KnowledgeHut Churn prediction capstone project - Kitti Babos

2023 May

Table of content

Problem Statement

Customers in the telecommunications industry have the freedom to choose their service provider and can easily switch to a different company. As a result of the high level of competition in this market, 15–25% of customers switch providers annually. Since it costs 5–10 times as much to acquire a new customer as it does to keep an existing one, customer retention has become much more vital than customer acquisition.

For many established service providers, the most important thing is to keep their best, most profitable customers. Telecom providers may significantly lower customer churn by identifying and focusing on retaining those most at risk of leaving.

For this project, we will look at customer-level data from a major telecom company, build prediction models to find clients who are at risk and find the most telling signs of customer turnover.

Tasks/Activities List

- Collect the data from the zip file linked here.
- Split the dataset into dependent and independent variables.
- Perform train, test split.
- Apply StandardScaler() to the train and test dependent variable.
- Fit the best parameters.
- Model Prediction
- Model Validation Statistics
- Create a FastAPI app
- Create routes for getting predictions
- Create a route for retraining the model

Steps

Data analysis, preprocessing

Import libraries

First I imported all the libraries that I wanted to use for the EDA (Exploratory Data Analysis): Pandas, Numpy, Matplotlib, Seaborn

Read data

To understand the provided data, I used basic functions to get familiar with it:

- pd.read_excel('telco_new.xlsx'): to read the provided Excel file into a pandas DataFrame
- .head(): check the first 5 rows of the DataFrame
- .shape: shape of the DataFrame: 1000 rows, 42 columns
- .info: to check the columns' names, datatypes and the count of Non-Nulls
- .describe(): to generate descriptive statistics, added *include = 'all'* to include all columns (not just numericals)

Checking for categorical and numerical data

According to the provided extra text file (Telco_Curn_Data_Dictionary) the variables can be divided into 2 categories: 'nominal' (categorical) and 'numeric' (numerical)

Categorical features: region, marital, ed, retire, gender, tollfree, equip, callcard, wireless, multiline, voice, pager, internet, callid, callwait, forward, confer, ebil, custcat, churn

Numerical feature: tenure, age, address, income, employ, reside, longmon, tollmon, equipmon, cardmon, wiremon, longten, tollten, equipten, cardten, wireten, loglong, logtoll, logequi, logcard, logwire, lninc

When checking the data types I noticed that logtoll, logequi, logcard, logwire were featured as object data types instead of numerical (float or int) so I decided to convert them with the pd.to_numeric() function. Checking the updated data after that showed too many missing values for these 4 features,

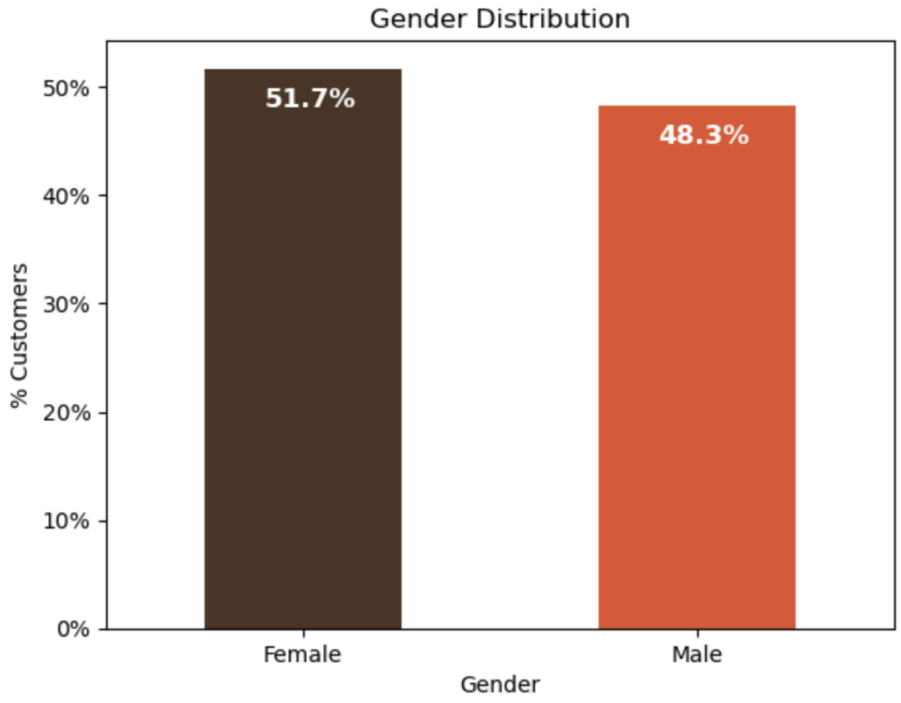
so I decided to drop them completely from the dataframe (dataframe.drop()), solving the missing values problem.

EDA

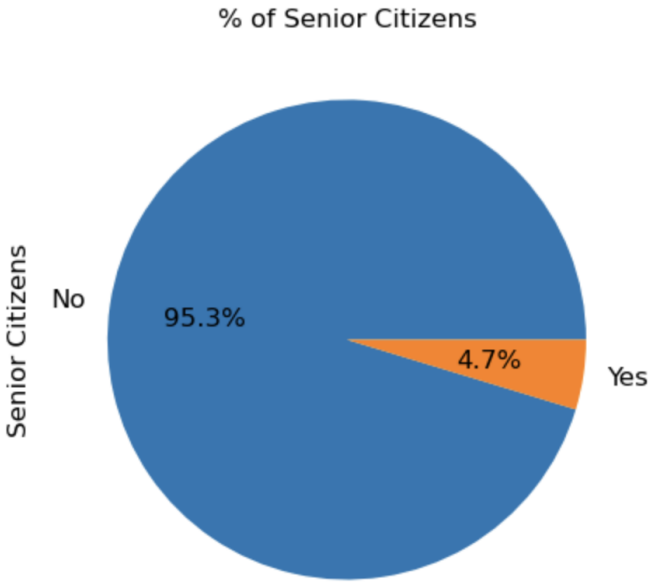
Demographics

To understand the gender, age range, partner and dependent status of the customers

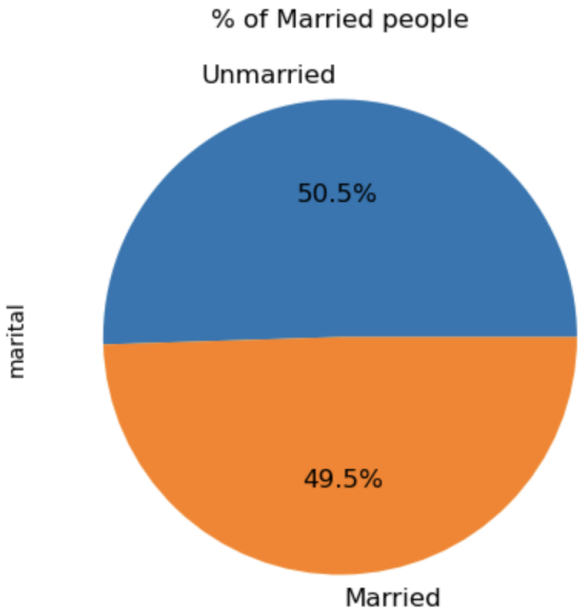
1. **Gender distribution**: both genders are well presented, female (51.7%), male (48.3%)



2. **Retirement rate** : There are only 4.7% of the customers who are retired. Thus most of our customers in the data are not retired people.



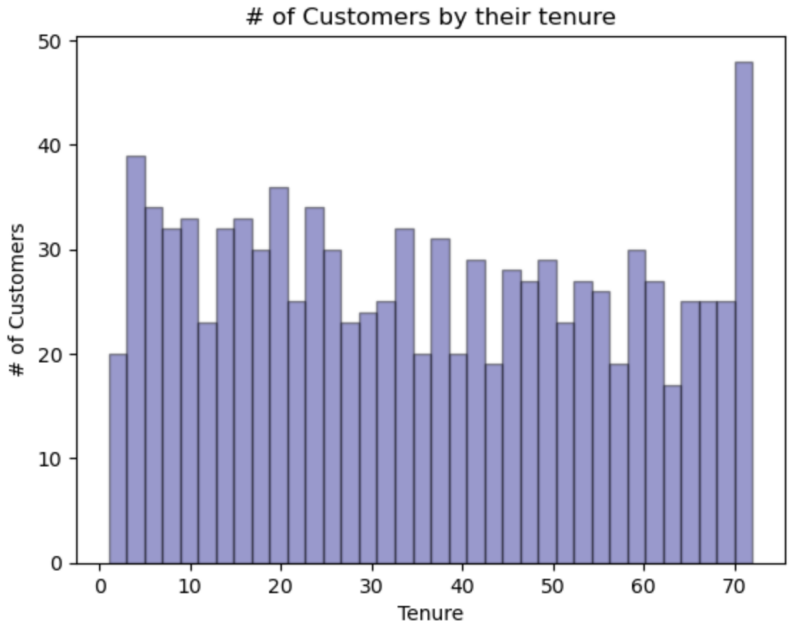
3. **Partner and dependent status**: nearly half of the customers are married



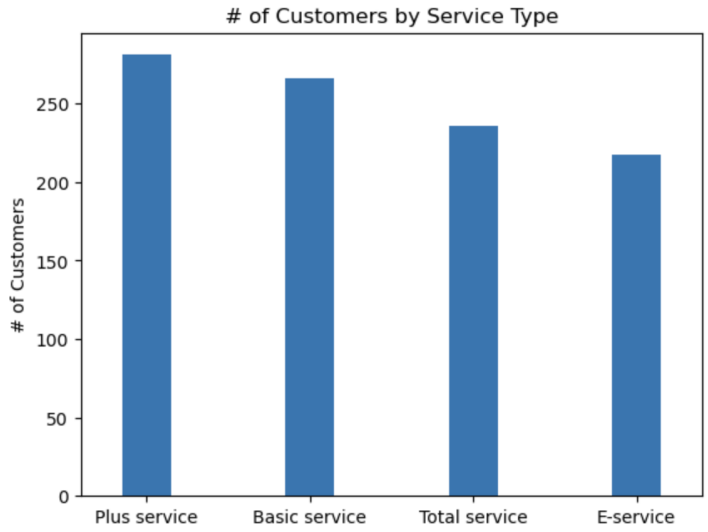
Customer Account Information

To check tenure (number of years (rather months?) a customer is using the service) and custcat (service category the customer is using)

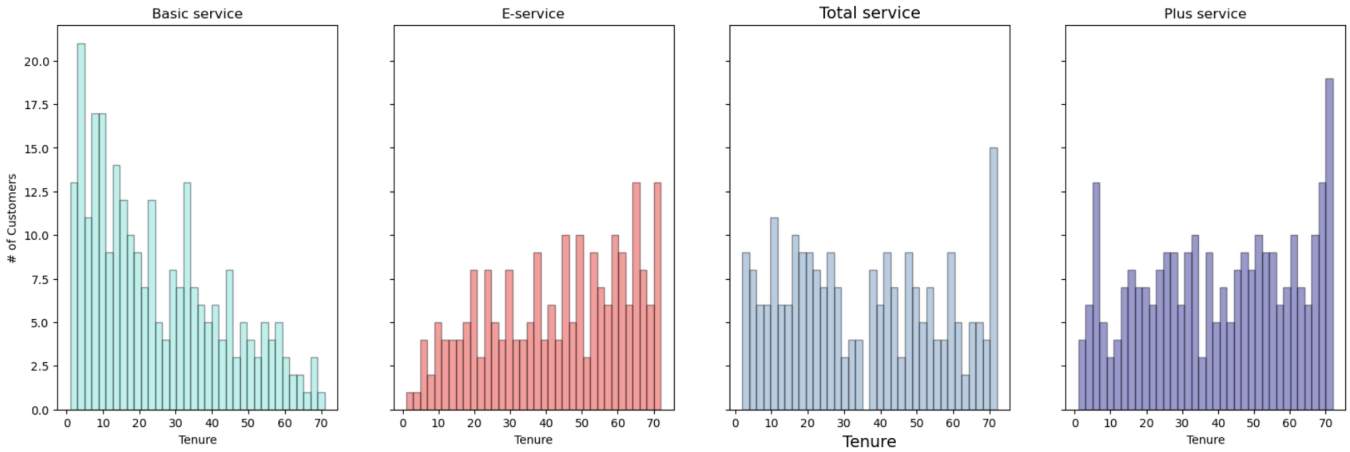
1. **Tenure**: many variations, but we can see a short tenure - fewer customer, long tenure - more customers tendency on the sides of the scale



2. **Custcat**: small differences, most popular service is the Plus service

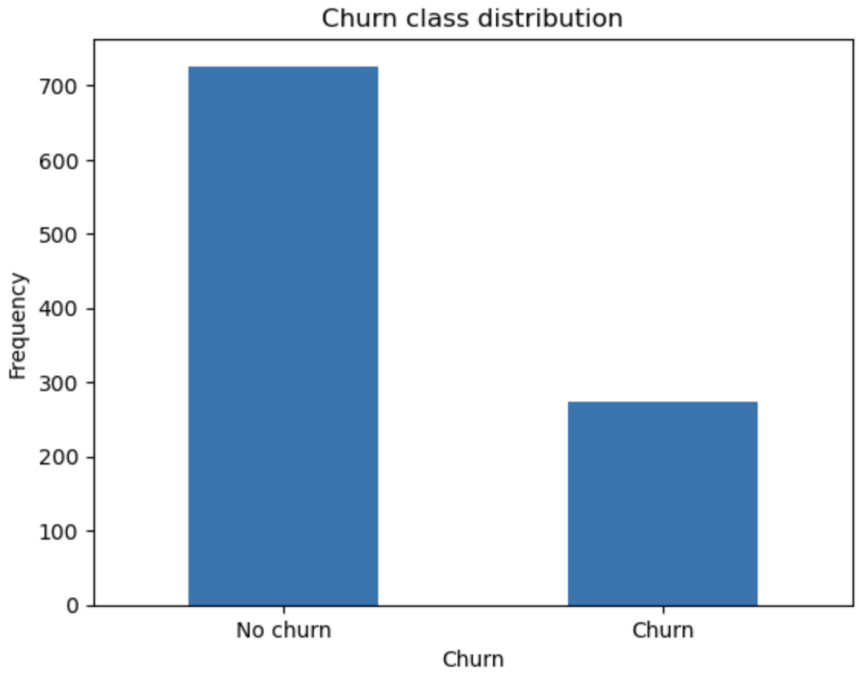


3. Checking **the tenure of customers based on their service type**: most of the basic service contracts last for a few months, while the contracts with extras tend to last much longer than that. This shows that the customers paying more for a service are more loyal to the company and tend to stay with it for a longer period of time



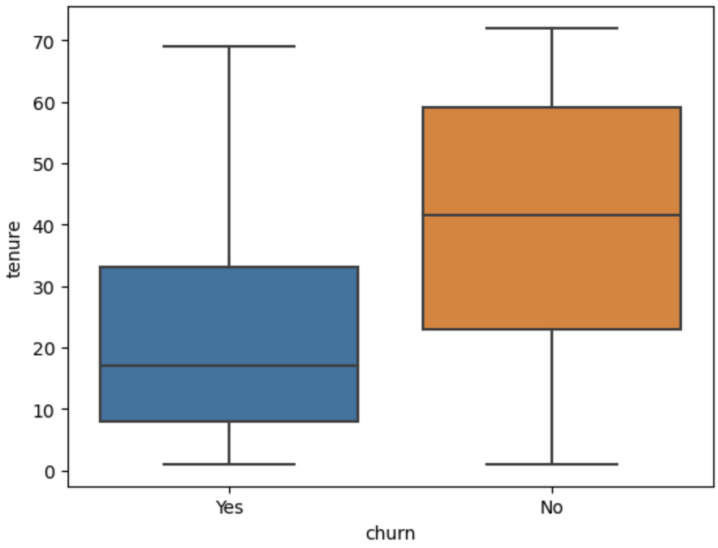
Churn

Analyzing our dependent feature**:** 72.6% of the customers do not churn

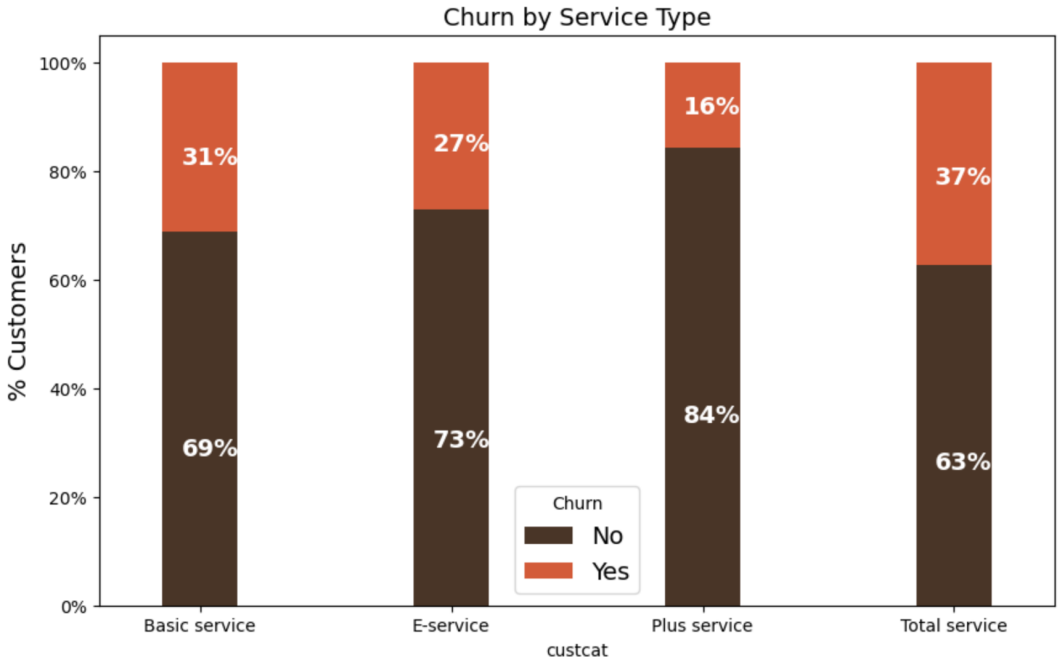


Figuring out **which features affect the churn rate**

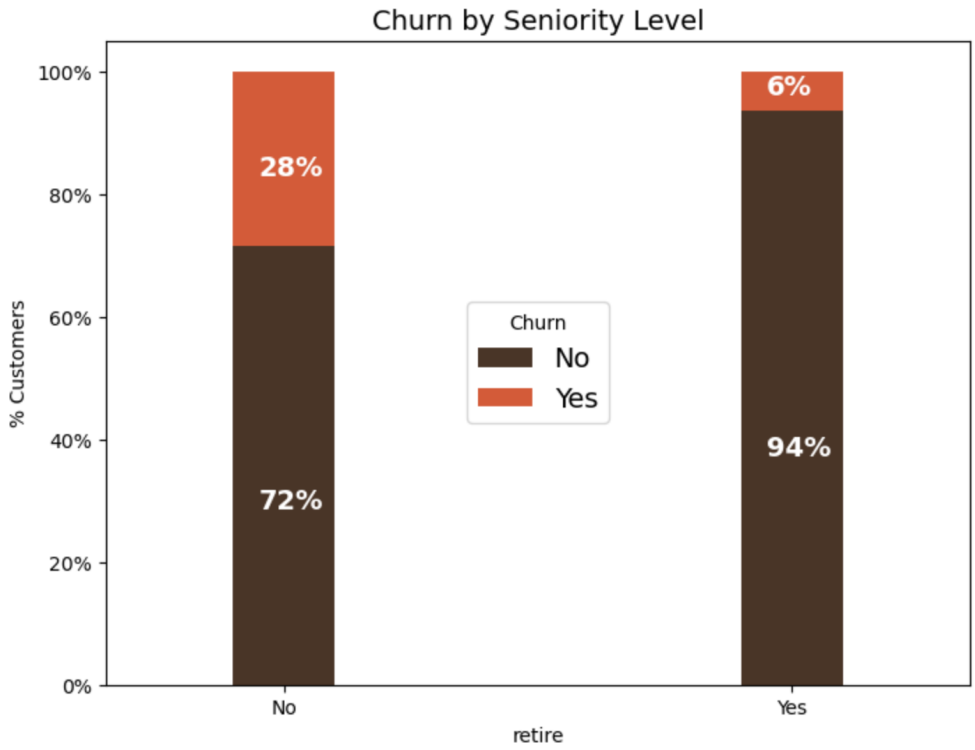
1. **Churn vs Tenure**: As we saw above, the customers who do not churn tend to stay for a longer tenure with the telecom company



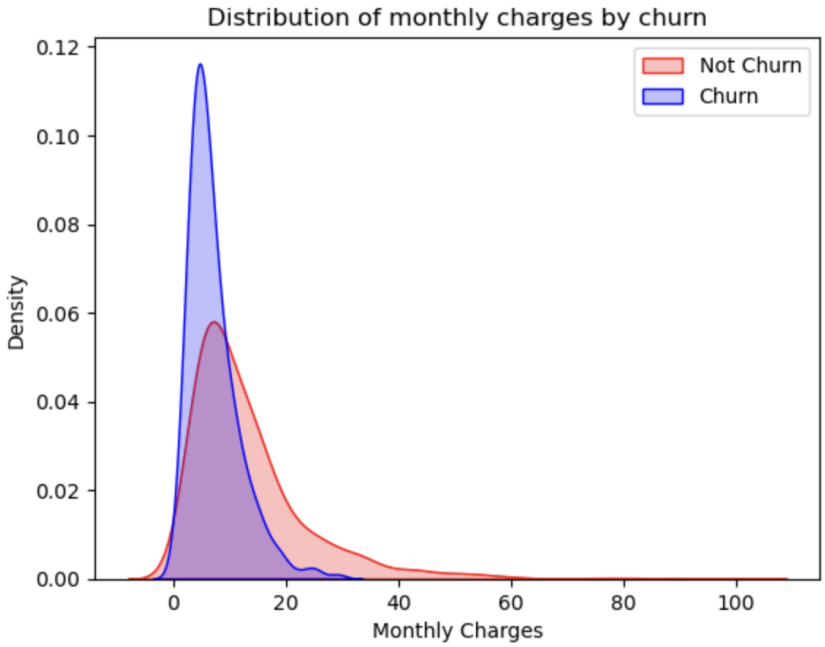
2. **Churn by Service Type**: similar numbers, customers with Plus service are less likely to churn



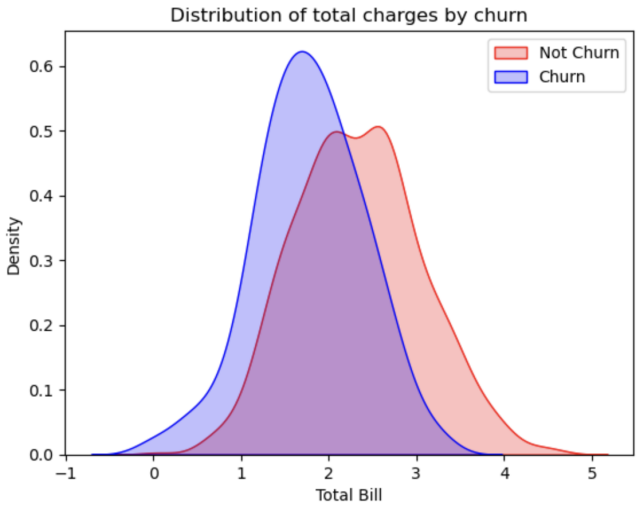
3. **Churn by Retirement:** based on this figure, non-retired citizens are more likely to churn



4. **Churn by Monthly Charges**: churn shows high density for lower monthly charges



5. **Churn by Total Charges**: similar figures

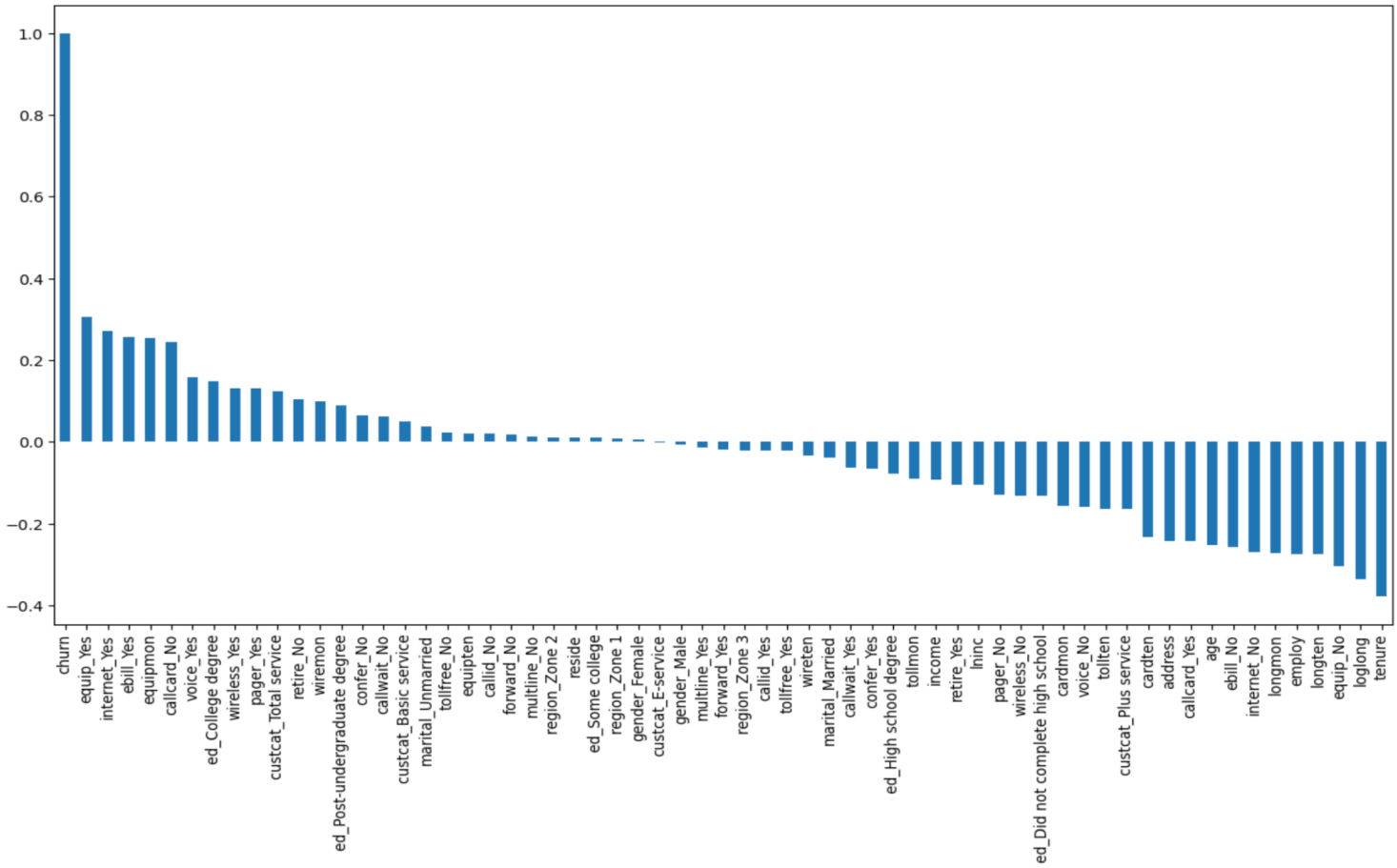


Data preprocessing

Standardize the data

a. Converting the predictor variable (churn) into a binary numeric variable (.replace())
b. Converting all the categorical variables into dummy variables (pd.get_dummies())

Get Correlation of churn with other variables



As we have many features, it is advisable to remove the irrelevant ones before creating and training our model on the dataset. We can avoid overfitting issues this way. To determine the most important independent variables, I decided to use the correlation coefficient: significant positive correlation: coeff > 0.3, significant negative correlation: coeff < -0.2

Based on this:

- **positive** correlation: equip_Yes, internet_Yes, ebill_Yes, equipmon, callcard_No
- **negative** correlation: cardten, address, callcard_Yes, age, ebill_No, internet_No, longmon, employ, longten, equip_No, loglong, tenure

To test if this really makes a different or not, I trained my models on all the features (df_dummies) and also on only the selected features based on the correlation coefficient (df_dummies_selected)

Machine learning

Import libraries

Machine learning models, train_test_split, metrics, scaler

Create a function to prepare the data for our models

This function defines the dependent (y) and independent (X) variables, use standardscaler to scale X and execute the train-test split, returning the inputs for our machine learning models: X_train, X_test, y_train, y_test

Model creating, training, prediction and evaluation

1. Logistic regression
    a. It measures the relationship between the categorical dependent variable and one or more independent variables by estimating the probability of occurrence of an event using its logistics function.
    b. Accuracy score: **0.805** (using the fine-tuned dataset instead or the whole data didn't make difference to this score)
2. Random forest
    a. Multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree, then taking the most popular result
    b. Accuracy score on selected dataset: **0.815**

3. Support Vector Machine (SVM)
    a. Its objective is to segregate the given dataset in the best possible way. The distance between the nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset.
    b. Accuracy score on selected dataset: **0.81**
4. Gaussian Naive Bayes
    a. It assumes that each parameter has an independent capacity of predicting the output variable. The combination of the prediction for all parameters is the final prediction, that returns a probability of the dependent variable to be classified in each group. The final classification is assigned to the group with the higher probability.
    b. Accuracy score on selected dataset: *0.68*
        i. It's not good enough, solution: hyperparameter tuning
            1. Var_smoothing (Variance smoothing) parameter specifies the portion of the largest variance of all features to be added to variances for stability of calculation.
            2. RepeatedStratifiedKFold: Repeats Stratified K-Fold n times with different randomization in each repetition.
            3. GridSearchCV: helps to loop through predefined hyperparameters and fit the estimator (model) on the training set
            4. PowerTransformer: makes the features more or less normally distributed
        ii. New accuracy score: **0.785**

Create a FastAPI app

What is FastAPI?

A modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints.

The **key features**

- Fast: Very high performance, one of the fastest Python frameworks available.
- Fast to code: Increases the speed to develop features
- Fewer bugs
- Intuitive: Great editor support. Completion everywhere. Less time debugging.
- Easy: Designed to be easy to use and learn. Less time reading docs.
- Short: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
- Robust: Production-ready code with automatic interactive documentation.
- Standards-based: Based on the open standards for APIs: OpenAPI and JSON Schema.

Steps of deploying machine learning models

- Save the trained models with pickle library
- Open Spyder to create API
    - Libraries to import: FastAPI (to create API in python), BaseModel (to specify the input formats), pickle (to load the saved model), json
    - Create a folder on computer ML_model_as_API
        - Create python_code folder there and add previously saved ML model files (churn_model_regression.sav, etc.) to this folder, Spyder file (ml_api.py) will be also added to this folder
    - Create *app* variable, loading an instance of FastAPI there
    - Create *model_input* class: mention all the input parameters there that the model needs (independent variables from df_dummies_selected) with their data types. E. g. equip_Yes : int

        - Telling the API that this is the format of the data that we're expecting
    - Loading one of the saved models with pickle.load()
    - Create the API *@app.post()*: post method when someone needs to add values to the API
        - We need to mention the ending of our URL here
            - @app.post('/churn_prediction_regression')
    - Create a function *churn_pred_regression*
        - Input parameters are the ones that the user will send to the API (model_input)
        - Input data will be posted to our API in the form of json which we need to convert to a dictionary and finally to a list of tuples
    - Pass values to our saved model to predict the label
        - If it predicts 0: return that the customer won't churn
        - If it predicts 1: return that the customer is likely to churn
    - Open Anaconda and open the terminal from the same environment
        - We need to mention the path of the file in which our python file is stored (python_code folder)
        - We can deploy our API from there: **uvicorn ml_api:app**
            - We get the local host URL where the FastAPI has been deployed: http://127.0.0.1:8000/
    - Testing the API
        - Create new file in Spyder (api_implementation.py)
            - Import json and requests libraries
            - Create variable with the local host URL and inpoint:
                - url = 'http://127.0.0.1:8000/churn_prediction_regression'
            - Create a dictionary with the input data for the model
            - Convert dictionary to json
            - Create response variable where we post the json file to our model
                - response = requests.post(url, data = input_json)
            - Print the value of it

```
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
Type "copyright", "credits" or "license" for more
information.

IPython 7.31.1 -- An enhanced Interactive Python.

In [1]: runcell(0, '/Users/kittibabos/Documents/
Phyton_data_bootcamp/Exercises/
Data_science_projects/drive-
download-20230411T091029Z-001/Teleco churn
prediction/ML_model_as_API/python_code/
api_implementation.py')
"Customer will not churn"
```

Create a route for retraining the model

- Same analogy as in the previous steps
    - Create a new tab on Spyder called 'ml_retrain_api.py'
        - Import libraries
        - Initiate app as FastAPI() instance
        - Create model_input where the user can choose the parameters for the retrain
            - Dataframe
            - Scaler
            - Test size
            - Random state
        - Create the API @app.post('/churn_prediction_regression')
            - Define the data_prep function with the details on how to use the user's inputs for the model training
            - It will return the accuracy score of the retrained model