Part 1

A1.
Using decision trees, is it possible to predict customer churn based on the following variables: Tenure, MonthlyCharge, Bandwidth_GB_Year, Multiple, Contract, StreamingTV, Area, PaymentMethod, and StreamingMovies?

A2.
A goal of the data analysis is to identify how certain variables influence customer churn. As a result, hopefully, telecommunication companies can evaluate retention strategies and make necessary changes.

Part 2

B1.
Decision trees for classification learn a sequence of if-else questions about features to make predictions. When a classification tree is trained on a dataset, it learns a sequence of if-else questions, where each question pertains to a specific feature and its associated split-point, representing a threshold. Depending on the response, the unlabeled data point moves through the classification tree until reaching an end, which corresponds to the label or final prediction. The algorithm predicts the unknown label by assigning it to the class specified at the end of the tree. For the selected dataset, I anticipate the decision tree will use if-else questions to learn the Churn dataset, and the selected features in the research question (Tenure, MonthlyCharge, Bandwidth_GB_Year, Multiple, Contract, StreamingTV, Area, PaymentMethod, and StreamingMovies) to utilize the responses to predict customer churn.

B2.
One assumption of the decision tree prediction method is that if the tree successfully identifies features and thresholds, it will lead to similar groups, and the predictive power of the model is enhanced due to learning the training data to make predictions on new unseen data. Decision trees work by selecting features and determining split points (or thresholds) that separate the data into homogeneous subsets with hopes of predicting the target variable (or outcome). These splits are based on the conditions of the selected features. The goal of decision trees is to create branches that lead to similar groups to predict the outcome, which means that within each branch, the unlabeled data points share similar characteristics.

B3.
In the process of conducting the analysis, Python was the chosen programming language, and the following packages and libraries were utilized to support various stages of the analysis. The choice of each package and/or library is justified based on its role in enhancing the overall analytical workflow.

- Pandas was used to import and manipulate the dataset. It facilitated the creation of bins for analysis in the data cleaning process, which is a crucial step in understanding the data distribution and determining the appropriate cleaning strategies. During the classification analysis, pandas was used for one-hot encoding, which is necessary in converting categorical variables into a format suitable for analysis.

Jasmine Cooper
D209
Predictive Analysis
Task 2
December 9, 2023
Western Governor's University

- NumPy was employed for array creation and mean calculation which contributed to the preparation of data and for analysis.
- Missingno was used to visualize and identify missing data patterns.
- Seaborn was used to generate an informative heatmap to visualize the correlation between missingness in variables.
- Matplotlib.pyplot was used to generate all visualizations such as bar charts and heatmaps. These visualizations displayed the distribution of missing values to make it easier to make informed decisions during the analysis.
- Fancyimpute / KNN and sklearn.impute / SimpleImputer was used to impute missing data for numerical and non-numerical variables.
- Sklearn.model_selection / train_test_split was used for dividing the dataset into training and testing sets to facilitate model evaluation.
- Sklearn.tree / DecisionTreeClassifier was used to build a decision tree model based on labeled training data.
- Sklearn.model_selection / GridSearchCV was used to identify the optimal values for max_depth (maximum depth of the decision trees), max_features (number of features considered when making a split), and min_samples_leaf (minimum number of samples a leaf node must have to make a prediction).
- Sklearn.metrics / roc_auc_score, classification_report, confusion_matrix, mean_squared_error, and accuracy_score metric tools were used to evaluate the performance of the classification model. These libraries provided insights into the AUC score, classification report, confusion matrix, mean squared error, and overall accuracy.

In summary, the chosen packages and libraries contributed to comprehensive and effective data cleaning, transformation, analysis, feature selection, model construction, and statistical calculations.

In text citations:
("Decision trees for classification", n.d.)
("Preprocessing data", n.d.)

Part 3

C1.
Preprocessing the data by applying one hot encoding was necessary to ensure that nominal categorical variables were properly evaluated in the model. Without proper encoding, there is a risk of the model interpreting these nominal variables as having some ordinal relationship, which could lead to inaccurate and misleading results. Although decision trees are capable of handling categorical variables, the model may not perform optimally when faced with nominal categorical variables that have not been encoded. This preprocessing technique ensures that the decision tree can accurately distinguish between different categories without imposing false hierarchy. In summary, one hot encoding is important for preserving the integrity of the categorical variables within the decision tree model and contributes to the overall effectiveness of the decision tress in making predictions.

C2.

Variable	Classification
Area	Categorical
Churn	Categorical
Contract	Categorical
Multiple	Categorical
StreamingTV	Categorical
StreamingMovies	Categorical
PaymentMethod	Categorical
Tenure	Numerical
MonthlyCharge	Numerical
Bandwidth_GB_Year	Numerical

C3.
The steps used to prepare the data for analysis involved data cleaning and preprocessing.

Data Cleaning Plan:

Step 1: Exploratory Data Analysis
1. Loaded CSV file into Jupyter notebook using pandas library and converted the CSV file into a Data Frame.
2. Used the methods, 'describe()', 'info()', and 'dtypes()' to observe the summary statistics, data types, and nullity, the data types of the data frame.
3. Identified if there was duplicated information using the 'duplicated()' method.
4. Visualized missing data across the entire data frame using missingno and matplotlib libraries, using the 'bar()' and 'show()' methods.
5. Created a correlation matrix using the 'corr()' method, and utilized the correlation matrix to create a heatmap using the seaborn library to understand the correlation between variables missing values.
6. Determined missingness type(s) if applicable.

```
import pandas as pd
import numpy as np
import missingno as msno
churn_raw_data = pd.read_csv ('/Users/jasminemoniquecooper/Downloads/churn_raw_data.csv')
pd.set_option('display.max_columns', None)
churn_raw_data.head(10)

churn_raw_data.dtypes
churn_raw_data.describe()

churn_duplicates = churn_raw_data.duplicated()
churn_raw_data[churn_duplicates]
```

```
churn_raw_data.isna().sum()
#overall summary of missing data in the data frame

#visualize missingness
column_order = churn_raw_data.isnull().sum().sort_values().index
msno.bar(churn_raw_data[column_order])
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt

# Generate a correlation matrix
correlation_matrix = churn_raw_data.corr()

# Set the figure size to make the heatmap larger
plt.figure(figsize=(12, 10))

# Create a heat map using seaborn with customizations
heatmap = sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5,
annot_kws={"size": 8})

# Increase the font size of the color bar (optional)
cbar = heatmap.collections[0].colorbar
cbar.ax.tick_params(labelsize=12)

# Increase the font size of the annotations
for text in heatmap.texts:
   text.set_size(8)

# Display the heat map
plt.show()
```

Step 2: Impute Missing Data & Create Clean Data file

1. Upon completion of investigation, the variables below had missing values:
    a. Children- 7505 (numerical)
    b. Income- 7510 (numerical)
    c. Techie- 7523 (non-numerical)
    d. Age- 7525 (numerical)
    e. Phone- 8974 (non-numerical)
    f. Bandwidth GB Year- 8979 (numerical)
    g. Tech Support- 9009 (non-numerical)
    h. Tenure- 9069 (numerical)
2. By analyzing the summary statistics, I noticed that the population had a minimum value of zero. Since it is highly unlikely population is zero, I replaced all population records with values equal to zero with NaN values utilizing the numpy library and the ‘nan’ and ‘isnan’ methods. This

allows Python to recognize missing values properly prior to imputation. The missing Population values were imputed using the K-Nearest Neighbor technique.

3. The numerical values appear to be missing at random because the missing values could probably be predicted by other observed values in the data set. For example, missing number of children values could be relative to age and/or marital status (e.g., if a customer is an unmarried young adult, they are likely to be within the age range 18-30, and not list the number of children that they have since it might not be applicable). To treat numerical missing values, I decided to use the K-Nearest Neighbor imputation technique because missing values can likely be predicted based on similar observed variables and values.
4. The non-numerical values also appear to be missing at random because the missing values could also probably be predicted by other observed values in the data set. For example, missing Techie values could also be relative to age (e.g., if a customer is between ages 50-90, it is unlikely that they would be considered a Techie). Likewise, missing values for Phone and Tech Support could likely be related to age due to the fast advancement of technology overtime. To impute non-numerical variables' values, I decided to use the Simple Imputation mode technique, and used frequency to fill in missing data. Furthermore, to validate this imputation technique, I analyzed the clean data to ensure that it fit the well-considered estimate, and the clean data portrayed that the group that are not techie's lies between ages 50-90.

```
churn_raw_data.Population[churn_raw_data.Population == 0].count()

#determine if missing income values are related to employment

# Select the rows where income is missing
missing_income = churn_raw_data[churn_raw_data['Income'].isnull()]

# Group the missing income data by employment status
grouped = missing_income.groupby(['Employment'])

# Count the number of missing income values for each employment status
missing_count = grouped.size()

# Calculate the percentage of missing income values for each employment status
percentage_missing = missing_count / churn_raw_data.groupby(['Employment']).size() * 100

# Display the results
print(percentage_missing)
#determine if missing income values are related to employment and age

# Select the rows where income is missing
missing_income = churn_raw_data[churn_raw_data['Income'].isnull()]

# Group the missing income data by employment status
grouped = missing_income.groupby(['Employment', pd.cut(missing_income['Age'], bins=[0, 18, 30, 50,
np.inf])])
```

```
# Count the number of missing income values for each employment status
missing_count = grouped.size()

# Calculate the percentage of missing income values for each employment status
percentage_missing = missing_count / churn_raw_data.groupby(['Employment',
pd.cut(churn_raw_data['Age'], bins=[0, 18, 30, 50, np.inf])]).size() * 100

# Display the results
print(percentage_missing)

#part time employment between ages 0-18 is missing data the most

import pandas as pd

# Select the rows where the 'Techie' column is null
missing_techie = churn_raw_data[churn_raw_data['Techie'].isnull()]

# Define age group bins
age_bins = [18, 30, 50, float('inf')]

# Group the missing 'Techie' data by age
grouped = missing_techie.groupby(pd.cut(missing_techie['Age'], bins=age_bins))

# Count the number of missing 'Techie' values for each age group
missing_count = grouped.size()

# Calculate the percentage of missing 'Techie' values for each age group
total_count_by_age_group = churn_raw_data.groupby(pd.cut(churn_raw_data['Age'],
bins=age_bins)).size()
percentage_missing = (missing_count / total_count_by_age_group) * 100

# Display the results
print(percentage_missing)

import pandas as pd

# Select the rows where 'number of children' is missing
missing_children = churn_raw_data[churn_raw_data['Children'].isnull()]

# Define age group bins
age_bins = [18, 30, 50, float('inf')]  # Adjust the bins as needed

# Group the missing 'number of children' data by marital status and age
grouped = missing_children.groupby(['Marital', pd.cut(missing_children['Age'], bins=age_bins)])

# Count the number of missing 'number of children' values for each marital status and age group
```

```
missing_count = grouped.size()

# Calculate the percentage of missing 'number of children' values for each marital status and age group
total_count_by_group = churn_raw_data.groupby(['Marital', pd.cut(churn_raw_data['Age'],
bins=age_bins)]).size()
percentage_missing = (missing_count / total_count_by_group) * 100

# Display the results
print(percentage_missing)

import numpy as np
churn_raw_data.loc[churn_raw_data.Population == 0, 'Population'] = np.nan
churn_raw_data.Population[np.isnan(churn_raw_data.Population)]

pip install fancyimpute
from fancyimpute import KNN

columns_to_impute = ['Children', 'Income', 'Age', 'Bandwidth_GB_Year', 'Population', 'Tenure']

# Create an instance of the KNN imputer
knn_imputer = KNN()

# Get the column indices of the columns to impute
columns_to_impute_indices = [churn_raw_data.columns.get_loc(col) for col in columns_to_impute]

# Perform imputation on the selected columns
imputed_values = knn_imputer.fit_transform(churn_raw_data.iloc[:, columns_to_impute_indices])

# Assign the imputed values back to the original dataset
churn_raw_data.iloc[:, columns_to_impute_indices] = imputed_values

# Verify if the imputations are in the original dataset
print(churn_raw_data.head())

from sklearn.impute import SimpleImputer

column_to_impute_two = ['Techie', 'Phone', 'TechSupport']

# Create the SimpleImputer object with strategy='most_frequent'
imputer = SimpleImputer(strategy='most_frequent')

# Fit and transform the categorical variables using the imputer
churn_raw_data[column_to_impute_two] =
imputer.fit_transform(churn_raw_data[column_to_impute_two])

import pandas as pd
```

```
# Define age group bins
age_bins = [18, 30, 50, float('inf')]

# Group the data by age
grouped = churn_raw_data.groupby(pd.cut(churn_raw_data['Age'], bins=age_bins))

# Count the number of 'Techie' values for each age group
tech_count = grouped['Techie'].value_counts().unstack(fill_value=0)

# Calculate the percentage of 'Techie' values for each age group
tech_percentage = (tech_count / tech_count.sum(axis=1).values[:, None]) * 100

# Display the results
print(tech_percentage)

#data clean
import matplotlib.pyplot as plt
column_order = churn_raw_data.isnull().sum().sort_values().index
msno.bar(churn_raw_data[column_order])
plt.show()

cleaned_data_two = churn_raw_data
new_name_two = 'churn_cleaned_data'
new_cleaned_data_two = cleaned_data_two.copy()
new_cleaned_data_two.name = new_name_two
new_cleaned_data_two.to_csv('new_cleaned_data_two.csv', index=False)
churn_raw_data.to_csv(r'/Users/jasminemoniquecooper/Downloads/new_cleaned_data_two.csv')
```

Step 3: One hot encoding

1. Drop the unnecessary columns from the clean data frame to prevent creating additional binary variables not needed.
2. Convert all categorical variables to numerical variables using one hot encoding. This process involves creating separate binary variables for each category in a categorical variable. The drop_first argument was used to prevent multicollinearity. For example, the Multiple variable, where only Multiple_Yes is created, and the absence of the phone service is represented by Multiple_Yes = 0 (or Phone_No = 1).
3. Separate binary variables created for Area and PaymentMethod were renamed for proper Python syntax and for ease of use in code.
4. Transform the Churn variable by mapping the ‘yes’ or ‘no’ options to numerical variables.

```
# Drop columns not needed for analysis
columns_to_drop = ['Customer_id', 'Interaction', 'CaseOrder']
churn_clean_data_one = churn_clean_data.drop(columns_to_drop, axis=1)

#one hot encoding
```

```
columns=['City', 'State', 'County', 'Timezone', 'Job', 'Education','Gender', 'Marital','Employment','Area',
'PaymentMethod', 'Techie', 'Contract', 'Port_modem', 'Tablet',
'InternetService','Phone','Multiple','OnlineSecurity', 'OnlineBackup','DeviceProtection',
'TechSupport','StreamingTV', 'StreamingMovies','PaperlessBilling']
churn_clean_encoded = pd.get_dummies(churn_clean_data_one, columns=columns, drop_first = True)

#Map the "Churn" variable to numeric values

churn_clean_encoded["Churn"] = (churn_clean_encoded["Churn"] == 'Yes').astype(int)

# Rename columns for proper Python syntax
column_name_mapping = {
   "PaymentMethod_Bank Transfer(automatic)": "Bank_Transfer",
   "PaymentMethod_Credit Card (automatic)": "Credit_Card",
   "PaymentMethod_Electronic Check": "Electronic_Check",
   "PaymentMethod_Mailed Check": "Mailed_Check",
   "Area_Suburban": "Suburban",
   "Area_Urban": "Urban",
   "Area_Rural": "Rural"
}

churn_clean_encoded.rename(columns=column_name_mapping, inplace=True)

churn_clean_encoded.head(10)
```

Code:
See code/script attached and below:

Please see Churn Clean Data.ipynb attached for this code.

C4.

Please see ‘new_cleaned_data_two.csv’ file attached to submission

In text citations:
(“Completeness”, n.d.)
(“Importing flat files using pandas”, n.d.)
(“Is data missing at random?”, n.d.)
(“Mean, median, & mode imputations”, n.d.)
(“Imputing using facncyimpute”, n.d.)
(“Measures of center”, n.d.)
(“Measures of spread”, n.d.)
("GeeksforGeeks," n.d.)
(“Centering and scaling”, n.d.)
(“Preprocessing data”, n.d.)

Part 4

D1.
Please see the following files attached to the submission: X_train_dt.csv, X_test_dt.csv, y_train_dt.csv, y_test_dt.csv

D2.
The analysis included hyperparameter tuning and model performance metrics. To begin my analysis, I utilized GridSearchCV to identify the optimal values for max_depth (maximum depth of the decision trees), max_features (number of features considered when making a split), and min_samples_leaf (minimum number of samples a leaf node must have to make a prediction). This method aims to search through the grid for optimal values based on a manual set of grids discrete hyperparameter values and the metric selected to score model performance. As a result, the search provided the optimal values and the corresponding cross validate score, reflecting the model's best performance during the cross-validation process. In my model, the outputs were max_depth = 6, max_features = 5, min_samples_leaf = 20, and an accuracy score of 85%.

After hyperparameter tuning, I conducted an analysis that encompassed the accuracy score, precision, recall, and F1-score as displayed in the classification report, along with the AUC. Observations from the classification report are as follows:

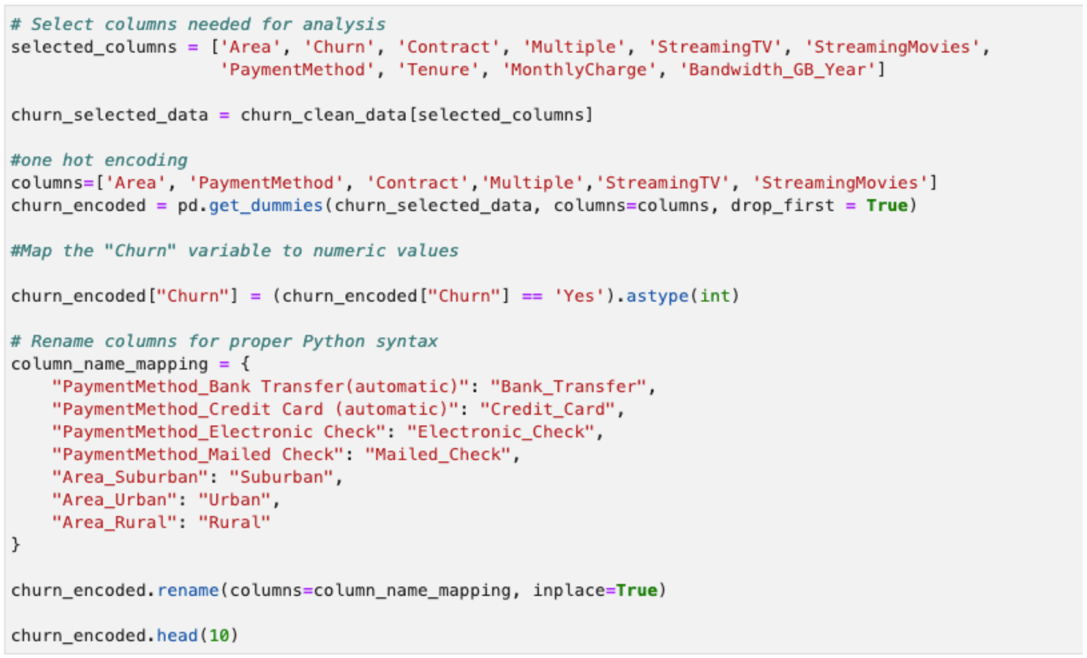
- For class 0 (customer did not churn):
    - Precision: 87% of the predicted positive instances were true positives.
    - Recall: 92% of the actual positive instances were predicted correctly.
    - F1-score: 89%, which indicates that there is a good balance between precision and recall in the model for class 0. This suggests that there is a relatively low number of false positives and false negatives, which implies that the model has good performance for class 0.
    - Support: 1470 data points that represent the number of actual instances of class 0 in the dataset.
- For class 1 (customer did churn)
    - Precision: 73% of the predicted positive instances were true positives.
    - Recall: 62% of the actual positive instances were predicted correctly.
    - F1-score: 67%, which indicates that there is a satisfactory balance between precision and recall in the model for class 1. In comparison to class 0, the F1-score highlights a challenge in correctly identifying the instances of churn.
    - Support: 530 data points that represent the number of actual instances of class 1 in the dataset.

Additionally, using the best hyperparameters the model correctly classified 84% of all instances, indicating the model with selected features performs well.

Given the robust classification metrics observed for both classes, I examined the AUC to obtain a comprehensive understanding of the model's overall performance. As a result, the AUC metric, calculated with selected features, yielded a value of 91%. This reinforces the hypothesis that the model excels in its performance and demonstrates proficiency in accurately classifying potential outcomes.

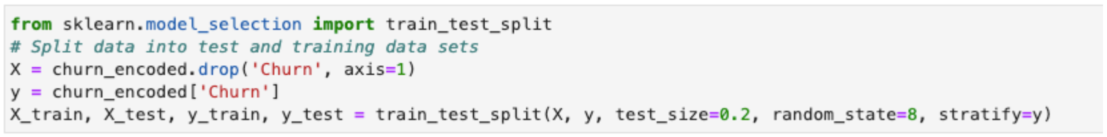
D3.

Code for one hot encoding:



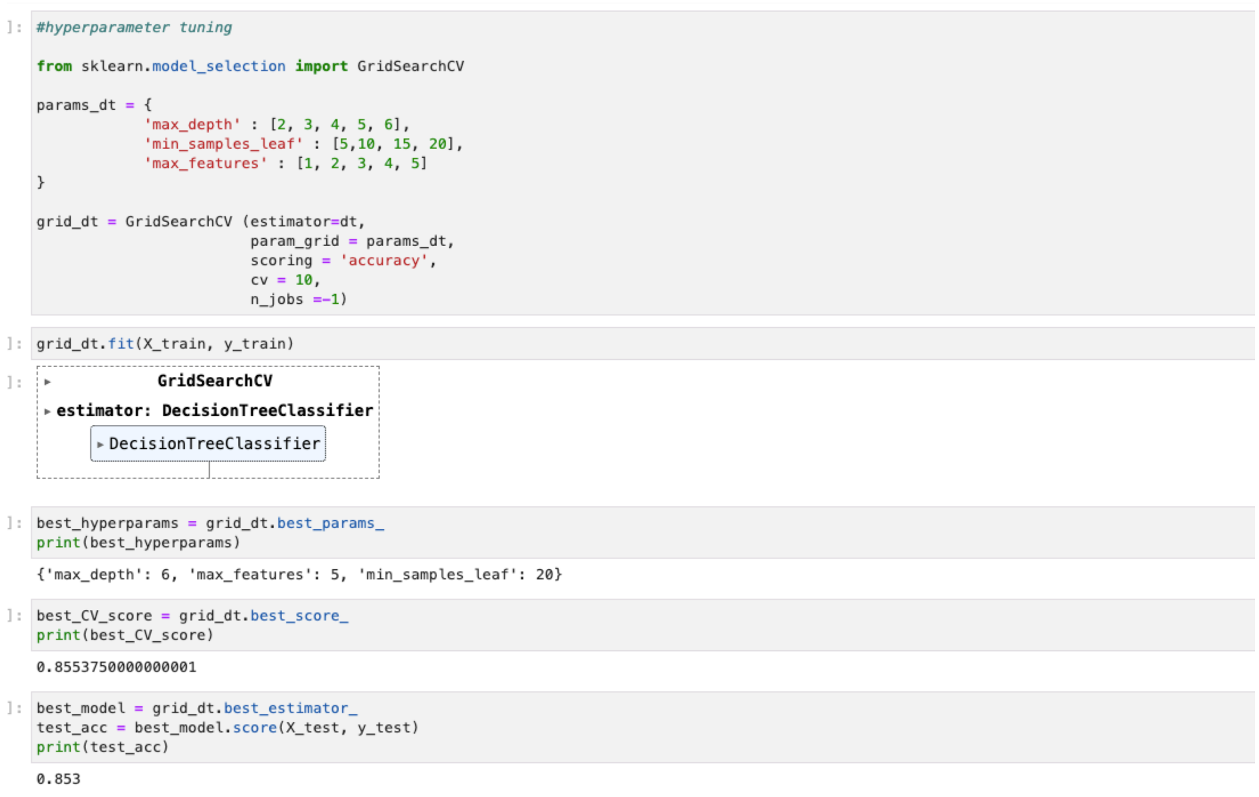
```
# Select columns needed for analysis
selected_columns = ['Area', 'Churn', 'Contract', 'Multiple', 'StreamingTV', 'StreamingMovies',
                    'PaymentMethod', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year']

churn_selected_data = churn_clean_data[selected_columns]

#one hot encoding
columns=['Area', 'PaymentMethod', 'Contract','Multiple','StreamingTV', 'StreamingMovies']
churn_encoded = pd.get_dummies(churn_selected_data, columns=columns, drop_first = True)

#Map the "Churn" variable to numeric values

churn_encoded["Churn"] = (churn_encoded["Churn"] == 'Yes').astype(int)

# Rename columns for proper Python syntax
column_name_mapping = {
    "PaymentMethod_Bank Transfer(automatic)": "Bank_Transfer",
    "PaymentMethod_Credit Card (automatic)": "Credit_Card",
    "PaymentMethod_Electronic Check": "Electronic_Check",
    "PaymentMethod_Mailed Check": "Mailed_Check",
    "Area_Suburban": "Suburban",
    "Area_Urban": "Urban",
    "Area_Rural": "Rural"
}

churn_encoded.rename(columns=column_name_mapping, inplace=True)

churn_encoded.head(10)
```

Code for test / train data split:



```
from sklearn.model_selection import train_test_split
# Split data into test and training data sets
X = churn_encoded.drop('Churn', axis=1)
y = churn_encoded['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=8, stratify=y)
```

Code for building the Decision Tree model (model fitting, model predictions, model evaluation):



```python
#hyperparameter tuning

from sklearn.model_selection import GridSearchCV

params_dt = {
            'max_depth' : [2, 3, 4, 5, 6],
            'min_samples_leaf' : [5,10, 15, 20],
            'max_features' : [1, 2, 3, 4, 5]
}

grid_dt = GridSearchCV (estimator=dt,
                        param_grid = params_dt,
                        scoring = 'accuracy',
                        cv = 10,
                        n_jobs =-1)
```

```python
grid_dt.fit(X_train, y_train)
```

```
▸            GridSearchCV
▸ estimator: DecisionTreeClassifier
      ▸ DecisionTreeClassifier
```

```python
best_hyperparams = grid_dt.best_params_
print(best_hyperparams)
```

```
{'max_depth': 6, 'max_features': 5, 'min_samples_leaf': 20}
```

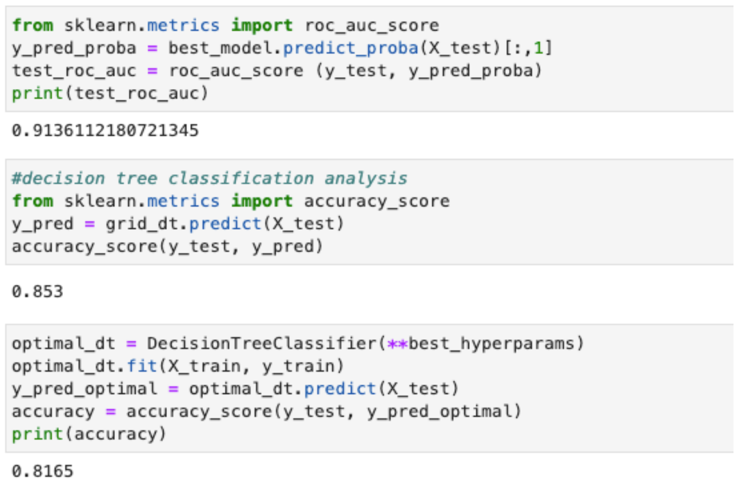
```python
best_CV_score = grid_dt.best_score_
print(best_CV_score)
```

```
0.8553750000000001
```

```python
best_model = grid_dt.best_estimator_
test_acc = best_model.score(X_test, y_test)
print(test_acc)
```

```
0.853
```

Jasmine Cooper
D209
Predictive Analysis
Task 2
December 9, 2023
Western Governor's University



```
from sklearn.metrics import roc_auc_score
y_pred_proba = best_model.predict_proba(X_test)[:,1]
test_roc_auc = roc_auc_score (y_test, y_pred_proba)
print(test_roc_auc)
```

```
0.9136112180721345
```

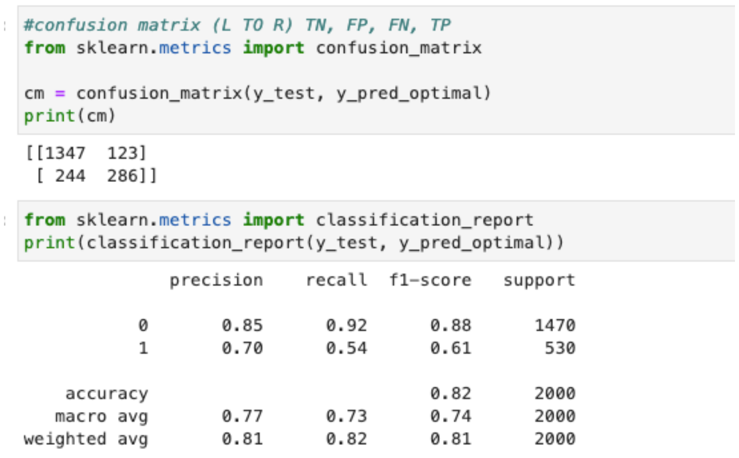
```
#decision tree classification analysis
from sklearn.metrics import accuracy_score
y_pred = grid_dt.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.853
```

```
optimal_dt = DecisionTreeClassifier(**best_hyperparams)
optimal_dt.fit(X_train, y_train)
y_pred_optimal = optimal_dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_optimal)
print(accuracy)
```

```
0.8165
```

Code for classification analysis (confusion matrix and classification report):



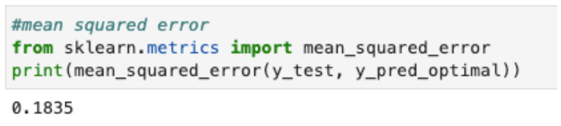
```
#confusion matrix (L TO R) TN, FP, FN, TP
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_optimal)
print(cm)
```

```
[[1347  123]
 [ 244  286]]
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_optimal))
```

```
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      1470
           1       0.70      0.54      0.61       530

    accuracy                           0.82      2000
   macro avg       0.77      0.73      0.74      2000
weighted avg       0.81      0.82      0.81      2000
```

Code for mean squared error analysis:



```
#mean squared error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred_optimal))
```

```
0.1835
```

In text citations:
("Measuring Model performance", n.d.)

Jasmine Cooper
D209
Predictive Analysis
Task 2
December 9, 2023
Western Governor's University

("Hyperparameter tuning", n.d.)
("How good is your model?", n.d.)
("Classification metrics", n.d.)
Tuning a CART's Hyperparameters
Decision tree for classification
Introduction to hyperparameter tuning

Part 5

E1.
To assess the performance and accuracy of the decision tree model, I utilized the accuracy and the mean squared error (MSE). The mean squared error measures how close the predictions are to the actual data. A smaller mean squared error indicates a good model, since there is less distance between the actual data points and predictions. It conveys that the model's predictions are well calibrated and effectively stabilizes class imbalance. In this context, I observed a mean squared error of approximately 16%, which suggests that the model has exceptional performance. Additionally, accuracy, which is the overall ability of the model to correctly predict the correct classification, was approximately 84%, which indicates robust performance.

E2.
The results of my decision trees analysis reveal the model's proficiency in classifying outcomes. When investigating how the variables related to the research question contribute to customer churn, it was intriguing to discover: Bandwidth_GB_Year stood out as the most influential variable contributing to customer churn. In sequential order, the following variables were identified as most impactful: MonthlyCharge, Contract_TwoYear, StreamingTV_Yes, Contract_One year, StreamingMovies_Yes, Tenure, and Multiple_Yes. This indicates that the average amount of data used in a year is the primary factor influencing a customer's inclination to churn. Subsequently, MonthlyCharge, which reflects the cost associated with data usage, follows in significance. It is logical to infer that the combination of these two variables can sway over a customer's decision to either stay or leave a company. The implications of these findings suggest the model not only predicts customer churn well, but also provides insights into which features are most crucial for retention policies aimed at reducing customer churn. Telecommunication companies can leverage this information to refine their strategies and focus on key variables that play a pivotal role in customer retention.

E3.
One limitation of my data analysis lies in the fact that the research question concentrated on only a subset of variables rather than considering the entire variable selection. Expanding the scope of the research question to encompass all variables would have allowed for a more comprehensive exploration of how various factors could impact customer churn. However, as observed in previous exercises, models without proper feature selection may exhibit inefficiencies. I assume this inefficiency could stem from an overload of information for the model to process. Including all variables without feature selection might lead to a model struggling to handle an overwhelming amount of data. In summary, the data analysis could have yielded more insightful results if it had initially included all variables.

E4.

Based on the findings of the feature importance analysis, I recommend that telecommunication companies examine the impact of Bandwidth_GB_Year on customer churn. The companies should conduct an in-depth analysis to understand how the amount of data usage influences the likelihood of a customer churning, and create policies related to usage. For instance, the companies could introduce contractual agreements specifying a certain data allowance for customers within the contract period, correlating with reduced churn rates. For example, if customers exhibit lower churn rates when maintaining an average annual data usage of 300 GB, then companies should introduce contracts with a minimum data usage requirement of 300 GB. While the analysis might yield varying results, it is likely that customers with higher data usage are less prone to churning. Implementing such policies and contractual agreements could optimize pricing strategies, potentially boosting profits as telecommunication companies effectively increase their sales to customers through increased data usage.

In text citations:
("Measuring model performance", n.d.)
("Classification metrics", n.d.)
("Accuracy metrics: regression models", n.d.)

<u>Part 5</u>

H.
Citations for code:

DataCamp. (n.d.). Completeness [Video file]. Retrieved from https://campus.datacamp.com/courses/cleaning-data-in-python/advanced-data-problems-3?ex=8

DataCamp. (n.d.). Importing flat files using pandas [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-importing-data-in-python/introduction-and-flat-files-1?ex=15

DataCamp. (n.d.). Mean, median, & mode imputations [Video file]. Retrieved from https://campus.datacamp.com/courses/dealing-with-missing-data-in-python/imputation-techniques?ex=1

DataCamp. (n.d.). Imputing using fancyimpute [Video file]. Retrieved from https://campus.datacamp.com/courses/dealing-with-missing-data-in-python/advanced-imputation-techniques?ex=1

Lekhana_Ganji. (2023, April 18). Machine Learning - One Hot Encoding of Datasets in Python. GeeksforGeeks. https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/

DataCamp. (n.d.). Working with model objects [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-regression-with-statsmodels-in-python/predictions-and-model-objects-2?ex=5

GeeksforGeeks. (n.d.). How to Rename Columns in Pandas DataFrame. Retrieved from https://www.geeksforgeeks.org/how-to-rename-columns-in-pandas-dataframe/

DataCamp. (n.d.). Preprocessing data [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/preprocessing-and-pipelines?ex=1

DataCamp. (n.d.). How good is your model? [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/fine-tuning-your-model?ex=1

DataCamp. (n.d.). Hyperparameter tuning [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/fine-tuning-your-model?ex=9

DataCamp. (n.d.). Classification metrics [Video file]. Retrieved from https://campus.datacamp.com/courses/model-validation-in-python/validation-basics?ex=9

DataCamp. (n.d.). Measuring model performance [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/classification?ex=9

DataCamp. (n.d.). Decision tree for classification [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/classification-and-regression-trees?ex=1

DataCamp. (n.d.). Accuracy metrics: regression models [Video file]. Retrieved from https://campus.datacamp.com/courses/model-validation-in-python/validation-basics?ex=5

I.
Citations for content:

DataCamp. (n.d.). The classification challenge [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/classification?ex=6

DataCamp. (n.d.). How good is your model? [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/fine-tuning-your-model?ex=1

DataCamp. (n.d.). Area under the ROC curve [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/fine-tuning-your-model?ex=7

DataCamp. (n.d.). Hyperparameter tuning [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/fine-tuning-your-model?ex=9

DataCamp. (n.d.). Classification metrics [Video file]. Retrieved from https://campus.datacamp.com/courses/model-validation-in-python/validation-basics?ex=9

DataCamp. (n.d.). Measuring model performance [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-scikit-learn/classification?ex=9

DataCamp. (n.d.). Decision tree for classification [Video file]. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/classification-and-regression-trees?ex=1

DataCamp. (n.d.). Accuracy metrics: regression models [Video file]. Retrieved from https://campus.datacamp.com/courses/model-validation-in-python/validation-basics?ex=5