Code Implementation & Analysis

Customer Segmentation using Clustering

In today's competitive market, businesses collect large amounts of customer data, but raw data alone does not provide direct value. To make data-driven decisions, it is essential to identify patterns and group customers with similar behaviors, preferences, or demographics.

This project focuses on applying unsupervised machine learning techniques — specifically clustering algorithms — to segment customers into meaningful groups. By understanding these customer segments, businesses can:

- Personalize marketing strategies
- Improve customer experience
- Identify potential high-value customers
- Reduce churn by targeting the right audience with the right message

We will use Python along with libraries such as Pandas, NumPy, Matplotlib, and Scikit-learn to:

- Clean and preprocess the dataset
- Perform exploratory data analysis (EDA) to understand key features
- Apply clustering algorithms (e.g., K-Means)
- Visualize clusters for better interpretability

[6]:
```
# Create config.yaml with absolute path
import yaml

config_data = {
    "data_source": r"C:\Users\ASUS\OneDrive\Desktop\STELLARA DATASET\WA_Fn-UseC_-Telco-Customer-Churn.xlsx"
}

with open("config.yaml", "w") as file:
    yaml.dump(config_data, file)

print("config.yaml created successfully.")
```

```
config.yaml created successfully.
```

[8]:
```
# Load configuration & dataset
import pandas as pd

with open("config.yaml", "r") as file:
    config = yaml.safe_load(file)

data_path = config["data_source"]
df = pd.read_excel(data_path)

print(" Dataset shape before cleaning:", df.shape)
```

```
 Dataset shape before cleaning: (7043, 21)
```

[10]:
```
# Data Cleaning
df = df[df.isnull().mean(axis=1) < 0.7]  # Remove >70% missing
df.fillna(df.mean(numeric_only=True), inplace=True)  # Fill with mean
df.drop_duplicates(inplace=True)  # Remove duplicates
print("Data cleaned. Shape after cleaning:", df.shape)
```

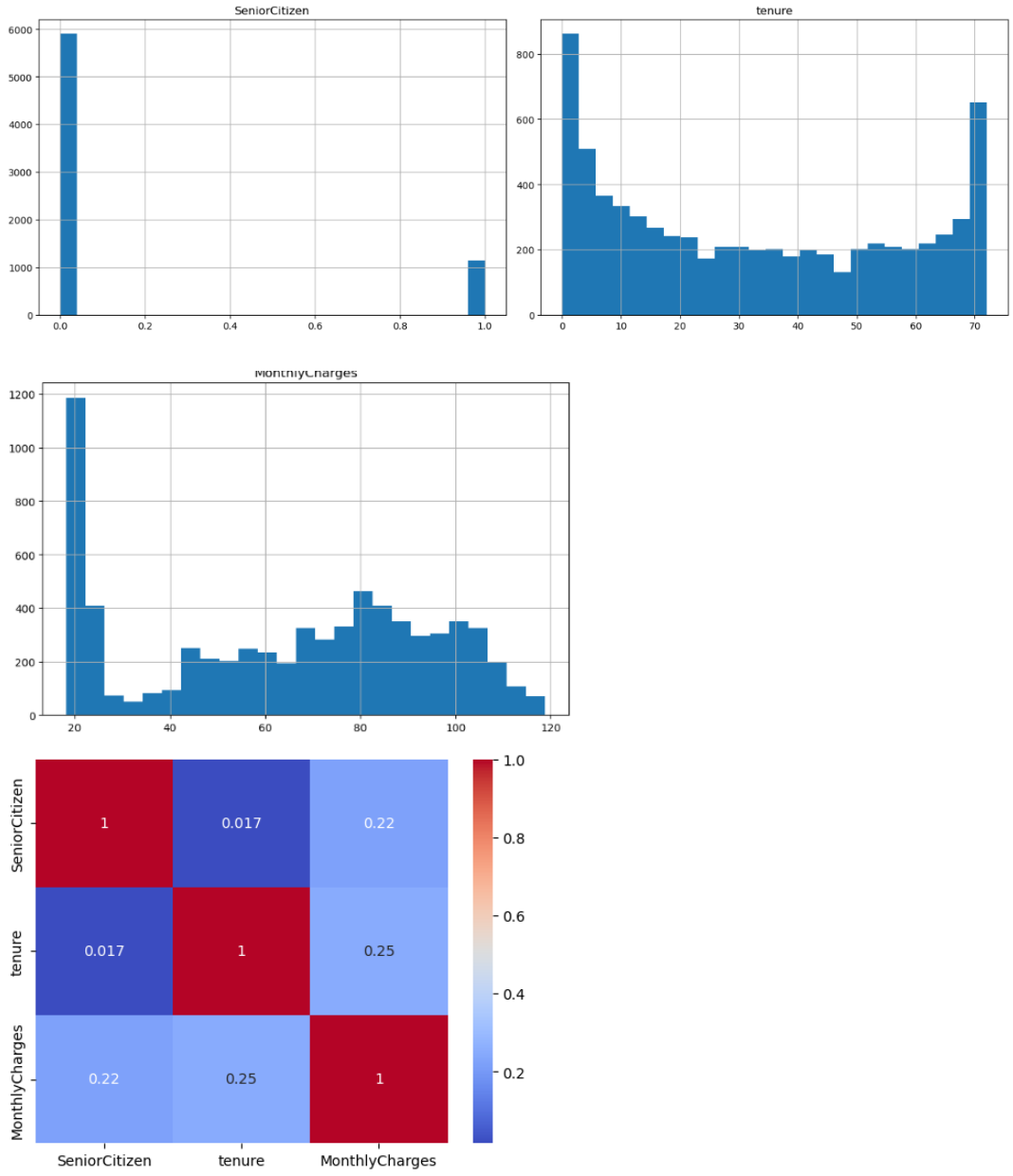
```
Data cleaned. Shape after cleaning: (7043, 21)
```

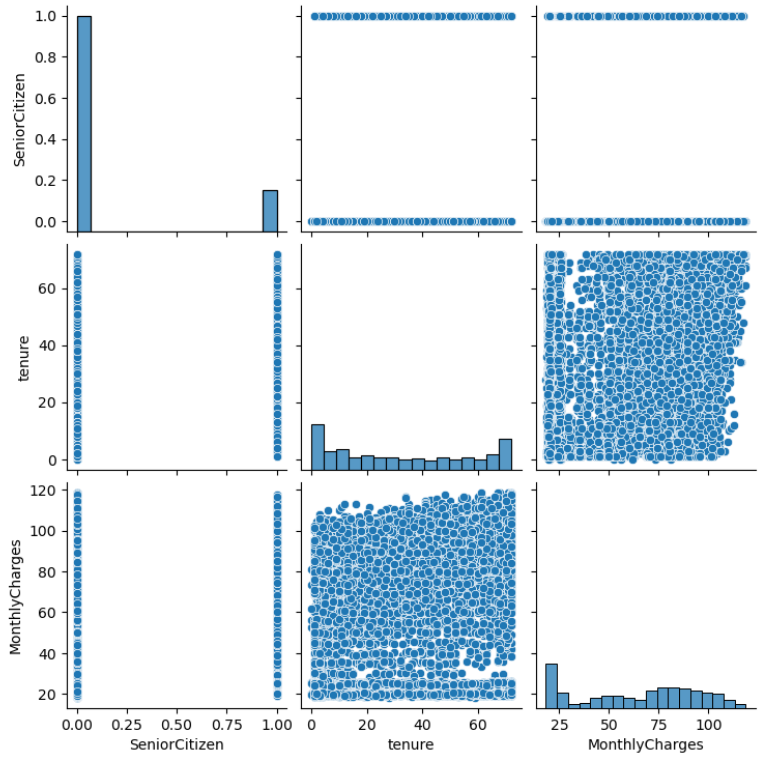
[12]:
```
# EDA (Univariate, Bivariate, Multivariate)
import seaborn as sns
import matplotlib.pyplot as plt

df.hist(bins=25, figsize=(15, 10))
plt.tight_layout()
plt.show()

sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm")
plt.show()

sns.pairplot(df.select_dtypes(include='number').iloc[:, :4])
plt.show()
```





[16]:
```python
# Scaling
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
num_df = df.select_dtypes(include='number')
scaled = scaler.fit_transform(num_df)
scaled_df = pd.DataFrame(scaled, columns=num_df.columns)
df[num_df.columns] = scaled_df
print("Data scaled")
```

```
Data scaled
```

[14]:
```python
# Outlier Removal (IQR)
num_df = df.select_dtypes(include='number')
Q1 = num_df.quantile(0.25)
Q3 = num_df.quantile(0.75)
IQR = Q3 - Q1
df = df[~((num_df < (Q1 - 1.5 * IQR)) | (num_df > (Q3 + 1.5 * IQR))).any(axis=1)]
print("Outliers removed. Shape:", df.shape)
```

```
Outliers removed. Shape: (5901, 21)
```

[18]:
```python
# Feature Selection (RandomForest, AdaBoost, XGBoost)
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder

target_column = "Churn"

# Encode target
y_raw = df[target_column]
if y_raw.dtype == "object":
    le = LabelEncoder()
    y = le.fit_transform(y_raw)
else:
    y = y_raw.astype(int)

# Features
X = df.drop(columns=[target_column])
X = pd.get_dummies(X, drop_first=True)
```

[36]:
```
# Random Forest
rf = RandomForestClassifier()
rf.fit(X, y)
print("Top 5 Features (RandomForest):\n", pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False).head())
```

```
Top 5 Features (RandomForest):
 MonthlyCharges                  0.044491
tenure                           0.041623
Contract_Two year                0.027985
InternetService_Fiber optic      0.027399
PaymentMethod_Electronic check   0.024558
dtype: float64
```

[42]:
```
from sklearn.impute import SimpleImputer
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier

# Create an imputer to handle missing values
imputer = SimpleImputer(strategy='mean')  # You can choose other strategies like 'median', 'most_frequent', etc.

# Fit and transform the data to handle missing values
X_imputed = imputer.fit_transform(X)

# Now we can use X_imputed in the AdaBoost model
ada = AdaBoostClassifier()
ada.fit(X_imputed, y)
print("Top 5 Features (AdaBoost):\n", pd.Series(ada.feature_importances_, index=X.columns).sort_values(ascending=False).head())
```

```
C:\Users\ASUS\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the def
ault) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
Top 5 Features (AdaBoost):
 Contract_Two year                0.08
InternetService_Fiber optic       0.04
TechSupport_No internet service   0.04
Contract_One year                 0.04
OnlineSecurity_Yes                0.02
dtype: float64
```

[44]:
```
# XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb.fit(X, y)
print("Top 5 Features (XGBoost):\n", pd.Series(xgb.feature_importances_, index=X.columns).sort_values(ascending=False).head())
```

```
C:\Users\ASUS\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:40:55] WARNING: C:\actions-runner\_work\xgboo
st\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Top 5 Features (XGBoost):
 Contract_Two year              0.346791
Contract_One year              0.167848
InternetService_Fiber optic    0.092792
OnlineSecurity_Yes             0.031399
InternetService_No             0.023542
dtype: float32
```

[20]:
```
# Clustering (DBSCAN)
from sklearn.cluster import DBSCAN
import pandas as pd
from sklearn.impute import SimpleImputer

# Handle missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean')  # You can choose 'mean', 'median', 'most_frequent', or 'constant'
X_imputed = imputer.fit_transform(X)

# Now apply DBSCAN on the imputed data
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(X_imputed)
df["Cluster"] = clusters
print("DBSCAN Clustering Done. Cluster counts:\n", pd.Series(clusters).value_counts())
```

```
DBSCAN Clustering Done. Cluster counts:
 -1    5901
Name: count, dtype: int64
```

```python
# Train/Test + Evaluation
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("\nClassification Report:\n", classification_report(y_test, y_pred))
print(f"Macro F1 Score: {f1_score(y_test, y_pred, average='macro'):.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.95      0.88       916
           1       0.62      0.26      0.37       265

    accuracy                           0.80      1181
   macro avg       0.72      0.61      0.63      1181
weighted avg       0.77      0.80      0.77      1181

Macro F1 Score: 0.6260
```