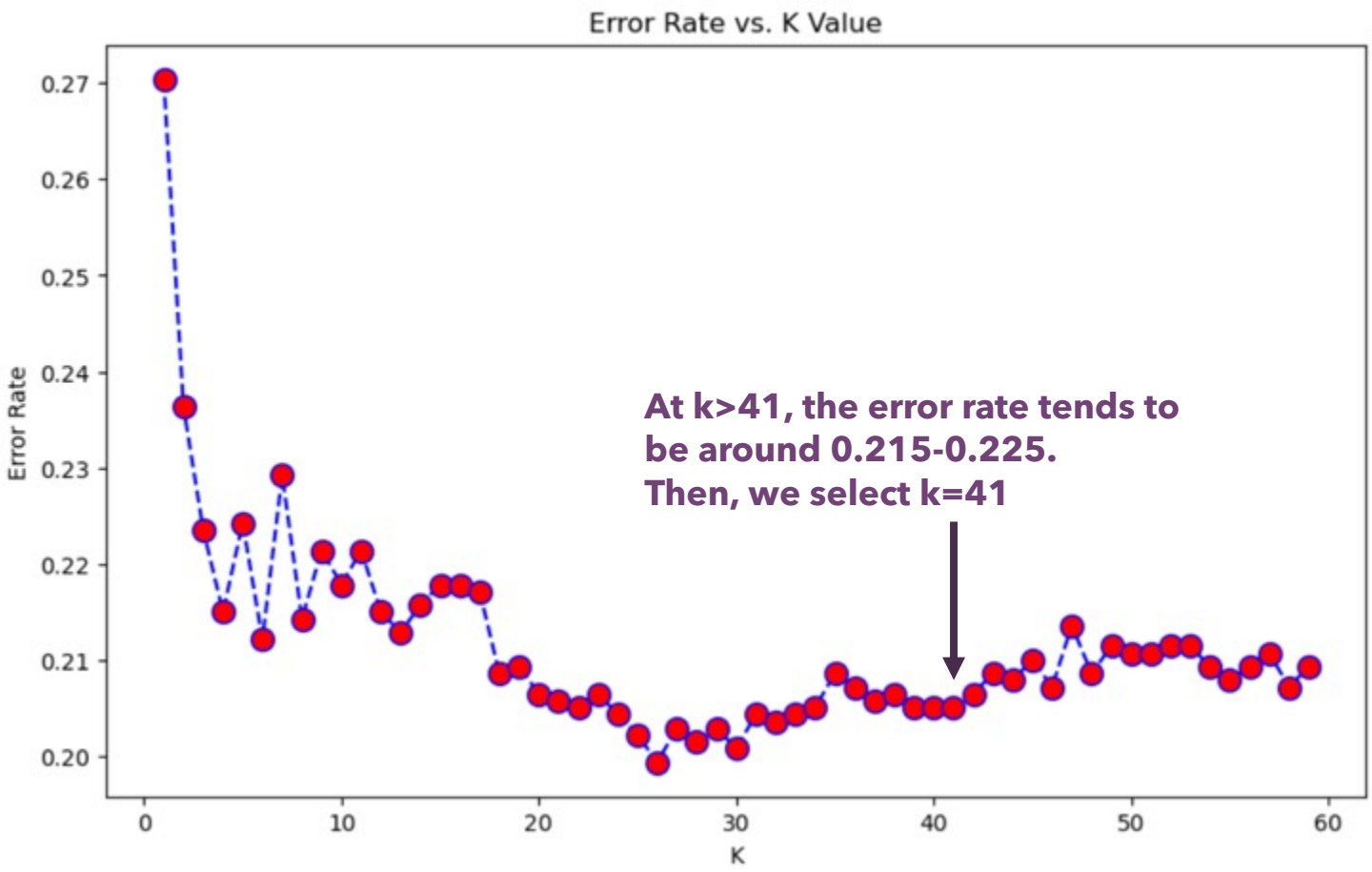
6. K-nearest neighbor with Grid SearchCV

In [36]:
```
# Import the necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from warnings import simplefilter

# Ignore future warning
simplefilter(action='ignore', category=FutureWarning)

# Choose the best k
error_rate = []
for k in range(1,60):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    pred_k = knn.predict(X_test)
    error_rate.append(np.mean(pred_k != y_test))

# Plot graph to see the k trend
plt.figure(figsize=(10,6))
plt.plot(range(1,60), error_rate,color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

❖ K-Nearest Neighbor with Grid Search CV (Cont..)

The graph displays the trend of k values

Out[36]: Text(0, 0.5, 'Error Rate')



❖ K-Nearest Neighbor with Grid Search CV (Cont..)

Implement the model

In [37]:

```
# Create a KNN classifier object
knn = KNeighborsClassifier()

# Define a dictionary of hyperparameters / also try to use other k values to compare with k = 41 that we selected
knn_param_grid = {'n_neighbors': [6, 16, 21, 41],
                  'weights': ['uniform', 'distance'],
                  'p': [1, 2]}

# Create a GridSearchCV object
knn_grid_search = GridSearchCV(knn, param_grid=knn_param_grid, cv=10, scoring='accuracy')

# Fit the GridSearchCV object to the training data
knn_grid_search.fit(X_train, y_train)

# Get the best parameters and the best score
knn_best_params = knn_grid_search.best_params_
knn_best_score = knn_grid_search.best_score_
print("The best parameter: ", knn_best_params)
print("The best score: ", knn_best_params)

# Fit the KNN model with the best parameters
knn_best = KNeighborsClassifier(n_neighbors=knn_best_params['n_neighbors'],
                                weights=knn_best_params['weights'],
                                p=knn_best_params['p'])
knn_best.fit(X_train, y_train)

# Predict on the test data using the best model
knn_gridSearch_predictions = knn_best.predict(X_test)

# Evaluate the model
print("\nThe result of the KNN classifier with grid search CV")
print("Classification report: \n", classification_report(y_test, knn_gridSearch_predictions))
knn_gridSearch_conf_matrix = confusion_matrix(y_test, knn_gridSearch_predictions)
knn_gridSearch_conf_matrix_df = pd.DataFrame(knn_gridSearch_conf_matrix, index=["Actual 0", "Actual 1"], columns=["Pred
print("Confusion Matrix: \n", knn_gridSearch_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, knn_gridSearch_predictions))
print("Accuracy: ", accuracy_score(y_test, knn_gridSearch_predictions))
```

Confirm with grid search CV again that k=41 is the best k

APPENDIX

❖ K-Nearest Neighbor with Grid Search CV (Cont..)

Confusion matrix of the model

```
The best parameter:  {'n_neighbors': 41, 'p': 1, 'weights': 'uniform'}
The best score:  {'n_neighbors': 41, 'p': 1, 'weights': 'uniform'}

The result of the KNN classifier with grid search CV
Classification report:
              precision    recall  f1-score   support

           0       0.85      0.88      0.86      1041
           1       0.62      0.57      0.59       368

    accuracy                           0.80      1409
   macro avg       0.73      0.72      0.73      1409
weighted avg       0.79      0.80      0.79      1409

Confusion Matrix:
           Predicted 0  Predicted 1
Actual 0           912          129
Actual 1           159          209

AUC-ROC:  0.7220077371256736
Accuracy:  0.7955997161107168
```

❖ Logistic Regression with Grid Search CV

Implement the model

7. Logistic Regression with Grid SearchCV

In [38]:

```
# import the necessary library
from sklearn.linear_model import LogisticRegression

# Define the parameter grid
log_reg_param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

# Create a logistic regression object
log_reg = LogisticRegression()

# Create a GridSearchCV object
log_reg_grid_search = GridSearchCV(log_reg, param_grid=log_reg_param_grid, scoring='accuracy')

# Fit the GridSearchCV object to the training data
log_reg_grid_search.fit(X_train, y_train)

# Get the best parameters and the best score
log_reg_best_params = log_reg_grid_search.best_params_
log_reg_best_score = log_reg_grid_search.best_score_
print("The best parameter: ", log_reg_best_params)
print("The best score: ", log_reg_best_score)

# Fit the logistic regression with the best parameters
log_reg_best = LogisticRegression(C=log_reg_best_params['C'])
log_reg_best.fit(X_train, y_train)

# Predict on the test data using the best model
log_reg_gridSearch_predictions = log_reg_best.predict(X_test)

# Evaluate the model
print("\nThe result of the logistic regression with grid search CV")
print("Classification report: \n", classification_report(y_test, log_reg_gridSearch_predictions))
log_reg_gridSearch_conf_matrix = confusion_matrix(y_test, log_reg_gridSearch_predictions)
log_reg_gridSearch_conf_matrix_df = pd.DataFrame(log_reg_gridSearch_conf_matrix, index=["Actual 0", "Actual 1"], column
print("Confusion Matrix: \n", log_reg_gridSearch_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, log_reg_gridSearch_predictions))
print("Accuracy: ", accuracy_score(y_test, log_reg_gridSearch_predictions))
```

❖ Logistic Regression with Grid Search CV (Cont.)

Confusion matrix of the model

```
The best parameter:  {'C': 10}
The best score:  0.8012088239419638

The result of the logistic regression with grid search CV
Classification report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      1041
           1       0.63      0.54      0.58       368

    accuracy                           0.80      1409
   macro avg       0.74      0.71      0.72      1409
weighted avg       0.79      0.80      0.79      1409

Confusion Matrix:
           Predicted 0  Predicted 1
Actual 0           925          116
Actual 1           169          199

AUC-ROC:  0.7146647767614751
Accuracy:  0.7977288857345636
```

APPENDIX

❖ Random Forest with Randomized Search CV

Implement the model

8. Random Forest with Randomized SearchCV

In [39]:

```
# import the necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

# Define the parameter distribution
rf_param_dist = {'n_estimators': [100, 200, 300]}

# Create a random forest object
rf = RandomForestClassifier()

# Create a RandomSearchCV object
rf_random_search = RandomizedSearchCV(rf,
                                      param_distributions=rf_param_dist,
                                      n_iter=3,
                                      cv=10,
                                      scoring='accuracy')

# Fit the RandomSearchCV object to the training data
rf_random_search.fit(X_train, y_train)

# Get the best parameters and the best score
rf_best_params = rf_random_search.best_params_
rf_best_score = rf_random_search.best_score_
print("The best parameter: ", rf_best_params)
print("The best score: ", rf_best_score)

# Fit the random forest with the best parameters
rf_best = RandomForestClassifier(n_estimators=rf_best_params['n_estimators'])
rf_best.fit(X_train, y_train)

# Predict on the test data using the best model
rf_randomSearch_predictions = rf_best.predict(X_test)

# Evaluate the model
print("\nThe result of the random forest with random search CV")
print("Classification report: \n", classification_report(y_test, rf_randomSearch_predictions))
rf_randomSearch_conf_matrix = confusion_matrix(y_test, rf_randomSearch_predictions)
rf_randomSearch_conf_matrix_df = pd.DataFrame(rf_randomSearch_conf_matrix, index=["Actual 0", "Actual 1"], columns=["Pr
print("Confusion Matrix: \n", rf_randomSearch_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, rf_randomSearch_predictions))
print("Accuracy: ", accuracy_score(y_test, rf_randomSearch_predictions))
```

❖ Random Forest with Randomized Search CV (Cont.)

Confusion matrix of the model

```
The best parameter:  {'n_estimators': 200}
The best score:  0.7859377952458335

The result of the random forest with random search CV
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.89      0.86      1041
           1       0.63      0.51      0.56       368

    accuracy                           0.79      1409
   macro avg       0.73      0.70      0.71      1409
weighted avg       0.78      0.79      0.79      1409

Confusion Matrix:
           Predicted 0  Predicted 1
Actual 0           928          113
Actual 1           179          189

AUC-ROC:  0.7025187424299378
Accuracy:  0.7927608232789212
```

APPENDIX

❖ Support Vector Machine with Grid Search CV

Implement the model

9. Support Vector Machine with Grid SearchCV

In [40]:

```
# Import the necessary library
from sklearn.svm import SVC

# Define the parameter grid
svm_param_grid = {'C': [0.1, 1, 10, 100]}

# Create a support vector machine object
svm = SVC()

# Create a GridSearchCV object
svm_grid_search = GridSearchCV(svm, param_grid=svm_param_grid, scoring='accuracy')

# Fit the GridSearchCV object to the training data
svm_grid_search.fit(X_train, y_train)

# Get the best parameters and the best score
svm_best_params = svm_grid_search.best_params_
svm_best_score = svm_grid_search.best_score_
print("The best parameter: ", svm_best_params)
print("The best score: ", svm_best_score)

# Fit the support vector machine with the best parameters
svm_best = SVC(C=svm_best_params['C'])
svm_best.fit(X_train, y_train)

# Predict on the test data using the best model
svm_gridSearch_predictions = svm_best.predict(X_test)

# Evaluate the model
print("\nThe result of the support vector machine with grid search CV")
print("Classification report: \n", classification_report(y_test, svm_gridSearch_predictions))
svm_gridSearch_conf_matrix = confusion_matrix(y_test, svm_gridSearch_predictions)
svm_gridSearch_conf_matrix_df = pd.DataFrame(svm_gridSearch_conf_matrix, index=["Actual 0", "Actual 1"], columns=["Pred
print("Confusion Matrix: \n", svm_gridSearch_conf_matrix_df)
print("\nAUC-ROC: ", roc_auc_score(y_test, svm_gridSearch_predictions))
print("Accuracy: ", accuracy_score(y_test, svm_gridSearch_predictions))
```

❖ Support Vector Machine with Grid Search CV (Cont.)

Confusion matrix of the model

```
The best parameter:  {'C': 1}
The best score:  0.7967719515020464

The result of the support vector machine with grid search CV
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1041
           1       0.64      0.50      0.56       368

    accuracy                           0.80      1409
   macro avg       0.74      0.70      0.71      1409
weighted avg       0.78      0.80      0.79      1409

Confusion Matrix:
           Predicted 0  Predicted 1
Actual 0           937          104
Actual 1           184          184

AUC-ROC:  0.7000480307396734
Accuracy:  0.7955997161107168
```

APPENDIX

❖ Creating summary table

```
In [41]: from sklearn.metrics import precision_score, recall_score, f1_score

result_table = {'Machine Learning Models': ['Decision Tree',
                                            'Naive Bayes Classifier',
                                            'AdaBoost Classifier',
                                            'Multilayer Perceptron',
                                            'Bagging along with RandomForest',
                                            'K-nearest neighbor with Grid SearchCV',
                                            'Logistic Regression with Grid SearchCV',
                                            'Random Forest with Randomized SearchCV',
                                            'Support Vector Machine with Grid SearchCV'],
                'Accuracy': [accuracy_score(y_test, dtree_predictions),
                             accuracy_score(y_test, nb_predictions),
                             accuracy_score(y_test, ab_predictions),
                             accuracy_score(y_test, mlp_predictions),
                             accuracy_score(y_test, bg_rf_predictions),
                             accuracy_score(y_test, knn_gridSearch_predictions),
                             accuracy_score(y_test, log_reg_gridSearch_predictions),
                             accuracy_score(y_test, rf_randomSearch_predictions),
                             accuracy_score(y_test, svm_gridSearch_predictions)],
                'AUC-ROC': [roc_auc_score(y_test, dtree_predictions),
                            roc_auc_score(y_test, nb_predictions),
                            roc_auc_score(y_test, ab_predictions),
                            roc_auc_score(y_test, mlp_predictions),
                            roc_auc_score(y_test, bg_rf_predictions),
                            roc_auc_score(y_test, knn_gridSearch_predictions),
                            roc_auc_score(y_test, log_reg_gridSearch_predictions),
                            roc_auc_score(y_test, rf_randomSearch_predictions),
                            roc_auc_score(y_test, svm_gridSearch_predictions)],
                'Precision': [precision_score(y_test, dtree_predictions),
                              precision_score(y_test, nb_predictions),
                              precision_score(y_test, ab_predictions),
                              precision_score(y_test, mlp_predictions),
                              precision_score(y_test, bg_rf_predictions),
                              precision_score(y_test, knn_gridSearch_predictions),
                              precision_score(y_test, log_reg_gridSearch_predictions),
                              precision_score(y_test, rf_randomSearch_predictions),
                              precision_score(y_test, svm_gridSearch_predictions)],
                'Recall': [recall_score(y_test, dtree_predictions),
                           recall_score(y_test, nb_predictions),
                           recall_score(y_test, ab_predictions),
                           recall_score(y_test, mlp_predictions),
                           recall_score(y_test, bg_rf_predictions),
                           recall_score(y_test, knn_gridSearch_predictions),
                           recall_score(y_test, log_reg_gridSearch_predictions),
                           recall_score(y_test, rf_randomSearch_predictions),
                           recall_score(y_test, svm_gridSearch_predictions)],
                'F1-Score': [f1_score(y_test, dtree_predictions),
                             f1_score(y_test, nb_predictions),
                             f1_score(y_test, ab_predictions),
                             f1_score(y_test, mlp_predictions),
                             f1_score(y_test, bg_rf_predictions),
                             f1_score(y_test, knn_gridSearch_predictions),
                             f1_score(y_test, log_reg_gridSearch_predictions),
                             f1_score(y_test, rf_randomSearch_predictions),
                             f1_score(y_test, svm_gridSearch_predictions)]}

result_df = pd.DataFrame(data=result_table)
```

APPENDIX

❖ Creating summary table (Cont.)

Sorting the table by descending of the column “Precision”

In [43]:
```
# Sorted by precision
sorted_prec_result_df = result_df.sort_values(by='Precision', ascending=False).reset_index(drop=True)
sorted_prec_result_df
```

Out[43]:

	Machine Learning Models	Accuracy	AUC-ROC	Precision	Recall	F1-Score
0	Support Vector Machine with Grid SearchCV	0.795600	0.700048	0.638889	0.500000	0.560976
1	Bagging along with RandomForest	0.795600	0.702683	0.636054	0.508152	0.564955
2	Logistic Regression with Grid SearchCV	0.797729	0.714665	0.631746	0.540761	0.582723
3	Random Forest with Randomized SearchCV	0.792761	0.702519	0.625828	0.513587	0.564179
4	AdaBoost Classifier	0.792761	0.710424	0.618750	0.538043	0.575581
5	K-nearest neighbor with Grid SearchCV	0.795600	0.722008	0.618343	0.567935	0.592068
6	Multilayer Perceptron	0.792051	0.715214	0.612613	0.554348	0.582026
7	Decision Tree	0.724627	0.647625	0.473545	0.486413	0.479893
8	Naive Bayes Classifier	0.668559	0.730899	0.432469	0.861413	0.575840

APPENDIX

❖ Plotting summary charts

Comparison of the precision score between each model

```
In [44]: # Precision graphs
result_prec_graph = sns.barplot(x='Precision',
                                y='Machine Learning Models',
                                data=sorted_prec_result_df,
                                palette='cool'
                               )

# Add values on the bar charts
for index, row in sorted_prec_result_df.iterrows():
    result_prec_graph.text(row['Precision'], index, "{:.3f}".format(row['Precision']), color='black', ha='right')
```