Project 2
Data Science Analytical Workflow

Overview

The objective of this project is to simulate a plausible data science workflow.

Rules

- In your submission, include all your code and the output within a single Jupyter Notebook.
- Your code must show output if the step requires it. For example, if the step is to generate a plot, if the plot is not generated it will be assumed it has an error in the code that is preventing it from being generated.
- The code within the Jupyter Notebook must reflect the order of intended execution (as reflected in the below requirements).
- Follow the principle to limit hard-coding of values
    - Your code should still function as intended if data is added/deleted from the dataset
- Please put a comment in each cell with the task the cell relates to.
- Not all tasks are equally weighted in terms of points.
- At the top of the Notebook, list the names of the students involved in completing the work.
- **Place the code for each sub-task in a separate Notebook cell.**

Note: the sections of this document map to the sections and data science workflow as presented in **L18 - Data Preparation**.

General Advice

When dealing with larger datasets it is easy to make 'undiscovered' errors. To reduce the risk of such errors creeping in, you may find the following techniques of some value:

- Often confirm the shape of your evolving DataFrame to see if the shape aligns with your expectations.

```
df.shape
```

- A common issue is confusion over datatypes. To confirm if the datatypes align with your expectations you may wish to run the following diagnostics:

```
df['ColX'].value_counts()

df['ColX'].unique()

df.dtypes
```

- Students may forget that datatype conversion or subsetting create copies rather than update the data structure/data values. To update a df (e.g., dropping a field), use assignment or modify your df *inplace*. If can be helped, your preference should be to update an existing df rather than create an additional one.
- Always be on the lookout for the effect of missing values, nulls, funny values, as all the fixes for these aspects of the data may not be explicitly listed below. However, you will be expected to fix such issues if they significantly affect the quality of your results.

1. Analytical Objective

Online services, such as internet access, streaming television, and premium email services, are usually provided in the form of subscription services – i.e., subscribers are required to pay a recurring fee to continue receiving the service. Accordingly, an important metric to many service providers is the number of subscribers gained, and lost, over time. For example, it was reported that AT&T lost 1.1 million streaming TV customers in Q2, 2019, alone.

The term used to describe subscribers stopping their subscriptions is churn. Churn quantifies the number (percentage) of customers who have unsubscribed or canceled their service contract. It is expensive to win back customers once lost, and therefore it is important for a company to understand the factors contributing to customer churn, as well as customer loyalty.

The analytical objective of this project is to identify any interesting patterns within our customer database in relation to churn/loyalty behavior. In this respect, we will attempt to create a prediction model of customer churn (i.e., a customer's propensity to stop their subscription, based on certain attribute values) using the Binary Logistic Regression method. Contrary to its name, a Binary Logistic Regression model is used for classification.

At this stage, we are not seeking to explain the 'why' behind customer behavior, as assertions of a causal nature require a more in-depth investigation, usually involving the use of multiple statistical techniques and using data from numerous sources (e.g., external market research, social medial analysis, A/B testing, and promotional experiments). So, in every sense, this is an 'exploratory' study.

2. Get the Data

This project will use the **Telcom Customer Churn Dataset** (which is available on Canvas). The raw dataset contains more than 7,000 entries.

Table 1 below provides a description of the fields within the Telcom Customer Churn dataset.

Field	Description
customerID	Customer ID
gender	Whether the customer is a male or a female
SeniorCitizen	Whether the customer is a senior citizen or not (1, 0)
Partner	Whether the customer has a partner or not (Yes, No)
Dependents	Whether the customer has dependents or not (Yes, No)
Tenure	Number of months the customer has stayed with the company
PhoneService	Whether the customer has a phone service or not (Yes, No)
MultipleLines	Whether the customer has multiple lines or not (Yes, No, No phone service)
InternetService	Customer's internet service provider (DSL, Fiber optic, No)
OnlineSecurity	Whether the customer has online security or not (Yes, No, No internet service)
OnlineBackup	Whether the customer has online backup or not (Yes, No, No internet service)
DeviceProtection	Whether the customer has device protection or not (Yes, No, No internet service)
TechSupport	Whether the customer has tech support or not (Yes, No, No internet service)
StreamingTV	Whether the customer has streaming TV or not (Yes, No, No internet service)
StreamingMovies	Whether the customer has streaming movies or not (Yes, No, No internet service)
Contract	The contract term of the customer (Month-to-month, One year, Two year)
PaperlessBilling	Whether the customer has paperless billing or not (Yes, No)
PaymentMethod	The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
MonthlyCharges	The amount charged to the customer monthly
TotalCharges	The total amount charged to the customer
StreamingTV	Whether the customer has streaming TV or not (Yes, No, No internet service)
Churn	Whether the customer churned or not (Yes or No)

Table 1: Telcom Customer Churn Dataset Metadata

The dataset, therefore, contains information about:

- Customer behavior (churned, or not).
- Services that each customer has signed up for — phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies.
- Customer account information — how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges.
- Demographic information about customers — gender, age range, and if they have partners and dependents.

Task #2a: Load the Telcom Customer Churn dataset into a DataFrame.

Task #2b: Confirm the import by executing the head method on your DataFrame.

Task #2c: Determine the 'shape' of your DataFrame. Then list the fields with their datatypes using the `info()` method. Note: you may find it valuable to query the shape of your DataFrame after each operation to ensure it is per expectations.

3. Data Cleaning

Note: Chapter 7 reviews some important data-cleaning techniques.

Do NOT use print() within the section.

Task #3a: A good first step is to review the number of missing values within the dataset:

- NaN in numeric arrays
- None or NaN in object arrays
- NaT in datetime-like fields.

Display the number of missing values for each field with the df.

Task #3b: Some cells within the Churn status field appear to be empty but actually contain whitespaces – we will need to remove these whitespaces. You can remove the whitespaces by executing across the Churn status field the following command (change *df* to the name of your DataFrame):

```
df['Churn'].replace(r'^\s*$', np.nan, regex=True, inplace=True)
```

Note: the *inplace* argument is needed to modify the DataFrame.

Task #3c: Some records do not have a Churn status value recorded. Display all these records using a boolean_vector_expression. However, restrict the fields displayed to customerID, and Churn.

Task #3d: Use `value_counts()` to display the number of No and Yes values across the Churn status values.

Now, change the Churn status for the records without a Churn status to be 'No'.

Again, use `value_counts()` to display the number of No and Yes values across the Churn status values. The number of 'No' values should have increased.

Task #3e: Display any rows with more than one missing value.

Requirement: perform this step twice using these two techniques:

- first, using the double-bracket approach (and present the output as a DataFrame), &
- second, reperform it using `.loc` slicing.

Hint: the boolean_vector_expression will need to feature method chaining.

Task #3f: Now, remove from the DataFrame any rows with more than one missing value – i.e., the rows you just displayed.

First, display the df shape statistic.

Remove the rows using `df.drop()`. Pass to `drop()` a list of the index values of the rows to be dropped.

Now, redisplay the df shape statistic to confirm the reduction in the number of records.

Note: An alternate approach is to use `dropna()` -- but this approach focuses on 'keeping' rows containing a certain number of non-NaN (rather than filtering rows with a certain number of missing values). There is a workaround - use the *thresh* parameter with a calculated number of columns (i.e., `len(df.columns) - 1`). However, for this step please use the passing indexes approach.

Task #3g: Eliminate any duplicate records. Update the shape statistic.

Task #3h: Replace all occurrences of 'No internet service' with NaN.

Task #3i: Recode 'Bank transfer (automatic)' and 'Credit card (automatic)' values to become 'Automatic'.

Task #3j: Create an explicit row index for your DataFrame using the customerID field. Name this index **CustomerID**.

Task #3k: Identify any potential outliers in the MonthlyCharges. Generate a visualization using the seaborn boxplot: either `seaborn.boxplot()` or `df.boxplot()`. Use Churn as the y argument.

Also, list the 10 largest monthly charges (use `df.nlargest()`).

Task #3l: Change any identified outliers in the distribution of monthly charges (MonthlyCharges) to match a designated ceiling amount (i.e., the threshold value you have chosen as differentiating between within-range and outlier values). The chosen threshold value (i.e., the 'ceiling amount') is up to your judgment (but some values must be identified as outliers).

Now, list the 15 largest monthly charges to verify your enacted changes are correct.

Task #3m: Re-generate the earlier visualization of the distribution of MonthlyCharges using the seaborn boxplot: either `seaborn.boxplot()` or `df.boxplot()`. This time, the 'boxiness' of the boxplot should be more apparent after the outliers have been removed.

Task #3n: Validating is the activity that surfaces data quality and consistency issues or verifies that they have been properly addressed by applied transformations. Validations should be conducted along multiple dimensions. At a minimum, assessing whether the values of an attribute/field adhere to syntactic constraints (e.g., boolean fields encoded as 'true'/'false' as opposed to '1'/'0' or 'T'/'F') as well as distributional

constraints (e.g., birth dates in a customer database should be close to being uniformly distributed over months of the year). Additional validations might involve cross-attribute/field checks like ensuring all negative bank transactions have the appropriate transaction type (e.g., 'withdrawal', 'bill pay', or 'check'). Food evaluations are similarly multi-dimensional – checking things like temperature, taste, appearance, and texture.

Again, update the df shape statistic.

Perform a cross-check of whether *Tenure* multiplied by MonthlyCharges is equal TotalCharges. If the cross-check fails to be within a 15% range of the recorded TotalCharges then drop the entire record.

Again, update the df shape statistic.

Something to consider: before doing anything, you may want to check the data type of TotalCharges to confirm it is per expectations. If modifying the `dtype`, you may also need 'coerce' the data to the new type.

4. Data Exploration

Note: for each visualization, you can take liberty in styling selections, but make sure each visualization has appropriate axis labels and a legend (although not all visualizations will need a legend). Some of the tasks ask for an explicit type of plot, while others are open-ended.

Task #4a: Generate a seaborn **countplot** for Churn status.

Task #4b: Generate a seaborn **barplot** comparing MonthlyCharges against the categories of InternetService.

Task #4c: Generate a seaborn **violinplot** comparing MonthlyCharges against the categories of InternetService. For this plot, also 'split' the categories of InternetService by Churn status (Yes, or No).

Task #4d: Generate a seaborn **stripplot** (with the plots presented horizontally) in which TotalCharges is the x-axis and Tenure (binned into 8 equal categorial quantile intervals) is the y-axis. Label the Tenure bins: B1 through B8. (Hint: use `cut()` / `qcut()` to generate the categories). To reduce the number of points plotted, use a sample of records within the DataFrame to generate the **stripplot**. You can sample the records using this command: `dfS = df.sample(frac=0.1)`

Task #4e: Add a new field to your DataFrame that contains the square root of MonthlyCharges for each respective row. To perform this operation, you will need to use the `sqrt()` function from the math library: `from math import sqrt`. Creating the square root values requires an element-wise operation across the MonthlyCharges field. Enact this requirement by using `apply()` on a series.

Note: adding field ➔

Task #4f: Generate a seaborn **kdeplot** comparing the square root of MonthlyCharges to the length of Tenure.

Note: your computer may need to take a while to generate this plot.

Task #4g: Generate a seaborn **histogram** with 30 bars (also known as bins, breaks, or buckets) showing the count distribution of MonthlyCharges. Make the width of bins = 1.

Task #4h: Generate seaborn **boxplots** showing MonthlyCharges against these fields: SeniorCitizen, and InternetService.

You can either include multiple plots into a single visualization or have two separate plots.

Task #4i: Create a trellis of comparative seaborn plots, comparing the distribution of monthly MonthlyCharges across each combination of contract types (one year, two year, etc.) and the types of internet service provider (DSL, etc.).

5. Data Transformation

Task #5a: Generate a seaborn **boxplot** comparing Tenure and MonthlyCharges.

As these variables are clearly on different scales, you will need to first normalize the values in both variables (use the min-max normalization approach).

To confirm your normalization was performed correctly, calculate the following stats:

```
df['TenureNorm'].max()
```
should be 100

```
len(pd.unique(df['TenureNorm']))
```
returns the number of boxplots

```
df['MthChargeNorm'].max()
```
should be 100

To improve 'readability, prior to showing your plot, perform the following:

- stretch the x-axis of the figure to be almost the length of the Jupyter cell (hint: use matplotlib's `figsize`).
- There are too many x-axis ticks – which hinders readability. Use `ax.set_xticks` and `ax.set_xticklabels` to only display only these labels (and ticks associated with them): *zero* (positioned under the first boxplot); *middle* (positioned under the middle boxplot); *max* (positioned under the last boxplot). Note: these aesthetic changes should be enacted after calling `sns.boxplot()` but prior to showing the plot.

  Ref: https://www.tutorialspoint.com/matplotlib/matplotlib_setting_ticks_and_tick_labels.htm

Task #5b: Add a new field (named: TenureCat) to the DataFrame to store the categorization of the values within the Tenure field across four intervals. The interval labels are: *Bronze, Silver, Gold*, & *Platinum*. Ensure the interval cutoffs provide a balanced number (or close to that) of customers within each category and choose appropriate brief labels for each category. (Hint: you can use `cut()` / `qcut()`).

Note: adding field ➔

Note: use of `cut()` / `qcut()` creates categorical data (dtype=category). The **category** dtype in pandas is a hybrid data type. It looks and behaves like a string in many instances but internally is represented by an array of integers. This allows the data to be sorted in a custom order and more efficiently store the data.

Generate a seaborn **countplot** to display the number of customers within each category to confirm the categorization has been performed correctly.

(Re)position TenureCat immediately to the right of the Tenure field within the DataFrame. Use `df.dtypes` to confirm the insertion.

Note: removing field ➔

As we will no longer use the Tenure field, then go ahead and remove it from the DataFrame. Use `df.dtypes` to confirm the deletion.

Task #5c: For each of the Tenure categories, provide a count of customers across the different contract types.

Hint: use `groupby()`

Note: The result of a grouping operation will produce a series (with a MultiIndex).

Then convert the output series containing the results of the grouping operation to a DataFrame.

Ref: https://pandas.pydata.org/docs/reference/api/pandas.Series.to_frame.html

When the contents of a DataFrame are displayed within Jupyter, the tabular-like object displayed is a **Styler Object** (an HTML <table>). You can control certain attributes of the Styler table to format the display – for example, coloring certain cells.

To control how a Table (**Styler Object**) displays, you can use the `DataFrame.style` attribute (API) (`pd.io.formats.style.Styler`).

In terms of enacting styling (i.e., changing how the **Styler Object** displays), there are two options:

- use built-in stylers – e.g., `style.highlight_min()`, `style.highlight_max()`
- create a custom styler (applied to the Table, using either `applymap()` (elementwise), or `apply()` (column-/row-/table-wise))

Refs:

- https://pandas.pydata.org/docs/user_guide/style.html

- https://pandas.pydata.org/docs/user_guide/style.html#Styler-Functions

Technical note: Once the **Styler Object** is created (i.e., is displayed in the Jupyter cell), it becomes fixed and no additional styling operations can be enacted.

Requirements:

Display the result of the above grouping operation with the cell with the highest count value: colored in white; with a bold font; and with a background-color of dark blue.

Note: the applicable example in the panda's doc is a continuation of a prior example and leaves out `style` (which needs to be included). You may find it useful to lift some of the provided examples and execute them within a notebook to better understand how they work.

Create a copy of the DataFrame you created above to store the grouped data. This time, highlight the minimum value (use the same formatting as above).

Task #5d: Calculate the MonthlyCharges mean for each of the InternetService and TenureCat categories. Hint: use `groupby()`.

Task #5e: For each TenureCat category, calculate the mean of MonthlyCharges for each type of contract. Hint: use `groupby()`. Present the output as a DataFrame. Round the presented values to 2 decimal places.

6. Data Enrichment

As this is an exploratory study, we will restrict our focus to the customer dataset. In the future, we plan to extend our study to include data about marketing promotions, location data of our customers, and to analyze the call transcripts made to our call center.

7. Structuring

We are fortunate that the customer dataset does not need restructuring and we can move forward.

8. Feature Engineering

Task #8a: Create a new column within the DataFrame named Techsavvy.

Note: adding field ➔ Rows within this column will have a value of either *high*, *medium*, or *low*.

- *High* Techsavvy customers use both online security and online backup and have a Fiber Optic internet service.
- *Medium* Techsavvy customers also use Fiber Optic internet service and online security, but not online backup.
- All other customers are *low* Techsavvy.

As we cannot really tell the level of tech-savvyness for customers who do not use the internet service (i.e., are '*No*'), do not give them any 'Techsavvy' rating – instead, put a placeholder value of '*UK*' (short for unknown)in the Techsavvy column.

Requirements: codify the above logic in a UDF and refer to it using `apply()`.

After updating your DataFrame, display the first 10 records so you can manually inspect that the expected changes have been made successfully.

Also, display a count of how many records are in each Techsavvy category (there is no need to plot this data).

Exclude from the DataFrame all rows with an 'unknown' Techsavvy value.

Check the `dtype` of the new Techsavvy field and convert it to a categorical `dtype` if is of a different dtype. Again, use `df.shape` to confirm removal of these rows.

Note: removing fields ➔

Our analysis from now on will focus on the Techsavvy field -- accordingly, remove from the DataFrame: OnlineSecurity and OnlineBackup. But keep for now: InternetService.

Task #8b: Create a new column within the DataFrame named Streamer.

Note: adding field ➔

Rows within this column will have a value of 'streamer' if they subscribe to both StreamingTV & StreamingMovies. Otherwise, a customer will have a streamer value of 'non-streamer'.

Again, preclude any customers that do not subscribe to any internet service – give these customers the value of 'UK'.

Exclude from the DataFrame all rows with an 'unknown' Streamer value.

Again, after updating your DataFrame, print the first 10 records so you can manually inspect that the expected changes have been made successfully.

Check the `dtype` of the new Streamer field and convert it to a categorical `dtype` if a different dtype.

Note: removing fields ➔

Our analysis will now focus on the Streamer field -- accordingly, remove from the DataFrame: InternetService, StreamingTV, and StreamingMovies.

Display the value counts for Streamer.

At this stage, it may be difficult to see your dataset, even if using `head()`.To verify your dataset content, write your DataFrame to a csv file, using `df.to_csv(filename,index=False)`. Open and inspect this file to see if the contents are per expectations. Note: there is no need to submit this file.

Task #8c: Generate a seaborn **countplot** comparing Churn status against Techsavvy and another **countplot** comparing Churn status against Streamer.

Task #8d: Our selected modeling approach will be **Logistic Regression** (see Section 10). This approach requires all categorical variables used in modeling to be re-coded into an indicator matrix (dummy matrix) featuring 0|1s.

First: as the values within Churn are either *No* or *Yes*, we must rewrite the values so that *No* becomes 0 and *Yes* becomes 1.

Second: one-hot encode (see Chp. 7; Lesson 18) the following categorical fields (Streamer, Techsavvy, TenureCat, PaymentMethod) using `pd.getdummies()` – which will turn this data into a binary vector representation of 0|1s.

Finally, concatenate the following to create a new DataFrame:

- the generated indicator matrixes
- Churn (encoded to be either 0|1)
- MonthlyCharges (dtype: float64)

This new DataFrame will be the dataset used for model training/testing (see Section 10).

9. Feature Selection & Feature Importance

An example of a feature selection technique is *Recursive Feature Elimination (RFE).* RFE is based on the idea of repeatedly constructing a model and choosing either the best or worst performing feature, setting the feature aside, and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. The goal of RFE is to select features by recursively considering smaller and smaller sets of features. However, we will opt not to apply such techniques to this project.

Another issue normally covered in a techniques course will be the issue of unequal sample size within the prediction class. Again, given this is not a machine learning 'techniques' class, we will not address this issue.

10. Modeling (Machine Learning)

The main event.

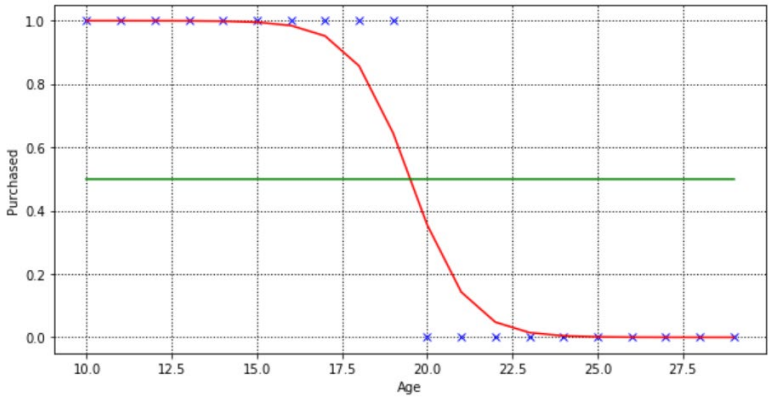
Logistic Regression

This course is not designed to be an introduction to various machine learning algorithms. However, it is useful to see how the data you have prepared in this project will be used in modeling. For this purpose, we will use **Logistic Regression** to classify our subscribers (i.e., Logistic Regression is a method for classification).

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. So, are dealing with binary classification – i.e., a binary logistic model has a dependent variable with two possible values.

Why 'log' in the name logistic model? In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value).

Rather than attempting to apply a linear model to training data, Logistic Regression attempts to fit training data to a sigmoidal curve, such as the example in the following image:



Binary logistic regression requires the dependent variable (in our case, Churn *status*) to be binary. As a supervised machine learning algorithm, we need to divide our data into **training** and **test** datasets

Task #10a: With the dataset now fully prepped for modeling, perform a binary logistic regression analysis.

Please do not confuse this requirement with a linear regression, which is something else entirely. In contrast, binary logistic regression is used for classification.

To perform machine learning, we will use the python ***scikit-learn*** package.

https://scikit-learn.org/stable/

This package is for professional Data Scientists rather than for the casual user, and perhaps is the foremost package for machine learning available (perhaps the primary reason for python's very high standing within the Data Science community).

You will need to import the following:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

We will need to complete two main tasks:

- Run the analysis
- Evaluate the quality of the model

To run the analysis, you can adapt examples shown on this webpage (but not the ones from the 'very confused' person (his quote) who made the initial posting).

https://stackoverflow.com/questions/60636444/what-is-the-difference-between-x-test-x-train-y-test-y-train-in-sklearn

You can find an explanation of Logistic Regression here:

https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

---

Technical Note: If you experience the following error:

lbfgs failed to converge (status=1): STOP: TOTAL NO. of ITERATIONS REACHED LIMIT

set the max_iter=1000 argument – ref:

https://stackoverflow.com/questions/62658215/convergencewarning-lbfgs-failed-to-converge-status-1-stop-total-no-of-iter

---

You will evaluate the quality of the model (classifier) by generating the following statistics of your model's performance:

- Logistic Regression train accuracy
- Logistic Regression test accuracy
- Confusion Matrix
- Classification Report

Your submission must display the above model statistics below the last Notebook cell. There is no need to add any personal comments – generating the required output is sufficient.

You have now arrived at the natural destination for this course – **congratulations**! In our journey this semester we have gone from x=1, all the way to applying a sophisticated machine learning algorithm on a highly manicured dataset. Sure, it's been a lot of hard work (for the Professor as well) – but the skills you have learned in this course will provide you with a clear advantage within a workplace that now places a significant premium on data literacy and analytical skills. But try not to let it go to your head

.

--- End of Project 2 ---