Business Understanding

Companies lose millions due to customer churn. Understanding why customers leave can help businesses take action to retain them, such as offering promotions or improving service quality.

Problem statement

How can the company predict customer churn using historical data, and what factors contribute most to customer churn?

Objectives

1.Build a classification model to predict churn based on customer data. 2.Compare multiple models.

3.Provide recommendations for reducing churn.

Research Questions

1. What factors influence customer churn the most?
2. How accurately can we predict churn using available data?
3. Which classification model performs best for this problem?

Data Understanding

```
#load the data
import kagglehub

# Download latest version
path = kagglehub.dataset_download("becksddf/churn-in-telecoms-dataset")

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/becksddf/churn-in-telecoms-dataset?dataset_version_number=1...
100%|██████████| 116k/116k [00:00<00:00, 37.8MB/s]Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/becksddf/churn-in-telecoms-dataset/versions/1
```

```
import zipfile
import pandas as pd
import os

#Extracting the ZIP file
with zipfile.ZipFile("/content/archive.zip", "r") as zip_ref:
    zip_ref.extractall("/content")

files = os.listdir("/content")
print("Extracted files:", files)

csv_file = "bigml_59c28831336c6604c800002a.csv"
df = pd.read_csv(f"/content/{csv_file}")

#first few rows
df.head()
```

```
Extracted files: ['.config', 'archive.zip', 'bigml_59c28831336c6604c800002a.csv', 'sample_data']
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	tot in minut
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	1(
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	1:
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	1:
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	(
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	1(

5 rows × 21 columns

```
#structure
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

```
#missin values
df.isnull().sum()
```

	0
state	0
account length	0
area code	0
phone number	0
international plan	0
voice mail plan	0
number vmail messages	0
total day minutes	0
total day calls	0
total day charge	0
total eve minutes	0
total eve calls	0
total eve charge	0
total night minutes	0
total night calls	0
total night charge	0
total intl minutes	0
total intl calls	0
total intl charge	0
customer service calls	0
churn	0

```
#class distribution
df['churn'].value_counts(normalize=True)
```

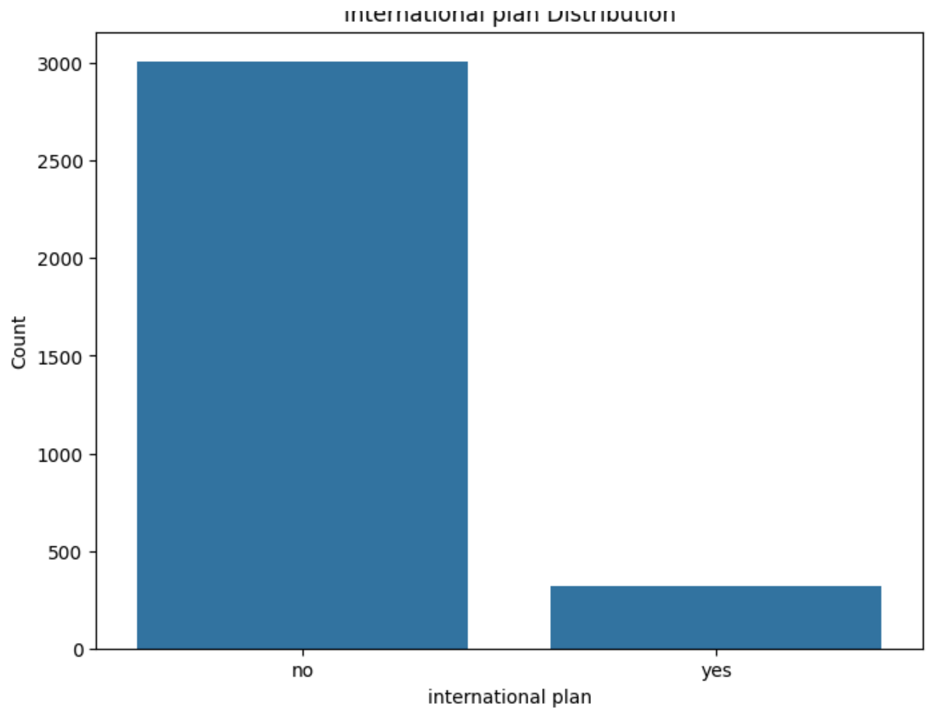
churn	proportion
False	0.855086
True	0.144914

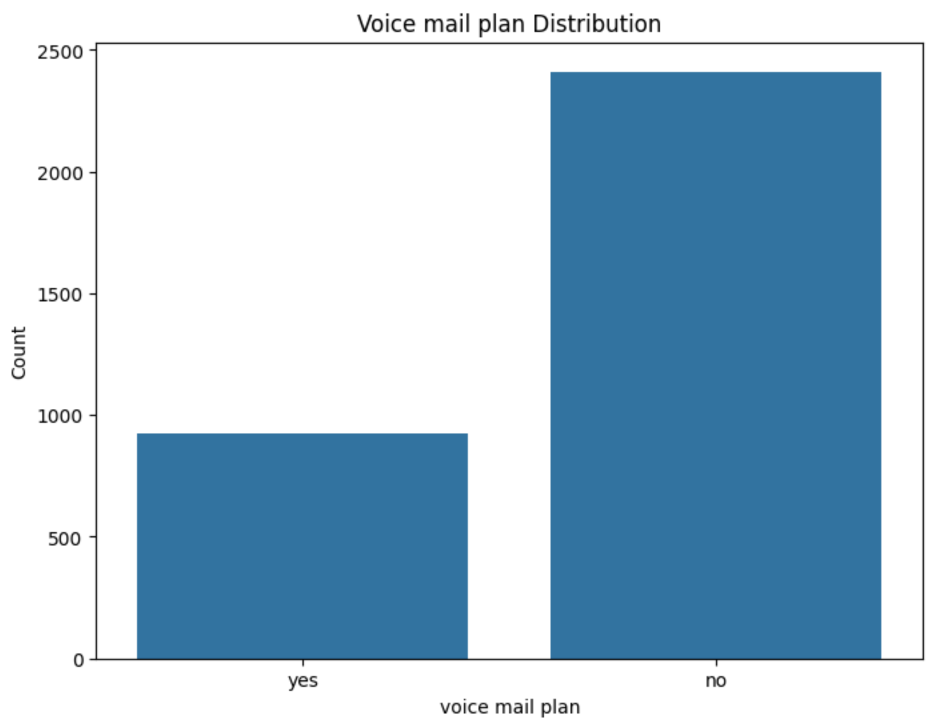
EDA

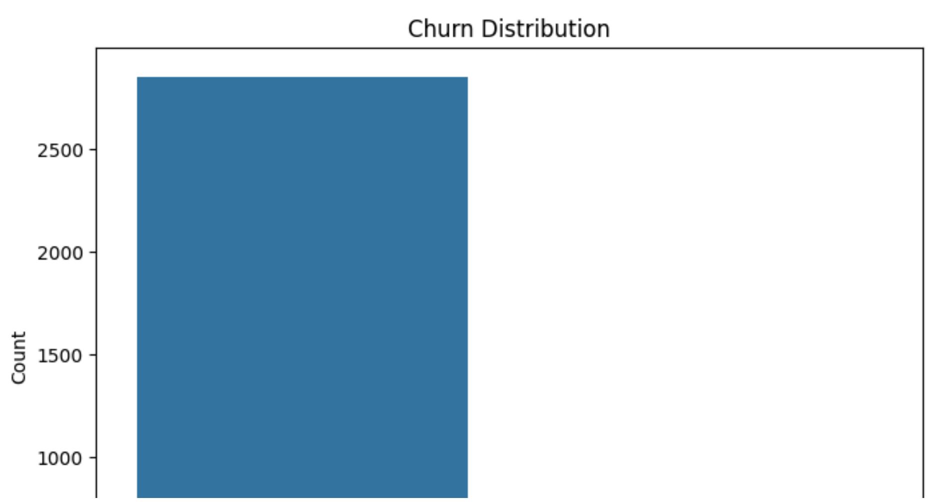
```
#summary
df.describe()
```

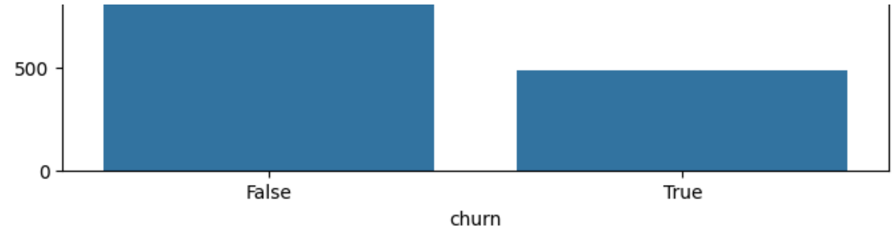
	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	333
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17.083540	200.872037	10
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847	1
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000	3
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000	8
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000	10
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000	11
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000	17

```
#import libraries
import matplotlib.pyplot as plt
import seaborn as sns


#Plot categorical feature distributions
categorical_cols = ['international plan', 'voice mail plan', 'churn']

for col in categorical_cols:
    plt.figure(figsize=(8, 6))
    sns.countplot(x=col, data=df)
    plt.title(f'{col.capitalize()} Distribution')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.show()
```

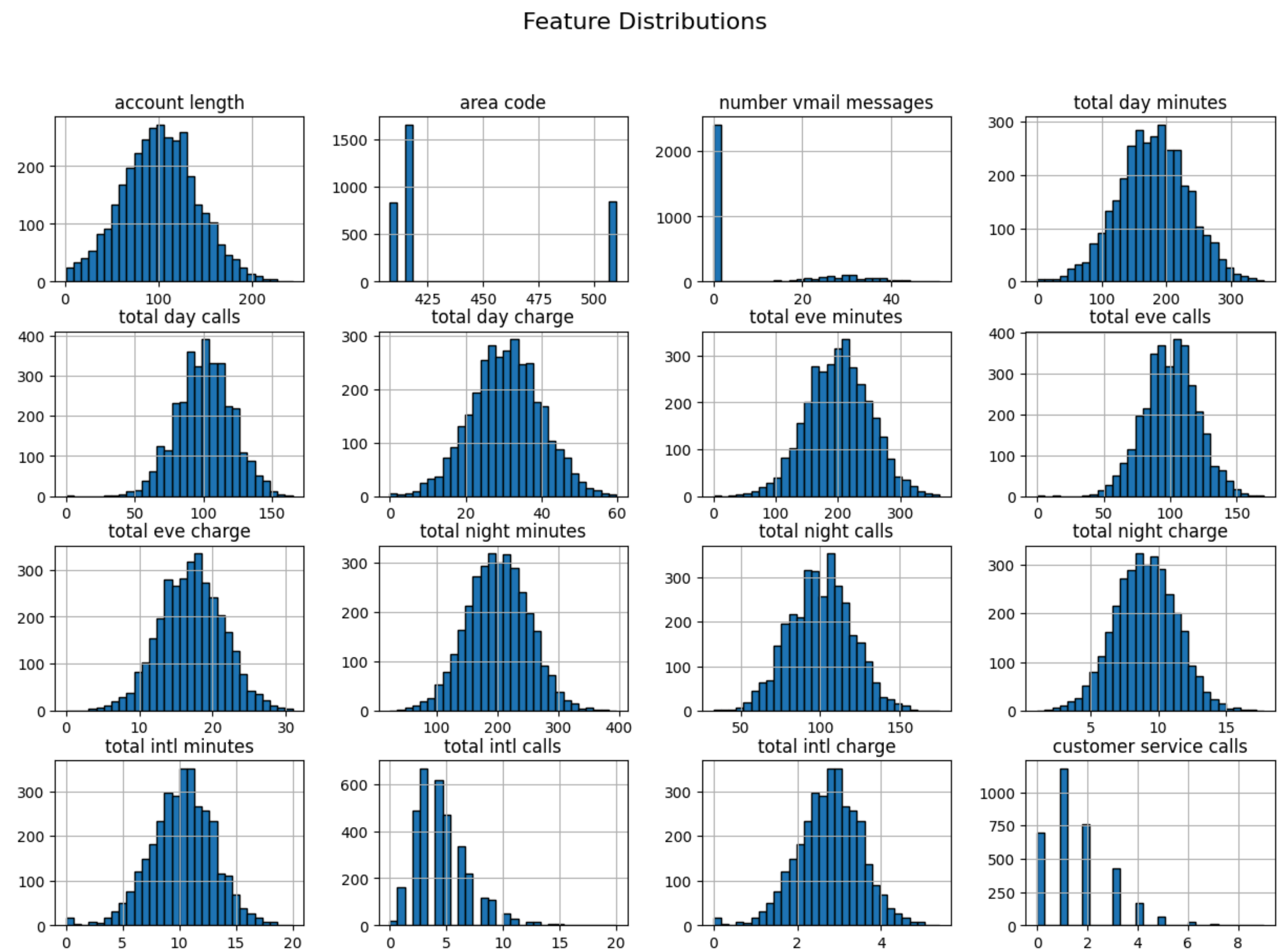

International plan Distribution
3000
2500
2000
1500
1000
500
0
Count
no
yes
international plan


Voice mail plan Distribution
2500
2000
1500
1000
500
0
Count
yes
no
voice mail plan

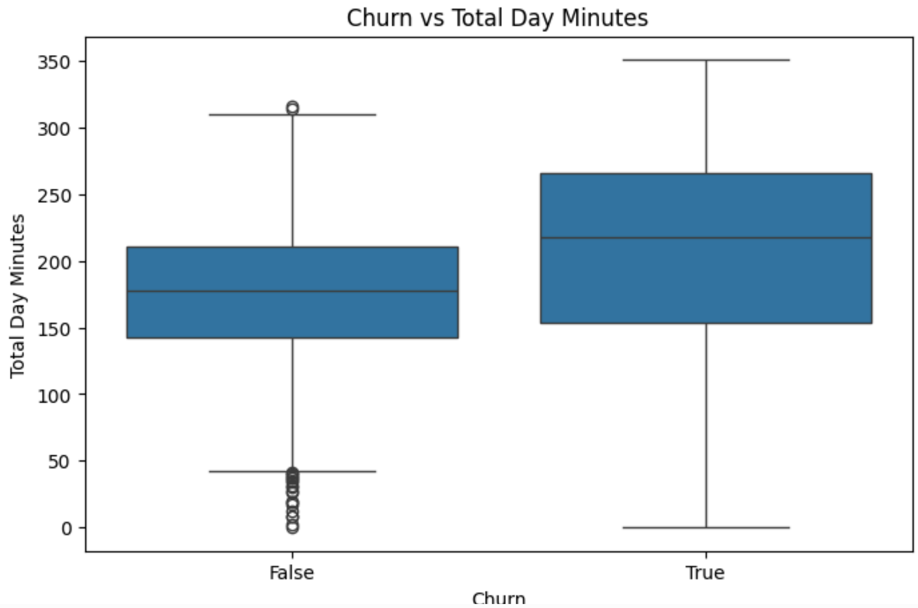

Churn Distribution
2500
2000
1500
1000
Count




```
#Plot distributions of numerical features
df.hist(figsize=(15,10), bins=30, edgecolor='black')
plt.suptitle("Feature Distributions", fontsize=16)
plt.show()
```




```
#churn vs numerical features
plt.figure(figsize=(8, 5))
sns.boxplot(x='churn', y='total day minutes', data=df)
plt.title('Churn vs Total Day Minutes')
plt.xlabel('Churn')
plt.ylabel('Total Day Minutes')
plt.show()
```

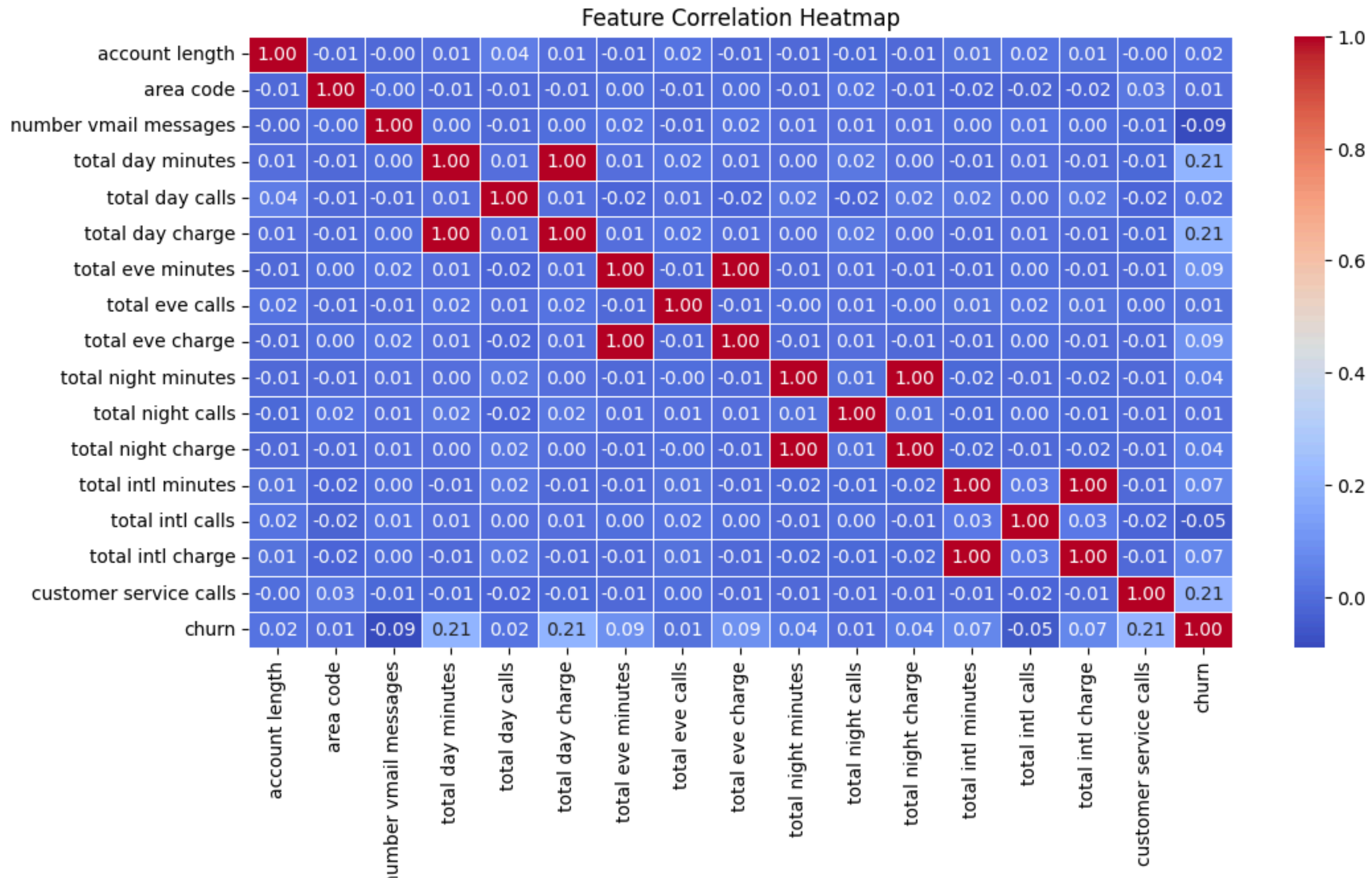


Customers who churn tend to have higher total day minutes than those who stay.

Preprocessing

Correlation analysis

```
import matplotlib.pyplot as plt
import seaborn as sns

#Select only numeric columns
numeric_df = df.select_dtypes(include=['number', 'bool'])

plt.figure(figsize=(12,6))
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```



Customers with higher total day minutes are slightly more likely to churn.

Higher customer service calls correlate with higher churn.

```
#Dropping highly correlated features
df.drop(columns=['total day charge', 'total eve charge', 'total night charge', 'total intl charge'], inplace=True)

df.shape
```

```
(3333, 17)
```

Encoding

```
from sklearn.preprocessing import LabelEncoder

#Converting categorical variables to numeric
encoder = LabelEncoder()
df['international plan'] = encoder.fit_transform(df['international plan'])
df['voice mail plan'] = encoder.fit_transform(df['voice mail plan'])
df['churn'] = df['churn'].astype(int)

#Dropping non-useful categorical features
df.drop(columns=['state', 'phone number'], inplace=True)

df.head()
```

	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	customer service calls	churn
0	128	415	0	1	25	265.1	110	197.4	99	244.7	91	10.0	3	1	0
1	107	415	0	1	26	161.6	123	195.5	103	254.4	103	13.7	3	1	0
2	137	415	0	0	0	243.4	114	121.2	110	162.6	104	12.2	5	0	0
3	84	408	1	0	0	299.4	71	61.9	88	196.9	89	6.6	7	2	0
4	75	415	1	0	0	166.7	113	148.3	122	186.9	121	10.1	3	3	0

Next steps: Generate code with df View recommended plots New interactive sheet

Scaling

```
#standardizing
from sklearn.preprocessing import StandardScaler

#selecting numeric columns an removing target variable
features = df.drop(columns=['churn'])
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

#Converting back to DataFrame
df_scaled = pd.DataFrame(scaled_features, columns=features.columns)
df_scaled['churn'] = df['churn']

df_scaled.head()
```

	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	tota int call
0	0.676489	-0.523603	-0.327580	1.617086	1.234883	1.566767	0.476643	-0.070610	-0.055940	0.866743	-0.465494	-0.085008	-0.60119
1	0.149065	-0.523603	-0.327580	1.617086	1.307948	-0.333738	1.124503	-0.108080	0.144867	1.058571	0.147825	1.240482	-0.60119
2	0.902529	-0.523603	-0.327580	-0.618396	-0.591760	1.168304	0.675985	-1.573383	0.496279	-0.756869	0.198935	0.703121	0.21153
3	-0.428590	-0.688834	3.052685	-0.618396	-0.591760	2.196596	-1.466936	-2.742865	-0.608159	-0.078551	-0.567714	-1.303026	1.02426
4	-0.654629	-0.523603	3.052685	-0.618396	-0.591760	-0.240090	0.626149	-1.038932	1.098699	-0.276311	1.067803	-0.049184	-0.60119

Next steps: Generate code with df_scaled View recommended plots New interactive sheet

Variance Inflation factor

```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(df):
    vif_data = pd.DataFrame()
    vif_data["Feature"] = df.columns
    vif_data["VIF"] = [variance_inflation_factor(df.values, i) for i in range(df.shape[1])]
    return vif_data

# Selecting only numeric features (excluding target variable)
numeric_columns = df.select_dtypes(include=['number']).columns
X = df[numeric_columns].drop(columns=['churn'], errors='ignore')

# Computing VIF
vif_df = calculate_vif(X)

# Displaying VIF values
print(vif_df)
```

```
          Feature        VIF
0  account length   7.298476
1       area code  60.999598
```

```
2      international plan   1.116556
3         voice mail plan  16.449088
4    number vmail messages  16.060222
5       total day minutes  11.469569
6         total day calls  23.588677
7       total eve minutes  15.606325
8         total eve calls  23.725547
9     total night minutes  15.756057
10      total night calls  24.623674
11      total intl minutes  13.664469
12        total intl calls   4.272018
13  customer service calls   2.404659
```

```
#droping high vif features
#Droping features with high multicollinearity
df.drop(columns=['area code', 'voice mail plan', 'number vmail messages',
                 'total day minutes', 'total eve minutes', 'total night minutes',
                 'total intl minutes', 'total day calls', 'total eve calls', 'total night calls'], inplace=True)

#Recalculating VIF
X_new = df.select_dtypes(include=['number']).drop(columns=['churn'], errors='ignore')
vif_df_new = calculate_vif(X_new)

#Displaying updated VIF values
print(vif_df_new)
```

```
                  Feature       VIF
0          account length  3.623377
1      international plan  1.102547
2        total intl calls  3.185682
3  customer service calls  2.110999
```

Train_test split

```
from sklearn.model_selection import train_test_split

#Defining feature columns (excluding the target variable)
sel_columns = ['account length', 'international plan', 'customer service calls', 'total intl calls']
X = df[sel_columns]  # Select features

#Defining target variable
y = df['churn']  # Target variable (churn: 1 = left, 0 = stayed)

#Spliting into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Displaying test labels
y_test.head()
```

	churn
438	0
2674	0
1345	1
1957	0
2148	0

Modeling

Baseline model logistic

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

#Initializing the logistic regression model
```

```
log_model = LogisticRegression(random_state=42)

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

#Making predictions on the test data
y_pred = log_model.predict(X_test)

#Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Baseline Logistic Regression Accuracy: {accuracy:.4f}")

#Displaying classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

#Displaying confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Baseline Logistic Regression Accuracy: 0.8486

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.98      0.92       566
           1       0.50      0.11      0.18       101

    accuracy                           0.85       667
   macro avg       0.68      0.54      0.55       667
weighted avg       0.81      0.85      0.80       667


Confusion Matrix:
[[555  11]
 [ 90  11]]
```

```
#training all models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

#Defining models in a dictionary
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(max_depth=5, random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=5),
    "Naive Bayes": GaussianNB(),
    "Support Vector Machine": SVC(kernel='linear', random_state=42)
}

#Training and evaluating each model
for name, model in models.items():
    print(f"\n🔷 Training & Evaluating: {name} 🔷")

    #Training model
    model.fit(X_train, y_train)

    #Making predictions
    y_pred = model.predict(X_test)

    #Evaluating performance
    accuracy = accuracy_score(y_test, y_pred)
    print(f"✅ Accuracy: {accuracy:.4f}")

    #Displaying classification report
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    #Displaying confusion matrix
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
```

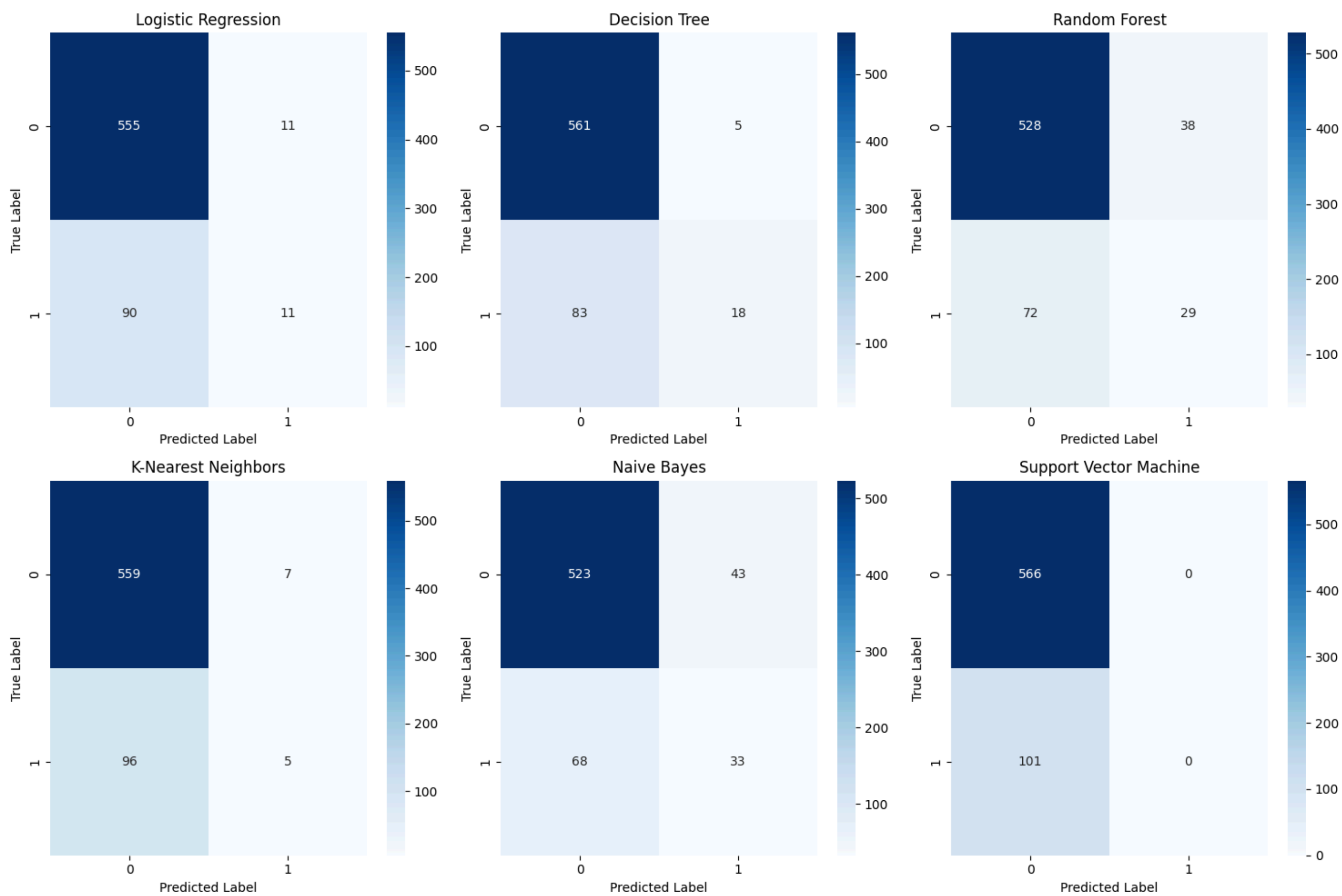
```
print("="*50)
```

```
           0       0.85      0.99      0.92       566
           1       0.42      0.05      0.09       101

    accuracy                           0.85       667
   macro avg       0.64      0.52      0.50       667
weighted avg       0.79      0.85      0.79       667


Confusion Matrix:
[[559   7]
 [ 96   5]]
==================================================

🔷 Training & Evaluating: Naive Bayes 🔷
✅ Accuracy: 0.8336

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       566
           1       0.43      0.33      0.37       101

    accuracy                           0.83       667
   macro avg       0.66      0.63      0.64       667
weighted avg       0.82      0.83      0.82       667


Confusion Matrix:
[[523  43]
 [ 68  33]]
==================================================

🔷 Training & Evaluating: Support Vector Machine 🔷
✅ Accuracy: 0.8486

Classification Report:
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       566
           1       0.00      0.00      0.00       101

    accuracy                           0.85       667
   macro avg       0.42      0.50      0.46       667
weighted avg       0.72      0.85      0.78       667


Confusion Matrix:
[[566   0]
 [101   0]]
==================================================
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Confusion matrix subplots

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

#Defining models
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(max_depth=5, random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=5),
    "Naive Bayes": GaussianNB(),
    "Support Vector Machine": SVC(kernel='linear', random_state=42)
}

#Training models and get confusion matrices
```

```
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.flatten()

for i, (name, model) in enumerate(models.items()):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    #Plotting confusion matrix
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[i])
    axes[i].set_title(f"{name}")
    axes[i].set_xlabel("Predicted Label")
    axes[i].set_ylabel("True Label")

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



Model accuracy comparison

```
import numpy as np

#Defining models again
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(max_depth=5, random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=5),
    "Naive Bayes": GaussianNB(),
    "Support Vector Machine": SVC(kernel='linear', random_state=42)
}

#Storing accuracy scores
```

```
model_names = []
accuracy_scores = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    model_names.append(name)
    accuracy_scores.append(accuracy)

#Plotting accuracy comparison
plt.figure(figsize=(10, 5))
sns.barplot(x=model_names, y=accuracy_scores, palette="viridis")
plt.xticks(rotation=45)
plt.xlabel("Model")
plt.ylabel("Accuracy Score")
plt.title("Model Accuracy Comparison")
plt.show()
```

```
<ipython-input-21-67a6ee230649>:27: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x=model_names, y=accuracy_scores, palette="viridis")
```